CSE216
Foundations of Computer Science

State University of New York, Korea

Review on
fold_left, fold_right

```
# List.fold_left ;;
- : ('acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc = <fun>
# List.fold_right ;;
- : ('a -> 'acc -> 'acc) -> 'a list -> 'acc -> 'acc = <fun>
```

1.

- Implement rev: ‘a list ->’a list using fold_left

2.

- Implement last: ‘a list ->’a option using fold_left

```
# last ["a"; "b"; "c"; "d"];;
- : string option = Some "d"
# last [];;
- : 'a option = None
```

3.

- Implement the following using fold_left
  - `contains : 'a -> 'a list -> bool` that returns `true` if and only if the list contains the specified element. Do not use any pre-existing functions.

```
# contains 4 [3; 4; 5];;
- : bool = true
```

4.

- Choose either fold_left or fold_right for implementing the following:
  - `evens : 'a list -> 'a list` which returns the 0th, 2nd, 4th, etc., elements of a list.

```
# evens [13; 5; 9; 0; 7; 8];;
- : int list = [13; 9; 7]
```

Review on types

Taken from

https://courses.grainger.illinois.edu/cs421/su2009/exams/mt-sampleqns.pdf

Thanks!

1. Give the types of each of the following Ocaml functions:

(a)
```
let alwaysfour x = 4
```

(b)
```
let add x y = x + y
```

(c)
```
let concat x y = x ^ y
```

(d)
```
let addmult x y = (x + y, x * y)
```

```
(e) let rec f x = if x=[] then [] else hd x @ f (tl x)

(f) let rec copy x = if x=[] then [] else hd x :: copy (tl x)

(g) let b (x,y) = x+y

(h) let c (x,y) = x
```

```
(i) let d x = match x with (a,b) -> a
```

```
(j) let e x = hd x + 1
```

```
(k) let f x y = match x with
        [] -> 0 | a::b -> a+y
```

```
(l) let g (a,b) (c,d) = (a+d, b^c)
```

(Recall that ^ is the string concatenation operation.)

```
(m) let rec h x = match x with
          [] -> 0 | (a,b)::r -> a + (h r)
```

Review on Algebraic Datatype

Taken from SBU cse216. Thanks!

1. Think of a binary tree to represent basic arithmetic operations. In such a tree, the leaf nodes will be of a numeric type while the internal non-leaf nodes will hold the arithmetic operations of addition, subtraction, multiplication, and division. This means, we need to define a binary tree data type over two distinct data types. Define such a binary tree in OCaml.

2. Next, write down a function that takes a binary tree of the type you defined, and returns a tuple with its first element being a list of operators, and its second element being a list of the numeric values in the nodes. That is, the function's type should be

```
val function name : ('a, 'b) tree -> 'b list * 'a list = <fun>
```

where 'a is the operator's type and 'b is the numeric type.