프로그래밍의 원리 2012 가을 - 실습 6
함수 테이블 만들기

서울대학교 프로그래밍 연구실
강동옥, 최민아

2012년 10월 24일

이번 실습의 목적은:

- 물건 중심 프로그래밍(imperative programming)을 연습해 본다.
- 수업시간에 배운 함수 테이블을 만들어 본다.

이번 시간에는 물건 중심 프로그래밍에 대해 공부 할 것입니다. 대표적인 명령형 함수로 set!, set-car!, set-cdr!이 있습니다. 이 세가지 함수에 대해 먼저 연습해봅시다.

- set!은 값을 변화시킵니다.

```
> (define x 1)
> x
1
> (set! x 4)
> x
4
> (define (f y) (+ y x))
> (f 1)
5
> (set! x 5)
> (f 1)
6
```

  함수 `f`를 `x`를 사용하여 정의 한 뒤에 `x`값을 변경했습니다. 변경 전에는 `f`가 `y+4`를 수행하는 함수였는데 `f`가 `y+5`를 수행하는 함수로 바뀐것을 보실 수 있습니다.

- set-car!, set-cdr!은 반드시 페어를 받습니다. 페어를 받아서 각각의 값을 변화시킵니다.

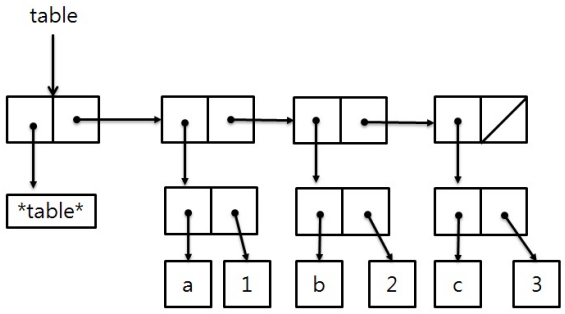
```
> (define p (cons 1 3))
> p
(1 . 3)
> (set-car! p 4)
> p
(4 . 3)
> (set-cdr! p 6)
> p
(4 . 6)
```

이와같이 페어의 값이 변하는것을 보실 수 있습니다. 명령형 프로그래밍을 하면 car, cdr로는 구현하기 어려운 것들을 구현 할 수 있게 됩니다. 연습문제를 통해서 lookup table을 구현해 봅시다.

연습문제

1. 1차원 lookup table 만들기 (컴퓨터 프로그램의 구조와 해석 346쪽)

   기본적으로 우리가 만들 표의 구조는 (열쇠 × 값)들의 리스트입니다.



나중에 2차원 테이블을 만들기 용이하도록 *table*이라는 테이블 이름을 추가로 넣어준 모양입니다. → (list '*table* ("a" . 1) ("b" . 2) ("c" . 3))

빈 테이블을 만드는 함수를 다음과같이 구현할 수 있겠습니다.

```
(define (make-table table-name) (list table-name))
```

이제 lookup함수를 만들기 전에 assoc 함수를 먼저 정의하겠습니다. assoc 연산은 열쇠와 (열쇠 × 값) 리스트를 받아 해당하는 (열쇠 × 값)을 결과로 내놓습니다. 열쇠가 없으면 false를 내놓습니다.

```
(define (assoc key records)
  (cond ((null? records) false)
        ((equal? key (caar records)) (car records))
        (else (assoc key (cdr records)))))
```

assoc 연산을 이용해서 lookup 함수를 만들어 봅시다. lookup 연산은 **열쇠**와 (**열쇠**×**값**) 리스트를 받아 해당하는 **값**을 결과로 내놓습니다. **열쇠**가 없으면 false를 내놓습니다. 스킴은 if문에서 조건절에 false가 아니면 모두 true로 생각합니다. 이를 이용해 lookup을 완성하실 수 있습니다.

```
(define (lookup key table)
  (define record (assoc key (cdr table)))
  (if ...
      (...)
      ...))
```

insert!는 **열쇠**와 **값**을 받아서 table에 이미 그 값이 있으면 값을 바꿔주고 그렇지 않으면 새로 추가해줍니다.

```
(define (insert! key value table)
  (define record (assoc key (cdr table)))
  (if ...
      (set-cdr! ...)
      (set-cdr! table
                (cons (...) (...))))
  )
```

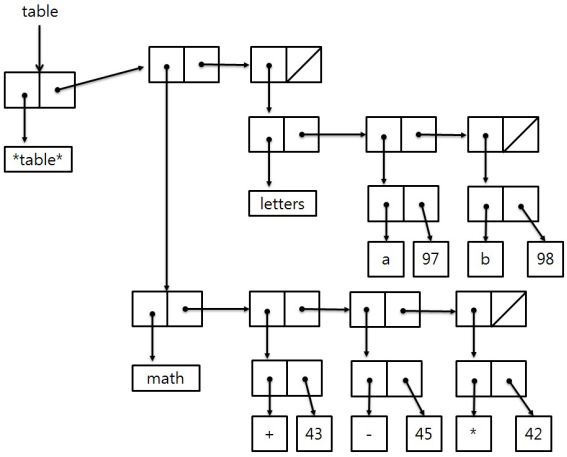
자 이제 lookup과 insert!가 잘 만들어졌는지 테스트 해보세요.

```
(define t1 (make-table "table"))
(lookup "a" t1)
(insert! "a" 1 t1)
(lookup "a" t1)
(insert! "b" 2 t1)
t1
(insert! "a" 4 t1)
t1
```

2. 2차원 lookup table 만들기 (컴퓨터 프로그램의 구조와 해석 348쪽)

