**Report of Structure and Interpretation of Computer Programs**

Chonbuk National University

Department of Computer Science

201514768

임유택

**Problem 1**

**code;**

#lang racket

;; By default, Racket doesn't have set-car! and set-cdr! functions. The

;; following line allows us to use those:

(require r5rs)

(require rackunit)

;; Unfortunately, it does this by making cons return a "mutable pair"

;; instead of a "pair", which means that many built-ins may fail

;; mysteriously because they expect a pair, but get a mutable pair.

;; Re-define a few common functions in terms of car and friends, which

;; the above line make work with mutable pairs.

(define first car)

(define rest cdr)

(define second cadr)

(define third caddr)

(define fourth cadddr)

;; We also tell DrRacket to print mutable pairs using the compact syntax

;; for ordinary pairs.

(print-as-expression #f)

(print-mpair-curly-braces #f)

(define (make-point x y) (cons x y))

(define (hash-a-point point N)

(modulo (+ (car point) (cdr point))

N))

(define (find-assoc key table)

(cond

((null? table) 'ERROR)

((equal? key (caar table)) (cadar table))

(else (find-assoc key (rest table)))))

(define (add-assoc key val alist)

(cons (list key val) alist))

(define table-tag 'hash-table)

(define (make-table size hashfunc)

(let ((buckets (make-vector size null)))

(list table-tag size hashfunc buckets)))

(define (size-of tbl) (cadr tbl))

(define (hashfunc-of tbl) (caddr tbl))

(define (buckets-of tbl) (cadddr tbl))

(define (make-buckets N v) (make-vector N v))

(define bucket-ref vector-ref)

(define bucket-set! vector-set!)

(define (table-get tbl key)

(let ((index

((hashfunc-of tbl) key (size-of tbl))))

(find-assoc key

(bucket-ref (buckets-of tbl) index))))

(define (table-put! tbl key val)

(let ((index

((hashfunc-of tbl) key (size-of tbl)))

(buckets (buckets-of tbl)))

(bucket-set! buckets index

(add-assoc key val

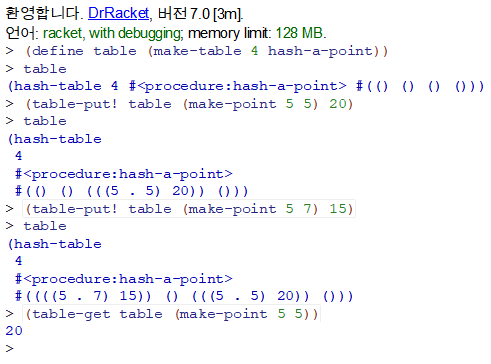
(bucket-ref buckets index)))))

;; Allow this file to be included from elsewhere, and export all defined

;; functions from it.

(provide (all-defined-out))

Problem1



**Problem 2**

**code;**

#lang racket

;; By default, Racket doesn't have set-car! and set-cdr! functions. The

;; following line allows us to use those:

(require r5rs)

(require rackunit)

;; Unfortunately, it does this by making cons return a "mutable pair"

;; instead of a "pair", which means that many built-ins may fail

;; mysteriously because they expect a pair, but get a mutable pair.

;; Re-define a few common functions in terms of car and friends, which

;; the above line make work with mutable pairs.

(define first car)

(define rest cdr)

(define second cadr)

(define third caddr)

(define fourth cadddr)

;; We also tell DrRacket to print mutable pairs using the compact syntax

;; for ordinary pairs.

