Test 2

NAME:_____ STUDENT NO:_____

1. (25%) Given the following language for specifying types in Haskell:

```
<Type>  ::=  Integer  | <TVar> | <Type> "->" <Type>  |  "[" <Type> "]"
<TVar> ::=  "a"  |  "b"  |  "c"   |   "d"
```

What is the *most general* type of `reduce`? Assume that `head` and `tail` are functions on lists. Explain as the best you can.

```
reduce  f  l  i =
    if l == [] then i
    else f (head l) (reduce  f  (tail l)  i)
```

2. (25%)

   (a) (10%) Using `reduce` from Question (1): what does the following expression return? Explain.

   ```
   reduce  (+)  [1..5]   0
   ```

   (b) (15%) Given a list of list of integers, we want to flatten this list into a single list, i.e., from

   ```
   [ [1,2], [3,4], [5,6,7], [8,9], [0] ]
   ```

   to

   ```
   [ 1,2,3,4,5,6,7,8,9,0 ]
   ```

   Discuss how to compute this function using `reduce`.

3. (25%) Reduce the following Lambda Expression into its *normal form*. Show all your steps.

$$(\lambda f.(f(\lambda x.\lambda y.x))) ((\lambda x.\lambda y.\lambda f.f\ x\ y)\ a\ b)$$

4. (25%)

   (a) (10%) Write a function `prods` which computes the element-by-element products of two input lists, i.e., `prods [2,3,4] [2,3,4] = [4,9,16]`.

   (b) (15%) You are given a function `ints` which returns an infinite list of integers starting from `n`

   ```
   ints  n  =  n : (ints (n+1))
   ```

   Write a recursive function `facs` which uses `ints` and `prods` and returns an infinite list of factorials, i.e., `[fac(0), fac(1), fac(2), fac(3), ...]`, where `fac` is the factorial function. (**Note**: You are now allowed to use an auxiliary `factorial` function. Use `ints` and `prods` only.)