Кодстайл для первокурсников ФПМИ МФТИ, версия 2022-23
автор документа: Мещерин Илья, вк/телеграм: mesyarik

Следование данному кодстайлу обязательно в задачах по C++ с код-ревью.

За основу мы берем кодстайл Гугла, а также кодстайл Яндекса, который в большинстве пунктов с ним совпадает.

Новые пункты еще могут добавляться, если выяснится, что мы что-то забыли.

****

1. Имена переменных с большой буквы - это плохо. Переменные, а также функции (не методы) нужно называть с маленькой буквы. Если название состоит из более чем одного слова, есть два способа: var_name либо varName. Можно выбрать любой из этих способов, но следует придерживаться одного и того же способа во всей программе.

2. Имена классов с маленькой буквы - это плохо (кроме особых случаев, о которых мы будем говорить еще нескоро). Классы и структуры нужно называть с большой буквы. Функции-методы классов можно называть с большой буквы, а можно и с маленькой, но необходимо придерживаться единого стиля во всей программе. (В большинстве случаев в ТЗ мы будем просить давать конкретные названия методам, поэтому такого вопроса у вас не встанет.)

3. Очень важный пункт хорошего кодстайла: по именам переменных и функций **должно быть без комментариев понятно, что это такое.**
    a. Если переменная или функция имеет тип bool, то ее именем должно быть либо прилагательное (например v.empty()), либо isSomething (например isPalindrome(str)). Имя должно отвечать на вопрос “какой?” или “является чем?”
    b. Если функция имеет тип void, то ее имя должно содержать глагол с ответом на вопрос, что эта функция делает. Например: push_back. Имя должно отвечать на вопрос “что сделать?”
    c. Если функция возвращает число или какой-то объект, то желательно, чтобы ее имя описывало возвращаемый объект (отвечало на вопрос

“что?”). Например: v.size(). В редких случаях, **в целях оптимизации эффективности**, если возвращаемый объект имеет сложную структуру, которую трудно описать одним именем, допускаются отступления от этого пункта, например map::insert возвращает пару из итератора и bool.

4. Однобуквенные имена переменных - это плохо. Нежелательно называть переменные одной буквой (исключения - счетчики в коротких циклах). Также нежелательно делать сокращения, например pos вместо position. Nedopustimo ispolsovat translit, naprimer vodka, babushka, balalaika. В идеале надо стремиться к тому, чтобы код читался как английский текст.

5. Другое замечание относительно объявлений переменных: их следует объявлять как можно ближе к месту первого использования. Так, если вам нужен счетчик для for, то его нужно объявить внутри for, а не заранее. Если вам нужна переменная tmp, например, для swap, то объявить ее надо прямо в тот момент, когда она нужна. Так гораздо легче: а) избежать конфликтов имен; б) понять, какая переменная для чего служит.

6. using namespace - это плохо. Запрещается использовать "using namespace". Таким образом вы делаете видимыми много имен, о которых даже не подозреваете, и в большом коде легко нарваться на конфликт имен (вы вызвали свою f, а в подключенном namespace была функция с таким же именем и по правилам перегрузки она оказалась более подходящей - в итоге вызвалась не та функция, а вы этого даже не заметили). Если нужно многократно использовать что-то из std, например vector, то можно написать using std::vector;

7. Табуляция в коде - это плохо. Запрещено использовать в коде горизонтальную табуляцию. Дело в том, что в разных системах, в разных редакторах размер табуляции и её внешний вид может быть принципиально разным - из-за этого ваш код начнет плясать, и понять, где какой отступ, будет невозможно (например, в случае со сложными отступами, см. следующий пункт). Поэтому всегда вместо одного таба ставьте 4 (или 2) пробела, но придерживайтесь единого стиля во всей программе. Все современные IDE (даже vim) умеют автоматически ставить пробелы вместо табов, когда вы нажимаете Tab.

8. Слишком длинные строки кода - это плохо. Рекомендуется делать строки не длиннее 100 символов. Если слишком длинный список аргументов функции или слишком длинное условие под ифом не помещается на одну строку, то его можно перенести на следующую строку **либо с двойным отступом, либо с отступом вплоть до открывающей скобки (,** так чтобы продолжение списка аргументов или условия не сливалось с дальнейшим кодом. Например:

```
if (x < 1 && y > 2 && a + b + c == 3 || x > 2 && y < 3
        || x > 5 && z < 1 && b - a == 1) {
    std::cout << 111;
}
Или:
if (x < 1 && y > 2 && a + b + c == 3 || x > 2 && y < 3
    || x > 5 && z < 1 && b - a == 1) {
```

```
  std::cout << 111;
}
```

