Linguaggi e Traduttori

28 gennaio 2013

Usando lo scanner JFLEX e il parser CUP, realizzare un programma in JAVA capace di eseguire delle operazioni su degli oggetti.

Linguaggio di Ingresso

Il file di ingresso è diviso in tre sezioni: header, lista di oggetti con relativi attributi, e una lista di operazioni da eseguire. Ogni sezione è separata dalla successiva mediante il token "###".

La sezione header è composta da due tipi di token:

- <codice>: inizia con **almeno 2 caratteri "X"** o **"Y"** disposti in qualsiasi ordine e **in numero pari** (es. XX, XYXY, XXXXXX, XXXXXY). Essi sono seguiti o da un **numero di 2 o 5** cifre o da una parola composta da lettere alfabetiche minuscole (**in numero dispari, almeno 3**). **Opzionalmente** <codice> è terminato da un numero pari di caratteri "+".
- <data>: una data con il formato "GG/MM/AAAA" compresa tra il 05/02/2012 e il 12/04/2012. Si ricordi che il mese di febbraio del 2012 era composto da 29 giorni.

I token <data> e <codice> possono apparire nella sezione header **in ogni ordine**, ma deve essere garantito che appaiano entrambi **almeno una volta**. (Lo si gestisca con la grammatica).

La sezione della lista di oggetti è formata da una lista di **almeno 2** oggetti in **numero pari**. Ogni elemento della lista è composto da un **<nome>** (che rappresenta il nome dell'oggetto), seguito dal simbolo ":", da un <id_oggetto> (che è **opzionale**), da un simbolo ":" e da un blocco, delimitato da "[[" e da "]]". All'interno del blocco è presente una lista **anche vuota** di <attributi> separati dal carattere ",". Ogni **<attributo>** è composto da un **<nome>**, dal simbolo ":" e da un numero intero e positivo. <nome> è una qualsiasi sequenza di lettere, numeri e caratteri "_", iniziante con una lettera; <id_oggetto> è il simbolo "%" seguito da un numero compreso **tra -31 e 145**.

La sezione delle operazioni è una lista **anche vuota** di <operazioni> con un numero **pari** di elementi. Ogni <operazione> è composta da una coppia <oggetto_attributo> (un <nome> che rappresenta il nome dell'oggetto, i caratteri "->" e un <nome> che rappresenta il nome dell'attributo). <oggetto_attributo> rappresenta il valore di un attributo di un oggetto, dovrà essere ricavato tramite l'accesso ad una struttura globale popolata nella sezione precedente.

Dopo <oggetto_attributo> c'è il carattere ":" e una lista di <equazioni> in cui ogni <equazione> è separata dalle altre dal carattere ",". Ogni <operazione> è terminata dal carattere ";".

Ogni <equazione> è una combinazione qualsiasi di numeri interi, di coppie <oggetto_attributo> e dei classici operatori aritmetici (+, -, *, /), con relative parentesi.

Scopo

Il compilatore analizzando la seconda sezione dovrà popolare una struttura dati contenente tutte le informazioni necessarie per eseguire la terza sezione. **Tale struttura dati sarà l'unica variabile globale permessa.**

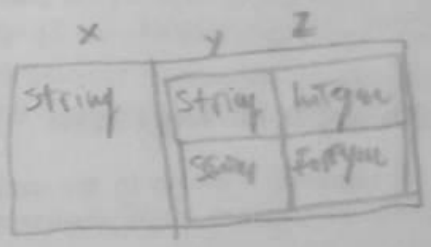
Nella terza sezione ogni singola <equazione> dovrà essere eseguita e il risultato **moltiplicato** per il valore <oggetto_attributo> posizionato all'inizio di ogni <operazione>. Il risultato di ogni singola moltiplicazione dovrà essere stampato a video. Usare gli attributi sintetizzati per eseguire le equazioni e gli attributi ereditati per accedere al valore <oggetto_attributo> posizionato all'inizio di <operazione>. **Non sono ammesse variabili globali.**

Alla fine di ogni <operazione> (si ricorda che un <operazione> è ogni singolo elemento della lista di <operazioni>) si richiede di stampare il **valore minimo** tra i risultati delle singole equazioni. Anche in questo caso **non sono ammesse variabili globali**.

Ad esempio con la seguente <operazione> e con x->n uguale a 2:

```
x->n : I-10 : 1+2, 4, 5, 6, 7;
```

il programma dovrà stampare:



```
6
8
10
12
14
MINIMO: 6
```

Esempio

Input:

```
XYabcde
29/02/2012
XXXXXY24++
YYYY12347++++
###
OGG1 : %-29 : [[ Alt:10, Larg:2, Prof1:1, peso:4 ]]
OGG2 : %115 : [[
  Alt:10, Larg:2,
  Prof:3
]]
OGG3 : %0 : [[ Alt:5, Larg:5, Prof:5, peso:7, val:2 ]]
X_1 : : [[ NUM:2 ]]
X2 : %-12 : [[ ]]
X3 : %-13 : [[ NUM:4 ]]
###
X_1->NUM : (OGG1->Alt*2+1) * OGG1->Prof1,
        OGG2->Larg*OGG2->Prof-2+2*3,
        OGG3->Larg*OGG3->Prof;
OGG3->val : OGG3->val*OGG1->peso, OGG3->val*OGG3->peso;
```

Output:

```
42
20
50
MINIMO: 20
16
28
MINIMO: 16
```