ОГЛАВЛЕНИЕ

ОБ АВТОРАХ

Бруно Кардос Лопес (Bruno Cardoso Lopes) получил степень доктора информационных технологий в университете города Капинас (Campinas), Бразилия. С 2007 года участвует в разработке LLVM, с нуля реализовал поддержку архитектуры MIPS и сопровождал ее в течение нескольких лет. Также занимался разработкой поддержки x86 AVX и ассемблера ARM. В настоящее время занимается исследованиями приемов сжатия кода и сужения системы команд (Instruction Set Architecture, ISA). В прошлом занимался разработкой драйверов для операционных систем Linux и FreeBSD.

Рафаэль Аулер (Rafael Auler) защитил кандидатскую диссертацию в университете города Капинас (Campinas), Бразилия. Имеет степени магистра информационных технологий и бакалавра в области конструирования вычислительных машин, полученные в том же университете. В дипломной работе на степень магистра реализовал экспериментальную версию инструмента, автоматически генерирующего генераторы выполняемого кода (backends) для поддержки архитектур в LLVM на основе файлов описаний. В настоящее время работает над докторской диссертацией и исследует приемы динамической двоичной трансляции, динамической (Just-in-Time) компиляции и компьютерные архитектуры. Также является лауреатом премии «Microsoft Research 2013 Graduate Research Fellowship Award».

О РЕЦЕНЗЕНТАХ

Эли Бендерски (Eli Bendersky) уже 15 лет профессионально занимается программированием, имеет богатый опыт системного программирования, включая компиляторы, компоновщики и отладчики. С начала 2012 года является одним из основных разработчиков проекта LLVM.

Логан Чен (Logan Chien) получил степень магистра информационных технологий в национальном университете Тайваня. Занимается исследованием архитектуры компиляторов, виртуальных машин и приемов оптимизации во время компиляции. Принимал участие в нескольких проектах по разработке открытого программного обеспечения, включая LLVM, Android и другие. Написал несколько исправлений к механизму обработки исключений с нулевыми накладными расходами для архитектуры ARM (zero-cost exception handling mechanism) и расширений для ассемблера ARM, интегрированного в LLVM. В 2012 проходил стажировку в Google как инженер-программист, во время которой занимался интеграцией LLVM в Android NDK.

Цзя Лю (Jia Liu) во время учебы в колледже начал заниматься разработкой программного обеспечения для GNU/Linux и уже после учебы участвовал в нескольких проектах по разработке открытого программного обеспечения. В настоящее время отвечает за все, что связано с программным обеспечением в China-DSP.

Интересуется технологиями компиляции и работает в этом направлении уже несколько лет. В свое свободное время все так же участвует в нескольких открытых проектах, таких как LLVM, QEMU и GCC/Binutils.

Был нанят китайским производителем микропроцессоров, компанией Glarun Technology, более широко известную, как China-DSP. China-DSP – это поставщик высокопроизводительных процессоров цифровой обработки сигналов (Digital Signal Processor, DSP). Ос-

нову бизнеса компании составляют: проектирование процессоров, разработка системного программного обеспечения и создание встраиваемых платформ параллельной обработки для электроэнергетики, связи, автомобилестроения, приборостроения и бытовой электроники.

Я хочу сказать спасибо моим родителям за то, что я появился на свет. Спасибо моей подруге, наставляющей меня на путь истинный. Спасибо моим коллегам за счастье работать с ними.

Джон Шакмейстер (John Szakmeister) получил степень магистра электротехники и электроники в университете Джонса Хопкинса (Балтимор, Мериленд, США) и является сооснователем Intelesys Corporation (www.intelesyscorp.com). Профессионально занимается программированием уже более 15 лет и обожает возиться с компиляторами, операционными системами, сложными алгоритмами и любыми встраиваемыми системами. Ярый сторонник Open Source и в свое свободное время принимает участие в развитии многих открытых проектов. В свободное от программирования время Джон упорно тренируется, стремясь к своему черному поясу по ниндзюцу, или читает техническую литературу.

