



# A norma da 42

Versão 2.0.2

Mathieu [mathieu@staff.42.fr](mailto:mathieu@staff.42.fr)

Gaëtan [gaetan@staff.42.fr](mailto:gaetan@staff.42.fr)

Lytchi [lytchi@staff.42.fr](mailto:lytchi@staff.42.fr)

*Sumário: Esse documento descreve a norma C em vigor na 42. Uma norma de programação definida por um conjunto de regras que regem a escrita de um código. É obrigatório respeitar a norma quando você escrever o C na 42, a não ser que especificado de outra forma.*

# Conteúdo

<b>I</b>	<b>Prefácio</b>	<b>2</b>
I.1	Por que impor uma norma? . . . . .	2
I.2	A norma nas suas entregas . . . . .	2
I.3	Sugestões . . . . .	2
I.4	Disclaimers . . . . .	2
<b>II</b>	<b>A Norma</b>	<b>3</b>
II.1	Conversões de nomenclatura . . . . .	3
II.1.1	Parte recomendada . . . . .	3
II.2	Formatação . . . . .	4
II.3	Parâmetros de função . . . . .	5
II.4	Funções . . . . .	5
II.4.1	Parte recomendada . . . . .	5
II.5	Typedef, struct, enum e union . . . . .	5
II.6	Headers . . . . .	6
II.6.1	Parte recomendada . . . . .	6
II.7	Macros e Pré-processador . . . . .	6
II.8	Coisas proibidas! . . . . .	6
II.9	Comentários . . . . .	7
II.9.1	Parte recomendada . . . . .	7
II.10	Os arquivos . . . . .	7
II.11	Makefile . . . . .	7
II.11.1	Parte recomendada . . . . .	7

# Capítulo I

## Prefácio

Esse documento descreve a norma C em vigor na 42. Uma norma de programação definida por um conjunto de regras que regem a escrita de um código. É obrigatório respeitar a norma quando você escrever o C na 42, a não ser que especificado de outra forma.

### I.1 Por que impor uma norma?

A norma tem dois objetivos principais: 1. Uniformizar os códigos para que todo mundo consiga ler facilmente, estudantes, staff e até você mesmo. 2. Guiá-lo na escrita de código simples, curto e claro.

### I.2 A norma nas suas entregas

Todos os seus arquivos de código C devem respeitar a norma da 42. A norma será verificada pelos seus corretores e o menor erro de norma receberá nota 0, seja no projeto ou no exercício. Além de correção dos colegas, o seu corretor deverá iniciar a "Norminette" presente nos dumps na sua entrega. Somente o resultado da "Norminette" deve ser considerado. Somente a parte obrigatória da norma é verificada pela "Norminette".

### I.3 Sugestões

Como você vai entender rapidamente, a norma não é apenas uma restrição. Pelo contrário, a norma é uma proteção para guiar você ao escrever um C simples e básico. É absolutamente vital que você codifique dentro da norma, mesmo que tenha que codificar mais lentamente nas primeiras horas. Um arquivo de fontes que contém um erro de norma é tão ruim quanto um arquivo que contém dez. Seja estudioso e aplicado, e a norma ficará automática rapidinho.

### I.4 Disclaimers

A "Norminette" é um programa, e todo programa está sujeito a bugs. Por favor, reporte-os pelo slack (em português ou em inglês). Contudo, a "Norminette" é confiável e suas submissões devem se adaptar ao seus bugs.

# Capítulo II

## A Norma

### II.1 Converções de nomenclatura

#### Parte obrigatória

- Um nome de estrutura deve começar com `s_`.
- Um nome de typedef deve começar com `t_`.
- Um nome de union deve começar com `u_`.
- O nome de enum deve começar com `e_`.
- Um nome de global deve começar com `g_`.
- Os nomes de variáveis, de funções devem ser compostos exclusivamente por minúsculas, números e '\_' (Unix Case).
- Os nomes de arquivos e pastas devem ser compostos exclusivamente por minúsculas, números e '\_' (Unix Case).
- O arquivo deve ser compilável.
- Os caracteres que não fazem parte da tabela ascii padrão são proibidos.

#### II.1.1 Parte recomendada

- Os objetos (variáveis, funções, macros, tipos, arquivos ou pastas) devem ter os nomes mais explícitos ou mnemônicos. Somente os 'contadores' podem ser nomeados do jeito que você quiser.
- As abreviações são toleradas na medida em que permitirem reduzir significativamente o tamanho do nome sem perder o sentido. As partes dos nomes compostos devem ser separadas por '\_'.
- Todos os identificadores (funções, macros, tipos, variáveis, etc.) devem estar em inglês.

- Toda utilização de variável global deve ser justificada.

