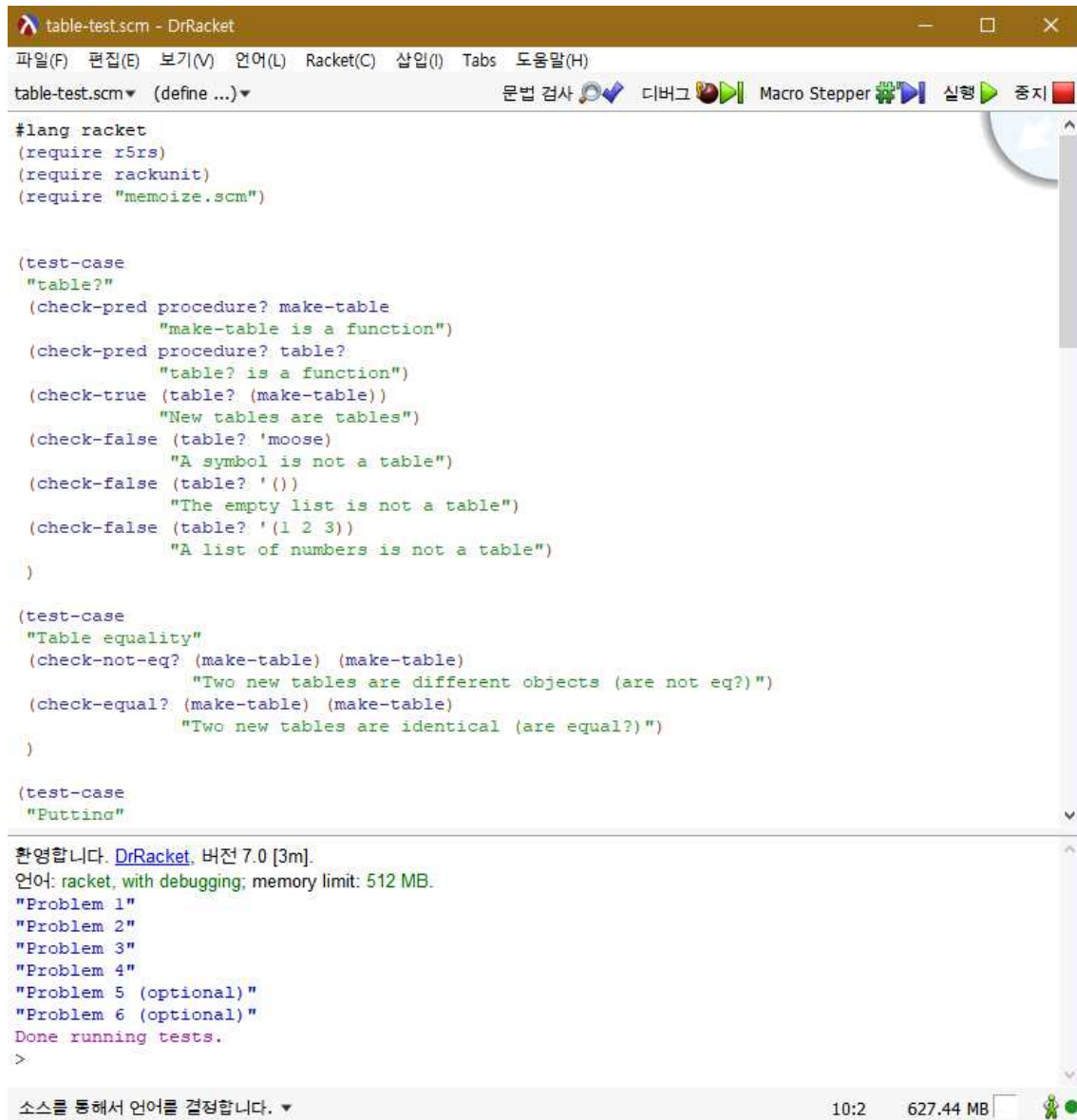


Assignment4 memoize.scm, table-test.scm



```
table-test.scm - DrRacket
파일(F) 편집(E) 보기(V) 언어(L) Racket(C) 삽입(I) Tabs 도움말(H)
table-test.scm (define ...) 문법 검사 디버그 Macro Stepper 실행 중지

#lang racket
(require r5rs)
(require rackunit)
(require "memoize.scm")

(test-case
  "table?"
  (check-pred procedure? make-table
    "make-table is a function")
  (check-pred procedure? table?
    "table? is a function")
  (check-true (table? (make-table))
    "New tables are tables")
  (check-false (table? 'moose)
    "A symbol is not a table")
  (check-false (table? '())
    "The empty list is not a table")
  (check-false (table? '(1 2 3))
    "A list of numbers is not a table")
  )

(test-case
  "Table equality"
  (check-not-eq? (make-table) (make-table)
    "Two new tables are different objects (are not eq?)")
  (check-equal? (make-table) (make-table)
    "Two new tables are identical (are equal?)")
  )

(test-case
  "Putting"

환영합니다. DrRacket, 버전 7.0 [3m].
언어: racket, with debugging; memory limit: 512 MB.
"Problem 1"
"Problem 2"
"Problem 3"
"Problem 4"
"Problem 5 (optional)"
"Problem 6 (optional)"
Done running tests.
>

소스를 통해서 언어를 결정합니다. 10:2 627.44 MB
```

table-test로 실행 시켰을 때 오류 없음.

Problem 1: A table for later

1-1. table과 관련된 함수들의 구현.

```
;find-assoc
(define (find-assoc key alist)
  (cond ((null? alist) #f)
        ((equal? (caar alist) key) (cadar alist))
        (else (find-assoc key (cdr alist)))))

;add-assoc
(define (add-assoc key val alist)
  (cons (list key val) alist))

;make-table
(define (make-table)
  (list 'table))

;table?
(define (table? table)
  (and
    (and (list? table) (not (equal? table '()))) (equal? (car table) 'table)))

;table-put!
(define (table-put! table key val)
  (if (table? table)
      (set-cdr! table (add-assoc key val (cdr table)))