Я хочу поблагодарить мою супругу Энн (Ann) и двух моих мальчиков, Мэттью (Matthew) и Эндрю (Andrew), за их терпение и понимание, когда я был занят рецензированием этой книги.

ПРЕДИСЛОВИЕ

Проект LLVM начинался со страсти к компиляторам единственного человека, Криса Латтнера (Chris Lattner). В событиях, последовавших за выходом первой версии LLVM, и дальнейшем подключении к проекту других разработчиков явно прослеживается сценарий развития, характерный для многих успешных открытых проектов: они появляются не в недрах крупных компаний, а благодаря простому человеческому интересу к той или иной теме. Например, первое ядро Linux появилось, благодаря интересу финского студента к операционным системам и его стремлению понять и увидеть на практике, как в действительности работают операционные системы.

И Linux, и LLVM развились до уровня первоклассного программного обеспечения, способного конкурировать с коммерческими аналогами, во многом благодаря вкладу большого числа программистов. И было бы несправедливо приписывать успех любого большого проекта одному человеку. Однако переход от студенческого проекта к сложному и надежному программному продукту в сообществе Open Source во многом зависит от такого важного фактора, как привлечение сторонних разработчиков, кому пришлось бы по душе тратить свое время на проект.

В школах присутствует атмосфера увлекательности, потому что образование предполагает изучение внутреннего порядка вещей. Для учащихся познание тайн устройства сложных механизмов являет собой процесс перехода от ощущения полной загадочности к чувству победы и уверенного владения знаниями. В такой среде, в университете города Урбана-Шампейн штата Иллинойс (UIUC), зародился проект LLVM, который использовался как прототип для дальнейших исследований и как учебное пособие для курса лекций по устройству компиляторов, который вел Викрам Адве (Vikram Adve), руководитель дипломного проекта Латтнера. Студенты помогали в выявлении первых ошибок и дали начальный импульс в развитии LLVM, как простого в изучении программного проекта с ясной архитектурой.

Вопиющее несоответствие теории и практики программирования сбивают с толку многих студентов – будущих программистов. Про-

стая и ясная концепция в теории может включать в себя столько наслоений из деталей реализации, что программные проекты становится просто невозможно охватить мысленно. Хорошо продуманная архитектура с мощными абстракциями помогает человеческому мозгу разобраться во всех наслоениях и на всех уровнях: от общего представления, как работает программа, до мельчайших подробностей ее реализации.

Это особенно верно для компиляторов. Студенты, испытывающие желание узнать, как работают компиляторы, часто сталкиваются с трудной задачей, когда дело доходит до изучения фактической реализации компилятора. До появления LLVM, одним из немногих компиляторов с открытым исходным кодом, доступных хакерам[1] и любопытным студентам для изучения практической реализации, был GCC.

Любой программный проект достаточно ясно отражает представления программистов, создавших его. Это можно наблюдать в абстракциях, используемых для отделения модулей друг от друга и представления данных в разных компонентах. Программисты могут иметь разные взгляды на одно и то же. Соответственно, давнишние и крупные проекты, такие как GCC (который развивается уже более 30 лет), часто являются комбинацией взглядов и представлений целых поколений программистов, что делает такие проекты чрезвычайно сложными в изучении для вступающих в них новых программистов и любознательных исследователей.

Проект LLVM привлекает не только опытных программистов – разработчиков компиляторов, – но также множество юных и любознательных умов, которые видят в нем более ясный и простой для изучения программный продукт, представляющий собой компилятор с огромным потенциалом. Это очевидно из большого числа научных трудов, выбравших LLVM за основу для исследований. Причина проста; в научном сообществе за практические аспекты реализации часто отвечают студенты, поэтому для научно-исследовательских проектов первостепенное значение имеет простота кодовой базы, чтобы студенты легко могли овладеть ею. Соблазненные новейшей архитектурой на основе языка C++ (вместо C, на котором написан GCC), модульной структурой (вместо монолитной в GCC) и концепциями, которые легко укладываются в современную теорию построения ком-

[1] Здесь слово «хакер» используется в его первоначальном смысле: «высококвалифицированный ИТ-специалист, человек, который понимает тонкости работы программ ЭВМ». – *Прим. перев.*

пиляторов, многие исследователи отметили, насколько просто освоить LLVM и использовать для реализации своих идей. Успех LLVM в академической среде определенно стал следствием уменьшением разрыва между теорией и практикой.

