

Оглавление

Избранные рекомендации по C++	14
Предисловие	17
Вступление	18
О книге.....	20
Код примеров.....	23
Благодарности	24
Об авторах	26
От издательства	28

ЧАСТЬ I BIKESHEDDING — ЭТО ПЛОХО

Глава 1.1. Р.2. Придерживайтесь стандарта ISO C++	30
Что такое стандарт ISO C++	30
История C++	30
Инкапсуляция вариаций	32
Вариации в окружении времени выполнения	32
Вариации на уровне языка C++ и компилятора	33
Расширения для C++	34
Защита заголовочных файлов	35
Вариации в основных типах	35
Нормативные ограничения	36
Изучение старых способов	37
Обратная совместимость в C++	37
Прямая совместимость и Y2K	38
Следите за последними изменениями в стандарте	39
IsoCpp	39
Конференции	40
Другие источники	40

Глава 1.2. F.51. Если есть выбор, используйте аргументы по умолчанию вместо перегрузки.....	42
Введение.....	42
Доработка ваших абстракций: дополнительные аргументы или перегрузка?	43
Тонкости разрешения перегрузки	45
Вернемся к примеру	47
Однозначная природа аргументов по умолчанию.....	49
Альтернативы перегрузке.....	50
Иногда без перегрузки не обойтись.....	51
Подведем итог	52
Глава 1.3. C.45. Не определяйте конструктор по умолчанию, который просто инициализирует переменные-члены; для этой цели лучше использовать внутриклассовые инициализаторы членов	53
Зачем нужны конструкторы по умолчанию	53
Как инициализируются переменные-члены.....	55
Что может случиться, если поддерживать класс будут два человека	58
Сборная солянка из конструкторов	58
Аргументы по умолчанию могут запутать ситуацию в перегруженных функциях.....	60
Подведем итог	60
Глава 1.4. C.131. Избегайте тривиальных геттеров и сеттеров	62
Архаичная идиома	62
Абстракции.....	63
Простая инкапсуляция.....	66
Инварианты класса.....	69
Существительные и глаголы.....	71
Подведем итог	72
Глава 1.5. E5.10. Объявляйте имена по одному в каждом объявлении	73
Позвольте представить.....	73
Обратная совместимость	76
Пишите более ясные объявления.....	77
Структурное связывание	78
Подведем итог	79

Глава 1.6. NR.2. Функции не обязательно должны иметь только один оператор возврата.....	80
Правила меняются	80
Гарантия очистки	83
Идиома RAII	85
Пишите хорошие функции	88
Подведем итог	90

ЧАСТЬ II НЕ НАВРЕДИТЕ СЕБЕ

Глава 2.1. P.11. Инкапсулируйте беспорядочные конструкции, а не разбрасывайте их по всему коду.....	92
Все одним глотком	92
Что означает инкапсулировать запутанную конструкцию	94
Назначение языка и природа абстракции.....	96
Уровни абстракции.....	100
Абстракция путем рефакторинга и проведения линии	101
Подведем итог	102
Глава 2.2. I.23. Минимизируйте число параметров в функциях	103
Сколько они должны получать?	103
Упрощение через абстрагирование	105
Делайте так мало, как возможно, но не меньше	107
Примеры из реальной жизни	109
Подведем итог	111
Глава 2.3. I.26. Если нужен кросс-компилируемый ABI, используйте подмножество в стиле C	112
Создавайте библиотеки.....	112
Что такое ABI.....	114
Сокращайте до абсолютного минимума.....	115
Распространение исключений.....	118
Подведем итог	119
Глава 2.4. C.47. Определяйте и инициализируйте переменные-члены в порядке их объявления.....	121
Подведем итог	131

Глава 2.5. CP.3. Сведите к минимуму явное совместное использование записываемых данных 132

Традиционная модель выполнения.....132

Подождите, это еще не все.....134

Предотвращение взаимоблокировок и гонок за данными137

Отказ от блокировок и мьютексов140

Подведем итог143

Глава 2.6. T.120. Используйте метапрограммирование шаблонов, только когда это действительно необходимо 144

std::enable_if => requires.....152

Подведем итог156

ЧАСТЬ III
ПРЕКРАТИТЕ ЭТО ИСПОЛЬЗОВАТЬ

Глава 3.1. I.11. Никогда не передавайте владение через простой указатель (T*) или ссылку (T&)..... 158

Использование области свободной памяти158

Производительность интеллектуальных указателей161

Использование простой семантики ссылок163

gsl::owner164

Подведем итог167

Глава 3.2. I.3. Избегайте синглтонов..... 168

Глобальные объекты — это плохо.....168

Шаблон проектирования «Синглтон»169

Фиаско порядка статической инициализации170

Как скрыть синглтон.....173

Только один из них должен существовать в каждый момент работы кода174

Подождите минутку.....176

Подведем итог179

Глава 3.3. C.90. Полагайтесь на конструкторы и операторы присваивания вместо memset и memсru 180

В погоне за максимальной производительностью.....180

Ужасные накладные расходы конструкторов.....181

Самый простой класс.....183

О чем говорит стандарт.....	185
А как же метасру?	188
Никогда не позволяйте себе недооценивать компилятор.....	189
Подведем итог	191
Глава 3.4. ES.50. Не приводите переменные с квалификатором const к неконстантному типу	192
Работа с большим количеством данных.....	193
Брандмауэр const	195
Реализация двойного интерфейса	196
Кэширование и отложенные вычисления	198
Два вида const.....	199
Сюрпризы const	201
Подведем итог	202
Глава 3.5. E.28. При обработке ошибок избегайте глобальных состояний (например, errno).....	204
Обрабатывать ошибки сложно	204
Язык C и errno.....	204
Коды возврата.....	206
Исключения	207
<system_error>.....	208
Boost.Outcome	209
Почему обрабатывать ошибки так сложно	210
Свет в конце туннеля	212
Подведем итог	214
Глава 3.6. SF.7. Не используйте using namespace в глобальной области видимости в заголовочном файле	215
Не делайте этого	215
Неоднозначность	216
Использование using.....	217
Куда попадают символы	219
Еще более коварная проблема.....	222
Решение проблемы операторов разрешения области видимости	223
Искушение и расплата	225
Подведем итог	226

