# Compiler Construction

## Chapter 1 – Lexical Analysis

Prof. Dr. Jörg Kreiker
joerg.kreiker@informatik.hs-fulda.de

Fachbereich Angewandte Informatik
Hochschule Fulda – University of Applied Sciences

April 20, 2017

# Agenda



1. Organization
2. General overview
3. Lexical Analysis
   1. Regular expressions
   2. Finite state machines
   3. Regular expressions $\rightarrow$ finite state machines

# Organization



- Sign up for Moodle class at
  https://elearning.hs-fulda.de/ai/course/view.php?id=415
- Slides and code examples will be provided
- All lectures will be videotaped
- Tutorial: practical application
- No additional exercises/homework (but a compiler project)

# Rough outline of the class



1. Lexical Analysis
2. Parsing
3. Symbol tables
4. Type checking
5. C-machine CMA
6. Code generation
7. Instruction selection
8. Register allocation
9. Peephole optimizations
10. Garbage collection

# Compiler Project



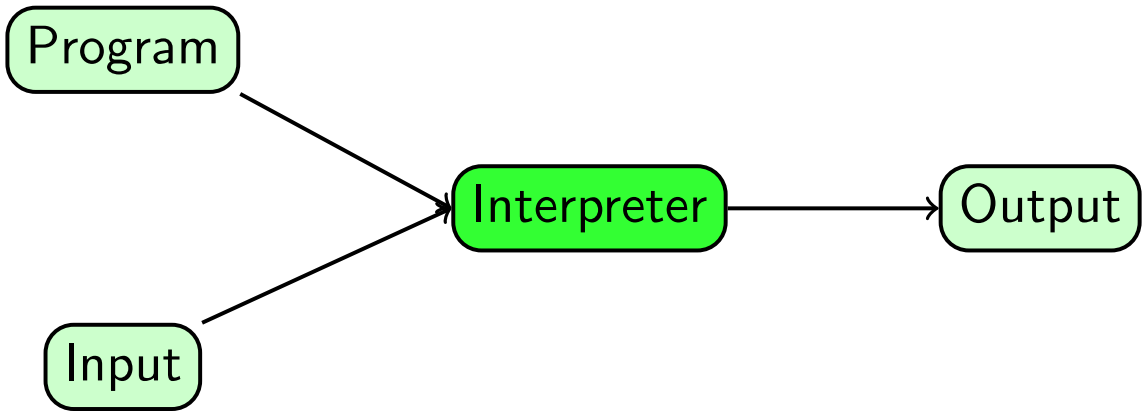
- You must implement a compiler
  - in groups of 3 people
  - compiling C0 (subset of C)
  - to virtual machine code (CMA)
  - written in Java
- Three milestones during the term (announced later)

# Grading



- On July 24 (tentatively) there will be individual oral exams
- An oral exam is 25 minutes
- 15 minutes discussion of your compiler (without a compiler, you will certainly fail this part, thus the exam)
- 10 minutes questions about one of the chapters (drawn randomly)

