Programming Principles

Homework No. 2 [24 points]

Due: October 22, 2018 10:30AM

Write your report in English. Submit your written paper report and racket file for the problems that are specified in the problem description. When "run" button is pushed, your interaction window must show your testing results. If you need to write your solution on the paper as specified, write on the paper. Submit your Racket file (in a single file) to TA (email: 204606131@qq.com), so that TA can test your program. Your file name must be "PP_hw2_*your_student_id_number.rkt*". Submit your answer sheet (if any) at the class. Don't forget documenting your programs. Make the first comment of your program be your student id and name, so that TA can find who you are.

1. a. [3 points] Write a procedure `(nmult n)` that takes one argument, a positive integer, and produces the number of multiplications required to raise something to that power using **`fast-expt`**. You must write the program **`nmult`** with Dr.Racket and test your program with n =1, 5, 7, 8. Submit your program to TA. **`fast-expt`** is defined as follows:

```
(define (fast-expt b n)
   (cond ((= n 0) 1)
         ((even? n) (square (fast-expt b (/ n 2))))
         (else (* b (fast-expt b (- n 1))))))
```

   b. [3 points] Can you determine a more concise description of the function computed by the procedure **`nmult`** than the program itself? Write your description on the paper.

   c. [3 points] Show that for $n > 1$, **`(numlt n)`** $\leq 3 \log n$ (to the base 2).
   Write your solution on the paper.

2. [3 points] Louis Reasoner is having great difficulty with the **nmult** procedure he wrote for the exercise above. No matter what argument *n* he gives it, it tells him that *n* multiplications are required to raise something to the *n*-th power using **`fast-expt`**. Louis calls his friend Eva Lu Ator over to help. When they examine Louis's code, they find that he has rewritten the **`fast-expt`** procedure to use an explicit multiplication, rather than calling **`square`**:

```
(define (fast-expt b n)
  (cond ((= n 0) 1)
        ((even? n) (* (fast-expt b (/ n 2))
                      (fast-expt b (/ n 2))))
        (else (* b (fast-expt b (- n 1))))))
```

His **nmult** procedure is based on this implementation. "I don't see what difference that could make," says Lois. "I do." says Eva. "By writing the procedure like that, you have transformed the Θ(log *n*) process into an Θ(*n*) process." Explain. Write your explanation on the paper.

3. [6 points] Exercise 1.31(a) on the page 60 of the text book(SICP 2nd Ed.).
   You must define **(product** `term a next b`**)**, **(factorial** `n`**)**, and **(pi-product** `n`**)** with Dr.Racket so that TA can test your programs. (Submit by email.)

4. [3 points] Exercise 1.32 (b) on the page 61 of the textbook. Submit your program by email.
   You must define your function like the following template:

```
(define (accumulate combiner null-value term a next b)
   (define (accumulate-iter a result)
      ..................................
    )
   (accumulate-iter a null-value))
```

5. [3 points] Exercise 1.41 on the page 77 of the text book.
   You must define **(double** `fun`**)** as specified in the problem. Submit your program by email.