9. (Спойлер: этот пункт, скорее всего, будет самой частой причиной, по которой ваш код будет отправляться на доработку)
   Копипаста в коде - это очень плохо!!! Если в вашем коде есть кусок длиной хотя бы в 3 строки, который встречается хотя бы дважды без изменений - значит, нужно придумать, как вынести его в отдельную функцию. Даже если в вашем коде повторяется один и тот же кусок с небольшими изменениями - все равно. Постарайтесь придумать, как выразить нужные действия через другие, которые вы раньше уже реализовывали.
   Наличие копипасты в промышленном коде - одно из самых плохих явлений. Для каждого нетривиального действия должно быть ровно одно место в программе, где это действие реализовано. Если это не так, то при необходимости что-то исправить в поведении программы вам будет недостаточно найти и исправить это место, вам придется искать все аналогичные места, неизвестно сколько их. Скорее всего, про одно из них вы забудете, и в итоге от ваших исправлений не будет толку.
   Как проверить свой код на соответствие этому пункту:
   1) посмотрите на свой код.
   2) Задайте себе вопрос: если бы вам нужно было его напечатать с чистого листа, захотелось ли бы вам хоть раз воспользоваться Ctrl+C Ctrl+V, чтобы не набирать дважды одно и то же?
   3) если да - поздравляю, у вас в коде копипаста. Исправляйте так, чтобы повода использовать Ctrl+C Ctrl+V при перепечатывании вашего кода не возникало.

10. Бинарные операторы следует окружать пробелами с двух сторон, кроме операторов “точка”, “запятая”, “стрелочка”, “двойное двоеточие”. После запятой ставится пробел, перед запятой не ставится.

11. **После) перед { ставится пробел.**
    Открывающая фигурная скобка { на отдельную строку не переносится.
    После слов if, for, switch, while перед круглой скобкой ставится пробел.
    Круглые скобки от их содержимого пробелами не отделяются.
    При функциональном вызове, как и при объявлении/определении функции, скобка (от названия функции пробелом не отделяется.
    Открывающая угловая скобка < от типа (например, vector<int>) пробелом не отделяется.
    Модификаторы типа * и & примыкают к типу, а не к названию переменной: int& a; не int &a.
    Пример:

```
std::vector<int> vec(2 * n, 0);
for (int i = 0, j = 0; i < n; ++i, j += 2) {
    if (i % 3 == 0)
       vec.at(i) = j;
}
```

12. Ифы и циклы без фигурных скобок вокруг тела - это плохо. Допускается делать ифы-однострочники без фигурных скобок, но к циклам такая поблажка не относится. Кроме того, если у вас после if стоят фигурные скобки и присутствует else, то секция else тоже должна быть с фигурными скобками. **После } перед словом else переход на новую строку не делается!**

13. Многоуровневые ифы - это плохо. Старайтесь не открывать новый уровень вложенности, когда это возможно.
    Наличие секции else - это редко нужная вещь. На практике в большинстве случаев иф может обходиться без else. Задумайтесь, действительно ли вам нужно сделать принципиально разные действия в случае выполнения условия и иначе? Или, может быть, часть действий вам надо сделать в любом случае, а другую часть - под ифом, но вы просто хотите скопипастить и написать дважды одни и те же по сути действия?

14. Приведения типов - это плохо. Старайтесь делать типы переменных такими, чтобы необходимость в кастах не возникала.
    Если все же нужен каст, то помните, что неявные касты - это плохо (кроме случаев тривиальных кастов между lvalue и rvalue, а также неконстанты в константу). Старайтесь явно прописывать, где происходит каст, и используйте static_cast. Если вам нужно привести указатели, то reinterpret_cast.
    Если вы собираетесь использовать reinterpret_cast к ссылке или const_cast в направлении от константы к неконстанте, задумайтесь: вы точно этого хотите? Вы никак не можете без этого? Если все еще считаете, что да, то спросите об этом в чате.
    Если вы собираетесь использовать C-style cast, вы точно неправы. Покайтесь.

15. После return круглые скобки не ставятся. Пишется return x, а не return (x). Казалось бы, какая разница, но в некоторых случаях наличие скобок после return существенно меняет поведение кода неожиданным образом. Если у вас не именно этот случай, в котором вы понимаете, что делаете - то скобки ставить не надо.

16. Старайтесь заменять ифы тернарными операторами, где это возможно, но знайте меру. Многоуровневый иф заменять тернарным оператором не стоит.
    А вот инструкции вида if (condition) x=a; else x=b; - обязательно стоит:
    Причем это делается **не так:**
    condition ? x = a : x = b;
    а так:
    x = (condition ? a : b);

17. Вместо x=x+c; следует писать x+=c; аналогично про другие составные присваивания.

