Lecture 1

CIS 341: COMPILERS

Administrivia

- **Instructor:** Steve Zdancewic
  Office hours: Tuesdays 3:30-5:00 & by appointment
  Levine 511

- **TAs:**
  - Dmitri Garbuzov
    Office Hours: Weds. 3:30-
  - Rohan Shah
    Office hours: Monday 5:00-7:00pm Levine 5th floor bump space

- **E-mail:** cis341@seas.upenn.edu
- **Web site:** http://www.seas.upenn.edu/~cis341
- **Piazza:** http://piazza.com/upenn/spring2015/cis341

HW1: Hellocaml

- Homework 1 is available on the course web site.
  - Individual project – no groups
  - Due: Thursday, 22 Jan. 2013 at 11:59pm
  - Topic: OCaml programming, an introduction

- OCaml head start on eniac:
  - Run "ocaml" from the command line to invoke the top-level loop
  - Run "ocamlbuild main.native" to run the compiler

- We recommend using either:
  - Eclipse with the OcaIDE plugin
  - Emacs/Vim + merlin
  - See the course web pages about the CIS341 tool chain to get started

How to represent programs as data structures.
How to write programs that process programs.

INTERPRETERS

Factorial: Everyone's Favorite Function

- Consider this implementation of factorial in a hypothetical programming language:

```
X = 6;
ANS = 1;
whileNZ (x) {
    ANS = ANS * X;
    X = X + -1;
}
```

- 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

```
<exp>  ::=
       |     <X>
       |     <exp> + <exp>
       |     <exp> * <exp>
       |     <exp> < <exp>
       |     <integer constant>
       |     (<exp>)

<cmd>  ::=
       |     skip
       |     <X> = <exp>
       |     ifNZ <exp> { <cmd> } else { <cmd> }
       |     whileNZ <exp> { <cmd> }
       |     <cmd>; <cmd>
```

- 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