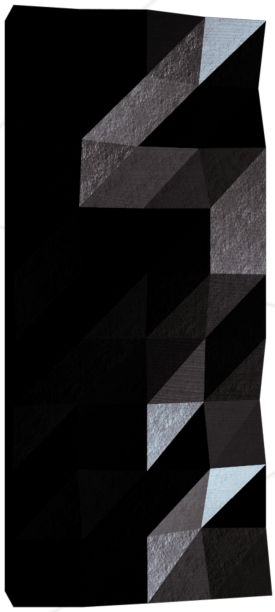

La Norma

Version 4

Resumen: Este documento describe la norma aplicable en 42. Una norma de programación define un conjunto de reglas a seguir al escribir código. La Norma se aplica a todos los proyectos de C dentro del Common Core por defecto, y a cualquier proyecto donde se especifique. This document describes the applicable standard (Norm) at 42. A programming standard defines a set of rules to follow when writing code. The Norm applies to all C projects within the Common Core by default, and to any project where it's specified.

Índice general

Capítulo I

Prefacio

La `norminette` está escrita en python y es de código abierto.
Su repositorio está disponible en https://github.com/42School/norminette.
¡Pull request, sugerencias e issues serán bien recibidos!

Capítulo II

¿Por qué?

La Norma ha sido cuidadosamente elaborada para cumplir con muchas necesidades pedagógicas. Aquí están las razones más importantes para todas las elecciones a continuación:

- Secuenciación: programar implica dividir una tarea grande y compleja en una larga serie de instrucciones elementales. Todas estas instrucciones se ejecutarán en secuencia: una tras otra. Un principiante que comienza a crear software necesita una arquitectura simple y clara para su proyecto, con una comprensión completa de todas las instrucciones individuales y el orden preciso de ejecución. Los lenguajes de programación crípticos que hacen múltiples instrucciones aparentemente al mismo tiempo son confusos, las funciones que intentan abordar múltiples tareas mezcladas en la misma porción de código son fuente de errores.
  La Norma te pide que crees piezas de código simples, donde la tarea única de cada pieza pueda ser claramente entendida y verificada, y donde la secuencia de todas las instrucciones ejecutadas no deje lugar a dudas. Por eso pedimos un máximo de 25 líneas en las funciones, también por qué se prohíben los `for`, `do .. while`, o ternarios.

- Aspecto: al intercambiar con tus amigos y compañeros de trabajo durante el proceso normal de aprendizaje entre pares, y también durante las evaluaciones entre pares, no quieres perder tiempo descifrando su código, sino hablar directamente sobre la lógica de la pieza de código.
  La Norma te pide que uses un aspecto específico, proporcionando instrucciones para el nombre de las funciones y variables, indentación, reglas de llaves, tabulaciones y espacios en muchos lugares... . Esto te permitirá echar un vistazo a otros códigos que te resultarán familiares y llegar directamente al punto en lugar de perder tiempo leyendo el código antes de entenderlo. La Norma también funciona como una marca registrada. Como parte de la comunidad 42, podrás reconocer el código escrito por otro estudiante o exalumno de 42 cuando estés en el mercado laboral.

- Visión a largo plazo: hacer el esfuerzo de escribir código comprensible es la mejor manera de mantenerlo. Cada vez que alguien más, incluyéndote a ti, tenga que corregir un error o agregar una nueva característica, no tendrá que perder su valioso tiempo tratando de entender lo que hace si previamente hiciste las cosas de la manera correcta. Esto evitará situaciones en las que los fragmentos de código dejen de ser mantenidos solo porque lleva 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 en realidad pensamos y leímos sobre qué hacer y cómo hacerlo. Te animamos encarecidamente a buscar por qué las funciones deben ser cortas y hacer una sola cosa, por qué el nombre de las variables debe tener sentido, por qué las líneas no deben ser más largas de 80 columnas, por qué una función no debe tener muchos parámetros, por qué los comentarios deben ser útiles, etc, etc, etc...

Capítulo III

La Norma

III.1. Denominación

- El nombre de una estructura debe comenzar con `s_`.
- El nombre de un typedef debe comenzar con `t_`.
- El nombre de un union debe comenzar con `u_`.
- El nombre de un enum debe comenzar con `e_`.
- El nombre de una variable global debe comenzar con `g_`.
- Los nombres de variables y funciones solo pueden contener minúsculas, dígitos y '_' (snake_case).
- Los nombres de archivos y directorios solo pueden contener minúsculas, dígitos y '_' (snake_case).
- Los caracteres que no forman parte de la tabla ASCII estándar están prohibidos.
- Las variables, funciones y cualquier otro identificador deben usar snake case. Sin letras mayúsculas, y cada palabra separada por un guión bajo.
- Todos los identificadores (funciones, macros, tipos, variables, etc.) deben estar en inglés.
- Los objetos (variables, funciones, macros, tipos, archivos o directorios) deben tener nombres lo más explícitos o mnemótecnicos posibles.
- El uso de variables globales que no están marcadas como const y static está prohibido y se considera un error de norma, a menos que el proyecto las permita explícitamente.
- El archivo debe compilar. Un archivo que no compila no se espera que pase la Norma.

