1 C++: с большой силой приходит большая ответственность

В этой главе

- ✓ Суть происхождения ошибок при программировании на C++
- ✓ Четыре аналитические характеристики качественного кода
- ✓ Подход к обнаружению и исправлению ошибок программирования
- ✓ Мотивация, которая помогает взяться за непростую задачу исправления ошибок

Перед каждым разработчиком C++ стоит сложная задача: писать и поддерживать тысячи, а иногда и миллионы строк кода. К счастью, *сложное* не означает *невыполнимое*! Учитесь на собственных ошибках и, что еще важнее, на ошибках предшественников — и вы сможете справляться практически с любыми трудностями, которые встретятся вам в будущем. Сопровождение проектов на C++ — это длительный процесс, и вы не освоите язык или большую кодовую базу за несколько месяцев. Разочарование — постоянная проблема разработчиков на любом языке: часто оно возникает из-за предыдущих решений, оказавшихся сомнительными или неудачными. Ищите хорошие, умные и элегантные решения в чужом коде — и вскоре вы научитесь находить такие решения самостоятельно.

В этой книге мы разберем 100 ошибок, которые вы скорее всего обнаружите (или допустите сами) в современном и унаследованном C++-коде. Мы постарались

превратить эти ошибки в практические знания, полезные для создания и поддержки более чистой и эффективной кодовой базы. Понимание причин и способов исправления каждой ошибки поможет вам находить и устранять аналогичные проблемы в будущем — или, что еще лучше, не допускать их вообще. Давайте приступим!

1.1. Ошибки

Ошибки — обычное явление в любой кодовой базе, и причины их возникновения могут быть самыми разными. За годы работы C++-разработчики изучили бесчисленное множество приложений, пытаясь понять природу ошибок. Среди них — неправильные или устаревшие приемы программирования, несовершенство самого языка, слабый дизайн приложений и неэффективные инструменты. И для всего этого уже были найдены методы решения! Были разработаны методологии, которые помогают устранить сбои в процессе разработки, а для компенсации изъянов языка были добавлены новые функции.

Типичные кодовые базы на C++ почти всегда можно улучшить. Код, написанный год или десять лет назад, использует методы и языковые возможности, которые были популярны и доступны в то время. С течением времени появились новые подходы, и несоответствия между стилями и архитектурными решениями стали очевидны. Этот прогресс в знаниях и инструментах не всегда отражается в существующем коде: технический долг накапливается, а несоответствия множатся. Эти проблемы необходимо решать, осуществляя поддержку, модификацию, обновление и отладку кода.

Не все разработчики готовы перепроектировать или переписывать большие объемы кода. Бизнес-цели редко предполагают улучшение существующего кода, когда можно просто создать новый код или новые функции. Но технический долг, вызванный старыми практиками и неудачными решениями, влияет и на новые проекты. Он проявляется в хрупкости (ненадежности) кода — тот становится трудным для изменения и интеграции. Вместо подхода «или/или», при котором старый код либо исправляют, либо игнорируют, разумнее искать и устранять повторяющиеся ошибки и шаблоны.

1.1.1. Простой пример

В разделе «Ошибка #57» мы рассмотрим широкий спектр предупреждений, полученных при выполнении полной сборки продукта и анализе результатов. При изучении вывода компиляции некоторые предупреждения будут появляться неоднократно. Один из подходов — сосредоточиться на конкретном предупреждении компилятора и исправить ошибку во всей или большей части кодовой

базы. Следующий пример демонстрирует минимальный тестовый сценарий, в котором предупреждение об адресе `-Waddress` появляется тремя способами.

Код в листинге 1.1 некорректно пытается:

- вызвать функцию без синтаксически необходимых круглых скобок;
- проверить возвращаемое значение функции;
- сравнить два значения объектов.

