Laboratorio di Programmazione a.a. 2011-12

Esercitazione su temi d'esame

11 Gennaio 2012

1. Procedure in Scheme

Completa la procedura *match* che, date due stringhe di lettere *u* e *v*, restituisce la stringa delle corrispondenze *w* così definita: *w* ha la lunghezza della stringa più corta (fra *u* e *v*); se in una certa posizione *u* e *v* contengono la stessa lettera, allora anche *w* contiene quella lettera nella posizione corrispondente; se invece *u* e *v* contengono lettere diverse, *w* contiene il simbolo "asterisco" nella posizione corrispondente. Per esempio, il valore dell'espressione Scheme *(match "astrazione" "estremi")* è rappresentato dalla stringa delle corrispondenze *"*str**i"*.

```
(define match
  (lambda (u v)
    (if ( ____________  (string=? u "") (string=? v ""))
        ""
        (let ( (uh (string-ref u 0))  ___________________________________
               (s  ____________________________________________________ )
             )
          (if ( ____________________________  uh vh)
              (string-append  ____________________________________  s)
              (string-append "*" s)
              ))
    )))
```

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)  ( _________________  (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  ____________________________________________________ ))
            (if (char=? dgt (last-digit base))
                (string-append  ____________________________________________
                                "0")
                (string-append (substring num 0 (- digits 1))  ____________________ )
                ))
    ))))
```

3. Definizione di procedure in Scheme

Derfinisci formalmente una procedura *cyclic-string* in Scheme che, dati come argomenti una stringa *pattern* e un numero naturale *length*, assuma come valore la stringa di lunghezza *length* risultante la ripetizione ciclica di *pattern*, eventualmente troncata a destra. Per esempio, il risultato della valutazione dell'espressione *(cyclic-string "abcd" n)* per $n = 0, 1, 2, 4, 5, 11$ deve essere, rispettivamente: *"", "a", "ab", "abcd", "abcda", "abcdabcdabc"*.

4. Definizione di procedure in Scheme

Definisci una procedura *av* in Scheme che, data una lista non vuota $(x_1\ x_2\ \ldots\ x_n)$ i cui n elementi x_i appartengono all'insieme $\{-1, 0, 1\}$, restituisca la lista $(y_1\ y_2\ \ldots\ y_{n-1})$ di $n-1$ elementi dello stesso insieme tale che $y_i = -1$ se $x_i + x_{i+1} < 0$, $y_i = 0$ se $x_i + x_{i+1} = 0$ e $y_i = 1$ se $x_i + x_{i+1} > 0$. Per esempio:

```
(av '(0 0 -1 -1 1 0 0 1 0))      →      (0 -1 -1 0 1 0 1 1)
```

5. Procedure in Scheme

Un numero naturale n si dice *perfetto* se coincide con la somma dei suoi divisori propri (cioè diversi da n stesso). Per esempio, *28* è un numero perfetto in quanto si ottiene sommando i suoi divisori propri: $28 = 1 + 2 + 4 + 7 + 14$.
Definisci una procedura *perfect* in Scheme che, dato un numero naturale $n \geq 2$, restituisce la lista dei divisori di n se n è perfetto, la lista vuota altrimenti.

6. Definizione di procedure in Scheme

Valori numerici nell'intervallo *[0,1)* possono essere rappresentati in forma binaria da una stringa di cifre `"0"` e `"1"` precedute dal carattere `"."` (punto), dove i singoli bit sono pesati da potenze negative di due. Per esempio, le stringhe `".1"` e `".011"` corrispondono ai numeri *0.5* e *0.375*, rispettivamente, nella consueta notazione in base dieci.
Definisci una procedura *r-val* in Scheme per determinare il valore numerico di stringhe del tipo descritto sopra (punto seguito da una o più cifre binarie).

7. Programmazione in Scheme

Definisci una procedura *shared* in Scheme che, date due liste u, v (strettamente) *ordinate* di numeri interi positivi, restituisca la lista ordinata degli elementi comuni a u e v. Per esempio:

```
(shared '(1 3 5 6 7 8 9 10) '(0 1 2 3 4 5 7 9))      →      (1 3 5 7 9)
```

8. Definizione di procedure in Scheme

Derfinisci una procedura `prime-factors` in Scheme che, dato un numero intero positivo n, restituisce la lista dei fattori primi di n. In tale lista ciascun fattore primo compare tante volte quanto è il relativo esponente nella fattorizzazione di n. Per esempio:

```
(prime-factors 1)    →   '()
(prime-factors 15)   →   '(3 5)
(prime-factors 64)   →   '(2 2 2 2 2 2)
(prime-factors 180)  →   '(2 2 3 3 5)
```

9. Ricorsione di coda

Trasforma la procedura *power* riportata qui sotto in un programma che applica la *ricorsione di coda* al posto della ricorsione generale, mantenendo la stessa logica risolutiva.

```
(define power
  (lambda (b e)
    (cond ((= (remainder e 2) 0)
           (if (= e 0) 1 (power (* b b) (quotient e 2))))
          ((= (remainder e 3) 0) (power (* b b b) (quotient e 3)))
          (else (* b (power b (- e 1))))
          )))
```