Оглавление

От коллектива переводчиков

В своих руках вы держите перевод очень интересной книги, впервые опубликованной почти десять лет назад. Её перевод начался тоже очень давно – почти семь лет назад, сначала в форме wiki, с помощью которого координировался процесс перевода и выкладывался переведенный текст. Несколько лет позже, когда была переведена большая часть этой книги, процесс замедлился, но от издательства «ДМК Пресс» поступило предложение издать перевод в бумажном виде с сохранением перевода в открытом виде. Результат вы держите в своих руках[1].

За время, прошедшее с момента начальной публикации оригинала, произошло множество событий – функциональное программирование становится мейнстримом, появляются новые языки, такие как Clojure[2], да и мир Common Lisp не стоит на месте: появляются библиотеки, улучшаются свободные реализации, такие как SBCL, и т.д. Lisp и его диалекты достаточно активно используются в коммерческом программировании: так, например, Google купил компанию ITA Software, которая использовала Common Lisp для разработки своих продуктов.

Информацию о Common Lisp можно найти и на русском языке. `http://lisper.ru/` – один из основных русскоязычных сайтов, посвященных Lisp. В «Русскую планету функционального программирования»[3] входит несколько блогов, авторы которых пишут о Common Lisp. Вы также можете интерактивно пообщаться с любителями Lisp в Jabber-конференции `lisp@conference.jabber.ru`.

Я хочу поблагодарить всех, кто принимал участие в подготовке данного издания – переводил, вычитывал и всячески помогал улучшать этот текст: Павла Алхимова, Стаса Бокарёва, Ивана Болдырева, Виталия Брагилевского, Александра Данилова, Дениса Дереку, Александра Дергачева, Сергея Дзюбина, Кирилла Горкунова, Алексея Замкового, Петра Зайкина, Илью Звягина, Евгения Зуева, Андрея Карташова, Виталия Каторгина, Сергея Катревича, Евгения Кирпичёва, Алекса Классика, Максима Колганова, Кирилла Коринского, Андрея Москвитина, Константина Моторного, Дмитрия Неверова, Дмитрия Панова, Сергея Раскина, Арсения Слободюка, Арсения Солокха, Михайло Сорочана, Илью Струкова, Александра Трусова, Валерия Федотова, Михаила Шеблаева, Михаила Шевчука, Вадима Шендера, Ивана Яни.

От коллектива переводчиков я бы хотел поблагодарить издательство «ДМК Пресс» за организацию официального оформления перевода и предоставление ресурсов, которые позволили сильно улучшить перевод.

Приятного чтения!

От коллектива переводчиков, Алекс Отт

---

[1] Если вы заметите ошибку или неточность в переводе, то пожалуйста, сообщите о ней на сайте проекта (`https://github.com/pcl-ru/pcl-ru`).

[2] Clojure – Lisp-подобный язык для JVM, ставший достаточно популярным

[3] http://fprog.ru/planet/.

Глава 1

Введение: почему Lisp?

Если вы считаете, что наибольшее удовольствие в программировании приносит написание лаконичных и выразительных программ, просто и прозрачно выражающих ваши мысли, тогда программирование на Common Lisp будет самым приятным из того, что вы можете делать на компьютере. Используя Common Lisp, вы будете успевать больше и быстрее, чем с практически любым другим языком.

Серьёзное заявление. Могу ли я доказать это? Да, но не на нескольких страницах введения. Вам придётся познакомиться с Lisp поближе и убедиться в этом самим, так что всё-таки придётся прочитать книгу до конца. Сначала я опишу свой путь к языку Lisp, а в следующей главе я объясню выгоды, которые вы получите от его изучения.

Я – один из немногих Lisp-хакеров во втором поколении. Мой отец начал заниматься компьютерами с написания на ассемблере операционной системы для машины, которую он использовал для сбора данных при подготовке своей докторской диссертации по физике. После работы с компьютерами в разных физических лабораториях, к 80-м, отец полностью оставил физику и стал работать в большой фармацевтической компании.

У этой компании был проект по созданию программы, моделирующей производственные процессы на химических заводах. Старая команда писала всё на языке FORTRAN, использовала половину бюджета и почти всё отведённое время, но не смогла продемонстрировать никаких результатов. Это было в 80-х, на пике развития искусственного интеллекта (ИИ), и Lisp так и витал в воздухе. Так что мой отец – в то время ещё не поклонник Lisp – пошёл в университет Карнеги-Меллона, чтобы пообщаться с людьми, работавшими над тем, что впоследствии стало языком Common Lisp, и узнать, сможет ли Lisp подойти для его проекта.

