Оглавление

Предисловие

Вы купили эту книгу, чтобы поднять свои навыки программирования на новый уровень. И это правильно, потому что вам, безусловно, пригодятся излагаемые в ней практические знания. Если у вас имеется большой опыт программирования на C, то вы в деталях узнаете, как принимаются хорошие проектные решения и какие у них есть плюсы и минусы. Если вы только начинаете знакомиться с C, то найдете здесь руководство по принятию решений и на примерах кода увидите, как эти решения применяются для построения больших программ.

В книге есть ответы на вопросы о том, как структурировать C-программу, как обрабатывать ошибки и как проектировать гибкие интерфейсы. Когда вы больше узнаёте о программировании на C, начинают возникать разные вопросы, например:

- следует ли возвращать имеющуюся информацию об ошибке?
- следует ли использовать для этой цели глобальную переменную `errno`?
- что лучше: немного функций с большим числом параметров или наоборот?
- как построить гибкий интерфейс?
- как реализовать базовые вещи, например итератор?

Для объектно ориентированных языков на большую часть этих вопросов почти исчерпывающий ответ дает книга «банды четырех»: *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides «Design Patterns: Elements of Reusable Object-Oriented Software»*[1]. Паттерны проектирования дают программисту проверенные опытом рекомендации, как должны взаимодействовать между собой объекты и как они связаны отношением владения. Кроме того, они показывают, как следует группировать объекты.

Однако на процедурных языках типа C большинство этих паттернов проектирования невозможно реализовать так, как описано «бандой четырех». В C нет встроенных объектно ориентированных механизмов. Наследование или полиморфизм можно эмулировать, но это не лучшее решение, потому что такой код будет непонятен программистам, привыкшим к программированию на C, но не владеющим программированием на объектно ориентированных языках типа C++ и незнакомым с использованием таких концепций, как наследование и полиморфизм. Такие программисты хотели бы придерживаться стиля программирования на C, к которому привыкли. Однако к нему применимы не все объектно ориентированные рекомендации или, по крайней мере, конкретная реализация идеи паттерна проектирования не годится для не объектно ориентированного языка.

[1] *Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес.* «Паттерны объектно ориентированного проектирования». Питер, 2022.

Итак, ситуация выглядит следующим образом: мы хотим писать на C, но не можем напрямую использовать большую часть знаний, документированных в виде паттернов проектирования. В этой книге показано, как преодолеть разрыв и практически реализовать эти знания на языке программирования C.

Зачем я написал эту книгу

Теперь я хочу рассказать, почему знания, собранные в этой книге, оказались столь важными для меня и почему их так трудно отыскать.

В школе я изучал C в качестве первого языка программирования. Как и любой начинающий программист на C, я удивлялся, почему нумерация элементов массива начинается с 0, и наугад пытался поместить операторы `*` и `&` в нужное место, чтобы заставить работать магию указателей.

В университете я узнал, как в действительности работают синтаксические конструкции C и как они транслируются в аппаратные биты и байты. Вооруженный этими знаниями, я смог писать небольшие программы, которые работали очень хорошо. Но я по-прежнему не понимал, почему более длинный код выглядит именно так, а не иначе, и, уж конечно, не мог сам придумать решения вроде:

```
typedef struct INTERNAL_DRIVER_STRUCT* DRIVER_HANDLE;
typedef void (*DriverSend_FP)(char byte);
typedef char (*DriverReceive_FP)();
typedef void (*DriverIOCTL_FP)(int ioctl, void* context);

struct DriverFunctions
{
  DriverSend_FP fpSend;
  DriverReceive_FP fpReceive;
  DriverIOCTL_FP fpIOCTL;
};

DRIVER_HANDLE driverCreate(void* initArg, struct DriverFunctions f);
void driverDestroy(DRIVER_HANDLE h);
void sendByte(DRIVER_HANDLE h, char byte);
char receiveByte(DRIVER_HANDLE h);
void driverIOCTL(DRIVER_HANDLE h, int ioctl, void* context);
```

При изучении этого кода возникает много вопросов:

- зачем нужны указатели на функции в `struct`?
- зачем функциям нужен этот `DRIVER_HANDLE`?
- что такое IOCTL и почему бы не написать вместо этого отдельные функции?
- зачем нужны явные функции создания и уничтожения?

Эти вопросы появились, когда я начал писать производственные приложения.

Я то и дело сталкивался с ситуациями, когда понимал, что мне не хватает знаний о C; например, как реализовать итератор или как обрабатывать ошибки в функциях. Я осознавал, что синтаксис-то я освоил, но понятия не имею, как им правильно воспользоваться. Я пытался чего-то добиться, но все получалось коряво или не получалось вовсе. Мне были необходимы рекомендации, показывающие, как решать конкретные задачи на языке C. Например:

- как проще всего захватывать и освобождать ресурсы?
- можно ли использовать `goto` для обработки ошибок?
- следует ли сразу проектировать интерфейс гибким или лучше изменять его, когда возникнет необходимость?
- следует ли использовать макрос `assert`, или нужно возвращать код ошибки?
- как реализовать итератор на C?

