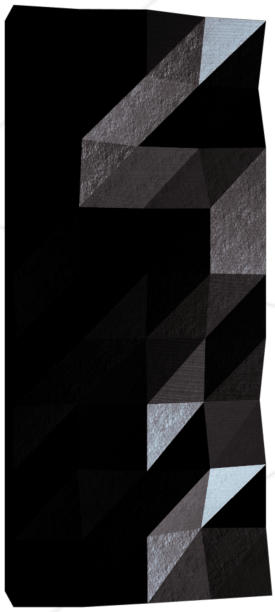

La Norma

Versión 3

Resumen: Este documento describe la norma de C en vigor en 42. Una norma de programación define un conjunto de reglas a seguir cuando se escribe código. La Norma se aplica por defecto a todos los proyectos C dentro del Common Core, y a cualquier proyecto en el que así se especifique.

Índice

Capítulo I

Prólogo

La `norminette` está en python y es open source.
Su repositorio está disponible en https://github.com/42School/norminette.
¡Pull requests, sugerencias e issues, bienvenidos!

Capítulo II

¿Por qué?

La Norma ha sido cuidadosamente elaborada para satisfacer muchas necesidades pedagógicas. Estas son las razones más importantes para todas las opciones que se presentan a continuación:

- Secuenciación: La programación implica dividir una tarea grande y compleja en una larga serie de instrucciones elementales. Todas estas instrucciones se ejecutarán secuencialmente, una tras otra. Un principiante que empieza a crear software necesita una arquitectura sencilla y clara para su proyecto, con una comprensión completa de todas las instrucciones individuales y el orden preciso de ejecución. Las sintaxis crípticas del lenguaje, que realizan múltiples instrucciones aparentemente al mismo tiempo, son confusas; las funciones que tratan de abordar múltiples tareas mezcladas en la misma porción de código son fuente de errores.
  La Norma pide que crees trozos de código sencillos, en los que la tarea única de cada trozo pueda ser claramente entendida y verificada, y en los que la secuencia de todas las instrucciones ejecutadas no deje lugar a dudas. Por eso se piden 25 líneas como máximo en las funciones y también por eso, están prohibidos los `for`, `do .. while`, o ternarios.

- Aspecto y sensación: cuando charlas con tus amigos y compañeros de trabajo durante el proceso normal de aprendizaje entre pares, y también durante las evaluaciones entre pares, no quieres dedicar tiempo a descifrar el código, sino hablar directamente de la lógica del código.
  La Norma pide que se utilice un aspecto específico, proporcionando instrucciones para la denominación de las funciones y variables, la sangría, las reglas de corchetes, el tabulador y los espacios en muchos lugares... . Esto te permitirá ver de un vistazo rápido el código de los demás ya que te resultarán familiar, e ir directamente al grano en lugar de perder tiempo leyéndolo antes para entenderlo. La Norma también es una marca registrada. Como parte de la comunidad 42, podrás reconocer el código escrito por otro estudiante o ex-alumno de 42 cuando te encuentres en el mercado laboral.

- Visión a largo plazo: hacer el esfuerzo de escribir un código comprensible es la mejor manera de mantenerlo. Cada vez que alguien, incluido tú, tenga que arreglar un error o añadir una nueva característica no tendrá que perder su precioso tiempo tratando de averiguar qué es lo que hace si tú previamente has hecho las cosas de forma correcta. Esto evitará situaciones en las que se dejan de mantener trozos de código solo porque eso supone una pérdida de tiempo, y eso puede marcar la

diferencia cuando hablamos de tener un producto exitoso en el mercado. Cuanto antes aprendas a hacerlo, mejor.

- Referencias: puedes pensar que algunas, o todas, las reglas incluidas en la Norma son arbitrarias, pero lo cierto es que lo hemos pensado mucho y leído qué hacer y cómo hacerlo. Te animamos a que busques en Google por qué las funciones deben ser cortas y hacer solo una cosa, por qué el nombre de las variables debe tener sentido, por qué las líneas no deben tener más de 80 columnas de ancho, por qué una función no debe tener muchos parámetros, por qué los comentarios deben ser útiles, etc, etc...

Capítulo III

La Norma

III.1. Denominaciones

- 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 minúsculas, cifras y '_' (Unix Case).
- Los nombres de ficheros y de directorios deben estar formados exclusivamente por minúsculas, cifras y '_' (Unix Case).
- Los caracteres que no forman parte de la tabla estándar ASCII están prohibidos.
- Las variables, funciones y cualquier otro indicador deben utilizar snake case: letras minúsculas solo y cada palabra separada por un guion bajo.
- Todos los identificadores (funciones, macros, tipos, variables, etc.) deben estar en inglés.
- Los objetos (variables, funciones, macros, tipos, archivos o directorios) deben tener los nombres más explícitos o más mnemotécnicos posibles.
- El uso de variables globales que no estén marcadas como const y static está prohibido y se considera un error de norma, a menos que el proyecto lo permita explícitamente.
- El archivo debe compilar. Un archivo que no se compila no se espera que pase la Norma.