Ребята из университета показали ему кое-что из своих разработок, и это его убедило. Отец, в свою очередь, убедил своих боссов позволить ему взять провальный проект и переделать его на Lisp. Год спустя, используя остатки бюджета, команда отца представила работающее приложение, обладающее возможностями, на реализацию которых старая команда уже и не надеялась. Мой папа считает, что причина успеха – в решении использовать Lisp.

Однако это всего лишь первый эпизод. Может быть, мой отец ошибался в причине своего успеха. Или, может быть, Lisp был лучше других языков лишь для того времени. В настоящее время мы имеем кучу новых языков программирования, многие из которых переняли часть достоинств Lisp. Правда ли, что использование языка Lisp может дать вам те же выгоды, что и моему отцу в 80-х? Читайте дальше.

Несмотря на все усилия отца, я не изучал Lisp в университете. После учёбы, которая не содержала много программирования на каком-либо языке, я увлёкся Web и вернулся

к компьютерам. Сначала я писал на Perl, изучив его достаточно, чтобы создать форум для сайта журнала Mother Jones, после этого я работал над большими (по тем временам) сайтами, такими как, например, сайт компании Nike, запущенный к олимпийским играм 1996 года. После этого я перешёл на Java, будучи одним из первых разработчиков в WebLogic (теперь эта компания – часть BEA). После WebLogic я участвовал в другом стартапе, где был ведущим программистом по построению транзакционной системы обмена сообщениями на Java. Со временем я освоил как популярные языки – C, C++ и Python, так и менее известные: Smalltalk, Eiffel и Beta.

Итак, я знал два языка вдоль и поперёк и был поверхностно знаком с несколькими другими. В конечном счёте я понял, что источником моего интереса к языкам программирования была идея, заложенная рассказами отца о Lisp, – идея, что разные языки программирования существенно различаются, и, несмотря на формальное равенство всех языков по Тьюрингу, вы действительно можете быть гораздо продуктивнее, используя одни языки вместо других, и получать при этом больше удовольствия. Поэтому я начал учить Lisp в свободное время. Меня воодушевляло то, насколько быстро я проходил путь от идеи к работающему коду.

Например, в одном из отпусков, имея около недели свободного времени, я решил написать на Lisp программу для игры в Го с использованием генетических алгоритмов, так как писал её до этого на Java. Даже с моими зачаточными знаниями Common Lisp (приходилось искать самые базовые функции) я работал более продуктивно, чем если бы я писал это на Java, несмотря на несколько лет работы с ней.

Похожий эксперимент привёл к созданию библиотеки, о которой я расскажу в главе 24. В начале моей карьеры в WebLogic я написал библиотеку на Java для анализа class-файлов. Она работала, но код был запутан, и его трудно было изменить или добавить новую функциональность. В течение нескольких лет я пытался переписать библиотеку, думая, что смогу использовать мои новые знания в Java и не увязнуть в куче дублирующегося кода, но так и не смог. Когда же я попробовал переписать её на Common Lisp, это заняло всего 2 дня, и я получил не просто библиотеку для разбора class-файлов Java, но библиотеку для разбора любых двоичных файлов. Вы увидите, как она работает, в главе 24 и воспользуетесь ею в главе 25 для разбора тегов ID3 в MP3-файлах.

1.1. Почему Lisp?

Сложно объяснить на нескольких страницах введения, почему пользователи языка любят какой-то конкретный язык, ещё сложнее объяснить, почему вы должны тратить своё время на его изучение. Личный пример не слишком убеждает. Может быть, я люблю Lisp из-за какой-то мозговой аномалии. Это может быть даже генетическим отклонением, так как у моего отца эта аномалия, похоже, тоже была. Так что прежде, чем вы погрузитесь в изучение языка Lisp, вполне естественно желание узнать, что это вам даст, какую выгоду принесёт.

Для некоторых языков выгода очевидна. Например, если вы хотите писать низкоуровневые программы для Unix, то должны выучить C. А если вы хотите писать кросс-платформенные приложения, то должны использовать Java. А многие компании до сих пор используют C++, так что если вы хотите получить работу в одной из них, то должны знать C++.

Тем не менее для большинства языков выгоду не так просто выделить. Мы имеем дело с субъективными оценками того, насколько язык удобно использовать. Защитники Perl любят говорить, что он «делает простые вещи простыми, а сложные – возможными», и радуются факту, озвученному в девизе Perl – «Есть более чем один способ сделать это»[1]. С другой стороны, фанаты языка Python думают, что Python – прозрачный и простой язык, и код на Python проще понять, потому что, как гласит их лозунг, «Есть лишь один способ сделать это».

