Programare funcţională – Laboratorul 6
Iterativitate

Isabela Drămnesc

April 2, 2014

1 Concepte

- iterativitate
- when, unless, for
- do - simulare
- definite functii iterative

2 Întrebări din Laboratorul 5

- Scrieţi varianta final recursivă a algoritmului de sortare prin inserţie.

3 Exerciţii

3.1 when; unless

(when test-expr body ...+)
Evalueaza test-expr. Dacă se evaluează la false, atunci rezultatul este ¡void¿.
Altfel, bodys sunt evaluate, şi va fi afişat rezultatul ultimei evaluări.

```
(when (even? 5) 3) ; prints nothing

(when (odd? 5) 3 4 8 9) ; prints the result of the last evaluation

(define (f x)
(when (odd? x) (print "la la "))
  )

> (f 11)

> (f 10)
```

(unless test-expr body ...+)
echivalent cu:
(when (not test-expr) body ...+)

```
> (unless (negative? -3) (print 8) (print 90))

> (negative? -9)

> (unless (negative? 3) (print 8) (print 90))

> (unless (negative? 3) 8 (print 90))
```

3.2 for

(for (for-clause ...) body-or-break ... body)

```
(for ([i '(1 2 3)]
      [j "abc"]
      #:when (odd? i)
      [k #(#t #f)])
  (display (list i j k)))
(printf "\n")

(for ([i '(1 2 3)]
      [j '(a b c)]
      )
  (display (list i j)))

(for/vector ([i '(1 2 3)]) (number->string i))

(for/vector #:length 2 ([i '(1 2 3 5)]) (number->string i))

(for/vector #:length 5 ([i '(1 2 3 5)]) (number->string i))

(for/vector #:length 5 #:fill "lala" ([i '(1 2 3 5)]) (number->string i))
```

3.3 do - o formă specială de iteraţie

```
(do ((var-1 init-1 stepper-1)
     (var-2 init-2 stepper-2)
                ...

     (var-n init-n stepper-n))
    (end-test
      end-form-1
        ...

      end-form-k
      return-value)
      body-1
        ...
      (return value)      ;optional
        ...
```

body-m)

```
;Simulati comportamentul interpretorului Racket pentru
;urmatoarele programe:

;Prin urmatoarele exemple veti intelege exact
; comportamentul lui do


1)
(printf "--- 1) ----- \n")
(do ((v1 1 (+ 1 v1))
     (v2 '() ( cons 'a v2))
     )
    ((> v1 5 ) v2 ( print 'end ) )
( print v1 )
)


2)
(printf "---- 2) ------- \n")
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     )
    ((> v1 3) (when (> v1 4) v1) v2)
  (print v1)
 )


3)
(printf "---- 3) ------- \n")
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     )
    ((> v1 3) (when (> v1 3) v2))
  (print v1) ; with print v1 prints 123'(a a a)
  )          ; what does it print without (print v1)?


4)
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2)))
    ((> v1 5) v1)
     (when (> v1 4 ) v2)
     ( print v1)
  )

5)
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
```

```
     (v3 9) ; no stepper -- remains unchanged
     )
  ((> v1 5) v2)
  '(car v2) ; the returned value is ignored
  (print v1) ;side effect
  )


6)
(do ((v1 1 (if (< v1 3) (+ 1 v1) (exit 'done)))
     (v2 '() (cons 'a v2))
     )
    ((> v1 5) v1)
    (print v1)
  )


7)
(do ((v1 1 (if #f 1 (exit 'done)))
     (v2 '() (cons 'a v2))
     )
    ((> v1 5) v1)
    (print v1)
     )



8)
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     (v3 9) ; no stepper
     )
    ((> v1 5) (print v3) ; side effect
              v2) ; the returned value
   (print v1) ; side effect
  )


9)
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     (v3 9) ; no stepper
     )
    ((> v1 5)) ; no return value
   (print v1) ; side effect
  )


10)
(do ((v1 1 (+ 1 v1))
```

```
     (v2 '() (cons 'a v2))
     (v3 9) ; no stepper
     )
    ((print v3)) ;funny end condition
                 ; no return value
   (print v1) ; side effect
  )


11)
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     (v3 9) ; no stepper
     )
    ((print v3) ;funny end condition
      33)    ; the returned value
   (print v1) ; side effect
  )


12)
(printf "---12---␣\n")
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     (v3 9) ; no stepper
     )
    ((print #f) ;funny end condition
      )    ; no return value
   (print v1) ; side effect
  )

13)
(do ((v1 1 (+ 1 v1))
     (v2 '() (cons 'a v2))
     (v3 9) ; no stepper
     )
    (#f ; what does this mean?
      )    ; no return value
   (print v1) ; side effect
  )
```

3.4 Factorial - iterativ

```
 ; ce trebuie sa schimbam in exemplul urmator
 ; in asa fel incat sa obtinem rezultatul corect?

(define (fact-it n)
  (let ([i 1] [f 1])
  (set! i 1)
  (for ([i (in-range n)] )
```

```
    (set! f (* f i))
    )
  (printf "the factorial of ~a is ~a" n f))
 )

> (fact-it 5)

(printf "\n")


; factorial utilizand do

(define (fact-it-do n)
  (do ((i 1 (+ i 1))
       (f 1 (* f i)))
      ((> i n) f))
  )

> (fact-it-do 5)
```

3.5 Definiţi o funcţie iterativă pentru calculul lungimii unei liste.

3.6 Definiţi o funcţie iterativă pentru calculul x la puterea y

4 Temă.

4.1 Simulati comportamentul interpretorului Racket pentru următoarele programe:

```
1)
(do ((x 0 (+ 1 x))
     (y 0 (+ x y))
     (z 0 (+ y z)))
    ((>= x 10)
     (list x y z))
    (print (list x y z)))

2)
(do ((v1  1 (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 5)
     v2)
    (print v1))

3)
(do ((v1 1 (print (+ 1 v1)))
     (v2 (print ()) (cons 'a v2)))
    ((> v1 0)
```

```
     v2)
    (print (list v1 v2)))


4)
(do ((v1 1         (+ 1 v1))
     (v2 'nothing (cons 'a v2)))
    ((> v1 0)
     v2)
    (print v1))


5)
(do ((v1  1 (+ 1 v1))
     (v2 () ))
    ((> v1 3)
     v2)
    (set v2 (cons 'a v2)))



6)
(do ((v1  1 (+ 1 v1))
     (v2 () (cons 'a v2)))
    ((> v1 3)
     v2)
    (set v2 (cons 'b v2)))


7)
(do ((v1   1  (+ 1 v1))
     (v2 '(b) (cons 'a v2))
     (v3   1  (list v1 v2 v3)))
    ((> v1 3)
     v2)
    (print (list v1 v2 v3)))


8)
(do ((v1 1 (+ 1 v1))
     (v2 () (cons 'a v2))
     (v3 9))      ; no stepper: value
                          ; remains unchanged --> should be in a LET
    ((print v1)  ; end cond
     )                ; no return value
    (print v3))
```

4.2 Definiţi o funcţie pentru aflarea sumei pătratelor numerelor unei liste (ignorând simbolurile), inversei unei liste, cmmdc a două numere în 3 moduri:

1. program recursiv
2. program final recursiv
3. program iterativ

Notă: Termen de realizare: laboratorul următor.