Lecture 2

COMPILER DESIGN
Administrivia

• **Web site:** [https://people.inf.ethz.ch/suz/teaching/252-0210.html](https://people.inf.ethz.ch/suz/teaching/252-0210.html)
• **Moodle:** [https://moodle-app2.let.ethz.ch/course/view.php?id=11669](https://moodle-app2.let.ethz.ch/course/view.php?id=11669)
• **E-mail for teaching staff:** TBA (on Moodle and course websites)

• Please check course Moodle and Web sites regularly
  – Announcements
  – Homework assignments
  – Exercise sessions
  – Q/A and discussions
The Compiler Project

• Course projects (50% of grades)
  – (5%) HW1: Ocaml programming (due 01.10)
  – (9%) HW2: X86lite interpreter (due 15.10)
  – (9%) HW3: LLVMlite compiler (due 29.10)
  – (9%) HW4: Lexing, parsing, simple compilation (due 12.11)
  – (9%) HW5: Higher-level features (due 26.11)
  – (9%) HW6: Analysis and optimizations (due 10.12)

• Goal: Build a complete compiler from a high-level, type-safe language to x86 assembly
HW1: Hellocaml

- Homework 1 is now available on the course Moodle site
  - Individual project – no groups (5% of overall grade)
  - Due: Tuesday, 1 Oct. at 23:59
  - Topic: OCaml programming, an introduction to interpreters
  - If you haven’t yet, please start learning OCaml now!

- OCaml head start
  - Run “ocaml” from the command line to invoke the top-level loop
  - Run “ocamlbuild main.native” to run the compiler

- We recommend using
  - Emacs/Vim + merlin
  - (less recommended: Eclipse with the OcaIDE plugin)

  - More information on the tool chain will be on course Moodle
How to represent programs as data structures.
How to write programs that process programs.
Consider this implementation of factorial in a hypothetical programming language:

\[
\begin{align*}
x &= 6; \\
\text{ANS} &= 1; \\
\text{whileNZ } & (x) \\ &\{ \\
&\quad \text{ANS} = \text{ANS} \ast x; \\
&\quad x = x + -1; \\
&\}
\end{align*}
\]

We need to describe the constructs of this hypothetical language

- **Syntax**: which sequences of characters count as a legal “program”?
- **Semantics**: what is the meaning (behavior) of a legal “program”?
Grammar for a Simple Language

\[
\begin{align*}
\text{<exp> } &::= \\
&| \quad \text{<X>} \\
&| \quad \text{<exp>} + \text{<exp>} \\
&| \quad \text{<exp>} * \text{<exp>} \\
&| \quad \text{<exp>} < \text{<exp>} \\
&| \quad \text{<integer constant>} \\
&| \quad (\text{<exp>})
\end{align*}
\]

\[
\begin{align*}
\text{<cmd> } &::= \\
&| \quad \text{skip} \\
&| \quad \text{<X>} = \text{<exp>} \\
&| \quad \text{ifNZ} \text{<exp>} \{ \text{<cmd>} \} \text{ else } \{ \text{<cmd>} \} \\
&| \quad \text{whileNZ} \text{<exp>} \{ \text{<cmd>} \} \\
&| \quad \text{<cmd>}; \text{<cmd>}
\end{align*}
\]

- Concrete syntax (grammar) for a simple imperative language
  - Written in “Backus-Naur form”
  - `<exp>` and `<cmd>` are nonterminals
  - ‘::=’, ‘|’, and `<...>` symbols are part of the meta language
  - keywords, like `skip` and `ifNZ` and symbols, like ‘{’ and ‘+’ are part of the object language

- Need to represent the abstract syntax (i.e. hide the irrelevant of the concrete syntax)
- Implement the operational semantics (i.e. define the behavior, or meaning, of the program)
OCaml Demo

simple.ml
translate.ml