      (error "error")))

;table-has-key?
(define (table-has-key? table key)
  (let ((record (find-assoc key (cdr table))))
    (if record #t #f)))

;table-get
(define (table-get table key)
  (if (table? table)
      (if (find-assoc key (cdr table))
          (find-assoc key (cdr table)) (error "ERROR")) (error "ERROR")))
```

find-assoc : 어떠한 key 값을 입력받아 그 key가 table에 있는지 찾는 함수.

add-assoc : key와 val을 받아 table에 추가하는 함수.

make-table : table 만드는 함수.


table-put! : table과 key, val을 받아 table이 테이블이면 key와val을 추가한다.

table-has-key : table과 key를 받아 table이 테이블이고 key가 있으면 #t를 없으면 #f를 반환

table-get : table과 key를 받아 table이 테이블이고 key가 있으면 key의 val을 반환

1-2. test-case

```
;test-case
(define my-table (make-table))
(table? my-table) ;; => #t
(table-put! my-table 'ben-bitdiddle 'chocolate) ;; => undefined
(table-put! my-table 'alyssa-p-hacker 'cake) ;; => undefined
(table-has-key? my-table 'ben-bitdiddle) ;; => #t
(table-has-key? my-table 'louis-reasoner) ;; => #f
(table-get my-table 'ben-bitdiddle) ;; => chocolate
(table-get my-table 'louis-reasoner) ;; => ERROR
```

환영합니다. DrRacket, 버전 7.0
언어: racket, with debugging; r
"Problem 1"
#t
#t
#f
chocolate
  ERROR
>

Problem 2: lambda-net is monitoring you

2-1. fib 함수가 몇 번 실행되었는지 반환하는 함수 구현.

```
"Problem 2"
;fibonacci
(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1)) (fib (- n 2)))))
;monitord
(define (make-monitored func)
  (let ((count 0))
    (lambda (label)
      (cond ((eq? label 'reset-call-count) (set! count 0))
            ((eq? label 'how-many-calls?) count)
            (else (and (set! count (+ 1 count))
                        (func label)))))))
```

make-monitored : 함수를 처음 실행 시키면 count를 0으로 초기화한다. set!으로 어떤 함수를 이 함수로 바꿔주게 되면 그 함수는 어떠한 인자를 하나 받는 함수가 되고, 받은 값이 어떤 건지 구별 한 후 reset~ 이면 count를 0으로 초기화, how~면 몇 번 실행 됐는지 반환, 그 외에는 받아온 함수를 실행 시키며 count를 1 늘려준다.

