Apoheliy 27 ноября 2019 в 14:33

Руководство Google по стилю в C++. Часть 8

C++ *

Из песочницы

Часть 1. Вступление

…

Часть 7. Ещё возможности C++

Часть 8. Именование

Часть 9. Комментарии

…

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 (fork на github), обновляемый перевод.

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 функц
    for (int foo_index = 0; foo_index < foos.size(); ++foo_index) {  // Лучше исполь
      ...
      ++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" 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).

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, 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) 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).

Имена макросов

Вы ведь не собираетесь определять макросы? На всякий случай (если собираетесь), они должны выглядеть так:
MY_MACRO_THAT_SCARES_SMALL_CHILDREN_AND_ADULTS_ALIKE.

Пожалуйста прочтите как определять макросы; Обычно, макросы *не* должны использоваться. Однако, если они вам абсолютно необходимы, именуйте их прописными буквами с символами подчёркивания.

```
#define ROUND(x) ...
#define PI_ROUNDED 3.0
```

Исключения из правил именования

Если вам нужно именовать что-то, имеющее аналоги в существующем C или C++ коде, то следуйте используемому в коде стилю.

bigopen()
имя функции, образованное от **open()**

uint
определение, похожее на стандартные типы

bigpos
struct или **class**, образованный от **pos**

sparse_hash_map
STL-подобная сущность; следуйте стилю STL

LONGLONG_MAX
константа, такая же как **INT_MAX**

Прим.: ссылки могут вести на ещё не переведённые разделы руководства.

Теги: C++, styleguide, перевод с английского

Хабы: C++

+10 25K 108

7 Карма 0 Рейтинг
@Apoheliy
Пользователь

Комментарии 21

SannX
27.11.2019 в 15:06
Еще один перевод студента, чтобы сдать зачет?
+7 Ответить

Exosphere
27.11.2019 в 15:14
Нет. Теперь все переводы будут в этом подозревать? :-) На самом деле, вроде один остался.
0 Ответить

SannX
27.11.2019 в 15:18
Ну, если препод и дальше так будет работать со студентами, то конца не будет гугл-транслейт-статьям. В общем, успехов Вам.
0 Ответить

Exosphere
27.11.2019 в 15:58
Спасибо, мы бдим =) Думаю, в следующий сет мы уже поищем выход из ситуации.
0 Ответить

Apoheliy
27.11.2019 в 16:41

Еще один перевод (— да) студента (— нет), чтобы сдать зачет (— нет)?

Попадание 1 из 3-х.
Не Нострадамус :).

0 Ответить

maaGames
27.11.2019 в 18:32

По какой причине пошла «мода» ставить константам префикс «k»? Я бы понял «c», но почему «k»?

+2 Ответить

KanuTaH
27.11.2019 в 19:47

Это идёт от так называемой венгерской нотации. В классической версии этой нотации префикс "c" использовался для char, а "k" — для констант.

0 Ответить

maaGames
28.11.2019 в 15:45

Т.е. из-за того, что «c» уже было занято, взяли любую другую какую-то букву просто так, без великого аббревиатурного смысла?

0 Ответить

KanuTaH
28.11.2019 в 15:52

Ну может потому, что «k» сама по себе произносится как «c» в слове «constant» :) А, может, потому, что Чарльз Симони был венгерского происхождения, а по-венгерски «константа» — это «konstans».

+1 Ответить

maaGames
28.11.2019 в 18:15

Во, вот такой вариант меня устраивает. Звёзды сошлись, я доволен:)

0 Ответить

NightShad0w
27.11.2019 в 21:57

За работу над переводом спасибо. По сути статьи — рекомендации не содержат пояснения и причин рекомендации, а значит вслепую применять затруднительно, не понимая, в той же ситуации ты или нет.
Для себя определился со стилем — все нижним регистром кроме типов как аргументов шаблонов. Нет разделения на константы или переменные. Все поля объектов — без дополнительных декораций. Имена типов с суффиксом _t, шаблонные типы — с суффиксом _tt (от template type). Суффиксы помогают ориентировать код на строгую типизацию и облегчают тестирование.

```
...
class foo_tt{};
...
// production
using foo_t = foo_tt<bar_t>;
void run(foo_t foo);
....
// tests
using foo_t = foo_tt<mock::bar_t>
...
```

Такой подход во-первых более соответствует публичному интерфейсу стандартной библиотеки, во-вторых — помогает фокусироваться на алгоритме и функционале вместо отвлечения на сущность имени — поле, аргумент, константа. Алгоритму должно быть все равно, константность обеспечит ключевое слово в конст-корректном коде, поле — внутреннее состояние, которое не отличимо по природе от переменной для чистой реализации.
Венгерская нотация недостаточна и одновременно избыточна для современного кода с шаблонами, лямбдами и прочими сущностями, невыразимымы через вменяемое количество суффиксов и префиксов.

+1 Ответить

Apoheliy
27.11.2019 в 23:22

Пожалуйста.

Пояснений и причин не будет — это тема большого холивара. Сомнительно, что будет расписываться руководство с приглашением поспорить.

Для собственного стиля: вам своих правил хватает? Или хочется там добавить, потом ещё …?
Гугловое руководство (полное, а не только именование) тем и хорошо, что сразу обо всём. И

когда свои правила становятся тесноваты — Гугл уже идёт к вам.

0 Ответить

CJay
16.12.2019 в 21:20

Не хотелось бы работать с кодом, в котором типы отличаются от переменных лишь наличием смайлика со шрамом o_t.
Почему вы игнорируете такой важный элемент выделения символов как регистр?