Листинг 1.1. Пример кода с предупреждениями компилятора

```
int cleanup() {
    return 0;
}

int main() {
    cleanup;

    if (!(cleanup)) {
        …
    }
    const char* message = "Hello, world";
    if (message == "Hello, world") {
        …
    }
    return 0;
}
```

Попытка вызова функции без круглых скобок — это просто обращение к адресу, и никакой код не выполняется; исправляется добавлением круглых скобок

Попытка использовать возвращаемое значение. Используется адрес функции, который всегда истинный (ненулевой); при отрицании он становится всегда ложным (нулевым), и тело кода никогда не выполняется; исправляется добавлением круглых скобок

Сравнение одного константного указателя с другим константным указателем всегда ложно (адреса не совпадают); исправляется использованием функции типа strcmp или строк C++ для сравнения значений, на которые они указывают

В кодовой базе, где были обнаружены эти предупреждения, проблемы впоследствии устранили в ходе другой работы. Если выбран такой подход, сообщите остальным разработчикам и ревьюерам, что вы намерены улучшать код постепенно. Лучше заручиться поддержкой команды: некоторые исправления могут оказаться нетривиальными, хотя на первый взгляд выглядят простыми. Действуйте с осторожностью и проводите тщательное тестирование. Не пытайтесь решить слишком много задач одновременно (несмотря на соблазн) — лучше двигаться в устойчивом темпе, решая управляемое количество проблем.

Другой вариант — устранить несколько ошибок в пределах одного файла. Если вы уже работаете над модификацией кода, то легко выборочно исправить и другие проблемы. Как и прежде, следует действовать предусмотрительно.

1.2. Анатомия ошибки

Каждая ошибка, обсуждаемая в этой книге, рассматривается отдельно. В реальных условиях некоторые ошибки влияют на другой код, и такая взаимосвязь усложняет диагностику и решение проблемы. В книге мы сознательно

игнорируем эти зависимости, чтобы сосредоточиться на сути конкретных проблем. Это, конечно, создает риск упрощения и неполноты картины, но позволяет глубже рассмотреть природу отдельных ошибок. В большинстве глав мы будем обсуждать обобщенную концепцию ошибок, включенных в раздел. Некоторые ошибки повторяются в разных главах, поскольку затрагивают несколько аспектов языка. Это свидетельствует о перекрестном характере многих функций языка C++: функция или ошибка редко ограничивается узким контекстом. Каждая ошибка анализируется с точки зрения четырех ключевых характеристик, которые затрагивают разработку и выполнение.

1.2.1. Корректность

Одна из важнейших характеристик кода — *корректность*, то есть способность решать поставленную задачу без ошибок. В конце концов, код может быть полной катастрофой по другим меркам (стиль, эффективность), но, если он корректен, мы можем с уверенностью сказать, что задача решена. Даже самый элегантный, академически сложный и выглядящий привлекательно код прежде всего должен быть правильным. Возьмите себе за правило писать код так, чтобы он точно решал поставленную задачу и при этом не вызывал побочных эффектов. В реальности достичь этого удается не всегда, но именно к этому нужно стремиться.

ПРИМЕЧАНИЕ В некоторых случаях некорректный код может привести к неопределенному поведению (undefined behavior, UB). Относитесь к UB как к крайне оскорбительной вульгарности, которую нужно избегать любой ценой (включите здесь тему из фильма *Shaft*[1]). UB — это действия программы, не определенные стандартом языка. Они могут вызвать непредсказуемые результаты, сбои или уязвимости безопасности. Компилятор вправе обработать UB *любым* способом, включая оптимизацию, при которой соответствующий код полностью удаляется.

1.2.2. Читаемость

Вторая характеристика — *читаемость*, которая определяет то, насколько ясно разработчик доносит идеи своего кода до других. У каждого программиста есть свой стиль, но его следует подкорректировать, если он препятствует пониманию кода. Старайтесь следовать общепринятым или командным правилам оформления. Практика показывает, что писать код следует так, чтобы он был удобен другим разработчикам (компилятору это безразлично). Формулируйте свои мысли таким образом, чтобы любой коллега понял ход ваших мыслей однозначно и быстро, и соблюдайте все обязательные правила стиля. Более опытные

[1] Shaft — «Шафт»; культовый американский боевик 1971 года. Фраза используется как шутка, «нагнетание пафоса». — *Примеч. ред.*

разработчики стремятся к элегантности, и это естественно, но иногда чрезмерная «элегантность» делает код трудным для восприятия менее опытными коллегами.

1.2.3. Эффективность

Эффективность отражает то, насколько полно программист использует возможности языка для ускорения и упрощения разработки. В ее основе лежат компактность выражения и умелое использование определенных функций. Время, необходимое для решения задачи, напрямую влияет на продуктивность разработчика, а косвенно — на корректность кода. Общее правило простое: используйте правильные (корректные) функции и конструкции по прямому назначению.