ЧАСТЬ IV ИСПОЛЬЗУЙТЕ НОВУЮ ОСОБЕННОСТЬ ПРАВИЛЬНО

Глава 4.1.	F.21. Для возврата нескольких выходных значений используйте структуры или кортежи.....	228
	Форма сигнатуры функции	228
	Документирование и аннотирование	230
	Теперь можно вернуть объект	231
	Можно также вернуть кортеж	234
	Передача и возврат по неконстантной ссылке	237
	Подведем итог	240
Глава 4.2.	Enum.3. Старайтесь использовать классы-перечисления вместо простых перечислений.....	241
	Константы.....	241
	Перечисления с заданной областью видимости.....	244
	Базовый тип	246
	Неявное преобразование	247
	Подведем итог	249
Глава 4.3.	ES.5. Минимизируйте области видимости.....	250
	Природа области видимости	250
	Область видимости блока	251
	Область видимости пространства имен.....	253
	Область видимости класса.....	256
	Область видимости параметров функции.....	258
	Область видимости перечисления	259
	Область действия параметра шаблона.....	260
	Область видимости как контекст	261
	Подведем итог	262
Глава 4.4.	Con.5. Используйте <code>constexpr</code> для определения значений, которые можно вычислить на этапе компиляции	263
	От <code>const</code> к <code>constexpr</code>	263
	C++ по умолчанию.....	265
	Использование <code>constexpr</code>	267
	<code>inline</code>	271
	<code>constexpr</code>	272

constinit	273
Подведем итог	275
Глава 4.5. Т.1. Используйте шаблоны для повышения уровня абстрактности кода	276
Повышение уровня абстракции	278
Шаблоны функций и абстракция	280
Шаблоны классов и абстракция	283
Выбор имени — сложная задача	285
Подведем итог	286
Глава 4.6. Т.10. Задавайте концепции для всех аргументов шаблона	287
Как мы здесь оказались?	287
Ограничение параметров	290
Как абстрагировать свои концепции	293
Разложение на составляющие через концепции	296
Подведем итог	297

ЧАСТЬ V ПИШИТЕ ХОРОШИЙ КОД ПО УМОЛЧАНИЮ

Глава 5.1. P.4. В идеале программа должна быть статически типобезопасной	300
Безопасность типов — это средство защиты в C++	300
Объединения	302
Приведение	304
Целые без знака	307
Буферы и размеры	310
Подведем итог	311
Глава 5.2. P.10. Неизменяемые данные предпочтительнее изменяемых	312
Неправильные значения по умолчанию	312
const в объявлениях функций	315
Подведем итог	319
Глава 5.3. I.30. Инкапсулируйте нарушения правил	320
Скрытие неприглядных вещей	320
Поддержание видимости, что все в порядке	322
Подведем итог	327

Глава 5.4. ES.22. Не объявляйте переменные, пока не получите значения для их инициализации	329
Важность выражений и операторов	329
Объявление в стиле C.....	330
Объявление с последующей инициализацией	332
Максимальное откладывание объявления.....	333
Локализация контекстно зависимой функциональности	335
Устранение состояния.....	337
Подведем итог	339
Глава 5.5. Per.7. При проектировании учитывайте возможность последующей оптимизации	340
Максимальная частота кадров.....	340
Работа вдалеке от железа	342
Оптимизация через абстракцию.....	346
Подведем итог	349
Глава 5.6. E.6. Используйте идиому RAII для предотвращения утечек памяти.....	350
Детерминированное уничтожение.....	350
Утечка файлов	353
Почему это так важно	356
Все это выглядит чересчур сложным: будущие возможности	358
Где все это получить.....	361
Заключение	364
Послесловие.....	366

*Брину,
Шинейд,
Рори и Лоис.*

С.47: с любовью, Гай Дэвидсон

*Джиму Эллисону, хотя он едва ли прочитает эти строки.
Весь в исследовательской работе. Хлое и Аише, которые
прежде не упоминались на первых страницах книг.*

Кейт Грегори

Избранные рекомендации по C++

P.2. Придерживайтесь стандарта ISO C++ (глава 1.1).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rp-Cplusplus>

P.4. В идеале программа должна быть статически типобезопасной (глава 5.1).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rp-typesafe>

P.10. Неизменяемые данные предпочтительнее изменяемых (глава 5.2).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rp-mutable>

P.11. Инкапсулируйте беспорядочные конструкции, а не разбрасывайте их по всему коду (глава 2.1).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rp-library>

I.3. Избегайте синглтонов (глава 3.2).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Ri-singleton>

I.11. Никогда не передавайте владение через простой указатель (T*) или ссылку (T&) (глава 3.1).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Ri-raw>

I.23. Минимизируйте число параметров в функциях (глава 2.2).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Ri-nargs>

I.26. Если нужен кросс-компилируемый ABI, используйте подмножество в стиле C (глава 2.3).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Ri-abi>

I.30. Инкапсулируйте нарушения правил (глава 5.3).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Ri-encapsulate>

