**Structures and Interpretation of Computer Program**

**Exercise Chapter 2.5 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 2.5.1 Generic Arithmetic Operation**

2. Why this works?

* The complex package has no interface to polar or rectangular magnitude function.
* As such, Louis needs to interface the function into the complex package.
* It is as simple as using the ‘put function with ‘complex tag.
* Calling magnitude using complex tag will return generic procedure that will call specific representation procedure like polar magnitude.

1. Trace all procedure in evaluating (magnitude z)

(magnitude z)

(apply-generic ‘magnitude z)

(let (type-tags (map type-tag z))) -> (type-tags 'complex)

(let (proc (get op type-tags))) -> (get 'magnitude 'complex) -> (proc (magnitude))

if proc

(apply magnitude (map contents z))

(magnitude (contents z))

(apply-generic 'magnitude (contents z)) ;

(let (type-tags (map type-tag z))) -> (type-tags 'polar)

(let (proc (get op type-tags))) -> (get 'magnitude 'polar) -> (define (magnitude z) (car z)) -> (proc (magnitude z))

if proc

(apply (magnitude z) (map contents z))

(magnitude (contents z))

1. How many times apply generic is invoked?

* Two times, one to apply generic package, the other one is to apply specific package, like polar or rectangular

1. What procedure is dispatched in each case?

* (magnitude z) is returned in every call to apply-generic. However, the internal definition of (magnitude z) is different in each case.
* For example, the first call to apply-generic returns (magnitude z) from complex package. The second call returns (magnitude z) from polar package.



(define (attach-tag type-tag contents)

(if (number? contents)

contents

(cons type-tag contents)))

(define (type-tag datum)

(cond ((number? datum) 'scheme-number)

((pair? datum) (car datum))

(else (error "Bad tagged datum -- TYPE-TAG" datum))))

(define (contents datum)

(cond ((number? datum) datum)

((pair? datum) (cdr datum))

(else (error "Bad tagged datum -- CONTENTS" datum))))



(define (install-scheme-number-pkg)

(put 'equ '(scheme-number scheme-number) =))

(define (install-rational-pkg)

(define (equ? x y)

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

(put 'equ? '(rational rational) equ?))

(define (install-complex-pkg)

(define (equ? x y)

(and (= (real-part x) (real-part y)) (= (imag-part x) (imag-part y))))

(put 'equ? '(complex complex equ?)))

(define (equ? x y) (apply-generic 'equ? x y))

* Install each equ? Implementation in each package



(define (install-scheme-number-pkg)

(define (=zero? x)

(= 0 x))

(put '=zero? '(scheme-number scheme-number) =zero?)

'done)

(define (install-rational-pkg)

(define (=zero? x)

(= 0 (numer x)))

(put '=zero? '(rational rational) =zero?)

'done)

(define (install-complex-pkg)

(define (=zero? x)

(and (= 0 (real-part x)) (= 0 (imag-part x))))

(put '=zero? '(complex complex) =zero?)

'done)

(define (=zero? x) (apply-generic '=zero? x))