Corso di Programmazione

II Accertamento del 13 Giugno 2011

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Astrazione sui dati in Scheme

Qui di seguito è riportato il codice della seconda realizzazione del dato astratto *"tavola rotonda"* discussa in classe. Il protocollo è definito dalle procedure `round-table`, per costruire una tavola rotonda con *n* cavalieri le cui etichette numeriche sono ordinate in senso orario e con la brocca assegnata al primo di essi; `last-knight-in?`, per verificare se in tavola è rimasto solo l'ultimo cavaliere; `knight-with-jug-in`, per conoscere l'etichetta del cavaliere con la brocca di sidro; `after-next-exit-from`, per effettuare un passo della conta (il cavaliere con la brocca serve il cavaliere alla sua sinistra, che esce, e passa la brocca al successivo).

```
(define round-table
  (lambda (n)
    (list (subrange 1 n) null n)
    ))


(define last-knight-in?
  (lambda (tab)
    (= (caddr tab) 1)
    ))


(define knight-with-jug-in caar)
```

```
(define after-next-exit-from
  (lambda (tab)
    (let ((p (caar tab))
          (n (caddr tab))
          )
      (if (= n 2)
          (list (list p) null 1)
          (let ((u (cdar tab))
                (v (cons p (cadr tab)))
                )
            (cond ((null? u)
                   (list (cdr (reverse v)) null (- n 1)))
                  ((null? (cdr u))
                   (list (reverse v) null (- n 1)))
                  (else
                   (list (cdr u) v (- n 1)))
                  )))
      )))
```

Si vuole estendere il protocollo introducendo una nuova procedura `next-knight-to-quit` che, data una (disposizione dei cavalieri attorno alla) tavola rotonda, restituisca l'etichetta del cavaliere che sta per essere servito e che sarà il prossimo ad abbandonare la tavola. Definisci la procedura `next-knight-to-quit` in modo che sia compatibile con la realizzazione considerata.

```
(define  next-knight-to-quit

  (lambda (tab)

    (if (null? (cdar tab))
        (car (reverse (cadr tab)))
        (cadar tab)
        )

     )  ; next-knight-to-quit
```

2. Astrazione sui dati in Scheme

Sulla base del protocollo esteso definito nell'esercizio precedente, scrivi un programma in Scheme per calcolare, dato il numero *n* di cavalieri, la lista delle etichette ordinata secondo l'ordine di uscita dalla tavola.

```
(define josephus
  (lambda (n)
    (count (round-table n))
    ))


;; segue
```

(seguito esercizio 2)

```
(define count
  (lambda (tab)
    (if (last-knight-in? tab)
        (list (knight-with-jug-in tab))
        (append (list (next-knight-to-quit tab)) (count (after-next-exit-from tab)))
        )
    ))
```

3. Programmi iterativi

Dati un intero *x* e un array ordinato di interi *v*, il seguente metodo statico in Java restituisce la posizione di *x* in *v*. Si assume che esista una componente dell'array di valore *x*. Completa il programma introducendo istruzioni appropriate negli spazi indicati a tratto punteggiato.

```
public static int pos( int x, int[] v ) {   // v: array ordinato (crescente); esiste un indice i tale che v[i] = x

  int p = 0;
  int q = v.length - 1;
  int m, n;

  while ( p < q ) {

    m = ( 2*p + q ) / 3;
    n = ( p + 2*q ) / 3;

    if ( x <= v[m] ) {

        q = m;

    } else if ( x > v[n] ) {

        p = n + 1;

    } else {

        p = m + 1;  q = n;

  }}
  return p;  // v[p] = x
}
```

4. Programmazione dinamica

Considera il seguente metodo statico formalizzato nel linguaggio Java:

```
public static long f( int x, int y ) {  // x, y ≥ 0
  if ( (x < 2) || (y < 2) ) {
    return  x * y;
  } else {
    return  f( x, y-1 ) + f( x-2, y ) + f( x-1, y+1 );
  }
}
```