F.21. Для возврата нескольких выходных значений используйте структуры или кортежи (глава 4.1).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rf-out-multi>

F.51. Если есть выбор, используйте аргументы по умолчанию вместо перегрузки (глава 1.2).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rf-default-args>

C.45. Не определяйте конструктор по умолчанию, который просто инициализирует переменные-члены; для этой цели лучше использовать внутри-классовые инициализаторы членов (глава 1.3).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rc-default>

C.47. Определяйте и инициализируйте переменные-члены в порядке их объявления (глава 2.4).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rc-order>

C.90. Полагайтесь на конструкторы и операторы присваивания вместо `memset` и `memsetu` (глава 3.3).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rc-memset>

C.131. Избегайте тривиальных геттеров и сеттеров (глава 1.4).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c131-avoid-trivial-getters-and-setters>

Enum.3. Старайтесь использовать классы-перечисления вместо простых перечислений (глава 4.2).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Renum-class>

ES.5. Минимизируйте области видимости (глава 4.3).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Res-scope>

ES.10. Объявляйте имена по одному в каждом объявлении (глава 1.5).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Res-name-one>

ES.22. Не объявляйте переменные, пока не получите значения для их инициализации (глава 5.4).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Res-init>

ES.50. Не приводите переменные с квалификатором `const` к неконстантному типу (глава 3.4).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Res-casts-const>

Per.7. При проектировании учитывайте возможность последующей оптимизации (глава 5.5).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rper-efficiency>

SR.3. Сведите к минимуму явное совместное использование записываемых данных (глава 2.5).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rconc-data>

E.6. Используйте идиому RAII для предотвращения утечек памяти (глава 5.6).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Re-raii>

E.28. При обработке ошибок избегайте глобальных состояний (например, errno) (глава 3.5).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Re-no-throw>

Con.5. Используйте constexpr для определения значений, которые можно вычислить на этапе компиляции (глава 4.4).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rconst-constexpr>

T.1. Используйте шаблоны для повышения уровня абстрактности кода (глава 4.5).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rt-raise>

T.10. Задавайте концепции для всех аргументов шаблона (глава 4.6).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rt-concepts>

T.120. Используйте метапрограммирование шаблонов, только когда это действительно необходимо (глава 2.6).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rt-metameta>

SF.7. Не используйте using namespace в глобальной области видимости в заголовочном файле (глава 3.6).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rs-using-directive>

NR.2. Функции не обязательно должны иметь только один оператор возврата (глава 1.6).

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rnr-single-return>

Предисловие

Я получил истинное удовольствие, прочитав книгу «Красивый C++». Особенно мне понравилось, что она представляет основные рекомендации C++ совсем не так, как *C++ Core Guidelines*¹. В Руководстве правила описываются довольно кратко, густо сдобрены техническими терминами и предполагают широкое использование средств статического анализа. Эта же книга рассказывает истории, в большинстве своем взятые из игровой индустрии и основанные на эволюции кода и методов на протяжении десятилетий. В ней правила представлены с точки зрения разработчика с акцентом на преимущества, которые можно получить, следуя этим правилам, и на неприятностях, которые могут возникнуть в результате их игнорирования. Каждое правило сопровождается более обширным обсуждением, чем может предложить *C++ Core Guidelines*.

Руководство стремится максимально полно охватить все правила. Естественно, подбор правил написания хорошего кода в этой книге по умолчанию не может быть полным руководством по языку. Но если ограничиться степенью полноты, необходимой для понимания обсуждаемой проблемы, то станет очевидно: Руководство C++ Core не предназначено для систематического чтения. Я рекомендую прочитать в нем разделы In: Introduction и P: Philosophy, чтобы получить представление о цели этого руководства и его концептуальной основе. А для выборочного знакомства с основными правилами создания хорошего кода, исходя из вкуса, своего видения и опыта, я советую прочитать книгу, которую вы держите в руках. Для истинных профессионалов это будет легким и увлекательным чтением. А большинство остальных разработчиков смогут узнать из нее что-то новое и полезное.

*Бьери Страуструп (Bjarne Stroustrup),
июль 2021 года*

¹ Руководство на английском языке свободно доступно по адресу <https://github.com/isocpp/CppCoreGuidelines>. — *Примеч. пер.*

Вступление

Сложность разработки программ на C++ уменьшается с выходом каждого нового стандарта и каждой новой книги. Конференций, блогов и книг более чем достаточно, и это хорошо. Но в мире не хватает инженеров с достаточно высоким уровнем подготовки для решения вполне реальных задач.

Несмотря на постоянное упрощение языка, еще многое предстоит узнать о том, как писать хороший код на C++. Бьерн Страуструп, изобретатель языка C++, и Герб Саттер (Herb Sutter), руководитель органа по стандартизации C++, посвятили много сил и времени созданию учебных материалов как для изучения C++, так и для повышения квалификации разработчиков на C++. Среди них можно назвать *The C++ Programming Language*¹ и *A Tour of C++*², а также *Exceptional C++*³ и *C++ Coding Standards*⁴.

Проблема книг даже такого скромного объема заключается в том, что они отражают состояние дел на момент их издания, тогда как C++ — это постоянно развивающийся язык. То, что было хорошим советом в 1998 году, может потерять актуальность. Развивающемуся языку нужен развивающийся путеводитель.

В руководстве содержатся простые и практические советы по улучшению стиля кода на C++, чтобы вы могли писать правильный, производительный и эффективный код с первой попытки.

¹ *Stroustrup B.* The C++ Programming Language, Fourth Edition. — Boston: Addison-Wesley, 2013 (*Страуструп Б.* Язык программирования C++. 4-е изд.).