Для меня оказалось неожиданным открытием, что, хотя у моих опытных коллег было много различных ответов на эти вопросы, никто не смог направить меня туда, где такие проектные решения были документированы вместе с описанием их плюсов и минусов.

Поэтому я обратился к интернету и снова испытал удивление: оказалось очень трудно найти убедительные ответы на эти вопросы, хотя язык C существует уже не один десяток лет. Я обнаружил, что, несмотря на изобилие литературы по основам и синтаксису языка C, нет почти ничего о продвинутом программировании и о том, как писать на C красивый код, который выдержит испытание производственным приложением.

И вот тут в игру вступает эта книга. Она поможет вам отточить свои навыки программирования на C и перейти от простеньких программок к большим системам, в которых ошибки обрабатываются должным образом и которые обладают достаточной гибкостью, чтобы быть готовыми к будущим изменениям требований и проекта. В этой книге используется концепция паттернов проектирования, чтобы познакомить вас со всеми шагами принятия решений и оценкой их достоинств и недостатков. Эти паттерны применяются к сквозным примерам когда, чтобы показать, как код эволюционирует и почему принимает именно такую, а не иную конечную форму.

Основы паттернов

Рекомендации по проектированию в этой книге приводятся в форме паттернов. Идея представлять знания и передовые практики в виде паттернов исходит от архитектора Кристофера Александра, который высказал ее в книге «The Timeless Way of Building» (Oxford University Press, 1979). Он использует небольшие проверенные временем фрагменты для решения важнейшей проблемы в своей области: как проектировать и возводить города. Подход на основе паттернов переняли разработчики программного обеспечения, и теперь проводятся конференции типа Pattern Languages of Programs (PLoP), имеющие целью расширить наши знания о паттернах. В особенности книга «банды четы-

рех» «Design Patterns: Elements of Reusable Object-Oriented Software» (Prentice Hall, 1997) оказала значительное влияние и познакомила разработчиков ПО с концепцией паттернов проектирования.

Но что же такое паттерн? Определений много, и, если эта тема вас сильно интересует, почитайте книгу Frank Buschmann et al. «Pattern-Oriented Software Architecture: On Patterns and Pattern Languages» (Wiley, 2007), где приведены точные описания и детали. А для наших целей достаточно считать, что паттерн дает проверенное временем решение какой-то практической задачи. Представленные в этой книге паттерны имеют структуру, описанную в табл. P.1.

Таблица P.1. Структура паттернов, представленных в этой книге

Часть паттерна	Описание
Название	Это легко запоминаемое имя паттерна. Предполагается, что программисты будут использовать его в повседневном общении (как в случае паттернов из книги «банды четырех», когда можно услышать, например, такую фразу: «И абстрактная фабрика создает объект»). Названия паттернов в этой книге начинаются с заглавной буквы
Контекст	Определяет обстановку, в которой действует паттерн. Говорит, при каких условиях паттерн можно применить
Проблема	Сообщает информацию о задаче, которую мы хотим решить. Эта часть начинается с основной постановки, выделенной полужирным шрифтом, после чего детально описывается, почему данную проблему трудно решить. (В других форматах паттернов детали вынесены в отдельную часть, называемую «Движущие силы».)
Решение	В этой части приводятся рекомендации по решению проблемы. В начале формулируется основная идея решения, выделенная полужирным шрифтом, а затем следуют детали. Также включен пример кода, содержащий конкретную реализацию
Последствия	В этом разделе перечисляются плюсы и минусы описанного решения. Применяя паттерн, вы должны быть уверены, что последствия вас устраивают
Известные примеры использования	Примеры использования убеждают в том, что предложенное решение действительно работает в реальных приложениях. Кроме того, они дают конкретные примеры, помогающие понять, как применяется паттерн

Основное преимущество представления рекомендаций в виде паттернов заключается в том, что паттерны можно применять один за другим. Для крупной проблемы проектирования трудно найти конкретную рекомендацию и конкретное решение именно этой проблемы. Вместо этого большая и весьма специфическая проблема разбивается на много меньших и более общих проблем, а затем эти проблемы решаются по очереди путем применения раз-

личных паттернов. Мы просто сравниваем ситуацию с описанием паттерна и применяем тот паттерн, который отвечает проблеме и имеет устраивающие нас последствия. Эти последствия могут порождать новую проблему, которая решается применением другого паттерна. Таким образом, мы проектируем программу постепенно, не стараясь сразу выложить на стол полный проект еще до того, как написана первая строчка кода.

Как читать эту книгу

Вы должны быть знакомы с основами программирования на C. Вы должны знать синтаксис и семантику C – например, эта книга не расскажет вам о том, что такое указатель и как им пользоваться. Приводятся рекомендации только по вопросам более высокого порядка.