Проект LLVM привлекает не только академическое сообщество, как инструмент для исследований, но и промышленность, из-за гораздо более либеральной лицензии, по сравнению с лицензией GPL, на условиях которой распространяется GCC. В академических кругах, где вырос проект, ложкой дегтя в бочке меда для исследователей, пишущих программный код, является страх, что он будет использоваться только в единственном эксперименте, по окончании которого этот код останется только выбросить. Чтобы исключить возможность такой участи, Крис Латтнер, давший жизнь LLVM в своем дипломном проекте, решил лицензировать свою разработку на условиях University of Illinois/NCSA Open Source License, допускающих коммерческое и некоммерческое использование продукта при сохранении упоминания об авторских правах. Цель состояла в том, чтобы максимально упростить распространение LLVM, и эта цель была достигнута в полной мере. В 2012 году, проект LLVM был удостоен премии «ACM Software System Award», присуждаемой за разработку программных систем, сыгравших важную роль в науке.

Многие компании воспользовались проектом LLVM для удовлетворения своих нужд и вносят свой вклад в его развитие, расширяя спектр языков, поддерживаемых LLVM, а также аппаратных архитектур, для которых эти компиляторы могут генерировать код. На этом новом этапе развития проекта было обеспечено непревзойденное совершенство библиотек и инструментов, позволившее навсегда оставить в прошлом уровень экспериментального академического программного продукта и перейти на уровень надежной основы для коммерческих разработок. Вместе с этим изменилось и имя проекта: название «Low Level Virtual Machine» (низкоуровневая виртуальная машина) сменилось на его аббревиатуру LLVM.

Решение сменить название «Low Level Virtual Machine» на аббревиатуру LLVM отражает изменение целей проекта. Первоначально LLVM создавалась как дипломный проект, с целью служить основой для изучения приемов программной оптимизации. Заложенные в проект идеи первоначально были опубликованы на международном симпозиуме 2003 MICRO (International Symposium on Microarchitecture), в статье под заглавием «LLVA: A Low-level Virtual Instruction Set Architecture», где описывался набор инструкций, и на

симпозиуме 2004 CGO (International Symposium on Code Generation and Optimization), в статье под заглавием «LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation».

Выйдя за академические рамки, LLVM превратился в удачно спроектированный компилятор с интересным свойством – сохранением на диск промежуточного представления программного кода. В коммерческих системах LLVM никогда не использовался в роли настоящей виртуальной машины, как, например, виртуальная машина Java (Java Virtual Machine, JVM), поэтому не имело смысла сохранять прежнее название Low Level Virtual Machine. С другой стороны, в наследство остались некоторые другие любопытные названия. Файл на диске, где хранится в программа в промежуточном представлении LLVM, обычно называют «LLVM bitcode» (LLVM-биткод), в пику названию «Java bytecode» (Java-байткод), подчеркивая разницу в размерах промежуточного представления в LLVM и Java.

Работая над этой книгой, мы ставили перед собой две цели. Во-первых, поскольку проект LLVM сильно разросся, мы хотели познакомить вас с небольшими его частями, по одной составляющей за раз, чтобы максимально упростить изучение и одновременно дать почувствовать радость владения мощной библиотекой компиляторов. Во-вторых, мы хотели вдохнуть в вас дух свободного хакерства, чтобы вы не ограничивали себя рамками концепций, представленных здесь, и никогда не прекращали расширять свой кругозор.

Успешной работы!

