



Проектирование компиляторов

Лекция 1.
Введение в компиляторы.

Маркин А. Л.
alexanius@gmail.com

2021

О курсе

Курс посвящён внутреннему устройству оптимизирующих компиляторов.

План курса:

- ▶ 13 лекций про общее устройство компиляторов и технологии оптимизации кода.
- ▶ 6 лабораторных работ в рамках которых будет написан собственный макет компилятора.
- ▶ 1 зачёт по всему курсу. Билет содержит 2 вопроса, ответ устный, но можно подготовиться.

О курсе

Критерии итоговой оценки:

- ▶ Допуск к зачёту возможен только при всех сданных лабораторных работах.
- ▶ Регулярная уместная активность на лекциях даёт +1 балл на зачёте.
- ▶ Треть средней оценки за ЛР прибавляется к оценке по зачёту.

Итого: хорошее выполнение практических заданий и активность на лекциях обеспечивает +4 балла просто так!

О курсе

Практические задания:

- ▶ В течении занятия по ЛР объясняется суть задания, устанавливаются требования к выполнению. Непосредственно выполнение задания производится в любое удобное время.
- ▶ Выполненное задание присыпается на почту преподавателю, после чего производится его проверка. При отсутствии ошибок и других замечаний можно приступать к написанию отчёта.
- ▶ Отчёт — записка в свободной форме (в формате .odt или .pdf), содержащая постановку задачи, детали выполнения, результаты и вывод.
- ▶ Задание считается выполненным если оно проходит тестирование, а также когда принят отчёт о его выполнении.

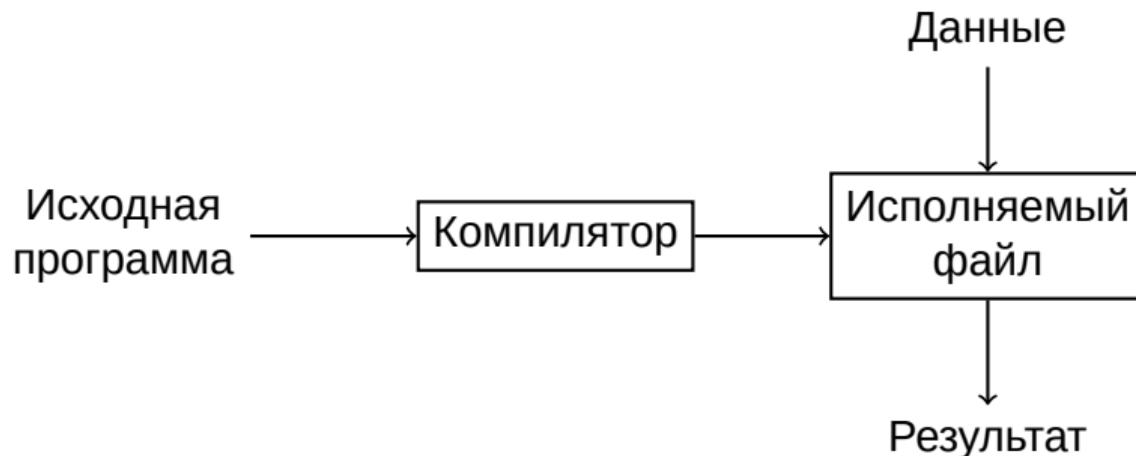
О курсе

Как не надо делать:

- ▶ Первое зафиксированное списывание практической работы лишает бонуса на экзамене.
- ▶ Каждое последующее зафиксированное списывание снижает оценку за зчёт на 1 балл.
- ▶ Списанные практические задания не принимаются.
- ▶ Удалённый формат проведения зчёта позволяет легко списать ответ. Но преподаватель знает все источники, а спианный ответ порождает много интересных дополнительных вопросов на понимание материала.

Введение

Компилятор



Компилятор — программа, принимающая на вход текст на одном (исходном) языке программирования, и возвращающая текст на другом (целевом) языке.

