Стандарты написания кода в 21-school: The Norm

Watch

Day 02 Piscine C

bjesse

2096 2019-01-29

Среди мануалов и файлов школы затерялся замечательный файл The Nom - Coding Standard - Academy+Plus, в котором описываются стандарты написания кода на c, без соблюдения которых наши программы никогда не смогут пройти проверку больше чем на 0 баллов.

Для проверки кода надо открыть в терминале в папку с файлами с исходным кодом и запустить в терминале norminette -R CheckForbiddenSourceHeader (© @ghildega).

Сам файл можно найти здесь:
https://cdn.intra.42.fr/pdf/pdf/960/norme.en.pdf 526

Что это за зверь такой - "стандарты написания кода"? https://ru.wikipedia.org/wiki/Стандарт_оформления_кода 131

Он полностью на английском, поэтому хочу поделиться с вами своим конспектом-переводом. Я буду рад, если вы поможете мне уточнить перевод в комментариях! Для этого достаточно владеть английским на уровне выше INTERMEDIATE и не пользоваться так же активно гугл-перевордчиком, как я)

И, как говорится, “Ставьте лайки, подписывайтесь”, ставьте uovotes (Стрелочка вверх), комментируйте)

ВНИМАНИЕ! СРЕДИ БУКВ НИЖЕ СМОЖЕТ БЫТЬ МНОЖЕСТВО ОШИБОК И НЕТОЧНОСТЕЙ! СВЕРЯЙТЕСЬ С ОРИГИНАЛОМ: https://cdn.intra.42.fr/pdf/pdf/960/norme.en.pdf 526

Глава 1

I.1 Зачем нужен стандарт?
▪ Для того, чтобы кто угодно (студенты, сотрудники и даже вы) могли легко прочитать ваш код и разобраться в нем
▪ Чтобы ваш код был простым и коротким
▪

I.2 Стандарт в 21 школе(?)
▪ Все ваши программы на C должны соответствовать стандарту - это будет проверяться! Если вы сделаете ошибку и ваш код не будет соответствовать стандарту - вы получите 0 за задание или даже за весь проект!
▪ (так же для проверяющих есть специальная программа “Norminette”, помогающая определить соответствие кода стандарту)

I.3 Советы
▪ Скоро вы поймете, что норма не так страшна, как кажется. Наоборот, она помогает вам: вы заметите, как легко будет читать код ваших одноклассников ваш!
▪ Для проверяющей системы все равно, сколько Nom Error в вашем коде: хоть 1, хоть 10
▪ Помните о стандарте! В начале это будет замедлять вашу работу, но скоро перейдет на уровень рефлексов!

I.4 Отказ от ответственности
▪ Norminette - это программа. И кака всякая программа, она подвержена ошибкам. Но скорее всего ошибка на вашей стороне)

Глава 2 Нормы:

II.1 Название переменных:

```
▪ Имя structure  должно начинаться с s_.
▪ Имя typedef должно начинаться с t_.
▪ Название union должно начинаться с u_.
▪ Имя enum должно начинаться с e_.
▪ Глобальное имя должно начинаться с g_.
▪ Имена переменных и функций могут содержать только строчные буквы, цифры и символы “_”.
▪ Имена файлов и каталогов могут содержать только строчные буквы, цифры и ‘’_”
▪ Файл должен скомпилироваться.
▪ Символы, которые не являются частью стандартной таблицы ascii, запрещены.
```

Рекомендации:

```
▪ Объекты (переменные, функции, макросы, типы, файлы или каталоги) должны иметь понятные имена, благ
одаря которым среди них можно будет легко ориентироваться и запоминать их.
▪ Только переменные-счетчики могут быть названы по вашему вкусу!
▪ Если вы сокращаете слово в названии переменной – то оно должно оставаться понятным! Если имя содер
жит более одного слова – слова должны быть разделены символом “_”.
▪ Все идентификаторы (функции, макросы, типы, переменные и т. д.) Должны быть на английском языке.
▪ Любое использование глобальной переменной должно быть оправданным.
```

II.2 Оформление:

