CSE216
Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

Many slides taken from Cornell's CS3110. Thanks!
https://www.cs.cornell.edu/courses/cs3110/2014fa/lecture_notes.php

Recitation (Lab)

How it proceeds

- You work for ~40 mins
- Feel free to work in groups
- Feel free to ask questions
- We will explain ideas in the last 5-10 minutes
- These problems, excluding very simple ones, will become a part of this week’s homework

Exercise 1

- Suppose a weighted undirected graph is represented as a list of edges. Each edge has a triple of the type string * string * int, where the two nodes are represented by strings, and the weight is an integer.
- 1. write a type *edge* to represent an edge, and a type *graph* to represent a weighted undirected graph.
- 2.Construct a weighted undirected graph of type *graph*
- 3. Write an OCaml function of type *graph -> edge* option to identify the minimum weight edge in this graph. Use pattern matching to solve this problem.

Exercise 2

Exercise 2 Binary trees can be defined as follows:

```
type btree = Empty | Node of int * btree * btree
```

For example, the following `t1` and `t2`

```
let t1 = Node(1,Empty,Empty)
let t2 = Node(1,Node(2,Node(3,Empty,Empty),Empty),Node(4,Empty,Empty))
```

are binary trees.

Write a function

```
mirror: btree -> btree
```

that exchanges the left and right subtrees all the ways down. For example,

```
mirror t1 = Node (1, Empty, Empty)

mirror t2 = Node(1,Node(4,Empty,Empty),Node(2,Empty,Node(3,Empty,Empty)))
```

Exercise 3

Exercise 3 Natural numbers can be defined as follows:

```
type nat = ZERO | SUCC of nat
```

For instance, `SUCC ZERO` denotes 1 and `SUCC (SUCC ZERO)` denotes 2. Write three functions that add, multiply, exponentiate natural numbers:

```
natadd : nat -> nat -> nat
natmul : nat -> nat -> nat
natexp : nat -> nat -> nat
```

For example,

```
# let two = SUCC (SUCC ZERO);;
val two : nat = SUCC (SUCC ZERO)
# let three = SUCC (SUCC (SUCC ZERO));;
val three : nat = SUCC (SUCC (SUCC ZERO))
# natadd two three;;
- : nat = SUCC (SUCC (SUCC (SUCC (SUCC ZERO))))
# natmul two three;;
- : nat = SUCC (SUCC (SUCC (SUCC (SUCC (SUCC ZERO)))))
# natexp two three;;
- : nat = SUCC (SUCC (SUCC (SUCC (SUCC (SUCC (SUCC (SUCC ZERO)))))))
```

Exercise 4

Exercise 4 Write a function

```
diff : aexp * string -> aexp
```

that differentiates the given algebraic expression with respect to the variable given as the second argument. The algebraic expression **aexp** is defined as follows:

```
type aexp =
 | Const of int
 | Var of string
 | Power of string * int
 | Times of aexp list
 | Sum of aexp list
```

For example, $x^2 + 2x + 1$ is represented by

```
Sum [Power ("x", 2); Times [Const 2; Var "x"]; Const 1]
```

and differentiating it (w.r.t. "x") gives $2x + 2$, which can be represented by

```
Sum [Times [Const 2; Var "x"]; Const 2]
```

Note that the representation of $2x + 2$ in **aexp** is not unique. For instance, the following also represents $2x + 2$:

```
Sum
 [Times [Const 2; Power ("x", 1)];
  Sum
   [Times [Const 0; Var "x"];
    Times [Const 2; Sum [Times [Const 1]; Times [Var "x"; Const 0]]]];
  Const 0]
```