Section 1 - Error Messages, Shadowing, SML Practice

This handout was composed by Porter Jones. There are probably plenty of typos/incorrect solutions/etc for you to catch! Please email me with any issues, comments. or feedback at pbjones@cs.washington.edu. All thoughts are welcome :)

Error Messages

Below is a program of SML bindings, many of which have errors. Take a few moments to find these errors. Once you find an error, **categorize it as one of the following:**

- ***Syntax error:*** **A problem with the literal writing of the SML code.** Can think of it as an issue with the physical sequence of characters the programmer wrote. Most will have an error message that begins with "Error: syntax error:"
- ***Type error:*** **Type checking failed on a particular section of code.** Usually an error message will include information about what part of the type checking failed, e.g. "Error: operator and operand do not agree" followed by the types given/expected.
- ***Evaluation error:*** **An error that occurs while evaluating SML code.**

This exercise can be completed by hand (and is great practice at understanding/recognizing SML code), but if you have your laptop handy, it would be nice to actually run some of the bindings to see the errors that are produced. Becoming familiar with the sometimes-not-so-helpful error messages SML gives will be useful for your homework assignments!

```
(* Dan Grossman, CSE341, Programming Languages *)
(* Section 1: Some Errors *)

(* This program has several errors in it so we can try to debug them. *)

val x = 34

y = x + 1

val z = if y then 34 else x < 4

val q = if y > 0 then 0

val a = -5

val w = 0

val fun = 34

val v = x / w

val 0cse341 = true
```

Shadowing

Below are three short SML programs that contain bindings that *shadow* other bindings. Shadowing can be potentially dangerous and make code much more difficult to read. Trace through each of the programs as though they were evaluated separately, and determine what values the variables are bound to at the end. For function calls, it may be helpful to remember the evaluation rules shown in lecture (see lecture slides for a refresher if needed).

`` val x = 10 val y = 16 fun f (x : int, y : int) = if x < y then ~1 else 1 val z = f (y, x) ```	``` val x = ~5 fun f (n : int) = if n < x then ~1 else 1 val y = f(3) val x = 5 val z = f(3) ```	``` val x = ~1 val y = 4 fun f_1 (n : int) = if n < (x + y) then ~3 else 3 val x = f_1 (2) val y = f_1 (8) fun f_2 (a : int, b : int, c : int) = if f_1 (y - x) < a then a else if f_1 (y - x) < b then b else c val z = f_2 (5, 6, 7) ```
x : _____ y : _____ z : _____	x : _____ y : _____ z : _____	x : _____ y : _____ z : _____

SML Practice

Implement the following SML functions. For reference, SML has a modulo operator mod which has similar syntax to div, with the expression 10 mod 3 evaluating to 1.

1. Write a function sum_of_evens that takes in an int and returns a sum of the nonnegative, even integers less than or equal to the given int. For example, sum_of_evens (7) would return 12 because the nonnegative even integers less than 7 are 0, 2, 4, and 6, which sum to 12. You may assume the given value is nonnegative.

2. Write a function to_binary that takes an int and returns an int that is the binary representation of the given value (i.e. completely made up of 1s and 0s). For example, to_binary (19) should return 10011 since that is the binary representation of 19. You may assume the given value is nonnegative.

Section 1 - Solutions

This handout was composed by Porter Jones. There are probably plenty of typos/incorrect solutions/etc for you to catch! Please email me with any issues, comments. or feedback at pbjones@cs.washington.edu. All thoughts are welcome :)

Error Messages

A fixed program appears below, with information related to the corresponding errors

```
val x = 34

(* Syntax: variable bindings must start with keyword val *)
val y = x + 1

(* Type: the types of the ‘then’ and the ‘else’ branches must be the same
   Type: the type of expression following ‘if’ must be bool *)
val z = if y > 0 then false else x < 4

(* Syntax: if-then-else expressions must have else *)
val q = if y > 0 then 0 else 42

(* Syntax: ‘-’ does not mean negative in SML, must use ‘~’ *)
val a = ~5

val w = 0

(* Syntax: fun is a keyword in SML, can’t be used for variables *)
val funny = 34

(* Type: must use ‘div’ for ints not ‘/’
   Evaluation: Cannot divide by 0
*)
val v = x div (w + 1)

(* Syntax: SML doesn’t allow variables to start w/numbers *)
val cse341 = true
```

Shadowing

It's important to remember that function calls are evaluated using an extension of the environment in which the function declaration was made!

x : 10 **y :** 16 **z :** 1	**x :** 5 **y :** 1 **z :** 1	**x :** ~3 **y :** 3 **z :** 5

SML Practice

There are many other solutions that give the correct result, below are two that I came up with!

```
fun sum_of_evens (n : int) =
    if n = 0
    then 0
    else if (n mod 2) = 0
    then n + sum_of_evens (n - 2)
    else sum_of_evens (n - 1)
```

```
fun to_binary (n : int) =
    if n < 2
    then n
    else (n mod 2) + 10 * to_binary (n div 2)
```

Section 1 - Cheat Sheet

This handout was composed by Porter Jones. There are probably plenty of typos/incorrect solutions/etc for you to catch! Please email me with any issues, comments. or feedback at pbjones@cs.washington.edu. All thoughts are welcome :)

Emacs/SML Info

Installation instructions can be located on the course website. Emacs is the supported editor for this course, and it has very nice SML support for indentation, syntax, and a REPL. You are welcome to use other editors if you feel more comfortable with them.

An SML REPL used to evaluate SML code. You can type in your own bindings and expressions, and also load them from a file with the command: use “<path-to-file>”;. All commands in the REPL must end in a semi-colon.

To avoid any complications with shadowing, anytime you have changed a SML file and wish to use the updated bindings, you should first restart the REPL completely (C-d quits the REPL) and then load in the bindings from the updated code as described above.

Emacs commands

‘C’ stands for CTRL, ‘M’ (meta) is for ALT (option on Macs)
(ex. C-s means hold down CTRL and press ‘s’)

Action	**Command**
Open a file	C-x C-f
Save a file	C-x C-s
Escape out of current command	C-g
Start SML REPL (when editing an SML file)	C-c C-s then Enter
Start SML REPL (when not editing an SML file)	M-x, type in sml-mode, then enter

More SML Stuff

Comparison operators:

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

Boolean operators:

andalso (similar to Java’s &&)	orelse (similar to Java’s ||)	not (similar to Java’s !)