3. Analisi sintattica

Implementazione in Java di un analizzatore sintattico a discesa ricorsiva per espressioni di un linguaggio di programmazione semplice

Esercizio 3.2

- Si consideri una grammatica per un semplice linguaggio di programmazione, dove i terminali della grammatica corrispondono ai token descritti in Sezione 2 , Tabella 1

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Case	`case`	259
When	`when`	260
Then	`then`	261
Else	`else`	262
While	`while`	263
Do	`do`	264
Assegnamento	`:=`	265
Print	`print`	266
Read	`read`	267
Disgiunzione	`||`	268
Congiunzione	`&&`	269
Negazione	`!`	33
Parentesi tonda sinistra	`(`	40
Parentesi tonda destra	`)`	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	`+`	43
Sottrazione	`-`	45
Moltiplicazione	`*`	42
Divisione	`/`	47
Punto e virgola	`;`	59
EOF	Fine dell'input	-1

Tabella 1: Descrizione dei token del linguaggio

Esercizio 3.2

$$
\begin{array}{rcl}
\langle prog \rangle & ::= & \langle statlist \rangle \ \texttt{EOF} \\
\langle statlist \rangle & ::= & \langle stat \rangle \ \langle statlistp \rangle \\
\langle statlistp \rangle & ::= & \texttt{;} \ \langle stat \rangle \ \langle statlistp \rangle \mid \varepsilon \\
\langle stat \rangle & ::= & \texttt{ID} := \langle expr \rangle \\
 & \mid & \texttt{print} \ (\ \langle expr \rangle\ ) \\
 & \mid & \texttt{read} \ (\ \texttt{ID}\ ) \\
 & \mid & \texttt{case} \ \langle whenlist \rangle \ \texttt{else} \ \langle stat \rangle \\
 & \mid & \texttt{while} \ (\ \langle bexpr \rangle\ ) \ \langle stat \rangle \\
 & \mid & \{\ \langle statlist \rangle\ \} \\
\langle whenlist \rangle & ::= & \langle whenitem \rangle \ \langle whenlistp \rangle \\
\langle whenlistp \rangle & ::= & \langle whenitem \rangle \ \langle whenlistp \rangle \mid \varepsilon \\
\langle whenitem \rangle & ::= & \texttt{when} \ (\ \langle bexpr \rangle\ ) \ \langle stat \rangle \\
\langle bexpr \rangle & ::= & \langle expr \rangle \ \texttt{RELOP} \ \langle expr \rangle \\
\langle expr \rangle & ::= & \langle term \rangle \ \langle exprp \rangle \\
\langle exprp \rangle & ::= & + \ \langle term \rangle \ \langle exprp \rangle \mid - \ \langle term \rangle \ \langle exprp \rangle \mid \varepsilon \\
\langle term \rangle & ::= & \langle fact \rangle \ \langle termp \rangle \\
\langle termp \rangle & ::= & * \ \langle fact \rangle \ \langle termp \rangle \mid / \ \langle fact \rangle \ \langle termp \rangle \mid \varepsilon \\
\langle fact \rangle & ::= & (\ \langle expr \rangle\ ) \mid \texttt{NUM} \mid \texttt{ID}
\end{array}
$$

- Si noti che RELOP corrisponde a un elemento dell'insieme {==; <>; <=; >=; <; >}
- Scrivere un analizzatore sintattico a discesa ricorsiva per la grammatica.
- Espressioni aritmetiche: ora possono anche comprendere ID (semplice estendere 3.1)
- Espressioni aritmetiche: compaiono nelle espressioni booleane, nell'assegnamento, etc. -> impatto su insieme guida

Approccio

- Lo stesso di 3.1 (da estendere opportunamente al linguaggio più ricco)
- Seguite lo stesso schema di implementazione suggerito a LFT teoria per grammatiche LL(1) e parsificazione deterministica look ahead
- Partite dall'insieme guida
- Nota bene: alcuni token di Tabella 1 non vengono usati (poco male). Quali?
  - Cosa avremmo potuto farcene?

Esercizio 3.2

$$
\begin{array}{rcl}
\langle prog \rangle & ::= & \langle statlist \rangle \ \texttt{EOF} \\
\langle statlist \rangle & ::= & \langle stat \rangle \ \langle statlistp \rangle \\
\langle statlistp \rangle & ::= & \texttt{;} \ \langle stat \rangle \ \langle statlistp \rangle \mid \varepsilon \\
\langle stat \rangle & ::= & \texttt{ID} := \langle expr \rangle \\
 & \mid & \texttt{print} \ (\ \langle expr \rangle \ ) \\
 & \mid & \texttt{read} \ (\ \texttt{ID} \ ) \\
 & \mid & \texttt{case} \ \langle whenlist \rangle \ \texttt{else} \ \langle stat \rangle \\
 & \mid & \texttt{while} \ (\ \langle bexpr \rangle \ ) \ \langle stat \rangle \\
 & \mid & \{ \ \langle statlist \rangle \ \} \\
\langle whenlist \rangle & ::= & \langle whenitem \rangle \ \langle whenlistp \rangle \\
\langle whenlistp \rangle & ::= & \langle whenitem \rangle \ \langle whenlistp \rangle \mid \varepsilon \\
\langle whenitem \rangle & ::= & \texttt{when} \ (\ \langle bexpr \rangle \ ) \ \langle stat \rangle \\
\langle bexpr \rangle & ::= & \langle expr \rangle \ \texttt{RELOP} \ \langle expr \rangle \\
\langle expr \rangle & ::= & \langle term \rangle \ \langle exprp \rangle \\
\langle exprp \rangle & ::= & + \ \langle term \rangle \ \langle exprp \rangle \mid - \ \langle term \rangle \ \langle exprp \rangle \mid \varepsilon \\
\langle term \rangle & ::= & \langle fact \rangle \ \langle termp \rangle \\
\langle termp \rangle & ::= & * \ \langle fact \rangle \ \langle termp \rangle \mid / \ \langle fact \rangle \ \langle termp \rangle \mid \varepsilon \\
\langle fact \rangle & ::= & (\ \langle expr \rangle \ ) \mid \texttt{NUM} \mid \texttt{ID}
\end{array}
$$

- La grammatica e' LL(1)