Главы книги независимы. Вы можете читать их в произвольном порядке или выбирать только те, которые вас интересуют. В следующем разделе приведен краткий обзор всех паттернов, что позволит перейти сразу к тем, что вам интересны. Так что если вы точно знаете, что ищете, то можете начать отсюда.

Если вы не ищете какой-то конкретный паттерн, а хотите получить общее представление о проектировании программ на C, то прочитайте часть I от начала до конца. Каждая глава этой части посвящена одной теме, начиная с таких простых, как обработка ошибок и управление памятью, и кончая такими более продвинутыми и специальными, как проектирование интерфейсов или платформенно независимого кода. В каждой главе описываются относящиеся к ее теме паттерны и сквозной пример кода, демонстрирующий их применение.

Во второй части книги приведено два больших примера, иллюстрирующих применение многих паттернов из первой части. Здесь вы увидите, как большая программа строится с использованием паттернов.

Краткий обзор паттернов

В табл. P.2–P.10 перечислены все паттерны. Каждая строка содержит краткое описание проблемы, после которого идет ключевое слово «Поэтому» и краткое описание решения.

Таблица P.2. Паттерны для обработки ошибок

Название паттерна	Краткое описание
Разбиение функции	На функции лежит несколько обязанностей, что затрудняет ее чтение и сопровождение. Поэтому разбейте ее на части. Выделите часть функции, которая кажется полезной сама по себе, создайте из нее новую функцию и вызовите ее
Проверка условий	Функцию трудно читать и сопровождать, потому что проверка предусловий совмещена в ней с основной логикой. Поэтому сначала проверьте выполнение обязательных предусловий и сразу же верните управление, если они не выполняются

Название паттерна	Краткое описание
Принцип самурая	Возвращая информацию об ошибке, вы предполагаете, что вызывающая сторона проверит эту информацию. Однако она может попросту опустить проверку, и ошибка останется незамеченной. Поэтому либо возвращайте управление, если все хорошо, либо не возвращайте вовсе. Если ошибку невозможно обработать, то аварийно завершайте программу
Переход к обработке ошибки	Код становится трудно читать, если он захватывает и освобождает несколько ресурсов в разных точках функции. Поэтому соберите все освобождение ресурсов и обработку ошибок в конце функции. Если ресурс невозможно захватить, то используйте предложение `goto` для перехода в точку освобождения ресурсов
Запись об очистке	Трудно сделать кусок кода удобным для чтения и сопровождения, если он захватывает и освобождает несколько ресурсов, особенно когда эти ресурсы зависят друг от друга. Поэтому после успешного вызова функций захвата ресурсов запоминайте, какие функции требуется вызвать для очистки. И вызывайте те, которые были зарегистрированы
Объектная обработка ошибок	Наличие нескольких обязанностей у одной функции, например захват ресурса, освобождение ресурса и его использование, затрудняет реализацию, чтение, сопровождение и тестирование функции. Поэтому поместите инициализацию и очистку в разные функции по аналогии с идеей конструкторов и деструкторов в объектно ориентированном программировании

Таблица P.3. Паттерны для возврата информации об ошибке

Название паттерна	Краткое описание
Возврат кода состояния	Вам нужен механизм возврата информации о состоянии вызывающей стороне, чтобы та могла на нее отреагировать. Механизм должен быть простым в использовании, а вызывающая сторона не должна путать разные ошибки. Поэтому используйте возвращаемое значение функции для возврата информации о состоянии. Возвращайте значение, представляющее конкретное состояние. Вызывающая и вызываемая сторона должны одинаково интерпретировать возвращаемые значения
Возврат существенной информации об ошибке	С одной стороны, вызывающая сторона должна иметь возможность реагировать на ошибки, а с другой стороны, чем больше информации об ошибке вы возвращаете, тем длиннее становится ваш код и код для обработки ошибки. Поэтому возвращайте только ту информацию об ошибке, которая существенна для вызывающей стороны. Информация существенна, если вызывающая сторона может на нее отреагировать

Название паттерна	Краткое описание
Специальное возвращаемое значение	Вы хотите вернуть информацию об ошибке, но явно возвращать коды состояния не годится, потому что тогда нельзя использовать возвращаемое функцией значение для возврата других данных. Если использовать выходные параметры, то вызывать функцию станет труднее. Поэтому используйте возвращаемое значение для возврата вычисленных функцией данных, но зарезервируйте одно или несколько специальных значений на случай ошибки
Протоколирование ошибок	Вы хотите быть уверены, что в случае ошибки сумеете легко определить ее причину. Но не хотите ради этого усложнять код обработки ошибок. Поэтому используйте разные каналы: один для предоставления информации об ошибке, существенной для вызывающей стороны, а другой для предоставления информации, существенной для разработчика. Например, записывайте отладочную информацию об ошибке в файл журнала и не возвращайте ее вызывающей стороне

