



Piscine C

C 00

Résumé: CE document est le sujet du module C 00 de la piscine C de 42.

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_putchar	7
IV	Exercice 01 : ft_print_alphabet	8
V	Exercice 02 : ft_print_reverse_alphabet	9
VI	Exercice 03 : ft_print_numbers	10
VII	Exercice 04 : ft_is_negative	11
VIII	Exercice 05 : ft_print_comb	12
IX	Exercice 06 : ft_print_comb2	13
X	Exercice 07 : ft_putnbr	14
XI	Exercice 08 : ft_print_combn	15

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices.
- Vos exercices seront corrigés par vos camarades de piscine.
- En plus de vos camarades, vous serez corrigés par un programme appelé la Moulinette.
- La Moulinette est très stricte dans sa notation. Elle est totalement automatisée. Il est impossible de discuter de sa note avec elle. Soyez d'une rigueur irréprochable pour éviter les surprises.
- La Moulinette n'est pas très ouverte d'esprit. Elle ne cherche pas à comprendre le code qui ne respecte pas la Norme. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction `main()` que si nous vous demandons un programme.
- La Moulinette compile avec les flags `-Wall -Wextra -Werror`, et utilise `gcc`.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec

votre voisin de gauche.

- Votre manuel de référence s'appelle `Google / man / Internet /`
- Pensez à discuter sur le forum Piscine de votre Intra, ainsi que sur le slack de votre Piscine !
- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Réfléchissez. Par pitié, par Odin ! Nom d'une pipe.



Pour ce module, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

La confiture de nouilles, selon Pierre Dac

Avant d'utiliser la nouille pour la confection de la confiture, il faut évidemment la récolter; avant de la récolter, il faut qu'elle pousse, et pour qu'elle pousse, il va de soi qu'il faut d'abord la semer.

Les semailles de la graine de nouille, c'est-à-dire les senouilles, représentent une opération extrêmement délicate. Tout d'abord, le choix d'un terrain propice à la fécondation de la nouille demande une étude judicieusement approfondie. Le terrain nouillifère type doit être, autant que possible, situé en bordure de la route départementale et à proximité de la gendarmerie nationale.

Avant de semer la graine de nouille, les nouilliculteurs préparent longuement le champ nouillifère pour le rendre idoine à la fécondation. Ils retournent la terre avec une charrue spéciale dont le soc est remplacé par une lame Gillette, ensuite délaissant les engrais chimiques, nettement contre-indiqués dans le cas présent, ils fument le champ nouillifère avec du fromage râpé. Cette opération s'effectue indifféremment avec une seringue ou une pompe à vélo.

Lorsque le champ est suffisamment imprégné de fromage râpé, on verse sur toute sa surface de l'alcool de menthe dans la proportion d'un verre à Bordeaux par hectare de superficie; cette opération qui est confiée à des spécialistes de l'École de Nouilliculture, est effectuée avec un compte-gouttes.

Après cela, on laisse fermenter la terre pendant toute la durée de la nouvelle lune et dès l'apparition du premier quartier, on procède alors aux senouilles de la graine de nouilles. Il ne faudrait pas vous imaginer, Mesdames et Messieurs, que la graine de nouilles est d'un commerce courant et qu'on la trouve communément chez les grainetiers ; si vous croyez cela, il est indiscutable que vous broutez les coteaux de l'erreur. La graine de nouilles ne s'obtient qu'après une très longue préparation de laboratoire, car elle est le produit d'un croisement de foie de veau avec le concombre adulte; voici d'ailleurs quelques précisions sur cette merveilleuse con]onction qui est la gloire de nos chimistes, dont la science n'a d'égale que la modestie.

On met côte à côte, dans une lessiveuse, une tranche de foie de veau et un

concombre adulte, on place le tout dans un autoclave et on l'y laisse 45 jours à une température de 120° sous la bienveillance d'un contrôleur de la Compagnie du Gaz; au bout de ce laps de temps, on ouvre l'appareil et on n'a plus qu'à recueillir la précieuse graine que l'on va verser dans la terre prête à la recevoir et qu'elle va féconder.