² *Stroustrup B.* A Tour of C++, Second Edition. — Boston: Addison-Wesley, 2018 (*Страуструп Б.* Язык программирования C++. Краткий курс. 2-е изд.).

³ *Sutter H.* Exceptional C++. — Reading, MA: Addison-Wesley, 1999 (*Саттер Г.* Новые сложные задачи на C++).

⁴ *Sutter H., Alexandrescu A.* C++ Coding Standards. — Boston: Addison-Wesley, 2004 (*Саттер Г., Александреску А.* Стандарты программирования на C++).

На конференции CppCon в 2015 году Бьерном Страуструпом и Гербом Саттером в ходе их двух¹ основных докладов² был запущен онлайн-ресурс C++ Core Guidelines³. В нем содержатся простые и практичные советы по улучшению стиля кода на C++, чтобы вы могли писать правильный, производительный и эффективный код с первой попытки. Это постоянно развивающийся документ, в котором нуждаются специалисты, пишущие на C++, и авторы будут рады, если вы пришлете им свои предложения с исправлениями и улучшениями. Желательно, чтобы все, от новичков до ветеранов, следовали советам из этого Руководства.

В конце февраля 2020 года на #include discord⁴ Кейт Грегори (Kate Gregory) заявила, что хотела бы издать книгу о Core Guidelines, и я, Гай Дэвидсон, сразу ухватился за эту идею. Кейт выступила на CppCon 2017⁵, где рассмотрела только десять основных правил. Я разделяю ее энтузиазм по продвижению передовых приемов программирования.

Я возглавляю отдел инженерно-технических методов в *Creative Assembly*, старейшей и крупнейшей британской студии, занимающейся разработкой игр. В нем я проработал большую часть из последних 20 с лишним лет, помогая превращать наших замечательных инженеров в великих специалистов. По нашему наблюдению, несмотря на доступность и простоту Core Guidelines, разработчики мало знакомы с этим Руководством. Мы в меру своих сил и возможностей пропагандируем его и поэтому решили написать книгу, потому что литературы о рекомендациях и правилах, перечисленных в Руководстве, пока недостаточно.

Собственно Core Guidelines можно найти по адресу <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>. Оно наполнено замечательными советами, из-за чего порой трудно понять, с чего начать. Можно, конечно, читать

¹ Youtube.com. 2021. CppCon 2015: *Stroustrup B.* Writing Good C++14. Доступно по адресу <https://www.youtube.com/watch?v=1OEu9C51K2A>.

² Youtube.com. 2021. CppCon 2015: *Sutter H.* Writing Good C++14... By Default. Доступно по адресу <https://www.youtube.com/watch?v=hEx5DNLWGgA>.

³ Isocpp.github.io. 2021. C++ Core Guidelines. Copyright © Standard C++ Foundation and its contributors. Доступно по адресу <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>.

⁴ #include <C++>. 2021. #include <C++>. Доступно по адресу <https://www.includecpp.org/>.

⁵ Youtube.com. 2021. CppCon 2017: *Gregory K.* 10 Core Guidelines You Need to Start Using Now. Доступно по адресу <https://www.youtube.com/watch?v=XkDEzfpdcSg>.

все подряд, сверху вниз, но понять и усвоить весь набор рекомендаций без повторного чтения практически невозможно. Рекомендации организованы в 22 основных раздела с такими названиями, как Interfaces («Интерфейсы»), Functions («Функции»), Concurrency («Конкуренция») и т. д. Каждый раздел включает отдельный набор правил и рекомендаций, иногда исчисляемых единицами, иногда десятками. Рекомендации идентифицируются по первой букве из названия раздела и номеру рекомендации в разделе. Например, «F.3. Функции должны быть короткими и простыми» — третья рекомендация в разделе F, Functions.

Все рекомендации построены по одному формату. Они начинаются с названия, которое представлено как посыл к действию (делайте это, не делайте этого, избегайте этого, предпочитайте это), затем следует причина и несколько примеров, а также исключения из правила (если они есть). В конце дается примечание, описывающее, как обеспечить соблюдение данного правила. Примечания по применению включают и советы авторам инструментов статического анализа, и советы разработчикам, как проводить проверку кода. Чтобы читать эти рекомендации, нужен определенный навык; решение о том, какие из них взять на вооружение, является вопросом личных предпочтений. Позвольте нам продемонстрировать, как начать пользоваться мудростью этого Руководства.

В C++ есть свои острые углы, есть и свои пыльные тайники, которые не так часто посещаются в современном C++. Мы хотим увести вас от них и показать, что C++ совсем не трудный и не сложный язык и его использование вполне можно доверить большинству разработчиков.

О КНИГЕ

В этой книге мы предлагаем 30 избранных рекомендаций по программированию на C++, которые сами считаем лучшими. Мы подробно объясняем эти рекомендации в надежде, что вы будете придерживаться хотя бы их, если решите не исследовать остальные советы в Core Guidelines. Рекомендации, которые мы отобрали, не обязательно являются самыми важными, но они, вне всяких сомнений, изменят ваш код к лучшему. Конечно, полезно ознакомиться также со множеством других рекомендаций и следовать им. Мы надеемся, что вы прочтете остальные советы и попробуете их

применить к своему коду. Руководство предназначено для всех разработчиков на C++, с любым уровнем опыта, и эта книга адресована тому же кругу людей. Материал не усложняется по ходу книги, и главы можно читать в любом порядке. Они не зависят друг от друга, хотя кое-где присутствуют ссылки на другие главы. Объем каждой главы примерно 3000 слов, так что вы можете считать эту книгу скорее томиком для чтения на диване по вечерам, чем учебником. Цель книги не в том, чтобы научить вас программировать на C++, а чтобы познакомить с советами, как улучшить свой стиль.

