Для написания книги нужны время и силы. Без поддержки и понимания моей жены Тани и дочери Полины эта книга не состоялась бы. Спасибо, что вы всегда на моей стороне!

Из-за некоторых проблем личного характера этот проект находился под угрозой, и я благодарен Эми, ставшей моим соавтором. Без нее книга получилась бы не такой хорошей.

И еще, коллектив издательства Packt не только подсказывал мне, как нужно писать, но и проявлял понимание, когда я работал медленно, и всегда настаивал на продолжении. Я всем им очень признателен.

– Кай Наке

2023 год стал поворотным в моей жизни, и привнесение моих знаний о LLVM в эту книгу сыграло в этом не последнюю роль. Я никогда не помышляла о том, чтобы присоединиться к Каю в этом волнующем предприятии с целью разделить LLVM 17 со всеми вами! Спасибо Каю за техническое наставничество и руководство, спасибо коллективу Packt и, конечно, моей семье и близким за поддержку и мотивацию во время работы над этой книгой.

– Эми Кван

Содержание

От издательства

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 на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и 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.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторах

Кай Наке – профессиональный ИТ-архитектор, который в настоящее время проживает в Торонто, Канада. Его дипломная работа по информатике, написанная в Техническом университете Дортмунда, Германия, была посвящена универсальным функциям хеширования и признана лучшей в семестре.

Имея за плечами 20 лет работы в ИТ-индустрии, Кай приобрел обширный опыт в разработке архитектуры и кода корпоративных приложений для бизнеса. В настоящее время он занят развитием компилятора на основе LLVM/clang.

В течение нескольких лет Кай занимался сопровождением LDC, компилятора языка D на основе LLVM. Он автор книг «D Web Development» и «Learn LLVM 12», вышедших в издательстве Packt. В прошлом выступал с докладом на секции разработчиков LLVM в рамках конференции **Free and Open Source Software Developers' European Meeting** (**FOSDEM**).

Эми Кван – разработчик компиляторов, в настоящее время проживает в Торонто, Канада. Родившаяся в прериях Канады, Эми получила степень бакалавра наук по информатике в Саскачеванском университете. В настоящее время использует технологию LLVM для разработки фазы генерирования кода компилятором. Выступала с докладом на конференции **LLVM Developer Conference** в 2022 г. вместе с Каем Наке.

О рецензентах

Акаш Котхари – младший научный сотрудник в Иллинойской исследовательской лаборатории LLVM-компиляторов. Степень доктора информатики получил в Иллинойском университете в Урбана-Шампейне. Акаш специализируется в области исследования производительности, синтеза программ и формальной семантики и ее верификации, но его интересы простираются также в историю компьютеров и систем программирования.

Шуо Ниу, магистр технических наук в области вычислительной техники – движущая сила в разработке компьютерных технологий. Проведя пять плодотворных лет в подразделении Intel PSG, специализирующемся на компиляторах для высокоуровневого проектирования ППВМ, он возглавил инновационные разработки в сфере фазы оптимизации компиляторов. Его опыт в создании передовых продуктов позволил пользователям достигать выдающейся производительности плат ППВМ.

Предисловие

Создание компилятора – трудная и увлекательная задача. Проект LLVM предлагает допускающие повторное использование компоненты для вашего компилятора и базовые библиотеки, позволяющие реализовать высококачественный оптимизирующий кодогенератор, который транслирует исходное машинно-независимое промежуточное представление машинного кода для всех популярных процессорных архитектур. Компиляторы многих языков программирования уже пользуются плодами технологии LLVM.

Эта книга научит вас, как реализовать свой компилятор, используя для данной цели LLVM. Вы узнаете, как фронтальная часть компилятора преобразует исходный код в абстрактное синтаксическое дерево, а затем генерирует из него **промежуточное представление** (*англ.* Intermediate Representation – **IR**). После этого вы познакомитесь с тем, как добавить в состав компилятора конвейер оптимизации, который преобразует IR в высокопроизводительный машинный код.