Les senouilles s'effectuent à l'aide d'un poêle mobile dans lequel est versée la graine, laquelle est projetée dans la terre par un dispositif spécial dont il ne nous est pas permis de révéler le secret pour des raisons de défense nationale que l'on comprendra aisément. Après ça, on arrose entièrement le champ avec des siphons d'eau de seltz, on sèche ensuite avec du papier buvard, on donne un coup de plumeau et on n'a plus qu'à s'en remettre au travail de la terre nourricière et à la nature immortelle, généreuse et démocratique. Lorsque les senouilles sont terminées, les nouilliculteurs qui sont encore entachés de superstition, consultent les présages; ils prennent une petite taupe, la font courir dans l'herbe et si elle fait : "ouh!" c'est que la récolte sera bonne; si elle ne fait pas "ouh!" c'est que la récolte sera bonne tout de même, mais comme cela les croyances sont respectées, et tout le monde est content.

Pendant la germination, il n'y a presque rien à faire ; tous les huit jours seulement, on arrose le champ avec de l'huile de cade, de la cendre de cigare, du jus de citron et de la glycérine pour éviter que la terre ne se crevasse.

Pendant la moisson, les nuits sont témoins de saines réjouissances auxquelles se livrent les travailleurs de la nouilliculture, la jeunesse danse et s'en donne à cœur joie aux sons d'un orchestre composé d'un harmonium, d'une mandoline et d'une trompette de cavalerie ; les jeunes gens revêtent leur costume régional composé d'une redingote, d'une culotte cycliste, d'espadrilles et d'un chapeau Cronstadt ; les jeunes filles, rougissantes de joie pudique, sont revêtues de ravissantes robes de toile à cataplasme, ornées d'empiècements en schpoutnoutz, et se ceignent le front d'une couronne d'œufs durs du plus gracieux effet Un feu d'artifice tiré avec des lampes Pigeon clôture la série des réjouissances et chacun rentre chez soi, content du labeur accompli, pour procéder alors à la confection de la confiture de nouilles, objet de la présente étude.

La nouille encore à l'état brut, est alors soigneusement triée et débarrassée de ses impuretés; après un premier stade, elle est expédiée à l'usine et passée immédiatement au laminouille qui va lui donner l'aspect définitif que nous lui connaissons - le laminouille est une machine extrêmement perfectionnée, qui marche au guignolet-cassis et qui peut débiter jusqu'à 80 kilomètres de nouilles à l'heure - ; à la sortie du laminouille, la nouille est passée au vernis cellulósique qui la rend imperméable et souple; elle est ensuite hachée menue à la hache d'abordage et râpée. Le râpage se fait encore à la main et avec une râpe à bois. Après le râpage, la nouille est alors mise en bouteilles, opération très délicate qui demande énormément d'attention ; on met ensuite les bouteilles dans un appareil appelé électronouille, dans lequel passe un courant de 210 volts; après un séjour de 12 heures dans cet appareil, les bouteilles sont


sorties et on vide la nouille désormais électrifiée dans un récipient placé lui-même sur un réchaud à alcool à haute tension.

On verse alors dans ledit récipient : du sel, du sucre, du poivre de Cayenne, du gingembre, de la cannelle, de l'huile, de la pomme de terre pilée, un flocon de magnésie bismurée, du riz, des carottes, des peaux de saucisson, des tomates, du vin blanc, et des piments rouges, on mélange lentement ces ingrédients avec la nouille à l'aide d'une cuiller à pot et on laisse mitonner à petit feu pendant 21 jours. La confiture de nouilles est alors virtuellement terminée. Lorsque les 21 jours sont écoulés, que la cuisson est parvenue à son point culminant et définitif, on place le récipient dans un placard, afin que la confiture se solidifie et devienne gélatineuse; quand elle est complètement refroidie, on soulève le récipient très délicatement, avec d'infinies précautions et le maximum de prudence et on balance le tout par la fenêtre parce que c'est pas bon!

Contrairement à la confiture de nouilles, le C c'est bon, mangez-en!

Chapitre III

Exercice 00 : ft_putchar

	Exercice : 00
ft_putchar	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : ft_putchar.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche le caractère passé en paramètre.
- Elle devra être prototypée de la façon suivante :


```
void ft_putchar(char c);
```

Pour afficher le caractère, vous devez utiliser la fonction **write** de la manière suivante.

```
write(1, &c, 1);
```


Chapitre IV

Exercice 01 : ft_print_alphabet


	Exercice : 01
ft_print_alphabet	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : ft_print_alphabet.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche l'alphabet en minuscule sur une seule ligne, dans l'ordre croissant, à partir de la lettre 'a'.
- Elle devra être prototypée de la façon suivante :

```
void ft_print_alphabet(void);
```

Chapitre V

Exercice 02 : ft_print_reverse_alphabet


	Exercice : 02
	ft_print_reverse_alphabet
	Dossier de rendu : <i>ex02/</i>
	Fichiers à rendre : ft_print_reverse_alphabet.c
	Fonctions Autorisées : write

