Corso di Programmazione

I Prova di accertamento del 21 Gennaio 2019 / A

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nelle intestazioni e su ciascun ulteriore foglio che intendi consegnare.

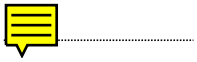
1. Programmi in Scheme

```
(define f          ; val: intero
  (lambda (x y)    ; x ≥ 0,  y > 0  interi
    (if (< x y)
        1
        (+ (f (- x 1) y) (f (- x y) y))
        )))
```

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

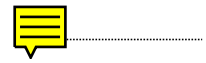
`(f 4 5)` → 

`(f 5 5)` → 

`(f 6 5)` → 

`(f 3 1)` → 

`(f 5 1)` → 

`(f 4 2)` → 

`(f 6 2)` → 

`(f 9 4)` → 

2. Programmazione in Scheme

Una stringa *s* è una *palindrome* se procedendo da sinistra a destra oppure da destra a sinistra si legge la stessa sequenza di caratteri, come nel caso di `"esose"`. Scrivi un programma in Scheme per realizzare la procedura a valori interi `palindrome-lev` per determinare il "livello di palindromicità" di una data stringa, cioè il numero di coppie di caratteri uguali in posizioni simmetriche rispetto al centro della stringa (quindi alla stessa distanza rispetto agli estremi), contando anche l'eventuale carattere autosimmetrico che si trovi esattamente al centro della stringa. Esempi:

```
(palindrome-lev "")       → 0        (palindrome-lev "esose")             → 3
(palindrome-lev "a")      → 1        (palindrome-lev "erodere")           → 3
(palindrome-lev "nono")   → 0        (palindrome-lev "ilredevevederli")   → 8
```

3. Ricorsione ad albero

Il programma impostato nel riquadro applica la logica risolutiva del problema della *sottosequenza comune più lunga* (LCS) per confrontare una stringa di riferimento `s` con una diversa versione `t`, dove si assume che sia `s` che `t` siano costituite esclusivamente da lettere maiuscole o minuscole, non da altri tipi di carattere. In particolare, la procedura `xlcs` restituisce una stringa così composta: le lettere di `s` che trovano corrispondenza in `t` sono sostituite da un asterisco `*`; quelle da "cancellare" nella sottosequenza comue più lunga sono rimpiazzate da una barra inclinata `/`; inoltre, le lettere di `t` da "aggiungere" (cioè senza corrispondenza) rispetto alla stringa di riferimento `s` vengono incluse nelle posizioni corrispondenti. Esempi:

```
(xlcs "" "")        →  ""          (xlcs "arto" "atrio")      →  "*/*ri*"
(xlcs "" "ma")      →  "ma"        (xlcs "atrio" "arto")      →  "*/*/t*"
(xlcs "ma" "")      →  "//"        (xlcs "flora" "lira")      →  "/*/i**"
(xlcs "ma" "ma")    →  "**"        (xlcs "cincia" "piani")    →  "/p*a*/*/"
```

Completa il programma inserendo espressioni appropriate negli spazi indicati.

```
(define xlcs                          ; val:  stringa
  (lambda (s t)                       ; s, t:  stringhe
    (cond ((string=? s "") t)

          ( ......................................................................

           (string-append .................................. (xlcs (substring s 1) t)))
          ((char=? (string-ref s 0) (string-ref t 0))
           (string-append "*" (xlcs (substring s 1) (substring t 1))))
          (else

           (better (string-append .............................................. )

                   (string-append .............................................. )
                   ))
          )))


(define better
  (lambda (u v)
    (if (< (stars u) (stars v))

        ........................................................................

        ........................................................................
        )))


(define stars
  (lambda (q)
    (if (string=? q "")
        0
        (let ((n (stars (substring q 1))))

          (if (char=? ............................................. ) (+ n 1) n)
          ))))
```

4. Verifica formale della correttezza

In relazione alla procedura `f` dell'esercizio **1**, si può dimostrare che per ogni intero $n \geq 0$:

$$(\texttt{f}\ n\ 1) \quad \rightarrow \quad 2^n$$

Dimostra questa proprietà per induzione sui valori di n attenendoti allo schema delineato qui sotto.