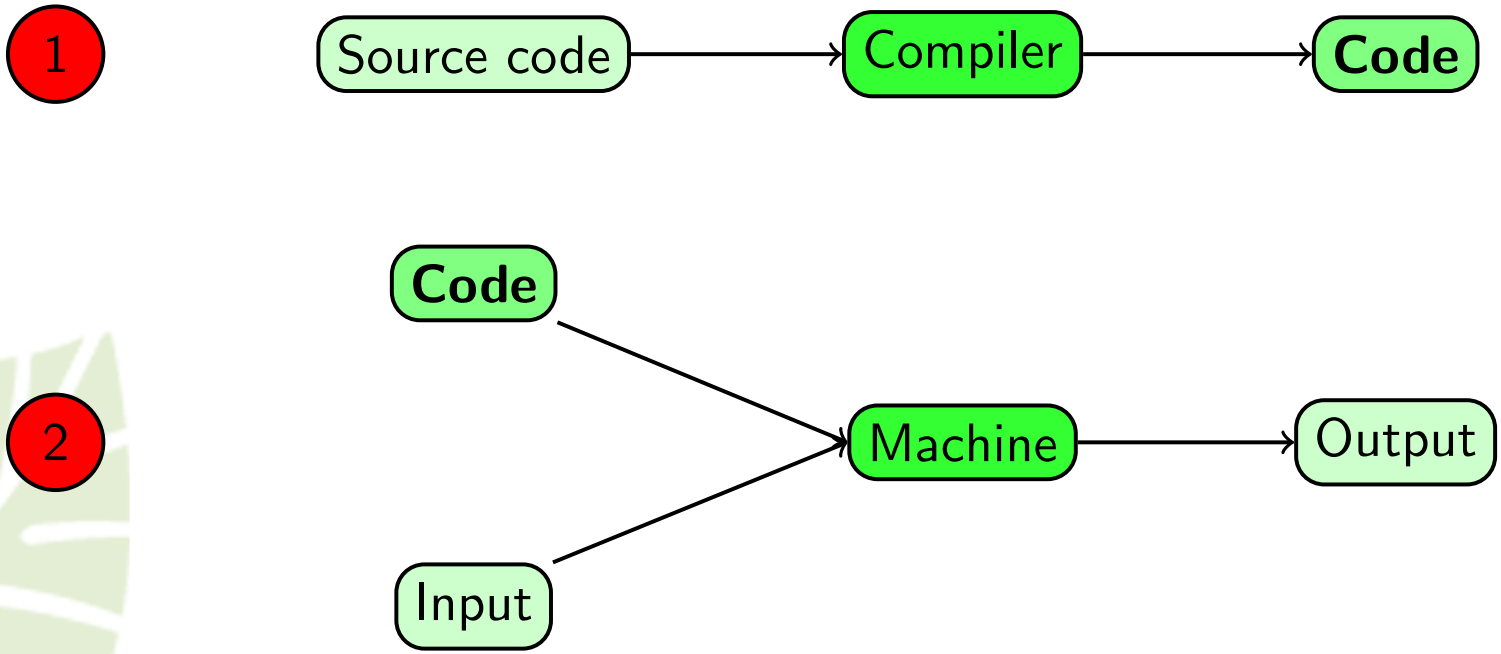
# Interpreter




- Benefits: No pre-computation on source code, no start-up time
- Disadvantage: Re-analysis of parts of the program, worse runtime

