B. Правила оформления кода

Алексей Мартынов Василий Касилов

21 октября 2025 г.

Версия 1.0

©2024—2025, доступно по лицензии Creative Commons “Attribution-NonCommercial-NoDerivs” 3.0

1. Исходные тексты программы должны содержать не более одного выражения на одной строке.
2. Недопустимо писать функции в одну строку; сигнатура функции всегда должна идти на отдельной строке или нескольких, если сигнатура имеет большую длину.
3. Длина одной строки не должна превышать 120 символов в большинстве случаев и 140 символов всегда.
4. Строки не должны заканчиваться пробелами и/или символами табуляции.
5. Файл должен заканчиваться символом конца строки.
6. Все имена должны быть составлены из правильных английских слов или очевидных для читающего сокращений или аббревиатур.
7. Переменные определяются максимально близко к месту использования. Они должны иметь минимально возможную область видимости.
8. Все переменные должны быть инициализированы в точке определения.
9. Все неизменяющиеся переменные должны быть отмечены квалификатором **`const`**.
10. Аргументы функции, переданные по ссылке или по указателю и не изменяющиеся внутри функции, должны быть отмечены квалификатором **`const`**. Параметры, передаваемые по значению *могут* иметь квалификатор **`const`** в определениях функций, но он должен отсутствовать в объявлении функции.
11. Единицей отступа являются 2 пробела.
12. Отступы должны отражать логическую структуру программы:
    (a) операторы, выполняемые внутри циклов, должны иметь отступ на 1 больший, чем заголовок цикла;
    (b) операторы, выполняемые внутри **`if`** и **`else`**, должны иметь отступ больший, чем строки с **`if`** и **`else`**;
    (c) Метки операторов **`case`** внутри **`switch`** должны быть выравнены на тот же отступ, что **`switch`**.
    (d) Операторы внутри **`switch`** должны иметь на 1 больший отступ, чем оператор **`switch`**.
13. Все операторы, входящие в тела циклов и условий, заключаются в фигурные скобки, даже если это один оператор.
14. Фигурные скобки должны быть расставлены единообразно. Допускаются 2 варианта:
    (a) Открывающая фигурная скобка переносится на следующую строку и имеет тот же отступ, что и предыдущая строка, закрывающая скобка находится на отдельной строке, и ее отступ совпадает с открывающей. Операторы внутри скобок имеют дополнительный отступ. Например:

```
if (condition)
{
  c = a + b;
}
else
{
  c = a - b;
}
```

    Ключевое слово **`else`** должно находится на отдельной строке.
    (b) Открывающая фигурная скобка завершает строку с оператором и отделяется слева пробелом, закрывающая скобка находится на отдельной строке, и ее отступ совпадает с отступом строки, содержащей открывающую скобку (так называемые «египетские скобки»). Операторы внутри скобок имеют дополнительный отступ. Например:

```
if (condition) {
  c = a + b;
} else {
  c = a - b;
}
```

Ключевое слово **else** должно находиться на строке вместе с закрывающей скобкой.

Открывающая фигурная скобка в реализации функции *всегда* переносится на другую строку:

```
1  void foo()
2  {
3    …
4  }
```

Способ расстановки фигурных скобок должен быть единообразен во всех работах.

15. Ключевые слова **if** и **else** могут быть объединены на одной строке для уменьшения отступов:

```
if (condition1) {
  c = a + b;
} else if (condition2) {
  c = a - b;
}
```

16. Ключевые слова **if**, **for**, **while**, **switch**, **catch** всегда отделяются пробелом от последующей открывающей скобки. В то же время список параметров при объявлении, определении и вызове функций никогда не отделяется пробелами.

17. Отдельные символы операторов в выражениях выделяются пробелами с 2-х сторон, кроме операторов:

    (a) индексации — квадратные скобки не отделяются пробелами от предшествующей переменной, а содержимое скобок не отделяется от них.