18. Префиксный инкремент при прочих равных предпочтительнее, чем постфиксный, аналогично про декремент.

19. Помните, что на каждый вызов new в программе должен быть, и притом единственный, вызов delete. Не путайте delete и delete[].
    Помните, что вызов оператора new - это дорогостоящая операция, потому что она может приводить к запросу у операционной системы дополнительных ресурсов, что приводит к приостановке вашей программы на некоторое время. Поэтому, когда пользуетесь динамической памятью, старайтесь минимизировать количество вызовов new в своей программе.

20. Используйте функции memcpy и memset, либо же std::fill и std::copy для заполнения памяти какими-либо значениями. Не делайте это вручную циклом: это будет работать в разы медленнее.

***

ООП

21. Используйте struct, если в вашем случае более удобно сделать поля публичными. Если же подразумевается, что поля извне свободно менять нельзя, то пишите class. В классах делайте все по максимуму приватным: методы, которые не предназначены для внешнего использования, не должны торчать наружу.

22. Не надо писать this, если это явно не необходимо! Если вы находитесь в методе класса, то к полям и методам текущего объекта можно обращаться напрямую, без this!

23. Друзья - это плохо, они нарушают принцип инкапсуляции. Используйте слово friend только в крайнем случае! И уж точно не надо делать все подряд бинарные операторы friend'ами. Максимум что имеет смысл сделать - friend оператор ввода из потока (хотя и без этого иногда можно обойтись).

24. Обязательно **используйте списки инициализации в конструкторах**, вместо того чтобы делать присваивания значений полей в теле конструктора. При этом обязательно соблюдайте порядок инициализации полей: нужно перечислять их в том порядке, в котором они объявлены в классе!

25. **The Rule Of Three:** если вашему классу понадобился явный конструктор копирования, или оператор присваивания, или деструктор, значит, ему нужны все три эти вещи.

26. Помните об идиоме copy-and-swap, когда реализовываете оператор присваивания. Это позволит избавиться от копипасты.

27. Помните, что от порядка перечисления полей в классе, вообще говоря, зависит размер класса. При прочих равных старайтесь выбирать такой порядок полей в классе, при котором "пустот" в памяти между ними не возникает.

28. Помните, что невозможно явно вызвать конструктор для данного объекта, когда вы уже находитесь в теле метода этого объекта! Если вам нужно вызвать какой-то конструктор до того, как войти в другой конструктор, используйте делегирующие конструкторы.

29. Если метод по смыслу должно быть можно вызывать от константных объектов, обязательно помечайте его словом const. При этом, если от константных и от неконстантных объектов метод должен работать по-разному, делайте перегрузку.

30. При перегрузке операторов + нужно выражать через +=, а не наоборот! Аналогично про другие арифметические операторы. Кроме того, бинарный оператор + не нужно делать членом класса: иначе его нельзя было бы вызывать от левого операнда - не объекта класса.

31. Когда определяете операторы сравнения, помните, что все сравнения принято выражать через <. Кстати, сравнение “a > b” выражается так: “b < a”, а не так “a >= b && a != b”.

32. Когда определяете операторы == и !=, помните, что сперва надо сделать простые проверки, а только потом те, которые могут занять линейное время. Например, сначала надо проверить, совпадает ли длина строк, а уже потом проверять посимвольно.

33. Помните о Return Value Optimization. Если вы создаете внутри метода локальный объект и его же возвращаете наружу (по значению), старайтесь писать код метода так, чтобы компилятору было очевидно, какой объект вы возвращаете - тогда он сможет сделать RVO.

34. Если какой-то метод вашего класса по смыслу не должен вызываться от объекта данного класса (а просто делает какие-то вспомогательные вычисления), то делайте этот метод статическим. Аналогично, делайте статическими те поля, которые по смыслу не относятся к конкретному объекту, а должны быть общими для всего класса.

35. Если у вашего класса есть конструктор от одного параметра, подумайте, не стоит ли сделать его explicit (скорее всего, стоит). И уж тем более стоит делать explicit операторы приведения к другим типам (включая bool).

36. Используйте публичное наследование, если вы хотите тем самым сказать читателю “этот класс является частным случаем другого”. Приватное наследование используется тогда, когда в реализации своего класса вы хотите использовать детали реализации другого класса, но внешним пользователям вашего класса знать об этом не нужно.

37. Обязательно ставьте слово override, когда в наследниках переопределяете виртуальные методы родителей, а если при этом предполагается, что никто от вас наследоваться больше не должен, пишите final. Из слов virtual, override и

final достаточно только одного слова. Писать одновременно, например, override final не надо, достаточно только final.

38. Если в вашем классе есть хотя бы одна виртуальная функция, обязательно сделайте виртуальный деструктор.