


POLITECNICO DI TORINO

Laboratorio di Compilatori
Corso di Linguaggi e Traduttori

Esercitazione I



Stefano Scanzio
mail: stefano.scanzio@polito.it
sito: <http://www.skenz.it/traduttori>

a.a 2008 / 2009

Linguaggi?

- **Lessico (Esercitazione 1)**
 - Sapete qual è il colmo per un tennista? Ridere sempre alle battute!
 - ▲ Le parole devono appartenere al vocabolario italiano
 - Spteea lauq è li omcol erp nu nintaste? Rirdee mperes llea tutteab!
 - Scanner JFlex: compie un'analisi lessicale
- **Grammatica (Esercitazione 2)**
 - Sapete qual è il colmo per un tennista? Ridere sempre alle battute!
 - È qual sapete colmo un per tennista? Sempre alle battute ridere!
 - Parser Cup: compie un'analisi grammaticale e semantica
- **Semantica (Esercitazione 3)**
 - Il cane rincorre il gatto
 - Il cane vola

http://www.skenz.it/traduttori Esercitazione 1 2

Espressioni regolari

- Costituiscono un metodo semplice ed efficace per descrivere insiemi di stringhe di caratteri.
- Attraverso le espressioni regolari si può definire il lessico di un linguaggio
- Opportuni operatori consentono di indicare

■ caratteri	'c' o anche c
■ classi di caratteri	[a,b,c] o [a-c]
■ opzionalità	exp ?
■ ripetizione (0 o più volte)	exp *
■ ripetizione (1 o più volte)	exp +
■ alternativa	exp1 exp2
■ concatenazione	exp1 exp2
■ ambito di operatori	(exp)

http://www.skenz.it/traduttori Esercitazione 1 3

Esempi di espressioni regolari


- numero intero positivo
 - [0-9]+
- numero intero positivo senza 0 iniziali
 - [1-9][0-9]*
- numero intero positivo o negativo
 - ('+'|-)? [0-9]+
- numero in virgola mobile
 - ('+'|-)? (([0-9]+ '.' [0-9]*) | ([0-9]* '.' [0-9]+))

gli apici consentono di distinguere un carattere in ingresso ('+') da un operatore (+).

http://www.skenz.it/traduttori Esercitazione 1 4

JFlex - un generatore di analizzatori lessicali

- Poiché la trasformazione di espressioni regolari in automi a stati finiti deterministici e la implementazione di questi ultimi sono processi meccanici (e noiosi), spesso si utilizza un generatore automatico di analizzatori lessicali.
- JFlex è un generatore che accetta in ingresso un insieme di espressioni regolari e di azioni associate a ciascuna espressione e produce in uscita un programma Java che riconosce tali espressioni.



```

graph LR
    A[Espressioni regolari] --> B[JFlex]
    B --> C[Programma Java]
            
```

http://www.skenz.it/traduttori Esercitazione 1 5

Espressioni regolari in JFlex

- Le espressioni regolari descrivono sequenze di caratteri ASCII ed utilizzano un certo numero di operatori:
 - "\ [^ - ? . * + | () \$ / { } % < >
- Lettere e numeri del testo di ingresso sono descritti mediante se stessi:
 - l'espressione regolare val1 rappresenta la sequenza 'v' 'a' 'l' '1' nel testo di ingresso
- I caratteri non alfabetici vengono rappresentati in JFlex racchiudendoli tra doppi apici, per evitare ambiguità con gli operatori:
 - l'espressione xyz"+" rappresenta la sequenza 'x' 'y' 'z' '+' '+' nel testo di ingresso.

http://www.skenz.it/traduttori Esercitazione 1 6

Espressioni regolari in JFlex

...continua...

- I caratteri non alfabetici possono essere anche descritti facendoli precedere dal carattere \
 - l'espressione xyz\+ rappresenta la sequenza 'x' 'y' 'z' '+' '+' nel testo di ingresso.
- Le classi di caratteri vengono descritte mediante gli operatori []:
 - l'espressione [0123456789] rappresenta una cifra nel testo di ingresso.
- Nel descrivere classi di caratteri, il segno - indica una gamma di caratteri:
 - l'espressione [0-9] rappresenta una cifra nel testo di ingresso.
 - L'espressione [a-z] tutti i caratteri minuscoli
 - L'espressione [a-zA-Z0-9] tutti i caratteri sia minuscoli che maiuscoli e le cifre

<http://www.skenz.it/traduttori> Esercitazione I 7

Espressioni regolari in JFlex

...continua...

