프로그래밍의 원리 2012 가을
실습 2
함수를 인자로 받는 함수, 타입, 계산복잡도

서울대학교 프로그래밍 연구실
강동옥

2012년 9월 20일

연습문제

1. even-list 구현의 빈칸을 채워넣어 봅시다.

```
(define (iseven n) (= (modulo n 2) 0))
(define (even-list items)
  (if (null? items)
      '()
      (cons <??> (even-list <??>)))
```

이 함수는 정수 리스트를 입력으로 받아서 각 정수가 짝수인지 여부를 리스트로 돌려줍니다.

```
(even-list '(1 2 3 4))
(#f #t #f #t)
```

2. even-list는 map을 사용하면 쉽게 만들 수 있습니다. 이 함수는 리스트의 각 원소에 인자로 받은 함수를 적용하여 그 결과를 리스트로 만들어 줍니다.

```
(define (even-list items)
  (map iseven items))
```

map과 똑같이 동작하는 my-map을 만들어 봅시다.

```
(my-map abs (list -1 -2 -3 4))
> (1 2 3 4)
```

3. fold 함수는 리스트와, 함수, 초기값을 입력으로 받습니다. 그리고 다음의 연산을 수행합니다.

$$\begin{aligned}(\texttt{fold}'(a_1\ \cdots\ a_n)\ f\ c\ ) &= (f\ a_1\ (f\ a_2\ (\ \cdots\ (f\ a_n\ c)\ \cdots\ ))) \\ (\texttt{fold}'()\ f\ c\ ) &= c\end{aligned}$$

풀어서 쓰면 다음과 같습니다.

```
(define (fold lst f c)
  (if (null? lst) c
      (f (car lst) (fold (cdr lst) f c))))

(fold '(1 2 3) + 0)
 = (+ 1 (fold '(2 3) + 0))
 = (+ 1 (+ 2 (fold '(3) + 0)))
 = (+ 1 (+ 2 (+ 3 (fold '() + 0 ))))
 = (+ 1 (+ 2 (+ 3 0)
```

주어진 fold 함수와 even-list, or 함수를 사용해서 어떤 정수 리스트에 짝수가 있는지를 알아보는 함수 has-even 을 만들어 봅시다. 사용법은 다음과 같습니다.

```
(define (has-even l) (...))

(has-even '(1 2 3 5))
#t
(has-even '(-1 1 3))
#f
```

4. 사전(dictionary)은 키워드와 그에 대응하는 값을 쌍으로 가지고 있습니다.
   사전을 만들고/사용하는 다음의 함수들을 정의 할 것입니다:

   empty-dic : *dictionary*
   add-dic : $dictionary \times string \times \tau \rightarrow dictionary$
   find-dic : $dictionary \times string \rightarrow \tau$

   dictionary 타입을 여러가지로 구현 할 수 있겠습니다. 다음과 같은 두가지 방식으로도 구현해 볼 수 있습니다. 한가지는 (키워드, 값)리스트로 만들 수 있습니다. 한가지는 (키워드→값) 함수로 만들 수 있습니다. 타입을 생각하면서 다음 함수들을 완성해 보세요.
   (find-dic에서 값을 못찾으면 "no such key"에러를 출력해줍니다.)

```
(define empty-dic-p '())
(define (add-dic-p dic key v)
  (cons (cons key v) dic)
  )
(define (find-dic-p dic key)
  ...
  )

(define empty-dic-f (lambda (key) (error "no such key")))
(define (add-dic-f dic search_key v)
  (lambda (key)
    (if ...)
    )
  )
(define (find-dic-f dic key)
  ...
  )
```

제대로 구현했다면 둘 모두 다음과 같은 식이 같은 결과를 줄 것입니다.

```
(define dic (add-dic (add-dic (empty-dic) "bc" "good") "a" 123))
(find-dic dic "bc")
(find-dic dic "zoo")
```

5. exp는 밑과 지수를 받아서 제곱 결과를 계산해줍니다. 어떻게 구현하느냐에 따라서 시간 복잡도가 $\theta(n)$이 될 수도 있고 $\theta(\log n)$이 될 수도 있습니다. 두가지 경우 모두 만들어 봅시다.

```
(define (expn b n)
...
)

(define (iseven n) (= (modulo n 2) 0))
(define (square n) (* n n))
(define (explogn b n)
  (cond ((= n 0) 1)
        ((= n 1) b)
        ...
        )
  )
```