- Formalizza la proprietà generale da dimostrare:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il caso / i casi base:

- Dimostra il passo induttivo:

5. Procedure con argomenti procedurali

Il programma impostato nel riquadro risolve il problema dei percorsi di Manhattan e restituisce una lista di interi, contenente tanti elementi quanti sono i percorsi diversi. Più specificamente, ciascun percorso è rappresentato nella lista dal numero di cambi di direzione — o *svolte* — che lo caratterizzano. Nel caso di una "mappa" reticolare 2 x 2, per esempio, sono possibili 6 percorsi diversi, che a due a due richiedono rispettivamente 1, 2 o 3 svolte; se punto di partenza e destinazione si trovano sulla stessa "strada", allora c'è un unico percorso senza svolte; se un lato del rettangolo ha lunghezza unitaria, sono possibili due percorsi con una sola svolta lungo il perimetro e altri percorsi con due svolte con attraversamenti all'interno del rettangolo:

`(mh 2 2)` → `(1 3 2 2 3 1)` `(mh 0 5)` → `(0)` `(mh 4 1)` → `(1 2 2 2 1)`

Completa il programma inserendo espressioni appropriate negli spazi indicati.

```
(define mh                      ; val:  lista di interi
  (lambda (i j)                 ; i, j:  interi non negativi
    (if (or (= i 0) (= j 0))
        (list 0)

        (append (md (- i 1) j)  ..........  )
        )))

(define md                      ; md:  passo precedente in giù
  (lambda (i j)
    (cond ((and (= i 0) (= j 0))  (list 0))
          ((= i 0)  (list 1))
          ((= j 0)  (list 0))
          (else
           (append (md (- i 1) j)

                   (map  ..........  (mr i (- j 1)))
                   ))
          )))

(define mr                      ; md:  passo precedente a destra
  (lambda (i j)
    (cond ((and (= i 0) (= j 0))  (list 0))
          ((= i 0)  (list 0))
          ((= j 0)  (list 1))
          (else

           (append  ..........

                    ..........
                   ))
          )))
```

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

Riesci a intuire quale problema potrebbe risolvere, 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.

Corso di Programmazione

I Prova di accertamento del 21 Gennaio 2019 / B

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nelle intestazioni e su ciascun ulteriore foglio che intendi consegnare.

1. Programmi in Scheme

```
(define f          ; val: intero
  (lambda (x y)    ; x, y: interi non negativi
    (if (<= x y)
        1
        (+ (f (- x (+ y 1)) y) (f (- x 1) y))
        )))
```

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

`(f 4 4)` → .....

`(f 6 4)` → .....

`(f 5 1)` → .....

`(f 7 1)` → .....

`(f 5 4)` → .....

`(f 4 0)` → .....

`(f 6 0)` → .....

`(f 7 2)` → .....

2. Programmazione in Scheme

Una stringa *s* è una *palindrome* se è perfettamente simmetrica rispetto al centro, cioè se procedendo da sinistra a destra oppure da destra a sinistra si legge la stessa sequenza di caratteri, come nel caso di `"ilredevevederli"` e `"acetonellenoteca"`. Scrivi un programma in Scheme per realizzare il predicato `palindrome?` (procedura a valori booleani) che consenta di verificare se una data stringa è una palindrome. Esempi:

```
(palindrome? "")       → true          (palindrome? "erodere")            → false
(palindrome? "a")      → true          (palindrome? "ilredevevederli")    → true
(palindrome? "nono")   → false         (palindrome? "acetonellenoteca")   → true
```

