Стандарты программирования на C++

C++ Coding Standards

101 Rules, Guidelines, and Best Practices

Herb Sutter
Andrei Alexandrescu

ADDISON–WESLEY

Boston

Стандарты программирования на C++

101 правило и рекомендация

Герб Саттер
Андрей Александреску

Издательский дом "Вильямс"
Москва • Санкт-Петербург • Киев
2008

ББК 32.973.26-018.2.75
С21
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция канд. техн. наук *И.В. Красикова*

По общим вопросам обращайтесь в Издательский дом "Вильямс"
по адресу: info@williamspublishing.com, http://www.williamspublishing.com

Саттер, Герб, **Александреску**, Андрей.

С21 Стандарты программирования на C++. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2008. — 224 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-0859-9 (рус.)

Эта книга поможет новичку стать профессионалом, так как в ней представлен сконцентрированный лучший опыт программистов на C++, обобщенный двумя экспертами мирового класса.

Начинающий программист найдет в ней простые и понятные рекомендации для ежедневного использования, подкрепленные примерами их конкретного применения на практике.

Опытные программисты найдут в ней советы и новые рекомендации, которые можно сразу же принять на вооружение. Программисты-профессионалы могут использовать эту книгу как основу для разработки собственных стандартов кодирования, как для себя лично, так и для группы, которой они руководят.

Конечно, книга рассчитана в первую очередь на профессиональных программистов с глубокими знаниями языка, однако она будет полезна любому, кто захочет углубить свои знания в данной области.

ББК 32.973.26-018.2.75

ISBN 978-5-8459-0859-9 (рус.)
ISBN 0-321-11358-6 (англ.)

Оглавление

Содержание

Миллионам нынешних программистов на C++

Предисловие

Идите проторенной дорогой — делайте одинаковые вещи одинаковыми способами. Накапливайте идиомы. ***Стандартизируйте****. Единственное отличие между вами и Шекспиром — в количестве используемых идиом, а не в размере словаря.*

— Алан Перлис (Alan Perlis) [выделено нами]

Лучшее в стандарте то, что он предоставляет богатый выбор.

— Приписывается разным людям

Мы бы хотели, чтобы эта книга стала основой для стандартов кодирования, используемых вашей командой, по двум основным причинам.

- *Стандарты кодирования должны отражать лучший опыт проб и ошибок всего сообщества программистов*. В них должны содержаться проверенные идиомы, основанные на опыте и твердом понимании языка. В частности, стандарт кодирования должен основываться на исчерпывающем анализе литературы по разработке программного обеспечения, и объединять воедино правила, рекомендации и наилучшие практические решения, которые в противном случае оказываются разбросанными по многочисленным источникам.
- *Природа не терпит пустоты*. Если вы не разработаете набор правил, то это сделает кто-то другой. Такие "самопальные" стандарты, как правило, грешат тем, что включают нежелательные для стандарта требования; например, многие из них, по сути, заставляют программистов использовать C++ просто как улучшенный C.

Множество таких плохих стандартов кодирования разработаны людьми, которые недостаточно хорошо понимают язык программирования C++ или пытаются чрезмерно детализировать его применение. Плохой стандарт кодирования быстро теряет кредит доверия, и в результате несогласие или неприятие программистами части его положений распространяется на весь стандарт целиком, перечеркивая содержащиеся в нем различные положительные советы и рекомендации. И это — в лучшем случае, потому что в худшем случае такой стандарт и его выполнение могут быть навязаны руководством.

Как пользоваться этой книгой

Думать. Надо добросовестно следовать умным советам, но делать это не вслепую. Во многих разделах этой книги есть подраздел "Исключения", в котором приводятся нестандартные, редко встречающиеся ситуации, когда совет из основного раздела может оказаться неприменим. Никакое количество даже самых хороших (мы на это надеемся) советов не могут заменить голову.

Каждая команда разработчиков отвечает за принятие собственных стандартов и несет ответственность за них и за неукоснительное следование им. Ваша команда — не исключение. Если вы — руководитель, предложите членам своей группы поучаствовать в разработке стандартов, которым группа будет следовать в своей работе. Люди всегда охотнее следуют правилам, которые они сами для себя вырабатывают, чем тем, которые им навязаны.