- Per includere il carattere - in una classe di caratteri, questo deve essere specificato come primo o ultimo della serie:
 - l'espressione [-+0-9] rappresenta una cifra o un segno nel testo di ingresso.
- Nelle descrizioni di classi di caratteri, il segno ^ posto all'inizio indica una gamma di caratteri da escludere:
 - l'espressione [^0-9] rappresenta un qualunque carattere che non sia una cifra nel testo di ingresso.
- L'insieme di tutti i caratteri eccetto il fine riga viene descritto mediante il simbolo ".".

<http://www.skenz.it/traduttori> Esercitazione I 8

Espressioni regolari in JFlex

...continua...

- Il carattere di fine riga viene descritto dall'espressione regolare
 - \n|\r|\r\n (r line feed - \n carriage return)
 - Questo perché jflex è scritto in java e deve poter funzionare correttamente sia su sistemi operativi Windows che Linux
 - NB:
 - ▲ \n|\r|\r\n -> Un solo simbolo di a capo
 - ▲ [\n\r]+ -> Più simboli di a capo: \n\n\r\r
- Il carattere di tabulazione viene descritto dal simbolo \t.
- L'operatore ? Indica che l'espressione precedente è opzionale:
 - ab?c indica sia la sequenza ac che abc.

<http://www.skenz.it/traduttori> Esercitazione I 9

Espressioni regolari in JFlex

...continua...

- L'operatore * indica che l'espressione precedente può essere ripetuta 0 o più volte:
 - ab*c indica tutte le sequenze che iniziano per a, terminano per c e hanno all'interno un numero qualsiasi di b.
- L'operatore + indica l'espressione precedente può essere ripetuta 1 o più volte:
 - ab+c indica tutte le sequenze che iniziano per a, terminano per c e hanno all'interno almeno un b.
 - abc, abbc, abbbc : OK
 - ac : NO!!!
- L'operatore | indica un'alternativa tra due espressioni:
 - ab|cd indica sia la sequenza ab che la sequenza cd.
- Le parentesi tonde consentono di esprimere la priorità tra operatori:
 - (ab|cd+)?ef indica sequenze tipo ef, abef, cdddef.

<http://www.skenz.it/traduttori> Esercitazione I 10

La struttura di un programma sorgente JFlex

- Un file sorgente per JFlex è composto di tre sezioni distinte separate dal simbolo '%%'.
 - La prima sezione contiene il codice utente e può essere vuota. Verrà chiamata sezione del codice.
 - La seconda sezione contiene opzioni e dichiarazioni. Verrà chiamata sezione delle dichiarazioni.
 - La terza sezione contiene regole lessicali sotto forma di coppie espressione_regolare azione. Verrà chiamata sezione delle regole.

Sezione del codice
%%
Sezione delle dichiarazioni
%%
Sezione delle regole

<http://www.skenz.it/traduttori> Esercitazione I 11

Sezione del codice

- Tutte le righe presenti nella sezione del codice vengono ricopiate senza alcuna modifica nel programma generato da JFlex.
- Solitamente in questa parte del codice si inseriscono dichiarazioni a librerie java che verranno utilizzate più avanti nel codice
- Esempio:


```
import java.io.*; (Se avessi intenzione di utilizzare la libreria di I/O di Java)

import java_cup.runtime.*; (Per avere la compatibilità con l'analizzatore grammaticale Cup)
```

<http://www.skenz.it/traduttori> Esercitazione I 12

Sezione delle dichiarazioni

- Per semplificare la gestione di espressioni regolari complesse o ripetitive, è possibile definire identificatori che designano sotto-espressioni regolari.
- Definizione di numero intero:

```
numero = [+ -]?[0-9]+
```
- La sotto-espressione così definita può essere utilizzata nella sezione delle regole, racchiudendone il nome tra parentesi graffe:

```
{numero} {
    System.out.println("trovato numero\n");
}
```
- Si può includere del codice Java nella sezione delle dichiarazioni tramite l'uso degli operatori '%{' e '%}'.