Содержание книги

Глава 1, «Сборка и установка LLVM», покажет, как установить пакет Clang/LLVM в Linux, Windows и Mac OS, а также, как собрать LLVM в Visual Studio и Xcode. Здесь также будут обсуждаться разные версии дистрибутивов LLVM и вопросы выбора лучшей версии для ваших условий: скомпилированные файлы, пакеты дистрибутивов или исходные коды.

Глава 2, «Внешние проекты», познакомит с внешними проектами LLVM, распространяемыми в отдельных пакетах или хранящимися в других репозиториях, такими как дополнительные инструменты Clang и расширение DragonEgg GCC, отладчик LLVM Debugger (LLDB) и пакет тестов LLVM.

Глава 3, «Инструменты и организация», описывает организацию инструментов в проекте LLVM, демонстрирует примеры их исполь-

зования для получения промежуточного кода из исходных текстов. Также рассказывает, как действует драйвер компилятора, и, наконец, как написать очень простой инструмент для LLVM.

Глава 4, «Анализатор исходного кода», знакомит с анализатором исходного кода LLVM – проектом Clang. Проведет вас через все этапы работы анализатора на примерах создания небольших программ, использующих каждую часть интерфейса. Завершается примером небольшого драйвера компилятора, использующего библиотеки Clang.

Глава 5, «Промежуточное представление LLVM», разъясняет ключевой аспект архитектуры LLVM: промежуточное представление (intermediate representation). Показывает отличительные характеристики этого представления, знакомит с синтаксисом, структурой и приемами разработки инструментов, генерирующих промежуточное представление LLVM IR.

Глава 6, «Генератор выполняемого кода», знакомит с устройством генератора выполняемого кода (backend) LLVM, ответственного за преобразование промежуточного представления LLVM IR в машинный код. Эта глава проведет вас через все этапы работы генератора кода, знание которых поможет вам создать собственный генератор для LLVM. Завершается примером создания такого генератора.

Глава 7, «Динамический компилятор», разъясняет инфраструктуру динамической компиляции (Just-in-Time) в LLVM, которая позволяет генерировать и выполнять машинный код по мере необходимости. Эта технология особенно востребована в приложениях, где во время выполнения имеется только исходный программный код, таких как интерпретаторы JavaScript в браузерах. Эта глава проведет вас через все этапы использования библиотек при разработке собственного JIT-компилятора.

Глава 8, «Кросс-платформенная компиляция», проведет вас через все этапы создания программ для других аппаратных архитектур, таких как ARM, с использованием Clang/LLVM. Покажет, как правильно настроить окружение для компиляции программ, которые будут выполняться за пределами этого окружения.

Глава 9, «Статический анализатор Clang», описывает мощный инструмент поиска ошибок в исходном программном коде до запуска программ за счет анализа этого кода. Данная глава также покажет, как дополнить статический анализатор Clang своими собственными средствами контроля.

Глава 10, «Инструменты Clang и фреймворк LibTooling», знакомит с фреймворком LibTooling и некоторыми встроенными инструментами Clang, дающими возможность выполнить рефакторинг исходного кода и простые виды его анализа. Эта глава завершается примером создания собственного инструмента рефакторинга исходного кода на языке C++ с применением данной библиотеки.

Когда мы работали над этой книгой, версия LLVM 3.5 еще не вышла. Но, несмотря на то, что в центре внимания этой книги находится версия LLVM 3.4, мы планируем обновить примеры до версии LLVM 3.5 к третьей неделе сентября 2014 года, чтобы дать вам возможность опробовать их с более новой версией LLVM. Обновленные примеры будут доступны по адресу: https://www.packtpub.com/sites/default/files/downloads/6924OS_Appendix.pdf.

Что потребуется для работы с книгой

Для исследования LLVM можно использовать операционные системы UNIX, Mac OS X и Windows, при наличии в них современного компилятора C++. Исходный код LLVM очень требователен к выбору компилятора C++ – компилятор должен соответствовать новейшим стандартам. Это означает, что в Linux вы должны использовать версию GCC не ниже 4.8.1; в Max OS X версию Xcode не ниже 5.1; и в Windows – как минимум Visual Studio 2012.