Таблица P.4. Паттерны для управления памятью

Название паттерна	Краткое описание
Сначала стек	Любому программисту часто приходится принимать решение о классе хранения и области памяти (стек, куча…) для размещения переменных. Если для каждой переменной по новой взвешивать все плюсы и минусы различных вариантов, то ни на что другое не останется времени. Поэтому по умолчанию размещайте переменные в стеке, это даст возможность воспользоваться механизмом автоматической очистки памяти
Вечная память	Хранить большие объемы данных и передавать их между вызовами функций трудно, потому что нужно гарантировать, что памяти для данных достаточно и что она не стирается между вызовами. Поэтому размещайте данные в памяти, которая остается доступной в течение всего времени работы программы
Отложенная очистка	Необходимость в динамической памяти возникает, если нужна память большого и заранее неизвестного размера. Но проблема очистки динамической памяти трудна и является источником многих ошибок. Поэтому только выделяйте динамическую память, а ее освобождение оставьте операционной системе в момент завершения программы
Единоличное владение	За удобство динамической памяти приходится расплачиваться необходимостью ее освобождения. В больших программах трудно гарантировать, что вся динамически выделенная память надлежащим образом освобождается. Поэтому уже в момент выделения памяти ясно и недвусмысленно определите, где она должна быть освобождена и кто за это отвечает

Название паттерна	Краткое описание
Обертка выделения	Любое выделение динамической памяти может завершиться неудачно, поэтому следует проверять результат выделения в своем коде и реагировать соответственно. Это громоздко, потому что такие проверки приходится делать во многих местах программы. Поэтому оберните вызовы функций выделения и освобождения памяти и реализуйте логику обработки ошибок или дополнительного управления памятью в этих обертках
Проверка указателя	Программные ошибки, связанные с доступом по недействительному указателю, ведут к неопределенному поведению программы, и отлаживать их трудно. Но поскольку код часто работает с указателями, велики шансы появления таких ошибок. Поэтому явно делайте недействительными неинициализированные или освобожденные указатели и всегда проверяйте действительность перед доступом по ним
Пул памяти	Частое выделение и освобождение памяти из кучи приводит к фрагментации памяти. Поэтому выделите большой участок памяти на все время работы программы. Во время выполнения получайте блоки фиксированного размера из этого пула памяти, а не непосредственно из кучи

Таблица P.5. Паттерны для возврата данных из C-функций

Название паттерна	Краткое описание
Возвращаемое значение	Части, на которые вы хотите разбить функцию, не являются независимыми. Как обычно бывает в процедурном программировании, одна часть производит результат, необходимый другой части. Части разбиваемой функции должны разделять какие-то данные. Поэтому используйте механизм C, предназначенный для получения информации о результате вызова функции: возвращаемое значение. Механизм возврата данных в C копирует результат функции и дает вызывающей стороне доступ к копии
Выходные параметры	Язык C поддерживает возврат только одного значения из функции, и вернуть несколько элементов информации затруднительно. Поэтому возвращайте все данные с помощью единственного вызова функции, эмулируя передачу аргументов по ссылке с помощью указателей
Агрегат	Язык C поддерживает возврат только одного значения из функции, и вернуть несколько элементов информации затруднительно. Поэтому поместите все данные в специально определенный тип. Пусть этот агрегат содержит все связанные данные, которые вы хотите использовать сообща. Определите этот тип в интерфейсе своего компонента, так чтобы вызывающая сторона могла обращаться ко всем данным, хранящимся в экземпляре агрегата

Название паттерна	Краткое описание
Неизменяемый экземпляр	Вы хотите передать информацию, хранящуюся в больших порциях неизменяемых данных, из своего компонента вызывающей стороне. Поэтому разместите экземпляр (например, `struct`), содержащий разделяемые данные, в статической памяти. Предоставляйте эти данные нуждающимся в них пользователям, но организуйте дело так, чтобы они не могли изменить данные
Буфер, принадлежащий вызывающей стороне	Вы хотите передать сложные или большие данные известного размера вызывающей стороне, и эти данные не являются неизменяемыми (т. е. могут быть изменены во время выполнения). Поэтому потребуйте, чтобы вызывающая сторона предоставила буфер и его размер функции, возвращающей данные. Внутри функции скопируйте данные в буфер при условии, что его размер достаточен
Вызываемая сторона выделяет память	Вы хотите передать сложные или большие данные известного размера вызывающей стороне, и эти данные не являются неизменяемыми (т. е. могут быть изменены во время выполнения). Поэтому выделите буфер нужного размера в самой функции, возвращающей данные. Скопируйте данные в буфер и верните указатель на этот буфер

