Corso di Programmazione

I Accertamento del 7 Dicembre 2006 / A

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 che si renda necessario.

1. Procedure in Scheme

Con riferimento alla procedura *f* così definita:

```
(define f
  (lambda (x)
    (let ((n (string-length x)))
      (cond ((< n 2) n)
            ((char=? (string-ref x 0) (string-ref x (- n 1)))
             (+ 2 (f (substring x 1 (- n 1)))))
            (else 0))
      )))
```

calcola i risultati della valutazione di ciascuna delle seguenti espressioni Scheme:

`(f "nono")`	→	0	`(f "osso")`	→	4
`(f "anilina")`	→	7	`(f "aciclica")`	→	6
`(f "eruttare")`	→	4	`(f "acquatica")`	→	4

2. Procedure in Scheme

Completa il programma *increment*, che calcola l'incremento di un numero naturale rappresentato come stringa di cifre in una base compresa fra *2* e *10*. Gli argomenti sono *num*, la stringa numerica, e *base*, di tipo intero; il valore restituito è una stringa numerica. Per esempio, il valore dell'espressione *(increment "1011" 2)* è *"1100"*, dove le stringhe rappresentano rispettivamente *11* e *12* in base *2*.

```
(define offset (char->integer #\0))

(define last-digit
  (lambda (base)  (integer->char (+ (- base 1) offset)) ))

(define next-digit
  (lambda (dgt)  ( __string_____  (integer->char (+ (char->integer dgt) 1))) ))

(define increment
  (lambda (num base)  ; 2 <= base <= 10

    (let ((digits (string-length num)))

      (if (= digits 0)

          "1"

          (let ((dgt  __(string-ref num (- digits 1))_____________________ ))

            (if (char=? dgt (last-digit base))

                (string-append  _(increment (substring num 0 (- digits 1)) base)_

                                "0")

                (string-append (substring num 0 (- digits 1))  _(next-digit dgt)_ )

                ))
    ))))
```

3. Definizione di procedure in Scheme

Derfinisci una procedura *string-and* in Scheme che, date due stringhe *x*, *y*, della stessa lunghezza *n* e costituite dai soli caratteri "0" e "1" (bit), assuma come valore la stringa *z* di lunghezza *n* che rappresenta l'*and* logico dei bit corrispondenti in *x* e *y*, cioè tale che l'*i*-imo bit di *z* è "1" se e solo se l'*i*-imo bit di *x* e l'*i*-imo bit di *y* sono entrambi "1". Per esempio, il risultato della valutazione dell'espressione *(string-and "010111" "110011")* è *"010011"*.

```
(define string-and
  (lambda (x y)
    (if (= (string-length x) 0)
        ""
        (let ((u (string-ref x 0)) (v (string-ref y 0)))
          (string-append
           (if (and (char=? u #\1) (char=? v #\1)) "1" "0")
           (string-and (substring x 1) (substring y 1))
           ))
    )))
```

4. Definizione di procedure in Scheme

Una funzione $q : \mathbf{N} \rightarrow \mathbf{Z}$ (dai naturali agli interi) può rappresentare la sequenza dei campioni di un suono registrato in formato digitale. Una tipica operazione consiste nello sfumare un suono per non interromperlo bruscamente, cosa che può essere fatta modulando l'intensità del suono con una funzione a decadimento esponenziale, per esempio dimezzandone l'intensità ogni *n* campioni: se $q(i)$ è l'*i*-imo campione del suono originale, allora il corrispondente campione del suono sfumato è $h(i) = round(q(i) / 2^{i/n})$ (arrotondamento intero del rapporto). Definisci in Scheme una procedura *fade-out* che, data una funzione *q* che rappresenta il suono originale e dato il numero *n* di campioni dell'intervallo di dimezzamento, restituisca la funzione *h* che rappresenta il suono sfumato.

```
(define fade-out
  (lambda (q n)
    (lambda (i)
      (inexact->exact (round (/ (q i) (expt 2 (/ i n))))))
    ))
```

5. Dimostrazioni per induzione

Considera la seguente procedura:

```
(define gf
  (lambda (n k)        ; 1 <= k <= n
    (cond ((= n 1) 1)
          ((even? n)   ; n pari
           (- (* 2 (gf (quotient n 2) (quotient (+ k 1) 2)))
              (remainder k 2)))
          (else
           (let ((x (gf (- n 1) (+ (remainder k (- n 1)) 1))))
             (if (> x k) (+ x 1) x)))
    )))
```

Dimostra per induzione che, data qualunque coppia di numeri naturali i, $v \in [1, 2^i]$, il risultato della valutazione dell'espressione $(gf\ 2^i\ v)$ è v. In particolare:

- Scrivi formalmente la proprietà che intendi dimostrare per induzione:

  $\forall\, i \in \mathbf{N}\ .\ \forall\, v \in [1, 2^i]\ .\ (gf\ 2^i\ v) \rightarrow v$

- Scrivi formalmente la proprietà che esprime i casi base:

  $\forall\, v \in [1, 1]\ .\ (gf\ 1\ v) \rightarrow v \qquad \equiv \qquad (gf\ 1\ 1) \rightarrow 1$

- Scrivi formalmente l'ipotesi induttiva: *fissato* $j \in \mathbf{N}^+$

  $\forall\, v \in [1, 2^{j-1}]\ .\ (gf\ 2^{j-1}\ v) \rightarrow v$

- Scrivi formalmente la proprietà che si deve dimostrare come passo induttivo: *per j fissato sopra*

  $\forall\, v \in [1, 2^j]\ .\ (gf\ 2^j\ v) \rightarrow v$

- Dimostra il caso base:

  immediato

- Dimostra il passo induttivo:

  $u \in [1, 2^j],\ j > 0$

  $(gf\ 2^j\ u) \rightarrow (-\ (*\ 2\ (gf\ 2^{j-1}\ \lfloor (u+1)/2 \rfloor))\ \ldots\ )$

  u pari: $\rightarrow (-\ (*\ 2\ (gf\ 2^{j-1}\ u/2))\ 0\ ) \rightarrow 2\,(u/2) - 0 = u$

  u dispari: $\rightarrow (-\ (*\ 2\ (gf\ 2^{j-1}\ (u+1)/2))\ 1\ ) \rightarrow 2\,((u+1)/2) - 1 = u$

6. Ricorsione di coda

Trasforma la procedura f dell'esercizio 1 in un programma che applica la *ricorsione di coda* al posto della ricorsione generale, mantenendo la stessa logica risolutiva.

```
(define f6
  (lambda (x)
    (f-tr x 0)
  ))


(define f-tr
  (lambda (x k)
    (let ((n (string-length x)))
      (cond ((< n 2) (+ n k))
            ((char=? (string-ref x 0) (string-ref x (- n 1)))
             (f-tr (substring x 1 (- n 1)) (+ 2 k)))
            (else k))
      )))
```

Corso di Programmazione

I Accertamento del 7 Dicembre 2006 / B

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 che si renda necessario.

1. Procedure in Scheme

Con riferimento alla procedura *f* così definita:

```
(define f
  (lambda (x)
    (let ((n (string-length x)))
      (cond ((< n 2) n)
            ((char=? (string-ref x 0) (string-ref x (- n 1)))
             (+ 2 (f (substring x 1 (- n 1)))))
            (else (f (substring x 1 (- n 1)))))
      )))
```

calcola i risultati della valutazione di ciascuna delle seguenti espressioni Scheme:

`(f "ala")`	→	3	`(f "rara")`	→	0
`(f "erigere")`	→	5	`(f "ottetto")`	→	7
`(f "eruttare")`	→	6	`(f "acquatica")`	→	5

2. Procedure in Scheme

Completa il programma *decrement*, che calcola il decremento di un numero naturale positivo rappresentato come stringa di cifre in una base compresa fra *2* e *10*. Gli argomenti sono *base*, di tipo intero, e *num*, la stringa numerica; il valore restituito è una stringa numerica. Per esempio, il valore dell'espressione *(decrement "1100" 2)* è *"1011"*, dove le stringhe rappresentano rispettivamente *12* e *11* in base *2*.

```
(define offset (char->integer #\0))

(define last-digit
  (lambda (base)  (integer->char (+ (- base 1) offset)) ))

(define prev-digit
  (lambda (dgt)  ( __string_____  (integer->char (- (char->integer dgt) 1))) ))

(define decrement
  (lambda (base num)  ; 2 <= base <= 10

    (let ((digits (string-length num)))

      (let ((dgt  __(string-ref num (- digits 1))_____________________________ ))

        (if (= digits 1)

            (prev-digit dgt)

            (if (char=? dgt #\0)

                (let ((prefix  _(decrement base (substring num (- digits 1)))_ ))

                  (string-append (if (string=? prefix "0") "" prefix)

                                 (string (last-digit base))))

                (string-append (substring num 0 (- digits 1))  _(prev-digit dgt)_ )

                ))
    ))))
```