Мы разделили рекомендации на пять частей по шесть глав в соответствии с первоначальной презентацией Кейт Грегори на CppCon в 2017 году.

В части I «Bikeshedding — это плохо» мы представляем рекомендации, которые помогут выбрать верное решение из нескольких вариантов и двигаться дальше с минимумом суеты и споров. Понятие bikeshedding¹ (bike — «велосипед», shed — «навес», то есть bikeshedding — «ставить велосипед под навес». — *Примеч. ред.*) заимствовано из «закона тривиальности» К. Норткота Паркинсона (C. Northcote Parkinson), согласно которому члены организации нередко придают несоразмерное значение тривиальным вопросам, таким как выбор цвета навеса для велосипедов, вместо критериев испытаний атомной электростанции, к которой он, навес, прилагается. Какова причина тривиального подхода? Да просто велосипед — это единственная вещь, о которой все хоть что-то знают.

В части II «Не навредите себе» мы предоставляем рекомендации по предотвращению травм при написании кода. Одна из проблем, связанных с остаточной сложностью C++, заключается в возможности в некоторых ситуациях выстрелить себе в ногу. Например, несмотря на допустимость заполнять список инициализации конструктора в любом порядке, никогда не следует этого делать.

Часть III называется «Прекратите это использовать» и касается элементов языка, сохраненных ради обратной совместимости, а также советов, которые раньше считались ценными, но потеряли свою значимость благодаря нововведениям в языке. По мере развития C++ то, что раньше казалось хорошей

¹ 2021. Доступно по адресу <https://exceptionnotfound.net/bikeshedding-the-daily-software-anti-pattern/>.

идеей, сейчас может оказаться не таким ценным, как предполагалось изначально. Процесс стандартизации исправляет эти моменты, но вы должны знать о них, потому что можете столкнуться с ними, сопровождая старые проекты. Язык C++ гарантирует обратную совместимость: код, написанный 50 лет назад на C, будет компилироваться и сегодня.

Цель книги не в том, чтобы научить вас программировать на C++, а чтобы познакомить с советами, как улучшить свой стиль.

Далее следует часть IV под заголовком «Используйте новую особенность правильно». Такие особенности, как концепции, `constexpr`, структурное связывание и т. д., требуют осторожности при развертывании. Опять же C++ — это развивающийся стандарт, и с каждым выпуском появляется что-то новое, требующее освоения для поддержки. Хотя эта книга не ставит своей целью научить вас новым возможностям, которые определяются стандартом C++20, эти рекомендации помогут вам понять, как воспринимать их.

Часть V, последняя, называется «Пишите хороший код по умолчанию». Здесь описываются простые рекомендации, которые, если следовать им, помогут вам писать хороший код, не сильно задумываясь о происходящем. Придерживаясь их, вы научитесь писать красивый идиоматический код на C++, и он будет понят и оценен вашими коллегами.

На протяжении всей книги, как и в любом хорошем повествовании, возникает и развивается обсуждение разных тем. Мы, авторы, испытывали особое удовольствие от возможности увидеть мотивы, стоящие за рекомендациями, и проанализировать широкое их применение. Надеемся, что такое же удовольствие испытаете и вы при чтении. Многие рекомендации, если внимательно присмотреться, просто иначе формулируют некоторые из фундаментальных истин разработки программного обеспечения. Знание этих истин значительно улучшит ваши навыки программирования.

Мы искренне надеемся, что эта книга вам понравится и принесет пользу¹.

¹ Как уже понятно из введения и вступительного слова авторов, эта книга не просто о программировании, а о программировании, поданном через призму другой книги, название которой — C++ Core Guideline. Чтобы отличать ее от прочих руководств по языку C, упоминаемых далее по тексту, она фигурирует как Руководство — с прописной буквы. Все остальные пособия и труды называются в книге просто руководствами. — *Примеч. ред.*

КОД ПРИМЕРОВ

Все примеры кода доступны на сайте Compiler Explorer. Мэтт Годболт (Matt Godbolt) любезно зарезервировал постоянные ссылки для каждой главы, которые формируются путем присоединения номера главы к базовому адресу <https://godbolt.org/z/cg30-ch>. Например, ссылка <https://godbolt.org/z/cg30-ch1.3> приведет вас к примерам кода для главы 1.3. Мы рекомендуем начать с <https://godbolt.org/z/cg30-ch0.0>, где приводятся инструкции по использованию веб-сайта и взаимодействию с кодом.

*Гай Дэвидсон (Guy Davidson),
@hatcat01 hatcat.com.*

*Кейт Грегори (Kate Gregory),
@gregcons gregcons.com.*

Октябрь 2021 года

Благодарности

Годы 2020-й и 2021-й были для нас довольно беспокойными, и мы хотели бы поблагодарить многих людей, так или иначе оказавших нам поддержку, когда мы работали над этой книгой.

Конечно же, мы хотим поблагодарить Бьерна Страуструпа и Герба Саттера за создание рекомендаций Core Guidelines и за то, что побудили нас написать о них. Мы также хотим поблагодарить участников CppCon за их помощь в обсуждении некоторых из этих рекомендаций.

Наши семьи оказали жизненно необходимую поддержку во время процесса написания, требующего уединения и сосредоточенности. Без поддержки со стороны самых близких нам было бы значительно тяжелее работать.

Легион друзей в дискорд-группе #include со штаб-квартирой на includerpp.org продолжает поддерживать нас в нашей повседневной практике использования C++ с июля 2017 года¹. Мы жертвуем вам одну десятую нашего дохода от этой книги. Низкий вам поклон.

