Esercizio 3.2. Seguono le produzioni di una grammatica per un semplice linguaggio di programmazione. Come nell'Esercizio 3.1, le variabili sono denotate con le parentesi angolari (per esempio, ⟨*prog*⟩, ⟨*statlist*⟩, ⟨*statlistp*⟩, ecc.). I terminali della grammatica corrispondono ai token descritti in Sezione 2 (in Tabella 1).

$$
\begin{array}{rcl}
\langle prog \rangle & ::= & \langle statlist \rangle \ \texttt{EOF} \\
\\
\langle statlist \rangle & ::= & \langle stat \rangle \ \langle statlistp \rangle \\
\\
\langle statlistp \rangle & ::= & \texttt{;} \ \langle stat \rangle \ \langle statlistp \rangle \ \mid \ \varepsilon \\
\\
\langle stat \rangle & ::= & \texttt{assign} \ \langle expr \rangle \ \texttt{to} \ \langle idlist \rangle \\
& \mid & \texttt{print (} \ \langle exprlist \rangle \ \texttt{)} \\
& \mid & \texttt{read (} \ \langle idlist \rangle \ \texttt{)} \\
& \mid & \texttt{while (} \ \langle bexpr \rangle \ \texttt{)} \ \langle stat \rangle \\
& \mid & \texttt{if (} \ \langle bexpr \rangle \ \texttt{)} \ \langle stat \rangle \ \texttt{end} \\
& \mid & \texttt{if (} \ \langle bexpr \rangle \ \texttt{)} \ \langle stat \rangle \ \texttt{else} \ \langle stat \rangle \ \texttt{end} \\
& \mid & \{ \ \langle statlist \rangle \ \} \\
\\
\langle idlist \rangle & ::= & \texttt{ID} \ \langle idlistp \rangle \\
\\
\langle idlistp \rangle & ::= & \texttt{,} \ \texttt{ID} \ \langle idlistp \rangle \ \mid \ \varepsilon \\
\\
\langle bexpr \rangle & ::= & \texttt{RELOP} \ \langle expr \rangle \ \langle expr \rangle \\
\\
\langle expr \rangle & ::= & \texttt{+ (} \ \langle exprlist \rangle \ \texttt{)} \ \mid \ \texttt{-} \ \langle expr \rangle \ \langle expr \rangle \\
& \mid & \texttt{* (} \ \langle exprlist \rangle \ \texttt{)} \ \mid \ \texttt{/} \ \langle expr \rangle \ \langle expr \rangle \\
& \mid & \texttt{NUM} \ \mid \ \texttt{ID} \\
\\
\langle exprlist \rangle & ::= & \langle expr \rangle \ \langle exprlistp \rangle \\
\\
\langle exprlistp \rangle & ::= & \texttt{,} \ \langle expr \rangle \ \langle exprlistp \rangle \ \mid \ \varepsilon
\end{array}
$$

Si noti che RELOP corrisponde a un elemento dell'insieme {==, <>, <=, >=, <, >}, NUM corrisponde a una costante numerica e ID corrisponde a un identificatore. Inoltre, si noti che le espressioni aritmetiche sono scritte in *notazione prefissa* o polacca, diversamente da quanto accadeva nell'esercizio precedente dove venivano scritte secondo la notazione infissa (standard). Analogamente le espressioni booleane sono scritte in notazione prefissa, seguendo la convenzione di porre l'operatore relazionale a sinistra delle espressioni. Modificare la grammatica per ottenere una grammatica LL(1) equivalente, e scrivere un analizzatore sintattico a discesa ricorsiva per la grammatica ottenuta.

