Welcome!

• We have 15 weeks to learn the fundamental concepts of programming languages
• With hard work, patience, and an open mind, this course makes you a much better programmer
  – Even in languages we won’t use
  – Learn the core ideas around which every language is built, despite countless surface-level differences and variations
  – Learn to program in SML and Ruby (and maybe Racket)
• Today’s class:
  – Dive into ML
  – Course mechanics
  – [A rain-check on motivation]
  – Homework 1 assigned on Wednesday, due on Jan. 30

Introductions

Course staff:
• Prof. Michael Clarkson
• Andrew Hirsch

Get to know us!

What this course is about

• Many essential concepts relevant in any programming language
  – And how these pieces fit together
  – Break complicated things down into brutally simple pieces
  – See how those pieces show up in many languages
• Use the languages SML and Ruby because:
  – They let many of the concepts “shine”
  – Using multiple languages shows how the same concept can “look different” or actually be slightly different
  – In many ways simpler than Java

Why?

• Learning to think about software in the “PL” way will make you a better programmer even if/when you go back to old ways
• It will also give you the mental tools and experience you need for a lifetime of confidently picking up new languages and ideas

Why?

• http://www.youtube.com/watch?v=gPK-CGG8INJ

4

5

6
Tabula Rasa

• For at least the next two weeks, please “let go of Java”
  – Learn ML as a “totally new way of programming”
  – Later we’ll contrast with what you know
  – Saying “oh that is kind of like that thing in Java” will confuse you, slow you down, and make you learn less
• In a few weeks, we’ll have the background to
  – Intelligently motivate the course
  – Understand how functional programming is often simple, powerful, and good style – even when “forced” to use Java or C
  – Understand why functional programming is increasingly important, if not wildly popular (yet!) in the ‘real world’

A strange environment

• The next 4-5 weeks will use
  – The SML language
  – The Emacs editor
  – A read-eval-print loop (REPL) for evaluating programs
• You need to get things installed, configured, and usable
• Only then can you focus on the content of Homework 1
• Working in new environments is a CS life skill

ML from the beginning

• A program is a sequence of bindings
• Type-check each binding in order using the static environment produced by the previous bindings
• Evaluate each binding in order using the dynamic environment produced by the previous bindings
  – Dynamic environment holds values, the results of evaluating expressions
• Today the only kind of binding is a variable binding

A very simple ML program

(* My first ML program *)
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 – z else z;
val abs_of_z_simpler = abs z

A variable binding

val x = (x + y) + (y + 2); (* comment *)
More generally:
val x = e;

• Syntax:
  – Keyword val and punctuation = and ;
  – Variable x
  – Expression e
  – many forms of these, most containing subexpressions

Expressions

• We’ve already seen many kinds of expressions:
  – 34 true false x e1+e2 e1<e2
  – if e1 then e2 else e3
  – Can get arbitrarily large since any subexpression can contain subsubexpressions, etc.
• Every kind of expression has
  – Syntax
  – Type-checking rules
    – Produces a type or fails (with a bad error message)
  – Type so far int -> bool -> unit
  – Evaluation rules (used only on things that type-check)
    – Produces a value (or exception or infinite-loop)
Variables

- Syntax:
  - Sequence of letters, digits, `_`
  - Can’t start with digit

- Type-checking:
  - Look up type in current static environment
  - If not there fail

- Evaluation:
  - Look up value in current dynamic environment

Addition

- Syntax:
  - $e_1 + e_2$ where $e_1$ and $e_2$ are expressions

- Type-checking:
  - If $e_1$ and $e_2$ have type int,
    then $e_1 + e_2$ has type int

- Evaluation:
  - If $e_1$ evaluates to $v_1$ and $e_2$ evaluates to $v_2$,
    then $e_1 + e_2$ evaluates to sum of $v_1$ and $v_2$

Values

- All values are expressions
- Not all expressions are values
- A value “evaluates to itself” in “zero steps”
- Examples:
  - 34, 17, 42 have type int
  - true, false have type bool
  - () has type unit

A slightly tougher one

What are the syntax, typing rules, and evaluation rules for conditional expressions?

Pragmatics

- Lecture has emphasized building up from simple pieces
- But in practice you make mistakes and get inscrutable messages
- Example gotcha: $x = 7$ instead of $\text{val } x = 7$
- Work on developing resilience to mistakes
  Don’t panic.
  Stop. Read. Think.

The foundation we need

- We have many more types, expression forms, and binding forms to learn before we can write “anything interesting”
- Syntax, typing rules, evaluation rules will guide us the whole way!
- For homework 1: functions, pairs, conditionals, lists, options, and local bindings
  - Earlier problems require less
- Will not add (or need):
  - Mutation (a.k.a. assignment): use new bindings instead
  - Statements: everything is an expression
  - Loops: use recursion instead
Note my writing style

- In writing, I use words very precisely and concisely
  - I’m a computer scientist and I write like one (a good thing!)
  - Technical issues deserve precise technical writing
  - Conciseness values your time as a reader
  - You should try to be precise, too
  - I deliberately use quotation marks when I’m being vague

- Skimming or not understanding why a word or phrase was chosen can make the homework harder

- By all means ask if a problem is confusing
  - Being confused is normal and understandable
  - Once you’re unconfused, you might agree the problem wording didn’t cause the confusion
  - But I do make mistakes, too

Concise to-do list

In the next 24 to 48 hours:

- Read today’s handouts. Also on course web page: http://faculty.cs.gwu.edu/~clarkson/courses/csci4223/2013sp/
- Register on Piazza
- Get set up using Emacs and SML
  - Installation/configuration/use instructions posted on web page tonight
  - Essential. No reason to delay!