Corso di Programmazione

I Prova di accertamento del 4 Febbraio 2020 / A

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

1. Programmi in Scheme

```
(define f                              ; val: intero
  (lambda (x y u v)                    ; x, y ≥ 0 interi; u, v interi
    (cond ((and (= u 0) (= v 0)) 0)
          ((= x 0) (if (= u 0) 0 1))
          ((= y 0) (if (= v 0) 0 1))
          (else (+ (f (- x 1) y (- u 1) v) (f x (- y 1) u (- v 1))))
          )))
```

In relazione al programma riportato sopra, determina il risultato della valutazione delle seguenti espressioni:

`(f 0 5 0 3)`	→	.....	`(f 2 2 2 2)`	→	.....
`(f 5 0 3 -1)`	→	.....	`(f 4 5 2 3)`	→	66
`(f 5 1 3 0)`	→	.....	`(f 5 4 3 2)`	→	.....
`(f 2 2 1 1)`	→	.....	`(f 5 5 3 3)`	→	.....

2. Programmazione in Scheme

Scrivi un programma in Scheme, basato sulla procedura `standard-form`, per standardizzare tutte le parole contenute in una lista in modo che la lettera iniziale sia sempre una maiuscola. Le lettere successive di una stessa parola non vanno invece modificate. Esempi:

```
(standard-form '("abete" "betulla" "faggio" "quercia" "tiglio"))
                                  → ("Abete" "Betulla" "Faggio" "Quercia" "Tiglio")
(standard-form '("biancoSpino" "Pervinca" "primula" "RODODENDRO" "viola"))
                        → ("BiancoSpino" "Pervinca" "Primula" "RODODENDRO" "Viola")
```

(esercizio 2)

3. Ricorsione di coda

```
(define btr-val                          ; val: intero
  (lambda (btr)                          ; btr: stringa di – / . / +
    (let ((k (string-length btr))
          )
      (if (= k 0)
          0
          (let ((p (substring btr 0 (- k 1)))
                (t (string-ref btr (- k 1)))
                )
            (+ (* 3 (btr-val p)) (btd-val t))
            )))
    ))
```

```
(define btd-val
  (lambda (t)
    (cond ((char=? t #\-) -1)
          ((char=? t #\.)  0)
          ((char=? t #\+) +1)
          )
    ))
```

Data la rappresentazione ternaria bilanciata di un numero, codificata da una stringa composta dalle cifre '-', '.', '+', la procedura `btr-val` ne determnina il valore intero. Il programma impostato nel riquadro sotto è inteso a raggiungere lo stesso obiettivo attraverso l'invocazione di `btr-val-tr`, ma utilizzando *esclusivamente* la ricorsione di coda. Completa il programma inserendo variabili ed espressioni appropriate negli spazi indicati.

```
(define btr-val-tr   ; val: intero
  (lambda (btr)      ; btr: stringa di – / . / +

    (btr-val-rec  ..... )
    ))

(define btr-val-rec

  (lambda ( ..... )

    (let ((k (string-length  ..... ))
          )
      (if (= k 0)

          .....

          (let ((q (substring  ..... ))

                (t (string-ref  ..... ))
                )

            (btr-val-rec  ..... )
            )))
    ))
```

4. Verifica formale della correttezza

Per la procedura `f` dell'esercizio **1**, si può dimostrare che per ogni coppia di interi n, k tali che $n \geq 0$ e $0 \leq k \leq n$:

$$(\texttt{f}\ 1\ n\ 0\ k) \quad \rightarrow \quad k$$

Dimostra questa proprietà per induzione attenendoti allo schema delineato qui sotto.

- Indica il valore rispetto al quale intendi impostare la dimostrazione per induzione:

- Formalizza la proprietà che esprime il caso / i casi base:

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il passo induttivo:

5. Ricorsione ad albero e argomenti procedurali

Dati due interi non negativi *k*, *n*, la procedura `combinations` restituisce la lista di tutte le possibili stringhe composte da `0`/`1` di lunghezza *n* contenenti non più di *k* caratteri '`1`'. Esempi:

```
(combinations 0 3)  →  ("000")
(combinations 2 3)  →  ("110" "101" "100" "011" "010" "001" "000")
(combinations 5 3)  →  ("111" "110" "101" "100" "011" "010" "001" "000")
```

Completa il programma impostato nel riquadro inserendo espressioni appropriate negli spazi indicati.

```
(define combinations  ; val: lista di stringhe
  (lambda (k n)       ; k, n: interi non negativi
    (if (= n 0)

        .....

        (let ((u (if (= k 0)

                     .....

                     (combinations (- k 1)  ..... )
                     ))
              (v (combinations k (- n 1)))
              )
          (append

           (map  ..... )

           (map  ..... v)
           )))
    ))
```

6. Astrazione funzionale (ai fini della valutazione, questo quesito ha un peso molto minore degli altri)

Riesci a intuire quale problema potrebbe risolvere, o quale funzione calcola, la procedura `f` definita nell'esercizio **1**? Spiega in poche parole le tue intuizioni e le ragioni (il perché) alla base di esse.