Свою помощь нам оказали также несколько членов комитета ISO WG21 C++, поддерживающего стандарт. Мы хотели бы поблагодарить Майкла Вонга (Michael Wong) и Тони ван Эрда (Tony van Eerd) за их участие.

Все примеры кода доступны на Compiler Explorer² по постоянным и понятным ссылкам, благодаря любезности Мэтта Годболта, создателя этого прекрасного сервиса. Мы выражаем ему нашу благодарность и уверяем, что его усилия оказались, несомненно, очень важными для сообщества C++.

Cppreference.com³ послужил нам отличным исследовательским инструментом при первоначальной подготовке каждой главы, поэтому мы весьма признательны создателю и владельцу сайта Нейту Колю (Nate Kohl), администраторам Повиласу Кананицкасу (Povilas Kanapickas) и Сергею

¹ <https://twitter.com/hatcat01/status/885973064600760320>

² <https://godbolt.org/z/cg30-ch0.0>

³ <https://ru.cppreference.com/w>

Зубкову (Sergey Zubkov), а также Тиму Сонгу (Tim Song) и всем другим участникам проекта и благодарим их за неустанную поддержку этого прекрасного ресурса. Они — герои сообщества.

После написания главы 3.6 нам стало ясно, что своим вдохновением мы во многом обязаны статье Артура О'Дуайера (Arthur O'Dwyer). Большое ему спасибо за его неизменную преданную службу обществу. В его блоге также можно найти рассказы о предпринимаемых им усилиях по раскрытию некоторых самых ранних приключенческих текстовых игровых программ 1970-х и 1980-х годов¹.

Для такой книги, как эта, нужна армия редакторов, поэтому мы выражаем благодарность Бьерну Страуструпу, Роджеру Орру (Roger Orr), Клэр Макрей (Clare Macrae), Артуру О'Дуайеру, Ивану Чукичу (Ivan Čukić), Райнеру Гримму (Rainer Grimm) и Мэтту Годболту.

Неоценимую помощь нам оказала и команда Addison-Wesley, поэтому мы выражаем огромную благодарность Грегори Доенчу (Gregory Doench), Одри Дойл (Audrey Doyle), Асвини Кумару (Aswini Kumar), Менке Мехте (Menka Mehta), Джули Нахил (Julie Nahil) и Марку Таберу (Mark Taber).

¹ <https://quuxplusone.github.io/blog>

Об авторах

Дж. Гай Дэвидсон впервые познакомился с компьютерами благодаря Acorn Atom в 1980 году. Еще будучи подростком, он писал игры для различных домашних компьютеров: Sinclair Research ZX81 и ZX Spectrum, а также Atari ST. После получения степени по математике в Университете Сассекса он увлекся театром и играл на клавишных инструментах в соул-группе. В начале 1990-х стал заниматься разработкой приложений для презентаций, а в 1997-м перешел в игровую индустрию, начав работать в Codemasters в их лондонском офисе.

В 1999 году перешел в Creative Assembly, где сейчас возглавляет отдел инженерно-технических методов. Работает над франшизой *Total War*, курируя дискографию, а также формулируя и развивая стандарты программирования в команде инженеров. Входит в состав консультативных советов IGGI, группы BSI C++ и комитета ISO C++. Занимает пост ответственного за стандарты в комитете ACCU и входит в программный комитет конференции ACCU. Является модератором на дискорд-сервере #include <C++>. Отвечает за внутреннюю политику и нормы в нескольких организациях. Его можно увидеть на конференциях и встречах по C++, особенно на посвященных добавлению методов линейной алгебры в стандартную библиотеку.

В свободное время он оказывает наставническую поддержку по вопросам программирования на C++ через Prospela и BAME in Games; помогает школам, колледжам и университетам через UKIE, STEMNet и в качестве Video Game Ambassador; практикует и преподает тай-чи в стиле У; изучает игру на фортепиано; поет первый бас в Брайтонском фестивальном хоре; управляет местным киноклубом; является членом BAFTA с правом голоса; дважды баллотировался (безуспешно) на выборах в местный совет от имени партии зеленых Англии и Уэльса; пытается выучить испанский. Иногда его можно встретить за карточным столом играющим в бридж по пенни за очко. Вероятно, у него есть и другие увлечения: он большой непоседа.

Кейт Грегори познакомилась с программированием в Университете Ватерлоо в 1977 году и никогда не оглядывалась назад с сомнением или

сожалением. Имеет степень в области химического машиностроения, что лишний раз подтверждает, что диплом не всегда говорит о наклонностях человека. На цокольном этаже ее сельского дома в Онтарио есть небольшая комната со старыми компьютерами PCT, C64, домашней системой 6502 и т. д., служащими напоминаниями о более простых временах. С 1986 года вместе с мужем руководит компанией Gregory Consulting, помогая клиентам по всему миру.

Кейт выступала с докладами на пяти континентах, любит искать заковыристые головоломки и затем делиться их решением, а также проводит много времени, добровольно участвуя в различных мероприятиях, посвященных языку C++. Самым уважаемым из них является группа `#include <C++>`, которая оказывает огромное влияние на эту отрасль, делает программирование на C++ более гостеприимным и дружелюбным. Их дискорд-сервер — теплое, уютное место для изучения C++ новичками и одновременно уютная компания для совместной работы над статьями для WG21, позволяющими взглянуть по-иному на язык, который мы все используем, или же... что-то среднее между ними двумя.