Так почему же Common Lisp? Здесь нет такой очевидной выгоды, как для C, Java или C++ (конечно, если вы не являетесь счастливым обладателем Lisp-машины). Выгоды от использования Lisp заключены скорее в опыте и ощущениях от его использования. В остальной части книги я буду показывать отличительные черты языка и учить вас ими пользоваться, а вы сможете по себе оценить, каково это. Для начала я попытаюсь дать вам введение в философию Lisp.

В качестве девиза для Common Lisp лучше всего подходит похожее на дзен-коан описание «программируемый язык программирования». Хотя данный девиз выглядит несколько запутанно, он тем не менее выделяет суть преимущества, которое Lisp до сих пор имеет перед другими языками программирования. Более, чем другие языки, Common Lisp следует философии: что хорошо для разработчика языка, то хорошо для его пользователей. Программируя на Common Lisp, вы, скорее всего, никогда не обнаружите нехватки каких-то возможностей в языке, которые упростили бы программирование, потому что, как будет показано далее, вы можете просто добавить эти возможности в язык.

Следовательно, в программах на Common Lisp обычно очень ярко проявляется соответствие между вашим представлением о том, как программа должна работать, и кодом, который вы пишете. Ваши идеи не замутняются нагромождением скучного кода и бесконечно повторяющимися типичными приёмами. Это упрощает поддержку кода, потому что вам больше не приходится бродить по нему всякий раз, когда вы хотите внести какие-то изменения. Даже систематические изменения в программе могут быть достигнуты относительно малыми изменениями исходного кода. Это также означает, что вы будете писать код быстрее; вы будете писать меньше кода и не будете терять времени на поиск способа красиво выразить свои идеи вопреки ограничениям, накладываемым языком программирования[2].

Common Lisp – это также прекрасный язык для программирования методом проб и ошибок: если в начале работы над программой вы ещё не знаете точно, как она будет устроена, то Common Lisp предоставляет ряд возможностей для инкрементальной и интерактивной разработки.

Интерактивный цикл read-eval-print, о котором я расскажу в следующей главе, позволяет непрерывно взаимодействовать с вашей программой по мере её разработки. Пишете новую функцию. Тестируете её. Изменяете её. Пробуете другие подходы. Вам не

[1] Perl также заслуживает изучения в качестве «изоленты для Internet».

[2] К сожалению, нет достоверных исследований продуктивности для разных языков программирования. Один из отчётов, показывающих, что Lisp не уступает C++ и Java в совокупной эффективности программ и труда программистов, находится по адресу `http://www.norvig.com/java-lisp.html`.

приходится долго ждать компиляции[3].

Ещё две особенности, способствующие непрерывному и интерактивному стилю программирования, динамическая типизация и система обработки условий. Первое позволяет вам тратить меньше времени на убеждение компилятора разрешить вам запустить программу и больше времени на её действительный запуск и работу с ней[4]. Второе позволяет интерактивно разрабатывать даже код обработки ошибок.

Ещё одно следствие «программируемости» Lisp – это то, что, кроме возможности вносить мелкие изменения в язык, упрощающие написание программ, есть возможность без труда отражать в языке значительные новые понятия, касающиеся общего устройства языка программирования. Например, первоначальная реализация Common Lisp Object System (CLOS) – объектной системы Common Lisp – была библиотекой, написанной на самом Common Lisp. Это позволило Lisp-программистам получить реальный опыт работы с возможностями, которые она предоставляла, ещё до того, как библиотека была официально включена в состав языка.

Какая бы новая парадигма программирования ни появилась, Common Lisp, скорее всего, без труда сможет впитать её без изменений в ядре языка. Например, один программист на Lisp недавно написал библиотеку AspectL, которая добавляет к Common Lisp поддержку аспектно-ориентированного программирования (AOP)[5]. Если будущее за AOP, то Common Lisp сможет поддерживать его без изменений в базовом языке и

[3] Психологи выделяют состояние сознания, называемое потоком (flow), в котором мы обладаем немыслимой концентрацией и производительностью (описан в книге Михая Чиксентмихайи (Mihaly Csikszentmihalyi) «Поток. Психология оптимального переживания»). Важность данного состояния при программировании была осознана в последние два десятилетия, с тех пор как данная тема была освещена в классической книге о человеческом факторе в программировании «Эффективные проекты и команды» Тома Демарко (Tom DeMarko) и Тимоти Листера (Tim Lister). Два ключевых факта о состоянии потока: требуется около 15 минут, чтобы войти в него, и даже короткие прерывания могут вывести из данного состояния; после этого на вход в него снова потребуется 15 минут. Демарко и Листер, как и многие последующие авторы, концентрируются на избавлении от прерываний, разрушающих состояние потока, таких как телефонные звонки и неподходящие визиты начальника. Но незаслуженно мало внимания уделяется не менее важному фактору – прерываниям из-за инструментов, которые мы используем в своей работе. Например, языки, требующие долгой компиляции, перед тем как можно будет запустить код, могут быть не менее губительными для потока, чем надоедливый начальник или звонки по телефону. Lisp может рассматриваться как язык, спроектированный для программирования в состоянии потока.