Trasforma il programma ricorsivo in un programma iterativo applicando la tecnica di *programmazione dinamica*.

```
public static long f( int x, int y ) {

  if ( (x < 2) || (y < 2) ) {
    return x * y;
  }
  long[][] h = new long[ x+1 ][];

  for ( int i=0; i<=x; i=i+1 ) {
    h[i] = new long[ y+x+1-i ];
    h[i][0] = 0;  h[i][1] = i;
  }
  for ( int j=2; j<=y+x; j=j+1 ) {
    h[0][j] = 0;
  }
  for ( int j=2; j<y+x; j=j+1 ) {
    h[1][j] = j;
  }
  for ( int i=2; i<=x; i=i+1 ) {
    for ( int j=2; j<=y+x-i; j=j+1 ) {
      h[i][j] = h[i][j-1] + h[i-2][j] + h[i-1][j+1];
  }}
  return  h[x][y];
}
```

5. Verifica formale della correttezza

Dimostra formalmente la correttezza parziale (cioè assumendo la terminazione) del seguente programma iterativo in *Java* per calcolare la potenza intera di un intero. A tale proposito fai riferimento alle asserzioni (precondizioni, postcondizioni e invarianti) riportate nei commenti.

```
public static int power( int b, int e ) {
  // Pre:  b > 0,  e ≥ 0
  int x = b, y = e, u = 1, v = 1, r = y % 3;
  while ( y > 0 ) {
    // Inv:  u·x^y = v·b^e,  ∃q∈N.( y = 3q+r ),  0 ≤ r < 3,  y ≥ 0
    if ( r == 0 ) {
      x = x * x * x;  y = y / 3;  r = y % 3;
    } else if ( r == 2 ) {
      y = y + 1;  v = v * x;  r = 0;
    } else { // r == 1
      y = y - 1;  u = u * x;  r = 0;
  }}
  return  u / v;
  // Post:  u/v = b^e
}
```

L'invariante vale all'inizio: $1 \cdot b^e = 1 \cdot b^e$, $\exists q \in \mathbb{N}.(\ e = 3q+(e \bmod 3)\ )$, $0 \leq e \bmod 3 < 3$, $e \geq 0$

La seconda e terza asserzione dell'invariante esprimono proprietà della divisione intera.

Supponendo che l'invariante sia verificato in corrispondenza alla condizione del ciclo *while* e che segua un ulteriore passo iterativo ($y > 0$), si vuole verificare se l'invariante sarà ancora valido dopo il passo iterativo, ovvero:

(i) se $r = 0$

$u \cdot x^{3(y/3)} = v \cdot b^e$, $\exists q' \in \mathbb{N}.(\ y/3 = 3q'+(y/3 \bmod 3)\ )$, $0 \leq (y/3 \bmod 3) < 3$, $y/3 \geq 0$

Poiché $y \bmod 3 = r = 0$ si ha che $3(y/3) = y$;
la seconda e terza asserzione esprimono proprietà della divisione intera.

(ii) se $r = 2$

$u \cdot x^{y+1} = vx \cdot b^e$, $\exists q' \in \mathbb{N}.(\ y+1 = 3q'+0\ )$, $0 \leq 0 < 3$, $y+1 \geq 0$

Poiché $y \bmod 3 = r = 2$ si ha che $y+1$ è divisibile per 3 ($q' = q+1$).

(iii) se $r = 1$

$ux \cdot x^{y-1} = v \cdot b^e$, $\exists q' \in \mathbb{N}.(\ y-1 = 3q'+0\ )$, $0 \leq 0 < 3$, $y-1 \geq 0$

Poiché $y \bmod 3 = r = 1$ si ha che $y-1$ è divisibile per 3 ($q' = q$);
inoltre $y\text{-}1 \leq 0$ per la condizione del while.

Alla fine dell'iterazione si ha $u \cdot x^0 = v \cdot b^e$, da cui la conclusione