Компилятор

Современные компиляторы:

- ▶ GCC (GNU Compiler Collection) — набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU;
- ▶ LLVM (Low Level Virtual Machine) — универсальная система анализа, трансформации и оптимизации программ, реализующая виртуальную машину с RISC-подобными инструкциями;
- ▶ Intel C++ compiler — оптимизирующий компилятор, разрабатываемый фирмой Intel для процессоров семейств x86, x86-64 и IA-64;
- ▶ Elbrus Compiler Collection — оптимизирующий компилятор, разрабатываемый фирмой МЦСТ для процессоров семейств Эльбрус и Sparc;

Интерпретатор



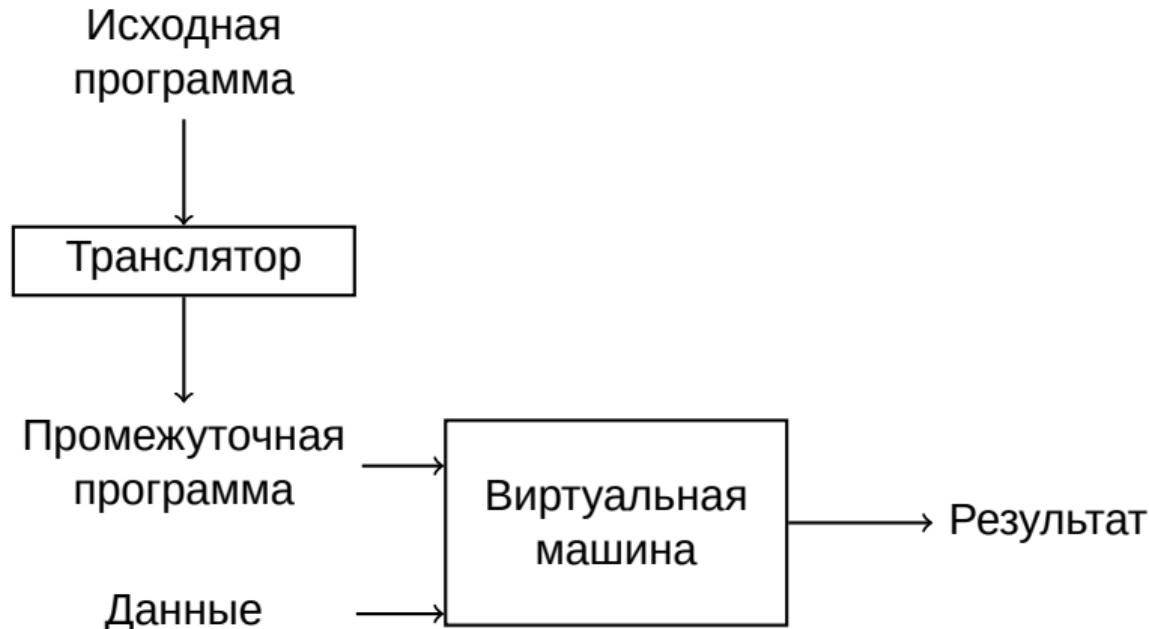
Интерпретатор — программа, принимающая на вход текст на одном (исходном) языке программирования, и выполняющая поданные на вход операции;

Интерпретатор

Современные интерпретаторы:

- ▶ Bash — командный процессор, работающий в интерактивном режиме. Используется для UNIX-подобных систем, в т.ч. для GNU/Linux;
- ▶ CPython — основной интерпретатор языка Python. Транслирует программу в промежуточное представление, а затем исполняет её.
- ▶ PHP — распространённый язык для создания динамических интернет-страниц.

Динамический компилятор



Динамический компилятор (JIT, Just-In-Time) — программа, принимающая на вход текст на одном (исходном) языке программирования, транслирующая его в байт-код, исполняемый на виртуальной машине.

Динамический компилятор

Современные динамические компиляторы:

- ▶ JVM (Java Virtual Machine) — виртуальная машина для исполнения байт-кода языка Java (а также Clojure, Groovy, Scala и т.д.);
- ▶ CLR (Common Language Runtime) — виртуальная машина стека Microsoft .NET.
- ▶ V8 — виртуальная машина для исполнения JavaScript кода от компании Google.

