Introducing Software Compilers Laboratory

Ettore Speziale

Politecnico di Milano

Contents

Contents

Topics

In this lessons we will see:

- how theoretical concepts (e.g. regular expressions) are exploited in compiler development
- how a compiler is internally organized and how it works
- how to modify a simple compiler

Some concepts can be applied in everyday work.

Exam

The lab is $\frac{3}{5}$ of the exam score:

- you need to pass the lab exam in order to pass the whole exam

Lab exam is composed by two homeworks:

- one about compiler front-end
- the other about compiler back-end
- performed in group of 4 people

Deadline is course last lesson.

Contents

Basic Assumptions

This is a basic course about advanced topics, so we require:

- a good knowledge of C language
- usage of compiler-related tools (e.g. `gcc`, `make`, ...)
- usage of a versioning tool (e.g. `mercurial`)
- your brain

A Tiny and Nice Compiler I

Compiler purpose is:

- translating a program written with language L_0 into a semantically equivalent program expressed with language L_1

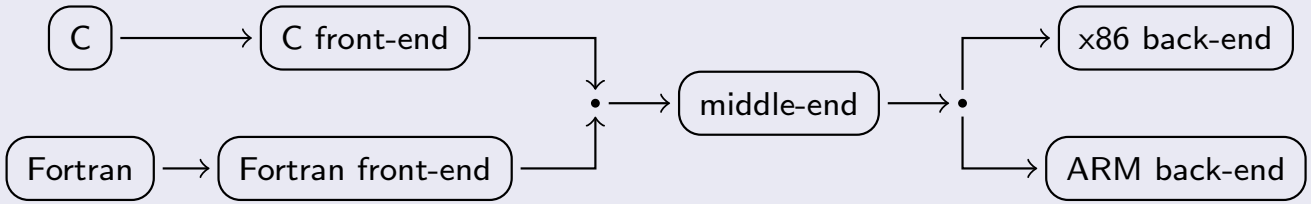
A compiler is organized like a pipeline:

- each stage applies transformation to the input program producing an output program

A Tiny and Nice Compiler II

A simple compiler contains at least three stages:

Simple compiler structure



Different stages for different purposes:

front-end abstract from the hardware

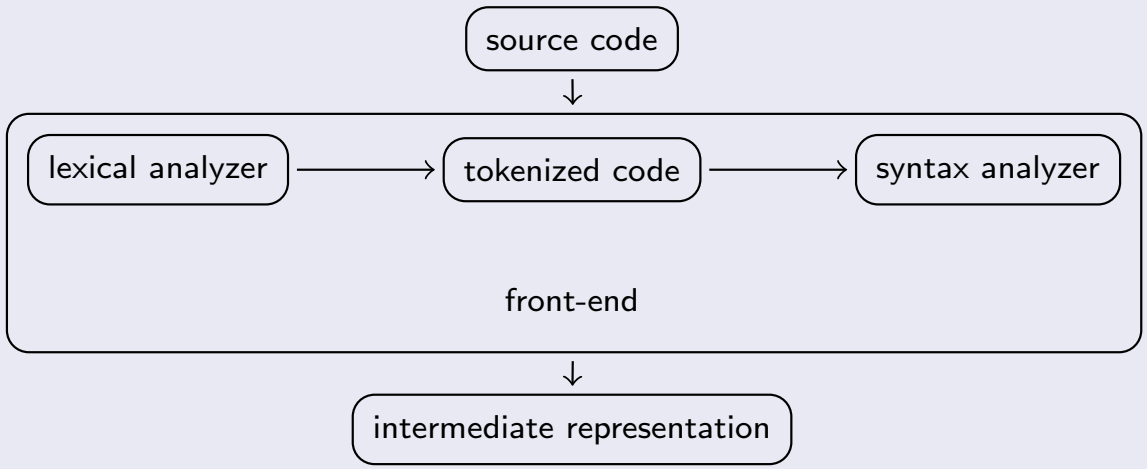
middle-end abstract from both high-level language and hardware

back-end abstract from the high-level language

Front-end

Front-end purpose is to translate code into a *intermediate form*.

Front-end structure



Main actions:

- recognize language constructs
- find syntax error

Back to Real World: GCC

Many front-ends:

- most of them target the *TREE* language

Common lowering to intermediate representation:

- *GIMPLE* and *GIMPLE-SSA* languages

At last:

- translation to *RTL* language
- back-ends emit native instructions

The dark side:

- language hooks

Contents

Think First

We will see very few-concepts:

- tokens
- statements
- control structures
- ...

You already know *how to use* them:

- you only need to understand how to *recognize* and *compile* them

Many statements are just a variation of a common idiom:

- *syntactic sugar* around a concept

UNIX is your friend

Every UNIX-derived OS contains a lot of compiler-related tools:

- to automate compilers development
- to automate tedious tasks

Few will works on compilers, almost all, soon or later, will find a tedious task:

- count the occurrences of a pattern
- substitute a parametric sentence with another
- ...

Tools (`grep`, `sed`, `awk`) can automate your work!

Contents

Bibliography

Formal Languages and Compilers Group.
Software Compilers.
http://compilergroup.elet.polimi.it, 2010.