Corso di Programmazione

I Prova di accertamento del 3 Febbraio 2017

cognome e nome

Risolvi i seguenti esercizi, riporta le soluzioni in modo chiaro negli appositi spazi e giustifica sinteticamente le risposte. Dovrai poi consegnare queste schede con le soluzioni, avendo cura di scrivere il tuo nome nell'intestazione e su ciascun eventuale foglio aggiuntivo.

1. Programmi in Scheme

```
(define f          ; intero
  (lambda (s)      ; lista di interi positivi
    (g s 0)
    ))
```

```
(define g
  (lambda (s b)
    (if (null? s)
        0
        (if (<= (car s) b)
            (g (cdr s) b)
            (max (g (cdr s) b)
                 (+ (g (cdr s) (car s)) 1))
            ))))
```

In relazione al programma riportato sopra, determina il risultato della valutazione delle seguenti espressioni:

`(f '(5))`	→	1	`(f '(5 3))`	→	1
`(f '(1 2))`	→	2	`(f '(5 3 1))`	→	1
`(f '(1 3 5))`	→	3	`(f '(1 5 3))`	→	2
`(f '(1 4 16))`	→	3	`(f '(4 2 3 1 5))`	→	3

2. Programmi ricorsivi in Scheme

Il seguente programma basato sulla procedura `mh` calcola il numero di percorsi di Manhattan che non cambiano direzione in due incroci consecutivi, collegati da un tratto unitario di strada. In altri termini, dopo aver cambiato direzione in corrispondenza a un incrocio, occorre attraversare almeno un successivo incrocio procedendo in linea retta prima di cambiare nuovamente direzione. Per esempio, solo 4 dei 6 percorsi di Manhattan relativi a una griglia 2×2 oppure 11 dei 35 percorsi nel caso 3×4 soddisfano l'ulteriore vincolo descritto sopra:

`(mh 2 2)` → 4 `(mh 3 4)` → 11

Completa la definizione del programma nel riquadro.

```
(define mh                      ; val: intero
  (lambda (i j)                 ; i, j: interi non negativi
    (if (or (= i 0) (= j 0))
        1
        (+ (md (- i 1) j) (mr i (- j 1)))
        )))

(define md                      ; spostamento precedente "in giù"
  (lambda (i j)
    (if (or (= i 0) (< j 2))
        1
        (+ (md (- i 1) j) (mr i (- j 2)))
        )))

(define mr                      ; spostamento precedente "a destra"
  (lambda (i j)
    (if (or (< i 2) (= j 0))
        1
        (+ (md (- i 2) j) (mr i (- j 1)))
        )))
```

3. Programmazione in Scheme

Con il termine *identificatore* si intende uno dei simboli utilizzati in un programma per rappresentare i nomi di costanti, variabili, procedure, ecc. In molti linguaggi di programmazione un identificatore deve essere costituito da una lettera iniziale (maiuscola o minuscola), seguita da zero o più caratteri alfanumerici (lettere e cifre, comunque mescolate, normalmente incluso anche il carattere *underscore* "_").

Definisci un programma in Scheme basato sulla procedura `is-identifier?` per verificare se una stringa rappresenta un identificatore nel senso appena precisato. Esempi:

```
(is-identifier? "")  →  false          (is-identifier? "Q")  →  true
(is-identifier? "5")  →  false         (is-identifier? "Iter_2")  →  true
(is-identifier? "3o")  →  false        (is-identifier? "latoBase")  →  true
(is-identifier? "_ape")  →  false      (is-identifier? "w3c")  →  true
```

```
(define is-identifier?
  (lambda (sym)
    (cond ((string=? sym "") false)
          ((not (letter? (string-ref sym 0))) false)
          (else (idf-chars? (substring sym 1)))
          )))

(define idf-chars?
  (lambda (chs)
    (cond ((string=? chs "") true)
          ((idf-ok? (string-ref chs 0)) (idf-chars? (substring chs 1)))
          (else false)
          )))

(define letter?
  (lambda (ch)
    (or (and (char<=? #\A ch) (char<=? ch #\Z))
        (and (char<=? #\a ch) (char<=? ch #\z))
        )))

(define idf-ok?
  (lambda (ch)
    (or (char=? ch #\_)
        (letter? ch)
        (and (char<=? #\0 ch) (char<=? ch #\9))
        )))
```

4. Verifica formale della correttezza

In relazione alla procedura g definita nell'esercizio **1** si può dimostrare per induzione sul parametro k che

$$(g\ (n{+}1\ n{+}2\ \ldots\ n{+}k)\ q) \quad \rightarrow \quad k$$

per qualsiasi terna di interi n, k, q tali che $k > 0$ e $n \geq q \geq 0$. (Il primo argomento denota una lista di interi.)
Completa l'impostazione della dimostrazione e dimostra la proprietà formulata attenendoti allo schema seguente:

- Formalizza la proprietà che esprime il caso / i casi base:

  $\forall\ n, q \in \mathbf{N}$ tali che $n \geq q$. $(g\ (n{+}1)\ q) \rightarrow 1$

- L'ipotesi induttiva è la seguente: preso come riferimento $h \in \mathbf{N}^+$

  si assume che $\forall\ n, q \in \mathbf{N}$ tali che $n \geq q$. $(g\ (n{+}1\ n{+}2\ \ldots\ n{+}h)\ q) \rightarrow h$

- Formalizza la proprietà da dimostrare come passo induttivo: per h scelto sopra

  $\forall\ n, q \in \mathbf{N}$ tali che $n \geq q$. $(g\ (n{+}1\ n{+}2\ \ldots\ n{+}h{+}1)\ q) \rightarrow h{+}1$

- Dimostra il caso / i casi base: ($\forall\ n, q \in \mathbf{N}$ tali che $n \geq q$)

  $(g\ (n{+}1)\ q) \rightarrow (\text{if } (\text{null? } (n{+}1))\ 0\ \ldots\ ) \rightarrow (\text{if } \mathit{false}\ 0\ \ldots\ )$

  $\rightarrow (\text{if } (<=\ n{+}1\ q)\ \ldots\ \ldots\ ) \rightarrow (\text{if } \mathit{false}\ \ldots\ \ldots\ )$; $n \geq q$

  $\rightarrow (\max\ (g\ ()\ q)\ (+\ (g\ (\text{cdr } (n{+}1))\ (\text{car } (n{+}1)))\ 1))$

  $\rightarrow (\max\ 0\ (+\ (g\ ()\ n{+}1)\ 1)) \rightarrow (\max\ 0\ (+\ 0\ 1))$

  $\rightarrow 1$

- Dimostra il passo induttivo: ($\forall\ n, q \in \mathbf{N}$ tali che $n \geq q$, h scelto sopra)

  $(g\ (n{+}1\ n{+}2\ \ldots\ n{+}h{+}1)\ q)$

  $\rightarrow (\text{if } (\text{null? } (n{+}1\ \ldots\ n{+}h{+}1))\ 0\ \ldots\ ) \rightarrow \ldots$; *come sopra*

  $\rightarrow (\max\ (g\ (n{+}2\ \ldots\ n{+}1{+}h)\ q)\ (+\ (g\ \ldots\ \ldots)\ 1))$; *poiché* $n{+}1, q \in \mathbf{N}$ e $n{+}1 > q$

  $\rightarrow (\max\ h\ (+\ (g\ \ldots\ \ldots)\ 1))$; *si applica l'ipotesi induttiva*

  $\rightarrow (\max\ h\ (+\ (g\ (n{+}2\ \ldots\ n{+}1{+}h)\ n{+}1)\ 1))$; $n{+}1, n{+}1 \in \mathbf{N}$ e $n{+}1 \geq n{+}1$

  $\rightarrow (\max\ h\ (+\ h\ 1))$; *per l'ipotesi induttiva*

  $\rightarrow h{+}1$

5. Ricorsione di coda

Scrivi un programma basato sulla *ricorsione di coda* per calcolare il valore numerico di una rappresentazione ternaria bilanciata (*BTR*: notazione posizionale a tre cifre "`-`", "`.`", "`+`" di valore rispettivamente –1, 0 e 1). In altri termini, scrivi una procedura funzionalmente equivalente a `btr-val` ma che utilizza esclusivamente questa forma di ricorsione.

```
(define btr-val-tr
  (lambda (r)
    (btr-val-rec r 0 (string-length r) 0)
    ))


(define btr-val-rec
  (lambda (r i n v)
    (if (< i n)
        (btr-val-rec r (+ i 1) n (+ (* 3 v) (btd-val (string-ref r i))))
        v
        )))


(define btd-val
  (lambda (btd)
    (cond ((char=? btd #\-) -1)
          ((char=? btd #\.)  0)
          ((char=? btd #\+) +1)
          )
    ))
```

6. Astrazione funzionale (ai fini della valutazione, questo quesito ha un peso molto minore degli altri)

Riesci a intuire quale problema risolve o quale funzione calcola la procedura `f` definita nell'esercizio **1**?
Spiega in poche parole le tue intuizioni e le ragioni (il perché) alla base di esse.

La procedura `f` calcola la *sottosequenza crescente più lunga* della lista passata come argomento.

La ricorsione ad albero corrisponde alle opzioni di includere o meno un certo elemento della lista.