Exercise sheet 5
for October 6, 2004

2004-28-09

You should try to solve the exercise 8.9 from H&R Chapter 8, the exercises 9.3–9.5 from H&R Chapter 9, and exercise E5.1–E5.3. After that, try exercise 8.2 and 8.4 from H&R Chapter 8, and the exercises 9.7–9.9 from Chapter 9.

Hand-in your solution to exercise E5.1 and E5.2.

Exercise Exercise E5.1 (Expressions)

Given the datatype:

```ocaml
datatype expr =
  Num of int
  | Add of expr * expr
  | Sub of expr * expr
  | Mult of expr * expr
```

(a) Extend expressions to include identifiers.

(b) Declare a function to count the number of identifiers in an expression.

(c) Declare a function that can evaluate an expression to its value. This function needs two arguments: the expression to evaluate and an environment that gives meaning to the identifiers in the expression.

(d) Extend expressions to include simple `let` expressions. That is, a simple `let` expression just binds an identifier to an expression (no need for an extra datatype for declarations).

(e) Declare a function that can give a textual representation of an expression.
(f) Declare a function that can evaluate expressions with let expressions.
Make sure that your function can handle expressions like:

```
let val x = 40 in (let val x = 2 in x end) + x end
```

**Exercise E5.2 (Higher order functions for production of HTML code)**

(a) Declare a function:

```
htmlrow : int * (int -> string) -> string
```

that build one row of a numeric HTML table. For example:

```
htmlrow(2, fn j => Int.toString((j + 2) * 8))
```

should produce the string:

```
"<tr><td align="right">16</td><td align="right">24</td></tr>"
```

(b) Declare a function:

```
htmltable : int * (int -> string) -> string
```

that builds an HTML table. For example:

```
htmltable(3, fn i => String.concat["<td>", Int.toString i, "</td>", Int.toString(i*8), "</td>"])
```

should produce the same string as the expression:

```
"<table>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>8</td></tr>
<tr><td>2</td><td>16</td></tr>
</table>"
```

Similarly,

```
htmltable(10, fn i => htmlrow(10, fn j => Int.toString((i+1)*(j+1))))
```

should produce a 10 × 10 multiplication table in HTML.
Exercise E5.3 (Working the folds)

Use foldr and foldl to solve the following exercises.

(a) Declare a function:

```
indexOf : 'a -> 'a list -> int option
```

such that `indexOf x ls` gives SOME `i` where `i` is the least index in `ls` for which the element at position `i` equals `x`; or return NONE if `x` can not be found in `ls`.

(b) Declare a function:

```
lastIndexOf : 'a -> 'a list -> int option
```

such that `indexOf x ls` gives SOME `i` where `i` is the greatest index in `ls` for which the element at position `i` equals `x`; or return NONE if `x` can not be found in `ls`.

(c) Declare the function `flatten` from Exercise sheet 3.

(d) Declare a function that takes a list of functions and composes them, such that the first function in the list (if any) is applied last.