Оформление кода

Изучите фрагмент кода ниже. Вам понятно что он делает?

```
uint8_t f(uint32_t a)
{if (a % 4 != 0) return 0; else if (a % 100 == 0 && a % 400 != 0) return 0; else return 1;}
```

Эта функция возвращает `1`, если год високосный и `0`, если он обычный.

> Алгоритм: год, номер которого кратен 400, — високосный; остальные годы, номер которых кратен 100, — не високосные; остальные годы, номер которых кратен 4, — високосные.

В приведённом фрагменте совершенно непонятно, что происходит и зачем это происходит. Название функции и переменной ни о чём не говорит, форматирование ужасное, — придётся потратить больше времени, чтобы разобраться. Конечно, это не ваш код, но подобный «стиль» часто встречается у начинающих разработчиков. Они сразу начнут отстаивать свою позицию, мол «А мне понятен код, — Я же его написал!». Память не вечна и вернувшись к написанию программы через, скажем, месяц — они вряд ли вспомнят что имели ввиду. Упростите себе и окружающим жизнь заранее.

Ваш код должен быть понятен не только вам, но и любому встречному. Стремитесь, чтобы код выглядел, как рассказ. Если название переменной или функции сразу не говорит вам о том, для чего она нужна, то, скорее всего, вы сделали что-то не так. Например, если это счетчик, назовите его `counter` (с англ. «счетчик»). Если функция возвращает температуру, назовите ее `get_temperature()` (с англ. «получить температуру»). Если вы не знаете, как пишется слово по английский, лучше загляните в словарик, а не используйте транслитерацию.

Давайте перепишем фрагмент выше, что бы код был более читаем и понятен.

```
#define DIV_BY_4(x)     (x % 4   == 0)
#define DIV_BY_100(x)   (x % 100 == 0)
#define DIV_BY_400(x)   (x % 400 == 0)
// ...
typedef enum {
    IS_REGULAR = 0,
    IS_LEAP    = 1,
} YEAR_t;
// ...
YEAR_t is_leap(uint32_t year) {
    if (!DIV_BY_4(year)) {
        return IS_REGULAR;
    } else if (DIV_BY_100(year) && !DIV_BY_400(year)) {
        return IS_REGULAR;
    } else {
        return IS_LEAP;
    }
}
```

Да, получилось больше букв, но сам бинарный файл будет весить столько же! Макросы просто подставятся в код, а `YEAR_t` — обычное целочисленное. Прочитайте строчку ниже, одновременно посматривая на код.

> Если год *не кратен 4*, то он точно обычный, а если он *кратен четырём*, *кратен на ста* и при этом *не кратен 400*, то он тоже обычный, в противном случае это високосный год.

Крупные компании вводят свои стандарты по оформлению кода, взгляните как код на C++ рекомендует оформлять **Google**. Весь код в курсе, будет следовать этим рекомендациям.

Часто можно встретить **венгерскую нотацию** в именовании. Её суть сводится к добавлению приставок к названиям функций. Вот пример из операционной системы реального времени, FreeRTOS:

```
void vTaskResume( TaskHandle_t xTaskToResume );
```

Буква `v` в начале говорит о том, что функция ничего не возвращает (`void`), далее идёт приставка в виде слова `Task`, она указывает, что эта функция из файла (модуля) `task.c`. В нашем проекте будет много модулей, но мы не будем использовать венгерскую нотацию полностью, а будем указывать только название модуля, откуда берётся функция. Есть речь идёт о кнопке, то приставка будет `button_`, если о светодиоде, то `led_` и т.д.

```
void mcu_init() {
    led_init();
    button_init();
    // ...
}
```

Если в названии больше одного слова, то они будут разделяться символом подчёркивания `_`, а не в согласно **CamelCase**, как во FreeRTOS.

Назад | **Оглавление** | **Дальше**