▪ Все ваши файлы должны начинаться со стандартного школьного заголовка (с первой строки файла). Этот заголовок по умолчанию доступен с emacs и vim в дампах.
▪ В качестве откупа используется 4-пробельная табуляция (Внимание! не 4 пробела!)
▪ Каждая функция должна содержать не более 25 строк, не считая собственных фигурных скобок функции.
▪ Каждая строка должна содержать не более 80 символов, включая комментарии. Предупреждение: один зна к табуляции считается за 4 символа!
▪ Одна инструкция на строку.
▪ Пустая строка должна быть пустой: без пробелов и табов
▪ Строка никогда не может заканчиваться пробелами или табами
▪ Вам нужно начинать новую строку после каждой фигурной скобки или конца управляющей структуры.
▪ За каждой запятой или точкой с запятой должен стоять пробел, если это не конец строки
▪ Каждый оператор (двоичный или троичный) или операнд должен быть разделен одним и только одним проб елом.
▪ Каждое ключевое слово C должно сопровождаться пробелом, за исключением ключевых слов для типов (та ких как int, char, float и т. Д.), А также sizeof.
▪ ?????Каждое объявление переменной должно иметь отступ в том же столбце.(Each variable declaration must be indented on the same column.)
▪ Звездочки указателей должны быть вплотную к именам переменных.
▪ Одно объявление переменной на строку
▪ Нельзя делать объявление и инициализацию в одной строке, за исключением глобальных и статических п еременных.
▪ Объявления переменных должны быть в начале функции и должны быть отделены пустой строкой.
▪ Не должно быть пустой строки между объявлениями или реализациями.
▪ Многократные назначения строго запрещены???
▪ Вы можете добавить новую строку после инструкции или управляющей структуры, но вам придется добави ть отступ в скобках или оператор присваивания. Операторы должны быть в начале строки. (You may add a new line after an instruction or control structure, but you'll have to add an indentation with brack ets or affectation operator. Operators must be at the beginning of a line.)

II.3 Параметры функции:

▪ Функция должна принимать максимум 4 именованных параметров.
▪ Функция, которая не принимает аргументы, должна быть явно прототипирована с помощью слово «void» в качестве аргумента.

II.4 Функции:

▪ Параметры в прототипах функций должны быть названы.
▪ Каждая функция должна быть отделена от следующей пустой строкой.
▪ Вы не можете объявить более 5 переменных в блоке.
▪ Возвращение функции должно быть между скобками.

Рекомендации:
▪ Идентификаторы ваших функций должны быть выровнены в одном и том же файле. То же самое касается заголовочных файлов.

II.5 Typedef, struct, enum и union Обязательная часть

▪ Добавьте табуляцию при объявлении struct, enum or union.
▪ При объявлении переменной типа struct, enum или union добавьте один пробел в тип.
▪ Добавляйте табуляцию между двумя параметрами typedef.
▪ При объявлении struct, enum or union с помощью typedef применяются все правила. Вы должны выровнят ь имя typedef с именем struct / union / enum.
▪ Вы не можете объявить struct в файле .c.

II.6 Обязательная часть заголовков

▪ В заголовочных файлах разрешены следующие вещи: включения заголовка (системные или нет), объявлени я, определения, прототипы и макросы.
▪ Все включения (.c или .h) должны быть в начале файла.
▪ Заголовки защищены от двойных включений. Если файл ft_foo.h, его bystander macro это FT_FOO_H.
▪ Прототипы функций должны быть исключительно в файлах .h.
▪ Неиспользуемые заголовки (.h) запрещены.
▪ Рекомендации
▪ Все включения заголовка должны быть обоснованы как в файле .c, так и в файле .h.

II.7 Макросы и препроцессоры

▪ Константы препроцессора (или #define), которые вы создаете, должны использоваться только для связывания литеральных и константных значений.
▪ Запрещены все #define, созданные для обхода нормального и / или запутанного кода. Этот момент должен быть проверен человеком.
▪ Вы можете использовать макросы, доступные в стандартных библиотеках, только если они разрешены в рамках данного проекта.
▪ Многострочные макросы запрещены.
▪ Все имена макросов должны быть в верхнем регистре!
▪ Вы должны сделать отступ для символов после #if, #ifdef или #ifndef.

II.8 Запрещено!!

▪ Вам запрещено использовать:
1. for
2. do...while
3. switch
4. case
5. goto
▪ Вложенные троичные операторы, такие как «?».
▪ VLA – массивы переменной длины.

II.9 Комментарии

▪ Вам разрешено комментировать свой код в исходных файлах.
▪ Комментарии не могут быть внутри(inside) тела функции.
▪ Комментарии начинаются и заканчиваются одной строкой. Все промежуточные линии должны совпадать и начинаться с «**».
▪ Нет комментариев с //.

Рекомендации:
▪ Ваши комментарии должны быть на английском языке. И они должны быть полезными.
▪ Комментарий не может оправдать «ублюдочную» функцию.

II.10 Файлы

▪ Вы не можете включить файл .c.
▪ Вы не можете иметь более 5 определений функций в файле .c.

II.11 Makefile

▪ $ (NAME), clean, fclean, re и все правила являются обязательными.
▪ Если make-файл перелинкует, проект будет считаться нефункциональным.(If the makefile relinks, the project will be considered non-functional)
▪ В случае мультибинарного проекта, в дополнение к правилам, которые мы видели, у вас должно быть правило, которое компилирует оба двоичных файла, а также определенное правило для каждого скомпилированного двоичного файла.
▪ В случае проекта, который вызывает библиотеку функций (например, libft), ваш make-файл должен автоматически скомпилировать эту библиотеку.
▪ Все исходные файлы, которые вам нужны для компиляции вашего проекта, должны быть явно названы в вашем Makefile.