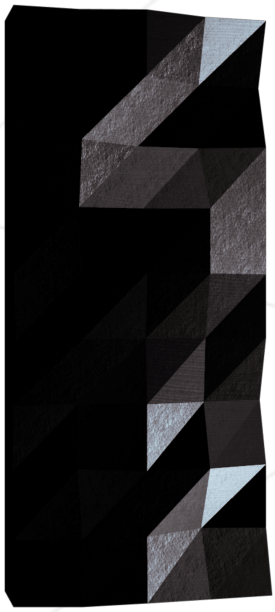

La norma de 42

Versión 2.0.2

Resumen: Este documento describe la norma de C en vigor en 42. Una norma de programación define un conjunto de normas que rigen la excritura de código. El respeto de la norma al programar en C en 42 es obligatorio.

Índice general

Capítulo I

Prólogo

Este documento describe la norma de C en vigor en 42. Una norma de programación define un conjunto de normas que rigen la excritura de código. El respeto de la norma al programar en C en 42 es obligatorio.

I.1. ¿Por qué imponer una norma ?

La norma pretende uniformizar el código con dos objetivos principales: que el código sea simple y claro, y que todo el mundo lo pueda leer fácilmente, tanto los alumnos como los responsables.

I.2. La norma en las entregas

Todos sus ficheros escritos en C deben respetar la norma de 42. Sus correctores verificarán la norma, y el mínimo error de norma pondrá a su ejercicio o a su proyecto un 0. Durante las correcciones entre pares, su corrector deberá ejecutar la "Norminette"sobre el volcado de sesión de su entrega. La "Norminette"sólo verificará la parte obligatoria de la norma.

I.3. Consejos

Rápidamente verá que la norma no es una restricción. Más bien al contrario, es una protección que le guiará en la escritura de un código C simple y básico. Por eso, es vital que codifique directamente aplicando la norma, aunque eso suponga codificar más despacio durante las primeras horas. Un fichero fuente con un error de norma es tan malo como un fichero con 10. Sea estudioso y aplicado, y la norma se convertirá en un automatismo en poco tiempo.

I.4. Descargo de responsabilidad

Inevitablemente, la "Norminette"puede tener errores. Le agradecemos por reportalos en el foro de la intranet, en inglés o en francés. Sin embargo, la "Norminette"dará fe de sus trabajos, que deberán adaptarse a esos errores. La "Norminette"no es infalible. Si

detecta errores de norma no detectados por la "Norminette", es su responsabilidad como corrector el calificar el proyecto de su para como es debido.

Capítulo II

Norma

II.1. Convenio de denominación

Parte obligatoria

- Un nombre de estructura debe empezar por `s_`.
- Un nombre de typedef debe empezar por `t_`.
- Un nombre de union debe empezar por `u_`.
- Un nombre de enum debe empezar por `e_`.
- Un nombre de global debe empezar por `g_`.
- Los nombres de variables y de funciones deben estar formados exclusivamente por minÃºsculas, cifras y '_' (Unix Case).
- Los nombres de ficheros y de directorios deben estar formados exclusivamente por minÃºsculas, cifras y '_' (Unix Case).
- El fichero debe ser compilable.
- No están autorizados los caracteres que no están en la tabla ascii estándar.

Parte aconsejada

- Los objetos (variables, funciones, macros, tipos, ficheros o directorios) deben tener nombre lo más explícito posible, o ser mnemotécnicos. Sólo es libre de nombrar como le parezca los contadores.
- Se toleran abreviaciones en la medida en la que permiten reducir significativamente el tamaño del nombre sin pérdida de significado. Las partes de los nombres compuestos estarán separadas por '_'.
- Todos los identificadores (funciones, macros, tipos, variables, etc) deben estar en inglés.
- TDeberá justificar el uso de cualquier variable global.

II.2. Formateo

Parte obligatoria

- Todos sus ficheros deben empezar por el header standard de 42 desde la primera línea. Este header está disponible por defecto en los editores `emacs` y `vim` de los volcados de sesión.
- Debe indentar su código con tabulaciones de tamaño igual a cuatro espacios. No son cuatro espacios. Son tabulaciones.
- Cada función tendrá como máximo 25 líneas sin contar las llaves del bloque de la función.
- Cada línea tendrá como máximo 80 columnas, comentarios incluidos. Cuidado: una tabulación no cuenta como una columna, sino como los cuatro espacios que representa.
- Una sola instrucción por línea.
- Una línea vacía no debe contener ni espacios ni tabulaciones.
- Una línea no debe terminar ni por un espacio ni por una tabulación. Cuando encuentre una llave de apertura o de cierre, o un fin de estructura de control, debe pasar a la línea.
- Cada coma o punto y coma debe ir seguido por un espacio si no se encuentra en el final de la línea.
- Cada operador, binario o ternario, y los operandos, deben separarse con espacio y sólo uno.
- Cada palabra clave de C debe ir seguida de un espacio, salvo las palabras clave de tipo (como int, char, float, etc.) así como sizeof.
- Cada declaración de variable irá indentada en la misma columna.
- Los asteriscos de los punteros deben estar pegados al nombre de la variable.
- Sólo se declarará una variable por línea.
- No se puede hacer una declaración y una inicialización en la misma línea, salvo para las variables globales y las estáticas.
- Las declaraciones deben estar en el principio de la función y deben separarse de la implementación por una línea vacía.
- No puede haber ninguna línea vacía en el medio de las declaraciones o de la implementación.
- No está permitida la asignación mÃºltiple.
- Puede incluir un salto de línea en una instrucción, o en una estructura de control, pero debe añadir una indentación por paréntesis o asignación de variable. En caso