4 Traduzione diretta dalla sintassi

Esercizio 4.1 (Valutatore di espressioni semplici)**.** Modificare l'analizzatore sintattico dell'esercizio 3.1 in modo da valutare le espressioni aritmetiche semplici, facendo riferimento allo schema

di traduzione diretto dalla sintassi seguente:

$$
\begin{array}{rcl}
\langle start \rangle & ::= & \langle expr \rangle \ \texttt{EOF} \ \{ \ print(expr.val) \ \} \\
\\
\langle expr \rangle & ::= & \langle term \rangle \ \{ \ exprp.i = term.val \ \} \ \langle exprp \rangle \ \{ \ expr.val = exprp.val \ \} \\
\\
\langle exprp \rangle & ::= & + \ \langle term \rangle \ \{ \ exprp_1.i = exprp.i + term.val \ \} \ \langle exprp_1 \rangle \ \{ \ exprp.val = exprp_1.val \ \} \\
& | & - \ \langle term \rangle \ \{ \ exprp_1.i = exprp.i - term.val \ \} \ \langle exprp_1 \rangle \ \{ \ exprp.val = exprp_1.val \ \} \\
& | & \varepsilon \ \{ \ exprp.val = exprp.i \ \} \\
\\
\langle term \rangle & ::= & \langle fact \rangle \ \{ \ termp.i = fact.val \ \} \ \langle termp \rangle \ \{ \ term.val = termp.val \ \} \\
\\
\langle termp \rangle & ::= & * \ \langle fact \rangle \ \{ \ termp_1.i = termp.i * fact.val \ \} \ \langle termp_1 \rangle \ \{ \ termp.val = termp_1.val \ \} \\
& | & / \ \langle fact \rangle \ \{ \ termp_1.i = termp.i / fact.val \ \} \ \langle termp_1 \rangle \ \{ \ termp.val = termp_1.val \ \} \\
& | & \varepsilon \ \{ \ termp.val = termp.i \ \} \\
\\
\langle fact \rangle & ::= & \texttt{(} \ \langle expr \rangle \ \texttt{)} \ \{ \ fact.val = expr.val \ \} \mid \texttt{NUM} \ \{ \ fact.val = \texttt{NUM}.value \ \}
\end{array}
$$

Si noti che il terminale `NUM` ha l'attributo *value*, che è il valore numerico del terminale, fornito dall'analizzatore lessicale.

Una possibile struttura del programma è la seguente.

Nota: come indicato, è fortemente consigliata la creazione di una nuova classe (chiamata `Valutatore` in Listing 8).

Listing 8: Valutazione di espressioni semplici

```
import java.io.*;

public class Valutatore {
    private Lexer lex;
    private BufferedReader pbr;
    private Token look;

    public Valutatore(Lexer l, BufferedReader br) {
        lex = l;
        pbr = br;
        move();
    }

    void move() {
        // come in Esercizio 3.1
    }

    void error(String s) {
        // come in Esercizio 3.1
    }

    void match(int t) {
        // come in Esercizio 3.1
    }


    public void start() {
        int expr_val;

        // ... completare ...

        expr_val = expr();
```

```
        match(Tag.EOF);

        System.out.println(expr_val);

        // ... completare ...
    }

    private int expr() {
        int term_val, exprp_val;

        // ... completare ...

        term_val = term();
        exprp_val = exprp(term_val);

        // ... completare ...
        return exprp_val;
    }

    private int exprp(int exprp_i) {
        int term_val, exprp_val;
        switch (look.tag) {
        case '+':
                match('+');
                term_val = term();
                exprp_val = exprp(exprp_i + term_val);
                break;

        // ... completare ...
        }
    }

    private int term() {
        // ... completare ...
    }

    private int termp(int termp_i) {
        // ... completare ...
    }

    private int fact() {
        // ... completare ...
    }

    public static void main(String[] args) {
        Lexer lex = new Lexer();
        String path = "...path..."; // il percorso del file da leggere
        try {
            BufferedReader br = new BufferedReader(new FileReader(path));
            Valutatore valutatore = new Valutatore(lex, br);
            valutatore.start();
            br.close();
        } catch (IOException e) {e.printStackTrace();}
    }
}
```

Riferimenti bibliografici

[1] Aho, Alfred V., Lam, Monica S., Sethi, Ravi, and Ullman, Jeffrey D. Compilatori: Principi, tecniche e strumenti. *Pearson Paravia Bruno Mondadori S.p.A.*, 2009.