Схема работы компилятора

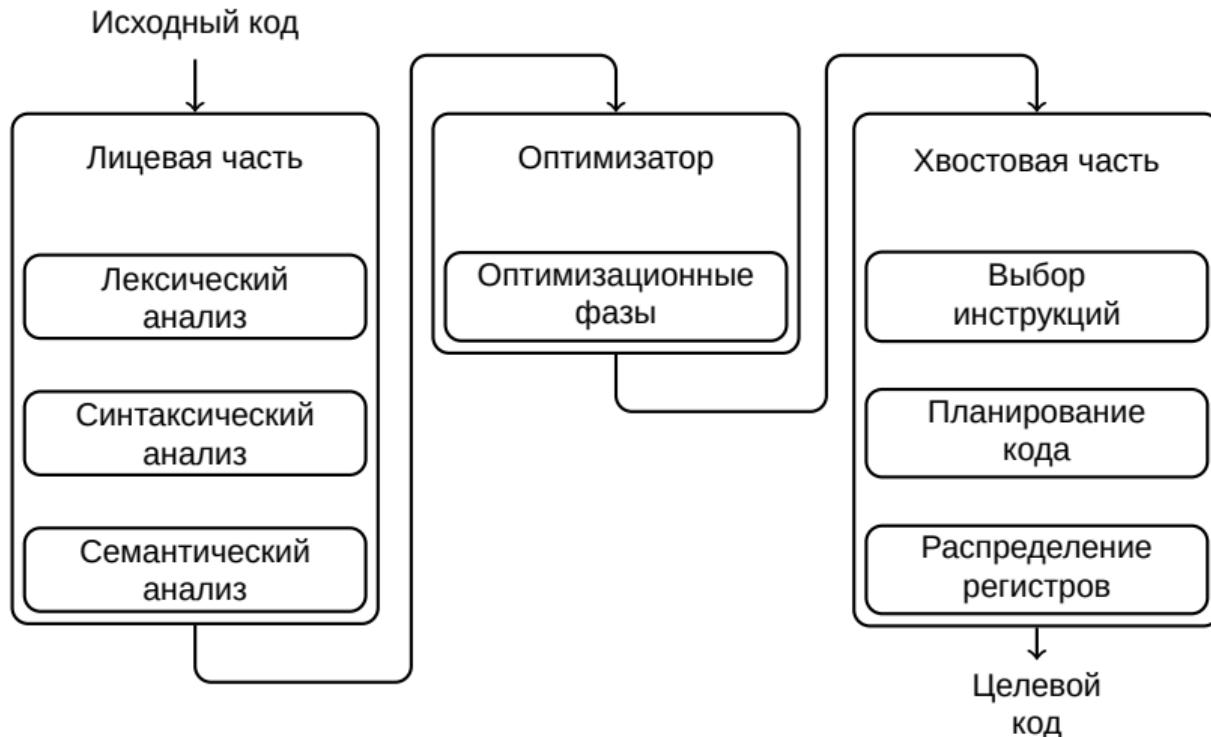


Схема работы компилятора

Лицевая часть проводит первоначальный разбор входной программы и преобразование её к структурированному виду:

- ▶ **Лексический анализ** — преобразование потока символов в поток классифицированных слов.
- ▶ **Синтаксический анализ** — определение принадлежности входного потока данному языку, построение дерева программы.
- ▶ **Семантический анализ** — создание таблицы символов и простановка типов объектов и выражений, более строгие проверки на соответствие правилам языка.

Схема работы компилятора

Оптимизатор проводит анализы и преобразования программы с целями её ускорения, проверок на ошибки, уменьшения размера:

- ▶ **Оптимизационная фаза** — подпрограмма, анализирующая или модифицирующая входную программу

Схема работы компилятора

Хвостовая часть проводит преобразование программы из промежуточного представления компилятора к целевому языку:

- ▶ **Выбор инструкций** — преобразование инструкций промежуточного представления в инструкции целевой машины.
- ▶ **Распределение регистров** — преобразование кода для использования конечного числа регистров.
- ▶ **Планирование** — перестановка инструкций для ускорения исполнения целевой программы.

