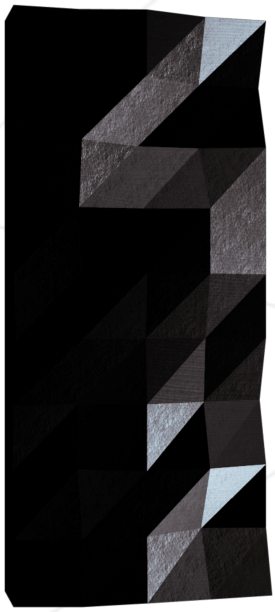

A Norma

Versão 3

Sumário: Este documento descreve o padrão aplicável (norma) na 42. Um padrão de programação define um conjunto de regras a seguir ao escrever um código. A norma aplica-se a todos os projetos C dentro do círculo interno por padrão, e para qualquer projeto onde é especificado.

Conteúdo

Capítulo I

Introdução

A norma é em Python e código aberto.
Seu repositório está disponível em https://github.com/42school/norminette.
Pull requests, sugestões e indicação de bugs são bem-vindos!

Capítulo II

Por quê ?

A Norma foi cuidadosamente elaborada para suprir diversas necessidades pedagógicas. Aqui estão alguns dos motivos mais importantes por trás das escolhas abaixo:

- Sequenciamento: programar implica dividir uma tarefa grande e complexa em uma série de instruções elementares. Todas essas instruções vão ser executadas em sequência: uma após a outra. Um iniciante, ao começar a criar software, precisa de uma arquitetura simples e clara para seu projeto, tendo o entendimento completo de todas as instruções individuais e da exata ordem de execução. Sintaxes de linguagens crípticas que aparentam executar múltiplas instruções ao mesmo tempo são confusas, funções que buscam abordar múltiplas tarefas misturadas na mesma porção de código são fontes de erros.
  A Norma pede que você escreva trechos simples de código cujas tarefas possam ser entendidas e verificadas facilmente, em que a sequência de execução das instruções não deixa dúvidas. Por este motivo que há o limite máximo de 25 linhas por função, e o porquê de `for`, `do .. while`, ou ternários serem proibidos.

- Estética: enquanto se relaciona com seus colegas durante o processo natural de aprendizado entre pares, e também durante as avaliações entre pares, você não quer gastar tempo decifrando o código deles, mas falar diretamente sobre a lógica daquele trecho de código.
  A Norma pede por uma estética específica, provendo instruções para nomear funções e variáveis, indentação, utilização das chaves, tabulações e espaços em diversos lugares... . Isso vai permitir que você olhe brevemente para o código de outros e o ache familiar, podendo ir direto ao assunto, ao invés de gastar tempo lendo o código antes de entendê-lo. A Norma também se caracteriza como uma marca registrada. Como parte da comunidade 42, você vai poder reconhecer código escrito por outro cadete ou alumni da 42 quando estiver no mercado de trabalho.

- Visão de longo prazo: esforçar-se para escrever um código compreensível é a melhor maneira de administrá-lo. Toda vez que alguém, incluindo você, precisar consertar um bug ou adicionar uma nova funcionalidade, não será necessário gastar tempo tentando entender o funcionamento se você escreveu seu código da maneira correta. Isso vai evitar situações em que trechos de código deixam de ser atualizados apenas por tomarem tempo, o que vai fazer a diferença ao falarmos sobre ter um produto bem sucedido no mercado. Quanto mais cedo aprender, melhor.

- Referências: você pode pensar que algumas, ou todas, as regras inclusas na Norma são arbitrárias, mas nós pensamos cuidadosamente e pesquisamos como elaborá-la. Nós encorajamos fortemente que você pesquise o porquê de funções precisarem ser curtas e possuir apenas uma tarefa, o porquê de nomes de variáveis precisarem ser compreensíveis, o porquê de linhas não poderem extrapolar o limite de 80 colunas de largura, o porquê de uma função não poder receber vários parâmetros, o porquê de comentários serem úteis, etc, etc, etc ...

Capítulo III

A norma

III.1 Denominação

- O nome de um struct deve começar por `s_`.
- O nome de um typedef deve começar por `t_`.
- O nome de um union deve começar por `u_`.
- O nome de um enum deve começar por `e_`.
- O nome de uma variável global deve começar por `g_`.
- Os nomes de variáveis e funções só podem conter letras minúsculas, dígitos e '_' (Unix Case).
- Os nomes de arquivos e diretórios só podem conter minúsculas, dígitos e '_' (Unix Case).
- Caracteres que não fazem parte da tabela ASCII padrão é proibida.
- Variáveis, funções e qualquer outro identificador deve usar snake case. Sem letras maiúsculas, e cada palavra separada por um sublinhado.
- Todos os identificadores (funções, macros, tipos, variáveis, etc.) devem estar em inglês.
- Objetos (variáveis, funções, macros, tipos, arquivos ou diretórios) devem ter os nomes mais explícitos ou mais mnemônicos possíveis.
- Usando uma variável global em um projeto onde não é explicitamente permitido é um erro de norma, exceto onde é obrigatório (manuseio de sinais, por exemplo).
- O arquivo deve compilar. Um arquivo que não compila não é esperado que passe na norma.