Эта книга предназначена для того, чтобы послужить основой для вашего стандарта кодирования и быть в той или иной мере включенной в него. Это — не ultima ratio в стандартах кодирования, и ваша группа может разработать (или прибавить) свои правила, в наибольшей степени подходящие для вашей конкретной группы или для конкретной решаемой задачи, так что вы не должны быть скованы этой книгой по рукам и ногам. Тем не менее, мы надеемся, что эта книга поможет вам сберечь время и усилия при разработке собственного стандарта кодирования, а также будет способствовать повышению его качества и последовательности.

Ознакомьте членов своей команды с этой книгой, и после того, как они полностью прочтут ее и познакомятся со всеми рекомендациями и их обоснованиями, решите, какие из них подходят вам, а какие в силу особенностей вашего проекта для вас неприменимы. После этого строго придерживайтесь выбранной стратегии. После того как командный стандарт принят, он не должен нарушаться иначе как по согласованному решению всей команды в целом.

И наконец, периодически творчески пересматривайте собственные стандарты с учетом практического опыта, приобретенного всей командой при работе над проектом.

Стандарты кодирования и вы

Хорошие стандарты кодирования могут принести немалую выгоду с различных точек зрения.

- *Повышение качества кода*. Работа в соответствии со стандартом приводит к однотипному решению одинаковых задач, что повышает ясность кода и упрощает его сопровождение.
- *Повышение скорости разработки*. Разработчику не приходится решать все задачи и принимать решения "с нуля".
- *Повышение уровня взаимодействия в команде*. Наличие стандарта позволяет уменьшить разногласия в команде и устранить ненужные дебаты по мелким вопросам, облегчает понимание и поддержку чужого кода членами команды.
- *Согласованность в работе*. При использовании стандарта разработчики направляют свои усилия в верном направлении, на решение действительно важных задач.

В напряженной обстановке, при жестких временны*х* рамках люди обычно делают то, чему их учили, к чему они привыкли. Вот почему в больницах в пунктах первой помощи предпочитают опытных, тренированных сотрудников — даже хорошо обученные и знающие новички склонны к панике.

У разработчиков программного обеспечения регулярно возникают ситуации, когда что-то надо было сделать еще вчера — на позавчера. Когда на нас давит график работ (который к тому же имеет тенденцию сдвигаться в одном направлении, и то, что по плану должно было заработать завтра, от нас начинают требовать еще вчера…), мы работаем так, как приучены. Неряшливые программисты, которые даже при обычной неспешной работе не помнят о правильных принципах разработки программного обеспечения (а то и вовсе не знакомы с ними), при нехватке времени окажутся еще небрежнее, а их код будет изобиловать ошибками. Соответственно, программист, который выработал в себе хорошие привычки и регулярно ими пользуется, при "повышенном давлении" будет продолжать выдавать качественный код.

Стандарты кодирования, приведенные в этой книге, представляют собой набор рекомендаций по написанию высококачественного кода на C++. В них сконцентрирован богатый коллективный опыт всего сообщества программистов на C++. Многие из этих знаний разбросаны по частям по самым разным книгам, но не меньшее количество знаний передается изустно. Мы постарались собрать разрозненные сведения в одной книге в виде коллекции ясных, компактных правил с пояснениями, простых для понимания и следования им.

Конечно, даже самые лучшие стандарты не могут помешать написанию плохого кода. То же можно сказать о любом языке программирования, процессе или методологии. Хороший набор стандартов воспитывает хорошие привычки и дисциплину, превышающую обычные

нормы. Это служит хорошим фундаментом для дальнейшего усовершенствования и повышения квалификации. Это не преувеличение и не красивые слова — перед тем, как начать писать стихи, надо владеть словарным запасом и знать грамматику. Мы надеемся, что наша книга упростит для вас путь к поэзии программирования.

Эта книга предназначена для программистов всех уровней.