Таблица P.6. Паттерны, относящиеся ко времени жизни данных и владению ими

Название паттерна	Краткое описание
Программный модуль без состояния	Вы хотите предоставить логически связанную функциональность вызывающей стороне и максимально упростить ее использование. Поэтому делайте функции простыми и не храните информацию о состоянии в реализации. Поместите все связанные функции в один заголовочный файл и предоставьте вызывающей стороне этот интерфейс к своему программному модулю
Программный модуль с глобальным состоянием	Вы хотите структурировать логически связанный код, нуждающийся в общей информации о состоянии, и максимально упростить его использование. Поэтому заведите один глобальный экземпляр, чтобы все связанные функции могли разделять его. Поместите все функции, работающие с этим экземпляром, в один заголовочный файл и предоставьте вызывающей стороне этот интерфейс к своему программному модулю
Экземпляр, принадлежащий вызывающей стороне	Вы хотите предоставить нескольким вызывающим сторонам или потокам доступ к некоторой функциональности с помощью функций, которые зависят друг от друга. При этом взаимодействие вызывающей стороны с вашими функциями приводит к образованию информации о состоянии. Поэтому потребуйте, чтобы вызывающая сторона передавала вашим функциям экземпляр, используемый для хранения ресурса и информации о состоянии. Предоставьте явные функции для создания и уничтожения таких экземпляров, чтобы вызывающая сторона могла контролировать время их существования

Название паттерна	Краткое описание
Разделяемый экземпляр	Вы хотите предоставить нескольким вызывающим сторонам или потокам доступ к некоторой функциональности с помощью функций, которые зависят друг от друга. При этом взаимодействие вызывающей стороны с вашими функциями приводит к образованию информации о состоянии, которую все вызывающие стороны хотят разделять. Поэтому потребуйте, чтобы вызывающая сторона передавала вашим функциям экземпляр, используемый для хранения ресурса и информации о состоянии. Используйте один и тот же экземпляр для нескольких вызывающих сторон и оставьте владение этим экземпляром за своим программным модулем

Таблица P.7. Паттерны для создания гибких API

Название паттерна	Краткое описание
Заголовочные файлы	Вы хотите, чтобы реализованная вами функциональность была доступна коду, реализованному другими лицами, но при этом хотите скрыть детали реализации от вызывающей стороны. Поэтому включите в свой API объявления функций, реализующих функциональность, предоставляемую пользователям. Скройте все внутренние функции, внутренние данные и определения (реализации) функций в файлах реализации и не передавайте эти файлы пользователям
Описатель	Вам нужно разделить информацию о состоянии или работать с разделяемыми ресурсами в реализациях своих функций, но вы не хотите, чтобы вызывающая сторона видела эту информацию и разделяемые ресурсы. Поэтому заведите функцию, создающую контекст, с которым будет работать вызывающая сторона, и возвращающую абстрактный указатель на внутренние данные в этом контексте. Потребуйте, чтобы вызывающая сторона передавала этот указатель всем вашим функциям, которые смогут тогда воспользоваться внутренними данными для хранения информации о состоянии и ресурсов
Динамический интерфейс	Должна быть возможность вызывать реализации с немного различающимся поведением, но при этом не хотелось бы дублировать код, даже код управления логикой реализации и объявления интерфейса. Поэтому определите общий интерфейс для различающейся функциональности в своем API и потребуйте, чтобы вызывающая сторона предоставила функцию обратного вызова для варианта этой функциональности, которую вы сможете вызвать из реализации своей функции

Название паттерна	Краткое описание
Управление функцией	Вы хотите вызывать реализации с немного различающимся поведением, но не хотите дублировать код, даже код управления логикой реализации и объявления интерфейса. Поэтому добавьте в функцию параметр, в котором функции передается метаинформация об этом вызове, определяющая, какая именно функциональность требуется

Таблица P.8. Паттерны для создания гибких интерфейсов итератора

Название паттерна	Краткое описание
Доступ по индексу	Вы хотите, чтобы пользователь мог удобно перебирать элементы вашей структуры данных, при этом нужно сохранить возможность изменения внутреннего устройства этой структуры, не изменяя пользовательский код. Поэтому предоставьте функцию, которая принимает индекс для адресации элемента в вашей структуре данных и возвращает его содержимое. Пользователь вызывает эту функцию в цикле для обхода всех элементов
Курсор	Вы хотите предоставить пользователю такой интерфейс итератора, который был бы устойчив относительно изменения элементов в процессе итерирования и который позволил бы изменять впоследствии структуру данных, не внося изменений в пользовательский код. Поэтому создайте экземпляр курсора, указывающий на элемент структуры данных. Функция итерирования принимает этот экземпляр итератора в качестве аргумента, получает элемент, на который указывает итератор, и модифицирует экземпляр итератора, так чтобы он указывал на следующий элемент. Пользователь в цикле вызывает эту функцию, чтоб получать элементы по одному
Итератор обратного вызова	Вы хотите предоставить устойчивый интерфейс итерирования, который не требовал бы от пользователя реализации цикла для обхода всех элементов и позволял бы в будущем вносить изменения в структуру данных, не изменяя пользовательский код. Поэтому воспользуйтесь своей уже существующей структурой данных и реализованными вами операциями для обхода всех элементов в ней и вызывайте предоставленную пользователем функцию для каждого элемента в процессе этого обхода. Эта пользовательская функция принимает содержимое элемента в качестве параметра и выполняет операции над этим элементом. Чтобы начать итерирование, пользователь вызывает всего одну функцию, а весь обход производится внутри вашей реализации

