CSE341: Programming Languages
Section 1

Brendan Murphy
Spring 2019

Adapted from slides by Dan Grossman, Eric Mullen and Ryan Doenges
Some Facts About Me

• Math/CS Double Major
• Scared of the word “pragmatic”
• TAed 341 once before (Winter ‘19)
• Loves Types
• Favorite Programming Language is Haskell
Course Resources

• We have a ton of course resources. Please use them!
• If you get stuck or need help:
  – Ask questions in Piazza
  – Come to Office Hours (on website, you don’t need a list of topics before you decide to stop by)
• We’re here for you
Agenda

• Emacs Basics
• Shadowing
• Debugging
• “Generics” and Equality Types
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 :)


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”
Shadowing

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

• Does the above code compile? If so, what do you think it does? Talk to a neighbor!
• Remember, SML doesn’t have mutation.
Shadowing

- 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
Shadowing

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.

```scala
val a = 1;  a -> 1
val b = 2;  a -> 1, b -> 2
val a = 3;  a -> 1, b -> 2, a -> 3
```
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?)
Comparison Operators

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

<table>
<thead>
<tr>
<th>=  (Equality)</th>
<th>&lt;  (Less than)</th>
<th>&lt;= (Less than or equal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&gt; (Inequality)</td>
<td>&gt;  (Greater than)</td>
<td>&gt;= (Greater than or equal)</td>
</tr>
</tbody>
</table>
Boolean Operators

- You can also perform logical operations over booleans!

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Type-Checking</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>andalso</td>
<td>e1 andalso e2</td>
<td>e1 and e2 have type bool</td>
<td>Same as Java’s e1 &amp;&amp; e2</td>
</tr>
<tr>
<td>orelse</td>
<td>e1 orelse e2</td>
<td>e1 and e2 have type bool</td>
<td>Same as Java’s e1</td>
</tr>
<tr>
<td>not</td>
<td>not e1</td>
<td>e1 has type bool</td>
<td>Same as Java’s !e1</td>
</tr>
</tbody>
</table>

- and is completely different, we may talk about it later
- `andalso/orelse` are SML built-ins as they use short-circuit evaluation, we will 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 *)
c
(* 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.
Parametric Polymorphism (“Generics”)

- What’s wrong with this code?

```haskell
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
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));
    }
}
Anything you can do, I can do better.

- We can make our `swap` function generic!

```haskell
fun swap(pair : 'a * 'b) =
  (#2 pair, #1 pair)

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

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

- “=” 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

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

```haskell
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
What happens if I write the following program?

```ml
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)
```