Если вы начинающий программист — мы надеемся, что рекомендации и их пояснения достаточно поучительны и помогут вам в понимании того, какие стили и идиомы C++ поддерживает наиболее естественным образом. В описании каждого правила и рекомендации приводится краткое обоснование и обсуждение, чтобы вы не просто механически запомнили правило, а поняли его суть.

Для программистов среднего и высокого уровня при описании каждого правила приводится список ссылок, который позволит вам углубленно изучить заинтересовавший вас вопрос, проведя поиск корней правила в системе типов, грамматике и объектной модели C++.

Каким бы ни был ваш уровень как программиста — вероятно, вы работаете над сложным проектом не в одиночку, а в команде. Именно в этом случае разработка стандартов кодирования окупается сполна. Вы можете использовать их для того, чтобы подтянуть всю свою команду к одному, более высокому уровню, и обеспечить повышение качества разрабатываемого кода.

Об этой книге

Основными принципами дизайна данной книги являются следующие.

- *Краткость — сестра таланта*. Чем больше стандарт кодирования по размеру, тем больше шансов, что он будет благополучно проигнорирован. Читают и используют обычно короткие стандарты. Длинные разделы, как правило, просто просматривают "по диагонали", короткие статьи обычно удостаиваются внимательного прочтения.
- *Никакой материал не должен вызывать дискуссий*. Эта книга документирует широко используемые стандарты, а не изобретает их. Если некоторая рекомендация не применима во всех ситуациях, то мы так и пишем — "подумайте о применении X" вместо "делайте X". Кроме того, к каждому правилу указаны все общепринятые исключения.
- *Весь материал должен быть обоснован*. Все рекомендации в этой книге взяты из существующих печатных работ. В конце книги приведен список использованной литературы по C++.
- *Материал не должен быть банален*. Мы не даем рекомендации, которым вы и так следуете, которые обеспечиваются компилятором или которые уже изложены в других разделах.
    - Например, "не возвращайте указатель/ссылку на локальную переменную" — хороший совет, но он не включен в данную книгу, поскольку практически все компиляторы выдают соответствующее предупреждение, к тому же этот вопрос раскрывается в первом разделе книги.
    - Рекомендация "используйте редактор (компилятор, отладчик)" — тоже хороший совет, но, конечно, вы используете эти инструменты и без нашего напоминания. Вместо этого мы рекомендуем использовать автоматизированные системы сборки программ и системы управления версиями.
    - Еще один совет — "не злоупотребляйте `goto`" — исходя из нашего опыта, и так широко известен всем программистам, так что нет смысла повторяться.

Каждый раздел состоит из следующих частей.

- *Заглавие*. Краткое название раздела, поясняющее, о чем будет идти речь.
- *Резюме*. Краткое изложение сути вопроса.

- *Обсуждение*. Расширенное пояснение рекомендации. Зачастую включает краткое обоснование, но учтите, что полную информацию по данному вопросу следует искать в приведенных ссылках.
- *Примеры* (если таковые имеются). Примеры, демонстрирующие правило или позволяющие лучше его понять и запомнить.
- *Исключения* (если таковые имеются). Описание ситуаций (обычно редких), когда приведенное правило неприменимо. Однако остерегайтесь попасть в ловушку, думая, что ваш случай особый и что в вашей ситуации это правило неприменимо, — обычно при здравом размышлении оказывается, что ничего особого в вашей ситуации нет и описанное правило может быть с успехом вами применено.
- *Ссылки*. В приведенной в этом подразделе литературе по C++ вы найдете более полный анализ рассматриваемого в разделе вопроса.

В каждой части книги имеется "наиболее важный раздел" — обычно это первый раздел части. Однако иногда это правило нарушалось в связи с необходимостью последовательного и связного изложения материала.

Благодарности

Мы от всей души благодарны редактору серии Бьярну Страуструпу (Bjarne Stroustrup), редакторам Питеру Гордону (Peter Gordon) и Дебби Лафферти (Debbie Lafferty), а также Тирреллу Албаху (Tyrrell Albaugh), Ким Бодихаймер (Kim Boedigheimer), Джону Фуллеру (John Fuller), Бернарду Гаффни (Bernard Gaffney), Курту Джонсону (Curt Johnson), Чанде Лири-Коту (Chanda Leary-Coutu), Чарли Ледди (Charles Leddy), Хитеру Муллэйн (Heather Mullane), Чути Прасертсиху (Chuti Prasertsith), Ларе Вайсонг (Lara Wysong) и всем остальным работникам издательства Addison-Wesley, помогавшим нам в нашей работе над этим проектом. Нам было очень приятно работать с ними.