Ее отрывают от клавиатуры внуки, озера и кемпинги Онтарио, весла для каноэ и дым костра, а также соблазны аэропортов по всему миру. Гурманка, игрок в настольные игры, безотказная палочка-выручалочка, не способная ответить отказом на просьбу о помощи, она так же активна в реальной жизни, как и в интернете, но менее ярка и заметна. После того как в 2016 году пережила меланому IV стадии, она стала меньше беспокоиться о том, что думают другие и чего от нее ожидают, и больше о том, чего она хочет для своего будущего. Это дает свои результаты¹.

¹ Вся книга написана от лица обоих авторов, но иногда слово в повествовании представляется одному Гаю Дэвидсону. Отступления в тексте, в которых он щедро делится с читателями личными знаниями, опытом и житейской мудростью, оформлены как исторические экскурсы. — *Примеч. ред.*

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.



BIKESHEDDING – ЭТО ПЛОХО

Глава 1.1 P.2. Придерживайтесь стандарта ISO C++.

Глава 1.2 F.51. Если есть выбор, используйте аргументы по умолчанию вместо перегрузки.

Глава 1.3 C.45. Не определяйте конструктор по умолчанию, который просто инициализирует переменные-члены; для этой цели лучше использовать внутриклассовые инициализаторы членов.

Глава 1.4 C.131. Избегайте тривиальных геттеров и сеттеров.

Глава 1.5 ES.10. Объявляйте имена по одному в каждом объявлении.

Глава 1.6 NR.2. Функции не обязательно должны иметь только один оператор возврата.

ГЛАВА 1.1

Р.2. Придерживайтесь стандарта ISO C++

ЧТО ТАКОЕ СТАНДАРТ ISO C++

Эта книга посвящена приемам написания хорошего кода. Поэтому первый совет — придерживайтесь стандарта ISO C++. Но что это означает на практике?

История C++

C++ первоначально не был стандартизированным языком. Это было расширение языка программирования C, изобретенное Бьерном Страуструпом¹. Оно получило название «C с классами». В то время язык C тоже не был стандартизированным: Бьерн представил свое расширение в виде препроцессора под названием Cpre. Препроцессор обеспечивал поддержку классов и производных классов с общедоступными/приватными уровнями доступа, дружественными классами, перегрузкой операторов присваивания, конструкторами и деструкторами. Кроме того, препроцессор поддерживал встраиваемые функции и аргументы функций по умолчанию, а также проверку типов аргументов функций.

В 1982 году Бьерн начал работу над новым проектом под названием C++, включающим дополнительные возможности, такие как виртуальные функции, перегрузка функций и операторов, ссылки, константы и динамическое распределение памяти. Он также создал интерфейс C++ для компиляторов C под названием Cfront. Код на C++ передавался Cfront, который

¹ *Stroustrup B.* 1995. A History of C++: 1979–1991, www.stroustrup.com/hopl2.pdf.

затем компилировал его в код на языке С. Он также написал книгу под названием *The C++ Programming Language* (широко известную как *TCPL*), опубликованную в 1985 году. Она послужила определяющим руководством по языку C++, благодаря ей стали появляться коммерческие компиляторы.

Даже после широкого распространения этих компиляторов Бьерн продолжал работать над C++, добавляя новые возможности в то, что стало называться C++2.0. К ним относятся множественное наследование, абстрактные базовые классы, статические и константные функции-члены, защищенный уровень доступа, а также улучшения существующих возможностей. В тот период наблюдался лавинообразный рост популярности C++. По оценкам Бьерна, количество пользователей удваивалось каждые 7,5 месяца.

Появились конференции, журналы и книги, а конкурирующие между собой реализации компилятора продемонстрировали, что необходимо нечто более точное и регламентирующее, чем *TCPL*. В 1989 году Дмитрий Ленков (Dmitry Lenkov) из HP написал предложение по стандартизации C++ в Американском национальном институте стандартов (American National Standards Institute, ANSI). В нем он указывал на необходимость тщательного и подробного определения каждой особенности языка для предотвращения неконтролируемого увеличения числа диалектов. Он также определял необходимость реализации дополнительных возможностей, таких как обработка исключений и создание стандартной библиотеки. Комитет ANSI C++, X3J16, впервые собрался в декабре 1989 года. *Аннотированное справочное руководство* (Annotated Reference Manual, ARM), написанное Маргарет Эллис (Margaret Ellis) и Бьерном и опубликованное в 1990 году, стало единым основополагающим описанием всего C++. Это руководство было создано специально, чтобы ускорить начало работ по стандартизации ANSI C++.

Конечно, это был не только американский проект, и в нем приняли участие многие представители других стран. В 1991 году был созван комитет ISO C++ WG21, и с тех пор эти два комитета проводили совместные заседания с целью выработать черновой стандарт для публичного рассмотрения через четыре года, с надеждой на появление официального стандарта двумя годами после. Однако первый стандарт, ISO/IEC 14882:1998, был опубликован только в сентябре 1998-го, почти через девять лет после упомянутого первого собрания.

Но история на этом не заканчивается. Работа над исправлением ошибок в стандарте продолжалась, и в 2003 году был опубликован стандарт C++03.

Конечно же, работы по добавлению дополнительных возможностей и тогда не прекратились, было продолжено дальнейшее развитие языка. В число новых возможностей вошли: `auto`, `constexpr`, `decltype`, семантика перемещения, `for` с диапазонами, унифицированная инициализация, лямбда-выражения, правосторонние (`rvalue`) ссылки, статические утверждения, вариативные шаблоны... Список продолжал пополняться, и график разработки растягивался. В конце концов, следующая версия стандарта вышла в 2011 году, еще до того, как все его авторы успели забыть, что C++ был когда-то растущим языком.