Схема работы компилятора

Что происходит при вызове программы gcc:

1. **cc1** — непосредственно компилятор. Преобразует входной файл в ассемблер
2. **as** — ассемблер. Преобразует файл с ассемблером в объектный файл
3. **collect2** — компоновщик. Производит связывание объектных файлов в итоговую программу.

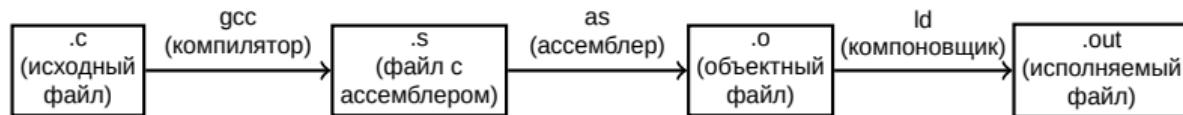


Схема работы компилятора

Дополнительные инструменты, необходимые для полноценной разработки помимо компилятора (некоторые поставляются вместе с ним, но чаще всего это отдельные проекты):

- ▶ интерпретатор промежуточного кода
- ▶ реализация стандартной библиотеки языка
- ▶ дополнительные библиотеки
- ▶ отладчик
- ▶ компоновщик
- ▶ профилировщик
- ▶ анализатор покрытия
- ▶ анализатор производительности
- ▶ прочие инструменты

Раскрутка компилятора

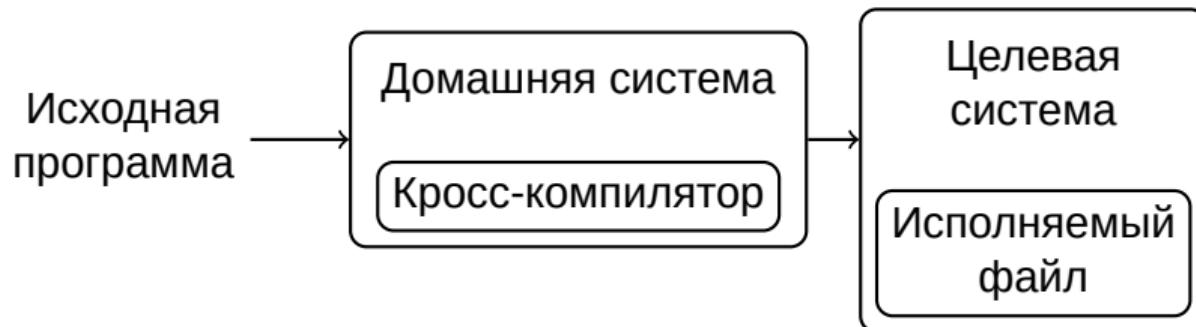
Как создать компилятор нового языка L ?

1. На одном из существующих языков реализовать подмножество языка $L_0 : L_0 \subset L$.
2. На созданном языке L_0 создать компилятор языка L_0 .
3. Начать расширение компилятора L_0 до языка L .

Такой подход к созданию новых языков называется **раскруткой** (bootstrapping).

Кросс-компиляция

Для случаев когда архитектура машины, на которой будет исполняться программа отличается от архитектуры на которой она собирается, применяется технология кросс-компиляции:



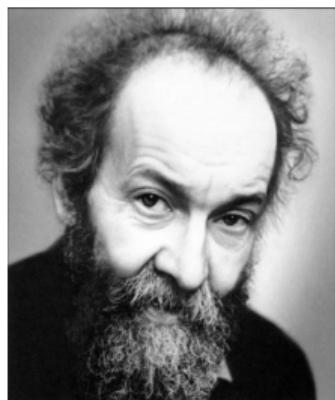
Домашняя система (host) — система, на которой работает компилятор.

Целевая система (target) — система, на которой будет исполняться полученная программа.