    ```
    a[1][index]
    ```

    (b) оператор последовательного выполнения следует грамматическим правилам английского языка — перед запятой пробел не ставится, а после — ставится, перенос строки выполняется строго после запятой;

    (c) запятая в списке параметров также следуют грамматическим правилам английского языка;

    (d) операторы доступа к полям составных объектов (`.` и `->`), а также к пространству имен `::`, не выделяются пробелами.

18. Двоеточие в метках **case** и переходах является частью синтаксической конструкции и не отделяется пробелом слева. Сразу за ним идет новая строка.

19. Двоеточие и точка с запятой в цикле **for** не должна отделяться пробелом слева, но после них обязательно идет пробел или перенос строки. В случае большого условия лучше определить λ-функцию или статическую функцию, чем переносить элементы заголовка цикла, так как перенос сильно ухудшает читаемость.

20. Имена переменных и параметров функций должны начинаться со строчной буквы и следовать одному из правил:

    (a) все слова пишутся строчными буквами и разделяются символами подчеркивания;

    (b) все слова, кроме первого, начинаются с заглавных букв и ничем не разделяются (аббревиатуры более 3 букв имеет смысл писать с заглавной строчными).

    Оформление должно быть однообразным во всех работах.

21. Имена классов должны начинаться с большой буквы. Если имя состоит из нескольких слов, они идут подряд без символов подчеркивания, каждое слово начинается с большой буквы (CapsCase). Например, `Queue` или `RoundedRectangle`. Исключением из правил могут являться функторы, имена которых могут быть в нижнем регистре. В этом случае отдельные слова в имени разделяются подчеркиваниями.

22. Имена классов должны представлять собой имена существительные с определениями и дополнениями. Исключения: имена функторов должны иметь в основе глагол, так как представляют собой действие.

23. Структуры в терминах C (без конструкторов и деструкторов, а также без методов) допустимо называть в нижнем регистре и разделять слова подчеркиваниями. В этом случае имя должно иметь суффикс «_t».

24. Секции внутри класса должны быть упорядочены в порядке убывания интереса к ним со стороны читающего код: первой должна идти секция **public**, так как это интерфейс класса для клиентов; затем секция **protected**, поскольку она представляет собой интерфейс для наследников; завершает класс секция **private**, так как эта информация интересна только разработчику класса.

25. Внутри каждой секции информация должна быть упорядочена следующим образом:

    (a) типы;

    (b) поля;

    (c) конструктор по умолчанию;

    (d) конструкторы копирования и перемещения;

    (e) все остальные конструкторы;

    (f) деструктор;

    (g) перегруженные операторы;

    (h) методы.

26. Имена функций и методов должны начинаться с строчной буквы, дополнительные слова должны начинаться с заглавной буквы и идти вместе с предыдущим (camelCase), например, `draw()` или `getArea()`.

27. Имена функций должны начинаться с глаголов, так как они представляют собой действие.

28. Имена параметров шаблонов должны начинаться с заглавной буквы, так как являются либо типами, либо специфическими константами.

29. Все методы, не меняющие объект логически, должны быть отмечены квалификатором **const**.

30. Имена полей класса должны следовать правилам для переменных и иметь отличительный признак поля, в качестве которого следует использовать либо префикс «m_», либо завершающий символ подчеркивания (предпочтительнее).

31. Недопустимо создавать имена, начинающиеся с подчеркиваний.

32. Имена макроопределений должны быть в верхнем регистре.

33. Перенос длинных операторов на другую строку должен выполняться следующим образом:

    (a) в начале следующей строки должен находиться перенесенный оператор (за исключением запятой);

    (b) отступ должен быть больше на 2 единицы отступа;

    (c) в случае переноса внутри сложного выражения с круглыми скобками, отступ должен отражать вложенность скобок, каждый уровень вложения добавляет 1 отступ на следующий строке.

