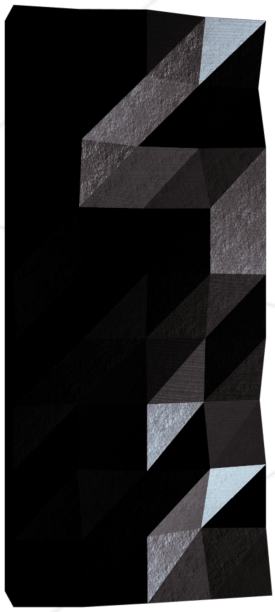

The Norm

Version 3

Summary: 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 Inner Circle by default, and to any project where it's specified.

Contents

Chapter I

Foreword

The Norm is in python and open source.
Its repository is available at https://github.com/42School/norminette.
Pull requests, suggestions and issues are welcome!

Chapter II

The Norm

II.1 Denomination

- A structure's name must start by `s_`.
- A typedef's name must start by `t_`.
- A union's name must start by `u_`.
- An enum's name must start by `e_`.
- A global's name must start by `g_`.
- Variables and functions names can only contain lowercases, digits and '_' (Unix Case).
- Files and directories names can only contain lowercases, digits and '_' (Unix Case).
- Characters that aren't part of the standard ASCII table are forbidden.
- Variables, functions, and any other identifier must use snake case. No capital letters, and each word separated by an underscore.
- All identifiers (functions, macros, types, variables, etc.) must be in English.
- Objects (variables, functions, macros, types, files or directories) must have the most explicit or most mnemonic names as possible.
- Using a global variable in a project where it's not explicitly allowed is a norm error, except where it's mandatory (signal handling for example).
- The file must compile. A file that doesn't compile isn't expected to pass the Norm.

II.2 Formatting

- You must indent your code with 4-space tabulations. This is not the same as 4 average spaces, we're talking about real tabulations here.
- Each function must be maximum 25 lines, not counting the function's own curly brackets.
- Each line must be at most 80 columns wide, comments included. Warning: a tabulation doesn't count as a column, but as the number of spaces it represents.
- Each function must be separated by a newline. Any comment or preprocessor instruction can be right above the function. The newline is after the previous function.
- One instruction per line.
- An empty line must be empty: no spaces or tabulations.
- A line can never end with spaces or tabulations.
- You can never have two consecutive spaces.
- You need to start a new line after each curly bracket or end of control structure.
- Unless it's the end of a line, each comma or semi-colon must be followed by a space.
- Each operator or operand must be separated by one - and only one - space.
- Each C keyword must be followed by a space, except for keywords for types (such as int, char, float, etc.), as well as sizeof.
- Each variable declaration must be indented on the same column for its scope.
- The asterisks that go with pointers must be stuck to variable names.
- One single variable declaration per line.
- Declaration and an initialisation cannot be on the same line, except for global variables (when allowed), static variables, and constants.
- Declarations must be at the beginning of a function.
- In a function, you must place an empty line between variable declarations and the remaining of the function. No other empty lines are allowed in a function.
- Multiple assignments are strictly forbidden.
- You may add a new line after an instruction or control structure, but you'll have to add an indentation with brackets or affectation operator. Operators must be at the beginning of a line.
- Control structures (if, while..) must have braces, unless they contain a single line or a single condition.

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);
}
```

II.3 Functions

- A function can take 4 named parameters maximum.
- A function that doesn't take arguments must be explicitely prototyped with the word "void" as the argument.
- Parameters in functions' prototypes must be named.
- Each function must be separated from the next by an empty line.
- You can't declare more than 5 variables per function.
- Return of a function has to be between parenthesis.
- Each function must have a single tabulation between its return type and its name.

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

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

II.4 Typedef, struct, enum and union

- Add a tabulation when declaring a struct, enum or union.
- When declaring a variable of type struct, enum or union, add a single space in the type.
- When declaring a struct, union or enum with a typedef, all indentation rules apply. You must align the typedef's name with the struct/union/enum's name.
- You must indent all structures' names on the same column for their scope.
- You cannot declare a structure in a .c file.

II.5 Headers

- The things allowed in header files are: header inclusions (system or not), declarations, defines, prototypes and macros.
- All includes must be at the beginning of the file.
- You cannot include a C file.
- Header files must be protected from double inclusions. If the file is `ft_foo.h`, its bystander macro is `FT_FOO_H`.
- Unused header inclusions (.h) are forbidden.
- All header inclusions must be justified in a .c file as well as in a .h file.

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

int g_variable;
struct s_struct;

#endif
```

II.6 Macros and Pre-processors

- Preprocessor constants (or #define) you create must be used only for literal and constant values.
- All #define created to bypass the norm and/or obfuscate code are forbidden. This part must be checked by a human.
- You can use macros available in standard libraries, only if those ones are allowed in the scope of the given project.
- Multiline macros are forbidden.
- Macro names must be all uppercase.
- You must indent characters following #if, #ifdef or #ifndef.

II.7 Forbidden stuff!

- You're not allowed to use:
  - for
  - do...while
  - switch
  - case
  - goto
- Ternary operators such as '?'.
- VLAs - Variable Length Arrays.
- Implicit type in variable declarations

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

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

II.8 Comments

- Comments cannot be inside functions' bodies. Comments must be at the end of a line, or on their own line
- You comments must be in English. And they must be useful.
- A comment cannot justify a "bastard" function.

II.9 Files

- You cannot include a .c file.
- You cannot have more than 5 function-definitions in a .c file.

II.10 Makefile

Makefiles aren't checked by the Norm, and must be checked during evaluation by the student.

- The $(NAME), clean, fclean, re and all rules are mandatory.
- If the makefile relinks, the project will be considered non-functional.
- In the case of a multibinary project, in addition to the above rules, you must have a rule that compiles both binaries as well as a specific rule for each binary compiled.
- In the case of a project that calls a function from a non-system library (e.g.: `libft`), your makefile must compile this library automatically.
- All source files you need to compile your project must be explicitly named in your Makefile.