CSCI 3155: Principles of Programming Languages

Exercise sheet #10

21 June 2007

Group name:__________________________

Functional Programming

This exercise is mostly a lab exercise, using Standard ML (specifically, *Standard ML of New Jersey*, which is one of the five most prominent implementations of the language).

You can interact with Standard ML using command-line actions or by EMACS interaction. The accompanying editor interaction sheet describes how to interact with SML.

Some of today's exercises ask for source code submissions. Leave your source code in a directory called `worksheet-10/$n.sml`, where `$n` is the exercise number.

SML of New Jersey won't count how often you interact with it, so feel free to experiment!

Hints:

- The easiest answers to most of the sub-exercises are very short, often only a single line.
- You can re-use your earlier results. For example, to re-use a definition you introduced in `1.sml`, start your SML program with

```
use "1.sml";
```

Exercise 1. The SML progrmaming language allows two types of constructions on its toplevel: definitions and expressions. We will first focus on the more fundamental expressions, so that you can familiarise yourselves with their syntax.

Remember that you must terminate each expression or definition with a semicolon in the interactive toplevel.

(a) What type or types does SML use for integer literals? Try entereing them into the interactive toplevel system; SML will then print out their type for you.

(b) What type does SML use for floating-point literals?

(c) What type does SML use for the values `true` and `false`?

(d) What is the type of character string literals?

(e) What is the type of `#"x"`? What do you expect this entity to represent?

(f) SML, like Python, supports *tuples*. Tuple values have the following BNF syntax:

$$\begin{array}{rcl} \langle \textit{TupleExpr} \rangle & \rightarrow & \texttt{(}\ \langle \textit{TupleElts} \rangle\ \texttt{)} \\ \langle \textit{TupleElts} \rangle & \rightarrow & \texttt{expr , exp} \\ & | & \texttt{expr ,}\ \langle \textit{TupleElts} \rangle \end{array}$$

Determine what types SML assigns to tuples.

(g) SML supports a "special tuple" `()`. Determine the type of this tuple.

Exercise 2. You can use val bindings to introduce new variables:

```
val x = ...
val y : int = ...
```

(a) Bind the variable t to a tuple of an int and a string.

(b) SML allows you to annotate arbitrary expressions with their types:

$$\begin{array}{rcl} \langle \textit{expr} \rangle & \rightarrow & \langle \textit{expr} \rangle : \mathsf{type} \\ & | & \ldots \end{array}$$

Using such type annotations, determine whether SML uses by-name or by-structure type equivalence for tuples.

(c) After binding t, use t as a stand-alone expression. What information does SML tell you about t?

(d) Try to determine the same information for the following built-in variables:

- not
- real

SML will only give you one of the two pieces of information it gave you for the tuple. What part of the information did it give you? Determine what this information means for at least one of these variables. Illustrate your observation with an appropriate example.

Exercise 3. SML supports the usual infix operators for addition, subtraction etc. SML also allows you to turn each infix operator into a regular function, by prefixing it with "op", as in "op +".

(a) Using the above hint, determine the type of "+".
(b) Determine the type of "/".
(c) Determine the type of "*".
(d) Despite what you just observed, try to multiply two floating-point literals. Explain your result.

Exercise 4. SML allows you to construct lists as e.g. [1, 2, 3].

(a) Construct a list of integer literals, as above. What is the type of this list?
(b) Construct a list of string literals. What is the type of this list?
(c) Can you construct a list that contains both integer and string literals?
(d) Construct an empty list. What is the type of this list?
(e) SML provides some helper functions for lists. Determine the types of

- hd
- tl
- null
- :: (this one is infix!)

(f) Now apply these three functions to various lists (don't forget the empty list!) to determine what they do:

- hd
- tl
- null
- ::

Exercise 5. Construct a function that returns a function. SML provides three ways to achieve this; in this exercise, you will explore two of them.

(a) Construct a function inc that takes one parameter, x, and returns a function that increments its own parameters by x, as in wednesday's lecture.

For this, you can use the let construct:

$$\begin{array}{rcl}
\langle \mathit{expr} \rangle & \to & \mathsf{let}\ \langle \mathit{decls} \rangle\ \mathsf{in}\ \langle \mathit{expr} \rangle\ \mathsf{end} \\
 & | & \ldots \\
\langle \mathit{decls} \rangle & \to & \varepsilon \\
 & | & \langle \mathit{decl} \rangle \langle \mathit{decls} \rangle \\
\langle \mathit{decl} \rangle & \to & \mathsf{val}\ \mathit{name}\ \langle \mathit{tyo} \rangle = \langle \mathit{expr} \rangle \\
 & | & \mathsf{fun}\ \mathit{name}\ \langle \mathit{parameters} \rangle \langle \mathit{tyo} \rangle = \langle \mathit{expr} \rangle \\
 & | & \ldots \\
\langle \mathit{tyo} \rangle & \to & :\ \langle \mathit{type} \rangle \\
 & | & \varepsilon
\end{array}$$

What type does your function have? Explain.

(b) Write the following function:

```
fun inc' (x) (y) = x + y
```

Determine the type of inc'. Compare the type of inc' to the type of inc. Run both through a handful of examples and try to compare their behaviour.

Exercise 6. You can use fun bindings to introduce new functions:

```
fun f(x) = ...
fun g() : int = ...
fun f(x : int, y, z) = ...
```

You do not have to specify types for the formal parameters or the returned value; SML will find these automatically (using a technique called *type inference*).

(a) Write an SML function. What type does SML give you for the function?

(b) Write a higher-order SML function that takes three parameters, f, x, and y. f should be a function parameter. Your function should return a tuple containing the result of calling f on x (for the first component of the tuple) and, separately, for f called on y (for the second component of the tuple). What type does SML give you for your function?

Exercise 7. Using the builtin hd, tl, and null functions, build some helper functions for dealing with lists. You will also need the if ... then ... else ... expression mentioned in the reading for today.

(a) Construct a function reverse that reverses an *arbitrary* list. What is the type of your function? Explain its type.

(b) Construct a function map that takes a function f as parameter and applies f to each element of the list. What is the type of your function? Explain the type.

(c) Illustrate both functions by using them to a list of integers. For map, use the function inc you constructed earlier.

(d) Standard ML includes a built-in map function. Start a fresh SML runtime without your definitions (e.g., on the command line), and determine the type of the built-in map. Compare this type to the type of your map. Are they different? If so, how, and which type do you prefer (and why)?