Решение сложных задач на C++

87 головоломных задач с решениями

Герб Саттер

Издательский дом “Вильямс”
Москва • Санкт-Петербург • Киев
2008

Exceptional C++

87 New Engineering Puzzles, Programming Problems, and Solutions

Herb Sutter

ADDISON–WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico • City

Решение сложных задач на C++

ББК 32.973.26-018.2.75
С21
УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *А.В. Слепцов*

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
115419, Москва, а/я 783; 031150, Киев, а/я 152

Саттер, Герб.

С21 Решение сложных задач на C++. Серия *C++ In-Depth*: Пер. с англ. — М. : Издательский дом "Вильямс", 2008. — 400 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-0352-5 (рус.)

В данном издании объединены две широко известные профессионалам в области программирования на языке C++ книги Герба Саттера *Exceptional C++* и *More Exceptional C++*, входящие в серию книг *C++ In-Depth*, редактором которой является Бьерн Страуструп, создатель языка C++. Материал этой книги составляют переработанные задачи серии *Guru of the Week*, рассчитанные на читателя с достаточно глубоким знанием языка C++, однако данная книга будет полезна каждому, кто хочет углубить свои знания в этой области.

ББК 32.973.26-018.2.75

ISBN 978-5-8459-0352-5 (рус.)
ISBN 0-201-77581-6 (англ.)

Оглавление

Содержание

От редактора

Книга, которую вы держите в руках, не нуждается в представлении. Кому из серьезных программистов на C++ не известен Web-узел *Guru of the Week* и его автор Герб Саттер? На основе представленных на этом Web-узле материалов Саттер издал две книги — *Exceptional C++* и *More Exceptional C++*, и в настоящее время работает над очередной книгой этой серии.

В связи с тем, что книга *More Exceptional C++* по сути представляет собой продолжение *Exceptional C++*, было принято решение объединить эти книги при издании на русском языке в одну. Благодаря четкому разделению материала книг по темам и практически идентичному стилю книг в результате получилось не механическое объединение двух книг под одной обложкой, а единая книга, вобравшая в себя опыт множества программистов высочайшего уровня.

Изложение материала в виде задач и их решений позволяет не просто получить теоретические знания, но и тут же закрепить их путем применения на практике. Строгое разделение материала книги по темам позволяет не просто прочесть ее один раз, но и использовать ее как настольную книгу, в которой всегда можно найти подсказку о том, как решать проблемы, возникающие перед вами в процессе работы над реальными проектами.

Следует сказать несколько слов об особенностях перевода книги на русский язык. Постоянно растущая сложность языка программирования C++ вносит свои коррективы в связанную с ним терминологию. Об особенностях русскоязычной терминологии C++, использованной при работе над данной книгой, достаточно полно рассказано в предисловии к русскому изданию книги [Stroustrup97]. В частности, это касается перевода термина *statement* как *инструкция*, а не как *оператор*, который в C++ имеет свое четко определенное значение. Впрочем, для серьезного программиста, на которого и рассчитана данная книга, смысл того или иного перевода очевиден из контекста.

В заключение хотелось бы поблагодарить Герба Саттера не только за замечательную книгу, но и за ряд пояснений и комментариев, полученных от него в процессе работы над русским изданием. Особую благодарность за помощь и советы при работе над переводом книги редакция выражает Александру Кротову (NIXU Ltd).

Предисловия

Это — замечательная книга, но только заканчивая ее читать, я понял, до какой степени она замечательна. Возможно, это первая книга, написанная для тех, кто хорошо знаком с C++ — *всем* C++. От базовых возможностей языка до компонентов стандартной библиотеки и современных технологий программирования — эта книга ведет нас от задачи к задаче, заставляя все время быть начеку и акцентируя все наше внимание, — как и реальные программы на C++. Здесь перемешано все — проектирование классов, поведение виртуальных функций, зависимости компиляции, операторы присваивания, безопасность исключений... Здесь все, как в реальных программах на C++. В книге водоворот из возможностей языка, библиотечных компонент, технологий программирования — водоворот, который завораживает и притягивает. Так же, как и реальные программы на C++...