[4] Эта точка зрения противоречит некоторым распространённым мнениям. Статическая типизация против динамической – одна из классических тем для словесных перепалок между программистами. Если вы пришли из мира C++ или Java (или из мира функциональных языков со статической типизацией, таких как ML или Haskell) и не представляете жизни без статических проверок типов, можете закрыть эту книгу. Но прежде чем сделаете это, вам, возможно, будет интересно узнать, что пишут о динамической типизации такие её поборники, как Мартин Фаулер и Брюс Эккель, в своих блогах – `http://www.artima.com/weblogs/viewpost.jsp?thread=4639` и `http://www.mindview.net/WebLog/log-0025`. С другой стороны, люди из мира SmallTalk, Python, Perl или Ruby будут чувствовать себя как дома при использовании этого аспекта Common Lisp.

[5] AspectL – интересный проект, так как его предшественника из мира Java, AspectJ, создал Грегор Кичалес (Gregor Kiczales), один из проектировщиков объектной и метаобъектной системы Common Lisp. Для многих Lisp-программистов AspectJ выглядел как попытка автора портировать идеи Lisp в Java. Тем не менее Паскаль Констанца (Pascal Costanza) – автор AspectL – считает, что в AOP есть интересные идеи, которые будут полезны в Common Lisp. Конечно, он смог реализовать AspectL в виде библиотеки благодаря немыслимой гибкости Common Lisp Meta Object Protocol, разработанного Кичалесом. Для реализации AspectJ пришлось написать отдельный компилятор для компиляции нового языка в Java-код. Страница проекта AspectL находится по адресу `http://common-lisp.net/project/aspectl/`.

без дополнительных препроцессоров и прекомпиляторов[6].

1.2. С чего всё началось

Common Lisp – современный потомок языка программирования Lisp, придуманного Джоном Маккарти (John McCarthy) в 1956 году. Lisp был создан для «обработки символьных данных»[7] и получил своё имя от одной вещи, в которой он был очень хорош: обработки списков (LISt Processing). Много воды утекло с тех пор, и теперь Common Lisp обогащён набором всех современных типов данных, которые вам только могут понадобиться, а также системой обработки условий, которая, как вы увидите в главе 19, предоставляет целый дополнительный уровень гибкости, отсутствующий в системах обработки исключений таких языков, как C++, Java, Python; мощной системой объектно-ориентированного программирования; несколькими особенностями, которых нет ни в одном другом языке. Как такое возможно? Что, вы спросите, обусловило превращение Lisp в такой богатый язык?

Что же, Маккарти был (и остаётся*) исследователем в области искусственного интеллекта, и многие особенности, заложенные им в первую версию Lisp, сделали этот язык замечательным инструментом для программирования задач искусственного интеллекта. Во время бума ИИ в 80-е Lisp оставался излюбленным языком для решения сложных задач, как то: автоматическое доказательство теорем, планирование и составление расписаний, компьютерное зрение. Это были задачи, решение которых требовало написания сложных программ, для чего нужен был мощный язык, так что программисты ИИ сделали Lisp таковым. Помогла и «холодная война», так как Пентагон выделял деньги Управлению перспективных исследовательских программ (DARPA), и часть этих денег попадала к людям, занимающимся моделированием крупных сражений, автоматическим планированием и интерфейсами на естественных языках. Эти люди также использовали Lisp и продолжали совершенствовать его, чтобы язык полностью удовлетворял их потребностям.

Те же силы, что развивали Lisp, также расширяли границы и в других направлениях – сложные проблемы ИИ требуют больших вычислительных ресурсов, как бы вы их ни решали, и если вы примените закон Мура в обратную сторону, то сможете себе представить, сколь скудными эти ресурсы были в 80-е. Так что разработчики должны были найти все возможные пути улучшения производительности своих реализаций языка. В результате этих усилий современные реализации Common Lisp часто включают в себя сложные компиляторы, переводящие программы в язык, понятный машине. Хотя сегодня благодаря закону Мура можно получить высокую производительность даже от интерпретируемых языков, это больше не представляет проблемы для Common Lisp. И, как я покажу в главе 32, используя специальные (необязательные) объявления, с помощью хорошего компилятора можно получить вполне приличный машинный код, сравнимый с тем, который выдаст компилятор C.

