CSE341 – Section 1

Emacs, SML Mode, Shadowing, Error Messages

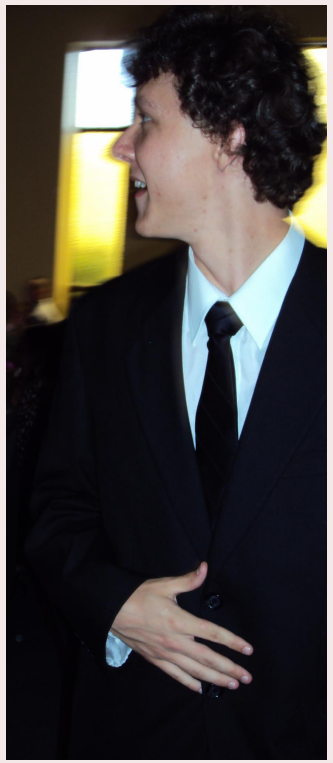
Cody A. Schroeder

January $10^{th}, 2013$

Intro

Hello! I'm Cody!!!

- One of the two main section leaders (with Eric).
  - We'll probably alternate weeks.
  - Remember: Cody is the fancy one. ⟶
- I'm one of the 5^{th} year masters students.
- Previously TA'd many times including 341 twice.
- This is one of my favorite classes!



Emacs

- Our EDITOR of choice this quarter!
- Not necessarily required but recommended.
- Plenty of cheat sheets and tutorials:
  - Stanford Cheat Sheet, Ref Card, OLD UW Tutorial, etc.
- Also the staff is *almost* always available for help!

Demo Time

- The basics!
- Any questions?

The REPL

- Read-Eval-Print-Loop
- Meant for iterative development and real-time testing.
- Usually load a file using **use**, test functions, edit source, restart REPL, and repeat.
- Very powerful tool for convenience.

Note: It's dangerous to call **use** more than once in the same REPL.

Shadowing of Variable Bindings

SML Example

```
1 val x = "Hello World!";
2 val x = 2;      (* Is this allowed? *)
3 val res = x*2; (* Is this 4 or a type error? *)
```

- There is no assignment in SML.
- However, a variable name can be bound multiple times.
- When looking up a variable, the latest binding in the current scope is used.
- Any previous bindings are said to be shadowed.
  - A similar shadowing effect occurs in other languages like Java.
- This is the reason calling **use** more than once on the same file can cause problems.

Shadowing of Variable Bindings (cont.)

Another shadowing example

```
1 fun absOriginal x = abs x; (* Save abs function. *)
2 fun abs (x,y) = (absOriginal x, absOriginal y);
```

FAQ

- Is it ever possible to use a shadowed variable? Yes! And no...
- It can be possible to uncover a shadowed variable when the latest binding goes out of scope.

Using a Shadowed Variable

```
1 val x = "Hello World!";
2 fun add1 (x : int) = x+1; (* Shadow x in function body *)
3 val y = add1 2;
4 val z = x^"!!!"; (* "Hello World!!!" *)
```

Dealing with error messages from SML

We've Got Errors...

```
1  val x = 34;
2  y = x + 1;
3  val z = if y then 34 else x < 4;
4  val q = if y > 0 then 0;
5  val a = -5;
6  val w = 0;
7  val fun = 34;
8  val v = x / w;
```

```
1  val x = 34;
2  val y = x + 1;                          (* Missing val *)
3  val z = if y > 4 then false else x < 4; (* if/else typing error *)
4  val q = if y > 0 then 0 else 1;         (* Missing else branch *)
5  val a = ~5;                             (* Used binary - *)
6  val w = 0;
7  val funn = 34;                          (* fun is a keyword *)
8  val v = x div (w+1);                    (* Can't (/) ints *)
```

Boolean Operators

The Operators

- **andalso** (same as Java's &&)
- **orelse** (same as Java's ||)
- **not** (just a function)

- Why can **not** be a function while the others cannot?
  - Because **andalso** and **orelse** may not evaluate both its left and right sides. They short-circuit evaluation.
- Be careful to always use **andalso** instead of **and**.
- **and** is completely different. We will get back to it later.

Exercises

Write the **xor** function.

Given three **int**s, return their min and max in a pair.

Write a function that computes the n^{th} Fibonacci number.

Implement a function that, given an **real** x, results in a **real** equal to the equation $x^2 - x/2 + 5$. Calculate $f(-2)$ with it.

Solutions

```
1 fun xor1 (b1 : bool, b2 : bool) = if b1 then not b2 else b2;
2 fun xor2 (b1 : bool, b2 : bool) = (b1 orelse b2) andalso
3                                   not (b1 andalso b2);
```

```
1 fun minmax (a : int, b : int, c : int) =
2     (Int.min (a, Int.min (b,c)), Int.max (a, Int.max (b,c)));
3 (* Or write a bunch of nested ifs. *)
```

```
1 fun fib (n : int) = if n < 2 then n else fib (n-1) + fib (n-2);
```

```
1 fun f (x : real) = x*x - x/2.0 + 5.0;
```