

Компилятор

Материал из Википедии — свободной энциклопедии

Компиля́тор — это программа, которая переводит текст, написанный на языке программирования, в набор машинных кодов.^{[1][2][3]}

Содержание

Основная функциональность и терминология

Виды компиляторов

Виды компиляции

Структура компилятора

Генерация кода

Генерация машинного кода

Генерация байт-кода

Динамическая компиляция

Трансляция байт-кода в машинный код

Декомпиляция

Раздельная компиляция

История

Примечания

Литература

Ссылки

Основная функциональность и терминология

Компиля́ция — сборка программы, включающая:

1. трансляцию всех модулей программы, написанных на одном или нескольких исходных языках программирования высокого уровня и/или языке ассемблера, в эквивалентные программные модули на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера)^{[2][3][4]} или непосредственно на машинном языке или ином двоичнокодовом низкоуровневом командном языке
2. и последующую сборку исполняемой машинной программы, в том числе вставка в программу кода всех функций, импортируемых из статических библиотек и/или генерация кода запроса к ОС на загрузку динамических библиотек, из которых программой функции будут вызываться.

Если компилятор генерирует исполняемую машинную программу на машинном языке, то такая программа непосредственно исполняется физической программируемой машиной (например компьютером). В других случаях исполняемая машинная программа выполняется соответствующей виртуальной машиной.

Входная информация для компилятора есть:

1. на фазе трансляции: исходный код программы, являющийся описанием алгоритма или программы на предметно-ориентированном языке программирования;
2. на фазе компоновки: сгенерированные на фазе трансляции файлы объектных кодов модулей программы, а также файлы объектных кодов статических библиотек и данные о используемых динамических библиотеках.

На выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код^[5], байт-код).

Компилировать — проводить сборку машинной программы, включая:

1. трансляцию с предметно-ориентированного языка на машинно-ориентированный язык^[3],
2. компоновку исполняемой машинно-ориентированной программы из сгенерированных на фазе трансляции объектных модулей — модулей, содержащих части кода программы на машинно-ориентированного кода программы.

Довольно часто компиляторы с языков высокого уровня выполняют лишь трансляцию исходного кода, компоновку же поручая внешнему компоновщику, — компоновщику, представляющему самостоятельную программу, вызываемую компилятором как внешняя подпрограмма. Вследствие этого компилятор многие считают разновидность транслятора, что неверно...

Виды компиляторов

- *Векторизующий*. Базируется на трансляторе, транслирующем исходный код в машинный код компьютеров, оснащённых векторным процессором.
- *Гибкий*. Сконструирован по модульному принципу, управляется таблицами и запрограммирован на языке высокого уровня или реализован с помощью компилятора компиляторов.
- *Диалоговый*. См.: диалоговый транслятор.
- *Инкрементальный*. Пересобирает программу, заново транслируя только изменённые фрагменты программы без перетрансляции всей программы.
- *Интерпретирующий (пошаговый)*. Последовательно выполняет независимую компиляцию каждого отдельного оператора (команды) исходной программы.
- *Компилятор компиляторов*. Транслятор, воспринимающий формальное описание языка программирования и генерирующий компилятор для этого языка.
- *Отладочный*. Устраняет отдельные виды синтаксических ошибок.
- *Резидентный*. Постоянно находится в оперативной памяти и доступен для повторного использования многими задачами.
- *Самокомпилируемый*. Написан на том же языке программирования, с которого осуществляется трансляция.
- *Универсальный*. Основан на формальном описании синтаксиса и семантики входного языка. Составными частями такого компилятора являются: ядро, синтаксический и семантический загрузчики.

Также все компиляторы условно можно разделить на две группы:

- Компиляторы с конкретных языков программирования. (Примеры: GCC, gnat, clang, xcode, gfortran.)
- Компиляторы как системы сборки программ. Таковы например довольно распространенная в UNIX- и Linux-системах система Makefile и распространенная в Windows-системах stake. Работа последних (например в Makefile) очень часто управляется встроенным входным интерпретируемым языком, на котором и прописывается порядок самой компиляции программы.

