



CSE341: Programming Languages Section 1

Spring 2020

Adapted from slides by Dan Grossman, Eric Mullen and Ryan Doenges

Agenda

- Introduction
- Course Resources
- Set up
- REPL
- Emacs Basics
- Shadowing
- Debugging
- Bonus: "Generics" and Equality Types

CSE 341: Programming Languages

2

Remote Quarter

- Feel free to share your video and ask questions!
 - Especially in section!
- Breakout rooms will be used to have some class discussion
- No midterm, no final!
 - 4 quizzes, 8 HWs
- Two late days for each HW
 - Work submitted after the due date may not be graded and returned before the next assignment is due and/or may be returned with less feedback.

CSE 341: Programming Languages

3

Course Resources

- We have a ton of course resources. Please use them!
- If you get stuck or need help:
 - Ask questions in [Ed](#)
 - Come to Office Hours via Zoom
- We're here for you

CSE 341: Programming Languages

4

Setup

- Excellent guide located on the course website under Resources
- We're going to spend about 5 minutes setting up now (so you can follow along for the rest of section)
- You need 3 things installed:
 - Emacs
 - SML
 - SML mode for Emacs

CSE 341: Programming Languages

5

Editor vs. IDE

- You may be familiar with IDEs (JGrasp, Eclipse, IntelliJ, etc.)
 - Handles compilation, error reporting, running, ...
- Emacs is an *editor*
 - Many similar features! e.g., Syntax highlighting, ...
 - Not tied to a specific language
 - (Vim is another alternative editor you can use)
- There is no clear distinction between these two concepts
- Running and compilation is done outside the editor
- You can code in all programming languages we cover in 341 with Emacs - so please get comfortable with it :)

CSE 341: Programming Languages

6

ML Development Workflow

- REPL is the general term for tools like “Run I/O” you have been using in JGRASP for CSE 142/3
- REPL means **Read Eval Print Loop**
- Read: ask the user for semicolon terminated input
- Evaluate: try to run the input as ML code
- Print: show the user the result or any error messages produced by evaluation
- Loop: give another prompt back to continue

CSE 341: Programming Languages

7

ML Development Workflow

- Simple Demo of REPL
 - You can type in any ML code you want, it will evaluate it
 - Useful to put code in .sml file for reuse
 - Every command must end in a semicolon (;)
 - Load .sml files into REPL with **use** command

CSE 341: Programming Languages

8

Emacs Basics

- Don't be scared!
- Commands have particular notation: C-x means hold Ctrl while pressing x
- Meta key is Alt (thus M-z means hold Alt, press z)
 - C-x C-s is Save File
 - C-x C-f is Open File
 - C-x C-c is Exit Emacs
- C-g is Escape (Abort any partial command you may have entered. If you get confused while typing use this)
- M-x is “Do a thing”

CSE 341: Programming Languages

9

Shadowing

```
val a = 1;
val b = 2 + a;
val a = 3;
```

- Does the above code compile? If so, what do you think it does and what is the value of b?
- Remember, SML doesn't have mutation.

CSE 341: Programming Languages

10

Shadowing

```
val a = 1;    a -> int
val b = 2 + a; a -> int, b -> int
val a = 3;    a -> int, b -> int, a -> int
```

- You can't change a variable, but you can add another with the same name
- When looking for a variable definition, most recent is always used
- Shadowing is usually considered bad style

CSE 341: Programming Languages

11

Shadowing

- This behavior, along with `use` in the REPL can lead to confusing effects
- Suppose I have the following program:


```
val x = 8;
val y = 2;
```
- I load that into the REPL with `use`. Now, I decide to change my program, and I delete a line, giving this:


```
val x = 8;
```
- I load that into the REPL without restarting the REPL. What goes wrong?
 - Hint: what is the value of y?

CSE 341: Programming Languages

12

Comparison Operators

- You can compare numbers in SML!
- Each of these operators has 2 subexpressions of type `int`, and produces a `bool`

<code>=</code> (<i>Equality</i>)	<code><</code> (<i>Less than</i>)	<code><=</code> (<i>Less than or equal</i>)
<code><></code> (<i>Inequality</i>)	<code>></code> (<i>Greater than</i>)	<code>>=</code> (<i>Greater than or equal</i>)

Boolean Operators

- You can also perform logical operations over `bool`s!

