

# Corso di Programmazione

Esame del 21 Gennaio 2013

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 Java

Un numero  $x$  compreso nell'intervallo  $] -0.5, +0.5[$  (cioè tale che  $|x| < \frac{1}{2}$ ) può essere codificato, almeno in forma approssimata, tramite una sequenza finita di cifre del sistema ternario bilanciato  $(-./+)$ , cifre che comparirebbero come parte frazionaria in questa notazione. Definisci in Java un metodo statico `fractPart` che, data una stringa *btf* di  $-./+$ , restituisce il valore numerico  $x$  di *btf*, interpretata come parte frazionaria nel sistema ternario bilanciato.

Esempi (i valori sono convenzionalmente approssimati alla terza cifra dopo il punto decimale):

```
fractPart( "--" ) → -0.444    fractPart( "-.+" ) → -0.296    fractPart( "." ) → 0.000
fractPart( ".++" ) → 0.148    fractPart( "+.-" ) → 0.296    fractPart( "+" ) → 0.333
```

## 2. Programmi in Scheme

Facendo riferimento alla procedura `tilings` definita nella pagina seguente, determina il risultato della valutazione di ciascuna delle espressioni riportate qui sotto:

(`tilings` 1) → .....

(`tilings` 4) → .....

(`tilings` 2) → .....

(`tilings` 6) → .....

(`tilings` 3) → .....

(`tilings` 8) → .....

```

(define tilings
  (lambda (n)
    (if (= n 0)
        1
        (+ (tilings (- n 1)) (red-tilings (- n 1)))))

(define red-tilings
  (lambda (n)
    (cond ((= n 0) 1)
          ((= n 1) 2)
          (else (+ (tilings (- n 1)) (tilings (- n 2))))))

```

### 3. Procedure con valori procedurali

Il *cifrario di Vigenère* (1523–1596) rappresenta una generalizzazione del cifrario di Cesare, in base al quale la rotazione delle lettere dell'alfabeto non è costante, ma varia ciclicamente in funzione di una parola chiave, concordata fra mittente e destinatario, che rappresenta le successive “mappature” della lettera iniziale A. Il meccanismo si può illustrare più facilmente ricorrendo a un esempio. Supponiamo che la parola chiave sia "AKEY". Questo significa che la prima lettera del messaggio verrà ruotata di 0 posizioni (A→A, B→B, C→C, ...); la seconda lettera del messaggio di 10 posizioni (A→K, B→L, C→M, ...); la terza di 4 (A→E, B→F, C→G, ...); la quarta di 24 (A→Y, B→Z, C→A, ...); la quinta di nuovo di 0 posizioni (A→A, ...) e così via ripetendo rotazioni della stessa entità con periodo 4 (= lunghezza della chiave). Quindi, a partire dal messaggio in chiaro "ASHORTSAMPLEMESSAGE" si può procedere secondo questo schema:

A S H O R T S A M P L E M E S S A G E	<i>messaggio in chiaro</i>
A K E Y A K E Y A K E Y A K E Y A K E	<i>parola chiave ripetuta</i>
A C L M R D W Y M Z P C M O W Q A Q I	<i>messaggio crittato</i>

Da cui risulta il testo crittato (terza riga): "ACLMRDWYMZPCMOWQAQI".

Data una parola chiave *key* (stringa di lettere maiuscole), la procedura `vigenere-cipher` restituisce la procedura di crittazione che applica il cifrario di Vigenère con chiave *key* per trasformare un messaggio in chiaro (stringa di lettere maiuscole) nel corrispondente testo crittato (stringa di lettere maiuscole). Per esempio:

```

(define encrypt (vigenere-cipher "AKEY"))

(encrypt "ASHORTSAMPLEMESSAGE") → "ACLMRDWYMZPCMOWQAQI"

```

Completa la procedura riportata qui sotto, che realizza `vigenere-cipher`, introducendo il codice Scheme appropriato negli spazi indicati a tratto punteggiato.

```

(define vigenere-cipher
  (lambda (key)

    ( .....

      (if (string=? msg "")

          .....

          (let ((r (- (char->integer (string-ref key 0)) cA))
                (c (- (char->integer (string-ref msg 0)) cA))
                (k (string-append (substring key 1) (substring key 0 1))))
            )
            (string-append

              (string (integer->char (+ cA ..... )))

              ( ..... (substring msg 1))
            )
          )
    )

    (define cA (char->integer #\A))
  )

```

#### 4. Verifica formale della correttezza

```
(define g
  (lambda (i j) ; i, j interi non negativi
    (+
      (if (or (= i 0) (= j 0)) 0 (- (g (- i 1) (- j 1)) 1))
      i
      j) ; somma di tre termini
    ))
```

In relazione alla procedura definita sopra è possibile dimostrare che per tutte le coppie di valori interi  $x, y \geq 0$ :

$$(g \ x \ y) \rightarrow xy + |x-y|$$

Dimostra per induzione questa proprietà; in particolare:

- 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. Memoization

Trasforma il programma in Scheme dell'esercizio 2 in un programma in Java più efficiente, applicando opportunamente una tecnica top-down di *memoization*.