Виды компиляции

Виды компиляции^[2]:

- *Пакетная*. Компиляция нескольких исходных модулей в одном задании.
- *Построчная*. Машинный код порождается и затем исполняется для каждой завершенной грамматической конструкции языка. Внешне воспринимается как интерпретация, но устройство имеет иное.
- *Условная*. На фазе трансляции результат трансляции зависит от условий, прописанных в исходном транслируемом тексте программы директивами компилятора. (Яркий пример — работа препроцессора языка C и производных от него.) Так, в зависимости от значения некой константы некая транслятор заданную часть транслируемого исходного текста программы транслирует или пропускает (игнорирует).

Структура компилятора

Процесс компиляции состоит из следующих этапов:

1. Трансляция программы — трансляция всех или только изменённых модулей исходной программы.
2. Компоновка машинно-ориентированной программы.

Структурные реализации компилятора могут быть следующими:

1. И транслятор, и компоновщик могут целиком входить в состав компилятора как исполняемые программы.
2. Компилятор сам выполняет лишь трансляцию компилируемой программы, компоновка же программы выполняется вызываемой компилятором отдельной программой-компоновщиком. Практически все современные компиляторы построены по такой схеме.
3. Пакет программ, включающий в себя трансляторы с разных языков программирования и компоновщики.

По первой схеме строились самые первые компиляторы, — для современных компиляторов такая схема построения нехарактерна.

По второй схеме построены все без исключения компиляторы с языков высокого уровня. Любой такой компилятор сам выполняет только трансляцию и далее вызывает компоновщик как внешнюю подпрограмму, который и компоует машинно-ориентированную программу. Такая схема построения легко позволяет компилятору работать и в режиме транслятора с соответствующего языка программирования. Это обстоятельство нередко служит поводом считать компилятор разновидностью транслятора, что естественно неверно, — все современные компиляторы такого

типа все же выполняют компоновку, пусть и силами вызываемого компилятором внешнего компоновщика, тогда как транслятор сам никогда не выполняет вызов внешнего компоновщика. Но это же обстоятельство позволяет компилятору с одного языка программирования на фазе компоновки включать в программу написанную на одном языке программирования функции-подпрограммы из уже оттранслированных соответствующим транслятором/компилятором, написанные на ином языке программирования. Так в программу на C/C++ можно вставлять функции написанные например на Pascal или Fortran. Аналогично и напротив написанная на C/C++ функции могут быть вставлены в Pascal- или Fortran-программу соответственно. Это было бы невозможно без поддержки многими современными компиляторами генерации кода вызова процедур (функций) в соответствии с соглашениями иных языков программирования. Например современные компиляторы с языка Pascal помимо организации вызова процедур/функций в стандарте самого Pascal поддерживают организацию вызова процедурой/функцией в соответствии с соглашениями языка C/C++. (Например чтобы на уровне машинного кода написанная на Pascal процедура/функция работала с входными параметрами в соответствии с соглашениями языка C/C++, — оператор объявления такой Pascal-процедуры/Pascal-функции должен содержать ключевое слово **cdecl**.)

Наконец по третьей схеме построены компиляторы, представляющие собой целые системы, включающие в себя трансляторы с разных языков программирования и компоновщики. Также любой такой компилятор может использовать в качестве транслятора любой способный работать в режиме транслятора компилятор с конкретного языка высокого уровня. Естественно такой компилятор может компилировать программу, разные части исходного текста которой написаны на разных языках программирования. Нередко такие компиляторы управляются встроенным интерпретатором того или иного командного языка. Яркий пример таких компиляторов — имеющийся во всех UNIX-системах (в частности в Linux) компилятор make.