На идею и оформление книги нас натолкнули несколько источников, включая такие, как [Cline99], [Peters99], а также работы легендарного и широко цитируемого Алана Перлиса (Alan Perlis).

Особая благодарность тем людям, чьи отзывы помогли нам сделать многие части книги намного лучшими, чем они были бы без этих замечаний. Особенно большое влияние на книгу оказали острые комментарии Бьярна Страуструпа. Мы очень хотим поблагодарить за активное участие в обсуждении материала и высказанные замечания таких людей, как Дэйв Абрамс (Dave Abrahams), Маршалл Клайн (Marshall Cline), Кевлин Хенни (Kevlin Henney), Говард Хиннант (Howard Hinnant), Джим Хайслоп (Jim Hyslop), Николаи Джосаттис (Nicolai Josuttis), Йон Калб (Jon Kalb), Макс Хесин (Max Khesin), Стен Липпман (Stan Lippman), Скотт Мейерс (Scott Meyers) и Дэвид Вандевурд (Daveed Vandevoorde). Кроме того, отдельное спасибо хотелось бы сказать Чаку Аллисону (Chuck Allison), Самиру Байяю (Samir Bajaj), Марку Барбуру (Marc Barbour), Тревису Брауну (Travis Brown), Нилу Кумбесу (Nil Coombes), Дамиану Дечеву (Damian Dechev), Стиву Дьюхарсту (Steve Dewhurst), Питеру Димову (Peter Dimov), Аттиле Фехеру (Attila Feher), Алану Гриффитсу (Alan Griffiths), Мичи Хеннингу (Michi Henning), Джеймсу Канзе (James Kanze), Бартошу Милевски (Bartosz Milewski), Мэтту Маркусу (Matt Marcus), Балогу Палу (Balog Pal), Владимиру Прусу (Vladimir Prus), Дэну Саксу (Dan Saks), Люку Вагнеру (Luke Wagner), Мэтью Вильсону (Matthew Wilson) и Леору Золману (Leor Zolman).

Как обычно, все оставшиеся в книге ошибки и недосмотры — на нашей совести, а не на их.

Герб Саттер (Herb Sutter)
Андрей Александреску (Andrei Alexandrescu)

Сиэтл, сентябрь 2004

Вопросы организации и стратегии

Если бы строители строили здания так же, как программисты пишут программы, — то первый же залетевший дятел разрушил бы всю цивилизацию.

— Джеральд Вайнберг (Gerald Weinberg)

Следуя великой традиции C и C++, мы начинаем отсчет с нуля. Главный совет — под номером 0 — говорит о том, что основной советчик по поводу стандартов кодирования — наши чувства и ощущения.

Остальная часть этой главы состоит из небольшого количества тщательно отобранных вопросов, которые не имеют прямого отношения к коду и посвящены важным инструментам и методам написания надежного кода.

В этом разделе книги наиболее важной мы считаем нулевую рекомендацию — “Не мелочитесь, или Что не следует стандартизировать”.

0. Не мелочитесь, или Что не следует стандартизировать

Резюме

Скажем кратко: не мелочитесь.

Обсуждение

Вопросы персонального вкуса, которые не влияют на корректность и читаемость кода, не относятся к стандарту кодирования. Любой профессиональный программист сможет легко прочесть и записать код, форматирование которого немного отличается от того, которым он обычно пользуется.

Используйте одно и то же форматирование в пределах одного исходного файла или даже целого проекта, поскольку переход от одного стиля форматирования к другому в пределах одного фрагмента исходного текста достаточно сильно раздражает. Но не пытайтесь обеспечить одно и то же форматирование в разных проектах или в целом по всей компании.

Вот несколько вопросов, в которых не важно точное следование правилу, а требуется всего лишь последовательное применение стиля, используемого в файле, с которым вы работаете.