Каркас LLVM можно расширить в нескольких направлениях, вы научитесь добавлять новые проходы и даже совершенно новый кодогенератор (*англ.* backend). Не остались без внимания и такие дополнительные темы, как компиляция для другой архитектуры процессора и расширение clang и статического анализатора clang собственными плагинами и проверщиками. В этой книге принят практический подход и имеется много примеров кода, что упрощает применение полученных знаний в собственных проектах.

Что нового в этом издании

Добавлена новая глава, посвященная знакомству с концепцией и синтаксисом языка TableGen, используемого внутри LLVM, на котором читатели могут определять классы, записи и весь кодогенератор LLVM. Кроме того, в книге сделан акцент на разработку кодогенератора и обсуждаются различные новые возможности, которые можно в нем реализовать, например каркас выбора команд GlobalISel и разработка проходов по машинным функциям.

Предполагаемая аудитория

Эта книга адресована разработчикам компиляторов, энтузиастам и инженерам, интересующимся каркасом LLVM. Она также будет полезна программистам, которые собираются использовать основанные на компиляторе инст-

рументы для анализа улучшения кода, а равно случайным пользователям библиотек LLVM, жаждущим расширить свои знания об основах LLVM. Для лучшего понимания излагаемого в книге материала необходимо владение C++ на среднем уровне.

Структура книги

В главе 1 «Установка LLVM» объясняется, как настроить и использовать среду разработки. По ходу чтения вы откомпилируете библиотеки LLVM и научитесь настраивать процесс сборки под свои нужды.

В главе 2 «Структура компилятора» приводится обзор компонентов компилятора. По ходу дела вы реализуете свой первый компилятор, порождающий промежуточное представление.

В главе 3 «Преобразование исходного файла в абстрактное синтаксическое дерево» вы узнаете, как реализовать фронтальную часть компилятора. Вы напишете фронтальную часть для небольшого языка программирования, и закончится это построением абстрактного синтаксического дерева.

В главе 4 «Основы генерации кода IR» показано, как сгенерировать LLVM IR из абстрактного синтаксического дерева. По ходу дела вы реализуете компилятор демонстрационного языка, который будет выводить код на языке ассемблера или объектные файлы.

В главе 5 «Генерация IR для конструкций языка высокого уровня» показано, как конструкции, встречающиеся в большинстве языков высокого уровня, транслируются в LLVM IR. Вы узнаете о трансляции агрегатных типов данных, различных вариантах реализации наследования классов и виртуальных функций, а также о совместимости с двоичным интерфейсом прикладных программ, принятым в вашей системе.

В главе 6 «Дополнительные сведения о генерации IR» объясняется, как генерировать LLVM IR для предложений обработки исключений исходного языка. Вы также научитесь добавлять метаданные для анализа псевдонимов на основе типов, добавлять отладочную информацию в сгенерированное LLVM IR и расширять генерируемые компилятором метаданные.

В главе 7 «Оптимизация IR» объясняется диспетчер проходов в LLVM. Вы реализуете собственный проход двумя способами: как часть LLVM и в виде плагина. Кроме того, вы узнаете, как добавить свой проход в конвейер оптимизации.

Глава 8 «Язык TableGen» посвящена внутреннему предметно-ориентированному языку LLVM, который называется TableGen. Его назначение – уменьшить объем кода, создаваемого разработчиком, и вы узнаете о различных способах определения данных на языке TableGen и о том, как это можно использовать в кодогенераторе.

В главе 9 «JIT-компиляция» обсуждается использование LLVM для реализации **своевременного**, или **JIT**-, компилятора. По ходу дела вы реализуете свой JIT-компилятор для LLVM IR двумя разными способами.

