

# **Правила оформления исходного кода**

версия 1.2

## 1. Идентификаторы

Имена переменных, функций, типов и др. должны быть осмысленными. Назначение должно быть очевидным из названия. Нужно избегать однобуквенных имен, кроме как для временных переменных и переменных цикла.

Имена записываются в нижнем регистре. Если имя состоит из нескольких слов, слова разделяются символом подчеркивания (например, `back_color`). Имя не должно начинаться с символа подчеркивания

Имена макроопределений записываются в верхнем регистре.

Если тип определен через `typedef`, то имя типа должно оканчиваться на «`_t`».

## 2. Использование пробелов и отступов

Всегда необходимо использовать четыре пробела для всех уровней отступа.

Объявления и определения функций, описание глобальных переменных, директивы препроцессора всегда должны примыкать к левой границе.

Операторные скобки следует располагать с одинаковым отступом от левой границы текста. Текст, находящийся между { и } следует смещать вправо на четыре символа относительно смещения { и }.

## 3. Оператор if

Правильно	Неправильно
<pre>if (a &lt; b)     op1;</pre>	<pre>if(a &lt; b) op1;</pre>
<pre>if (a &lt; b) {     op1;     op2; } else {     op3;     op4; }</pre>	<pre>if (a &lt; b) {     op1;     op2; } else {     op3;     op4; }</pre>

Обратите внимание на наличие пробела между ключевым словом `if` и открывающей скобкой ( .

В случае большого количества альтернатив в условном операторе нужно использовать так называемый «каскадный» оператор.

## 4. Оператор for

Правильно	Неправильно
<pre>for (int i = 0; i &lt; 10; i++) {     op1;     op2; }</pre>	<pre>for (int i = 0; i &lt; 10; i++) {     op1;     op2; }</pre>

<code>for (int i = 0; i &lt; 10; i++)   op;</code>	
--	--

## 5. Оператор while

Правильно	Неправильно
<code>while (x &lt; j) {     op1;     op2; }</code>	<code>while (x &lt; j) {     op1;     op2; }</code>
<code>while (x &lt; j)   op;</code>	

## 6. Оператор do-while

Правильно	Неправильно
<code>do {     op1;     op2; } while (j &lt;= 25);</code>	<code>do {     op1;     op2; } while (j &lt;= 25);</code>

## 7. Оператор switch

Правильно	Неправильно
<code>switch (a) {     case 1:         op1;         op2;         break;      case 2:         op1;         op2;         break;      default:         op1;         op2;         break; }</code>	

## 8. Использование пустых строк и пробельных символов

Пустые строки могут повысить читабельность путем группирования секций кода, которые логически связаны между собой. Пустые строки должны использоваться в следующих местах:

- после группы директив `#include` для стандартных фалов;
- после конца группы директив `#include`;
- между группами директив `#define`;
- между определениями типов;
- между реализациями функций;
- между группами операторов, реализующих структурные единицы кода.

Пробелы запрещены к использованию:

- до или после операции «.», «->»;
- между именем функции и открывающей скобкой;
- между унарным оператором и его операндом;
- между выражением приведения (cast) и приводимым выражением;
- после открывающей скобки или перед закрывающей;
- после открывающей квадратной скобки [ и перед закрывающей ];
- перед точкой с запятой;

## 9. Описание переменных

Переменные описываются в начале блока.

Все переменные с их типами должны быть описаны на различных строках. Допускается описание переменных одного типа в одной строке, если они связаны между собой по смыслу:

```
int x, y, z;
```

При этом переменные, различные по смыслу, следует размещать в отдельной строке, даже если они имеют одинаковый тип:

```
int x, y;  
int left;
```

По возможности при описании переменные должны быть сразу же проинициализированы.

Когда объявляется список параметров для функции, пользуйтесь следующими рекомендациями:

- придерживайтесь следующего порядка в параметрах: сначала входные параметры, затем входные/выходные, затем выходные;
- используйте const для параметров-указателей, которые адресуют не изменяемые при работе функции данные.

```
double get_max(int n, const double* arr);
```

```
void sort(int n, double *arr);
```

## 10. Оформление заголовочных файлов

Заголовочному файлу (\*.h) соответствует один файл реализации (\*.c). Файлы имеют одинаковое имя, но различаются расширением.

Заголовочный файл должен быть оформлен следующим образом:

```
/**  
 * Описание назначения модуля  
 */  
  
#ifndef __HEADER__H__  
  
#define __HEADER__H__  
  
// Директивы препроцессора
```

```
// Описание типов  
  
// Объявления функций  
  
#endif      // __HEADER__H__
```

В заголовочный файл с помощью директивы `#include` включаются только те заголовочные файлы, которые необходимы для описания содержащихся в нем типов и функций.

Заголовочный файл обязательно включается в соответствующий файл реализации.

Макроопределения с помощью директивы `#define` не следует выносить в заголовочный файл, если они не используются за пределами соответствующего файла реализации.

## 11. Комментарии

Каждый файл должен начинаться с комментария в формате doxygen, который описывает назначение этого файла.

У каждой функции должен быть комментарий в формате doxygen, который поясняет ее назначение, описывает сделанные при ее реализации допущения и все параметры этой функции.

У каждой структуры должен быть комментарий не только описывающий ее назначение, но и комментарий к каждому члену структуры.

Глобальные переменные (если они есть в программе) должны быть обязательно прокомментированы.

## 12. Разное

На каждой строке должен располагаться только один оператор.

Для описания типа структурной переменной не рекомендуется использовать `typedef`, если речь не идет о реализации абстрактного типа данных.

Заголовочные файлы, подключаемые с помощью директивы `#include`, нужно перечислять в следующем порядке:

- стандартные заголовочные файлы;
- заголовочные файлы внешних (чужих) библиотек;
- собственные заголовочные файлы.

Если длина строки превышает 80 символов, ее необходимо разбить на две и более. Разбиение исходной строки лучше выполнять по знакам операций (см. примеры ниже).

```
if (a < b &&  
    c < d)  
  
a = b + c /  
    (d + e);
```

Уровень вложенности не должен превышать трех.

```
while (a < 5)                // ok
{
    if (b < 10)              // ok
    {
        if (c > 0)          // ok
        {
            if (d == 5)      // not ok!
            {
                ...
            }
        }
    }
}
```