(print-as-expression #f)

(print-mpair-curly-braces #f)

(define global-table '())

(define (put op type item)

(define (put-helper k array)

(cond ((null? array) (list(list k item)))

((equal? (car (car array)) k) array)

(else (cons (car array) (put-helper k (cdr array))))))

(set! global-table (put-helper (list op type) global-table)))

(define (get op type)

(define (get-helper k array)

(cond ((null? array) #f)

((equal? (car (car array)) k) (cadr (car array)))

(else (get-helper k (cdr array)))))

(get-helper (list op type) global-table))

(define (square x) (\* x x))

(define (attach-type type contents)

(cons type contents))

(define (type datum)

(car datum))

(define (contents datum)

(cdr datum))

(define (rectangular? z)

(eq? (type z) 'rectangular))

(define (polar? z)

(eq? (type z) 'polar))

(define (scheme-number? z)

(eq? (type z) 'scheme-number))

;;유리수

(define (install-rational-package)

(define (numer x) (car x))

(define (denom x) (cdr x))

(define (make-rat x y) (cons x y))

(define (+rational z1 z2) (+rat z1 z2))

(define (-rational z1 z2) (-rat z1 z2))

(define (\*rational z1 z2) (\*rat z1 z2))

(define (/rational z1 z2) (/rat z1 z2))

(define (+rat x y)

(make-rat (+ (\* (numer x) (denom y))

(\* (denom x) (numer y)))

(\* (denom x) (denom y))))

(define (-rat x y)

(make-rat (- (\* (numer x) (denom y))

(\* (denom x) (numer y)))

(\* (denom x) (denom y))))

(define (\*rat x y)

(make-rat (\* (numer x) (numer y))

(\* (denom x) (denom y))))

(define (/rat x y)

(make-rat (\* (numer x) (denom y))

(\* (denom x) (numer y))))

(define (tag x) (attach-type 'rational x))

(put 'ADD '(rational rational)

(lambda (x y) (tag (+rational x y))))

(put 'SUB '(rational rational)

(lambda (x y) (tag (-rational x y))))

(put 'MUL '(rational rational)

(lambda (x y) (tag (\*rational x y))))

(put 'DIV '(rational rational)

(lambda (x y) (tag (/rational x y))))

(put 'make 'rational

(lambda (n d) (tag (make-rat n d))))

'done)

(define (make-rat n d)

((get 'make 'rational) n d))

;;복소수

(define (make-rectangular x y)

(attach-type 'rectangular (cons x y)))

(define (real-part-rectangular z) (car z))

(define (imag-part-rectangular z) (cdr z))

(define (magnitude-rectangular z)

(sqrt (+ (square (car z))

(square (cdr z)))))

(define (angle-rectangular z)

(atan (cdr z) (car z)))

(define (make-polar r a)

(attach-type 'polar (cons r a)))

(define (real-part-polar z)

(\* (car z) (cos (cdr z))))

(define (imag-part-polar z)

(\* (car z) (sin (cdr z))))

(define (magnitude-polar z) (car z))

(define (angle-polar z) (cdr z))

(define (real-part z)

(cond ((rectangular? z)

(real-part-rectangular

(contents z)))

((polar? z)

(real-part-polar

(contents z)))))

(define (imag-part z)

(cond ((rectangular? z)

(imag-part-rectangular

(contents z)))

((polar? z)

(imag-part-polar

(contents z)))))

(define (magnitude z)

(cond ((rectangular? z)

(magnitude-rectangular

(contents z)))

((polar? z)

(magnitude-polar

(contents z)))))

(define (angle z)

(cond ((rectangular? z)

(angle-rectangular

(contents z)))

((polar? z)

(angle-polar

(contents z)))))

(define (install-complex-package)

(define (make-rectangular x y)

(attach-type 'rectangular (cons x y)))

(define (make-polar r a)

(attach-type 'polar (cons r a)))

(define (+complex z1 z2) (make-complex (+c z1 z2)))

(define (-complex z1 z2) (make-complex (-c z1 z2)))

(define (\*complex z1 z2) (make-complex (\*c z1 z2)))

(define (/complex z1 z2) (make-complex (/c z1 z2)))

(define (+c z1 z2)

(make-rectangular (+ (real-part z1) (real-part z2))

(+ (imag-part z1) (imag-part z2))))

(define (-c z1 z2)