2-2. test-case

```
;test-case
(fib 8) ;; => 21
(set! fib (make-monitored fib))
(fib 8) ;; => 21
(fib 'how-many-calls?) ;; => 67
(fib 8) ;; => 21
(fib 'how-many-calls?) ;; => 134
(fib 'reset-call-count)
(fib 'how-many-calls?) ;; => 0
```

환영합니다. [DrRacket](#), 버전 7.0 [3m].
언어: racket, with debugging; memory limit: 512 MB.
"Problem 2"
21
21
67
21
134
0

Problem 3: Back to the table

3-1. max값까지의 피보나치수열의 값을 보여주는 함수.

```
"Problem 3"
;; make-num-calls-table
(define (make-num-calls-table func val)
  (define fib-table (make-table))
  (define (iter count min)
    (func count)
    (table-put! fib-table count (func 'how-many-calls?))
    (func 'reset-call-count)
    (if (equal? count min) (display fib-table)
        (iter (- count 1) min)))
  (iter val 1))
```

make-num-calls-table : 함수 안에 fib-table, iter 라는 함수를 정의 한다. 함수의 body 부분에서 (iter val 1)을 실행 시킨다. iter 함수는 일단 (func count)를 실행 시키는데 func는 fib, count는 val값이다. 그렇게 되면 val 값일 때의 fib값이 나오는데 이 값을 fib-table에 key는 val값, val에는 피보나치 함수가 실행된 숫자를 넣어 준다. 그다음엔 count 값을 다시 0으로 초기화 하게 되고, 만약에 count 값과 1이 같다면 그때의 테이블에 있는 모든 값들을 출력 하게 되고 아니라면 count를 1 만큼 감소시킨 후 iter를 다시 실행 시킨다.

3-2. test-case

```
(make-num-calls-table fib 10)
(make-num-calls-table fib 20)
(make-num-calls-table fib 30)
```

환영합니다. [DrRacket](#), 버전 7.0 [3m].

언어: racket, with debugging; memory limit: 512 MB.

"Problem 3"

(table (1 1) (2 3) (3 5) (4 9) (5 15) (6 25) (7 41) (8 67) (9 109) (10 177))

(table (1 1) (2 3) (3 5) (4 9) (5 15) (6 25) (7 41) (8 67) (9 109) (10 177) (11 287) (12 465) (13 753) (14 1219) (15 1973) (16 3193) (17 5167) (18 8361) (19 13529) (20 21891)) 2

(table (1 1) (2 3) (3 5) (4 9) (5 15) (6 25) (7 41) (8 67) (9 109) (10 177) (11 287) (12 465) (13 753) (14 1219) (15 1973) (16 3193) (17 5167) (18 8361) (19 13529) (20 21891) (21 35421) (22 57313) (23 92735) (24 150049) (25 242785) (26 392835) (27 635621) (28 1028457) (29 1664079) (30 2692537)) 2 2 2

Problem 4: Remembering the past

4-1. fib 함수를 처음의 기능을 하도록 되돌려주는 함수.

```
"Problem 4"
;; memoize
(define (memoize func)
  (define memo-table (make-table))
  (define (fib-put val)
    (table-put! memo-table val (func val)))
  (lambda (val)
    (cond ((table-has-key? memo-table val) (table-get memo-table val))
          (else (and (fib-put val) (func val))))))
```

memoize : 일단 어떤 함수를 set! 으로 memoize를 적용 시키면 그 함수는 프로시저가 되고, 어떠한 값을 받아서 함수를 실행 시킨다. 일단 memo-table을 하나 정의 하고, fib-put이라는 table-put을 사용하는 함수를 하나 정의 해놓는다. 어떤 값을 받아 함수가 실행되면 body 부분에서 cond를 이용, 받은 값이 memo-table 안에 있으면 그때의 값을 리턴 한다 하지만 처음 실행 시키면 아무것도 없으므로, and를 이용 fib-put으로 어떠한 값과 그때의 피보나치 값을 저장 한 후 다시 이 함수를 실행 시킨다. 그다음에 다시 실행되었을 때는 값이 테이블 안에 저장 되어 있으므로 그 값을 출력해주면, 처음 fib함수를 실행 시키는 것과 같은 결과가 나온다.

