CSE 341

Section 1 (9/27)

Daniel Snitkovskiy: OH Wednesdays 1:30-2:30, CSE 4th Floor Breakout

Lanhao Wu: OH Mondays 3:00-4:00, CSE 2nd Floor Breakout

Agenda

- Introduction
- Setup: get everything running
- Emacs Basics
- ML development workflow
- Shadowing
- Debugging
- Comparison Operators
- Boolean Operators
- Testing

Introduction

Daniel Snitkovskiy

- Interested in Data Science, CS Education, and Systems.
- Information systems are all around us, and it is a hard problem to make sure that these systems are reliable, accessible, and trustworthy
- Enjoy music and playing with my dog
- I try to be approachable but that doesn’t always work out (but don’t let that stop you from asking lots of questions!)

Introduction

Lanhao Wu

- Senior student at UW CSE, interest in NLP, probabilistics.
- 341 is cool, and it’s mind changing
- Daniel’s music is pretty good IMO :)
- Feel free to ask questions! I always get inspired by others questions
- Section didn’t go as expected? Talk to us!

Icebreaker Time!

Joke Completer

- Notecards will be passed out that have statements and 1-letter identifiers in the top left corner.
  - Notecards with “S” on the top-right are “set-ups” to cheesy CS jokes, and “P” indicates a punchline.
- **How this will work:**
  - Every person with an “S” will stand-up.
  - We will go around the room, each person reading their set-up.
  - For each person that reads a set-up, the person with the matching punchline (“P” card) should also stand up and read their card.
  - After a full joke has been read, both people can sit down, and we will move onto the next person with an “S” card.
  - Repeat until every cheesy joke has been read.

Course Resources

We have a ton of course resources. Please use them!

If you get stuck or need help:

- Email the staff list! cse341-staff@cs.washington.edu

- Come to Office Hours (Every Weekday, see website)

We're here for you

Setup

Excellent guide located on the course website:
https://courses.cs.washington.edu/courses/cse341/18au/sml_emacs.pdf

You need 3 things installed:

- Emacs
- SML
- SML mode for Emacs

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)

ML Development Workflow

REPL means **R**ead **E**val **P**rint **L**oop

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

Shadowing

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

a -> int

a -> int, b -> int

a -> int, b -> int, ~~a -> 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

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?)

Debugging

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.

Comparison Operators

You can compare numbers in SML!

Each of these operators has 2 subexpressions of type `int`, and produces a `bool`

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

Boolean Operators

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

Operation	Syntax	Type-Checking	Evaluation
`andalso`	e1 `andalso` e2	e1 and e2 have type bool	Same as Java's e1 && e2
`orelse`	e1 `orelse` e2	e1 and e2 have type bool	Same as Java's e1 || e2
`not`	`not` e1	e1 has type bool	Same as Java's !e1

Technical note: andalso/orelse are SML builtins as they use short-circuit evaluation.

Testing

We don’t have a unit testing framework (too heavyweight for 5 weeks)

You should still test your code!

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

(* Neat trick for creating hard-fail tests: *)

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