История появления компиляторов и языков программирования

История появления языков программирования

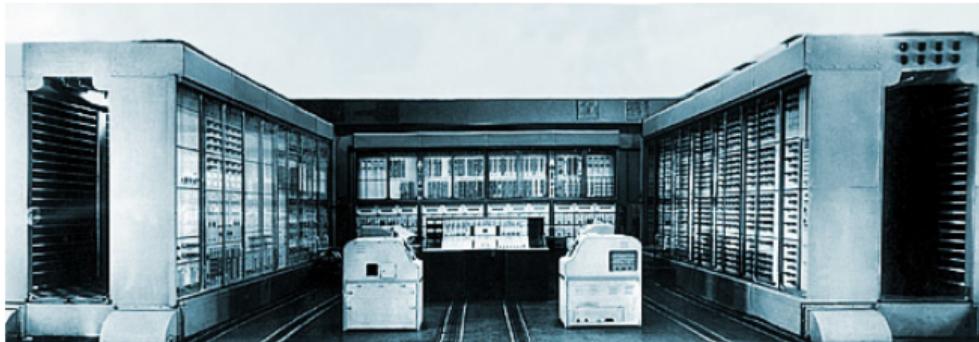
- 1952 А.А. Ляпунов впервые читает курс по программированию в МГУ, в котором формирует понятие «операторного метода» — удобной абстракции для формализации алгоритма.
- 1953 На основе операторного метода Ляпунов выводит задачу автоматизации программирования. Эти его труды послужили толчком к созданию первых трансляторов и становлению программирования как науки.



Алексей Андреевич Ляпунов, 1911-1973 гг.
Один из основоположников кибернетики

История появления языков программирования

1954 На основе курса Ляпунова, С.С. Камынин и Э.З. Любимский создают первый транслятор ПП-1 (Программирующая программа - 1) для машины «Стрела».



ЭВМ «Стрела-1», 1953 г.

История появления языков программирования

- 1954 На основе ПП-1, М.Р. Шура-Бура создаёт промышленную систему трансляции, названную ПП-2. Постепенно на основе этого транслятора для машины «Стрела» развивались и другие средства программирования: компоновщики, загрузчики, отладчики, редакторы.
- 1956 Параллельно с ПП-2, А.П. Ершов начинает разработку ПП для БЭСМ, и по завершении у него получается первый транслятор с высокоуровневого языка.



Андрей Петрович Ершов, 1931-1988 гг.
Пионер теоретического и системного
программирования.



Михаил Романович Шура-Бура,
1918-2008 гг.
Внёс огромный вклад в развитие
программирования.

История появления языков программирования

- 1963 Независимые команды создают трансляторы языка Алгол. Первыми появились проекты ТА-1 и ТА-2. Это первый момент пересечения развития советского и западного системного программного обеспечения.
- 1964 Под руководством А.П. Ершова создаётся «Альфа» — транслятор с языка Алгол, впервые использующий оптимизирующую компиляцию.
- 1966 Создаётся язык «АЛМО» — язык, реализующий работу на байткоде и поддерживающий технологию раскрутки.

Основы проектирования языков и компиляторов уже сложились к этому моменту. Дальнейшее их развитие не менее интересно, и является предметом изучения отдельного учебного курса.

История появления языков программирования

Зарубежная линия развития языков и компиляторов:

1952 Первый компоновщик (часто считается первым компилятором) A-0 system (Arithmetic Language version 0) был создан в 1952 году Грейс Хоппер (Grace Hopper) для машины UNIVAC I (первый коммерческий компьютер в США).



Грейс Хоппер, 1906-1992 гг.
Разработчик первых компиляторов и
высокоуровневых языков программирования.

История появления языков программирования

- 1953 Первый высокоуровневый язык Speedcoding для машины IBM 701, созданный Джоном Бэкусом (John Backus). Программы на этом языке работали в 10 - 20 раз медленнее чем написанные в машинных кодах.
- 1954 IBM создаёт первый массовый компьютер с поддержкой плавающих операций IBM 704.



