CS220 Programming Principles

Homework No. 2 [30 points]

Due: April 3, 2017 9:00AM

Don't forget documenting your programs. You may write comments in Korean or English.

1. [3 points] Exercise 1.31(a) on the page 60 of the text book(SICP 2nd Ed.).
2. [3 points] Exercise 1.32 (b) [iterative process only] on the page 61 of the textbook.
3. [3 points] Exercise 1.35 on the page 70 of the text book.
4. [3 points] Exercise 1.41 on the page 77 of the text book.

5. [18 points] The **`square`** procedure computes the square of a number and the **`sqrt`** procedure computes the square root of a number. Mathematically, these are inverse functions – that is, $\sqrt{n^2} = \sqrt{n}^2$ for all nonnegative *n*. But on the computer, which cannot represent numbers with infinite precision, **`square`** and **`sqrt`** will not be perfect inverses. If we take the square root of a number and square the result, we might not get back exactly the original number. In this exercise, we will write procedures to help investigate the behavior of inverse procedures.

a. Write a procedure **`sqrt-of-square`** that takes a number as argument and tests whether the square root of the square of the number is equal to the original number. Try it on a few numbers. For example, try (**`sqrt-of-square 4.1).`**

b. Write a procedure **`square-of-sqrt`** that takes a number as argument and tests whether the square of the square root of the number is equal to the original number. Try it on a few numbers. For example, try (**`squar-of-sqrt 5.0).`**

c. We might want to test other pairs of inverse procedures, not just **`sqrt`** and **`square`**. Abstract the idea of testing a pair of inverse procedures by writing a procedure **`inverse-test`** that can be used as follows:
`(inverse-test square sqrt 4.0)` should be the same as
`(square-of-sqrt 4.0)` and
`(inverse-test sqrt square 4.0)` should be the same as
`(sqrt-of-square 4.0)`.

Try **inverse-test** on some of the same numbers you tried above.

d. In testing different numbers with the same procedures, it is a nuisance to keep typing the same procedure names as arguments to **inverse-test**. Write a procedure **make-inverse-test** that takes two procedures as arguments and returns an inverse-testing procedure. For example, if we

```
(define square-of-sqrt (make-inverse-test square sqrt))
(define sqrt-of-square (make-inverse-test sqrt square))
```

then **square-of-sqrt** and **sqrt-of-square** can be used to test numbers just as they were above.

e. So far we can only test one-argument procedures; but we might also want to test two-argument procedures. For example, does $(a/b)\cdot b=a$? Does $(a\cdot b)/b=a$? Does $(a+b)-b=a$? Does $(a-b)+b=a$? Write a procedure **binary-inverse-test** that is like **inverse-test** except that it takes two numbers instead of one. For example,
(binary-inverse-test / * 3.0 4.0) should test whether
$(3.0\cdot 4.0)/4.0=3.0$ and
(binary-inverse-test * / 3.0 4.0) should test whether
$(3.0/4.0)\cdot 4.0=3.0$.

f. By analogy with **make-inverse-test** (part d), write a procedure called **make-binary-inverse-test** that takes two procedures as arguments and returns a two-argument inverse-testing procedure. For example, if we

```
(define */test (make-binary-inverse-test * / ))
```

then **(*/test 3.0 4.0)** should do the same thing as
(binary-inverse-test * / 3.0 4.0).