3. Definizione di procedure in Scheme

Derfinisci una procedura *string-or* in Scheme che, date due stringhe *x*, *y*, della stessa lunghezza *n* e costituite dai soli caratteri "0" e "1" (bit), assuma come valore la stringa *z* di lunghezza *n* che rappresenta l'*or* logico dei bit corrispondenti in *x* e *y*, cioè tale che l'*i*-imo bit di *z* è "0" se e solo se l'*i*-imo bit di *x* e l'*i*-imo bit di *y* sono entrambi "0". Per esempio, il risultato della valutazione dell'espressione *(string-or "010110" "110011")* è *"110111"*.

```
(define string-or
  (lambda (x y)
    (if (= (string-length x) 0)
        ""
        (let ((u (string-ref x 0)) (v (string-ref y 0)))
          (string-append
           (if (or (char=? u #\1) (char=? v #\1)) "1" "0")
           (string-or (substring x 1) (substring y 1))
           ))
    )))
```

4. Definizione di procedure in Scheme

Una funzione $q : \boldsymbol{N} \rightarrow \boldsymbol{R}$ (dai naturali ai reali) può rappresentare la sequenza dei campioni di un suono registrato in formato digitale. Una tipica operazione consiste nello sfumare un suono per non interromperlo bruscamente, cosa che può essere fatta modulando l'intensità del suono con una funzione a decadimento esponenziale, per esempio dimezzandone l'intensità ogni *n* campioni: se *q(i)* è l'*i*-imo campione del suono originale, allora il corrispondente campione del suono sfumato è $g(i) = (0.5)^{-i/n} \cdot q(i)$. Definisci in Scheme una procedura *fade-out* che, data una funzione *q* che rappresenta il suono originale e dato il numero *n* di campioni dell'intervallo di dimezzamento, restituisca la funzione *g* che rappresenta il suono sfumato.

```
(define fade-out
  (lambda (q n)
    (lambda (i)
      (* (expt 0.5 (/ i n)) (q i)))
    ))
```

5. Dimostrazioni per induzione

Considera la seguente procedura:

```
(define fg
  (lambda (k n)        ; 1 <= k <= n
    (cond ((= n 1) 1)
          ((odd? n)    ; n dispari
           (let ((x (fg (+ (remainder k (- n 1)) 1) (- n 1))))
             (+ x (if (<= x k) 0 1))))
          (else
           (- (* 2 (fg (quotient (+ k 1) 2) (quotient n 2)))
              (remainder k 2)))
     )))
```

Dimostra per induzione che, data qualunque coppia di numeri naturali j, $u \in [1, 2^j]$, il risultato della valutazione dell'espressione $(fg\ u\ 2^j)$ è u. In particolare:

- Scrivi formalmente la proprietà che intendi dimostrare per induzione:

  $\forall\, j \in \mathbf{N}\ .\ \forall\, u \in [1, 2^j]\ .\ (fg\ u\ 2^j) \rightarrow u$

- Scrivi formalmente la proprietà che esprime i casi base:

  $\forall\, u \in [1, 1]\ .\ (fg\ u\ 1) \rightarrow u \quad \equiv \quad (fg\ 1\ 1) \rightarrow 1$

- Scrivi formalmente l'ipotesi induttiva: *fissato* $i \in \mathbf{N}^+$

  $\forall\, u \in [1, 2^{i-1}]\ .\ (fg\ u\ 2^{i-1}) \rightarrow u$

- Scrivi formalmente la proprietà che si deve dimostrare come passo induttivo: *per i fissato sopra*

  $\forall\, u \in [1, 2^i]\ .\ (fg\ u\ 2^i) \rightarrow u$

- Dimostra il caso base:

  immediato

- Dimostra il passo induttivo:

  $v \in [1, 2^i],\ i > 0$

  $(fg\ v\ 2^j) \rightarrow (-\ (*\ 2\ (fg\ \lfloor (v+1)/2 \rfloor\ \ 2^{i-1}))\ \ldots\ )$

  v pari: $\rightarrow (-\ (*\ 2\ (fg\ v/2\ \ 2^{i-1}))\ 0\ ) \rightarrow 2\,(v/2) - 0 = v$

  v dispari: $\rightarrow (-\ (*\ 2\ (fg\ (v+1)/2\ \ 2^{i-1}))\ 1\ ) \rightarrow 2\,((v+1)/2) - 1 = v$

6. Ricorsione di coda

Trasforma la procedura f dell'esercizio 1 in un programma che applica la *ricorsione di coda* al posto della ricorsione generale, mantenendo la stessa logica risolutiva.

```
(define f6
  (lambda (x)
    (f-tr x 0)
  ))

(define f-tr
  (lambda (x k)
    (let ((n (string-length x)))
      (cond ((< n 2) (+ n k))
            ((char=? (string-ref x 0) (string-ref x (- n 1)))
             (f-tr (substring x 1 (- n 1)) (+ 2 k)))
            (else (f-tr (substring x 1 (- n 1)) k)))
      )))
```