В главе 10 «Отладка с помощью инструментов LLVM» рассматриваются детали различных библиотек и компонентов LLVM, которые помогают находить ошибки в приложениях. Вы будете использовать контролеры для идентификации переполнения буфера и других ошибок. С помощью библиотеки libFuzzer вы научитесь тестировать функции, подавая им на вход случайные данные, а XRay позволит обнаруживать узкие места в производительности программы. Вы будете использовать статический анализатор clang для поиска ошибок в исходном коде и узнаете, как добавить свой собственный проверщик в анализатор. Вы также научитесь расширять clang своими плагинами.

В главе 11 «Описание целевой архитектуры» объясняется, как добавить поддержку новой процессорной архитектуры. Обсуждаются необходимые и факультативные шаги, в т. ч. определение регистров и команд, разработка механизма выбора команд и поддержка ассемблера и дизассемблера.

В главе 12 «Выбор команд» демонстрируются два подхода к выбору команд. Конкретно, объясняется, как работают DAG выбора и GlobalISel, и показано, как реализовать эту функциональность в целевой архитектуре, опираясь на пример из предыдущей главы. Кроме того, вы научитесь отлаживать и тестировать механизм выбора команд.

В главе 13 «За пределами выбора команд» объясняется, как завершить реализацию кодогенератора, для чего рассматриваются концепции, выходящие за рамки выбора команд. Сюда относится добавление новых машинных проходов для реализации зависящих от целевой архитектуры задач, а также ряд вопросов, не нужных при разработке простого кодогенератора, но представляющих интерес для высокооптимизированных кодогенераторов, например кросс-компиляция для другой процессорной архитектуры.

Что необходимо для чтения этой книги

Вам понадобится компьютер с операционной системой Linux, Windows, Mac OS X или FreeBSD с установленным набором инструментальных средств. Необходимые инструменты перечислены в таблице ниже. Все они должны быть прописаны в пути поиска оболочки.

Программное и аппаратное обеспечение, рассматриваемые в этой книге	**Требования к ОС**
Компилятор C/C++: gcc 7.1.0 или выше, clang 5.0 или выше, Apple clang 10.0 или выше, Visual Studio 2019 16.7 или выше	Linux (любая), Windows, Mac OS X или FreeBSD
CMake 3.20.0 или выше	
Ninja 1.11.1	
Python 3.6 или выше	
Git 2.39.1 или выше	

Для создания пламенной диаграммы в главе 10 нужно будет установить скрипты со страницы https://github.com/brendangregg/FlameGraph. Для запуска

скрипта понадобится также актуальная версия Perl, а для просмотра графа – браузер, поддерживающий отображение SVG-файла (любой современный браузер подойдет). Для просмотра визуализации для Chrome Trace Viewer в той же главе нужно будет установить браузер Chrome.

Если вы читаете электронную версию книги, то мы рекомендуем вводить код самостоятельно или брать его из репозитория на GitHub (ссылка дана в следующем разделе). Это позволит избежать потенциальных ошибок из-за копирования и вставки кода.

Скачивание кода примеров

Код примеров для этой книги можно скачать из GitHub по адресу https://github.com/PacktPublishing/Learn-LLVM-17. Все обновления кода выкладываются в этот репозиторий.

В разделе https://github.com/PacktPublishing/ есть и другие пакеты кода для нашего обширного каталога книг и видео. Не пропустите!

Графические выделения

В этой книге применяется ряд соглашений о наборе текста.

`CodeInText`: код в тексте, имена таблиц базы данных, папок и файлов, расширения имен файлов, пути к файлам, данные, вводимые пользователем. Например: «В коде определяется квантовая схема и переменная `numOnes`».