- *Не следует определять конкретный размер отступа, но следует использовать отступы для подчеркивания структуры программы*. Для отступа используйте то количество символов, которое вам нравится, но это количество должно быть одинаково, как минимум, в пределах файла.
- *Не определяйте конкретную длину строки, но она должна оставлять текст удобочитаемым*. Используйте ту длину строки, которая вам по душе, но не злоупотребляйте ею. Исследования показали, что легче всего воспринимается текст, в строке которого находится до десяти слов.
- *Следует использовать непротиворечивые соглашения об именовании, не слишком мелочно регламентируя его*. Имеется только два императивных требования по поводу именования: никогда не используйте имена, начинающиеся с подчеркивания или содержащие двойное подчеркивание, и всегда используйте для макросов только прописные буквы (`ONLY_UPPERCASE_NAMES`), при этом никогда не применяя в качестве имен макросов обычные слова или сокращения (включая распространенные параметры шаблонов, такие как `T` или `U`; запись `#define T` *`anything`* может привести к крупным неприятностям). В остальных случаях используйте непротиворечивые значимые имена и следуйте соглашениям, принятым для данного файла или модуля. (Если вы не можете сами разработать соглашение об именовании, попробуйте воспользоваться следующим: имена классов, функций и перечислений должны выглядеть как `LikeThis`, имена переменных — `likeThis`, имена закрытых членов-данных — `likeThis_`, и имена макросов — `LIKE_THIS`.)
- *Не предписывайте стиль комментариев (кроме тех случаев, когда специальный инструментарий использует их для документирования), но пишите только нужные и полезные комментарии*. Вместо комментариев пишите, где это возможно, код (см., например, руководство 16). Не пишите комментарии, которые просто повторяют код. Комментарии должны разъяснять использованный подход и обосновывать его.

И наконец, не пытайтесь заставлять использовать устаревшие правила (см. примеры 2 и 3), даже если они имеются в старых стандартах кодирования.

Примеры

Пример 1. Размещение фигурных скобок. Нет никакой разницы в плане удобочитаемости следующих фрагментов:

```
void using_k_and_r_style() {
// ...
}
void putting_each_brace_on_its_own_line()
{
// ...
}
void or_putting_each_brace_on_its_own_line_indented()
    {
// ...
    }
```

Все профессиональные программисты могут легко читать *и писать* в каждом из этих стилей без каких-либо сложностей. Но следует быть последовательным. Не размещайте скобки как придется или так, что их размещение будет скрывать вложенность областей видимости, и пытайтесь следовать стилю, принятому в том или ином файле. В данной книге размещение скобок призвано обеспечить максимальную удобочитаемость, при этом оставаясь в рамках, определенных редакторскими ограничениями.

Пример 2. Пробелы или табуляция. В некоторых командах использование табуляции запрещено (например, [BoostLRG]) на том основании, что размер табуляции варьируется от редактора к редактору, а это приводит к тому, что отступы оказываются слишком малы или слишком велики. В других командах табуляции разрешены. Важно только быть последовательным. Если вы позволяете использовать табуляцию, убедитесь, что такое решение не будет мешать ясности кода и его удобочитаемости, если члены команды будут сопровождать код друг друга (см. руководство 6). Если использование табуляции не разрешено, позвольте редактору преобразовывать пробелы в табуляции при чтении исходного файла, чтобы программисты могли работать с ними в редакторе. Однако убедитесь, что при сохранении файла табуляция будет вновь преобразована в пробелы.

Пример 3. Венгерская запись. Запись, при которой информация о типе включается в имя переменной, приносит пользу в языке программирования, небезопасном с точки зрения типов (особенно в C); возможна, хотя и не приносит никакой пользы (только недостатки) в объектно-ориентированных языках; и невозможна в обобщенном программировании. Таким образом, стандарт кодирования C++ не должен требовать использования венгерской записи, более того, может потребовать ее запрета.