# Compiler






1. Compilation of source code to machine code
2. Execution of machine code on given input

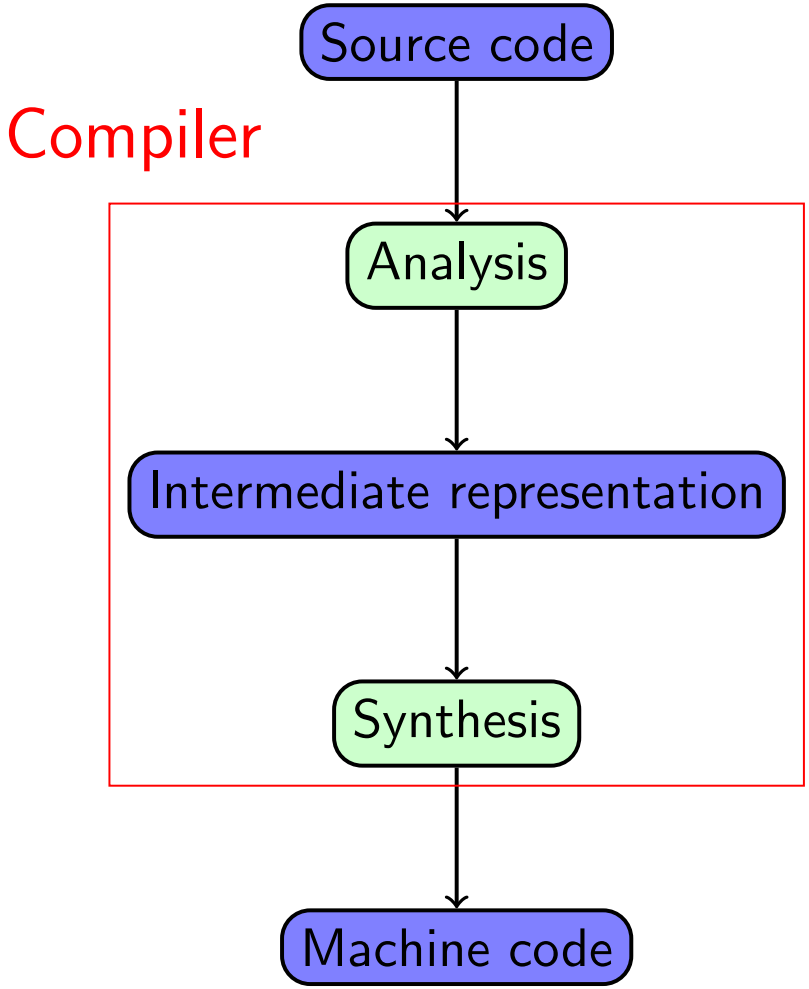
# Benefits of Compilation



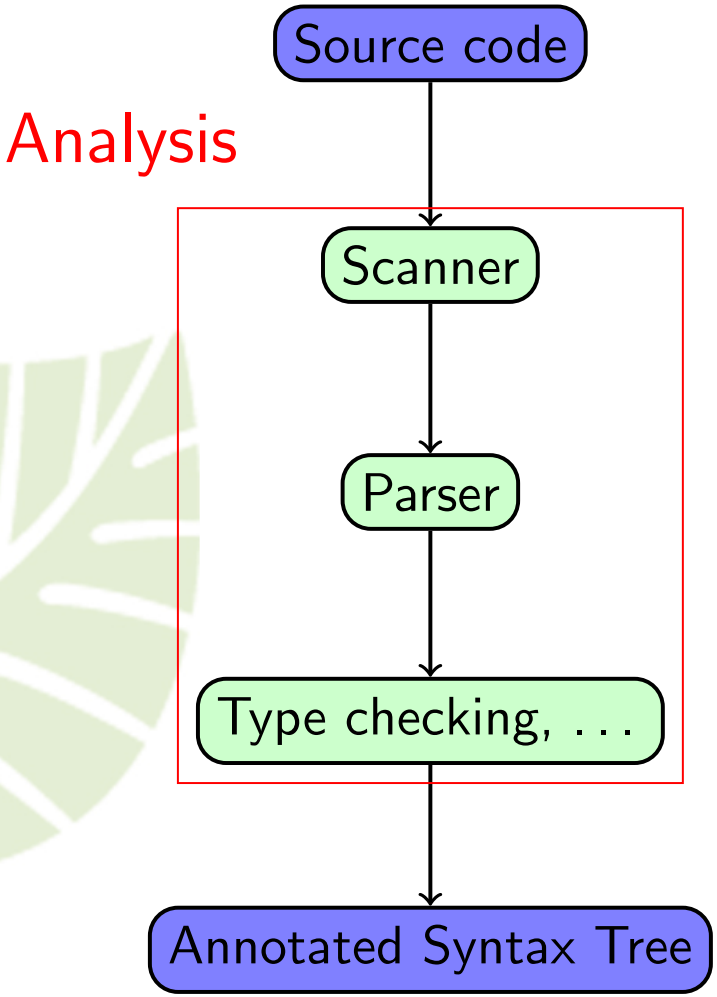
- Cleverer management of variables
- Potential for global optimizations
- More efficient execution: crucial for complex programs and programs executed often
- But: compilation takes its time

# Compiler Overview





# Analysis Phase



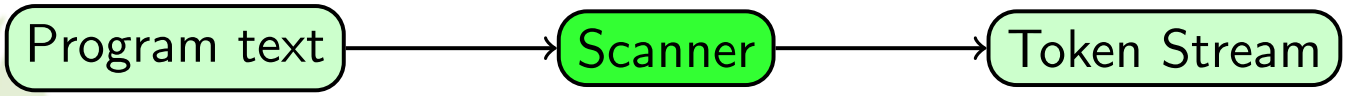

Lexical analysis → Token stream

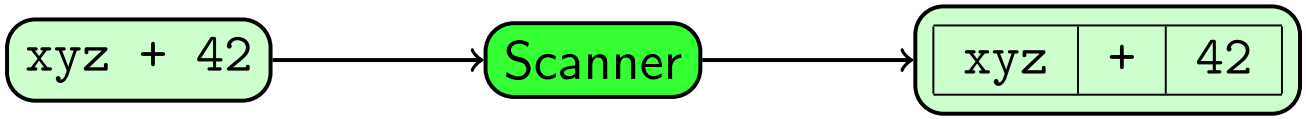
Syntactical analysis → Syntax tree

Semantical analysis

# Lexical Analysis








- Token: a sequence of characters that belong together
- Token classes
  - Names or identifiers like `xyz`, `catch22`
  - Constants like `42`, `3.14`
  - Operators like `+`
  - key words or reserved words like `while`, `else`

# Weeder



When tokens are classified, they can be weeded

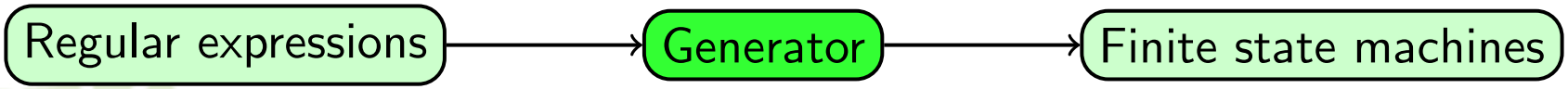
- Remove white spaces, comments, ...
- Replace certain tokens by their meaning, e.g. constants and names

→ weeding

- Scanner and weeder are typically summarized
- Both are generated (sometimes even together with the parser)

# Scanner Generation






- Specification of token classes: regular expressions
- Generated implementation: Finite state machines