    Пример переноса (ширина строки не соблюдена):

```
if (condition) {
  a = a + b
      * (c
        + d / e);
}
```

    Исключение: в выражениях ввода и вывода с потоками допускается выравнивать операторы на перенесенных строка друг под другом, если это не создает слишком большого отступа:

```
std::cout << a << b
          << c << d;
```

34. Список инициализации в конструкторе должен инициализировать не более одной переменной или базового класса на строке. Двоеточие перед списком инициализации должно оставаться на строке со списком параметров, недопустимо переносить его на следующую строку. Оно не отделяется слева пробелом. Аналогично оформляются запятые, разделяющие элементы списка инициализации: запятая остается на строке с предыдущим элементом. Отступ в списке инициализации увеличивается на 1 по сравнению со строкой, где начинается определение конструктора (строки перенесены для демонстрации):

```
SampleClass::SampleClass(int a,
    int b):
  a_(a),
  b_(b)
{}
```

35. В списках инициализации массивов допускается применение «висящей» запятой после последнего элемента, так как это уменьшает «визуальный шум» в изменениях при увеличении списка инициализации.

36. Внутри выражений круглые скобки должны применяться для исключения неоднозначностей. За исключением очевидных из школьной арифметики приоритетов, все выражения оформляются так, чтобы при чтении не требовалось знать таблицу приоритетов операторов.

37. Не допускается применение «магических значений», вместо этого должны использоваться константы. Есть исключения: 0 и одноразово использованные строковые литералы.

38. Не допускается использование конструкций **`using namespace`** на верхнем уровне в заголовочных файлах.

39. Заголовочные файлы должны содержать header guards независимо от наличия поддержки какого-либо другого механизма предотвращения повторного включения, так как все эти механизмы нестандартны и непереносимы, за исключением **`#pragma`** `once`.

40. Порядок включения заголовочных файлов важен:

    (a) внутри файла реализации первым включается соответствующий ему заголовок;
    (b) стандартные заголовки;
    (c) заголовки использованных библиотек;
    (d) свои заголовки.

41. Зависимости от заголовков должны быть минимизированы. Не допускается включение заголовков с реализациями, если достаточно объявлений.

42. В файлах реализации все функции должны иметь полные имена, в которых пространства имен и имена классов отделены от имени функции при помощи `::`. Например, для функции в пространстве имен `lab::detail` с именем `foo()` реализация должна выглядеть так:

```
void lab::detail::foo()
{
  …
}
```

    Реализация функции в точке объявления допустима только для статических функций внутри единицы трансляции, шаблонных функций, методов шаблонных классов и однострочных методов, наподобие get-методов.

43. Форматирование λ-функций должно соответствовать оформлению обычных функций, с учетом того, что список захвата замещает название функции. Недопустимо писать их в одну строку.

```
[](const Object & obj) -> std::string {
  if (obj.getName().empty()) {
    return "<EMPTY>";
  } else {
    return obj.getName();
  }
}
```

44. Пустые строки должны использоваться разумно. Допускается использовать не более 1 пустой строки для:

    - разделения логических групп заголовков (п. 40);
    - отделения header guards;
    - разделения секций с различным доступом в классах (может быть вставлена *перед* ключевым словом, если оно не первое в определении класса;
    - разделения различных частей определения класса (п. 25);
    - отделения определений переменных от остального кода;
    - разделения определений классов, функций и т.п.;
    - отделения логически связанных фрагментов кода от других таких же фрагментов.

    Не следует отделять каждую переменную (или объявление функции и т.д.) пустой строкой, так как это излишне растягивает код.

    Для разделения классов и функций *не допускается* использование линеек из символов «минус» и «равно» для разделения блоков кода.

45. В программе не должно быть закомментированного кода. Весь неиспользуемый код должен быть удален. В случае необходимости, он может быть восстановлен из системы контроля версий.

Общий признак хорошего кода: его можно прочитать по телефону и он будет понятен на той стороне. Наиболее простым способом следовать этим правилам будет настроить редактор так, чтобы он выполнял такое форматирование автоматически.