Пример 4. Один вход, один выход (Single entry, single exit — "SESE"). Исторически некоторые стандарты кодирования требуют, чтобы каждая функция имела в точности один выход, что подразумевает одну инструкцию `return`. Такое требование является устаревшим в языках, поддерживающих исключения и деструкторы, так что функции обычно имеют несколько неявных выходов. Вместо этого стоит следовать стандарту наподобие рекомендации 5, которая требует от функций простоты и краткости, что делает их более простыми для понимания и более устойчивыми к ошибкам.

Ссылки

[BoostLRG] • [Brooks95] §12 • [Constantine95] §29 • [Keffer95] p. 1 • [Kernighan99] §1.1, §1.3, §1.6-7 • [Lakos96] §1.4.1, §2.7 • [McConnell93] §9, §19 • [Stroustrup94] §4.2-3 • [Stroustrup00] §4.9.3, §6.4, §7.8, §C.1 • [Sutter00] §6, §20 • [SuttHysl01]

1. Компилируйте без замечаний при максимальном уровне предупреждений

Резюме

Следует серьезно относиться к предупреждениям компилятора и использовать максимальный уровень вывода предупреждений вашим компилятором. Компиляция должна выполняться без каких-либо предупреждений. Вы должны понимать все выдаваемые предупреждения и устранять их путем изменения кода, а не снижения уровня вывода предупреждений.

Обсуждение

Ваш компилятор — ваш друг. Если он выдал предупреждение для определенной конструкции, зачастую это говорит о потенциальной проблеме в вашем коде.

Успешная сборка программы должна происходить молча (без предупреждений). Если это не так, вы быстро приобретаете привычку не обращать внимания на вывод компилятора и можете пропустить серьезную проблему (см. рекомендацию 2).

Чтобы избежать предупреждений, надо понимать, что они означают, и перефразировать ваш код так, чтобы устранить предупреждения и сделать предназначение кода более понятным как для компилятора, так и для человека.

Делайте это всегда — даже если программа выглядит корректно работающей. Делайте это, даже если убедились в мягкости предупреждения. Даже легкие предупреждения могут скрывать последующие предупреждения, указывающие на реальную опасность.

Примеры

Пример 1. Заголовочный файл стороннего производителя. Библиотечный заголовочный файл, который вы не можете изменить, может содержать конструкцию, которая приводит к (вероятно, мелкому) предупреждению. В таком случае “заверните” этот файл в свой собственный, который будет включать исходный при помощи директивы `#include` и избирательно отключать для него конкретные предупреждения. В своем проекте вы будете использовать собственный заголовочный файл, а не исходный. Например (учтите — управление выводом предупреждений варьируется от компилятора к компилятору):

```
// Файл: myproj/my_lambda.h – "обертка" для lambda.hpp из
// библиотеки Boost. Всегда включайте именно этот файл и не
// используйте lambda.hpp непосредственно. Boost.Lambda
// приводит к выводу компилятором предупреждений, о
// безвредности которых нам доподлинно известно. Когда
// разработчики сделают новую версию, которая не будет
// вызывать предупреждений, мы удалим из этого файла
// соответствующие директивы #pragma, но сам заголовочный
// файл останется.
//
#pragma warning(push) // Отключение предупреждений только
                      // для данного заголовочного файла
#pragma warning(disable:4512)
#pragma warning(disable:4180)
```

```
#include <boost/lambda/lambda.hpp>
#pragma warning(pop)  // Восстанавливаем исходный уровень
                      // вывода предупреждений
```

Пример 2. "Неиспользуемый параметр функции". Убедитесь, что вы в самом деле сознательно не используете параметр функции (Например, это "заглушка" для будущего расширения или требуемая стандартом часть сигнатуры, которую ваш код не использует). Если этот параметр вам действительно не нужен, просто удалите его имя:

```
// ... внутри пользовательского распределителя подсказка не
// используется ...

// Предупреждение: "неиспользуемый параметр 'localityHint'"
pointer allocate( size_type numObjects,
                  const void *localityHint = 0 ) {
    return static_cast<pointer>(
            mallocShared( numObjects * sizeof(T) ) );
}

// Новая версия: предупреждение устранено
pointer allocate( size_type numObjects,
                  const void * /* localityHint */ = 0 ) {
    return static_cast<pointer>(
            mallocShared( numObjects * sizeof(T) ) );
}
```

