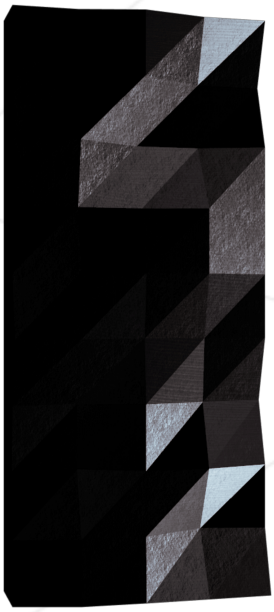

La norme de 42

Version 2.0.0

Mathieu mathieu@staff.42.fr
Gaëtan gaetan@staff.42.fr

Résumé: Ce document décrit la norme C en vigueur à 42. Une norme de programmation définit un ensemble de règles régissant l'écriture d'un code. Il est obligatoire de respecter la norme lorsque vous écrivez du C à 42.

Table des matières

Chapitre I

Avant-propos

Ce document décrit la norme C en vigueur à 42. Une norme de programmation définit un ensemble de règles régissant l'écriture d'un code. Il est obligatoire de respecter la norme lorsque vous écrivez du C à 42.

I.1 Pourquoi imposer une norme ?

La norme a deux objectifs principaux : Uniformiser vos codes afin que tout le monde puisse les lire facilement, étudiants et encadrants. Ecrire des codes simples et clairs.

I.2 La norme dans vos rendus

Tous vos fichiers de code C doivent respecter la norme de 42. La norme sera verifiée par vos correcteurs et la moindre faute de norme donnera la note de 0 à votre projet ou à votre exercice. Lors des peer-corrections votre correcteur devra lancer la "Norminette" sur votre rendu, seul le résultat de la "Norminette" doit être pris en compte.

I.3 Conseils

Comme vous le comprendrez rapidement, la norme n'est pas une contrainte. Au contraire, la norme est un garde-fou pour vous guider dans l'écriture d'un C simple et basique. C'est pourquoi il est absolument vital que vous codiez directement à la norme, q uite à coder plus lentement les premières heures. Un fichier de sources qui contient une faute de norme est aussi mauvais qu'un fichier qui en compte dix. Soyez studieux et appliqués, et la norme deviendra un automatisme sous peu.

I.4 Disclamers

Des bugs existent forcément sur la "Norminette", merci de les reporter sur la section du forum de l'intra. Néanmoins, la "Norminette" fait foi et vos rendus doivent s'adapter à ses bugs :).

Chapitre II

La norme de 42

II.1 Convention de dénomination

Partie obligatoire

- Un nom de structure doit commencer par s_.
- Un nom de typedef doit commencer par t_.
- Un nom d'union doit commencer par u_.
- Un nom d'enum doit commencer par e_.
- Un nom de globale doit commencer par g_.
- Les noms de variables, de fonctions doivent être composés exclusivement de minuscules, de chiffres et de '_' (Unix Case).
- Les noms de fichiers et de répertoires doivent être composés exclusivement de minuscules, de chiffres et de '_' (Unix Case).
- Le fichier doit être compilable.
- Les caractère ne faisant pas partie de la table ascii standart ne sont pas autorisés.

Partie conseillée

- Les objets (variables, fonctions, macros, types, fichiers ou répertoires) doivent avoir les noms les plus explicites ou mnémoniques. Seul les 'compteurs' peuvent être nommés à votre guise.
- Les abréviations sont tolérées dans la mesure où elles permettent de réduire significativement la taille du nom sans en perdre le sens. Les parties des noms composites seront séparées par '_'.
- Tous les identifiants (fonctions, macros, types, variables, etc.) doivent être en anglais.
- Toute utilisation de variable globale doit être justifiée.

