Laboratorio 4

Insegnamento di Linguaggi e computabilità

- L’analizzatore lessicale che viene generato da JFlex non è altro che l’implementazione di un automa a stati finiti
- Finora abbiamo ignorato gli stati, lasciando l’automa sempre nell’unico stato iniziale presente per impostazione predefinita (ovvero, YYINITIAL)
- È possibile definire nuovi stati (in realtà di due tipi diversi, ma la loro trattazione va oltre le nostre finalità)

Esempio «TrovaCommenti» … che stampa i commenti estratti da un ipotetico testo, nella forma /* commento */ … usando gli stati

```
1.  %%
2.  %class TrovaCommenti
3.  %standalone
4.  %state COMMENTO
5.  INIZIOCOMMENTO = "/*"
6.  FINECOMMENTO = "*/"
7.  %%
8.  <YYINITIAL > {
9.  {INIZIOCOMMENTO} {    yybegin(COMMENTO); } [^] { }              }
10. <COMMENTO > { .* {FINECOMMENTO} {
11. System.out.println("Trovato commento: " + yytext().substring (0,
    yylength() - 2));
12. yybegin(YYINITIAL); }
13. }
```

- È definito un nuovo stato di nome COMMENTO
- Quando l'automa si trova nello stato iniziale e il frammento di input viene rilevato essere l'inizio di un commento, l'automa cambia stato usando il metodo yybegin(NUOVO_STATO)
- Si entra nello stato COMMENTO da cui si esce dopo una sequenza qualsiasi di caratteri terminante con il pattern di fine commento. A quel punto si eseguono le due azioni associate
  - viene stampato il commento trovato (riga 11) eliminando gli ultimi due caratteri che rappresentano il fine commento e
  - viene riportato l'automa nello stato iniziale con yybegin(YYINITIAL)

Esercizio da consegnare- FileConfigurazione (1)

1. Scrivere parser e lexer per permettere che un file di configurazione conforme alla seguente sintassi:

```
nome_gruppo_1: {
        nome_proprietà_1: valore_proprietà_1
        nome_proprietà_2: valore_proprietà_2
        ...
        };
nome_gruppo_2: {
        nome_proprietà_x: valore_proprietà_x
        nome_proprietà_y: valore_proprietà_y
        ...
        };
```

sia trasformato in un file di configurazione corrispondente e conforme alla seguente sintassi:

```
[nome_gruppo_1]
        nome_proprietà_1 = valore_proprietà_1
        nome_proprietà_2 = valore_proprietà_2
        ...
[nome_gruppo_2]
        nome_proprietà_x = valore_proprietà_x
        nome_proprietà_y = valore_proprietà_y
```

Esempio

```
global: {
  num thread: 4
  config: /etc/conf.ini
};

host: {
  ip: 10.0.0.1
  name: Pluto
};
```

➔

```
[global]
num thread=4
config=/etc/conf.ini

[host]
ip=10.0.0.1
name=Pluto
```

Esercizio da consegnare - FileConfigurazione (2)

Modificare il parser e lexer realizzati in modo che, dato questo input:

Studente: {

"nome" : ”PROPRIO NOME E COGNOME",

"matricola" : 0

}

venga restituito con il seguente formato e inserendo il proprio numero di matricola al posto dello 0

[Studente]

nome = Lorenzo

matricola = 817151

Suggerimenti per lo svolgimento

- È possibile modificare la parte di codice user defined e le definizioni iniziali dell'esempio della calcolatrice (calc.l e calc.y distribuiti con il materiale per il Secondo laboratorio)
- Identifico tutti i possibili token: parole chiave (es., "global", "ip", ecc.), spazi e ritorni a capo, altri simboli (es., parentesi, punti e virgola, due punti, ecc.), e altre stringhe di caratteri
- Dichiaro nel file .l le macro che definiscono i pattern per il riconoscimento dei token, e gli stati per le start conditions (se necessari);
- Dichiaro nel file .y tutti i token e i simboli non terminali della grammatica (eventualmente associando loro un tipo di dato "sval", "ival", "dval", ecc.)
- Definisco tutte le produzioni della grammatica nella seconda parte del file .y, utilizzando le variabili $ (cioè, $$, $1, $2, ecc.) per il passaggio dei valori da un simbolo all'altro (i.e., dal corpo della produzione alla testa)
- Richiamando le macro, oppure usando direttamente i pattern, definisco le regole lessicali, tenendo presente che:
    - Per informare il parser che è stato riconosciuto un token, l'ultima istruzione delle azioni di una regola lessicale deve essere una return, che usi la classe Parser e le sue costanti che identificano i token, ad esempio: return Parser.GROUP_NAME;
    - Per restituire il valore di un token, nel caso non sia una sequenza prefissata di caratteri ma un testo variabile, bisogna usare la classe ParserVal, nel seguente modo: yyparser.yylval = new ParserVal(yytext());
- N.B, - se il valore da passare al parser non è una stringa, si può operare come segue (nel caso di valore intero, ma il caso di valore double è analogo): yyparser.yylval = new ParserVal(new Integer(yytext()));
    - Per passare da uno stato del lexer ad un altro si usa la funzione yybegin(): yybegin(STATO);