Corso di Programmazione

I Prova di accertamento del 4 Febbraio 2020 / B

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

1. Programmi in Scheme

```
(define f                              ; val: intero
  (lambda (u v x y)                    ; u, v, x, y ≥ 0 interi
    (cond ((and (= x u) (= y v)) 0)
          ((= x 0) (if (= u 0) 0 1))
          ((= y 0) (if (= v 0) 0 1))
          (else (+ (f u v (- x 1) y) (f u v x (- y 1))))
          )))
```

In relazione al programma riportato sopra, determina il risultato della valutazione delle seguenti espressioni:

`(f 2 1 6 0)`	→	.....	`(f 1 1 2 2)`	→	.....
`(f 2 0 6 0)`	→	.....	`(f 1 1 4 5)`	→	56
`(f 0 4 1 6)`	→	.....	`(f 1 1 5 4)`	→	.....
`(f 0 0 2 2)`	→	.....	`(f 1 1 5 5)`	→	.....

2. Programmazione in Scheme

Scrivi un programma in Scheme, basato sulla procedura `lower-first`, per standardizzare tutte le parole contenute in una lista in modo che la lettera iniziale sia sempre una minuscola. Le lettere successive di una stessa parola non vanno invece modificate. Esempi:

```
(lower-first '("Abete" "Betulla" "Faggio" "Quercia" "Tiglio"))
                                  → ("abete" "betulla" "faggio" "quercia" "tiglio")
(lower-first '("BiancoSpino" "pervinca" "Primula" "RODODENDRO" "Viola"))
                        → ("biancoSpino" "pervinca" "primula" "rODODENDRO" "viola")
```

(esercizio 2)

3. Ricorsione di coda

```
(define ter-val                          ; val: intero
  (lambda (ter)                          ; ter: stringa di 0 / 1 / 2
    (let ((k (string-length ter))
          )
      (if (= k 0)
          0
          (let ((p (substring ter 0 (- k 1)))
                (t (string-ref ter (- k 1)))
                )
            (+ (* 3 (ter-val p)) (ted-val t))
            )))
    ))
```

```
(define ted-val
  (lambda (t)
    (cond ((char=? t #\0) 0)
          ((char=? t #\1) 1)
          ((char=? t #\2) 2)
          )
    ))
```

Data la rappresentazione in base tre di un numero, codificata da una stringa composta dalle cifre '`0`', '`1`', '`2`', la procedura `ter-val` ne determnina il valore intero. Il programma impostato nel riquadro sotto è inteso a raggiungere lo stesso obiettivo attraverso l'invocazione di `ter-val-tr`, ma utilizzando *esclusivamente* la ricorsione di coda. Completa il programma inserendo variabili ed espressioni appropriate negli spazi indicati.

```
(define ter-val-tr   ; val: intero
  (lambda (ter)      ; ter: stringa di 0 / 1 / 2

    (ter-val-rec  ..... )
    ))

(define ter-val-rec

  (lambda ( ..... )

    (let ((k (string-length ..... ))
          )
      (if (= k 0)

          .....

          (let ((q (substring ..... ))

                (t (string-ref ..... ))
                )

            (ter-val-rec ..... )
            )))
    ))
```

4. Verifica formale della correttezza

Per la procedura `f` dell'esercizio **1**, si può dimostrare che per ogni coppia di interi s, n tali che $n \geq 0$ e $0 \leq s \leq n$:

$$(\texttt{f}\ \ s\ \ 1\ \ n\ \ 1) \quad \rightarrow \quad n - s$$

Dimostra questa proprietà per induzione attenendoti allo schema delineato qui sotto.

- Indica il valore rispetto al quale intendi impostare la dimostrazione per induzione:

- Formalizza la proprietà che esprime il caso / i casi base:

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il passo induttivo:

5. Ricorsione ad albero e argomenti procedurali

Dati due interi non negativi *n*, *j*, la procedura `sequences` restituisce la lista di tutte le possibili stringhe composte da +/– di lunghezza *n* contenenti non più di *j* caratteri '–'. Esempi:

```
(sequences 3 0)  →  ("+++")
(sequences 3 2)  →  ("--+" "-+-" "-++" "+--" "+-+" "++-" "+++")
(sequences 3 5)  →  ("---" "--+" "-+-" "-++" "+--" "+-+" "++-" "+++")
```

Completa il programma impostato nel riquadro inserendo espressioni appropriate negli spazi indicati.

```
(define sequences   ; val: lista di stringhe
  (lambda (n j)     ; n, j: interi non negativi
    (if (> n 0)
        (let ((u (sequences (- n 1) j))
              (v (if (> j 0)

                     (sequences  ..........  (- j 1))

                     ..........
                     ))
              )
          (append

           (map  .......... )

           (map  .......... u)
           ))

         ..........
        )))
```

6. Astrazione funzionale (ai fini della valutazione, questo quesito ha un peso molto minore degli altri)

Riesci a intuire quale problema potrebbe risolvere, o quale funzione calcola, la procedura `f` definita nell'esercizio **1**? Spiega in poche parole le tue intuizioni e le ragioni (il perché) alla base di esse.