0 Ответить

sonntex
28.11.2019 в 10:02

Хорошо конечно, что у Google все подумано до мелочей, но напоминает это в какой-то степени Венгерскую нотацию, широко используемую в WinAPI (повсюду заклавные буквы и префиксы (в меньшей мере)). Продолжительное время разрабатываю в Linux, но те боль и мучения, связанные с разработкой в институте и в начале карьеры, еще отчетливо лежат в памяти.

Предполагаю, что нотация должа исходить от того, в каком фреймворке вы программируете или какие библиотеки используете. Если вы пишете в Qt — надо придерживаться стиля написания Qt. Если Modern C++ и Boost — Undescope. В последнем случае Google C++ Code Style выглядит совсем инородным.

Я бы давал вольности при разработке, ограничивая лишь базовые вещи: количество пробелов, наличие табов, расположение открывающейся и закрывающейся скобок. Все остальное от контекста. Если код на C — указатель пишем слитно с переменной, если на C++ — наоборот. Ну и так далее… А там настроить .clang-format под конкретный проект — занятие на 1-2 часа.

+1 Ответить

savostin
28.11.2019 в 10:57

Что-то какой-то разброд и шатание, никакой логики. Я понимаю, конечно, что большинство «правил» — результат многолетней традиции, но всё же…

0 Ответить

Apoheliy
28.11.2019 в 14:29

По поводу логики в правилах, использования других стилей и общих принципов:
В начале статьи есть ссылка на обновляемый перевод и в нём уже переведено вступление с целями документа и другими рассуждениями общего характера.

Для отдельной статьи это маловато/скучновато, поэтому можно смотреть там.

0 Ответить

izvolov
06.12.2019 в 16:25

Девочка с фотографии, похоже, слегка прифигела от прочитанного. Потому что даже она знает, что есть CppCoreGuidelines.

0 Ответить

Apoheliy
06.12.2019 в 17:54

Есть два возражения на Ваш комментарий:
1. гайдов по стилю найти можно много. Какой выбрать — решает команда. Некоторые команды вообще пишут свой гайд. Если использовать «внешний» гайд, то желательно оценить его дополнительную полезность. Например, Google настоятельно рекомендует использовать свой гайд (логично, правда?) для проектов, интегрируемых с Гуглом. С другой стороны, Страуструп и Сеттер люди известные. С третьей, CppCoreGuidelines вроде бы никто стандартом не объявлял. Так что: на вкус и цвет все фломастеры разные…
2. чисто практический вопрос: Izvolov, так что там с переводом CppCoreGuidelines? Может займётесь?

0 Ответить

izvolov
07.12.2019 в 01:37

Гугловские рекомендации тоже никто стандартом не объявлял. И не объявит. Чего только стоит идиотская рекомендация передавать выходные параметры функции по указателю.

Что касается перевода — не переживайте. Я свой посильный вклад в сообщество делаю. И вам предлагаю не тратить время на гугловское дерьмище, и если хочется заняться переводом, то переводить core guidelines.
А начать всегда можно с этого: https://translate.yandex.ru/translate?url=https%3A%2F%2Fgithub.com%2Fisocpp%2FCppCoreGuidelines%2Fblob%2Fmaster%2FCppCoreGuidelines.md&lang=en-ru

0 Ответить

Explorus
16.12.2019 в 23:38

А кто-нибудь может пояснить, почему в классах члены заканчиваются подчеркиванием, а в структурах вдруг нельзя?
Мне лично привычнее добавлять впереди «m_», и сразу понятно, что у нас тут используется,

простите, член.

0 Ответить

Apoheliy
16.12.2019 в 23:51

Понятно, что Гугл об этом не дискутирует, но…

Раньше был упор на тип данных. Отсюда венгерская нотация и m_. Причём m_ — половинчатый костыль (обычно к функциям не применяют).
Сейчас есть мощные IDE и пошла мода на «как применять» данные, функции и др.
Также стараются явно уйти от венгерской нотации.
Поэтому функции именуют одним образом, а члены данных — другим образом.
Отличие классов от структур связано, подозреваю, с историческим применением:
структуры — это «почти» POD и все поля в открытом доступе (понятно, что можно запретить. Но обычно — так)
классы — в них члены данных закрыты (типа так правильно), а для доступа есть функции.
Как результат — члены данных классов явно отличают от полей структур и функций.
Почему Гугл решил разделить именование функции классов и поля структур? Есть стайлгайды, где они именуются одинаково. Ну а у Гугла свои исторические предпосылки…

0 Ответить

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

31 октября в 16:43

Руководство Google по стилю в C++. Часть 7

+2 3.5K 32 0

25 августа 2020 в 01:45

Руководство Google по стилю в C++. Часть 2

+7 15K 73 21 +21

16 декабря 2019 в 01:14

Руководство Google по стилю в C++. Часть 1

+5 24K 239 0

+80 19K 170 34 +34

вчера в 11:00

Объяснение фильтра Калмана в картинках

+66 7.2K 139 8 +8

вчера в 13:23

Самоделка — полная...: ПХМ-1 из пластика и кремния

+51 8.8K 13 13 +13

вчера в 16:47

Как спасти миллионы жизней

+45 4.4K 37 9 +9

вчера в 14:01

Псориаз: что бывает при слишком быстрой регенерации

+45 8.8K 45 16 +16

ЧИТАЮТ СЕЙЧАС

Как вести переговоры с террористами... партнерами, детьми и вообще с кем угодно

19K 34 +34

Как Илон Маск уволил своего ассистента после просьбы о прибавке

78K 160 +160

Эксперимент Базермана: как мы ежедневно теряем деньги

119K 227 +227

Куда катятся зарплаты разработчиков в IT?

71K 213 +213

Умелец собрал огромный таймер 555 из дискретных компонентов

6.9K 23 +23

Wi-Fi в большом городе: телемедицина, футбол и умные светофоры

Мегапост

РАБОТА