Когда я сравниваю свои решения задач из книги с ответами автора, я очень часто вижу, что в очередной раз попал в подготовленную ловушку — гораздо чаще, чем мне того хотелось бы. Некоторые могут сказать, что это доказывает лишь, что я не так хорошо знаю C++. Другие сочтут это показателем того, насколько сложен C++, и что никто не в состоянии стать настоящим мастером в программировании на C++. Я же считаю, что это всего лишь указатель на то, что, программируя на C++, надо всегда быть предельно внимательным и иметь ясное представление о том, что именно ты делаешь. C++ — мощный язык, созданный для решения различных задач, и при его использовании очень важно все время оттачивать свое знание языка, его библиотеки и идиом программирования. Широта изложенного материала в данной книге поможет вам в этом.

Читатели-ветераны групп новостей, посвященных C++, знают, насколько трудно оказаться объявленным “Гуру недели” (Guru of the Week). Ветераны-участники знают об этом еще лучше. В Internet, само собой, каждую неделю может быть только один гуру, но, снабженные информацией этой книги, вы можете рассчитывать на то, что каждая ваша программа будет иметь качество, говорящее о том, что над ней работал настоящий гуру.

Скотт Мейерс (Scott Meyers), июнь 1999 г.

Как стать экспертом? Ответ один и тот же для любой области человеческой деятельности.

1. Изучить основы.
2. Изучать этот же материал снова, но в этот раз сконцентрироваться на деталях, важность которых вы не могли осознать во время первого чтения.

Если вы обратите внимание на нужные детали и подробности и овладеете соответствующими знаниями, вы существенно приблизитесь к уровню эксперта. Но, пока вы еще не эксперт, как определить, на какие именно детали следует обратить особое внимание? Вы гораздо быстрее поднимите свой уровень и получите гораздо большее удовлетворение от изучения, если найдется кто-то, кто сможет выбрать эти детали для вас.

В качестве примера я хочу рассказать, как я однажды познакомился с владельцем небольшой фотостудии Фредом Пикером (Fred Picker). Он рассказал, что в фотографии есть две основные сложности: куда направить камеру и когда нажать спуск. Затем он потратил немало времени, рассказывая о таких технических деталях, как экспозиция, проявление и печать, — деталях, которые следует хорошо знать, чтобы иметь возможность со знанием дела сконцентрироваться на основных "сложностях".

В C++, как описано ниже, наиболее интересный способ изучения деталей — пытаться отвечать на вопросы о программах на C++, например такие.

- Одинаково ли действие "`f(a++);`" и "`f(a); ++a;`"?
- Можно ли использовать итератор для изменения содержимого контейнера `set`?
- Предположим, что вы используете `vector v`, который вырос до очень большого размера. Вы хотите очистить вектор и вернуть память системе. Поможет ли вам в этом вызов `v.clear()`?

Вы, вероятно, догадываетесь, что ответы на эти простые вопросы должны быть "нет", иначе зачем бы я вас об этом спрашивал? Но знаете ли вы, почему они отрицательны? Вы уверены?

Эта книга отвечает на все перечисленные и множество других не менее интересных вопросов о казалось бы простых программах. Имеется немного других книг, подобных этой. Большинство посвященных C++ книг, претендующих на звание книг "для профессионалов", либо посвящено узкоспециализированным темам (что неплохо, если вы хотите повысить свои знания и умения в этой конкретной области, но не совсем годится для получения более глубоких знаний для ежедневного применения), либо это "для профессионалов" указано просто для привлечения читателей.

Тщательно разобравшись с вопросами и ответами в этой книге, вам больше не придется задумываться над деталями написания ваших программ, и вы сможете полностью сконцентрироваться на решении стоящей перед вами задачи.

Эндрю Кёниг (Andrew Koenig), июнь 2001 г.

Введение

Греческий философ Сократ обучал своих учеников, задавая им вопросы, которые были разработаны таким образом, чтобы руководить учениками и помогать им сделать верные выводы из того, что они уже знают, а также показать им взаимосвязи в изучаемом материале и этого материала с другими знаниями. Этот метод обучения стал так знаменит, что сегодня мы называем его "методом Сократа". С точки зрения учащихся подход Сократа включает их в процесс обучения, заставляет думать и помогает связать и применить уже имеющиеся знания к новой информации.