- Écrire une fonction qui affiche l'alphabet en minuscule sur une seule ligne, dans l'ordre décroissant, à partir de la lettre 'z'.
- Elle devra être prototypée de la façon suivante :

```
void ft_print_reverse_alphabet(void);
```

Chapitre VI

Exercice 03 : ft_print_numbers


	Exercice : 03
ft_print_numbers	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : ft_print_numbers.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche tous les chiffres sur une seule ligne, dans l'ordre croissant.
- Elle devra être prototypée de la façon suivante :

```
void ft_print_numbers(void);
```

Chapitre VII

Exercise 04 : ft_is_negative


	Exercice : 04
	ft_is_negative
	Dossier de rendu : <i>ex04/</i>
	Fichiers à rendre : ft_is_negative.c
	Fonctions Autorisées : write

- Écrire une fonction qui affiche 'N' ou 'P' suivant le signe de l'entier passé en paramètre. Si **n** est négatif alors afficher 'N'. Si **n** est positif ou nul alors afficher 'P'.
- Elle devra être prototypée de la façon suivante :

```
void ft_is_negative(int n);
```

Chapitre VIII

Exercice 05 : ft_print_comb

	Exercice : 05
	ft_print_comb
	Dossier de rendu : <i>ex05/</i>
	Fichiers à rendre : ft_print_comb.c
	Fonctions Autorisées : write

- Écrire une fonction qui affiche, dans l'ordre croissant, toutes les différentes combinaisons de trois chiffres différents dans l'ordre croissant - oui, la répétition est volontaire.
- Cela donne quelque chose comme ça :


```
$>./a.out | cat -e  
012, 013, 014, 015, 016, 017, 018, 019, 023, ..., 789$>
```

- 987 n'est pas là car 789 est déjà présent
- 999 n'est pas là car ce nombre ne comporte pas exclusivement des chiffres différents les uns des autres
- Elle devra être prototypée de la façon suivante :

```
void ft_print_comb(void);
```

Chapitre IX

Exercice 06 : ft_print_comb2

	Exercice : 06
	ft_print_comb2
	Dossier de rendu : <i>ex06/</i>
	Fichiers à rendre : ft_print_comb2.c
	Fonctions Autorisées : write

- Écrire une fonction qui affiche toutes les différentes combinaisons de deux nombres entre 0 et 99, dans l'ordre croissant.
- Cela donne quelque chose comme ça :


```
$>./a.out | cat -e  
00 01, 00 02, 00 03, 00 04, 00 05, ..., 00 99, 01 02, ..., 97 99, 98 99$>
```

- Elle devra être prototypée de la façon suivante :

```
void ft_print_comb2(void);
```

Chapitre X

Exercice 07 : ft_putnbr

	Exercice : 07
ft_putnbr	
Dossier de rendu : <i>ex07/</i>	
Fichiers à rendre : ft_putnbr.c	
Fonctions Autorisées : write	


- Écrire une fonction qui affiche un nombre passé en paramètre. La fonction devra être capable d'afficher la totalité des valeurs possibles dans une variable de type `int`.
- Elle devra être prototypée de la façon suivante :

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

- Par exemple :
 - `ft_putnbr(42)` affiche "42".

Chapitre XI

Exercice 08 : ft_print_combn

	Exercice : 08
ft_print_combn	
Dossier de rendu : <i>ex08/</i>	
Fichiers à rendre : ft_print_combn.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche toutes les différentes combinaisons de **n** chiffres dans l'ordre croissant.
- **n** sera tel que : $0 < n < 10$.
- Si **n** = 2, cela donne quelque chose comme ça :

```
$>./a.out | cat -e  
01, 02, 03, ..., 09, 12, ..., 79, 89$>
```

- Elle devra être prototypée de la façon suivante :

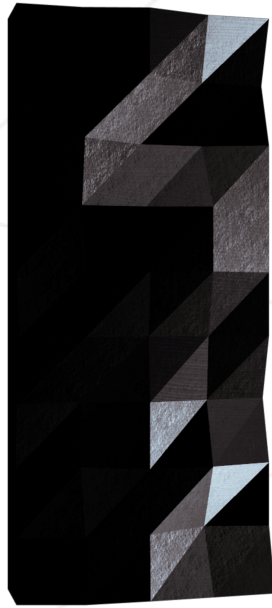
```
void ft_print_combn(int n);
```




The Norm

Version 3

Summary: This document describes the applicable standard (Norm) at 42. A programming standard defines a set of rules to follow when writing code. The Norm applies to all C projects within the Inner Circle by default, and to any project where it's specified.



