Principles of Programming, Fall 2009
Practice 6
Patterns in Function Definition and Type System

Woosuk Lee, Suwon Jang, Sungkeun Cho
Programming Research Lab.@SNU

October 19, 2009

1. Define a procedure `reverse` that takes a list as argument and returns a list of the same elements in reverse order.

```
> let x = reverse [1; 2; 3; 4];;
val x: int list = [4; 3; 2; 1]
```

2. Mergesort is an $O(n \log n)$ sorting algorithm invented by John von Neumann in 1945. Conceptually, mergesort works as follows:[1]

   (a) Divide the unsorted list into two sublists of about half the size.

   (b) Divide each of thw two sublists recursively until we have list sizes of length 1, in which case the list itself is returned.

   (c) Merge the two sorted sublists back into one sorted list.

   Consider the `mergeSort` procedure in the following. Define the sub procedures, `split` and `merge`, in order to run `mergeSort` well.

```
let rec mergeSort = function [] -> []
  | [a] -> [a]
  | l ->
      let (m, n) = split l in
      let m' = mergeSort m in
      let n' = mergeSort n
      in
        merge m' n'
```

3. The `diff` procedure takes a polynomial as its argument and differentiates the given polynomial. Fill in the missing expressions in the following definition of `diff`.

[1] The description of mergesort algorithms is excerpted from Wikipedia.

```
type poly = Add of poly * poly
          | Term of coef * expo
 and coef = int
 and expo = int

let rec diff : poly -> poly =
 function Add (p, q) -> <??>
  | Term (c, 0) -> <??>
  | Term (c, e) -> <??>
```

If `diff` has been completed correctly, it will have a result such as the following.

```
> let z = diff Add (Add (Term (1, 3), Term (4, 1)), Term (-5, 0)) ;;
val z: poly = Add (Add (Term (3, 2), Term (4, 0)), Term (0, 0))
```

4. Define a procedure `eval` that evaluates a polynomial in x at a given value of x. It takes a polynomial and integer as its argument.

```
> let p = Add (Add (Term (1, 3), Term (4, 1)), Term (-5, 0)) ;;
val p : poly = Add (Add (Term (1, 3), Term (4, 1)), Term (-5, 0))
> eval p 3 ;;
- : int = 34
```

5. Define a procedure `add` which can be the addition operator of four types of values. `add` plays four different parts.

   - Integer addition.
   - Floating-point addition.
   - List concatenation.
   - String concatenation.

   You can use the operators `+`, `+.`, `@`, `^` in order to define a procedure `add` The types of `add` can be described as follows.

```
val add : 'a value -> 'a value -> 'a value

type 'a value = INT of int
   | FLOAT of float
   | LIST of 'a list
   | STRING of string
```