Эта книга предполагает, что вам приходится заниматься написанием промышленного программного обеспечения на языке C++, и использует вопросы и ответы для обучения эффективному использованию стандарта C++ и его стандартной библиотеки, ориентируясь на разработку надежного программного обеспечения с использованием всех современных возможностей C++. Многие из рассмотренных в книге задач появились в результате работы автора и других программистов над своими программами. Цель книги — помочь читателю сделать верные выводы, как из хорошо известного ему материала, так и из только что изученного, и показать взаимосвязь между различными частями C++.

Данная книга не посвящена какому-то конкретному аспекту C++. Нельзя, однако, сказать, что она охватывает все детали C++ — для этого потребовалось бы слишком много книг, — но, тем не менее, в ней рассматривается широкая палитра возможностей C++ и стандартной библиотеки и, что немаловажно, показывается, как кажущиеся на первый взгляд несвязанными между собой вещи могут совместно использоваться для получения новых решений старых хорошо известных задач. Здесь вы обнаружите материал, посвященный шаблонам и пространствам имен, исключениям и наследованию, проектированию классов и шаблонам проектирования, обобщенному программированию и магии макросов, — и не просто винегрет из этих тем, а задачи, выявляющие взаимосвязь всех этих частей современного C++.

Большинство задач первоначально было опубликовано в Internet и некоторых журналах; в частности это расширенные версии первых 62 задач, которые можно найти на моем узле *Guru of the Week* [GotW], а также материалы, опубликованные мною в таких журналах, как *C/C++ User Journal*, *Dr. Dobb's Journal*, бывшем *C++ Report* и др.

Что следует знать для чтения этой книги

Я полагаю, что читатель уже хорошо знаком с основами C++. Если это не так, начните с хорошего введения и обзора по C++. Для этой цели могу порекомендовать вам такие книги, как [Stroustrup00], [Lippman98], а также [Meyers96] и [Meyers97].

Как читать данную книгу

Каждая задача в книге снабжена заголовком, выглядящим следующим образом.

Задача №. Название задачи ***Сложность: X***

Название задачи и уровень ее сложности подсказывают вам ее предназначение. Замечу, что уровень сложности задач — мое субъективное мнение о том, насколько

сложно будет решить ту или иную задачу большинству читателей, так что вы вполне можете обнаружить, что задача с уровнем сложности 7 решена вами гораздо быстрее, чем задача с уровнем сложности 5.

Чтение книги от начала до конца — неплохое решение, но вы не обязаны поступать именно так. Вы можете, например, читать только интересующие вас разделы книги. За исключением того, что я именую "мини-сериями" (связанные между собой задачи с одинаковым названием и подзаголовками "Часть 1", "Часть 2" и т.д.), задачи в книге практически независимы друг от друга, и вы можете совершенно спокойно читать их в любом порядке. Единственная подсказка: мини-серии лучше читать вместе; во всем остальном — выбор за вами.

В этой книге немало рекомендаций, в которых следующие слова имеют особое значение.

- **Всегда**. Это абсолютно необходимо; не пренебрегайте данной рекомендацией.
- **Предпочтительно**. Обычно лучше поступать так, как сказано. Поступать иначе можно только в тех случаях, когда ситуация настоятельно того требует.
- **Избегайте**. Обычно поступать так — не лучший способ, и это может оказаться опасно. Рассмотрите альтернативные способы достижения того же результата. Поступать так можно только в тех случаях, когда ситуация настоятельно того требует.
- **Никогда**. Это очень опасно, даже не думайте так поступать!

Соглашения, используемые в этой книге

В книге я даю определенные рекомендации читателям и не хотел бы давать рекомендации, которых не придерживаюсь сам. Сюда входит код примеров программ, приведенных в книге.

Если в коде примера вы видите директиву `using` в области видимости файла, а в соседней задаче — в области видимости функции, то причина этого всего лишь в том, что именно представляется мне более корректным (и эстетичным) в каждом конкретном случае.

При объявлении параметров шаблонов можно использовать как ключевое слово `class`, так и `typename`, — никакой функциональной разницы между ними нет. Однако поскольку я иногда сталкиваюсь с людьми, которые утверждают, что использовать `class` — это слишком устарело, я постарался почти везде использовать ключевое слово `typename`.