[6] Или, выражаясь более технически грамотно, Common Lisp сам предоставляет возможность интеграции компиляторов для встроенных языков.

[7] Lisp 1.5 Programmer's Manual (M. I. T. Press, 1962).

* Джон Маккарти умер 24 октября 2011 г. — *Прим. перев.*

Но я изучал Lisp раньше, и это было не так, как вы описываете!

Если вы изучали Lisp в прошлом, то возможно ваше представление о нём не имеет ничего общего с Common Lisp. Хотя Common Lisp вытеснил большинство диалектов, от которых он был порождён, это не единственный сохранившийся диалект, и в зависимости от того, где и когда вы встретились с Lisp, вы могли хорошо изучить один из этих, отличных от Common Lisp, диалектов.

Кроме Common Lisp, активное сообщество пользователей есть у диалекта Lisp общего назначения под названием Scheme. Common Lisp позаимствовал из Scheme несколько важных особенностей, но никогда не пытался заменить его.

Язык Scheme, разработанный в Массачусетском технологическом институте (MIT), был быстро принят в качестве языка для обучения программированию на младших курсах университетов. Scheme изначально занимал отдельную нишу, в частности проектировщики языка постарались сохранить ядро Scheme настолько малым и простым, насколько это возможно. Это давало очевидные выгоды при использовании Scheme как языка для обучения, а также для исследователей в области языков программирования, которым важна возможность формального доказательства утверждений о языке.

Было также ещё одно преимущество: язык легко можно было изучить по спецификации. Все эти преимущества достигнуты за счёт отсутствия многих удобных особенностей, стандартизованных в Common Lisp. Конкретные реализации Scheme могут предоставлять эти возможности, но такие отклонения от стандарта делают написание переносимого кода на Scheme более сложным, чем на Common Lisp.

В Scheme гораздо большее внимание, чем в Common Lisp, уделяется функциональному стилю программирования и использованию рекурсии. Если вы изучали Lisp в университете и остались под впечатлением, что это академический язык, не подходящий для применения в реальной жизни, вероятно, что вы изучали именно Scheme. Не то, чтобы это было справедливым утверждением о Scheme, но под это определение ещё меньше подходит Common Lisp, создававшийся для реальных инженерных задач, а не для теоретических исследований.

Также, если вы изучали Scheme, вас могут сбить с толку некоторые различия между Scheme и Common Lisp. Эти различия являются поводом непрекращающихся религиозных войн между горячими поклонниками этих диалектов. В данной книге я постараюсь указать на наиболее существенные различия.

Два других распространённых диалекта Lisp – Elisp, язык расширений для редактора Emacs, и Autolisp, язык расширений для программы Autodesk AutoCAD. Хотя, возможно, суммарный объём кода, написанного на этих диалектах, превосходит весь остальной код на Lisp вместе взятый, оба их диалекта могут использоваться только в рамках приложений, которые они расширяют. Кроме того, они являются устаревшими, по сравнению и с Common Lisp, и с Scheme. Если вы использовали один из этих диалектов, приготовьтесь к путешествию на Lisp-машине времени на несколько десятилетий вперёд.

80-е – это также эра Lisp-машин. Несколько компаний, самая известная из которых Symbolics, выпускали компьютеры, которые могли запускать Lisp-код непосредственно на своих чипах. Так Lisp стал языком системного программирования, который использовали для написания операционных систем, текстовых редакторов, компиляторов и практически всего остального программного обеспечения Lisp-машин.

Фактически к началу 80-х существовало множество Lisp-лабораторий и несколько компаний, каждая со своей реализацией Lisp, их было так много, что люди из DARPA стали высказывать свои опасения о разобщённости Lisp-сообщества. Чтобы достигнуть

единства, группа Lisp-хакеров собралась вместе и начала процесс стандартизации нового языка, Common Lisp, который бы впитал в себя лучшие черты существующих диалектов. Их работа запечатлена в книге Common Lisp the Language (CLtL) Гая Стила (Guy Steele, Digital Press, 1984).

К 1986 году существовало несколько реализаций стандарта, призванного заменить разобщённые диалекты. В 1996-м организация The American National Standards Institute (ANSI) выпустила стандарт, расширяющий Common Lisp на базе CLtL, добавив в него новую функциональность, такую как CLOS и систему обработки условий. Но и это не было последним словом: как CLtL до этого, так и стандарт ANSI теперь намеренно позволяет разработчикам реализаций экспериментировать с тем, как лучше сделать те или иные вещи: реализация Lisp содержит богатую среду исполнения с доступом к графическому пользовательскому интерфейсу, многопоточности, сокетам TCP/IP и многому другому. В наши дни Common Lisp эволюционирует, как и большинство других языков с открытым кодом: пользователи языка пишут библиотеки, которые им необходимы, и часто делают их доступными для всего сообщества. В последние годы, в частности, заметно усиление активности в разработке Lisp-библиотек с открытым кодом.