<http://www.skenz.it/traduttori>

Esercitazione 1

13

Sezione delle regole

- Ad ogni espressione regolare è associata in JFlex un'azione che viene eseguita all'atto del riconoscimento.
- Le azioni sono espresse sotto forma di codice Java e devono essere racchiuse tra parentesi graffe.
- L'azione più semplice consiste nell'ignorare il testo riconosciuto: si esprime un'azione nulla con il simbolo {};

AZIONE:

```
\n | \r | \r\n {
    System.out.println("Trovato simbolo di a capo");
}
```

<http://www.skenz.it/traduttori>

Esercitazione 1

14

Azioni associate alle espressioni regolari

- Il testo riconosciuto viene accumulato in un buffer e può essere letto attraverso la funzione:
 - string yytext()
- Il numero di caratteri riconosciuti viene restituito dalla funzione:
 - int yylength()
- Restituisce il carattere corrispondente ad una determinata posizione:
 - char yycharat(int pos)
- Contengono rispettivamente la linea e la colonna corrente:
 - int yyline
 - int yycolumn
- Contiene il numero di carattere attuale conteggiato a partire dalla posizione 0 del file di ingresso (solo con la direttiva %char)
 - int yychar

<http://www.skenz.it/traduttori>

Esercitazione 1

15

Esempio

```
%%
euro = [1-9][0-9]*" "[0-9][0-9] | 0? "[0-9][0-9]
lire = [1-9][0-9]*
%%
{euro} { System.out.println("Euro: " + yytext()); }
{lire} { System.out.println("Lire: " + yytext()); }
```

INGRESSO	USCITA
0.02	Euro: 0.02
.10	Euro: .10
2000.30	Euro: 2000.30
1.50	Euro: 1.50
15000	Lire: 15000

<http://www.skenz.it/traduttori>

Esercitazione 1

16

Risoluzione delle ambiguità lessicali

- Esistono due tipi di ambiguità lessicali:
 - la parte iniziale di una sequenza di caratteri riconosciuta da un'espressione regolare è riconosciuta anche da una seconda espressione regolare.
 - La stessa sequenza di caratteri è riconosciuta da due espressioni regolari distinte.
- Nel primo caso verrà eseguita l'azione associata all'espressione regolare che ha riconosciuto la sequenza più lunga.
- Nel secondo caso sarà eseguita l'azione associata all'espressione regolare dichiarata per prima nel file sorgente di JFlex.

<http://www.skenz.it/traduttori>

Esercitazione 1

17

Esempio

- Dato il file

```
%%
for { System.out.println("FOR_CMD"); }
format { System.out.println("FORMAT_CMD"); }
[a-z]+ { System.out.println("GENERIC_ID"); }
```
- Data la stringa di ingresso "format", viene stampato a video il valore FORMAT_CMD,
 - preferendo la seconda regola alla prima, perché descrive una sequenza più lunga
 - preferendo la seconda regola alla terza, perché definita prima nel file sorgente

<http://www.skenz.it/traduttori>

Esercitazione 1

18

Risoluzione delle ambiguità lessicali

- Date le regole di risoluzione dell'ambiguità, è necessario definire prima le regole per le parole chiave e poi quelle per gli identificatori.
- Il principio di preferenza per le corrispondenze più lunghe può essere pericoloso:

```
'.*' { System.out.println( "QUOTED_STRING" ); }
```

cerca di riconoscere il secondo apice il più lontano possibile: così, dato il seguente ingresso

```
'first' quoted string here, 'second' here
```

riconoscerà 36 caratteri invece di 7.

- Una regola migliore è la seguente:

```
'[^\\n\\r]+' { System.out.println( "QUOTED_STRING" ); }
```

<http://www.skenz.it/traduttori>

Esercitazione I

19

Dipendenza dal contesto

- Può essere necessario limitare la validità di un'espressione regolare all'interno di un determinato contesto.
- Esistono meccanismi diversi per specificare la dipendenza dal contesto destro (ciò che segue la sequenza di caratteri che si sta riconoscendo) rispetto alla dipendenza dal contesto sinistro (ciò che la precede).
- Fa eccezione la gestione del contesto di inizio e fine riga.