Пример 3. "Переменная определена, но не используется". Убедитесь, что вы действительно не намерены обращаться к данной переменной (к таким предупреждениям часто приводят локальные объекты, следующие идиоме "выделение ресурса есть инициализация", см. рекомендацию 13). Если обращение к объекту действительно не требуется, часто можно заставить компилятор замолчать, включив "вычисление" самой переменной в качестве выражения (такое вычисление не влияет на скорость работы программы):

```
// Предупреждение: "переменная 'lock' определена, но не
// используется"
void Fun() {
    Lock lock;
    // ...
}

// Новая версия: предупреждение не должно выводиться
void Fun() {
    Lock lock;
    lock;
    // ...
}
```

Пример 4. "Переменная может использоваться, не будучи инициализированной". Инициализируйте переменную (см. рекомендацию 19).

Пример 5. "Отсутствует return". Иногда компиляторы требуют наличия инструкции `return` несмотря на то, что поток управления не может достичь конца функции (например, при наличии бесконечного цикла, инструкции `throw`, других инструкций `return`). Такое предупреждение не стоит игнорировать, поскольку вы можете *только думать*, что управление не достигает конца функции. Например, конструкция `switch`, у которой нет выбора `default`, при внесении изменений в программу может привести к неприятностям, так что следует иметь выбор `default`, который просто выполняет `assert(false)` (см. также рекомендации 68 и 90):

```
// Предупреждение: отсутствующий "return"
int Fun( Color c ) {
    switch( c ) {
        case Red: return 2;
        case Green: return 0;
```

```
        case Blue:
        case Black: return 1;
    }
}

// Новая версия: предупреждение устранено
int Fun( Color c ) {
    switch( c ) {
        case Red: return 2;
        case Green: return 0;
        case Blue:
        case Black: return 1;
        // Значение !"string" равно false:
        default: assert(!"should never get here!");
                 return -1;
    }
}
```

Пример 6. "Несоответствие `signed/unsigned`". Обычно не возникает необходимость сравнивать или присваивать числа с разным типом знаковости. Измените типы сравниваемых переменных так, чтобы они соответствовали друг другу. В крайнем случае, воспользуйтесь явным преобразованием типов. (Компилятор все равно вставляет в код преобразование типов и предупреждает именно об этом, так что лучше сделать то же самостоятельно.)

Исключения

Иногда компилятор может выдавать утомительные, а порой и просто ложные предупреждения, но у вас нет способа их устранения (или такой способ заключается в нереальной или непроизводительной переделке текста программы). В таких редких случаях по решению всей команды разработчиков можно пойти на отключение конкретных мелких предупреждений, которые на самом деле не несут никакой особой информации и являются не более, чем результатом чрезмерной осторожности компилятора. Такие предупреждения можно отключить, но только данные конкретные предупреждения, а не все, максимально локализовав при этом область отключения и сопроводив ее ясным и подробным комментарием, почему такой шаг необходим.

Ссылки

[Meyers97] §48 • [Stroustrup94] §2.6.2

2. Используйте автоматические системы сборки программ

Резюме

Нажимайте на одну (единственную) кнопку: используйте полностью автоматизированные ("в одно действие") системы, которые собирают весь проект без вмешательства пользователя.

Обсуждение

Процесс сборки программы "в одно действие" очень важен. Он должен давать надежный и воспроизводимый результат трансляции ваших исходных файлов в распространяемый пакет. Имеется богатый выбор таких автоматизированных инструментов сборки, так что нет никакого оправдания тому, что вы их не используете. Выберите один из них и применяйте его в своей работе.

Мы встречались с организациями, где подобное требование игнорировалось. Некоторые полагают, что настоящий процесс сборки должен состоять в том, чтобы пощелкать мышью там и сям, запустить пару утилит для регистрации серверов COM/CORBA и вручную скопировать несколько файлов. Вряд ли у вас есть лишнее время и энергия, чтобы растрачивать их на то, что машина сделает быстрее и лучше вас. Вам нужна надежная автоматизированная система сборки программы "в одно действие".