Operation	Syntax	Type-Checking	Evaluation
<code>andalso</code>	<code>e1 andalso e2</code>	<code>e1</code> and <code>e2</code> have type <code>bool</code>	Same as Java's <code>e1 && e2</code>
<code>orelse</code>	<code>e1 orelse e2</code>	<code>e1</code> and <code>e2</code> have type <code>bool</code>	Same as Java's <code>e1 e2</code>
<code>not</code>	<code>not e1</code>	<code>e1</code> has type <code>bool</code>	Same as Java's <code>!e1</code>

- `and` is completely different, we may talk about it later
- `andalso`/`orelse` are SML built-ins as they use short-circuit evaluation
 - We'll talk about why they have to be built-ins later

And... Those Bad Styles

- Language does not need `andalso`, `orelse`, or `not`

```
(* e1 andalso e2 *)
if e1
then e2
else false

(* e1 orelse e2 *)
if e1
then true
else e2

(* not e1 *)
if e1
then false
else true
```

- Using more concise forms generally much better style
- And definitely please do not do this:

```
(* just say e (!!!) *)
if e
then true
else false
```

Debugging

DEMO

- Errors can occur at 3 stages:
 - Syntax:** Your program is not "valid SML" in some (usually small and annoyingly nitpicky) way
 - Type Check:** One of the type checking rules didn't work out
 - Runtime:** Your program did something while running that it shouldn't
- The best way to debug is to read what you wrote carefully, and think about it.

Testing

- We don't have a unit testing framework
- You should still test your code!
- Just do something like this:

```
val test1 = ((4 div 4) = 1);
```

Parametric Polymorphism ("Generics")

- What's wrong with this code?

```
fun swap(pair : int * string) =
  (#2 pair, #1 pair)

val x = swap ("hello", 123)
```

- Technically correct answer: there's a type error
- Better answer: `swap` should have a more general type

CSE 14X Time: How do Java?

```

class Pair<A, B> {
    final A fst; final B snd;
    Pair (A fst, B snd) {
        this.fst = fst;
        this.snd = snd;
    }
}
class Main {
    static <A, B> Pair<B, A> swap(Pair<A, B> p) {
        return new Pair(p.snd, p.fst);
    }
    public static void main(String[] args) {
        Pair<Integer, String> x =
            Main.swap(new Pair("hello", 123));
    }
}

```

19

Anything you can do, I can do better.

- We can make our `swap` function generic!

```

fun swap(pair : 'a * 'b) =
    (#2 pair, #1 pair)
val c = swap ("hello", 123)

```

- What do you think the type of `swap` is?

CSE 341: Programming Languages

20

Equality

- “=” is the hardest concept in Programming Language Theory
- Unlike Java, SML doesn't have equality for every type
- This is good! Equality doesn't always make sense
- One reason: Floating Point is weird

```

val x = 0.1 + 0.2;
val y = 0.3;
val z = x - y;
(* z is not zero!!! *)

```

CSE 341: Programming Languages

21

Equality (cont.)

- “=” is the hardest concept in Programming Language Theory
- Unlike Java, SML doesn't have equality for every type
- This good! Equality doesn't always make sense
- One reason: Floating Point is weird
- Other reason: It doesn't make sense for functions

```

fun f(n : int) =
    if n > 100 then n-1 else n+1
fun g(n : int) = n - 1
(* How could we check f = g? *)

```

- Bonus for those who've taken CSE 311: “Do these two programs do the same thing” is reducible to the halting problem

CSE 341: Programming Languages

22

Parametric Polymorphism & Equality

- What happens if I write the following program?

```

fun f(n, a, b) =
    if a = b then n - 1 else n + 1
val x = f(1, 2, 3)
val y = f(1, 2.0, 3.0)

```

CSE 341: Programming Languages

23