1.2.4. Производительность

Наконец, *производительность* — это выбор наилучшего подхода, алгоритма, структуры данных или способа решения задачи, обеспечивающих оптимальную работу программы на конкретном оборудовании. Этот аспект сложнее всего правильно реализовать, поскольку интуиция не всегда верно подсказывает, что именно делает код наиболее производительным. Для выявления реальных узких мест и «горячих точек»[1] необходимо использовать инструменты профилирования.

Эта характеристика — наиболее академически строгая из всех четырех перечисленных. Выбор правильного алгоритма обычно является первым шагом к решению задачи. Для этого требуется определенное понимание принципов computer science — оценки вычислительных затрат и выбора оптимального способа решения.

1.3. Учимся на своих ошибках

Любой автор надеется, что он правильно определил потребности аудитории и восполнил пробелы в существующей литературе. Прочитав эту книгу, вы сможете распознавать описанные и прочие ошибки, поймете, почему они являются проблемой, узнаете, как их решить. Это поможет вам избегать подобных ошибок в будущем.

---

[1] Хот-спот (hotspot), или «горячая точка», — участок программы, на который приходится бóльшая часть выполняемых процессором инструкций или выполнение которого занимает значительное время (одни инструкции исполняются быстрее, а другие — медленнее). — *Примеч. пер.*

1.3.1. Как распознать ошибку

Осведомленность — это первый шаг к лучшему программированию. Иногда мои студенты смотрят на меня в полной растерянности, когда я показываю им примеры. Они стараются понять концепцию, но «прорыв» пока не наступил. Чтобы облегчить их страдания, я часто говорю, что не жду от них самостоятельного понимания. Я показываю и объясняю примеры, чтобы привлечь внимание к конкретным проблемам, научить распознавать их, а затем — понять, как их решить.

Мои собственные знания и опыт — результат работы тех, кто учил меня и делился примерами. Хотя я кое-что знаю и, может быть, кажусь умным, я (как и все) должен постоянно учиться. Распознавание типовых ошибок — важнейший элемент профессионального роста: как только мы замечаем знакомый паттерн, у нас сразу появляется инструмент для его решения. Эта книга поможет вам распознавать и устранять ошибки в существующем коде, а не множить их.

1.3.2. Суть ошибок

Распознать ошибку — это важный шаг; но чтобы понять ее природу, мы должны разобраться, почему она вообще является ошибкой. Некоторые практики, которые когда-то считались лучшими, сегодня уже устарели. Программирование развивается по научной модели: мы выдвигаем гипотезы, проверяем их, оцениваем их эффективность. Иногда мы оказываемся правы, а иногда — нет.

Никого не должно удивлять такое развитие событий — это естественный процесс обучения. Совершая ошибки, мы выявляем пробелы в своих знаниях (или убеждаемся, что их у нас нет). Заполнение этих пробелов затрагивает всё новые и новые области, расширяя наш кругозор и понимание взаимосвязей между элементами программирования. Я хочу донести до вас, что каждая ошибка может стать интеллектуальной и эмоциональной возможностью для роста. Настоящие неудачи случаются только тогда, когда мы сдаемся или отказываемся учиться. Эта книга поможет вам понять, почему тот или иной подход к написанию кода является ошибкой.

1.3.3. Исправление ошибок

Исправление ошибок — вот истинная причина появления этой книги. Хотя устранение проблем очень важно, оно является лишь одной частью более сложного процесса. Без понимания того, почему ошибка является проблемой, мы становимся механическими устройствами, модифицирующими исходный код (гипотетически этому можно обучить и обезьяну).

Мы хотим исправлять ошибки, потому что понимаем, в чем заключается их суть. Такое понимание особенно важно при работе в команде. Мы все совершаем

ошибки, но, если удается найти и устранить их до того, как они попадут в продакшен, это принесет пользу разработчикам, пользователям и компании. Эта книга поможет вам научиться не только самостоятельно исправлять, но и предотвращать ошибки при написании кода.

