**Руководство Google по стилю в C++. Часть 8**

[C++ \*](https://habr.com/ru/hub/cpp/)

Из песочницы

[Часть 1. Вступление](https://habr.com/ru/post/480422/)  
…  
[Часть 7. Ещё возможности C++](https://habr.com/ru/post/586550/)  
**Часть 8. Именование**  
[Часть 9. Комментарии](https://habr.com/ru/post/480424/)  
…  
  
  
  
Все мы при написании кода пользуемся правилами оформления кода. Иногда изобретаются свои правила, в других случаях используются готовые стайлгайды. Хотя все C++ программисты читают на английском легче, чем на родном, приятнее иметь руководство на последнем.  
Эта статья является переводом части руководства Google по стилю в C++ на русский язык.  
[Исходная статья](https://github.com/evgenykislov/styleguide/blob/gh-pages/cppguide.html) (fork на github), [обновляемый перевод](https://evgenykislov.com/wp-content/custom/cpp_codestyle/cppguide_ru.html).

**Именование**

Основные правила стиля кодирования приходятся на именование. Вид имени сразу же (без поиска объявления) говорит нам что это: тип, переменная, функция, константа, макрос и т.д. Правила именования могут быть произвольными, однако важна их согласованность, и правилам нужно следовать.

**Общие принципы именования**

* Используйте имена, который будут понятны даже людям из другой команды.
* Имя должно говорить о цели или применимости объекта.
* Не экономьте на длине имени, лучше более длинное и более понятное (даже новичкам) имя.
* Поменьше аббревиатур, особенно если они незнакомы вне проекта.
* Используйте только известные аббревиатуры (Википедия о них знает?).
* Не сокращайте слова.

В целом, длина имени должна соответствовать размеру области видимости. Например, **n** — подходящее имя внутри функции в 5 строк, однако при описании класса это может быть коротковато.

class MyClass {

public:

int CountFooErrors(const std::vector<Foo>& foos) {

int n = 0; // Чёткий смысл для небольшой области видимости

for (const auto& foo : foos) {

...

++n;

}

return n;

}

void DoSomethingImportant() {

std::string fqdn = ...; // Известная аббревиатура полного доменного имени

}

private:

const int kMaxAllowedConnections = ...; // Чёткий смысл для контекста

};

class MyClass {

public:

int CountFooErrors(const std::vector<Foo>& foos) {

int total\_number\_of\_foo\_errors = 0; // Слишком подробное имя для короткой функции

for (int foo\_index = 0; foo\_index < foos.size(); ++foo\_index) { // Лучше использовать `i`

...

++total\_number\_of\_foo\_errors;

}

return total\_number\_of\_foo\_errors;

}

void DoSomethingImportant() {

int cstmr\_id = ...; // Сокращённое слово (удалены буквы)

}

private:

const int kNum = ...; // Для целого класса очень нечёткое имя

};

Отметим, что типовые имена также допустимы: **i** для итератора или счётчика, **T** для параметра шаблона.  
  
В дальнейшем при описании правил «word» / «слово» это всё, что пишется на английском без пробелов, в том числе и аббревиатуры. В слове первая буква может быть заглавной (зависит от стиля: "[camel case](https://en.wikipedia.org/wiki/Camel_case)" или «Pascal case»), остальные буквы — строчные. Например, предпочтительно **StartRpc()**, нежелательно **StartRPC()**.  
  
Параметры шаблона также следуют правилам своих категорий: Имена типов, Имена переменных и т.д…

**Имена файлов**

Имена файлов должны быть записаны только строчными буквами, для разделения можно использовать подчёркивание (**\_**) или дефис (**-**). Используйте тот разделитель, который используется в проекте. Если единого подхода нет — используйте "\_".  
  
Примеры подходящих имён:

* **my\_useful\_class.cc**
* **my-useful-class.cc**
* **myusefulclass.cc**
* **myusefulclass\_test.cc // \_unittest and \_regtest are deprecated.**

C++ файлы должны заканчиваться на **.cc**, заголовочные — на  
**.h**. Файлы, включаемые как текст должны заканчиваться на **.inc** (см. также секцию [Независимые заголовочники](https://evgenykislov.com/wp-content/custom/cpp_codestyle/cppguide_ru.html#Self_contained_Headers)).  
  
Не используйте имена, уже существующие в **/usr/include**, такие как **db.h**.  
  
Старайтесь давать файлам специфичные имена. Например, **http\_server\_logs.h** лучше чем **logs.h**. Когда файлы используются парами, лучше давать им одинаковые имена. Например, **foo\_bar.h** и **foo\_bar.cc** (и содержат класс **FooBar**).

**Имена типов**

Имена типов начинаются с прописной буквы, каждое новое слово также начинается с прописной буквы. Подчёркивания не используются: **MyExcitingClass**, **MyExcitingEnum**.  
  
Имена всех типов — классов, структур, псевдонимов, перечислений, параметров шаблонов — именуются в одинаковом стиле. Имена типов начинаются с прописной буквы, каждое новое слово также начинается с прописной буквы. Подчёркивания не используются. Например:

// classes and structs

class UrlTable { ...

class UrlTableTester { ...

struct UrlTableProperties { ...