Отдельно стоящие фрагменты кода набраны так:

```
#include "llvm/IR/IRPrintingPasses.h"
#include "llvm/IR/LegacyPassManager.h"
#include "llvm/Support/ToolOutputFile.h"
```

Желая привлечь внимание к какой-то части кода, мы выделяем ее полужирным шрифтом, например:

```
  switch (Kind) {
// Много других ветвей
  case m88k:            return "m88k";
}
```

Полужирный: новые термины и важные слова, а также части пользовательского интерфейса. Так выделяются команды меню и текст в диалоговых окнах, например: «При разработке на OS X лучше всего установить **Xcode** из магазина Apple».

Так обозначаются советы.

Так обозначаются примечания общего характера.

Часть I

ОСНОВЫ ПОСТРОЕНИЯ С ПОМОЩЬЮ LLVM

В этой части вы научитесь самостоятельно компилировать LLVM и подстраивать процесс сборки под свои нужды. Вы узнаете, как организованы проекты LLVM, и создадите свой первый проект. Наконец, вы познакомитесь с общей структурой компилятора, для чего создадите свой собственный игрушечный компилятор.

Глава 1

Установка LLVM

Чтобы научиться работать с системой LLVM, лучшего всего начать с компиляции ее исходного кода. LLVM представляет собой зонтичный проект, а в репозитории на GitHub хранится исходный код всех входящих в его состав проектов. Каждому проекту LLVM отведен каталог верхнего уровня в репозитории. Одного клонирования репозитория недостаточно, в вашей системе также должны быть установлены все инструментальные средства, необходимые системе сборки. В этой главе будут рассмотрены следующие вопросы:

- подготовка инструментария, необходимого для организации системы сборки;
- клонирование репозитория и сборка из исходного кода; это позволит нам познакомиться с устройством исходного кода LLVM, а также компиляцией и установкой основных библиотек LLVM и clang с помощью CMake и Ninja;
- настройка процесса сборки и возможности влиять на него.

Компиляция LLVM или установка двоичных файлов?

Получить двоичные файлы LLVM можно из разных источников. Если вы работаете в Linux, то дистрибутив уже содержит библиотеки LLVM. Так зачем же тогда компилировать LLVM самостоятельно?

Прежде всего в некоторых установочных пакетах содержатся не все файлы, необходимые для разработки с участием LLVM. Самостоятельная компиляция и установка LLVM позволит избежать таких проблем. Другая причина в том, что LLVM допускает настройку в очень широких пределах. В процессе сборки LLVM вы узнаете, как ее настроить под себя, а это позволит диагностировать проблемы, которые могут возникнуть при переносе вашего приложения LLVM на другую платформу. И наконец, в третьей части книги мы займемся расширением самой LLVM, а уж для этого точно пригодится навык сборки из исходников.

Однако на первых порах вполне можно обойтись и без компиляции LLVM. Если вы хотите пойти по этому пути, то должны будете только установить необходимые инструменты, как описано в следующем разделе.

Во многих дистрибутивах Linux LLVM разбита на несколько пакетов. Обязательно установите пакет разработчика. Например, в Ubuntu следует установить пакет llvm-dev. Также убедитесь, что устанавливаете версию LLVM 17. Для других версий приведенные в книге примеры, возможно, придется модифицировать.

Подготовка среды

Для работы с LLVM ваша система разработки должна работать в одной из операционных систем Linux, FreeBSD, macOS или Windows. Собирать LLVM и clang можно в разных режимах. Для сборки с отладочными символами понадобится до 30 ГБ места на диске. Сколько именно, зависит от заданных параметров сборки. Например, если собираются только основные библиотеки LLVM в выпускном режиме для одной платформы, то потребуется примерно 2 ГБ – это абсолютный минимум.

Чтобы сократить время компиляции, стоит обзавестись быстрым процессором (например, четырехъядерным CPU с тактовой частотой 2.5 Ггц) и быстрым SSD-диском. LLVM можно собрать даже на таком маломощном устройстве, как Raspberry Pi, – только это будет очень долго. Примеры, приведенные в этой книге, разрабатывались на ноутбуке с четырехъядерным Intel CPU с тактовой частотой 2.7 ГГц, 40 ГБ оперативной памяти и SSD-диском емкостью 2.5 ТБ. Такая система вполне подходит для разработки.