1.3.4. Учимся на ошибках

Обнаружение, понимание и исправление ошибок необходимо для получения хорошего кода, но из этого процесса можно извлечь еще больше пользы. По мере накопления опыта вам будет легче анализировать проблемный программный код. Распознавание и понимание определенных паттернов помогут выявить и другие ошибки. Это можно сравнить с тренировками в спортзале, где тело развивают для различных видов активности, не сосредоточиваясь на чем-то одном.

Знание этих ошибок сделает вас ценным источником информации для других — как во время разработки, так и на этапе проверки кода. Делясь своими знаниями, вы помогаете коллегам расти и развивать те же навыки.

Эта книга подготовит вас к тому, чтобы делиться опытом и способствовать развитию других.

1.4. Где мы находим ошибки

Любой опытный разработчик C++ скажет, что существует гораздо больше 100 типов ошибок, которые можно допустить при работе с этим языком. Я выбрал только те, с которыми сталкивался сам, которые изучал или которые оказали на меня влияние. Считайте этот список моим личным «топ-100» — другой автор, скорее всего, составил бы другой перечень. Ошибки условно разделены на несколько категорий.

1.4.1. Проектирование классов

C++ был разработан, чтобы привнести объектно-ориентированную парадигму в популярный язык C. Разработка классов — это процесс определения новых типов данных, которые компилятор воспринимает и обрабатывает так же, как встроенные типы. Разработчик полностью отвечает за то, что представляют собой новые классы, как они работают и какой смысл в них заложен.

Компилятор обеспечивает соблюдение синтаксиса и правил языка, но накладывает минимум ограничений на вновь создаваемые типы данных. Такая свобода становится отличной почвой для ошибок. Чтобы избежать их, разработчикам просто необходимо понимать и минимизировать эти проблемы, а не полагаться на ограничения языка.

1.4.2. Реализация программных решений

В программировании, как правило, есть несколько областей, где неудачное проектирование или реализация могут вызвать проблемы. C++ — гибкий язык, поэтому необходимо осознавать наличие таких зон риска, чтобы предотвращать ошибки. Некоторые из них напрямую связаны с проектированием классов, но, поскольку эти проблемы являются более общими, они отнесены к этому разделу. Корректное применение языковых возможностей имеет решающее значение для создания качественного программного обеспечения.

1.4.3. Проблемы библиотек

Код отдельного разработчика часто составляет лишь крошечную часть всей программы. Большинство приложений активно используют библиотеки и их функции. Кроме того, сам язык обладает инструментами для обобщенного решения проблем — например, шаблонами, позволяющими описывать целые семейства решений, оставляя компилятору генерацию конкретного кода.

Библиотечные функции предоставляют обширную функциональность, самостоятельное написание которой было бы утомительным, трудоемким и чреватым ошибками. Правильное использование этих функций позволяет разработчику писать более чистый, надежный код, сосредоточившись на прикладных задачах. Однако неправильное понимание принципов работы библиотек может привести к серьезным проблемам.

1.4.4. Современный C++

Одной из целей C++ всегда было обеспечение полной обратной совместимости по мере развития языка. Это позволяет современным компиляторам успешно собирать унаследованный код. Но совместимость не означает, что старый код написан хорошо. С развитием языка часть прежних проблем была устранена, а более удачные решения стали частью стандарта.

Многие нововведения позволяют программисту создавать более корректный, устойчивый и простой код, выполняющий те же задачи, что и раньше. Часто эти улучшения затрагивают один или несколько аспектов ошибок: корректность, читаемость, эффективность и производительность. Цена, которую мы платим за эти усовершенствования, — необходимость изучать новые возможности языка и понимать, как они могут положительно влиять на наше мышление и процесс программирования.

1.4.5. Старые стандарты и способы их применения

Код, написанный до появления новых возможностей языка, очевидно, не сможет воспользоваться их преимуществами. А код, созданный уже после добавления

этих возможностей, может просто их не использовать. В результате программа становится функционально устаревшей, менее выразительной и потенциально более подверженной ошибкам. Например, строки в стиле C++ устраняют множество проблем, связанных со строками в стиле C. Но чтобы получить эти преимущества и избежать типичных ошибок, разработчик должен начать применять именно строки C++.