III.2. Formato

- Debes incluir en tu código una sangría de 4 tabulaciones. No es lo mismo que 4 espacios normales, estamos hablando de tabulaciones reales.
- 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 el número de espacios que representa.
- Cada función debe estar separada por una nueva línea. Cualquier comentario o instrucción del preprocesador puede estar justo encima de la función. La nueva línea va después de la función anterior.
- Una sola instrucción por línea.
- Una línea vacía no debe contener ni espacios ni tabulaciones.
- Una línea no puede terminar ni en un espacio ni en una tabulación.
- Nunca puede haber dos espacios consecutivos.
- Hay que empezar una nueva línea después de cada corchete o al final de la estructura de control.
- A menos que sea el final de una línea, cada coma o punto y coma debe ir seguido de un espacio.
- Cada operador u operando debe estar separado por un - y solo un - espacio.
- Cada palabra clave de C debe ir seguida de un espacio, excepto las palabras clave para tipos (como int, char, float, etc.) así como sizeof.
- Cada declaración de variable debe tener una sangría en la misma columna de su ámbito.
- Los asteriscos que acompañan a los punteros deben ir pegados a los nombres de las variables.
- Una sola declaración de variable por línea.
- La declaración y la inicialización no pueden estar en la misma línea, excepto en el caso de las variables globales (cuando se permiten), las variables estáticas y las constantes.
- Las declaraciones deben estar al principio de una función.
- En una función, debe colocar una línea vacía entre las declaraciones de las variables y el resto de la función. No se permiten otras líneas vacías en una función.
- Las asignaciones múltiples están estrictamente prohibidas.

- Se puede añadir una nueva línea después de una instrucción o estructura de control, pero habrá que añadir una sangría con corchetes o un operador de asignación. Los operadores deben estar al principio de una línea.
- Las estructuras de control (if, while..) deben tener llaves, a menos que contengan una sola línea.
- Las llaves que siguen a funciones, declaradores o estructuras de control deben ir precedidas y seguidas de una nueva línea.

Ejemplo general:

```
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);
}
```

III.3. Funciones

- Una función tendrá como máximo 4 parámetros con nombre.
- Una función sin argumentos se prototipará explícitamente con el argumento void.
- Los parámetros de los prototipos de función deben tener un nombre.
- Cada función debe estar separada de la siguiente por una línea vacía.
- Se puede declarar un máximo de 5 variables por función.
- El retorno de una función se hará entre paréntesis.
- Cada función debe tener una única tabulación entre su tipo de retorno y su nombre.

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

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

III.4. Typedef, struct, enum y union

- Se debe poner una tabulación cuando declare una struct, un enum o una union.
- Cuando se declare una variable de tipo struct, enum o union, se pone sólo un espacio en el tipo.
- Cuando se declare una struct, una union o un enum con un typedef, se aplicarán todas las reglas.
- El nombre del typedef debe ir precedido de un tabulador.
- Todos los nombres de las estructuras de un mismo ámbito tienen que tener sangría.
- No se puede declarar una estructura en un archivo .c

III.5. Headers - a.k.a incluir archivos

- Lo que se permite en los archivos de cabecera es lo siguiente: inclusiones de cabecera (del sistema o no), declaraciones, defines, prototipos y macros.
- Todas las inclusiones deben estar al principio del archivo.
- No se puede incluir un archivo C.
- Los archivos de cabecera deben estar protegidos contra las dobles inclusiones. Si el archivo es `ft_foo.h`, su macro es `FT_FOO_H`.
- Las inclusiones de headers (.h) no utilizadas están prohibidas.
- Todas las inclusiones de cabecera deben estar justificadas en un archivo .c así como en un archivo .h.

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

int               g_variable;
struct        s_struct;

#endif
```

III.6. El 42 header - a.k.a iniciar un archivo con estilo

- Todos los archivos .c y .h deben comenzar con la cabecera estándar 42: un comentario de varias líneas con un formato especial que incluye información útil. La cabecera estándar está por defecto disponible en los ordenadores de los clústeres para varios editores de texto (emacs : con `C-c C-h`, vim con `:Stdheader` o `F1`, etc...).

- La cabecera 42 debe contener varias informaciones actualizadas, incluyendo el creador con nombre de usuario y correo electrónico, la fecha de creación, el nombre de usuario y la fecha de la última actualización. Cada vez que el archivo se guarda en el disco, la información debe actualizarse automáticamente.

III.7. Macros y Pre-processadores

- Las constantes de pre-procesador (#define) que se creen solo se usarán para asociar valores literales y constantes.
- 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.
- Pueden usarse las macros disponibles en las bibliotecas estándar, solo si éstas están permitidas en el ámbito del proyecto en cuestión.
- Macros multilínea están prohibidos.
- Los nombres de las Macros deben escribirse en mayúsculas.
- Los caracteres posteriores a un #if, un #ifdef o un #ifndef deben incluir una sangría.

III.8. ¡Cosas prohibidas!

- No puedes utilizar:
  - for
  - do...while
  - switch
  - case
  - goto
- Los operadores ternarios como '?'.
- VLAs - Variable Length Arrays.
- Tipo implícito en las declaraciones de variables

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

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

III.9. Comentarios

- No puede haber comentarios en los cuerpos de las funciones. Los comentarios deben estar al final de una línea, o en su propia línea.
- Los comnentarios deben estar en inglés y ser útiles.
- Los comentarios no justificarán una función mal concebida o inútil.

III.10. Archivos

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

III.11. Makefile

Los Makefiles no son revisados por la Norma, y deben ser revisados durante la evaluación por el estudiante.

- Las reglas $(NAME), clean, fclean, re y all son obligatorias.
- Si el Makefile hace un relink, el proyecto no será considerado funcional.
- En el caso de un proyecto multibinario, 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 archivos necesarios para la compilación de tu programa deben citarse explícitamente en el Makefile.