Несмотря на то, что здесь рассказывается, как собрать LLVM в Windows с помощью Visual Studio, эта книга не нацелена на данную платформу, потому что некоторые возможности LLVM на ней недоступны. Например, в Windows LLVM не поддерживает загружаемые модули, но мы покажем, как писать расширения в виде разделяемых библиотек. Поэтому, чтобы познакомиться с такими возможностями на практике, необходимо использовать Linux или Mac OS X.

Если вы не желаете заниматься сборкой LLVM самостоятельно, воспользуйтесь пакетом готовых двоичных файлов. Правда скомпилированные дистрибутивы доступны не для всех платформ.

Кому адресована эта книга

Эта книга адресована энтузиастам, студентам, изучающим информационные технологии, и разработчикам компиляторов, интересую-

щимся фреймворком LLVM. Читатели должны знать язык программирования C++ и, желательно, иметь некоторые представления о теории компиляции. И для начинающих, и для опытных специалистов эта книга послужит практическим введением в LLVM, не содержащим сложных сценариев. Если вас интересует данная технология, тогда эта книга определенно для вас.

Типографские соглашения

В этой книге используется несколько разных стилей оформления текста, с целью обеспечить визуальное отличие информации разных типов. Ниже приводится несколько примеров таких стилей оформления и краткое описание их назначения.

Программный код в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, пути в файловой системе, адреса URL, ввод пользователя и ссылки в Twitter оформляются, как показано в следующем предложении:

«Предварительно скомпилированные пакеты для Windows распространяются в виде простого в использовании инсталлятора, который сам распакует дерево каталогов LLVM в папку `Program Files`.»

Блоки программного кода оформляются так:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

int main() {
    uint64_t a = 0ULL, b = 0ULL;
    scanf ("%lld %lld", &a, &b);
    printf ("64-bit division is %lld\n", a / b);
    return EXIT_SUCCESS;
}
```

Когда нам потребуется привлечь ваше внимание к определенному фрагменту в блоке программного кода, мы будем выделять его жирным шрифтом:

```
KEYWORD(float  , KEYALL)
KEYWORD(goto   , KEYALL)
KEYWORD(inline , KEYC99|KEYCXX|KEYGNU)
KEYWORD(int    , KEYALL)
KEYWORD(return , KEYALL)
KEYWORD(short  , KEYALL)
KEYWORD(while  , KEYALL)
```

Ввод и вывод в командной строке будет оформляться так:

```
$ sudo mv clang+llvm-3.4-x86_64-linux-gnu-ubuntu-13.10 llvm-3.4
$ export PATH="$PATH:/usr/local/llvm-3.4/bin"
```

Новые термины и важные определения будут выделяться в обычном тексте жирным. Надписи, которые вы будете видеть на экране, например в меню или в диалогах, будут выделяться в тексте так: «Во время установки не забудьте отметить флажок **Add CMake to the system PATH for all users** (Добавить CMake в системную переменную PATH для всех пользователей)».

> **Примечание.** *Так будут оформляться предупреждения и важные примечания.*

> **Совет.** *Так будут оформляться советы и рекомендации.*

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или может быть не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф в разделе «Читателям – Файлы к книгам».

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки всё равно случаются. Если вы найдёте ошибку в одной из наших книг – возможно, ошибку в тексте или в коде – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдёте какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства ДМК Пресс и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, и помогающую нам предоставлять вам качественные материалы.

ГЛАВА 1. Сборка и установка LLVM

Фреймворк LLVM доступен для разных версий **Unix** (GNU/**Linux**, **FreeBSD**, **Mac OS X**) и **Windows**. В этой главе мы расскажем, какие шаги необходимо выполнить, чтобы получить действующий фреймворк LLVM во всех этих системах. Для некоторых систем доступны предварительно скомпилированные пакеты LLVM и Clang, но вы можете сами выполнить сборку из исходных текстов.

Пользователи, только начинающие изучать LLVM, должны знать, что компиляторы на основе LLVM включают в себя библиотеки и инструменты из обоих пакетов – LLVM и Clang. Поэтому все инструкции в этой главе описывают сборку и установку обоих пакетов. На протяжении всей книги мы будем подразумевать версию LLVM 3.4. Однако, имейте в виду, что LLVM – это молодой проект и продолжает активно развиваться, поэтому в последующих версиях некоторые аспекты могут измениться.

> **Совет.** *Когда мы работали над этой книгой, версия LLVM 3.5 еще не вышла. Но, несмотря на то, что в центре внимания этой книги находится версия LLVM 3.4, мы планируем обновить примеры до версии LLVM 3.5 к третьей неделе сентября 2014 года, чтобы дать вам возможность опробовать их с более новой версией LLVM. Обновленные примеры будут доступны по адресу:* `https://www.packtpub.com/sites/default/files/downloads/6924OS_Appendix.pdf`.

В этой главе рассматриваются следующие темы:

- порядок нумерации версий LLVM;
- установка предварительно скомпилированных файлов LLVM;
- установка LLVM с использованием диспетчера пакетов;
- сборка LLVM из исходных текстов для Linux;
- сборка LLVM из исходных текстов для Windows и Visual Studio;
- сборка LLVM из исходных текстов для Mac OS X и Xcode.

Порядок нумерации версий LLVM

Проект LLVM продолжает быстро развиваться, благодаря усилиям многих программистов. За 10 лет после выпуска первой версии и до момента появления версии 3.4, в репозиторий SVN (в качестве системы управления версиями используется Subversion) было передано более 200 000 исправлений и изменений. За один только 2013 год в репозиторий было передано 30 000 новых изменений. Как следствие, постоянно появляются новые возможности, а прежние – быстро устаревают. Как и в любом другом крупном проекте, разработчики должны точно следовать плотному графику выпуска стабильных версий, уверенно проходящих разнообразные тесты, чтобы дать пользователям возможность испытать новые возможности.

На протяжении всей своей истории проект LLVM следует стратегии выпуска двух стабильных версий в год. В каждой следующей версии увеличивается младший номер. Например, при переходе от версии 3.3 к версии 3.4 изменяется младший номер версии. Когда младший номер достигает значения 9, в следующей версии будет увеличен старший номер, например, после версии LLVM 2.9 была выпущена версия LLVM 3.0. Изменение старшего номера версии вовсе не означает наличие существенных изменений, в сравнении с предыдущей версией, а всего лишь служит границей очередного пятилетнего отрезка развития проекта.

Для проектов, зависящих от LLVM, типично использовать *стволовую* (*trunk*) версию, то есть, наиболее свежую, доступную в репозитории SVN, которая иногда может оказаться нестабильной. Недавно, начиная с версии 3.4, в сообществе LLVM было решено выпускать *промежуточные версии*, введя дополнительный номер версии – номер выпуска. Первым результатом такого решения стал выход версии LLVM 3.4.1. Цель выпуска промежуточных версий – включение исправлений из стволовой версии в последнюю помеченную (tagged) версию, не имеющую новых особенностей, чтобы обеспечить полную совместимость. Промежуточные версии должны выходить через три месяца после выхода последней основной версии. Так как такая нумерация версий еще достаточно нова, мы сосредоточимся на установке версии LLVM 3.4. Существует большое число предварительно скомпилированных пакетов LLVM 3.4, но, следуя нашим инструкциям, вы без особых усилий сможете самостоятельно собрать LLVM 3.4.1 или более новую версию.

Установка скомпилированных пакетов LLVM

Чтобы упростить задачу установки программного обеспечения и избавить вас от необходимости выполнять компиляцию самостоятельно, поставщики LLVM создают предварительно скомпилированные пакеты для определенных платформ. Компиляция программного обеспечения иногда может оказаться очень непростым делом; она может потребовать некоторого времени и должна выполняться, только если вы используете другую платформу или принимаете активное участие в разработке проекта. Поэтому, для желающих побыстрее приступить к исследованию LLVM, существуют предварительно скомпилированные пакеты. Однако, в этой книге мы советуем использовать исходные тексты LLVM. Вы должны быть готовы скомпилировать LLVM самостоятельно.

Существует два основных способа установки предварительно скомпилированных пакетов LLVM: на официальном веб-сайте или на сайтах проектов, выпускающих дистрибутивы для GNU/Linux и Windows.

Установка скомпилированных пакетов с официального сайта

На официальном сайте проекта LLVM можно загрузить следующие предварительно скомпилированные пакеты для версии 3.4:

Таблица 1.1. Официальные скомпилированные пакеты для версии LLVM 3.4

Архитектура	Версия
x86_64	Ubuntu (12.04, 13.10), Fedora 19, Fedora 20, FreeBSD 9.2, Mac OS X 10.9, Windows и openSUSE 13.1.
i386	openSUSE 13.1, FreeBSD 9.2, Fedora 19, Fedora 20 и openSUSE 13.1
ARMv7/ARMv7a	Linux

Более полный перечень вы найдете на сайте http://www.llvm.org/releases/download.html, в разделе **Pre-built Binaries** (предварительно скомпилированные пакеты) для версии LLVM, которую вы желали бы загрузить. Например, чтобы загрузить и установить LLVM в Ubuntu 13.10, можно выполнить следующие команды:

```
$ sudo mkdir -p /usr/local; cd /usr/local
$ sudo wget http://llvm.org/releases/3.4/clang+llvm-3.4-x86_64-linux-
gnu-ubuntu-13.10.tar.xz
$ sudo tar xvf clang+llvm-3.4-x86_64-linux-gnu-ubuntu-13.10.tar.xz
$ sudo mv clang+llvm-3.4-x86_64-linux-gnu-ubuntu-13.10 llvm-3.4
$ export PATH="$PATH:/usr/local/llvm-3.4/bin"
```

После этого LLVM и Clang готовы к использованию. Не забывайте о необходимости постоянно изменять системную переменную окружения `PATH`, потому что изменение, которое было выполнено в последней строке, действует только до завершения текущего сеанса. Чтобы проверить корректность установки, можно попробовать выполнить простую команду, которая выведет версию Clang:

```
$ clang -v
```

Если при попытке выполнить эту команду возникли какие-либо проблемы, попробуйте запустить выполняемый файл непосредственно из каталога, куда он был установлен, чтобы убедиться, что проблема не вызвана неправильной настройкой переменной окружения `PATH`. Если и в этом случае запустить Clang не удалось, возможно вы установили предварительно скомпилированную версию, несовместимую с вашей системой. Не забывайте, что во время компиляции, двоичные файлы компонуются с динамическими библиотеками определенных версий. Сообщение об отсутствии библиотек, появляющееся при попытке запустить приложение, служит явным признаком, что двоичные файлы были скомпилированы в системе, несовместимой с вашей.

> **Совет.** *В Linux, например, текст сообщения об отсутствии библиотеки содержит имя выполняемого файла и имя динамической библиотеки, которую не удалось загрузить. Обратите внимание на имя динамической библиотеки – это явный признак, что динамический компоновщик и загрузчик не смог загрузить данную библиотеку, потому что программа была собрана в несовместимой системе.*

Чтобы установить предварительно скомпилированные пакеты в других системах (кроме Windows), можно выполнить точно такую же последовательность шагов. Предварительно скомпилированные пакеты для Windows распространяются в виде простого в использовании инсталлятора, который сам распакует дерево каталогов пакета LLVM в папку `Program Files`. Инсталлятор предусматривает возможность автоматического изменения переменной окружения `PATH`, чтобы можно было запускать Clang из любого окна командной строки.