Все примеры кода — всего лишь отрывки программ, если не оговорено иное, и не следует ожидать, что эти отрывки будут корректно компилироваться при отсутствии остальных частей программы. Для этого вам придется самостоятельно дописывать недостающие части.

И последнее замечание — об URL: нет ничего более подвижного, чем Web, и более мучительного, чем давать ссылки на Web в печатной книге: зачастую эти ссылки устаревают еще до того, как книга попадает в типографию. Так что ко всем приведенным в книге ссылкам следует относиться критически, и в случае их некорректности — пишите мне. Дело в том, что все ссылки даны через мой Web-узел `www.gotw.ca`, и в случае некорректности какой-либо ссылки я просто обновлю перенаправление к новому местоположению страницы (если найду ее) или укажу, что таковой больше нет (если не смогу найти).

Благодарности

Я выражаю особую признательность редактору серии Бьярну Страуструпу (Bjarne Stroustrup), а также Дебби Лафферти (Debbie Lafferty), Марине Ланг (Marina Lang), Тирреллу Альбах (Tyrrell Albaugh), Чарльзу Ледди (Charles Leddy) и всем остальным из

команды Addison-Wesley за их помощь и настойчивость в работе над данным проектом. Трудно представить себе лучшую команду для работы над данной книгой; их энтузиазм и помощь помогли сделать эту книгу тем, чем она, я надеюсь, является.

Еще одна группа людей, заслуживающая особой благодарности, — это множество экспертов, чья критика и комментарии помогли сделать материал книги более полным, более удобочитаемым и более полезным. Особую благодарность я хотел бы выразить Бьярну Страуструпу (Bjarne Straustrup), Скотту Мейерсу (Scott Meyers), Андрею Александреску (Andrei Alexandrescu), Стиву Клэймэджу (Steve Clamage), Стиву Дьюхарсту (Steve Dewhurst), Кэю Хорстману (Cay Horstmann), Джиму Хайслопу (Jim Hyslop), Брендону Кехоу (Brendan Kehoe), Дэннису Манклу (Dennis Mancl), Яну Кристиану ван Винклю (Jan Christiaan van Winkel), Кэвлину Хенни (Kevlin Henney), Эндрю Кёнигу (Andrew Koenig), Патрику Мак-Киллену (Patric McKillen) и ряду других анонимных рецензентов. Все оставшиеся в книге ошибки, описки и неточности — только на моей совести.

И наконец, огромное спасибо моей семье и друзьям за то, что они всегда рядом, — как в процессе работы над этим проектом, так и в любое другое время.

Герб Саттер (Herb Sutter)

1

Обобщенное программирование и стандартная библиотека C++

Одна из наиболее мощных возможностей C++ — поддержка обобщенного программирования. Эта мощь находит непосредственное отражение в гибкости стандартной библиотеки C++, особенно в той ее части, в которой содержатся контейнеры, итераторы и алгоритмы, известной как библиотека стандартных шаблонов (standard template library, STL).

Этот раздел посвящен тому, как наилучшим образом использовать стандартную библиотеку C++, в частности STL. Когда и как лучше всего применять `std::vector` и `std::deque`? Какие ловушки подстерегают вас при использовании `std::map` и `std::set` и как благополучно их избежать? Почему на самом деле `std::remove()` ничего не удаляет?

Здесь вы познакомитесь с разными полезными технологиями программирования и разнообразными ловушками при написании своего собственного обобщенного кода, включая код, который работает с STL и расширяет ее. Предикаты какого типа безопасны при использовании совместно с STL, а какие нет, и почему? Какие методы можно использовать при написании кода мощного шаблона, поведение которого может изменяться в зависимости от возможностей типов, с которыми ему предстоит работать? Как можно просто переключаться между различными типами входных и выходных потоков? Как работает специализация и перегрузка шаблонов? И что означает это странное ключевое слово `typename`?

Все это и многое другое вы найдете в задачах, посвященных общим вопросам программирования и стандартной библиотеки C++.

Задача 1.1. Итераторы — *Сложность: 7*

Каждый программист, использующий стандартную библиотеку, должен быть знаком с распространенными (и не очень распространенными) ошибками при использовании итераторов. Сколько из них вы сумеете найти?

