Principles of Programming 2011 Fall
Practice 6
HW5, Remind recursion and iteration

ROPAS
Seungjung Lee, Youngseok Lee

2011/10/31

1. Verification of "prefix-free" character.

   Everytime we finish the homework, we concern if this homework is right or not. So let's make a function which verify our homework is doing fine.

   Let's make `is-prefix-free-list?` function. This function take a list of the codes and check it follows "prefix-free". The code is the list of 0 or 1. Before make this function, first thing we should do is that making a function of checking the single code "prefix-free" with other codes. The function `is-prefix-free-list?` uses that function for each code in the list.

   It is not necessary to make a function with following way.

```
(define (is-prefix-free? path path_list)
   ... )

(define (is-prefix-free-list? path_list)
   ... )
```

2. Let's check following two functions to calcuate $n!$ and think what is the difference of them.

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))


(define (factorial n)
  (define (iter x count)
    (if (> count n)
        x
        (iter (* count x) (+ count 1))))
  (iter 1 1))
```

3. As we seen before, recusion functions which have several calls themselves has a problem that do the same calculation everytime.

   The following function calcuates fibonacci sequence.

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))

> (fib 35)
9227465
> (fib 100)
...
```

   Let's make a function which has a same operation with iteration fill the `<??>` part.

```
(define (fib n)
  (define (iter a b count)
    (if (= count <??>)
        <??>
        (iter <??> (+ count 1)))
  (iter <??>))

> (fib 35)
9227465
> (fib 100)
...
```

4. Let's make `enumerate-pair` function. This function takes two natural numbers a와 b ($a < b$) as inputs and returns a list of pair (i, j) where i and j is more or equal than a, less or equal than b.

```
> (enumerate-pair 0 2)
((0 . 0) (0 . 1) (0 . 2) (1 . 0) (1 . 1) (1 . 2) (2 . 0) (2 . 1) (2 . 2))
```

   And also, make a function `enumerate-pair2` that follow $i < j$

```
> (enumerate-pair2 0 2)
((0 . 1) (0 . 2) (1 . 2))
```

5. Let's make function `triple-sum`. It takes two integer n and s as an input. If there is three different number $i, j, k (i < j < k)$, it returns a list of these three numbers when the sum of the numbers are same with s.

```
> (triple-sum 7 12)
((1 4 7) (1 5 6) (2 3 7) (2 4 6) (3 4 5))
```