1. Introduction

Why studying compilers construction

Andrea Polini

Formal Languages and Compilers
Master in Computer Science
University of Camerino

ToC

WARNING

Slides are distributed to help students in their preparation to the exam. In **no way** they intend to substitute text books. Instead a **thorough study of text books** constitutes the **most wise strategy** to maximize the chances to pass the final exam.

ToC

Teacher and Course

- Andrea Polini
  - e-mail: andrea.polini@unicam.it
  - web: http://www.cs.unicam.it/polini/
- Formal Languages and Compilers
  - lessons:
    - Tuesday 11am to 1pm
    - Wednesday 3pm to 5pm
  - web: http://didattica.cs.unicam.it/...
- Exam dates:
  - July 5th and 19th, 2017 – 11am-1pm – room AB1
  - September 6th and 20th, 2017 – 11am-1pm – room AB1
  - February 7th and 21st, 2017 – 11am-1pm – room AB1

Course Objective

At the end of the course:

- you will know the basic theories and methodologies behind the construction of a compiler
- you should be able to understand the basic issues related to compilers construction
- you should have acquired basic skills to develop a compiler/transformer for a simple language

Study material

- **Reference book**:

  
  Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman
  Compilers – Principles, Techniques and Tools, 2nd Ed.
  Addison-Wesley, 2007.

  
  Terence Parr
  The Definitive ANTLR4 Reference
  The Pragmatic Programmers, 2012.

- **Further references possibly provided by the teacher**

Final Exam!!!

1. Group project (2 members)
   - You will be asked to develop a compiler for a simplified language using the ANTLR4 parser generator
2. Written paper – date fixed for the exam
   - The paper will contain exercises that ask to the student to solve problems not solved during classes, or questions on more theoretical aspects. Instead during lessons we will discuss solutions to similar exercises.
3. [Oral paper] – student choice

Correct steps

Students have to follow the order. (1) They deliver the project sending the workspace in zip format to the teacher (at least one week before the exam date). (2) They come to the first exam date. If they pass the exam (minimum mark 16) (3) they come to discuss the project (date fixed by the teacher). In case they would like to do the oral paper (4) they can do it the same day.

ToC

Compilers vs. Interpreters

Two approaches to permit the execution of a program, written using an high level language, on a physical machine:

- Compilers: use of a program that can read a program in one language (source) and translate it into an **equivalent** program in another language (target)
- Interpreters: use of a program that takes in input the program and data and run the program on the data without the need to make an explicit translation into the machine code

Java?

Compilers vs. Interpreters

Two approaches to permit the execution of a program, written using an high level language, on a physical machine:

- Compilers: use of a program that can read a program in one language (source) and translate it into an **equivalent** program in another language (target)
- Interpreters: use of a program that takes in input the program and data and run the program on the data without the need to make an explicit translation into the machine code

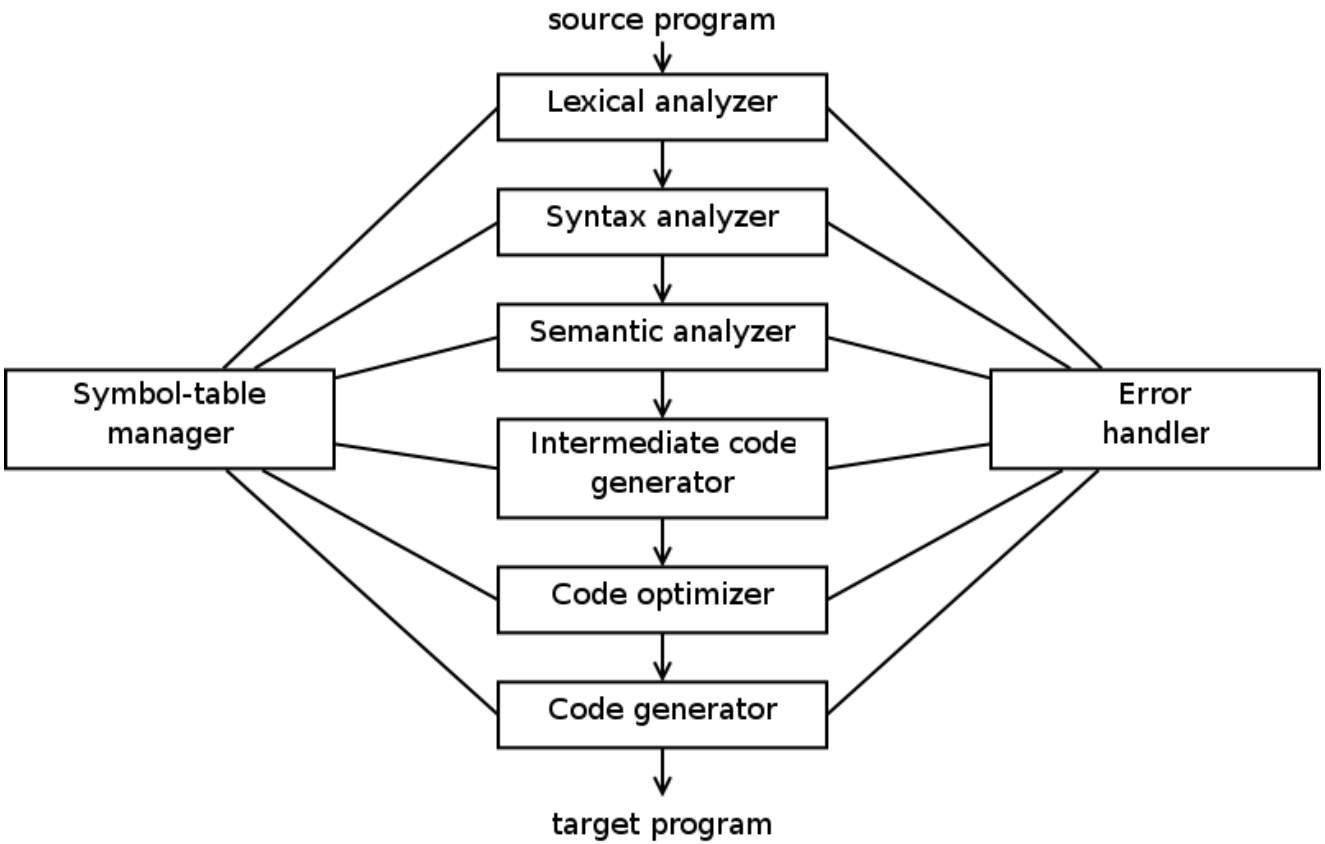
Java?