<http://www.skenz.it/traduttori>

Esercitazione I

20

Contesto di inizio e fine riga

- Il carattere '^' all'inizio di un'espressione regolare indica che la sequenza descritta deve essere posta all'inizio di riga.
- Ciò significa che o si è posizionati all'inizio del file di ingresso o che l'ultimo carattere letto è stato un carattere di fine riga.
- Il carattere '\$' al termine di un'espressione regolare indica che la sequenza descritta deve essere seguita da un carattere di fine riga.
- Tale carattere non viene incluso nella sequenza riconosciuta, deve essere riconosciuto da un'altra regola.
 - end\$ I caratteri 'e' 'n' 'd' posti alla fine della riga
 - \\r | \\n | \\n Lo aggiungo per riconoscere il carattere di fine riga

<http://www.skenz.it/traduttori>

Esercitazione I

21

Dipendenza dal contesto destro

- L'operatore binario '/' separa un'espressione regolare dal suo contesto destro.
- Pertanto, l'espressione

$$ab / cd$$

indica la stringa "ab", ma solo se seguita da "cd".

- I caratteri che formano il contesto destro vengono letti dal file di ingresso ma non introdotti nel testo riconosciuto. A tale scopo viene utilizzato un apposito buffer fornito da JFlex.
- L'espressione ab\$ è equivalente a ab / (\\n | \\r | \\n).

<http://www.skenz.it/traduttori>

Esercitazione I

22

Dipendenza dal contesto sinistro

- È utile poter avere diversi insiemi di regole lessicali da applicare in porzioni diverse del file di ingresso, in genere in funzione di ciò che precede, ovvero del contesto sinistro.
- Esistono tre approcci distinti per affrontare il problema:
 - uso di condizioni iniziali sulle regole o stati.
 - uso congiunto di più analizzatori lessicali.

<http://www.skenz.it/traduttori>

Esercitazione I

23

Uso di condizioni iniziali sulle regole (stati inclusivi)

- Le regole la cui espressione regolare inizia con

$$\langle state \rangle$$

sono attive solo quando l'analizzatore si trova nello stato *state*.

- Gli stati possibili devono essere definiti nella sezione delle dichiarazioni mediante la parola-chiave %state.
- Lo stato di default è lo stato YYINITIAL.
- Si entra in uno stato quando viene eseguita l'azione

$$yybegin(state);$$

<http://www.skenz.it/traduttori>

Esercitazione I

24

Uso di condizioni iniziali sulle regole (stati inclusivi) continua

- Quando uno stato viene attivato, le regole dello stato si aggiungono (or-inclusivo) alle regole base dell'analizzatore.
- Tale stato rimane attivo fino a che non se ne attiva un altro. Per tornare alla situazione iniziale si deve eseguire il comando

```
yybegin(YYINITIAL);
```

- Una regola può essere preceduta dal nome di più stati, separati da virgola, per indicare che tale regola è attiva in ciascuno di essi.

<http://www.skenz.it/traduttori>

Esercitazione I

25

Esempio

- Il seguente programma gestisce pseudo-commenti del tipo `// $var+`

```
%%
%state comment
%%
<comment>\${a-zA-Z}+[-+] {process(Yylex.yytext());}
"/" {yybegin(comment);}
\n|\r|\r\n {yybegin(YYINITIAL);}
" " /* ignora spazi bianche*/
\t /* e tabulazioni */
... Altre regole
```

<http://www.skenz.it/traduttori>

Esercitazione I

26

Uso congiunto di più analizzatori lessicali (stati esclusivi)

- È possibile raggruppare un insieme di regole all'interno di uno stato esclusivo.
- Quando l'analizzatore si trova in uno stato esclusivo:
 - le regole di default risultano disabilitate,
 - sono attive solo le regole esplicitamente attivate nello stato.
- In tal modo è possibile specificare dei "mini-analizzatori" che esaminano porzioni particolari del flusso di caratteri in ingresso, quali ad esempio i commenti o le stringhe.
- La parola chiave `%xstate` introduce uno stato esclusivo.