(make-rectangular (- (real-part z1) (real-part z2))

(- (imag-part z1) (imag-part z2))))

(define (\*c z1 z2)

(make-polar (\* (magnitude z1) (magnitude z2))

(+ (angle z1) (angle z2))))

(define (/c z1 z2)

(make-polar (/ (magnitude z1) (magnitude z2))

(- (angle z1) (angle z2))))

(define (tag z) (attach-type 'complex z))

(put 'ADD '(complex complex)

(lambda (z1 z2) (tag (+complex z1 z2))))

(put 'SUB '(complex complex)

(lambda (z1 z2) (tag (-complex z1 z2))))

(put 'MUL '(complex complex)

(lambda (z1 z2) (tag (\*complex z1 z2))))

(put 'DIV '(complex complex)

(lambda (z1 z2) (tag (/complex z1 z2))))

(put 'make-complex-rectangular 'complex

(lambda (x y) (tag (make-rectangular x y))))

(put 'make-complex-polar 'complex

(lambda (r a) (tag (make-polar r a))))

'done)

(define (make-complex z) (attach-type 'complex z))

(define (make-complex-rectangular x y)

((get 'make-complex-rectangular 'complex) x y))

(define (make-complex-polar r a)

((get 'make-complex-polar 'complex) r a))

;;기본수

(define (install-scheme-number-package)

(define (tag x)

(attach-type 'scheme-number x))

(put 'ADD '(scheme-number scheme-number)

(lambda (x y) (tag (+ x y))))

(put 'SUB '(scheme-number scheme-number)

(lambda (x y) (tag (- x y))))

(put 'MUL '(scheme-number scheme-number)

(lambda (x y) (tag (\* x y))))

(put 'DIV '(scheme-number scheme-number)

(lambda (x y) (tag (/ x y))))

(put 'make 'scheme-number

(lambda (x) (tag x)))

'done)

(define (make-scheme-number n)

((get 'make 'scheme-number) n))

(define (ADD x y) (apply-generic 'ADD x y))

(define (SUB x y) (apply-generic 'SUB x y))

(define (MUL x y) (apply-generic 'MUL x y))

(define (DIV x y) (apply-generic 'DIV x y))

(define (apply-generic op . args)

(let ((type-tags (map type args)))

(let ((proc (get op type-tags)))

(if proc

(apply proc (map contents args))

(error

"No method for these types — APPLY-GENERIC"

(list op type-tags))))))

(install-rational-package)

(install-complex-package)

(install-scheme-number-package)

(define c1 (make-complex-rectangular 3 5))

(define c2 (make-complex-rectangular 1 2))

(define c3 (make-complex-polar 3 5))

(define c4 (make-complex-polar 1 2))

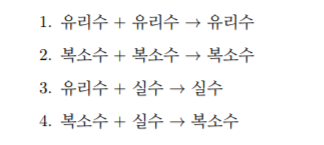
(define r1 (make-rat 3 5))

(define r2 (make-rat 5 6))

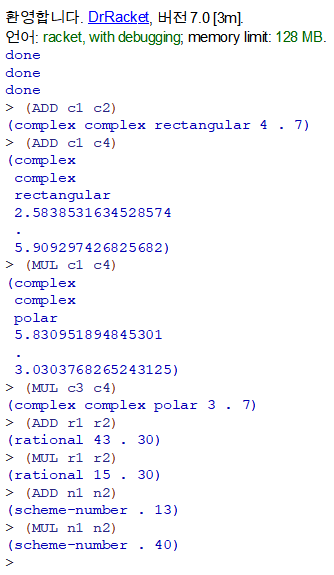
(define n1 (make-scheme-number 5))

(define n2 (make-scheme-number 8))

Problem2



1, 2의 조건에 따른 test



3, 4의 조건에 따른 test ({유리수, 실수}, {복소수, 실수} 연산 ,{복소수, 유리수} 예외처리)