The Norm

Version 3

Summary: This document describes the applicable standard (Norm) at 42. A programming standard defines a set of rules to follow when writing code. The Norm applies to all C projects within the Inner Circle by default, and to any project where it's specified.

Contents

I	Foreword	2
II	The Norm	3
II.1	Denomination	3
II.2	Formatting	4
II.3	Functions	6
II.4	Typedef, struct, enum and union	7
II.5	Headers	8
II.6	Macros and Pre-processors	9
II.7	Forbidden stuff!	10
II.8	Comments	11
II.9	Files	12
II.10	Makefile	13
II.11	Why ?	14

Chapter I

Foreword

The Norm is in python and open source.

Its repository is available at <https://github.com/42School/norminette>.

Pull requests, suggestions and issues are welcome!

Chapter II

The Norm

II.1 Denomination

- A structure's name must start by `s_`.
- A typedef's name must start by `t_`.
- A union's name must start by `u_`.
- An enum's name must start by `e_`.
- A global's name must start by `g_`.
- Variables and functions names can only contain lowercases, digits and '`_`' (Unix Case).
- Files and directories names can only contain lowercases, digits and '`_`' (Unix Case).
- Characters that aren't part of the standard ASCII table are forbidden.
- Variables, functions, and any other identifier must use snake case. No capital letters, and each word separated by an underscore.
- All identifiers (functions, macros, types, variables, etc.) must be in English.
- Objects (variables, functions, macros, types, files or directories) must have the most explicit or most mnemonic names as possible.
- Using global variables that are not marked `const` and `static` is forbidden and is considered a norm error, unless the project explicitly allows them.
- The file must compile. A file that doesn't compile isn't expected to pass the Norm.

II.2 Formatting

- You must indent your code with 4-space tabulations. This is not the same as 4 average spaces, we're talking about real tabulations here.
- Each function must be maximum 25 lines, not counting the function's own curly brackets.
- Each line must be at most 80 columns wide, comments included. Warning: a tabulation doesn't count as a column, but as the number of spaces it represents.
- Each function must be separated by a newline. Any comment or preprocessor instruction can be right above the function. The newline is after the previous function.
- One instruction per line.
- An empty line must be empty: no spaces or tabulations.
- A line can never end with spaces or tabulations.
- You can never have two consecutive spaces.
- You need to start a new line after each curly bracket or end of control structure.
- Unless it's the end of a line, each comma or semi-colon must be followed by a space.
- Each operator or operand must be separated by one - and only one - space.
- Each C keyword must be followed by a space, except for keywords for types (such as int, char, float, etc.), as well as sizeof.
- Each variable declaration must be indented on the same column for its scope.
- The asterisks that go with pointers must be stuck to variable names.
- One single variable declaration per line.
- Declaration and an initialisation cannot be on the same line, except for global variables (when allowed), static variables, and constants.
- Declarations must be at the beginning of a function.
- In a function, you must place an empty line between variable declarations and the remaining of the function. No other empty lines are allowed in a function.
- Multiple assignments are strictly forbidden.
- You may add a new line after an instruction or control structure, but you'll have to add an indentation with brackets or assignment operator. Operators must be at the beginning of a line.
- Control structures (if, while..) must have braces, unless they contain a single line or a single condition.

General example:

```
int          g_global;
typedef struct s_struct
{
    char      *my_string;
    int       i;
}             t_struct;
struct       s_other_struct;

int          main(void)
{
    int       i;
    char      c;

    return (i);
}
```

II.3 Functions

- A function can take 4 named parameters maximum.
- A function that doesn't take arguments must be explicitly prototyped with the word "void" as the argument.
- Parameters in functions' prototypes must be named.
- Each function must be separated from the next by an empty line.
- You can't declare more than 5 variables per function.
- Return of a function has to be between parenthesis.
- Each function must have a single tabulation between its return type and its name.

```
int my_func(int arg1, char arg2, char *arg3)
{
    return (my_val);
}

int func2(void)
{
    return ;
}
```


II.4 Typedef, struct, enum and union

- Add a tabulation when declaring a struct, enum or union.
- When declaring a variable of type struct, enum or union, add a single space in the type.
- When declaring a struct, union or enum with a typedef, all indentation rules apply. You must align the typedef's name with the struct/union/enum's name.
- You must indent all structures' names on the same column for their scope.
- You cannot declare a structure in a .c file.

