CSE 341 AB
Programming Languages

Section 1
🎉5 January 2017🎉

Adapted from slides by Konstantin Weitz, Nicholas Shahan, and Dan Grossman

Hello, it's me Ryan

4th year math undergraduate

Grew up in La Conner, Pennsylvania, and Olympia

I do research with James. We verify distributed systems.

I like cooking, going for walks, and listening to computer music (check out http://vapor.cab/)

Logistics

Join the AB group on Piazza!

I am almost always on Slack in #341-17wi.

My office hours are Wednesdays at 5pm in CSE 218, or by appointment (email me).

I will be at a conference the week after next, so there probably won't be office hours. We'll get someone to cover section.

Come👏to👏office👏hours👏

Attend lecture and section too.

You don't need a list of specific technical questions lined up before you decide to stop by 218.

I'm happy to chat about high-level concerns and questions—you just have to bring them to office hours!

Today

SML workflow

1. The REPL
2. Debugging errors
3. Emacs demo

ML details

1. Variable shadowing
2. How to use `use`
3. Boolean operators

What's the REPL do?

1. Read: ask the user for semicolon-terminated input.
2. Evaluate: try to run the input as ML code.
3. Print: show the user the result or any error messages produced by evaluation.
4. Loop.

Shadowing of variable bindings

```
val a = 1; (* a -> 1 *)
val b = a; (* a -> 1, b -> 1 *)
val a = 2; (* a -> 2, b -> 1 *)
```

Expressions in bindings are evaluated eagerly.

- Before the variable binding "finishes"
- Afterwards, the expression producing the value is irrelevant

Shadowing (using the same name for multiple variable bindings) is allowed.

- When looking up a variable, ML will use the latest binding in the current environment.

Remember: there's no way to "assign" to a variable in ML.

- Can only shadow them in a later environment.
- After binding, the variable's value is an immutable constant.

Try to avoid shadowing

```
val x = "Hello World";
val x = 2;              (* type error? *)
val res = x * 2         (* 4, or a type error? *)
```

Shadowing can be confusing and is usually considered poor style.

Reintroducing variable bindings in the same REPL session may...

- make it seem like *wrong* code is *correct*; or
- make it seem like *correct* code is *wrong*.

Using a shadowed variable

Is it ever possible to use a shadowed variable again? Well, yes and no.

You recover a shadowed binding if the more recent binding goes out of scope:

```
val x = "Hello World";
fun add1(x : int) = x + 1; (* shadow x *)
val y = add1 2;
val z = x^"!!";          (* "Hello World!!" *)
```

Use `use` wisely

`use "code.sml";` feeds the contents of `code.sml` directly into the REPL.

Previous uses of `use` on the same file will haunt your REPL session with stale bindings.

- *Restart the REPL when you want to reload a file!*

Using `use` on two different files with shared variable names will cause undesired shadowing.

- *Work with one file at a time unless you know their top-level bindings don't overlap!*

Demo!

Booleans

operation	syntax	typing rules	evaluation rules
`andalso`	`e1 andalso e2`	`e1` and `e2` must have type `bool`	same as Java's `e1 && e2`
`orelse`	`e1 orelse e2`	`e1` and `e2` must have type `bool`	same as Java's `e1 || e2`
`not`	`not e`	`e` must have type `bool`	same as Java's `!e`

- `not` is just a pre-defined function, but `andalso` and `orelse` are built into the language. They can't* be implemented as functions in ML because they "short-circuit" evaluation.
- Be careful to use `andalso` rather than `and`, which is something completely different. We will bring up `and` later in the course.

Style with booleans

* ~~Okay, we *can* implement **`andalso`** and **`orelse`** in ML, but we have to do so in terms of another "short-circuiting" construct.~~ *I said this in class, but it's not actually true: if we defined* `orelse (e1, e2)` *as a function in terms of the if expression below and invoked it, SML would still evaluate* `e1` *and* `e2` *because of its call-by-value semantics: arguments are always completely evaluated before the body of the function is evaluated.*

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

If you find yourself writing code that looks like the above, just use the appropriate operator instead. It's Good Style™.

And please don't do this:

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

Comparisons

For comparing **`int`** values:

`=` **`<>`** **`>`** **`<`** **`>=`** **`<=`**

Order comparisons (**`<`** **`<=`** **`>`** **`>=`**) may also be used with two **`real`** operands, but do not support comparing **`int`** values to **`real`** values.

Equality comparisons (**`=`** **`<>`**) can be used in any "equality type" but not with **`real`**. We'll cover equality types later in the course.