III.2. Formato

- Debes indentar tu código con tabulaciones de 4 espacios. Esto no es lo mismo que 4 espacios normales, estamos hablando de verdaderas tabulaciones.
- Cada función debe tener un máximo de 25 líneas, sin contar las llaves de la función.
- Cada línea debe tener como máximo 80 columnas de ancho, incluidos los comentarios. Advertencia: 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 de preprocesador puede estar justo encima de la función. La nueva línea está después de la función anterior.
- Una instrucción por línea.
- Una línea vacía debe estar vacía: sin espacios o tabulaciones.
- Una línea no puede terminar con espacios o tabulaciones.
- No puedes tener dos espacios consecutivos.
- Debes comenzar una nueva línea después de cada llave de apertura o cierre o después de una estructura de control.
- Salvo que sea el final de una linea, cada coma o punto y coma debe ser seguido por 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 estar indentada en la misma columna dentro de su scope.
- Los asteriscos que acompañan a los punteros deben estar 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 para las variables globales (cuando se permiten), variables estáticas y constantes.
- Las declaraciones deben estar al principio de una función.
- En una función, debes colocar una línea vacía entre las declaraciones de variables y el resto de la función. No se permiten otras líneas vacías en una función.
- Las asignaciones múltiples están completamente prohibidas.
- Puedes agregar una nueva línea después de una instrucción o estructura de control, pero tendrás que agregar una indentación con llaves o operador de asignación. Los operadores deben estar al principio de una línea.

- Las estructuras de control (if, while, etc...) deben tener llaves, salvo que contengan una sola línea.
- Las llaves que siguen a las funciones, declaradores o estructuras de control deben estar precedidas y seguidas por una nueva línea.

Ejemplo general de formato:

```
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 no puede recibir más de 4 parámetros.
- Una función que no recibe argumentos debe ser prototipada con la palabra "voidçomo argumento.
- Los parámetros en los prototipos de las funciones deben tener nombre.
- Cada función debe estar separada de la siguiente por una línea vacía.
- No puedes declarar más de 5 variables por función.
- El retorno de una función debe estar entre paréntesis.
- Cada función debe tener una sola 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

- Agrega una tabulación al declarar un struct, enum o union.
- Al declarar una variable de tipo struct, enum o union, agrega un solo espacio en el tipo.
- Al declarar un struct, union o enum con un typedef, se aplican todas las reglas de indentación.
- El nombre del typedef debe ir precedido por una tabulación.
- Debes indentar todos los nombres de las estructuras en la misma columna dentro de su scope.
- No puedes declarar una estructura en un archivo .c.

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

- Las cosas permitidas en los archivos de cabecera son: inclusiones de cabecera (de sistema o no), declaraciones, defines, prototipos y macros.
- Todos los includes deben estar al principio del archivo.
- No puedes incluir un archivo C.
- Los archivos de cabecera deben estar protegidos de inclusiones dobles. Si el archivo es `ft_foo.h`, su macro de protección es `FT_FOO_H`.
- Las inclusiones de cabecera no utilizadas (.h) 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. La cabecera de 42 - a.k.a cómo empezar un archivo con estilo

- Todos los archivos .c y .h deben comenzar inmediatamente con la cabecera 42 estándar: un comentario de varias líneas con un formato especial que incluye información útil. La cabecera estándar está naturalmente disponible en las computadoras en clusters para varios editores de texto (emacs: usando `C-c C-h`, vim usando `:Stdheader` o `F1`, etc...).
- La cabecera 42 debe contener varias informaciones actualizadas, incluyendo el creador con login y correo electrónico, la fecha de creación, el login y la fecha de la última actualización. Cada vez que el archivo se guarda en disco, la información debe actualizarse automáticamente.

III.7. Macros y Preprocesadores

- Las constantes del preprocesador (o #define) que crees, deben usarse solo para valores literales y constantes.
- Todos los #define creados para evitar la norma y/o para ofuscar el código están prohibidos. Esta parte debe ser verificada por un humano.
- Puedes usar macros disponibles en bibliotecas estándar, solo si esas están permitidas en el alcance del proyecto dado.
- Las macros de varias líneas están prohibidas.
- Los nombres de las macros deben estar en mayúsculas.
- Debes indentar los caracteres que siguen a #if, #ifdef o #ifndef.
- Las instrucciones del preprocesador están prohibidas fuera del alcance global.

III.8. ¡Cosas Prohibidas!

- No puedes utilizar:
  - for
  - do...while
  - switch
  - case
  - goto
- Operadores ternarios como '?'.
- VLAs - o arrays de longitud variable.
- Tipos implícitos en 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

- Los comentarios no pueden estar dentro de los cuerpos de las funciones. Los comentarios deben estar al final de una línea, o en su propia línea.
- Tus comentarios deben estar en inglés. Y deben ser útiles.
- Un comentario no puede justificar la creación de un carryall o una mala función.

Un carryall o mala función generalmente viene con nombres que no son explícitos como f1, f2... para la función y a, b, i,.. para las declaraciones. Una función cuyo único objetivo es evitar la norma, sin un propósito lógico único, también se considera como una mala función. Por favor, recuerda que es preferible tener funciones claras y legibles que logren una tarea clara y simple cada una. Evita cualquier técnica de obfuscación de código, como la de una sola línea.

III.10. Archivos

- No puedes incluir un archivo .c
- No puedes tener más de 5 definiciones de funciones en un archivo .c.

III.11. Makefile

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

- El $(NAME), clean, fclean, re y all son reglas obligatorias.
- Si el makefile hace relinks, el proyecto se considerará no funcional.
- En el caso de un proyecto multibinario, además de las reglas anteriores, debes tener una regla que compile ambos binarios, así como una regla específica para cada binario compilado.
- En el caso de un proyecto que llama a una función de una librería no del sistema (por ejemplo: libft), tu makefile debe compilar esta biblioteca automáticamente.
- Todos los archivos fuente que necesitas para compilar tu proyecto deben estar nombrados explícitamente en tu Makefile.