Учитывая, что C++03 был исправлением C++98, между первым стандартом и стандартом C++11 имел место 13-летний разрыв. Стало понятно, что такой длительный период между публикациями стандартов никому не интересен, поэтому была разработана «модель транспорта»: новый стандарт будет публиковаться каждые три года, и если какая-то особенность языка не будет готова к этому моменту, отчет о ее готовности пойдет «следующим рейсом» — пополнит версию, которая будет опубликована три года спустя. С тех пор стандарты C++14, C++17 и C++20 выходили согласно графику.

ИНКАПСУЛЯЦИЯ ВАРИАЦИЙ

Вариации в окружении времени выполнения

В стандарте очень мало говорится о требованиях к окружению, в котором выполняется программа на C++. Операционная система не регламентируется. Хранилища файлов не являются обязательными. Экран тоже необязателен. Получается, что программа, написанная для типичного окружения рабочего стола, может нуждаться во вводе с помощью мыши и выводе в оконном режиме, что потребует специального кода для каждой конкретной системы.

Написание полностью переносимого кода для такой программы невозможно. Стандарт ISO C++ имеет очень маленькую библиотеку по сравнению с такими языками, как C# и Java. Это всего лишь спецификация для разработчиков, реализующих стандарт ISO C++. Все дело в том, что стандартные библиотеки C# и Java предоставляются владельцами языка, но C++ не имеет финансируемой организации по разработке библиотек. Вы должны использовать уникальные особенности каждой целевой среды для поддержки тех частей функционала, которые недоступны в стандартной библиотеке языка. Обычно они предлагаются в виде заголовочного файла и библиотеки. Как правило, таких файлов очень много в каждой системе.

Насколько возможно, прячьте их за вашими собственными интерфейсами. Минимизируйте количество вариаций между версиями кодовой базы, предназначенными для разных систем.

Например, вам может понадобиться узнать, нажимается ли конкретная клавиша на клавиатуре. Один из возможных подходов — применение препроцессора для определения используемой платформы и вызов соответствующего фрагмента кода, например:

```
#if defined WIN32
auto a_pressed = bool{GetKeyState('A') & 0x8000 != 0};
#elif defined LINUX
auto a_pressed = /* в действительности здесь следует большой фрагмент кода */
#endif
```

Очень неуклюжее решение: оно работает на неправильном уровне абстракции. Код, относящийся к Windows и Linux¹, должен находиться в отдельных файлах, соответствующих системе, и импортироваться через заголовочные файлы, чтобы использующий их вызов выглядел так:

```
auto a_pressed = key_state('A');
```

Функция `key_state` представляет собой интерфейс, инкапсулирующий это расширение. Реализация делает все, что нужно для соответствующей платформы, и ваш поток управления не захламляется мусором в виде макросов препроцессора. Разделение каждой реализации по разным файлам обеспечивает дополнительную поддержку этой абстракции.

Вариации на уровне языка C++ и компилятора

Разработчики компиляторов C++ должны максимально полно и точно поддерживать стандарт, если намерены объявить свой компилятор соответствующим стандарту. Однако это требование не связывает им руки и оставляет открытой дверь для добавления дополнительных возможностей или расширений. Например, в GCC были добавлены дополнительные свойства типов (type trait), такие как `__has_trivial_constructor` и `__is_abstract`, до того как они появились в стандарте. Оба присутствовали в библиотеке свойств типов, начиная с C++11, под другими именами: `std::is_trivially_constructible` и `std::is_abstract`.

Обратите внимание, что имя `__is_abstract` начинается с двойного подчеркивания: имена, начинающиеся с двойного подчеркивания, зарезервированы

¹ <https://stackoverflow.com/questions/41600981/how-do-i-check-if-a-key-is-pressed-on-c>

стандартом для разработчиков компилятора. Разработчикам *не* разрешается добавлять новые идентификаторы в пространство имен `std`, так как впоследствии они могут быть добавлены в стандарт с совершенно другим значением. На практике это означает, что программист, работая на C++, может ненамеренно написать код, который выглядит как использующий стандартные функции, но в действительности использует функции, характерные для специфического компилятора. Хороший способ защититься от таких случайностей — написать и протестировать свою программу с несколькими компиляторами и операционными системами, чтобы обнаружить не закрепленный в стандарте код.

Две упомянутые выше функции были реализованы по причине их полезности для метапрограммирования. В действительности они оказались настолько полезными, что позже были добавлены в стандарт. Многие части стандарта, касающиеся как самого языка, так и стандартной библиотеки, появлялись сначала как дополнительные возможности в популярных инструментах и библиотеках. Иногда использование нестандартных функций неизбежно.

Расширения для C++

Некоторые разработчики библиотек добавляют свои расширения. Например, библиотека Qt¹ использует так называемые сигналы и слоты для организации взаимодействий между объектами. С этой целью в библиотеку добавлены три символа: `Q_SIGNALS`, `Q_SLOTS` и `Q_EMIT`. В исходном коде эти ключевые слова выглядят как любые другие ключевые слова языка. Qt предоставляет инструмент под названием `moc`, который анализирует эти ключевые слова и преобразует их в конструкции, понятные компилятору C++, точно так же, как раньше `Sfront` преобразовывал код на C++ в конструкции, понятные компиляторам языка C.

Следует иметь в виду, что стандарт предлагает то, чего нет в этих расширениях: строго определенную семантику. Стандарт ISO C++ однозначен, и это одна из причин, почему его так трудно читать. Можно без ограничений использовать расширения языка, но не забывать о факторе переносимости. В частности, Qt прилагает титанические усилия для достижения переносимости кода между различными платформами. Однако никто не гарантирует, что эти расширения будут присутствовать в других реализациях или что они будут иметь там такое же значение.

¹ <https://doc.qt.io/>