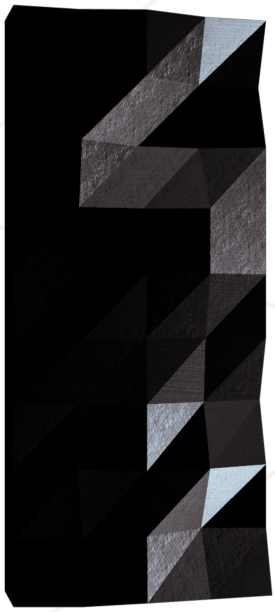

La norme de 42

Version 2.0.2

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" présente sur les dumps sur votre rendu. Seule la partie obligatoire de la norme est vérifiée par la "Norminette".

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, quite 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 (en français ou en anglais). Néanmoins, la "Norminette" fait foi et vos rendus doivent s'adapter à ses bugs. Elle n'est cependant pas infaillible. Si vous repérez des erreurs de norme non-détectées par la "Norminette" il en va de votre responsabilité de noter le projet de votre pair comme il se doit.

Chapitre II

Norme

II.1 Conventions 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 standard 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 Formatage

Partie obligatoire

- Tous vos fichiers devront commencer par le header standard de 42 dès la première ligne. Ce header est disponible par defaut dans les éditeurs `emacs` et `vim` sur les dumps.
- Vous devez indenter votre code avec des tabulations de la taille de 4 espaces. Ce n'est pas équivalent à 4 espace, ce sont bien des tabulations.
- 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, commentaires 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.
- 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 opérandes doivent être séparés par un espace et seulement un.
- Chaque mot-clé du C doit être suivi d'un espace, sauf pour les mot-clefs de type (comme int, char, float, etc.) ainsi que sizeof.
- 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 initialisation sur une même ligne, à l'exception des variables globales et des variables statiques.
- Les déclarations doivent être en début de fonction 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.
- La multiple assignation est interdite.
- Vous pouvez retourner à la ligne lors d'une même instruction ou structure de

II.6 Headers

Partie obligatoire

- Seuls les inclusions de headers (système ou non), les déclarations, 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 constantes de préprocesseur (#define) que vous créez ne doivent être utilisés que pour associer des valeurs littérales et constantes, et rien d'autre.
- Les #define rédigés dans le but de contourner la norme et/ou obfusquer du code interdit par la norme sont interdites. Ce point doit être vérifiable par des humains.
- Vous pouvez utiliser les macros présentes dans les bibliothèques standards, si cette dernière est autorisée dans le projet ciblé.
- Les #define 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
  - do...while
  - switch
  - case
  - goto

- Les opérateurs ternaires ‘ ?’ imbriqués
- Les tableaux à taille variable (VLA - Variable Length Array)

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 ‘**’.
- Les zones de commentaires commencant par // sont interdits.

Partie conseillée

- Vos commentaires doivent être en anglais et utiles.
- Les commentaires ne peuvent pas justifier une fonction bâtarde.

II.10 Les fichiers

Partie obligatoire

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

II.11 Makefile

Partie obligatoire

- 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.
- Les sources nécessaires à la compilation de votre programme doivent être explicitement cités dans votre Makefile.