В вашей системе разработки должны быть уже установлены определенные программы. Ниже указаны минимальные необходимые версии соответствующих пакетов.

Для извлечения исходного кода из GitHub необходима Git (https://git-scm.com/). Никаких требований к конкретной версии не предъявляется. На страницах справки по GitHub рекомендуется использовать версию не ниже 1.17.10. Но из-за имевших место в прошлом проблем с безопасностью рекомендуется взять самую последнюю версию, каковой на момент написания книги была 2.39.1.

В проекте LLVM в качестве генератора сборочных файлов используется CMake (https://cmake.org/). Необходима версия не ниже 3.20.0. CMake умеет генерировать сборочные файлы для различных систем сборки. В этой книге используется Ninja (https://ninja-build.org/), потому что она работает быстро и доступна на всех платформах. Рекомендуется последняя версия, 1.11.1.

Очевидно, понадобится компилятор C/C++. Проекты LLVM написаны на современном C++, основанном на стандарте C++17. Нужен совместимый с ним компилятор и стандартная библиотека. Известно, что следующие компиляторы работают с LLVM 17:

- gcc 7.1.0 или старше;
- clang 5.0 или старше;

- Apple clang 10.0 или старше;
- Visual Studio 2019 16.7 или старше.

Имейте в виду, что с развитием проекта LLVM требования к компилятору, скорее всего, будут изменяться. Вообще говоря, следует использовать последнюю имеющуюся для вашей системы версию.

Python (https://python.org/) применяется для генерирования сборочных файлов и прогона комплекта тестов. Необходима версия не ниже 3.8.

Хотя в книге это не рассматривается, у вас могут быть причины использовать Make, а не Ninja. В таком случае понадобится **GNU Make** (https://www.gnu.org/software/make/) версии не ниже 3.79. Обе системы сборки применяются очень похоже. Достаточно заменить `ninja` во всех описанных ниже командах на `make`.

LLVM также пользуется библиотекой `zlib` (https://www.zlib.net/). Необходима версия не ниже 1.2.3.4. Как обычно, мы рекомендуем последнюю версию, 1.2.13.

Для установки требуемых компонентов проще всего использовать менеджер пакетов, входящий в состав операционной системы. В следующих разделах приведены команды установки программ для наиболее популярных операционных систем.

Ubuntu

В Ubuntu 22.04 используется менеджер пакетов `apt`. Большинство базовых утилит уже установлены, отсутствуют только инструменты разработки. Для установки всех пакетов сразу выполните команду

```
$ sudo apt -y install gcc g++ git cmake ninja-build zlib1g-dev
```

Fedora и RedHat

Менеджер пакетов в Fedora 37 и RedHat Enterprise Linux 9 называется `dnf`. Как и в Ubuntu, большинство базовых утилит уже установлены. Для установки всех пакетов сразу выполните команду

```
$ sudo dnf -y install gcc gcc-c++ git cmake ninja-build zlib-devel
```

FreeBSD

В FreeBSD 13 и выше следует использовать менеджер пакетов `pkg`. FreeBSD отличается от систем на базе Linux тем, что компилятор clang уже установлен. Для установки всех пакетов сразу выполните команду

```
$ sudo pkg install -y git cmake ninja zlib-ng
```

OS X

Для разработки в OS X лучше всего установить **Xcode** из магазина Apple. Xcode IDE в этой книге не используется, но в ее состав входят необходимые компиляторы C/C++ и вспомогательные утилиты. Для установки остальных инструментов можно использовать менеджер пакетов **Homebrew** (https://brew.sh/). Для установки всех пакетов сразу выполните команду

```
$ brew install git cmake ninja zlib
```