Так что, с одной стороны, Lisp – один из классических языков информатики (Computer Science), основанный на проверенных временем идеях[8]. С другой стороны, Lisp – современный язык общего назначения, с дизайном, отражающим практический подход к решению сложных задач с максимальной надёжностью и эффективностью. Единственным недостатком «классического» наследия Lisp является то, что многие всё ещё топчутся вокруг представлений о Lisp, основанных на определённом диалекте этого языка, который они открыли для себя в середине прошлого столетия, в то время когда Маккарти разработал Lisp. Если кто-то говорит вам, что Lisp – только интерпретируемый язык, что он медленный или что вы обязаны использовать рекурсию буквально для всего, спросите вашего оппонента, какой диалект Lisp'а имеется в виду и носили ли люди брюки-клёш, когда он изучал Lisp[9].

1.3. Для кого эта книга?

Эта книга для вас, если вы интересуетесь Common Lisp, независимо от того, уверены ли вы в своём желании его использовать или просто хотите понять, из-за чего вокруг

[8] Вот некоторые идеи, впервые реализованные в Lisp: конструкция `if-then-else`, рекурсивный вызов функций, динамическое распределение памяти, сборка мусора, представление функций как полноценных объектов, лексические замыкания, интерактивное программирование, инкрементальная компиляция и динамическая типизация.

[9] Один из наиболее распространённых мифов о Lisp гласит, что он мёртв. Хотя Common Lisp действительно используется не так широко, как, скажем, Visual Basic или Java, странно называть мёртвым язык, который постоянно приобретает новых пользователей и используется для разработки новых проектов. Несколько последних случаев успешного применения Lisp – проект Viaweb Пола Грэхема, впоследствии купленный Yahoo и получивший название Yahoo Store; система заказа авиабилетов ITA Software*; QPX, используемый компанией Orbitz и другими для продажи билетов он-лайн; игра «Jak and Daxter» компании Naughty Dog для PlayStation 2, значительная часть которой написана на специализированном диалекте Lisp GOAL, компилятор которого, в свою очередь, написан на Common Lisp; Roomba – автоматический робот-пылесос, программная начинка которого написана на L – подмножестве Common Lisp. Возможно, ещё более убедительны рост объёма и популярности сайта `http://common-lisp.net`, на котором размещаются проекты на Common Lisp с открытым кодом, и стремительно возросшее в последние годы число локальных групп пользователей Lisp.

него разгорелась вся эта шумиха.

Если вы уже изучали Lisp, но не смогли перейти от академических упражнений к созданию реальных полезных программ, данная книга поможет вам сделать это. С другой стороны, необязательно испытывать желание применять Lisp, чтобы получить пользу от этой книги.

Если вы упёртый прагматик, желающий знать достоинства Common Lisp перед другими языками, такими как Perl, Python, Java, C или C#, эта книга даст вам пищу для размышлений. Или, может быть, вам нет никакого дела до использования Lisp и вы уверены, что он ничуть не лучше языков, которые вы уже знаете, но вам надоели заявления какого-нибудь Lisp-программиста, что вы просто «не в теме». Если так, то в данной книге вы найдёте краткое введение в Common Lisp. Если после чтения этой книги вы по-прежнему будете думать, что Common Lisp ничем не лучше, чем ваши любимые языки, у вас будут веские обоснованные аргументы.

В книге описываются не только синтаксис и семантика языка, но и подходы к написанию на нём полезных программ. В первой части книги я описываю сам язык и даю несколько практических примеров. После описания большей части языка (включая несколько областей, оставленных в других книгах на самостоятельное изучение) следует девять практических глав, в которых я помогу вам написать несколько полезных программ среднего размера: спам-фильтр, разборщик двоичных файлов, каталог MP3, вещание MP3 по сети и веб-интерфейс к каталогу MP3 на сервере.

По окончании чтения книги вы будете знакомы с большинством основных возможностей языка и с тем, как их следует использовать. Вы приобретёте опыт использования Common Lisp для написания нетривиальных программ и будете готовы к дальнейшему самостоятельному изучению языка. И хотя у каждого свой путь к Lisp, я надеюсь, данная книга поможет вам на этом пути. Итак, приступим!

Глава 2

Намылить, смыть, повторить: знакомство с REPL

В этой главе вы настроите среду программирования и напишете свои первые программы на Common Lisp. Мы воспользуемся лёгким в установке дистрибутивом Lisp in a Box, разработанным Matthew Danish и Mikel Evins, включающим в себя реализацию Common Lisp, Emacs – мощный текстовый редактор, прекрасно поддерживающий Lisp, а также SLIME[1] – среду разработки для Common Lisp, основанную на Emacs.

