Functional Programming – Laboratory 6
Iterativity

Isabela Drămnesc

April 2, 2014

1 Concepts

- iterativity
- when, unless, for
- do - simulating the behavior
- defining iterative functions

2 Questions from lab 5

- Write the definition of the insertion sort algorithm in the tail recursive version.

3 Exercises

3.1 when; unless

(when test-expr body ...+)

Evaluates test-expr. If the result is false, then the result of the when expression is void. Otherwise, the bodys are evaluated, and the last body is in tail position with respect to the when form.

```
(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 ...+)
equivalent to:
(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 '(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 - a special iteration form

```
(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)

;Simulate the behavior of Racket for the
;following programs:
```

```
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 - iterative version

```
 ; what do we have to change in the following example
 ; such that we will get the correct result?

(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 using do

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

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

3.5 Define an iterative function which returns the length of a list.

3.6 Define an iterative function for x to the power of y.

4 Homework.

4.1 Simulate the behavior of the Racket interpreter for the following:

```
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 Define functions for: the squared sum of the numbers of a list (by ignoring the symbols), the reverse of a list, GCD in 3 ways:

1. the recursive version
2. the tail recursive version
3. the iterative version

Deadline: next lab.