Таблица P.9. Паттерны для организации файлов в модульных программах

Название паттерна	Краткое описание
Охрана включения	Включить один и тот же заголовочный файл легко, но это приведет к ошибкам компиляции, если в файле имеются определения типов или макросы определенного вида, поскольку в процессе компиляции они будут определены повторно. Поэтому защищайте содержимое заголовочных файлов от повторного включения, чтобы разработчику, пользующемуся ими, не нужно было думать, сколько раз включен файл. Для этого можно использовать директиву `#ifdef` или `#pragma once`
Каталоги программных модулей	Разнесение кода по нескольким файлам увеличивает количество файлов в кодовой базе. Если хранить все файлы в одном каталоге, то становится трудно обозревать их, особенно когда кодовая база велика. Поэтому помещайте тесно связанные заголовочные файлы и файлы реализации в один каталог. Назовите этот каталог, так чтобы имя отражало функциональность, предоставляемую заголовочными файлами
Глобальный каталог include	Чтобы включить файлы из других программных модулей, приходится использовать относительные пути вида *`../othersoftwaremodule/file.h`*. Вы должны знать точное местоположение других заголовочных файлов. Поэтому заведите в кодовой базе один глобальный каталог, содержащий API всех программных модулей. Добавьте этот каталог в глобальный список путей к включаемым файлам, поддерживаемый вашим инструментарием
Автономные компоненты	Структура каталогов не позволяет увидеть зависимости в коде. Любой программный модуль может просто включить заголовочные файлы из любого другого программного модуля, поэтому проверить зависимости с помощью компилятора невозможно. Поэтому идентифицируйте программные модули, которые содержат схожую функциональность и должны развертываться вместе. Поместите эти модули в общий каталог и заведите подкаталог для тех заголовочных файлов, которые нужны вызывающей стороне
Копия API	Вы хотите разрабатывать, присваивать номера версий и развертывать части кодовой базы независимо друг от друга. Однако для этого необходимо иметь четко определенные интерфейсы между частями кода и возможность хранить этот код в различных репозиториях. Поэтому, чтобы использовать функциональность другого компонента, скопируйте его API. Отдельно соберите этот компонент и скопируйте артефакты сборки и его публичные заголовочные файлы. Поместите эти файлы в каталог внутри своего компонента и сконфигурируйте этот каталог как путь к глобальному include

Таблица P.10. Паттерны, позволяющие избежать ада `#ifdef`

Название паттерна	Краткое описание
Избегание вариантов	Использование разных функций для каждой платформы затрудняет чтение и написание кода. От программиста требуется понимать, правильно использовать и тестировать эти многочисленные функции, чтобы обеспечить одинаковую функциональность на нескольких платформах. Поэтому пользуйтесь стандартизованными функциями, имеющимися на всех платформах. Если стандартизованных функций нет, то подумайте, не стоит ли отказаться от соответствующей функциональности
Изолированные примитивы	Варианты кода, организованные с помощью директив `#ifdef`, делают код нечитаемым. Очень трудно следить за потоком программы, который реализован несколько раз для разных платформ. Поэтому изолируйте свои варианты кода. В файле реализации поместите многовариантный код в отдельные функции и вызывайте эти функции из основной программы, которая таким образом будет содержать только платформенно независимый код
Атомарные примитивы	Функцию, которая содержит варианты и вызывается из основной программы, все равно трудно понять, потому что код, содержащий многочисленные `#ifdef`, был помещен туда только для того, чтобы избавиться от него в основной программе, но проще он от этого не стал. Поэтому делайте примитивы атомарными. В каждой функции обрабатывайте только один вариант. Если, например, нужно обработать несколько вариантов операционных систем и оборудования, то заведите для каждого свою функцию
Уровень абстракции	Вы хотите использовать функциональность, которая отвечает за платформенные варианты в нескольких местах кодовой базы, но не хотите дублировать код этой функциональности. Поэтому предоставьте API для каждого вида функциональности, требующего платформенно зависимого кода. В заголовочном файле объявляйте только платформенно независимые функции, а весь платформенно зависимый код, заключенный внутри `#ifdef`, помещайте в файл реализации. Вызывающая сторона должна будет включить только ваш заголовочный файл, но не платформенно зависимые файлы
Разделение реализаций вариантов	Платформенно зависимые реализации по-прежнему содержат директивы `#ifdef`, чтобы различать варианты кода. Из-за этого трудно понять, какую часть кода следует собирать для какой платформы. Поэтому помещайте реализацию каждого варианта в отдельный файл и пофайлово выбирайте, что хотите компилировать для какой платформы

Графические выделения

В книге применяются следующие графические выделения.