III.2 Formatação

- Você deve indentar seu código com tabulação de 4 espaços. Isto não é o mesmo que 4 espaços, estamos falando de tabulações reais aqui.
- Cada função deve ter no máximo 25 linhas, não contando suas próprias chaves '{}'.
- Cada linha deve ter no máximo 80 colunas de largura, comentários incluídos. Aviso: uma tabulação não conta como uma coluna, mas como o número de espaços que representa.
- Cada função deve ser separada por uma nova linha. Qualquer comentário ou instrução de pré-processador pode estar logo acima da função. A nova linha é após a função anterior.
- Apenas uma instrução por linha.
- Uma linha vazia deve estar vazia: sem espaços ou tabulações.
- Uma linha nunca pode terminar com espaços ou tabulações.
- Você nunca pode ter dois espaços consecutivos.
- Você precisa iniciar uma nova linha após cada chave '{}' ou final da estrutura de controle.
- A menos que seja o fim de uma linha, cada vírgula ou ponto-e-vírgula deve ser seguido por um espaço.
- Cada operador ou operando deve ser separado por um - e apenas um - espaço.
- Cada palavra-chave do C deve ser seguida por um espaço, exceto Palavras-chave para tipos (como int, char, float, etc.), bem como sizeof.
- Cada declaração de variável deve ser indentada na mesmo coluna para seu escopo.
- Os asteriscos que vão com ponteiros devem estar juntos aos nomes das variáveis.
- Uma única declaração variável por linha.
- Declaração e uma inicialização não podem estar na mesma linha, exceto para variáveis globais (quando permitido), variáveis estáticas e constantes.
- Declarações devem estar no início de uma função.
- Em uma função, você deve colocar uma linha vazia entre as declarações de variáveis e o restante da função. Nenhuma outra linha vazia é permitida em uma função.

- Atribuições múltiplas são estritamente proibidas.

- Você pode adicionar uma nova linha após uma instrução ou estrutura de controle, mas você terá que adicionar um indentação com chaves ou operador de atribuição. Os operadores devem estar no início de uma linha.

- Estruturas de controle (if, while ...) devem ter chaves, a menos que contenham uma única linha ou uma única condição.

General example:

```
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 Funções

- Uma função pode ter até 4 parâmetros definidos no máximo.
- Uma função que não tem argumentos deve ser explicitamente prototipada com a palavra "void"como o argumento.
- Parâmetros em protótipos de funções devem ser nomeados.
- Cada função deve ser separada da próxima por uma linha vazia.
- Você não pode declarar mais de 5 variáveis por função.
- O retorno de uma função deve estar entre parênteses.
- Cada função deve ter uma tabulação única entre seu tipo de retorno e seu nome.

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

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

III.4 Typedef, struct, enum e union

- Adicione uma tabulação ao declarar um struct, enum ou union.
- Ao declarar uma variável do tipo struct, enum ou union, adicione um único espaço no tipo.
- Ao declarar um struct, union ou enum com um typedef, todas as regras de indentação aplicam-se.
- O nome de um typedef deve ser precedido por uma tabulação.
- Você deve recuar todos os nomes de estruturas na mesma coluna para o escopo deles.
- Você não pode declarar uma estrutura em um arquivo .c.

III.5 Cabeçalhos

- As coisas permitidas em arquivos de cabeçalho são: Inclusões de cabeçalho (sistema ou não), declarações, definições, protótipos e macros.
- Todas inclusões devem estar no início do arquivo.
- Você não pode incluir um arquivo C.
- Os arquivos de cabeçalho devem ser protegidos contra inclusões duplas. Se o arquivo é `ft_foo.h`, a macro que o acompanhará é `FT_FOO_H`.
- Inclusões de cabeçalho não utilizadas (.h) são proibidas.
- Todas as inclusões de cabeçalho devem ser justificadas em um arquivo .c bem como em um arquivo .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 O Header da 42

- Todo arquivo .c e .h deve, imediatamente, começar com o header da 42: um comentário multilinha com um formato específico, incluindo informações úteis. O header padrão se encontra naturalmente disponível nos computadores dos clusters para diversos editores de texto (emacs: usando `C-c C-h`, vim: usando `:Stdheader` ou `F1`, etc...)

- O header da 42 deve conter informações atualizadas, incluindo o criador com login e email, a data de criação e a data da atualização mais recente.

III.7 Macros e pré-processadores

- Constantes do pré-processador (ou #define) que você criar devem ser usadas apenas para valores literais e constantes.
- Todos #define criados para ignorar a norma e / ou ofuscação de código são proibidos. Esta parte deve ser verificada por um humano.
- Você pode usar macros disponíveis em bibliotecas padrão, apenas se estas são permitidas no escopo do projeto.
- Macros multilinhas são proibidas.
- Nomes de macro devem ser todos maiúsculos (uppercase).
- Você deve recuar caracteres que seguirem #if, #ifdef or #ifndef.

III.8 Coisas proibidas!

- Você não tem permissão para usar:
  - for
  - do...while
  - switch
  - case
  - goto
- Operadores ternários como '?'.
- VLAs - Matrizes de comprimento variável.
- Tipo implícito em declarações variáveis.

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

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

III.9 Comentários

- Os comentários não podem estar dentro do corpo das funções. Os comentários devem estar no final de uma linha, ou em sua própria linha
- Seus comentários devem estar em inglês. E eles devem ser úteis.
- Um comentário não pode ser usado para justificar uma função mal feita.

III.10 Arquivos

- Você não pode incluir um arquivo .c.
- Você não pode ter mais de 5 definições de função em um arquivo .c.

III.11 Makefile

Makefiles não são verificados pela norma e devem ser verificados durante a avaliação pelo estudante.

- AS regras $(NAME), clean, fclean, re and all são obrigatórias.
- Se o makefile fizer relink, o projeto será considerado não funcional.
- No caso de um projeto multibinário, além das regras acima, você deve ter uma regra que compila ambos os binários, bem como uma regra específica para cada binário compilado.
- No caso de um projeto que chama uma função de uma biblioteca não-sistema (por exemplo: `libft`), seu makefile deve compilar esta biblioteca automaticamente.
- Todos os arquivos de origem que você precisa compilar seu projeto deve ser explicitamente nomeado em seu makefile.