Corso di Programmazione

Esame del 1 Febbraio 2010

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. Memoization

Considera il seguente metodo formalizzato nel linguaggio *Java*:

```
public static int f( int i, int j ) {  // i, j >= 0
  if ( (i < 2) || (j < 2) ) {
    return i * j;
  } else {
    return f( i-2, j+1 ) + f( i+1, j-2 );
  }
}
```

Trasforma il programma ricorsivo applicando opportunamente la tecnica di *memoization*.

2. Procedure in Scheme

Con riferimento alla procedura `q` così definita:

```
(define q
  (lambda (x)
    (let ((n (string-length x)))
      (if (= n 0)
          "+"
          (let ((y (string-ref x (- n 1))))
            (if (char=? y #\+)
                (string-append (q (substring x 0 (- n 1))) "-")
                (string-append (substring x 0 (- n 1)) (if (char=? y #\-) "." "+"))
                ))
          ))))
```

calcola il risultato della valutazione di ciascuna delle seguenti espressioni Scheme:

`(q "-")` → _____ `(q "-.")` → _____

`(q ".")` → _____ `(q "+-+")` → _____

`(q "+")` → _____ `(q "+++")` → _____

`(q "--")` → _____ `(q "-+++")` → _____

3. Procedure con valori procedurali

Data la lista `args` degli arogmenti e la lista `vals` dei corrispondenti valori, dove le due liste hanno la stessa lunghezza e gli elementi di `args` sono tutti diversi fra loro, l'oggetto restituito dalla procedura `tab-fun` rappresenta la funzione che applicata al *k*-imo elemento di `args` assume come valore il *k*-imo elemento di `vals`. Tale funzione è definita solo per argomenti che appartengono alla lista `args`. Per esempio, sulla base della definizione

```
(define f (tab-fun '(0 1 2 3 4 5) '(2 3 5 7 11 13)))
```

ne risultano le seguenti valutazioni:

`(f 0)` → `2` `(f 3)` → `7`

`(f 1)` → `3` `(f 5)` → `13`

Completa la definizione della procedura `tab-fun` riportata qui sotto, introducendo il codice appropriato negli spazi indicati a tratto punteggiato.

```
(define tab-fun        ; valore: procedura
  (lambda (args vals)  ; args, vals: liste di uguale lunghezza
    ( ..........................................................................................
      (if (= ..................................  (car args))
          (car vals)
          ..........................................................................................
          ))
     ))
```

4. Verifica formale della correttezza

```
(define range-sum  ; valore: intero
  (lambda (a b)    ; a, b: interi tali che 0 <= a <= b
    (if (= a b)
        a
        (let ((k (quotient (+ a b) 2)))
          (+ (range-sum a k) (range-sum (+ k 1) b))
          ))
    ))
```

In relazione alla procedura definita sopra è possibile dimostrare che per tutte le coppie di interi m, n con $0 \le m \le n$:

$$\texttt{(range-sum } m\ n\texttt{)} \quad \rightarrow \quad (m+n)\cdot(n-m+1) / 2$$

Dimostra questa proprietà, procedendo per induzione sul valore della "distanza" $n-m$ fra i parametri; 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. Oggetti in Java

Supponi che sia disponibile una classe `Stack`, simile a quella discussa a lezione, per rappresentare stack i cui elementi sono di tipo `String`. Il protocollo di questa classe comprende: un costruttore che crea una struttura vuota; il metodo `empty()` che consente di determinare se lo stack è vuoto; il metodo `push(`*s*`)` che aggiunge la stringa *s* in cima allo stack; il metodo `pop()` che rimuove dalla struttura e restituisce la stringa in cima allo stack.

Attraverso un procedimento analogo a quello seguito per valutare numericamente un'espressione in "forma polacca inversa" (RPN), è possibile *tradurre* un'espressione RPN nell'espressione infissa equivalente, la cui valutazione determina l'applicazione delle stesse operazioni agli stessi operandi e nello stesso ordine. Per esempio, l'espressione RPN `"2 3 * 12 3 + 18 24 6 / - * +"` equivale a `"((2 * 3) + ((12 + 3) * (18 - (24 / 6))))"`. Come illustrato dall'esempio, ai fini di un trattamento uniforme delle (sotto)espressioni conviene introdurre una coppia di parentesi per ciascuna operazione aritmetica, anche se le parentesi risultano ridondanti.

Per ottenere questo risultato basta inserire nello stack le stringhe che codificano (sotto)espressioni Scheme, in sostituzione dei corrispondenti valori interi, a partire da quelle più elementari che rappresentano direttamente numeri. All'occorrenza di ciascuna operazione nell'espressione RPN, invece di procedere alla valutazione numerica, si combinano opportunamente le stringhe recuperate dallo stack, che rappresentano le sottoespressioni a cui l'operazione si applica, per formare un'espressione infissa più complessa.

Formalizza in Java un metodo `public static String rpnToInfix( String rpn )` che, data un'espressione RPN, restituisce l'espressione infissa equivalente (dove argomento e valore sono rappresentati da oggetti di tipo `String`), calcolata con l'ausilio di uno stack secondo le indicazioni fornite sopra. Per esempio, si vuole che:

```
rpnToInfix( "2 3 * 12 3 + 18 24 6 / - * +" ) ⟶ "((2 * 3) + ((12 + 3) * (18 - (24 / 6))))"
```