// typedefs

typedef hash\_map<UrlTableProperties \*, std::string> PropertiesMap;

// using aliases

using PropertiesMap = hash\_map<UrlTableProperties \*, std::string>;

// enums

enum UrlTableErrors { ...

**Имена переменных**

Имена переменных (включая параметры функций) и членов данных пишутся строчными буквами с подчёркиванием между словами. Члены данных классов (не структур) дополняются подчёркиванием в конце имени. Например: **a\_local\_variable**, **a\_struct\_data\_member**, **a\_class\_data\_member\_**.

**Имена обычных переменных**

Например:

std::string table\_name; // OK - строчные буквы с подчёркиванием

std::string tableName; // Плохо - смешанный стиль

**Члены данных класса**

Члены данных классов, статические и нестатические, именуются как обычные переменные с добавлением подчёркивания в конце.

class TableInfo {

...

private:

std::string table\_name\_; // OK - подчёркивание в конце

static Pool<TableInfo>\* pool\_; // OK.

};

**Члены данных структуры**

Члены данных структуры, статические и нестатические, именуются как обычные переменные. К ним не добавляется символ подчёркивания в конце.

struct UrlTableProperties {

std::string name;

int num\_entries;

static Pool<UrlTableProperties>\* pool;

};

См. также [Структуры vs Классы](https://evgenykislov.com/wp-content/custom/cpp_codestyle/cppguide_ru.html#Structs_vs._Classes), где описано когда использовать структуры, когда классы.

**Имена констант**

Объекты объявляются как constexpr или const, чтобы значение не менялось в процессе выполнения. Имена констант начинаются с символа «k», далее идёт имя в смешанном стиле (прописные и строчные буквы). Подчёркивание может быть использовано в редких случаях когда прописные буквы не могут использоваться для разделения. Например:

const int kDaysInAWeek = 7;

const int kAndroid8\_0\_0 = 24; // Android 8.0.0

Все аналогичные константные объекты со статическим типом хранилища (т.е. статические или глобальные, подробнее тут: [Storage Duration](http://en.cppreference.com/w/cpp/language/storage_duration#Storage_duration)) именуются также. Это соглашение является необязательным для переменных в других типах хранилища (например, автоматические константные объекты).

**Имена функций**

Обычные функции именуются в смешанном стиле (прописные и строчные буквы); функции доступа к переменным (accessor и mutator) должны иметь стиль, похожий на целевую переменную.  
  
Обычно имя функции начинается с прописной буквы и каждое слово в имени пишется с прописной буквы.

void AddTableEntry();

void DeleteUrl();

void OpenFileOrDie();

(Аналогичные правила применяются для констант в области класса или пространства имён (namespace) которые представляют собой часть API и должны выглядеть как функции (и то, что они не функции — некритично))  
  
Accessor-ы и mutator-ы (функции get и set) могут именоваться наподобие соответствующих переменных. Они часто соответствуют реальным переменным-членам, однако это не обязательно. Например, **int count()** и **void set\_count(int count)**.

**Именование пространства имён (namespace)**

Пространство имён называется строчными буквами. Пространство имён верхнего уровня основывается на имени проекта. Избегайте коллизий ваших имён и других, хорошо известных, пространств имён.  
  
Пространство имён верхнего уровня — это обычно название проекта или команды (которая делала код). Код должен располагаться в директории (или поддиректории) с именем, соответствующим пространству имён.  
  
Не забывайте правило **не использовать аббревиатуры** — к пространствам имён это также применимо. Коду внутри вряд ли потребуется упоминание пространства имён, поэтому аббревиатуры — это лишнее.  
  
Избегайте использовать для вложенных пространств имён известные названия. Коллизии между именами могут привести к сюрпризам при сборке. В частности, не создавайте вложенных пространств имён с именем **std**. Рекомендуются уникальные идентификаторы проекта (**websearch::index**, **websearch::index\_util**) вместо небезопасных к коллизиям **websearch::util**.  
  
Для **internal / внутренних** пространств имён коллизии могут возникать при добавлении другого кода (внутренние хелперы имеют свойство повторяться у разных команд). В этом случае хорошо помогает использование имени файла для именования пространства имён. (**websearch::index::frobber\_internal** для использования в **frobber.h**)

**Имена перечислений**

Перечисления (как с ограничениями на область видимости (scoped), так и без (unscoped)) должны именоваться *либо* как **константы**, либо как **макросы**. Т.е.: либо **kEnumName**, либо **ENUM\_NAME**.  
  
Предпочтительно именовать отдельные значения в перечислителе как константы. Однако, допустимо именовать как макросы. Имя самого перечисления **UrlTableErrors** (и **AlternateUrlTableErrors**), это тип. Следовательно, используется смешанный стиль.

enum UrlTableErrors {

kOk = 0,

kErrorOutOfMemory,

kErrorMalformedInput,

};

enum AlternateUrlTableErrors {

OK = 0,

OUT\_OF\_MEMORY = 1,

MALFORMED\_INPUT = 2,

};

Вплоть до января 2009 года стиль именования значений перечисления был как у макросов. Это создавало проблемы дублирования имён макросов и значений перечислений. Применение стиля констант решает проблему и в новом коде предпочтительно использовать стиль констант. Однако, старый код нет необходимости переписывать (пока нет проблем дублирования).