Трансляция программы как неотъемлемая составляющая компиляции включает в себя:

1. Лексический анализ. На этом этапе последовательность символов исходного файла преобразуется в последовательность лексем.
2. Синтаксический (грамматический) анализ. Последовательность лексем преобразуется в дерево разбора.
3. Семантический анализ. На этой фазе дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их объявлениям, типам данных, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением/кодом», и может быть дополненным деревом разбора, новым деревом, абстрактным набором команд или чем-то ещё, удобным для дальнейшей обработки.
4. Оптимизация. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах — например, над промежуточным кодом или над конечным машинным кодом.
5. Генерация кода. Из промежуточного представления порождается код на целевом машинно-ориентированном языке.

Генерация кода

Генерация машинного кода

Большинство компиляторов переводит программу с некоторого высокоуровневого языка программирования в машинный код, который может быть непосредственно выполнен физическим процессором. Как правило, этот код также ориентирован на исполнение в среде конкретной операционной системы, поскольку использует предоставляемые ею возможности (системные вызовы, библиотеки функций). Архитектура (набор программно-аппаратных средств), для которой компилируется (собирается) машинно-ориентированная программа, называется *целевой машиной*.

Результат компиляции — исполнимый программный модуль — обладает максимально возможной производительностью, однако привязан к конкретной операционной системе (семейству или подсемейству ОС) и процессору (семейству процессоров) и не будет работать на других.

Для каждой целевой машины (IBM, Apple, Sun, Эльбрус и т. д.) и каждой операционной системы или семейства операционных систем, работающих на целевой машине, требуется написание своего компилятора. Существуют также так называемые кросс-компиляторы, позволяющие на одной машине и в среде одной ОС генерировать код, предназначенный для выполнения на другой целевой машине и/или в среде другой ОС. Кроме того, компиляторы могут оптимизировать код под разные модели из одного семейства процессоров (путём поддержки специфичных для этих моделей особенностей или расширений наборов команд). Например, код, скомпилированный под процессоры семейства Pentium, может учитывать особенности распараллеливания инструкций и использовать их специфичные расширения — MMX, SSE и т. п.

Некоторые компиляторы переводят программу с языка высокого уровня не прямо в машинный код, а на язык ассемблера. (Пример: PureBasic, транслирующий бейсик-код в ассемблер FASM.) Это делается для упрощения части компилятора, отвечающей за генерацию кода, и повышения его переносимости (задача окончательной генерации кода и привязки его к требуемой целевой платформе перекладывается на ассемблер), либо для возможности контроля и исправления результата компиляции (в том числе ручной оптимизации) программистом.

Генерация байт-кода

Результатом работы компилятора может быть программа на специально созданном низкоуровневом языке двоично-кодowych команд, выполняемых виртуальной машиной. Такой язык называется псевдокодом или байт-кодом. Как правило, он не есть машинный код какого-либо компьютера и программы на нём могут исполняться на различных архитектурах, где имеется соответствующая виртуальная машина, но в некоторых случаях создаются аппаратные платформы, напрямую выполняющие псевдокод какого-либо языка. Например, псевдокод языка Java называется байт-кодом Java и выполняется в Java Virtual Machine, для его прямого исполнения была создана спецификация процессора picoJava. Для платформы .NET Framework псевдокод называется Common Intermediate Language (CIL), а среда исполнения — Common Language Runtime (CLR).

Некоторые реализации интерпретируемых языков высокого уровня (например, Perl) используют байт-код для оптимизации исполнения: затратные этапы синтаксического анализа и преобразование текста программы в байт-код выполняются один раз при загрузке, затем соответствующий код может многократно использоваться без перекомпиляции.

Динамическая компиляция

Из-за необходимости интерпретации байт-код выполняется значительно медленнее машинного кода сравнимой функциональности, однако он более переносим (не зависит от операционной системы и модели процессора). Чтобы ускорить выполнение байт-кода, используется *динамическая компиляция*,

когда виртуальная машина транслирует псевдокод в машинный код непосредственно перед его первым исполнением (и при повторных обращениях к коду исполняется уже скомпилированный вариант).

Наиболее популярной разновидностью динамической компиляции является JIT. Другой разновидностью является инкрементальная компиляция.

CIL-код также компилируется в код целевой машины JIT-компилятором, а библиотеки .NET Framework компилируются заранее.