Этот набор предоставляет программисту современную среду разработки для Common Lisp, поддерживающую инкрементальный интерактивный стиль разработки, характерный для программирования на этом языке. Среда SLIME даёт дополнительное преимущество в виде унифицированного пользовательского интерфейса, не зависящего от выбранных вами операционной системы и реализации Common Lisp. В книге я буду ориентироваться на среду Lisp in a Box, но те, кто хочет изучить другие среды разработки, например графические интегрированные среды разработки (IDE – Integrated Development Environment), предоставляемые некоторыми коммерческими поставщиками, или среды, основанные на других текстовых редакторах, не должны испытывать больших трудностей в понимании[2].

2.1. Выбор реализации Lisp

Первое, что вам предстоит сделать, – выбрать реализацию Lisp. Это может показаться несколько странным для тех, кто раньше занимался программированием на таких

[1] Superior Lisp Interaction Mode for Emacs.

[2] Если у вас уже был неудачный опыт работы с Emacs, то вам следует рассматривать Lisp in a Box как IDE, которая использует Emacs в качестве текстового редактора. Однако для программирования на Lisp от вас не требуется быть гуру Emacs. С другой стороны, программировать на Lisp гораздо удобнее в редакторе, который имеет хотя бы минимальную поддержку этого языка. Наверняка вам захочется, чтобы редактор автоматически помечал парные скобки и сам мог расставить отступы в коде на Lisp. Так как Emacs почти целиком написан на одном из диалектов Lisp, Elisp, он имеет очень хорошую поддержку редактирования такого кода. История Emacs неразрывно связана с историей Lisp и культурой Lisp-хакеров: первые версии Emacs, как и его непосредственные предшественники TECMACS и TMACS, были написаны заинтересованными в Lisp разработчиками в Массачусетском технологическом институте (MIT). Редакторами, использовавшимися на Lisp-машинах, были версии Emacs, целиком написанные на Lisp. Под влиянием любви хакеров к рекурсивным акронимам две первые реализации Emacs для Lisp-машин были названы EINE и ZWEI, что означало «EINE Is Not Emacs» и «ZWEI Was EINE Initially» соответственно. Некоторое время был распространён производный от ZWEI редактор, названный более прозаично, ZMACS.

языках, как Perl, Python, Visual Basic (VB), C# или Java. Разница между Common Lisp и этими языками заключается в том, что Common Lisp определяется своим стандартом: не существует единственной его реализации, контролируемой «великодушным диктатором» (как в случае с Perl и Python), или канонической реализации, контролируемой одной компанией (как в случае с VB, C# или Java). Любой желающий может создать свою реализацию на основе стандарта. Кроме того, изменения в стандарт должны вноситься в соответствии с процессом, контролируемым Американским национальным институтом стандартов (ANSI). Этот процесс организован таким образом, что «случайные лица», такие как частные поставщики программных решений, не могут вносить изменения в стандарт по своему усмотрению[3]. Таким образом, стандарт Common Lisp – это договор между поставщиком Common Lisp и использующими Common Lisp разработчиками; этот договор подразумевает, что если вы пишете программу, использующую возможности языка так, как это описано в стандарте, вы можете рассчитывать, что эта программа запустится на любой совместимой реализации Common Lisp.

С другой стороны, стандарт может не описывать всё то, что вам может понадобиться в ваших программах. Более того, некоторые аспекты языка спецификация намеренно опускает, чтобы дать возможность экспериментальным путём решить спорные вопросы языка. Таким образом, каждая реализация предоставляет пользователям как возможности, входящие в стандарт, так и выходящие за его пределы. В зависимости от того, что вы хотите программировать, вы можете выбрать реализацию Common Lisp, поддерживающую именно те дополнительные возможности, которые вам больше всего понадобятся. С другой стороны, если вы пишете код, которым будут пользоваться другие разработчики, то вы, вероятно, захотите – конечно, в пределах возможного – писать переносимый код на Common Lisp. Для нужд написания переносимого кода, который в то же время использует возможности, не описанные в стандарте, Common Lisp предоставляет гибкий способ писать код, «зависящий» от возможностей текущей реализации. Вы увидите пример такого кода в главе 15, когда мы будем разрабатывать простую библиотеку, «сглаживающую» некоторые различия в обработке имён файлов разными реализациями Lisp.

