CSE 341: Programming Languages

Dan Grossman

Spring 2004

Lecture 1— Course Introduction

Welcome!

We have 10 weeks to learn *different paradigms* and *fundamental concepts* of programming languages.

With diligence, patience, and an open mind, this course makes you a much better programmer (in languages we won't use).

Today in class:

- Course mechanics
- Course overview and a rain-check on motivation
- Dive into ML (homework 1 due Wednesday April 7)

In the next 24 hours:

- Join the class mailing list
- Email homework 0 (worth 0 points) to me

`http://www.cs.washington.edu/education/courses/cse341/04sp/`

Who and What

- 3 class meetings (slides, code, and questions)
  - Material on-line after class, but take notes.
- 1 section (David Richardson)
  - Essential material on tools, style, examples, language-features, ...
- Office hours (Evan Martin, David, me)
  - Use them!!!

Homeworks

- Approximately weekly (1 or 2 "double homeworks" later)
- Doing the homework involves:
  1. Understanding the concepts being addressed
  2. Writing code demonstrating understanding of the concepts
  3. Testing your code to ensure you understand
  4. "Playing around" with variations, incorrect answers, etc.

  You turn in only (2), but focusing on (2) makes the homework *harder*

Collaboration: The Gilligan's Island Rule

Extra Credit: Terrible use of your time grade-wise, but great otherwise

Academic Integrity

Read every word of the course policy very carefully.

Always explain any unconventional action on your part.

Promoting and enforcing academic integrity has been a personal focus of mine for 11 years now:

- I trust you completely
- I have no sympathy for trust violations, nor should you

Honest work is the most important feature of a university.

Exams

- Midterm: April 28 in class
- Final: June 10, 8:30–10:20
- Do not miss them

Same concepts, but very different format from homework.

Now where were we?

Meetings, homeworks, and exams. . . about what?

Programming languages:

- Essential concepts relevant in any language
- Specific examples "in natural setting" using ML, Scheme, and Smalltalk
- Focus on "functional languages" because they are simpler and teach good practices

First half of course uses ML:

- Gives us time to build knowledge before "starting over"
- But we need to get comfortable with the basics and environment *as soon as possible*.
- "Let go of Java" for now (we will return to it)

A strange environment

The ML part of the course uses:

- The emacs editor
- A read-eval-print loop for evaluating programs
- Available on Windows and UNIX in the lab, but remotely via UNIX

We have prepared "getting started" materials, but leave plenty of time for the *content* of homework 1.

- Read the materials
- Attend section
- Then ask questions fast (wasted hours are wasted hours)

Adjusting to new environments is a "CS life skill"

Before we dive in

We'll return to the course goals and "why learn something other than C/C++/Java/Perl" next week or so.

But one more thing about the course:

- I know this has not always been the most popular course
- Let's change that

ML, from the beginning

- A program is a sequence of *bindings*
- One kind of binding is a *variable binding*

  `val x = e ;` (semicolon optional in a file)

- A program is evaluated by evaluating the bindings in order
- A *variable binding* is evaluated by:
  - Evaluating the expression in the environment created by the previous bindings. This produces a *value*.
  - Extending the (top-level) environment to bind the variable to the value.

Much easier to understand with an example...

That was a lot at once

- "Atomic" expressions so far: variables and constant integers
- "Compound" expressions so far: addition, subtraction, less than, conditionals
- Types: every expression has a type. So far, `int`, `bool`, `unit`
- The read-eval-print loop:
  - Enter a sequence of bindings. For each, it tells you the value and type of the new binding
  - If you just enter `e;`, then that is the same as `val it = e;`
  - `use "foo.sml"` enters the bindings in a file, and then binds `it` to `()`, which has type `unit`
  - Messages like "`GC #0.0.0.0.1.18:    (0 ms)`" are stupid
  - *Illegal* expressions lead to (bad) error messages and no change to the environment

Parts worth repeating

Our very simple program demonstrates many critical language concepts:

- Expressions have a *syntax* (written form)
  - E.g.: A constant integer is written as a digit-sequence
  - E.g.: Addition expression is written `e1 + e2`
- Expressions have *types* given their environment
  - E.g.: In any environment, a constant integer has type `int`
  - E.g.: If `e1` and `e2` have type `int` in the current environment, then `e1+e2` has type `int`
- Expressions *evaluate to values* given their environment
  - E.g.: In any environment, a constant integer evaluates to itself
  - E.g.: If `e1` and `e2` evaluate to `c1` and `c2` in the current environment, then `e1+e2` evaluates to the sum of `c1` and `c2`

More expression forms

What are the syntax-rules, typing-rules, and evaluation-rules for:

- variables
- less-than comparisons
- conditional expressions

Lots more to do

We have many more types, expression forms, and binding forms to learn before we can write "anything interesting".

Must develop resilience to mistakes and bad messages. Example gotcha: `x = 7` instead of `val x = 7`.

Rest of the week: functions, pairs, lists, and local bindings

But there are some things we will *not* add:

- assignment: changing the value of an environment binding
  - make a new binding instead
- statements: expressions will do just fine, thank you
- loop-constructs: recursive functions are more powerful

What is a programming language?

Here are separable concepts for defining and evaluating a language:

- syntax: how do you write the various parts of the language?
- semantics: what do programs mean? (One way to answer: what are the evaluation rules?)
- idioms: how do you typically use the language to express computations?
- libraries: does the language provide "standard" facilities such as file-access, hashtables, etc.? How?
- tools: what is available for manipulating programs in the language?