Трансляция байт-кода в машинный код

Трансляция байт-кода в машинный код специальным транслятором байт-кода как указано выше неотъемлемая фаза динамической компиляции. Но трансляция байт-кода применима и для простого преобразования программы на байт-коде в эквивалентную программу на машинном языке. В машинный код может транслироваться как заранее скомпилированный байт-код. Но также трансляция байт-кода в машинный код может выполняться компилятором байт-кода сразу следом за компиляцией байт-кода. Практически всегда в последнем случае трансляция байт-кода выполняется внешним транслятором, вызываемым компилятором байт-кода.

Декомпиляция

Существуют программы, которые решают обратную задачу — перевод программы с низкоуровневого языка на высокоуровневый. Этот процесс называют декомпиляцией, а такие программы — декомпиляторами. Но поскольку компиляция — это процесс с потерями, точно восстановить исходный код, скажем, на C++, в общем случае невозможно. Более эффективно декомпилируются программы в байт-кодах — например, существует довольно надёжный декомпилятор для Flash. Разновидностью декомпиляции является дизассемблирование машинного кода в код на языке ассемблера, который почти всегда благополучно выполняется (при этом сложность может представлять само модифицирующий код или код, в котором собственно код и данные не разделены). Связано это с тем, что между кодами машинных команд и командами ассемблера имеется практически взаимно-однозначное соответствие.

Раздельная компиляция

Раздельная компиляция (англ. *separate compilation*) — трансляция частей программы по отдельности с последующим объединением их компоновщиком в единый загрузочный модуль^[2].

Исторически особенностью компилятора, отражённой в его названии (англ. *compile* — собирать вместе, составлять), являлось то, что он производил как трансляцию, так и компоновку, при этом компилятор мог порождать сразу машинный код. Однако позже, с ростом сложности и размера программ (и увеличением времени, затрачиваемого на перекомпиляцию), возникла необходимость разделять программы на части и выделять библиотеки, которые можно компилировать независимо друг от друга. В процессе трансляции программы сам компилятор или вызываемый компилятором транслятор порождает объектный модуль, содержащий дополнительную информацию, которая потом — в процессе компоновки частей в исполнимый модуль — используется для связывания и разрешения ссылок между частями программы. Раздельная компиляция также позволяет писать разные части исходного текста программы на разных языках программирования.

Появление отдельной компиляции и выделение компоновки как отдельной стадии произошло значительно позже создания компиляторов. В связи с этим вместо термина «компилятор» иногда используют термин «транслятор» как его синоним: либо в старой литературе, либо когда хотят подчеркнуть его способность переводить программу в машинный код (и наоборот, используют термин «компилятор» для подчёркивания способности собирать из многих файлов один). Вот только использование в таком контексте терминов «компилятор» и «транслятор» неправильно. Даже если компилятор выполняет трансляцию программы самостоятельно, поручая компоновку вызываемой внешней программе-компоновщику, такой компилятор не может считаться разновидностью транслятора, — транслятор выполняет трансляцию исходной программы и только. И уж тем более не являются трансляторами компиляторы вроде системной утилиты-компилятора **make**, имеющейся во всех UNIX-системах.

Собственно утилита **make** — яркий пример довольно удачной реализации отдельной компиляции. Работа утилиты **make** управляется сценарием на интерпретируемом утилитой входном языке, известном как **makefile**, содержащемся в задаваемом при запуске утилиты входном текстовом файле. Сама утилита не выполняет ни трансляцию, ни компоновку — де-факто утилита **make** функционирует как диспетчер процесса компиляции, организующий компиляцию программы в соответствии с заданным сценарием. В частности в ходе компиляции целевой программы утилита **make** вызывает трансляторы с языков программирования, транслирующие разные части исходной программы в объектный код, и уже после этого вызывается тот или иной компоновщик, компоновщик конечный исполняемый программный или библиотечный программный модуль. При этом разные части программы, оформляемые в виде отдельных файлов исходного текста, могут быть написаны как на одном языке программирования, так и на разных языках программирования. В процессе перекомпиляции программы транслируются только изменённые части-файлы исходного текста программы, вследствие чего длительность перекомпиляции программы значительно (порой на порядок) сокращается.