Сейчас, однако, наиболее важная характеристика реализации – её способность работать в вашей любимой операционной системе. Сотрудники компании Franz, занимающейся разработкой Allegro Common Lisp, выпустили пробную версию своего продукта, работающего на GNU/Linux, Windows и OS X, предназначенную для использования с этой книгой. У читателей, предпочитающих реализации с открытым исходным кодом, есть несколько вариантов. SBCL[4] – высококачественная открытая реализация, способная компилировать в машинный код и работать на множестве различных UNIX-

[3] На самом деле существует очень малая вероятность пересмотра стандарта языка. Хотя есть некоторое количество недостатков, которые пользователи языка могут желать исправить, согласно процессу стандартизации ANSI, существующий стандарт не подлежит открытию для внесения небольших изменений, и эти недостатки на самом деле, не вызывают ни у кого серьёзных трудностей. Возможно, будущее стандартизации Common Lisp – за стандартами «де-факто», больше похожими на «стандартизацию» Perl и Python, когда различные разработчики экспериментируют с интерфейсами прикладного программирования (API) и библиотеками для реализации вещей, не описанных в стандарте языка, а другие разработчики могут принимать их; или заинтересованные программисты будут разрабатывать переносимые библиотеки для сглаживания различий между реализациями возможностей, не описанных в стандарте языка.

[4] Steel Bank Common Lisp.

систем, включая Linux и OS X. SBCL – «наследник» CMUCL[5] – реализации Common Lisp, разработанной в университете Carnegie Mellon, и, как и CMUCL, является всеобщим достоянием (public domain), за исключением нескольких частей, покрываемых BSD-подобными (Berkley Software Distributions) лицензиями. CMUCL – тоже хороший выбор, однако SBCL обычно легче в установке и поддерживает 21-разрядный Unicode[6]. OpenMCL будет отличным выбором для пользователей OS X: эта реализация способна компилировать в машинный код, поддерживать работу с потоками, а также прекрасно интегрируется с инструментальными комплектами Carbon и Cocoa. Кроме перечисленных, существуют и другие свободные и коммерческие реализации. В главе 32 перечислены источники для получения дополнительной информации.

Весь код на Lisp, приведённый в этой книге, должен работать на любой совместимой реализации Common Lisp, если явно не указано обратное, и SLIME будет «сглаживать» некоторые различия между реализациями, предоставляя общий интерфейс для взаимодействия с Lisp. Сообщения интерпретатора, приведённые в этой книге, сгенерированы Allegro, запущенным на GNU/Linux. В некоторых случаях другие реализации Lisp могут генерировать сообщения, незначительно отличающиеся от приведённых.

2.2. Введение в Lisp in a Box

Поскольку Lisp in a Box спроектирован с целью быть «дружелюбным» к новичкам, а также предоставлять первоклассную среду разработки на Lisp, всё, что вам нужно для работы, – выбрать пакет, соответствующий вашей операционной системе, с веб-сайта Lisp in a Box (см. главу 32), а затем просто следовать инструкциям по установке.

Так как Lisp in a Box использует Emacs в качестве текстового редактора, вам стоит хоть немного научиться им пользоваться. Возможно, лучший способ начать работать с Emacs – это изучать его по встроенному учебнику (tutorial). Чтобы вызвать учебник, выберите первый пункт меню Help – Emacs tutorial. Или же зажмите `Ctrl` и нажмите `h`, затем отпустите `Ctrl` и нажмите `t`. Большинство команд в Emacs доступно через комбинации клавиш, поэтому они будут встречаться довольно часто, и чтобы долго не описывать комбинации (например: «зажмите `Ctrl` и нажмите `h`, затем...»), в Emacs существует краткая форма записи комбинаций клавиш. Клавиши, которые должны быть нажаты вместе, пишутся вместе, разделяются тире и называются связками. Связки разделяются пробелами. `C` обозначает `Ctrl`, а `M` – `Meta` (`Alt`). Например, вызов tutorial будет выглядеть следующим образом: `C-h t`.

Учебник также описывает много других полезных команд Emacs и вызывающих их комбинаций клавиш. Более того, у Emacs есть расширенная онлайн-документация, для просмотра которой используется специальный браузер – Info. Чтобы её вызвать, нажмите `C-h i`. У Info, в свою очередь, есть своя справка, которую можно вызвать, нажав клавишу h, находясь в браузере Info. Emacs предоставляет ещё несколько способов получить справку – это все сочетания клавиш, начинающиеся с `C-h`, – полный

[5] CMU Common Lisp.

[6] SBCL стал «ответвлением» CMUCL, так как его разработчики хотели сосредоточиться на упорядочивании его внутренней организации и сделать его легче в сопровождении. «Ответвление» вышло очень удачным. Исправления ошибок привели к появлению серьёзных различий между двумя проектами, и, как теперь поговаривают, их снова планируют объединить.