## II.2 Formatação

### Parte obrigatória

- Todos os seus arquivos devem começar com o header padrão da 42 desde as primeiras linhas. Esse header está disponível por padrão nos editores `emacs` e `vim` nos dumps.
- Você deve escrever o seu código com tabulações de 4 espaços. Isso não é equivalente a 4 espaços, são definitivamente tabulações.
- Cada função deve ter no máximo 25 linhas sem contar as chaves do bloco de função.
- Cada linha não pode ter mais que 80 colunas, comentários inclusos. Atenção: uma tabulação não conta como coluna mas como os n espaços que ela representa.
- Uma única instrução por linha.
- Uma linha vazia não deve conter espaços ou tabulações.
- Uma linha nunca deve terminar com espaços ou tabulações.
- Quando você encontrar uma chave de abertura ou de fechamento ou um fim de estrutura de controle, você deve quebrar a linha.
- Cada vírgula ou ponto-vírgula deve ser seguido de um espaço se não estivermos no final de uma linha.
- Cada operador (binário ou ternário) e operandos devem ser separados por um espaço e somente um.
- Cada palavra-chave do C deve ser seguida por um espaço, exceto para as palavras-chaves de tipo (como `int`, `chr`, `float`, etc.) assim como `sizeof`.
- Cada declaração de variável deve ser indentada na mesma coluna.
- Os asteriscos dos ponteiros devem estar colados no nome da variável.
- Uma única declaração de variável por linha.
- Não se pode fazer uma declaração e uma inicialização em uma mesma linha.
- As declarações devem estar no início da função e devem ser separadas da implementação por uma linha vazia.
- Nenhuma linha vazia deve estar presente no meio das declarações ou da implementação.

- A múltipla atribuição é proibida.
- Você pode quebrar a linha em uma mesma instrução ou estrutura de controle, mas deve adicionar uma intenção entre parênteses ou operador de atribuição. Os operadores devem estar no início da linha.

## II.3 Parâmetros de função

### Parte obrigatória

- Uma função tem no máximo 4 parâmetros nomeados.
- Uma função que não tem argumento deve explicitamente ser prototipada com a palavra void como argumento.

## II.4 Funções

### Parte obrigatória

- Os parâmetros dos protótipos de funções devem ser nomeados
- Cada definição de função deve estar separada da seguinte por uma linha vazia.
- Você só pode declarar 5 variáveis por bloco no máximo.
- O return de uma função deve estar entre parênteses

### II.4.1 Parte recomendada

- Seus identificadores de função devem ser alinhados em um mesmo arquivo. A mesma coisa para os headers.

## II.5 Typedef, struct, enum e union

### Parte obrigatória

- Você deve colocar uma tabulação quando declarar uma struct, enum ou union.
- Quando fizer uma declaração de uma variável de tipo struct, enum ou union, deve colocar apenas um espaço no tipo.
- Você deve usar uma tabulação entre os parâmetros de um typedef.
- Quando você declarar um struct, union ou enum com um typedef, todas as regras se aplicam e você deve alinhar o nome do typedef com o nome da struct, union ou enum.
- Você não pode declarar uma estrutura em um arquivo .c.

## II.6 Headers

### Parte obrigatória

- Somente as inclusões de headers (do sistema ou não), as declarações, as defines, os protótipos e os macros são autorizados nos arquivos headers.
- Todos os includes de .h devem ser feitos no início do arquivo (.c ou .h).
- Você deve proteger os headers contra a dupla inclusão. Se o arquivo é `ft_foo.h`, a macro testemunha é `FT_FOO_H`.
- Os protótipos de função devem estar exclusivamente nos arquivos .h.
- Uma inclusão de header (.h) não usado é proibida.

### II.6.1 Parte recomendada

- Toda inclusão de header deve ser justificada tanto em um .c quanto em um .h.

## II.7 Macros e Pré-processador

### Parte obrigatória

- Os defines que definem o código são proibidos.
- As macros com muitas linhas são proibidas.
- Somente os nomes de macros devem estar em maiúsculo
- É preciso indentar os caracteres que seguem um `#if` , `#ifdef` ou `#ifndef`

## II.8 Coisas proibidas!

### Parte obrigatória

- Você não pode usar:
  - `for`
  - `do...while`
  - `switch`
  - `case`
  - `goto`
- Os operadores ternários `'?'` aninhados

- Array de tamanho variável (VLA - Variable Length Array)

## II.9 Comentários

### Parte obrigatória

- Você pode comentar no seu código fonte.
- Não pode haver comentários no corpo das funções.
- Os comentários começam e terminam com uma linha única. Todas as linhas intermediárias se alinham entre si e começam com `/**`.
- Nada de comentários com `//`.

### II.9.1 Parte recomendada

- Os seus comentários devem ser em inglês. E devem ser úteis.
- O comentário não pode justificar uma função bastarda.

## II.10 Os arquivos

### Parte obrigatória

- Você não pode incluir um `.c`.
- Você não pode ter mais que 5 definições de funções em um `.c`.

## II.11 Makefile

### II.11.1 Parte recomendada

- As regras `$(NAME)`, `clean`, `fclean`, `re` e `all` são obrigatórias.
- O projeto é considerado como não funcional se o makefile fizer `relink`.
- No caso de um projeto multibinário, além das regras precedentes, você deve ter uma regra `all` que compile os binários, assim como uma regra específica para cada binário compilado.
- No caso de um projeto chamar uma biblioteca de funções (por exemplo um `libft`), o seu makefile deve compilar automaticamente essa biblioteca.
- Todos os arquivos de código fonte necessários para compilar seu projeto devem ser explicitamente nomeados no Makefile (nada de wildcards - `*`)