de salto de línea en una asignación con operadores, los operadores deben estar al principio de la línea.

II.3. Parámetros de función

Parte obligatoria

- Una función tendrá como máximo 4 parámetros con nombre.
- Una función sin argumentos se prototipará explícitamente con el argumento void.

II.4. Funciones

Parte obligatoria

- Los parámetros de los prototipos de función deben tener un nombre.
- Cada definición de función debe estar separada de la siguiente por una línea vacía.
- Puede declarar un máximo de 5 variables por bloque.
- El retorno de una función se hará entre paréntesis.

Parte aconsejada

- Sus identificadores de función deben estar alineados en un mismo fichero. Esto es aplicable a los headers C.

II.5. Typedef, struct, enum y union

Parte obligatoria

- Debe poner una tabulación cuando declare una struct, un enum o una union.
- Cuando declare una variable de typo struct, enum o union, ponga sólo un espacio en el tipo.
- Debe usar una tabulación entre los dos parámetros de un typedef.
- Cuando declare una struct, una union o un enum con un typedef, todas las reglas se aplicarán y deberá alinear el nombre del typedef con el nombre de la struct, la union o el enum.
- No puede declarar una estructura en un fichero .c.

II.6. Headers

Parte obligatoria

- En los ficheros header sólo se autoriza la inlusión de headers (del sistema o no), las declaraciones, los define, los prototipos y las macros.

- Todos los includes de .h se harán al inicio del fichero (.c o .h).
- Se protegerán los headers de la doble inclusión. Si el fichero es `ft_foo.h`, la macro testigo es `FT_FOO_H`.
- Los prototipos de función estarán exclusivamente en los ficheros .h.
- No se permite incluir un header (.h) que no se utiliza.

Parte aconsejada

- Se debe justificar toda inlusión de header, tanto en un .c como en un .h.

II.7. Macros y Pre-procesador

Parte obligatoria

- Las constantes de pre-procesador (#define) que cree sólo se usarán para asociar valores literales y constantes. Nada más.
- Los #define creados con el fin de burlar la norma o de ofuscar código prohibido quedan prohibidos. Este punto debe ser comprobable por seres humanos.
- Puede utilizar macros de las bibliotecas estándar, siempre que estén autorizadas en el proyecto objeto del ejercicio o proyecto.
- Los #define multilínea están prohibidos.
- Sólo se escribirán en mayÃºscula los nombres de macro.
- Hay que indentar los caracteres posteriores a un #if , un #ifdef o un #ifndef

II.8. Cosas prohibidas !

Parte obligatoria

- No puede utilizar:
  - for
  - do...while
  - switch
  - case
  - goto
- Les operadores ternarios '?' anidados
- Las tablas de tamaño variable (VLA - Variable Length Array)

II.9. Comentarios

Parte obligatoria

- Puede haber comentarios en todos los ficheros fuente.
- No puede haber comentarios en los cuerpos de las funciones.
- Ni la primera ni la Ã°ltima línea de Los comentarios contendrán texto. Todas las líneas intermediarias se alinearán con ellas y empezarán con '**'.
- Están prohibidas las zonas de comentarios que empiecen por //.

Parte aconsejada

- Los comnentarios deben estar en inglés y ser Ã°tiles.
- Los comentarios no justificarán una función mal concebida o inÃ°til.

II.10. Los ficheros

Parte obligatoria

- No se puede incluir un .c.
- No puede meter más de 5 definiciones de función en un .c.

II.11. Makefile

Parte obligatoria

- Las reglas $(NAME), clean, fclean, re y all son obligatorias.
- El proyecto no será considerado funcional si el Makefile hace un relink.
- En el caso de un proyecto mulibinario, además de las reglas anteriores, debe tener una regla all que compile los dos binarios así como una regla específica para cada binario compilado.
- Si el proyecto llama a una biblioteca de funciones (por ejemplo un `libft`), el makefile debe compilar automáticamente esa biblioteca.
- Los fuentes necesarios para la compilación de su programa deben citarse explícitamente en el Makefile.