27 aprile 2011

Linguaggi di Programmazione

Cognome e nome	
N° fogli consegnati	

1. Specificare la definizione regolare relativa al simbolo lessicale **indirizzo**, che rappresenta l'indirizzo civico di una persona, come nei seguenti esempi:

```
Angelo B. Rossi
Via Europa 145
25100 Brescia
```

```
Angelo B. Matteo Rossi
Via Africa 12
27100 Pavia
```

```
T. Bianchi
Via Asia 2
64123 Bastia Franca
```

L'indirizzo deve rispettare i seguenti vincoli lessicali:

- La prima riga specifica il nome (in generale, uno o più nomi) e il cognome (uno solo) della persona;
- La seconda riga specifica la via (unico nome preceduto dalla keyword `Via`) e il numero civico;
- La terza riga specifica il CAP e la città (eventualmente composta da più nomi);
- Ogni elemento sulla stessa riga è separato dal successivo mediante un **blank**;
- Ogni riga è separata dalla successiva mediante un **newline**;
- Ogni identificatore (nome, cognome, indirizzo, città) inizia con una maiuscola ed è seguito da una o più minuscole;
- Un nome (o più) della persona può essere abbreviato dal primo carattere seguito dal punto;
- Il numero civico è composto da non più di 3 cifre e non può iniziare con una sequenza di zeri;
- Il CAP è composto da cinque cifre;

2. Specificare la grammatica BNF di un linguaggio in cui ogni frase è una lista (anche vuota) di dichiarazioni di variabili, come nel seguente esempio:

```
n, m: int;
a, b, c: record(x1, x2: real, y, z, w: string, i: int);
m: vector [10,30,20] of real;
lista1, lista2: sequence of string;
```

Oltre ai tipi semplici **`int`**, **`real`** e **`string`**, le espressioni di tipo coinvolgono i costruttori **`record`** (struttura), **`vector`** (vettore multidimensionale) e **`sequence`** (sequenza). Le dimensioni di un vettore sono rappresentate da costanti intere. I costruttori di tipo sono ortogonali tra loro ad eccezione del fatto che gli elementi di un vettore non possono essere ne record, ne sequenze, ne vettori.

3. Specificare la semantica operazionale dell'operatore relazionale di differenza (insiemistica) di tabelle:

```
X \ Y
```

sulla base dei seguenti requisiti:

- È richiesta solo la specifica operazionale dell'istanza del risultato (quindi, non lo schema);
- Si assume che le variabili che rappresentano le tabelle siano definite ed abbiano un valore (anche vuoto);
- Gli schemi dei due operandi devono essere compatibili per struttura;
- Per la specifica, sono disponibili le seguenti funzioni ausiliarie (di cui non è richiesta l'implementazione):

  `schema(t)`: schema della tabella `t`, espresso come array di coppie `(attributo,tipo)`;

  `istanza(t)`: istanza della tabella `t`, espressa come array di tuple;

  `length(a)`: numero di elementi dell'array `a`;

  `member(elem,a)`: appartenenza di `elem` all'array `a`;

  `insert(elem,a)`: inserimento di `elem` nell'array `a` (in coda);

- Nel caso di errore semantico, il risultato della differenza è `errore`.

4. Definire nel linguaggio *Scheme* la funzione `clear`, avente in ingresso una `lista`, che restituisce la lista in ingresso privata di tutti i suoi atomi (ad ogni livello). Ad esempio:

```
(clear '(x (y 10 (z w h)) (1) (a b)))
((()) () ())
```

5. È data la seguente dichiarazione nel linguaggio *Haskell*, relativa ad espressioni di liste di interi:

```
data Lexpr = List [Int]
           | Var String
           | Cat Lexpr Lexpr
           | Rev Lexpr

type State = [(String, [Int])]
```

in cui `List`, `Var`, `Cat` e `Rev` si riferiscono, rispettivamente, a una lista di interi, una variabile di tipo lista di interi, una concatenazione di liste di interi e una inversione di lista di interi, mentre `State` si riferisce alla associazione tra le variabili e le corrispondenti liste di interi. Si chiede di definire in *Haskell*, mediante la notazione di pattern-matching, la funzione `eval` (protocollo incluso) che, ricevendo in ingresso una espressione di liste di interi `e` ed uno stato `s`, genera il valore di `e` nello stato `s`. Si può fare uso delle funzioni della libreria standard di manipolazione delle liste (di cui non è richiesta la specifica).

6. È data una base di fatti *Prolog* che specifica la grammatica BNF di un linguaggio in termini di simboli nonterminali (di cui il primo è l'assioma), simboli terminali e produzioni. Ecco un esempio:

$S \rightarrow \mathbf{a}\ A\ \mathbf{b}$
$A \rightarrow A\ \mathbf{c}\ B$
$B \rightarrow \mathbf{c}\ A\ \mathbf{c} \mid \mathbf{a}\ \mathbf{c}\ B$
$C \rightarrow \mathbf{c}\ A \mid \mathbf{a}$

⇨

```
terminali([a,b,c]).
nonterminali(['S','A','B','C']).
produzioni([prod('S',[a,'A',b]),
            prod('A',['A',c,'B']),
            prod('B',[c,'A',c]),
            prod('B',[a,c,'B']),
            prod('C',[c,'A']),
            prod('C',[a])]).
```

Si chiede di specificare in *Prolog* il predicato `ricorsiva(N)`, che risulta vero se e solo se la grammatica include una produzione (direttamente) ricorsiva relativa al nonterminale `N`. Ad esempio:

```
?- ricorsiva(X).
X = 'A' ;
X = 'B' ;
false.
```

7. Dopo aver illustrato il significato generale della forma funzionale `foldr` in *Haskell*, stabilire come viene interpretata la seguente espressione e determinarne il risultato:

```
foldr (++) [] ["alfa","beta","gamma"]
```