   2차원 lookup table은 1차원 lookup table을 조합하여 다음과같이 만들 수 있습니다. 1차원 테이블의 이름을 **열쇠**, 레코드 리스트를 **값**으로 보고 조합 하면 됩니다. 이런 형태가 되겠지요. (list '*table* ("math" . records1) ("letters" .

records2))



1차원 lookup 함수와 insert!함수를 조합하여 lookup2함수와 insert2!함수를 만들어 봅시다.

```
(define (make-table2 table-name) (list table-name))
(define (lookup2 key-1 key-2 table2)
  (define table1 (assoc key-1 (cdr table2)))
  (if ...
      (lookup ... ...)
      ...
      )
  )

(define (insert2! key-1 key-2 value table2)
  (define table1 (assoc key-1 (cdr table2)))
  (define record (if table1 (assoc key-2 (cdr table1)) false))
  (if ...
      (... ... ...)
      (if ...
          (insert! ... ... ...)
          (set-cdr! ...
                    (cons (cons key-1
                                (cons (cons key-2 value)
                                      ()))
                          (cdr table2)))))
  )
```

자 이제 lookup2과 insert2!가 잘 만들어졌는지 테스트 해보세요.

```
(define t2 (make-table2 "table"))
(lookup2 "a" "1" t2)
(insert2! "a" "1" 23 t2)
(lookup2 "a" "1" t2)
(insert2! "a" "2" 43 t2)
t2
(insert2! "b" "3" 21 t2)
t2
(insert2! "a" "1" 1 t2)
t2
```

3. 함수 테이블 만들기

   이제 수업시간에 배웠던 함수테이블을 실제로 구현해 봅시다. 우리는 실습4에서 만들었던 복소수 함수들에 대해 함수 테이블을 만들 것입니다.

```
(define (make-from-real-imag real imag)
  (cons 'rect (cons real imag))
  )
(define (is-rect? complex) (equal? 'rect (car complex)))
(define (real-rect complex) (cadr complex))
(define (imag-rect complex) (cddr complex))
(define (mag-rect complex) (sqrt (+ (expt (cadr complex) 2) (expt (cddr complex) 2))))
(define (angle-rect complex) (atan (cddr complex) (cadr complex)))

(define (make-from-mag-angle mag angle)
  (cons 'polar (cons mag angle))
  )
(define (is-polar? complex) (equal? 'polar (car complex)))
(define (real-polar complex) (* (cadr complex) (cos (cddr complex))))
(define (imag-polar complex) (* (cadr complex) (sin (cddr complex))))
(define (mag-polar complex) (cadr complex))
(define (angle-polar complex) (cddr complex))
```

   복소수 함수 테이블에서 적절한 함수를 찾으려면 복소수가 어떤 종류인지 알아야 합니다. `rep-tag`를 구현해보세요. `rep-tag`는 복소수를 받아서 복소수의 표현방식 심볼을 돌려줍니다.

```
(define (rep-tag complex)
  (cond ((...) 'rectangular)
```

```
        ((...) ...)
        (else (error "It's not implemented resentation")))
  )
```

이제 앞서 만들었던 2차원 테이블을 이용하여 복소수 함수 테이블을 만들어봅시다.

```
(define ftn-table (make-table2 '*complex-operation*))
(insert2! 'rectangular 'real real-rect ftn-table)
(insert2! 'rectangular 'imag imag-rect ftn-table)
...
(insert2! 'polar 'angle angle-polar ftn-table)
ftn-table
```

이제 함수테이블을 활용하여 다양한 복소수 표현을 모두 계산할 수 있는 함수를 만들어봅시다. lookup으로 함수를 찾아와서 사용하면 되겠지요.

```
(define (real complex) ((lookup2 (...) 'real ftn-table) complex))
(define (imag complex) (...)
(define (mag complex) (...))
(define (angle complex) (...))
```

각각의 함수들이 다양한 구현을 받아도 잘 처리하고 있는지 테스트해 봅시다.

```
(define c1 (make-from-real-imag 3 -4))
(define c2 (make-from-mag-angle 5 0.6))
(mag c1)
(angle c1)
(real c2)
(imag c2)
```

이제 우리는 새로운 구현을 추가하고 싶으면 `rep-tag`에 추가해주고 테이블에 `insert2!`만 하면 됩니다. 각각의 함수들이 활용되는 곳에는 손댈 필요가 없습니다. 복소수를 행렬로 표현하는 구현을 추가해 봅시다.

(참고 : $a + b$i의 행렬 표현)

$$\begin{pmatrix} a & -b \\ b & a \end{pmatrix}$$

```
(define (make-from-matrix mat2x2)
  (cons 'matrix mat2x2)
  )
```

```
(define (is-matrix? complex) (equal? 'matrix (car complex)))
(define (real-matrix complex) (cadr complex))
(define (imag-matrix complex) (cadddr complex))
(define (mag-matrix complex)
  (define (determinant mat2x2)
    (sqrt (- (* (car mat2x2) (cadddr mat2x2)) (* (cadr mat2x2) (caddr mat2x2))))
    )
  (determinant (cdr complex))
  )
(define (angle-matrix complex) (atan (cadddr complex) (cadr complex)))

(insert2! ... ... ftn-table)
(insert2! ... ... ftn-table)
(insert2! ... ... ftn-table)
(insert2! ... ... ftn-table)
```

위에 테스트했던 코드를 다시 테스트해봅시다. 변함없이 잘 됩니다. 게다가 이제는
각각의 함수가 행렬로 된 복소수까지도 처리할 수 있게 되었습니다.

```
(define c1 (make-from-real-imag 3 -4))
(define c2 (make-from-mag-angle 5 0.6))
(mag c1)
(angle c1)
(real c2)
(imag c2)

(define c3 (make-from-matrix (list 4 -3 3 4)))
(mag c3)
(real c3)
```