# Homework 1
# COSE212, Fall 2017

Hakjoo Oh

**Due: 10/15, 24:00**

**Academic Integrity / Assignment Policy**

- *All assignments must be your own work.*
- Discussion with fellow students is encouraged including how to approach the problem. However, your code must be your own.
  - Discussion must be limited to general discussion and must not involve details of how to write code.
  - You must write your code by yourself and must not look at someone else's code (including ones on the web).
  - Do not allow other students to copy your code.
  - Do not post your code on the public web.
- Violating above rules gets you 0 points for the entire HW score.

**Problem 1** (10pts) Consider the task of computing the exponential of a given number. We would like to write a function that takes as arguments a base $b$ and a positive integer exponent $n$ to compute $b^n$. Read the remaining problem description carefully and devise an algorithm that has time complexity of $\Theta(\log n)$.

One simple way to implement the function is via the following recursive definition:

$$\begin{array}{rcl} b^0 & = & 1 \\ b^n & = & b \cdot b^{n-1} \end{array}$$

which translates into the OCaml code:

```
let rec expt b n =
  if n = 0 then 1
  else b * (expt b (n-1))
```

However, this algorithm is slow; it takes $\Theta(n)$ steps.

We can improve the algorithm by using successive squaring. For instance, rather than computing $b^8$ as

$$b \cdot (b \cdot (b \cdot (b \cdot (b \cdot (b \cdot (b \cdot b))))))$$

we can compute it using three multiplications as follows:

$$\begin{aligned} b^2 &= b \cdot b \\ b^4 &= b^2 \cdot b^2 \\ b^8 &= b^4 \cdot b^4 \end{aligned}$$

This method works only for exponents that are powers of 2. We can generalize the idea via the following recursive rules:

$$\begin{aligned} b^n &= (b^{n/2})^2 && \text{if } n \text{ is even} \\ b^n &= b \cdot b^{n-1} && \text{if } n \text{ is odd} \end{aligned}$$

Use the rules to write a function `fastexpt` that computes exponentials in $\Theta(\log n)$ steps:

```
fastexpt: int -> int -> int
```

**Problem 2** (10pts) Write a function

```
smallest_divisor: int -> int
```

that finds the smallest integral divisor (greater than 1) of a given number $n$. For example,

```
smallest_divisor 15 = 3
smallest_divisor 121 =11
smallest_divisor 141 = 3
smallest_divisor 199 = 199
```

Ensure that your algorithm runs in $\Theta(\sqrt{n})$ steps.

**Problem 3** (10pts) Define the function `iter`:

```
iter : int * (int -> int) -> (int -> int)
```

such that

$$\texttt{iter}(n, f) = \underbrace{f \circ \cdots \circ f}_{n}.$$

When $n = 0$, $\texttt{iter}(n, f)$ is defined to be the identity function. When $n > 0$, $\texttt{iter}(n, f)$ is the function that applies $f$ repeatedly $n$ times. For instance,

```
iter(n, fun x -> 2+x) 0
```

evaluates to $2 \times n$.

**Problem 4** (10pts) Write a higher-order function

```
product : (int -> int) -> int -> int -> int
```

such that `product f a b` computes

$$\prod_{i=a}^{b} f(i).$$

For instance,

```
product (fun x -> x) 1 5
```

evaulates to 120. In general, we can use `product` to define the factorial function:

```
fact n = product (fun x -> x) 1 n
```

**Problem 5** (10pts) Use `product` to define a function

```
dfact : int -> int
```

that computes double-factorials. Given a non-negative integer $n$, its double-factorial, denoted $n!!$, is the product of all the integers of the same parity as $n$ from 1 to $n$. That is, when $n$ is even

$$n!! = \prod_{k=1}^{n/2} (2k) = n \cdot (n-2) \cdot (n-4) \cdots 4 \cdot 2$$

and when $n$ is odd,

$$n!! = \prod_{k=1}^{(n+1)/2} (2k-1) = n \cdot (n-2) \cdot (n-4) \cdots 3 \cdot 1$$

For example, $7!! = 1 \times 3 \times 5 \times 7 = 105$ and $6!! = 2 * 4 * 6 = 48$.

**Problem 6** (10pts) Write a function `drop`:

```
drop : 'a list -> int -> 'a list
```

that takes a list $l$ and an integer $n$ to take all but the first $n$ elements of $l$. For example,

```
drop [1;2;3;4;5] 2 = [3; 4; 5]
drop [1;2] 3 = []
drop ["C"; "Java"; "OCaml"] 2 = ["OCaml"]
```

**Problem 7** (10pts) Write a function

```
unzip: ('a * 'b) list -> 'a list * 'b list
```

that converts a list of pairs to a pair of lists. For example,

```
unzip [(1,"one");(2,"two");(3,"three")] = ([1;2;3],["one";"two";"three"])
```

**Problem 8** (30pts) Consider the problem of counting the number of different ways of making coin-changes of a given amount of money. For example, when three types of coins (1, 5, 10 wons) are available, there are four different ways of making changes of 12 won:

$$
\begin{aligned}
&12 \text{ won} = 10 \text{ won} * 1 + 1 \text{ won} * 2\\
&12 \text{ won} = 5 \text{ won} * 2 + 1 \text{ won} * 2\\
&12 \text{ won} = 5 \text{ won} * 1 + 1 \text{ won} * 7\\
&12 \text{ won} = 1 \text{ won} * 12
\end{aligned}
$$

Write a function

```
change: int list -> int -> int
```

that takes a list of the denominations of the coins and an amount of money to change, and returns the number of ways to make changes. For example,

```
change [1;5;10] 12 = 4
change [1;5;10;25;50] 100 = 292
```

Note that special cases are defined as follows:

- When the amount is 0, we count that as 1 way to make change: e.g.,

  ```
  change [1;5;10] 0 = 1
  ```

- When the amount is less than 0, we count that as 0 ways to make change: e.g.,

  ```
  change [1;5;10] -5 = 0
  ```

- When the number of coin kinds is 0, we count that as 0 ways to make change: e.g.,

  ```
  change [] 10 = 0
  ```