История

На заре развития компьютеров первые компиляторы (трансляторы) называли «программирующими программами»^[6] (так как в тот момент программой считался только машинный код, а «программирующая программа» была способна из человеческого текста сделать машинный код, то есть запрограммировать ЭВМ).


Примечания

1. ГОСТ 19781-83 // Вычислительная техника. Терминология: Справочное пособие. Выпуск 1 / Рецензент канд. техн. наук Ю. П. Селиванов. — М.: Издательство стандартов, 1989. — 168 с. — 55 000 экз. — ISBN 5-7050-0155-X.; см. также ГОСТ 19781-90
2. Першиков, 1991.
3. Вычислительная техника.
4. Борковский А. Б. Англо-русский словарь по программированию и информатике (с толкованиями). — М.: Русский язык, 1990. — 335 с. — 50 050 (доп.) экз. — ISBN 5-200-01169-3.
5. Толковый словарь по вычислительным системам = Dictionary of Computing / Под ред. В. Иллингворта и др.: Пер. с англ. А. К. Белоцкого и др.; Под ред. Е. К. Масловского. — М.: Машиностроение, 1990. — 560 с. — 70 000 (доп.) экз. — ISBN 5-217-00617-X (СССР), ISBN 0-19-853913-4 (Великобритания).
6. Н. А. Криницкий, Г. А. Миронов, Г. Д. Фролов. Программирование / Под ред. М. Р. Шура-Бура. — М.: Государственное издательство физико-математической литературы, 1963.

Литература

- Вычислительная техника. Терминология. Указ. соч.
- *Першиков В. И., Савинков В. М.* Толковый словарь по информатике / Рецензенты: канд. физ.-мат. наук А. С. Марков и д-р физ.-мат. наук И. В. Поттосин. — М.: Финансы и статистика, 1991. — 543 с. — 50 000 экз. — ISBN 5-279-00367-0.
- *Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман.* Компиляторы: принципы, технологии и инструментарий = *Compilers: Principles, Techniques, and Tools*. — 2-е изд. — М.: Вильямс, 2010. — 1184 с. — ISBN 978-5-8459-1349-4.
- *Робин Хантер.* Основные концепции компиляторов = *The Essence of Compilers*. — М.: Вильямс, 2002. — 256 с. — ISBN 0-13-727835-7.
- *Хантер Р.* Проектирование и конструирование компиляторов / Пер. с англ. С. М. Круговой. — М.: Финансы и статистика, 1984. — 232 с.
- *Д. Креншоу.* Давайте создадим компилятор! (<http://www.kulichki.net/kit/crenshaw/crenshaw.html>)
- *Серебряков В. А., Галочкин М. П.* Основы конструирования компиляторов (<http://www.citforum.ru/programming/theory/serebryakov/>).

Ссылки

-
- [Compilers \(http://curlie.org/Computers/Programming/Compilers/\)](http://curlie.org/Computers/Programming/Compilers/) в каталоге ссылок [Open Directory Project \(dmoz\)](#)
- [Новым процессорам — новые компиляторы \(http://www.osp.ru/os/2013/01/13033990/\)](http://www.osp.ru/os/2013/01/13033990/)
- [Textbook: Compiler Design: Theory, Tools, and Examples \(http://elvis.rowan.edu/~bergmann/books/\)](http://elvis.rowan.edu/~bergmann/books/).
- *В. Э. Карнов.* Классическая теория компиляторов (<http://rema44.ru/resurs/study/compiler1/Compiler1.pdf>) 

Источник — <https://ru.wikipedia.org/w/index.php?title=Компилятор&oldid=113659232>

Эта страница в последний раз была отредактирована 17 апреля 2021 в 18:18.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike; в отдельных случаях могут действовать дополнительные условия.

Wikipedia® — зарегистрированный товарный знак некоммерческой организации Wikimedia Foundation, Inc.