Test 2

NAME:____________________ STUDENT NO:____________________

1. (20%) Given the following `map` function in Haskell,

```
map  f  l  =
    if (null l) then l
    else (f (head l)) : (map  f  (tail l))
```

What is the expected value for each of the following expressions?

(a) (5%)

```
map (\x-> [1..x]) [1..5]
```

(b) (5%)

```
map (\x-> if (odd x) then 1 else 0) [1..5]
```

(c) (5%)

```
map (\x-> x++x) (map (\x-> [x]) [1..5])
```

(d) (5%)

```
map (\x-> \y -> x+y)) [1..5]
```

2. (20%) Given the following `filter` function in Haskell,

```
filter f l  =
    if (null l) then l
    else if (f (head l))
           then (head l) : (filter f (tail l))
           else (filter f (tail l))
```

Use `filter` and/or `map` to construct an expression or a function to solve the following problems.

(a) (10%) You are given a mathematical integer function f, show how you can calculate all its roots, i.e., for all x such that $f(x) = 0$, x ranges from 0 to 100.

(b) (10%) Two functions f and g *intersect* if there exists an x such that $f(x) = g(x)$. You are given a list of functions $f_1, f_2, ..., f_n$ and another function g. Find all functions f_i such that f_i and g intersect in the range between 0 and 100. (Note: Assume all f_i and g are unary functions.)

3. (20%) Reduce the following Lambda Expression into its *normal form* (i.e., no more redexes). Show **all** your steps.

$$(\lambda m.\lambda n.\lambda f.m\ (n\ f))\ (\lambda f.\lambda x.x)\ (\lambda f.\lambda x.f\ x)$$

4. (20%)

   (a) (5%) What is α-conversion? Explain with a simple example.

   (b) (5%) Haskell is a *non-strict* functional programming language. What does it mean by *non-strict*? Explain.

   (c) (5%) Explain the difference between *normal*-order and *applicative*-order reduction strategies of Lambda Calculus.

   (d) (5%) What is *currying*? Explain with an example.

5. (20%)

   (a) (5%) Write a function `length` which computes the length of a list, e.g. `length [1,2,3,4,5] => 5`.

   (b) (5%) A one-step *convolution* of two integer vectors $<x_1, x_2, ..., x_n>$ and $<y_1, y_2, ..., y_n>$ is defined by:

   $$z = x_1 * y_1 + x_2 * y_2 + ... + x_n * y_n$$

   Write a function `mac` (multiply-and-accumulate) which computes this one-step convolution of two lists. You may assume both lists are of equal length. (e.g., `mac [1,2,3] [1,2,3] => 1 + 4 + 9 => 14`)

(c) (10%) Now, consider the general case of *convolution*, where $<x_1, x_2, ..., x_n>$ is a fixed vector of length n, and $<y_1, y_2, ...>$ is an infinite vector of integers, a *1st-order* convolution of x over y is defined by:

$$z_i = x_1 * y_i + x_2 * y_{i+1}$$

where z_i is the ith-element of the output vector. A *2nd-order* convolution is defined by:

$$z_i = x_1 * y_i + x_2 * y_{i+1} + x_3 * y_{i+2}$$

Write a function `convolute` which takes a **fixed** length list `x`, an **infinite** list `y`, and an integer `n`, and computes the `n`-th order convolution of `x` over `y`. You may assume that `n` is less than the length of `x`. For example,

```
convolute [1,1,1] [1..] 2 => [ 1+2+3, 2+3+4, 3+4+5, ... ]
convolute [1,0,0] [1..] 2 => [ 1,2,3,4, ... ]
convolute [1,1,1] [1..] 1 => [ 1+2, 2+3, 3+4, ... ]
```