II.2 Formattage

Partie obligatoire

- Tous vos fichiers devront commencer par le header standard de 42 dès la première ligne.
- Chaque fonction doit faire au maximum 25 lignes sans compter les accolades du bloc de la fonction.
- Chaque ligne ne peut faire plus de 80 colonnes, commentaire compris. Attention : une tabulation ne compte pas pour une colonne mais bien pour les n espaces qu'elle represente.
- Une seule instruction par ligne.
- Une ligne vide ne doit pas contenir d'espace ou de tabulation.
- Une ligne ne devrait jamais se terminer par des espaces ou des tabulations.
- Quand vous rencontrez une accolade ouvrante ou fermante ou une fin de structure de controle, vous devez retourner à la ligne.
- Vous devez aussi indenter votre code avec des tabulations de 4 espaces. (Ce n'est pas équivalent à 4 espace, ce sont bien des tabulations.)
- Chaque virgule ou point-virgule doit être suivi d'un espace si nous ne sommes pas en fin de ligne.
- Chaque opérateur (binaire ou ternaire) et opérande doivent être séparé par un espace et seulement un.
- Chaque mot clef du C doit être suivi d'un espace sauf pour les mot-clefs de type de variable (comme int, char, float, etc.) ainsi que sizeof. (On a gardé parce que KRP a dit 82 qui accessoirement est pair)
- Chaque déclaration de variable doit être indentée sur la même colonne.
- Les étoiles des pointeurs doivent être collés au nom de la variable.
- Une seule déclaration de variable par ligne.
- On ne peut faire une déclaration et une instanciation sur une même ligne, à l'exception des variables globales et des variables statiques.
- Les déclarations doivent être en début de fonctions et doivent être séparées de l'implémentation par une ligne vide.
- Aucune ligne vide ne doit être présente au milieu des déclarations ou de l'implémentation.
- Vous pouvez retourner à la ligne lors d'une même instruction ou structure de contrôle, mais vous devez rajouter une indentation par parenthèse ou opérateur d'affectation. Les opérateurs doivent être en début de ligne.

```
toto = 42 + 5
    - 28;               //RIGHT (Mais lecture difficile :P)

if (toto == 0
    && titi == 2
    && (tutu == 3                   //RIGHT (Mais lecture difficile :P)
        || tata == 4)
    || rara == 3)
```

- La multiple assignation est interdite.

II.3 Paramètres de fonction

Partie obligatoire

- Une fonction prend au maximum 4 paramètres nommés.
- Une fonction qui ne prend pas d'argument doit explicitement être prototypée avec le mot void comme argument.

II.4 Fonctions

Partie obligatoire

- Les prototypes de fonction doivent posséder des noms de variable.
- Chaque définition de fonction doit être séparé par une ligne vide.

Partie conseillée

- Vos noms de fonctions doivent être alignés dans un même fichier. Cela s'applique aux headers.

II.5 Typedef, struct, enum et union

Partie obligatoire

- Vous devez mettre une tabulation lorsque vous déclarez une struct, enum ou union.
- Lors de la déclaration d'une variable de type struct, enum ou union vous ne mettrez qu'un espace dans le type.
- Vous devez utiliser une tabulation entre les deux paramètres d'un typedef.
- Lorsque vous déclarez une struct, union ou enum avec un typedef, toutes les règles s'appliquent et vous devez aligner le nom du typedef avec le nom de la struct, union ou enum.
- Vous ne pouvez pas déclarer une structure dans un fichier .c.

II.6 Headers

Partie obligatoire

- Seuls les inclusions de headers (système ou non), les définitions, les defines, les prototypes et les macros sont autorisés dans les fichiers headers.
- Tous les includes de .h doivent se faire au début du fichier (.c ou .h).
- On protégera les headers contre la double inclusion. Si le fichier est ft_foo.h, la macro témoin est 'FT_FOO_H'.
- Les prototypes de fonctions doivent se trouver exclusivement dans des fichiers .h.
- Une inclusion de header (.h) dont on ne se sert pas est interdite.

Partie conseillée

- Toute inclusion de header doit etre justifiée autant dans un .c que dans un .h.

II.7 Macros et Pré-processeur

Partie obligatoire

- Les defines définissant du code sont interdits.
- Les macros multilignes sont interdites.
- Seuls les noms de macros sont en majuscules
- Il faut indenter les caractères qui suivent un #if , #ifdef ou #ifndef

II.8 Choses Interdites !

Partie obligatoire

- Vous n'avez pas le droit d'utiliser :
  - for (parce que c'est tombé sur face)
  - do...while
  - switch
  - case
  - goto
- Les opérateurs ternaires ' ?' imbriqués

II.9 Commentaires

Partie obligatoire

- Les commentaires peuvent se trouver dans tous les fichiers source.
- Il ne doit pas y avoir de commentaires dans le corps des fonctions.
- Les commentaires sont commencés et terminés par une ligne seule. Toutes les lignes intermédiaires s'alignent sur elles, et commencent par '**'.
- Pas de commentaire en //.

Partie conseillée

- Vos commentaires doivent être en anglais et utiles.
- Le commentaire ne peut pas justifier une fonction bâtarde.

II.10 Les fichiers

Partie obligatoire

- Vous ne pouvez inclure un .c.
- Vous ne pouvez avoir plus de 5 définitions de fonction dans un .c.

II.11 Makefile

Partie conseillée

- Les règles $(NAME), clean, fclean, re et all sont obligatoires.
- Le projet est considéré comme non fonctionnel si le makefile relink.
- Dans le cas d'un projet multibinaire, en plus des règles précédentes, vous devez avoir une règle all compilant les deux binaires ainsi qu'une règle spécifique à chaque binaire compilé.
- Dans le cas d'un projet faisant appel à une bibliothèque de fonctions (par exemple une libft), votre makefile doit compiler automatiquement cette bibliothèque.
- L'utilisation de 'wildcard' (*.c par exemple) est interdite.