<http://www.skenz.it/traduttori>

Esercitazione I

27

Eliminazione dei commenti

- Questo è un analizzatore che riconosce e scarta commenti del linguaggio C, mantenendo un conteggio del numero della riga di ingresso.

```
%%
public int line_num = 1;
public int line_num_comment = 1;
%xstate comment
nl = \n | \r | \r\n
%%
\n { ++line_num; }
"/" { yybegin(comment); }
<comment>[^{nl}]* {;}
<comment>"+[*/]{nl}*" {;}
<comment>{nl} { ++line_num_comment; }
<comment>"+*/" { yybegin(YYINITIAL); }
... altre regole
```

<http://www.skenz.it/traduttori>

Esercitazione I

28

Regole di fine file

- La regola speciale `<<EOF>>` introduce le azioni da intraprendere alla fine del file.
- Questa regola può essere utile unitamente alle *start condition* per intercettare simboli con delimitatori non bilanciati:

```
\" { yybegin(quote); }
...
<quote><<EOF>> { System.out.println("EOF in
string"); }
```

<http://www.skenz.it/traduttori>

Esercitazione I

29

Compilazione di un programma JFlex

euroLire.jflex :

```
%%
%class Lexer
%standalone

euro = [1-9][0-9]*"."[0-9][0-9] | 0?."[0-9][0-9]
lire = [1-9][0-9]*

{euro} { System.out.println("Euro: " + yytext()); }
{lire} { System.out.println("Lire: " + yytext()); }
```

- `%standalone`: permette di creare un metodo main dentro il file generato
 - Il metodo main si aspetta una lista di file su cui applicare lo scanner
- `%class Lexer`: il file generato avrà il nome `Lexer.java` e conterrà la classe `Lexer`

<http://www.skenz.it/traduttori>

Esercitazione I

30

Compilazione di un programma JFlex



- Per eseguire la compilazione bisogna compiere i seguenti passi:
`jflex euroLire.jflex`
`javac Lexer.java`
`java Lexer <nome_file_1> ... <nome_file_n>`

Analizzatore lessicale generato

- *Jflex* genera una classe Java che implementa lo scanner. Il suo costruttore richiede come input un oggetto di tipo *java.io.Reader*, attraverso il quale il file di ingresso viene letto.
- *yylex()* è il metodo che si occupa dello scanning. Esso scandisce il file di ingresso:
 - Eseguendo delle azioni sulle espressioni regolari riconosciute
 - Stampando a schermo il testo non riconosciuto
- Nel caso in cui si utilizzi la direttiva *%cup* al fine di integrare lo scanner *jflex* con il parser *cup* il metodo *next_token()* è utilizzato per avviare lo scanner
 - Tale metodo sarà richiamato direttamente dal parser *cup*

Switching di file

- In molte situazioni uno scanner deve sospendere l'analisi del file corrente e aprire un altro:
 - Ad Es. direttiva `#include` nel C
- *Jflex* gestisce questo meccanismo con una strategia a stack e fornisce una serie di primitive per farlo:
 - Se si vuole analizzare un altro file si fa un operazione di push del file appena utilizzato
 - Una volta che *Jflex* è giunto alla fine del file si fa un'operazione di pop, recuperando dallo stack i file la cui analisi non è ancora finita

Switching di file continua

- `void ypushStream(java.io.Reader reader)`
 - Colloca lo stream corrente nello stack e legge dal nuovo stream.
- `void ypopStream(void)`
 - Chiude lo stream di input corrente e continua la lettura dal primo stream in cima allo stack
- `boolean ymoreStreams()`
 - Restituisce TRUE se ci sono ancora degli stream nello stack

```

Esempio:
#include {FILE} {
    ypushStream(new FileReader(getFile(yytext())));
}
...
<<EOF>> {if(ymoreStreams())yypopStream();else return EOF;}
    
```

Inclusione di file

- Esempio di analizzatore che elabora file inclusi con annidamento.

```

import java.io.FileReader;          <INCL>[^\n\r]* { /* Metto il file nello stack */
                                   ypushStream(new FileReader(yytext()));
                                   yybegin(YYINITIAL);
}%%
}

%state INCL DELETENR
%standalone                       <DELETENR>[^\n\r]* {yybegin(YYINITIAL);}

%%

include {yybegin(INCL);}          <<EOF>> { /* Prelevo il file dallo stack */
.+ {System.out.print(yytext());}  yypopStream();
                                   yybegin(DELETENR);
                                   }else return YYEOF;
/* Elimina spazi bianchi e tabulazioni */
<INCL>[^\n\r]* {} }
    
```

Copyright

Copyright (c) 2000, MarcoTorchiano (torchiano@polito.it).
 Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts.

A copy of the license can be found at the URL:
<http://www.gnu.org/copyleft/fdl.html>.

Jflex and Cup Adaptation by Stefano Scanzio
 (stefano.scanzio@polito.it)