Linguaggi e Traduttori

11 febbraio 2011

Usando lo scanner JFLEX e il parser CUP, realizzare un programma in JAVA capace di interpretare un semplice linguaggio di programmazione.

Linguaggio di Ingresso

Il file di ingresso è diviso in due sezioni: un header seguito dal linguaggio di programmazione da interpretare. La sezione degli header e del linguaggio di programmazione sono separati dai caratteri "##". Nel file **non sono presenti righe vuote**.

La sezione header è composta da tre tipi di token:

- <data>: una data con il formato "GG/MM/AAAA" compresa tra il 03/07/2010 e il 11/02/2011. Si ricordi che i mesi di settembre e novembre sono composti da soli 30 giorni.
- <codice1>: è composto da un **numero pari (almeno 4)** di caratteri "!" seguiti da un numero compreso tra -18 e 285, o da un **numeri dispari (almeno 5)** di caratteri "?".
- <codice2>: è composto dal carattere "X" seguito **opzionalmente** da un numero di lunghezza 4 o 7 caratteri.

Nel header i tre tipi di token possono apparire **in qualsiasi ordine e numero (anche 0)**, ad eccezione del token <codice1> che deve apparire **esattamente due** volte (lo si gestisca con la grammatica). I token sono posti uno per riga.

La sezione del linguaggio di programmazione è composto da una lista di istruzioni in **numero dispari, almeno 3** (si produca una lista **ricorsiva destra**). Esistono 3 tipi di istruzioni: PUT, ADD e EQ. L'istruzioni PUT e EQ possono avere come operandi delle <formula_matematica>, mentre l'istruzione ADD può avere sia <formula_matematica> che <offset>. Una <formula_matematica> è qualsiasi tipo di espressione matematica racchiusa tra parentesi tonde, ad es. $(3 + 2 * 5)$ o $((3+2) * 5)$. <offset> è un numero negativo. Ogni istruzione è ultimata da un carattere di new line, come nel caso dei tocken visti in precedenza.

- **PUT <formula_matematica>**: Esegue l'operazione matematica e ne stampa il risultato.
- **EQ <formula_matematica1>, <formula_matematica2>**: confronta gli ultimi due risultati ottenuti dalle operazioni eseguite precedentemente (R_{-1} e R_{-2}). Nel caso in cui $R_{-1} = R_{-2}$ esegue <formula_matematica1> e ne stampa il risultato, **alternativamente se** R_{-1} <> R_{-2} **esegue <formula_matematica2> e ne stampa il risultato. Esiste anche una versione compatta dell'istruzione EQ in cui non esiste <formula_matematica2> (EQ <formula_matematica1>), in tal caso, nel caso i valori** R_{-1} **e** R_{-2} **siano diversi restituirà come risultato il valore 0.**

- **ADD <lista_operandi>**: esegue la somma degli elementi di <lista_operandi> e ne stampa il risultato. <lista_operandi> può essere anche vuota, in tal caso il risultato della funzione ADD sarà 0. Operandi possono essere di tipo <formula_matematica> o di tipo <offset>. <offset> è un numero negativo che rappresenta i risultati di operazioni fatte in precedenza. In particolare -1 rappresenta il risultato R_{-1} cioè il risultato dell'ultima operazione eseguita, e così via. -2 rappresenta il risultato R_{-2}, cioè il risultato della penultima operazione eseguita. Ad esempio `ADD -3, (2*5-4)` deve restituire come risultato R_{-3} (cioè il risultato ottenuto dalla terzultima operazione) sommato al risultato dell'operazione $(2 * 5 - 4)$. Nel caso di $R_{-3} = 2$ dovrà fornire come risultato 8.

Scopo

Il compilatore dovrà eseguire il linguaggio descritto. Si ipotizzi che le funzioni siano chiamate in modo corretto (ad esempio EQ non può essere chiamata come prima istruzione del programma in quanto non avrebbe i due risultati R_{-1} e R_{-2} su cui essere applicata).

Non sono accettate variabile globali: usare direttamente lo stack del parser e gli attributi ereditati al fine di ottenere i risultati delle precedenti operazioni.

CONSIGLIO: Fare la lista delle istruzioni **ricorsiva destra**, così tutti i risultati delle precedenti operazioni rimangono nello stack del parser (e possono essere acceduti tramite attributi ereditati dalle istruzioni successive). Essi verranno ridotti solo quando viene eseguita l'ultima istruzione del programma.

Esempio

Input:

```
!!!!!!-12
X1234
X
???????
05/08/2010
##
PUT (10)
PUT (10+2*2)
EQ (2*4*5)
PUT (0)
EQ ((10+2)*2), (10)
ADD
ADD (10*2), -2, -6
```

Output:

```
10
14
0
0
24
0
54
```