Курсив

Новые термины, URL-адреса, адреса электронной почты, имена и расширения файлов.

Полужирный

Выделение формулировки проблемы и ее решения для каждого паттерна.

`Моноширинный`

Листинги программ, а также элементы кода в основном тексте: имена переменных и функций, базы данных, типы данных, переменные окружения, предложения и ключевые слова языка.

Так обозначается замечание общего характера.

Так обозначается предупреждение или предостережение.

О примерах кода

Примеры кода в этой книге представляют собой короткие фрагменты, акцентированные на какой-то одной идее, с целью продемонстрировать паттерны и их применение. Сами по себе эти фрагменты не компилируются, потому что некоторые вещи в них опущены (в частности, заголовочные файлы). Полный код, который компилируется, можно скачать с GitHub по адресу https://github.com/christopher-preschern/fluent-c.

Вопросы технического характера, а также замечания по примерам кода следует отправлять по адресу bookquestions@oreilly.com.

Эта книга призвана помогать вам в работе. Поэтому вы можете использовать приведенный в ней код в собственных программах и в документации. Спрашивать у нас разрешения необязательно, если только вы не собираетесь воспроизводить значительную часть кода. Например, никто не возбраняет включить в свою программу несколько фрагментов кода из книги. Однако для продажи или распространения примеров из книг издательства O'Reilly разрешение требуется. Цитировать книгу и примеры в ответах на вопросы можно без ограничений. Но для включения значительных объемов кода в документацию по собственному продукту нужно получить разрешение.

Мы высоко ценим, хотя и не требуем ссылки на наши издания. В ссылке обычно указывается название книги, имя автора, издательство и ISBN, например: «Fluent C by Christopher Preschern (O'Reilly). Copyright 2023 Christopher Preschern, 978-1-492-09733-4».

Если вы полагаете, что планируемое использование кода выходит за рамки изложенной выше лицензии, пожалуйста, обратитесь к нам по адресу **permissions@oreilly.com**.

Все паттерны в этой книги взяты из существующего кода, в котором они применяются. В списке ниже приведены ссылки на эти примеры кода:

- игра NetHack (https://oreil.ly/nxO5w);
- проект OpenWrt (https://oreil.ly/qeppo);
- библиотека OpenSSL (https://oreil.ly/zzsMO);
- сетевой анализатор Wireshark (https://oreil.ly/M55B5);
- портлендский репозиторий паттернов (https://oreil.ly/wkZzb);
- система управления версиями Git (https://oreil.ly/7F9Oz);
- переносимая среда исполнения Apache (https://oreil.ly/ysaM6);
- веб-сервер Apache (https://oreil.ly/W6SMn);
- операционная система B&R Automation Runtime (проприетарный закрытый код компании B&R Industrial Automation GmbH);
- визуальный редактор системы автоматизации B&R Visual Components, проприетарный закрытый код компании B&R Industrial Automation GmbH);
- система управления данными NetDRMS (https://oreil.ly/eR0EV);
- платформа программирования и численных расчетов MATLAB (https://oreil.ly/UpvJK);
- библиотека GLib (https://oreil.ly/QoUwT);
- веб-анализатор реального времени GoAccess (https://oreil.ly/L1Eij);
- программа физических расчетов Cloudy (https://oreil.ly/phLBb);
- собрание компиляторов GNU (GCC) (https://oreil.ly/KK4jY);
- система баз данных MySQL (https://oreil.ly/YKXxs);
- диспетчер памяти ION для Android (https://oreil.ly/2JV7h);
- Windows API (https://oreil.ly/nnzyX);
- Apple Cocoa API (https://oreil.ly/sQual);
- операционная система реального времени VxWorks (https://oreil.ly/UMUaj);
- текстовый редактор sam (https://oreil.ly/k3SQl);
- функции из стандартной библиотеки C: реализация glibc (https://oreil.ly/9Qr95);
- проект Subversion (https://oreil.ly/8Yz5R);
- инструмент исследования сети Nmap (https://oreil.ly/sg9sz);
- монитор производительности в реальном времени и система визуализации Netdata (https://oreil.ly/1sDZz);

- файловая система OpenZFS (https://oreil.ly/VWeQL);
- каркас обратной разработки Radare (https://oreil.ly/TUYfh);
- цифровые обучающие программы Education First (https://www.ef.com);
- текстовый редактор VIM (https:/gitub.com/vim/vim);
- программа построения графиков GNUplot (https://oreil.ly/PIQPj);
- движок базы данных SQLite (https://oreil.ly/5Knfz);
- программа сжатия данных gzip (https://oreil.ly/it40Z);
- веб-сервер lighttpd (https://github.com/lighttpd);
- начальный загрузчик U-Boot (https://oreil.ly/IKVYV);
- система моделирования дискретных событий (https://oreil.ly/NJnCH);
- платформа Nokia Maemo (https://oreil.ly/RwDtt).