4-2. test-case

```
(set! fib (memoize fib))
(fib 8)
```

환영합니다. [DrRacket](#), 버전 7.0 [3m].

언어: racket, with debugging; memory limit: 512 MB.

"Problem 4"

21

Problem 5: (Optional) A word of advice

5-1. 어떠한 함수를 실행 시킬 때 함수를 실행시키기 전->
실행시키는 중 -> 실행시킨 후를 눈으로 볼 수 있게 하는 함수.

```
"Problem 5 (optional)"
;; advise
(define (advise func before after)
  (lambda (val)
    (let ((result 0))
      (before)
      (set! result (func val))
      (after)
      result))))
```

advise : 인자로써 함수, 전, 후 값을 받는다. 전 후 값은 주로 display 하는 함수가 오므로 따로 다른 수행을 해 줄 필요는 없으니 함수를 실행시키는 부분만 구현을 하면 된다고 생각했다. 일단 함수는 프로시저이고, 어떠한 값을 하나 받는다. 인자를 받아서 함수를 실행 시키면 result를 0으로 초기화해준다 그리고 before를 실행 시킨 후 result를 (함수, 받은 값) 으로 바꿔준다 이때 함수는 add-1 함수이다. 그 다음 after를 실행 시킨 후 result를 출력 시킨다.

5-2. test-case

```
(define (add-1 x) (+ x 1))
(define advised-add-1
  (advise add-1
    (lambda () (displayln "calling add-1"))
    (lambda () (displayln "add-1 done"))))

(advised-add-1 5)
```

환영합니다. [DrRacket](#), 버전 7.0 [3m].

언어: racket, with debugging; memory limit: 512 MB.

```
"Problem 5 (optional)"
calling add-1
add-1 done
6
"Problem 6 (optional)"
>
```


Problem 6: (Optional) Yep, lambda-net is still monitoring you

6-1. 항상된 monitor 함수.

```
"Problem 6 (optional)"
;; make-monitored-with-advice
(define (make-monitored-with-advice func)
  (let ((count 0)
        (num-steps 0))
    (advise func
      (lambda () (set! count (+ count 1))
                (set! num-steps (+ num-steps 1)))
      (lambda () (set! num-steps (- num-steps 1))
                (if (equal? num-steps 0)
                    (and (and (and (display "Num calls: ") (display count)) (newline))
                        (set! count 0)) void))))))
```

make-monitored-with-advice : 일단 이 함수를 set!을 이용해 어떠한 함수에 적용 하게 되면 5번 문제에 있던 advice 함수를 이용해서 인자로 받은 함수가 몇 번 실행되었는지 와 그때의 값을 출력하는 함수가 된다. 일단 함수를 실행 시키면 count와 num-steps를 0으로 초기화 해준다. 그다음 advice 함수를 실행 시키는데 before 부분에서 count와 num-steps를 1씩 증가 시켜준다. 그렇게 되면 advice 함수 내에서는 재귀적으로 before를 실행 시키고 z를 fib으로 set! 하는 과정에서 다시 advice에 들어가고 또 before를 실행시키고 이게 반복적으로 수행 된다. 그러다 fib 함수의 마지막 단계에 들어가게 되면 after를 실행시키면서 함수들을 빠져나오게 된다. 이때 after에서는 num-steps를 1씩 감소하면서 돌아오게 된다. 그러다 num-steps가 0이 되면 그때의 count값을 출력 하게 되고, 모든 after가 실행이 종료 되면 advice 내부의 z를 출력시키기 때문에 그때의 피보나치 값이 나오게 된다.

6-2. test-case

```
(set! fib (make-monitored-with-advice fib))
(fib 10)
•Num calls: 177
```

환영합니다. DrRacket, 버전 7.0 [3m].
언어: racket, with debugging; memory limit: 512 MB.
"Problem 6 (optional)"
Num calls: 177
55
>