ЭВМ «IBM-704»

Проблема: стоимость разработки ПО начинает превышать стоимость аппаратуры

История появления языков программирования

- 1954 Бэкус с командой представляет черновик спецификации The IBM Mathematical Formula Translating System.
- 1957 Представлен первый компилятор языка FORTRAN. Этот язык получил широкое распространение и является первым массовым высокоуровневым языком программирования.



Джон Бэкус, 1924-2007 гг.
Разработчик первых высокоуровневых языков
программирования, знаковый учёный в области
информатики.

История появления языков программирования

- 1958 Джон Мак Карти (John McCarthy) язык LISP, основанный на λ -счислении Алонзо Чёрча (Alonzo Church). Lisp считается вторым по старшинству высокоуровневым языком. Он является первым языком с автоматическим управлением памятью и сборкой мусора.
- 1959 Основываясь на работах Хоппер создан язык COBOL, применяющийся в бизнес приложениях. Он считается третьим старейшим высокоуровневым языком программирования.



Джон Маккарти, 1927-2011 гг.

Разработчик первых высокоуровневых языков программирования, знаковый учёный в области информатики, автор термина «искусственный интеллект».

История появления языков программирования

1973 В лабораториях Bell Labs Денис Ритчи и Кен Томпсон создали язык программирования Си.



Денис Ритчи, 1941-2011 гг.

Создатель языков программирования B, BCPL, C, участвовал в разработке ОС Multics и Unix.



Кен Томпсон, 1943 г.

Создатель языка Си и ОС Unix.

Заключение

Мотивация

Почему мы учим компиляторы:

- ▶ Техники компиляции нужны везде:
 - ▶ «Одна из самых умных вещей, которые я увидел в Doom, это использование их лексического анализатора и парсера по всей программе».
 - ▶ Чтение конфигурационных файлов.
 - ▶ Создание собственных языков.
 - ▶ Обработка данных.
- ▶ Понимание принципов работы машин:
 - ▶ Создание компилятора требует не только понимания языка программирования, но и системы команд целевой машины, принципов работы процессора, памяти, иногда периферии, взаимодействия с ОС.

Мотивация

Почему мы учим компиляторы:

- ▶ Повышение собственной квалификации:
 - ▶ Различные конструкции языка по-разному транслируются в целевой код и имеют разную скорость исполнения. Знание этих особенностей позволяет писать более эффективный код
 - ▶ В компиляторах применяется широкий спектр алгоритмов из разных областей. Их изучение позволит проще понимать другие области информатики.
- ▶ Это интересно:
 - ▶ Компиляторы находятся на стыке теоретической и практической информатики.
 - ▶ Реализация своего языка программирования - это круто!

Связь

Преподаватель: Маркин Алексей Леонидович

Почта: alexanius@gmail.com

Литература

- ▶ *A. B. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман* Компиляторы: принципы, технологии и инструменты.
- ▶ *С. З. Свердлов* Конструирование компиляторов.
- ▶ *K. D. Cooper, L. Torczon* Engineering a compiler.
- ▶ *A. W. Appel* Modern compiler implementation in C.
- ▶ *D. Grune, H. E. Bal, C. J. H. Jacobs, K. G. Langendoen* Modern Compiler Design.
- ▶ *C. Lattner* The Architecture of Open Source Applications. LLVM.
- ▶ *Н. Вирт* Построение компиляторов.
- ▶ *Н. Вирт, Ю. Гуткнхт* Разработка операционной системы и компилятора. Проект Оберон.
- ▶ *А. П. Ершов, М. Р. Шура-Бура* Становление программирования в СССР (начальное развитие).

Учебные курсы

- ▶ CS 143 — Compilers, Stanford University
- ▶ COMP 412 — Compiler Construction for Undergraduates, Rice University
- ▶ CS 553 — Programming Language Design and Implementation, Colorado State University
- ▶ CS 412/413 — Introduction to Compilers, Cornell University
- ▶ CS352 — Compilers: Principles and Practice, Purdue University