1.4.6. Утраченный опыт и искаженное обучение

Опытных разработчиков всегда будут сменять молодые специалисты. Этим программистам приходится разбираться со старым кодом — часто сложным, громоздким и неочевидным. Ситуацию усугубляет то, что во многих академических курсах по-прежнему недостаточно внимания уделяется современному C++. Есть немало современных учебников по C++, которые все еще содержат устаревшие примеры и рекомендации, не учитывающие лучшие практики разработки.

Некоторые ошибки возникают из-за применения древних подходов, другие — из-за плохого преподавания, а третьи — из-за того, что бывшие когда-то передовыми методы больше не работают. Независимо от природы этих ошибок, знакомство с ними и их последствиями, а также с возможными способами их устранения поможет каждому разработчику C++ создавать корректный, читаемый, эффективный и производительный код.

1.5. Несколько слов о структуре книги

Книга построена в обратном хронологическом порядке. Сначала рассматриваются современные проблемы C++. Затем — переходные темы, затрагивающие как современные (C++11 и более поздние версии), так и предыдущие стандарты (C++98 и C++03). Наконец, внимание уделяется более ранним версиям C++, где многие унаследованные кодовые базы ограничены в использовании современных методологий.

Код в этой книге в основном базируется на версиях, принятых до появления стандарта C++11 (приятная смесь C++98 и C++03). Для компиляции кода использовался набор инструментов GNU C++ toolset (версия 11.3.0), запущенный в операционной системе Ubuntu (22.04 LTS под Windows Subsystem for Linux [WSL] 2). Эта версия компилятора была выбрана за простоту и стабильность работы на указанной платформе. Более поздние версии уже есть и продолжают выходить. Любая из них может обнаружить проблемы и недостатки кода, о которых говорится в этой книге. Использование более старых версий компилятора может привести к иным результатам.

Бизнес-ограничения, вероятно, заставят некоторых читателей собирать код в режиме C++98, и тем самым они упустят все замечательные возможности

современных функций языка. Печально, но факт: значительная часть моей кодовой базы ограничена C++03, поскольку работа требует стабильности. Заставить команду разработчиков операционной системы и инструментальных средств проверить более поздние версии оказалось не так просто.

Хотя многие из рассматриваемых далее ошибок встречаются в классическом C++ (до C++11), первые главы посвящены современному C++. Эти темы особенно полезны практикующему программисту, поэтому они размещены в начале.

Далее в книге решения и примеры приводятся с использованием только классического C++. Некоторые из этих проблем встречаются и в более современном коде, но новые методы следует использовать там, где это возможно. Позже мы обсудим ситуации, когда программист не может использовать современные компиляторы — увы, но такое случается. Знание того, как решать эти задачи в условиях ограниченных технологий, по-прежнему важно. Обратите внимание, что раздел «Смотри также» в конце каждого пункта «Ошибка» может содержать перекрестные ссылки как на соответствующие современные, так и на классические ошибки и решения, если это необходимо.

Итоги

Эта книга поможет вам:

- понять причины распространенных ошибок программирования на C++;
- выявлять типичные ошибки, встречающиеся во многих существующих кодовых базах C++;
- анализировать код на предмет корректности, читаемости, эффективности и производительности;
- определять, где и как вносить изменения, чтобы устранить эти ошибки;
- писать более качественный код, избегая этих ошибок и используя современные подходы, когда это возможно;
- делиться своими знаниями с другими разработчиками, помогая им распознавать и устранять аналогичные проблемы.

Часть 1

Современный C++

Ключевой аспект современного C++ — строгий подход к проектированию надежных классов и управлению типами данных. Классы в C++ давно вышли за рамки простых контейнеров и стали фундаментальными компонентами надежной программной инфраструктуры. Эффективное управление динамической памятью с помощью умных указателей позволяет минимизировать утечки и избежать появления «висячих» указателей. Применяя передовые методы проектирования классов и используя новые возможности с типами, разработчики могут создавать отказоустойчивые и адаптируемые структуры, прокладывая путь к более надежным приложениям.

Современный C++ предлагает целый набор инструментов, которые делают код лаконичным и точным: лямбда-выражения, цикл по диапазону и контекстные ключевые слова. Распознавание и интеграция этих пока еще недостаточно используемых функций в повседневную практику может значительно сократить количество ошибок и повысить эффективность кода. Переход от традиционных методов к современным позволяет разработчикам с большей уверенностью и креативностью справляться с задачами разработки, создавая корректный и устойчивый к будущим изменениям код.