

CSE 341

Section 1 (April 4th)

Lanhao Wu: Monday 3:30pm - 4:30pm, Gates 151

Alex Xu: Friday 10:30am - 12:00pm, Gates 151

Agenda

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

Icebreaker Time!


What's your name?

One ~~fun~~ fact of you. / What you've done during the spring break?

Introduction

Lanhao Wu



- BS/MS student at UW CSE, interest in NLP and PL!
- Third time TA CSE 341
- Enjoy cooking
- Use  a lot, (However, only Emacs works best for SML 🐼)
- Dongkai is my roommate

Dongkai/Alex/Sharpnel Xu

CS Senior

Took 341 two years ago,
so relearning SML/Emacs



Introduction

此段文字为占位符，用于展示排版效果。其内容为重复的无意义字符序列。

StepMania/EternaOnline
Memorizing meaningless things...
Numbers, algorithms, math

Beef noodles
Games
Made my own game...
Comics

Interests:

Messing with Johnny's Slides
Lanhao
He told me to add a slide :P

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/cse341/19sp/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;
```

a -> 1

```
val b = 2;
```

a -> 1, b -> 2

```
val a = 3;
```

a -> 1, b -> 2, a -> 3

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`?)

Because of shadowing...

Something weird could happen...

Always reopen the REPL when you need to reload a file.

- Use c-d to close the sml REPL
- Use c-c, c-s to reopen the sml REPL
- Then use “use” to load the file in
- You may use c-c, o to change the focus of Emacs

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.

SML Basic Math

Math operations:

- +
- -
- *
- / (for `floats`), e.g. `(5.0 / 2.0)`, evaluates to `2.5`
- `div` (for `ints`), e.g. `(5 div 3)`, evaluates to `1`
- `mod` (for `ints`), e.g. `(5 mod 3)`, evaluates to `2`
- `~` (negative), e.g. `~5`

Comparison Operators

You can compare numbers in SML!

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

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

Boolean Operators

You can also perform logical operations over `bools`!

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>

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

Testing

We don't have a unit testing framework (too much learning overhead)

You should still test your code!

For example:

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