Compiler Construction I

Dr. Michael Petter

SoSe 2020
Organizing

- Master or Bachelor in the 6th Semester with 5 ECTS
- Prerequisites
  - Basic Programming: **Java**
  - *Introduction to Theory of Computation*
  - Basic Principles: Operating Systems and System Software
  - Automata Theory
- Delve deeper with
  - Virtual Machines
  - Programm Optimization
  - Programming Languages
  - Labcourse Compiler Construction

**Materials:**
- TTT-based lecture recordings
- The slides
- Related literature list online (⇒ Wilhelm/Seidl/Hack Compiler Design)
- Tools for visualization of abstract machines (VAM)
- Tools for generating components of Compilers (JFlex/CUP)
Flipped Classroom

... is a concept to focus more on students learning process – and fits quite well into plague time.

Content delivery:

- Mandatory recordings:
  http://ttt.in.tum.de/lectures/index_ws.php?year=20&s=true#COMP
- Presented as lessons
- To be prepared single-handedly within a week
- Starting Apr 23rd

Virtual Classroom:

Thursdays 14:00-16:00 via bbb.in.tum.de, starting Thu, Apr 23rd

- Discussion
- AMA (Ask me [almost] Anything)
- Content Practice
- Further Insights
Flipped Classroom

Tutorial:

Monday 14:15-15:45 via either bbb.in.tum.de or tum-conf.zoom.us (will be announced on Moodle)

- Exercise sheet released each week to be solved at home
- In the tutorial: Discussion of the solution and your questions
- Recording of tutorial will also be published
- First session: May 4th
- For questions about the tutorial, email Michael Schwarz at m.schwarz@tum.de
- All information about the tutorial and exercise sheets:
  https://www.moodle.tum.de/course/view.php?id=53342

Exam:

- **One Exam** in the summer, *none* in the winter
- The date will be announced by the central examination committee
Topic:

Overview
Extremes of Program Execution

Interpretation:
- Program
- Input
- Interpreter
- Output

Compilation:
- Program
- Code
- Compiler
- Code
- Code
- Machine
- Output
- Output
Interpretation vs. Compilation

**Interpretation**
- No precomputation on program text necessary
  - no/small startup-overhead
- More context information allows for specific aggressive optimization

**Compilation**
- Program components are analyzed once, during preprocessing, instead of multiple times during execution
  - smaller runtime-overhead
- Runtime complexity of optimizations less important than in interpreter
General Compiler setup:

Program code

Compiler

Int. Representation

Analysis

Synthesis

Compiler

Code
Compiler

General Compiler setup:

Program code

Analysis

Int. Representation

Synthesis

Code
The *Analysis*-Phase consists of several subcomponents:

- Program code
The Analysis-Phase consists of several subcomponents:

- **Program code**
- **Scanner**
  - Token-Stream
  - lexicographic Analysis:
    - Partitioning in tokens
The Analysis-Phase consists of several subcomponents:

- **Program code**

- **Scanner**
  - lexicographic Analysis:
  - Partitioning in tokens

- **Parser**
  - syntactic Analysis:
  - Detecting hierarchical structure
The Analysis-Phase consists of several subcomponents:

- **Scanner**
  - lexicographic Analysis: Partitioning in tokens

- **Parser**
  - syntactic Analysis: Detecting hierarchical structure

- **Type Checker...**
  - semantic Analysis: Inferring semantic properties

Program code

(annotated) Syntax tree
Content on the Way

- Regular expressions and finite automata
- Specification and implementation of scanners
- Context free grammars and pushdown automata
- Top-Down/Bottom-Up syntax analysis
- Attribute systems
- Typechecking
- Codegeneration for register machines