Windows

Как и OS X, Windows не содержит менеджера пакетов. Для получения компилятора C/C++ нужно скачать **Visual Studio Community 2022** (https://visualstudio.microsoft.com/vs/community/), бесплатную для использования в личных целях. Не забудьте установить рабочую нагрузку **Desktop Development with C++**. Для установки остальных пакетов можно использовать менеджер пакетов **Scoop** (https://scoop.sh/). После установки Scoop в соответствии с инструкциями на сайте выполните команду **x64 Native Tools Command Prompt** для VS 2022 из меню Windows. Для установки необходимых пакетов выполните команды

```
$ scoop install git cmake ninja python gzip bzip2 coreutils
$ scoop bucket add extras
$ scoop install zlib
```

Внимательно следите за тем, что печатает Scoop. Для пакетов Python и zlib она рекомендует добавить некоторые разделы реестра. Они необходимы, чтобы другие программы могли найти эти пакеты. Для добавления разделов в реестр лучше всего скопировать и вставить вывод Scoop, который выглядит примерно так:

```
$ %HOMEPATH%\scoop\apps\python\current\install-pep-514.reg
$ %HOMEPATH%\scoop\apps\zlib\current\register.reg
```

После выполнения каждой команды появляется окно сообщений редактора реестра, в котором вас спрашивают, действительно ли вы хотите импортировать этот раздел. Ответьте **Yes**. Теперь все необходимые инструменты установлены.

Во всех примерах в этой книге необходимо использовать консоль **x64 Native Tools Command Prompt** для VS 2022. В этом случае компилятор будет автоматически добавлен в путь поиска.

Кодовая база LLVM очень велика. Чтобы перемещаться по ней комфортно, мы рекомендуем использовать IDE, которая позволяет переходить к определению класса и производить поиск в исходном коде. Нам кажется, что с расширяемой кроссплатформенной IDE **Visual Studio Code** (https://code.visualstudio.com/download) работать очень удобно. Однако для проработки приведенных в книге примеров это необязательно.

Клонирование репозитория и сборка из исходного кода

Подготовив среду, можно приступать к извлечению всех проектов LLVM из GitHub и сборке LLVM. Этот процесс одинаков на всех платформах.

1. Сконфигурировать Git.
2. Клонировать репозиторий.
3. Создать каталог сборки.
4. Сгенерировать сборочные файлы системы.
5. Собрать и установить LLVM.

Начнем с конфигурирования Git.

Конфигурирование Git

В проекте LLVM для управления версиями используется Git. Если вам не доводилось работать с Git раньше, то, перед тем как продолжить, нужно будет задать его базовую конфигурацию: определить имя и электронный почтовый адрес пользователя. То и другое нужно на случай, если бы будете фиксировать изменения. Проверить, заданы ли уже имя и адрес в Git, можно с помощью следующих команд:

```
$ git config user.email
$ git config user.name
```

Эти команды выводят соответственно почтовый адрес и имя пользователя, заданные ранее при использовании Git. Однако если вы раньше этого не делали, то следует ввести показанные ниже команды для задания начальной конфигурации. Просто замените `Jane` своим именем, а `jane@email.org` – своим почтовым адресом:

```
$ git config --global user.email "jane@email.org"
$ git config --global user.name "Jane"
```

Эти команды изменяют глобальную конфигурацию Git. Внутри репозитория Git можно локально переопределить их, опустив параметр `--global`.

По умолчанию Git пользуется редактором **vi** для ввода сообщений, описывающих фиксацию. Если вы предпочитаете другой редактор, можете похожим образом внести изменения в конфигурацию. Например, чтобы использовать редактор **nano**, выполните команду

```
$ git config --global core.editor nano
```

Дополнительные сведения о Git можно найти в книге «Git Version Control Cookbook» (https://www.packtpub.com/product/git-version-control-cookbook-second-edition/9781789137545).

Теперь все готово к клонированию репозитория LLVM из GitHub.