アルゴリズムとデータ構造入門 第九回課題

1029-24-9540 山崎啓太郎

December 25, 2012

1 複素数システム

```
(define (square z) (* z z))
(define (real-part z) (car z))
(define (imag-part z) (cdr z))
(define (magnitude z)
  (sqrt (+ (square (real-part z))
           (square (imag-part z)) )))
(define (angle z)
  (atan (imag-part z) (real-part z)))

(define (make-from-real-imag x y)
  (cond
    ((and (number? x) (number? y))  (cons x y))
    ((number? x) (make-from-real-imag (- x (imag-part y)) (real-part y)))
    ((number? y) (make-from-real-imag (real-part x) (+ (imag-part x) y)))
    (else (make-from-real-imag (- (real-part x) (imag-part y)) (+ (imag-par

(define (add-complex z1 z2)
  (make-from-real-imag
    (+ (real-part z1) (real-part z2))
    (+ (imag-part z1) (imag-part z2)) ))
(define (sub-complex z1 z2)
  (make-from-real-imag
    (- (real-part z1) (real-part z2))
    (- (imag-part z1) (imag-part z2)) ))
(define (mul-complex z1 z2)
  (make-from-real-imag
    (- (* (real-part z1) (real-part z2)) (* (imag-part z1) (imag-part z2)))
    (+ (* (imag-part z1) (real-part z2)) (* (real-part z1) (imag-part z2)))
(define (div-complex z1 z2)
  (make-from-real-imag
    (/ (+ (* (real-part z1) (real-part z2)) (* (imag-part z1) (imag-part z2
    (/ (- (* (imag-part z1) (real-part z2)) (* (real-part z1) (imag-part z2

(define stringify-complex (lambda (z)
  (cond
    ((= (imag-part z) 0) (number->string (real-part z)))
    ((= (real-part z) 0)
      (cond
        ((= (imag-part z) 1) ”i”)
        ((= (imag-part z) -1) ”-i”)
        (else (string-append (number->string (imag-part z)) ”i”)) ))
    ((> (imag-part z) 0)
      (if (= (imag-part z) 1)
        (string-append (number->string (real-part z)) ”+” ”i”)
        (string-append (number->string (real-part z)) ”+” (number->string (
    (else
      (if (= (imag-part z) -1)
        (string-append (number->string (real-part z)) ”-” ”i”)
        (string-append (number->string (real-part z)) (number->string (imag
```

2 実行例

(stringify-complex (make-from-real-imag 3 4)) => ”3+4i”
(stringify-complex (make-from-real-imag 3 0)) => ”3”
(stringify-complex (make-from-real-imag 0 3)) => ”3i”
(stringify-complex (make-from-real-imag 0 1)) => ”i”
(stringify-complex (make-from-real-imag 0 -1)) => ”-i”
(stringify-complex (make-from-real-imag 3 1)) => ”3+i”
(stringify-complex (make-from-real-imag 3 -2)) => ”3-2i”
(stringify-complex (make-from-real-imag 3 -1)) => ”3-i”
(stringify-complex (add-complex (make-from-real-imag 3 -1) (make-from-real-imag 2 4))) => ”5+3i”
(stringify-complex (sub-complex (make-from-real-imag 3 -1) (make-from-real-imag 2 4))) => ”1-5i”
(stringify-complex (mul-complex (make-from-real-imag 3 -1) (make-from-real-imag 2 4))) => ”10+10i”
(stringify-complex (div-complex (make-from-real-imag 3 -1) (make-from-real-imag 2 4))) => ”0.1-0.7i”
(stringify-complex (make-from-real-imag (make-from-real-imag 3 -1) (make-from-real-imag 2 4))) => ”-1+i”

3　説明

make-from-real-imag 関数を使って、x+iy となる形の複素数を生成している。
add-complex、sub-complex、mul-complex、div-complex 関数にて四則演算が可能である。
また、make-from-real-imag 関数には複素数を渡すこともでき、その際には返り値が完結になるよう計算される。
簡略化は、make-from-real-imag 関数で生成された複素数を文字列化する stringify-complex 関数にて行なっている。
例として、簡略化されないと”0+1i”と表示されるものが、”i”と表示されるような処理がされている。（その他の例は実行例に有）