Birth

- 1954 – IBM develops the 704 (software cost $>$ hardware cost)
- 1954 - 1957 – FORTRAN I (FORmula TRANslating system) is developed (In 1958 50% of code is written in FORTRAN)
  - The definition of the first compiler led to an enourmous body of theoretical work

Compiler constuction is a complex engineering activity (practice) which need to be based on well defined theoretical background (theory)

Structure of a Compiler



Two main parts:

Analysis(front end) and Synthesis (back end)

Lexical analysis

After having defined the alphabet to be used, the first things to do is to recognize words

This is a sentence

The lexical analysis divides the program text into words and produce a sequence of tokens ($\langle token\text{-}name, attribute\text{-}value \rangle$)

```
position = initial + rate * 60
```

Lexical analysis

After having defined the alphabet to be used, the first things to do is to recognize words

This is a sentence

The lexical analysis divides the program text into words and produce a sequence of tokens ($\langle$*token-name, attribute-value*$\rangle$)

```
position = initial + rate * 60
```

Syntax Analysis

After having understood the words we need to understand the sentence structure. Not so much different from the syntax of what we do for understanding natural languages

This line includes a long sentence

Semantic Analysis

Once the structure of the sentence is clear we need to understand the meaning:

- Humans can manage quite well this activity, the same is not so true for machines

Examples:

- Jack said Jerry left his assignment at home
- Jack said Jack left his assignment at home?
- Jack left her assignment at home

- Compilers perform many semantic checks besides variable bindings.

Intemediate Code Generation

Easy to produce and easy to translate code format. Tipically based on a *three-address code* form:

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Code Optimization

Not so important for natural language! It is now the most complex and effort prone activity in the construction of modern compilers

- The compilers modify the program so that it
  - runs faster
  - uses less memory
  - uses less power
  - makes less database accesses
  - uses less bandwidth
  - ...

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Code generation

Permits to produce assembly code to be run on the target machine

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

Proportions of the various phases changed from the pioneering era

Code generation

Permits to produce assembly code to be run on the target machine

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

Proportions of the various phases changed from the pioneering era

General remarks

- New computer architectures need new compilers
  - parallelism
  - memory hierarchies
- New linguistic construction ask for the development of new algoritms and new data structure to translate the code
- Code optimization faces many undecidable problems, so theory alone is not enough and we need heuristics, good engineers, and good programmers
- Scope (static and dynamic), environment and states
- Parameter passing mechanisms

General remarks

- New computer architectures need new compilers
  - parallelism
  - memory hierarchies
- New linguistic construction ask for the development of new algoritms and new data structure to translate the code
- Code optimization faces many undecidable problems, so theory alone is not enough and we need heuristics, good engineers, and good programmers
- Scope (static and dynamic), environment and states
- Parameter passing mechanisms

General remarks

- New computer architectures need new compilers
  - parallelism
  - memory hierarchies
- New linguistic construction ask for the development of new algoritms and new data structure to translate the code
- Code optimization faces many undecidable problems, so theory alone is not enough and we need heuristics, good engineers, and good programmers
- Scope (static and dynamic), environment and states
- Parameter passing mechanisms

General remarks

- New computer architectures need new compilers
  - parallelism
  - memory hierarchies
- New linguistic construction ask for the development of new algoritms and new data structure to translate the code
- Code optimization faces many undecidable problems, so theory alone is not enough and we need heuristics, good engineers, and good programmers
- Scope (static and dynamic), environment and states
- Parameter passing mechanisms

General remarks

- New computer architectures need new compilers
  - parallelism
  - memory hierarchies
- New linguistic construction ask for the development of new algoritms and new data structure to translate the code
- Code optimization faces many undecidable problems, so theory alone is not enough and we need heuristics, good engineers, and good programmers
- Scope (static and dynamic), environment and states
- Parameter passing mechanisms

Interesting Questions?

- Why are there so many programming languages?
  - Application domains have distinctive needs (scientific computing, business applications, system programming, etc.)
- Why are there new programming languages?
  - Cost of training programmers is the dominant cost for a programming language
  - productivity $>$ training cost?
- What is a good programming language?
  - Is it the one programmers use?

Interesting Questions?

- Why are there so many programming languages?
  - Application domains have distinctive needs (scientific computing, business applications, system programming, etc.)
- Why are there new programming languages?
  - Cost of training programmers is the dominant cost for a programming language
  - productivity $>$ training cost?
- What is a good programming language?
  - Is it the one programmers use?

Interesting Questions?

- Why are there so many programming languages?
  - Application domains have distinctive needs (scientific computing, business applications, system programming, etc.)
- Why are there new programming languages?
  - Cost of training programmers is the dominant cost for a programming language
  - productivity $>$ training cost?
- What is a good programming language?
  - Is it the one programmers use?