II.5 Headers

- The things allowed in header files are: header inclusions (system or not), declarations, defines, prototypes and macros.
- All includes must be at the beginning of the file.
- You cannot include a C file.
- Header files must be protected from double inclusions. If the file is `ft_foo.h`, its bystander macro is `FT_FOO_H`.
- Unused header inclusions (`.h`) are forbidden.
- All header inclusions must be justified in a `.c` file as well as in a `.h` file.

```
#ifndef FT_HEADER_H
# define FT_HEADER_H
# include <stdlib.h>
# include <stdio.h>
# define FOO "bar"

int          g_variable;
struct      s_struct;

#endif
```

II.6 Macros and Pre-processors

- Preprocessor constants (or `#define`) you create must be used only for literal and constant values.
- All `#define` created to bypass the norm and/or obfuscate code are forbidden. This part must be checked by a human.
- You can use macros available in standard libraries, only if those ones are allowed in the scope of the given project.
- Multiline macros are forbidden.
- Macro names must be all uppercase.
- You must indent characters following `#if`, `#ifdef` or `#ifndef`.

II.7 Forbidden stuff!

- You're not allowed to use:
 - for
 - do...while
 - switch
 - case
 - goto
- Ternary operators such as '?'.
 - VLAs - Variable Length Arrays.
 - Implicit type in variable declarations

```
int main(int argc, char **argv)
{
    int    i;
    char    string[argc]; // This is a VLA

    i = argc > 5 ? 0 : 1 // Ternary
}
```

II.8 Comments

- Comments cannot be inside functions' bodies. Comments must be at the end of a line, or on their own line
- Your comments must be in English. And they must be useful.
- A comment cannot justify a "bastard" function.

II.9 Files

- You cannot include a .c file.
- You cannot have more than 5 function-definitions in a .c file.

II.10 Makefile

Makefiles aren't checked by the Norm, and must be checked during evaluation by the student.

- The `$(NAME)`, `clean`, `fclean`, `re` and all rules are mandatory.
- If the makefile relinks, the project will be considered non-functional.
- In the case of a multibinary project, in addition to the above rules, you must have a rule that compiles both binaries as well as a specific rule for each binary compiled.
- In the case of a project that calls a function from a non-system library (e.g.: `libft`), your makefile must compile this library automatically.
- All source files you need to compile your project must be explicitly named in your Makefile.

II.11 Why ?

The Norm has been carefully crafted to fulfill many pedagogical needs. Here are the most important reasons for all the choices above:

- Sequencing: coding implies splitting a big and complex task in a long series of elementary instructions. All these instructions will be executed in sequence: one after another. A beginner that starts creating software needs a simple and clear architecture for their project, with a full understanding of all individual instructions and the precise order of execution. Cryptic language syntaxes that do multiple instructions apparently at the same time are confusing, functions that try to address multiple tasks mixed in the same portion of code are source of errors.

The Norm asks you to create simple pieces of code, where the unique task of each piece can be clearly understood and verified, and where the sequence of all the executed instructions leaves no doubt. That's why we ask for 25 lines maximum in functions, also why `for`, `do .. while`, or ternaries are forbidden.

- Look and Feel: while exchanging with your friends and workmates during the normal peer-learning process, and also during the peer-evaluations, you do not want to spend time to decrypt their code, but directly talk about the logic of the piece of code.

The Norm asks you to use a specific look and feel, providing instructions for the naming of the functions and variables, indentation, brace rules, tab and spaces at many places... . This will allow you to smoothly have a look at other's codes that will look familiar, and get directly to the point instead of spending time to read the code before understanding it. The Norm also comes as a trademark. As part of the 42 community, you will be able to recognize code written by another 42 student or alumni when you'll be in the labor market.

- Long-term vision: making the effort to write understandable code is the best way to maintain it. Each time that someone else, including you, has to fix a bug or add a new feature they won't have to lose their precious time trying to figure out what it does if previously you did things in the right way. This will avoid situations where pieces of code stop being maintained just because it is time-consuming, and that can make the difference when we talk about having a successful product in the market. The sooner you learn to do so, the better.
- References: you may think that some, or all, the rules included on the Norm are arbitrary, but we actually thought and read about what to do and how to do it. We highly encourage you to Google why the functions should be short and just do one thing, why the name of the variables should make sense, why lines shouldn't be longer than 80 columns wide, why a function should not take many parameters, why comments should be useful, etc, etc, etc...