В приведенном коде имеется, как минимум, четыре ошибки, связанные с использованием итераторов. Сколько из них вы сможете обнаружить?

```
int main()
{
    vector<Date> e;
    copy(istream_iterator<Date>(cin),
         istream_iterator<Date>(),
         back_inserter(e));
    vector<Date>::iterator first =
        find(e.begin(), e.end(), "01/01/95");
```

```
    vector<Date>::iterator last =
        find(e.begin(), e.end(), "12/31/95");
    *last = "12/30/95";
    copy(first, last,
         ostream_iterator<Date>(cout,"\n"));
    e.insert(--e.end(), TodaysDate());
    copy(first, last,
         ostream_iterator<Date>(cout,"\n"));
}
```

Решение

```
int main()
{
    vector<Date> e;
    copy(istream_iterator<Date>(cin),
         istream_iterator<Date>(),
         back_inserter(e));
```

До сих пор все в порядке. Класс `Date` снабжен функцией с сигнатурой `operator>>(istream&, Date&)`, которую `istream_iterator<Date>` использует для чтения объектов `Date` из потока `cin`. Алгоритм `copy()` заносит эти объекты в вектор.

```
vector<Date>::iterator first =
    find(e.begin(), e.end(), "01/01/95");
vector<Date>::iterator last =
    find(e.begin(), e.end(), "12/31/95");
*last = "12/30/95";
```

Ошибка: это присваивание может быть некорректным, поскольку `last` может оказаться равным `e.end()` и, таким образом, не может быть разыменован.

Если значение не найдено, алгоритм `find()` возвращает свой второй аргумент (конечный итератор диапазона). В таком случае, если `"12/31/95"` не содержится в `e`, `last` окажется равным `e.end()`, который указывает на элемент за концом контейнера и, следовательно, корректным итератором не является.

```
copy(first, last,
     ostream_iterator<Date>(cout,"\n"));
```

Ошибка заключается в том, что `[first, last)` может не быть корректным диапазоном — на самом деле `first` может находиться после `last`. Например, если `"01/01/95"` не содержится в `e`, но зато в нем имеется `"12/31/95"`, то итератор `last` будет указывать на что-то внутри контейнера (точнее — на объект `Date`, равный `"12/31/95"`), в то время как итератор `first` — за его пределы (на позицию элемента, следующего за последним). Однако `copy()` требует, чтобы `first` указывал на позицию элемента, находящегося до элемента, на который в том же множестве указывает `last`, — т.е. `[first, last)` должен быть корректным диапазоном.

Если только вы не используете версию стандартной библиотеки, которая проверяет за вас наличие такого рода проблем, то наиболее вероятный симптом происшедшей ошибки — аварийный сброс сложного для диагностики образа памяти программы в процессе выполнения `copy()` или вскоре после него.

```
e.insert(--e.end(), TodaysDate());
```

Первая ошибка: выражение `--e.end()` вполне может оказаться неверным. Причина этого проста, если немного задуматься: в популярных реализациях STL `vector<Date>::iterator` зачастую представляет собой просто `Date*`, а язык C++ не позволяет изменять временные переменные встроенного типа. Например, следующий фрагмент кода неверен.

```
Date* f();    // Функция, возвращающая Date*
p = --f();     // Ошибка. Здесь можно использовать "f()-1"
```

К счастью, мы знаем, что `vector<Date>::iterator` — итератор с произвольным доступом, так что без потери эффективности можно записать более корректный вызов `e.insert()`.

```
e.insert(e.end() - 1, TodaysDate());
```

Теперь у нас появляется вторая ошибка, заключающаяся в том, что если контейнер `e` пуст, то любые попытки получить итератор, указывающий на позицию перед `e.end()` (то, чего мы пытались добиться записью "`--e.end()`" или "`e.end()-1`"), будут некорректны.

```
copy(first, last,
     ostream_iterator<Date>(cout,"\n"));
```

Здесь ошибка в том, что `first` и `last` могут больше не быть корректными итераторами. Вектор растет "кусками", чтобы при каждой вставке элемента в вектор не приходилось увеличивать буфер последнего. Однако иногда вектор все же заполняется, и перераспределение памяти становится совершенно необходимо.