Успешная сборка должна происходить молча, без каких бы то ни было предупреждений (см. рекомендацию 1). Идеальный процесс сборки должен выдать только одно журнальное сообщение: "Сборка успешно завершена".

Есть две модели сборки — инкрементная и полная. При инкрементной сборке компилируются только те файлы, которые претерпели изменения со времени последней инкрементной или полной сборки. Следствие: вторая из двух последовательных инкрементных сборок не должна перезаписывать никакие файлы. Если она это делает — по всей видимости, у вас в проекте имеется циклическая зависимость (см. рекомендацию 22), либо ваша система сборки выполняет ненужные операции (например, создает фиктивные временные файлы, чтобы затем просто удалить их).

Проект может иметь несколько видов полной сборки. Рассмотрите возможность параметризации процесса сборки при помощи таких важных параметров, как целевая архитектура, отладочная или коммерческая версии, вид пакета (программа-инсталлятор или просто набор файлов). Одни установки сборки могут давать только наиболее существенные исполнимые и библиотечные файлы, другие — добавлять к ним вспомогательные файлы, а окончательная сборка — создавать программу-инсталлятор, которая включает в себя все ваши файлы, файлы сторонних производителей и код инсталляции.

Со временем размер проекта обычно возрастает, растет и стоимость отказа от автоматизированной системы сборки. Если вы не используете ее с самого начала, вы теряете время и ресурсы. Все равно со временем потребность в такой системе станет непреодолимой, но при этом вы окажетесь под гораздо большим давлением, чем в начале проекта.

В больших проектах возможна даже специальная должность "хозяина сборки", который отвечает за работу этой системы.

Ссылки

[Brooks95] §13, §19 • [Dewhurst03] §1 • [GnuMake] • [Stroustrup00] §9.1

3. Используйте систему контроля версий

Резюме

Как гласит китайская пословица, плохие чернила лучше хорошей памяти: используйте системы управления версиями. Не оставляйте файлы без присмотра на долгий срок. Проверяйте их всякий раз после того, как обновленные модули проходят тесты на работоспособность. Убедитесь, что внесенные обновления не препятствуют корректной сборке программы.

Обсуждение

Практически все нетривиальные проекты требуют командной работы и/или более недели рабочего времени. В таких проектах вы будете просто *вынуждены* сравнивать различные версии одного и того же файла для выяснения того, когда (и/или кем) были внесены те или иные изменения. Вы будете *вынуждены* контролировать и руководить внесением изменений в исходные файлы проекта.

Когда над проектом работают несколько разработчиков, они вносят изменения в проект параллельно, возможно, одновременно в разные части одного и того же файла. Вам нужен инструмент, который бы автоматизировал работу с разными версиями файлов и, в определенных случаях, позволял объединять одновременно внесенные изменения. Система управления версиями (version control system, VCS) автоматизирует все необходимые действия, причем выполняя их более быстро и корректно, чем вы бы могли сделать это вручную. И вряд ли у вас есть лишнее время на игры в администратора — у вас хватает и своей работы по разработке программного обеспечения.

Даже единственный программист нередко вынужден выяснять, как и когда в программу проникла та или иная ошибка. VCS, автоматически отслеживая историю изменений каждого файла, позволяет вам "перевести стрелки часов назад" и ответить не только на вопрос, как именно выглядел файл раньше, но и когда это было.

Не портите сборку. Код, сохраненный VCS, всегда должен успешно собираться.

Огромное разнообразие инструментария данного типа не позволяет оправдать вас, если вы не используете одну из этих систем. Наиболее дешевой и популярной является `CVS` (см. ссылки). Это гибкий инструмент с возможностью обращения по TCP/IP, возможностью обеспечения повышенных мер безопасности (с использованием протокола `ssh`), возможностью администрирования с применением сценариев и даже графическим интерфейсом. Многие другие продукты VCS рассматривают `CVS` в качестве стандарта либо строят новую функциональность на ее основе.

Исключения

Проект, над которым работает один программист, причем не более недели, вероятно, в состоянии выжить и без применения VCS.

Ссылки

[BetterSCM] • [Brooks95] §11, §13 • [CVS]