Как с нами связаться

Вопросы и замечания по поводу этой книги отправляйте в издательство:

O'Reilly Media, Inc.
1005 Gravenstein Highway North Sebastopol, CA 95472
800-998-9938 (в США и Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс).

Для этой книги имеется веб-страница, на которой публикуются списки замеченных ошибок, примеры и прочая дополнительная информация. Адрес страницы: https://www.oreil.ly/fluent-c.

Замечания и вопросы технического характера следует отправлять по адресу bookquestions@oreilly.com.

Дополнительную информацию о наших книгах, конференциях и новостях вы можете найти на нашем сайте по адресу http://www.oreilly.com.

Ищите нас в LinkedIn: https://linkedin.com/company/oreilly-media.

Следите за нашей лентой в Twitter: http://twitter.com/oreillymedia.

Смотрите нас на YouTube: http://www.youtube.com/oreillymedia.

Благодарности

Я хочу поблагодарить свою жену Силке, которая теперь даже знает, что такое паттерны :-), и свою дочь Илви. Обе они делают меня счастливее и обе следят за тем, чтобы я не сидел за компьютером все время, а наслаждался жизнью.

Эта книга не увидела бы свет без помощи многих энтузиастов паттернов. Я благодарен всем участникам семинара Writers' Workshops на Европейской конференции по языкам паттернов в программах за отзывы о паттернах. В частности, я хочу выразить благодарность следующим лицам, которые дали очень полезные отзывы во время так называемого пастырского процесса на этой конференции, среди них Яри Раухамяки, Тобиас Раутер, Андреа Холлер,

Джеймс Коплиен, Уве Здун, Томас Разер, Иден Бэртон, Клаудиус Линк, Валентино Вранич и Сумит Калра. Отдельное спасибо моим коллегам по работе, в особенности Томасу Гавловецу, который проследил за тем, чтобы все детали программирования на C в моих паттернах были правильны. Роберт Ханмер, Майкл Вейсс, Дэвид Гриффитс и Томас Круг потратили немало времени на рецензирование этой книги и поделились со мной мыслями о том, как сделать ее лучше, – большое вам спасибо! Также я признателен всему коллективу издательства O'Reilly, помогавшему мне в работе над этой книгой. Особенно хочу поблагодарить редактора-консультанта Корбина Коллинза и выпускающего редактора Джонатона Оуэна.

Текст этой книги основан на следующих статьях, которые были приняты на Европейской конференции по языкам паттернов в программах и опубликованы в изданиях ACM. Эти статьи можно скачать бесплатно на сайте http://www.preschern.com.

- «A Pattern Story About C Programming», EuroPLoP '21: 26th European Conference on Pattern Languages of Programs, July 2015, article no. 53, 1–10, https://dl.acm.org/doi/10.1145/3489449.3489978.
- «Patterns for Organizing Files in Modular C Programs», EuroPLoP '20: Proceedings of the European Conference on Pattern Languages of Programs, July 2020, article no. 1, 1–15, https://dl.acm.org/doi/10.1145/3424771.3424772.
- «Patterns to Escape the #ifdef Hell», EuroPLop '19: Proceedings of the 24th European Conference on Pattern Languages of Programs, July 2019, article no. 2, 1–12, https://dl.acm.org/doi/10.1145/3361149.3361151.
- «Patterns for Returning Error Information in C», EuroPLop '19: Proceedings of the 24th European Conference on Pattern Languages of Programs, July 2019, article no. 3, 1–14, https://dl.acm.org/doi/10.1145/3361149.3361152.
- «Patterns for Returning Data from C Functions», EuroPLop '19: Proceedings of the 24th European Conference on Pattern Languages of Programs, July 2019, article no. 37, 1–13, https://dl.acm.org/doi/10.1145/3361149.3361188.
- «C Patterns on Data Lifetime and Ownership», EuroPLop '19: Proceedings of the 24th European Conference on Pattern Languages of Programs, July 2019, article no. 36, 1–13, https://dl.acm.org/doi/10.1145/3361149.3361187.
- «Patterns for C Iterator Interfaces», EuroPLoP '17: Proceedings of the 22nd European Conference on Pattern Languages of Programs, July 2017, article no. 8, 1–14, https://dl.acm.org/doi/10.1145/3147704.3147714.
- «API Patterns in C», EuroPlop '16: Proceedings of the 21st European Conference on Pattern Languages of Programs, July 2016, article no. 7, 1–11, https://dl.acm.org/doi/10.1145/3011784.3011791.
- «Idioms for Error Handling in C», EuroPLoP '15: Proceedings of the 20th European Conference on Pattern Languages of Programs, July 2015, article no. 53, 1–10, https://dl.acm.org/doi/10.1145/2855321.2855377.