В нашей ситуации при выполнении операции `e.insert()` вектор может вырасти (а может и нет), что по сути означает, что его память может оказаться перемещена (или нет). Из-за этой неопределенности мы должны рассматривать все существующие итераторы, указывающие на элементы контейнера, как более недействительные. В приведенном коде при реальном перемещении памяти некорректное копирование вновь приведет к аварийному сбросу образа памяти.

Рекомендация

Никогда не используйте некорректные итераторы.

Резюмируем: при использовании итераторов всегда задавайте себе четыре основных вопроса.

1. Корректность значений. Возможно ли разыменование данного итератора? Например, написание "`*e.end()`" всегда является ошибкой.
2. Корректность времени жизни объекта. Остался ли корректен используемый вами итератор после выполнения предыдущих действий?
3. Корректность диапазона. Представляет ли пара итераторов корректный диапазон? Действительно ли `first` указывает на позицию, располагающуюся до позиции (или равную ей), на которую указывает `last`? Указывают ли оба итератора на элементы одного и того же контейнера?
4. Некорректные встроенные операции. Не пытается ли код модифицировать временный объект встроенного типа (как в случае "`--e.end()`")? (К счастью, такого рода ошибки компилятор часто находит сам; кроме того, для итераторов, типы которых представляют собой классы, автор библиотеки для удобства может разрешить применение таких действий.)

Задача 1.2. Строки, нечувствительные к регистру. Часть 1 *Сложность: 7*

Вам нужен строковый класс, нечувствительный к регистру символов? Ваша задача — создать его.

Эта задача состоит из трех взаимосвязанных частей.

1. Что означает “нечувствительный к регистру”?
2. Напишите класс `ci_string`, идентичный стандартному классу `std::string`, но который является нечувствительным к регистру символов в том же смысле, что и распространенное расширение библиотеки C++ `stricmp()`[1]. Класс `ci_string`[2] должен использоваться следующим образом.

```
ci_string s("AbCdE");
// Нечувствительно к регистру
assert( s == "abcde" );
assert( s == "ABCDE" );
// Остается чувствительно к регистру
assert(strcmp(s.c_str(),"AbCdE") == 0);
assert(strcmp(s.c_str(),"abcde") != 0);
```

3. Разумно ли делать нечувствительность к регистру свойством объекта?

Решение

Вот ответы на поставленные вопросы.

1. Что означает “нечувствительный к регистру”?

Что это означает, в действительности полностью зависит от вашего приложения и языка. Например, многие языки вообще не поддерживают различные регистры символов. В ходе ответа на этот вопрос вам надо также определить, считаете ли вы акцентированные символы эквивалентными неакцентированным или нет (например, одинаковы ли символы в строке “aáâãä”), и рассмотреть многие другие вопросы. Данная задача представляет собой руководство, каким образом следует реализовать нечувствительность к регистру стандартных строк в любом смысле, который вы вложите в это понятие.

2. Напишите класс `ci_string`, идентичный стандартному классу `std::string`, но который является нечувствительным к регистру символов в том же смысле, что и распространенное расширение библиотеки C++ `stricmp()`.

Вопрос *как создать нечувствительную к регистру символов строку* настолько распространен, что, вероятно, заслуживает собственного FAQ[3] (чем, пожалуй, и является эта задача). Итак, вот что мы хотим получить.

```
ci_string s("AbCdE");
// Нечувствительно к регистру
assert( s == "abcde" );
assert( s == "ABCDE" );
// Остается чувствительно к регистру
assert(strcmp(s.c_str(),"AbCdE") == 0);
assert(strcmp(s.c_str(),"abcde") != 0);
```

Ключевым моментом здесь является понимание того, чем на самом деле является `string` в стандарте C++. Если вы заглянете внутрь заголовочного файла `string`, то обнаружите там нечто типа

```
typedef basic_string<char> string;
```

---

[1] Функция `stricmp()`, выполняющая сравнение строк без учета регистров символов, хотя и не является частью стандарта C или C++, но достаточно широко распространена в качестве расширения стандартной библиотеки.

[2] Префикс `ci` означает “case-insensitive”, т.е. нечувствительный к регистру. — *Прим. перев.*

[3] Frequently Asked Questions — дословно: часто задаваемые вопросы; общепринятая форма представления материала (в основном для новичков). — *Прим. перев.*