3. Ricorsione ad albero

Il programma impostato nel riquadro applica la logica risolutiva del problema della *sottosequenza comune più lunga* (LCS) per confrontare una stringa di riferimento `r` con una diversa versione `q`, dove si assume che sia `q` che `r` siano costituite esclusivamente da lettere maiuscole o minuscole, non da altri tipi di carattere. In particolare, la procedura `xlcs` restituisce una stringa così composta: le lettere di `r` che trovano corrispondenza in `q` sono sostituite da un asterisco `*`; quelle da "cancellare" nella sottosequenza comue più lunga sono rimpiazzate da una barra inclinata `/`; inoltre, le lettere di `q` da "aggiungere" (cioè senza corrispondenza) rispetto alla stringa di riferimento `r` vengono incluse nelle posizioni corrispondenti. Esempi:

```
(xlcs "" "")       →  ""          (xlcs "arto" "atrio")     →  "*/*/t*"
(xlcs "" "ma")     →  "//"        (xlcs "atrio" "arto")     →  "*/*ri*"
(xlcs "ma" "")     →  "ma"        (xlcs "flora" "lira")     →  "f*/o**"
(xlcs "ma" "ma")   →  "**"        (xlcs "cincia" "piani")   →  "/c*/*c*a"
```

Completa il programma inserendo espressioni appropriate negli spazi indicati.

```
(define xlcs                    ; val: stringa
  (lambda (q r)                 ; q, r: stringhe
    (cond ((string=? r "") q)

          (  ..........

           (string-append  ..........  (xlcs q (substring r 1))))
          ((char=? (string-ref q 0) (string-ref r 0))
           (string-append "*" (xlcs (substring q 1) (substring r 1))))
          (else

           (better (string-append  .......... )

                   (string-append  .......... )
                   ))
          )))


(define better
  (lambda (u v)
    (if (> (stars u) (stars v))

        ..........

        ..........

        )))


(define stars
  (lambda (s)
    (if (string=? s "")
        0
        (let ((n (stars (substring s 1))))

          (if (char=?  .......... ) (+ n 1) n)
          ))))
```

4. Verifica formale della correttezza

In relazione alla procedura `f` dell'esercizio **1**, si può dimostrare che per ogni intero $n \geq 0$:

$$(\texttt{f}\ n\ 0) \quad \rightarrow \quad 2^n$$

Dimostra questa proprietà per induzione sui valori di n attenendoti allo schema delineato qui sotto.

- Formalizza la proprietà generale da dimostrare:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il caso / i casi base:

- Dimostra il passo induttivo:

5. Procedure con argomenti procedurali

Il programma impostato nel riquadro risolve il problema dei percorsi di Manhattan e restituisce una lista di interi, contenente tanti elementi quanti sono i percorsi diversi. Più specificamente, ciascun percorso è rappresentato nella lista dal numero di cambi di direzione — o *svolte* — che lo caratterizzano. Nel caso di una "mappa" reticolare 2 x 2, per esempio, sono possibili 6 percorsi diversi, che a due a due richiedono rispettivamente 1, 2 o 3 svolte; se punto di partenza e destinazione si trovano sulla stessa "strada", allora c'è un unico percorso senza svolte; se un lato del rettangolo ha lunghezza unitaria, sono possibili due percorsi con una sola svolta lungo il perimetro e altri percorsi con due svolte con attraversamenti all'interno del rettangolo:

`(mh 2 2)` → `(1 3 2 2 3 1)` `(mh 0 5)` → `(0)` `(mh 4 1)` → `(1 2 2 2 1)`

Completa il programma inserendo espressioni appropriate negli spazi indicati.

```
(define mh                      ; val:  lista di interi
  (lambda (i j)                 ; i, j:  interi non negativi
    (if (or (= i 0) (= j 0))
        (list 0)

        (append (mx true (- i 1) j)  ..........  )
        )))


(define mx                      ; down:  booleano  = true se passo precedente in giù /  = false se a destra
  (lambda (down i j)
    (cond ((and (= i 0) (= j 0))
           (list 0))
          ((= i 0)
           (list (if down 1 0)))
          ((= j 0)
           (list (if down 0 1)))
          (down
           (append (mx true (- i 1) j)

                   (map  ..........  (mx false i (- j 1)))
                   ))
          (else

           (append  ..........

                    ..........
                   ))
          )))
```

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

Riesci a intuire quale problema potrebbe risolvere, 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.