Istruzioni per l'utilizzo di Antlr

Sia data la seguente grammatica:

S → aSb

S → ab

Definizione delle regole lessicali e sintattiche

1. Creare un file `MyGrammar.g4` dove definire le regole per l'analisi lessicale e per l'analisi sintattica
    - Nella prima riga del file viene definito il nome della grammatica. Il nome del file .g4 e quello definito al suo interno devono coincidere: `grammar MyGrammar; ...`
    - **Regole per il Parser**: si tratta di riscrivere la grammatica nella forma EBNF (Extended Backus–Naur form):

      ```
      rule: prod_1 | prod_2 | ... | prod_n;
      ```

      Queste regole vanno definite sempre prima delle regole per il Lexer. Per gestire il problema dei prefissi, aggiungere un regola iniziale che produce l'assioma della nostra grammatica seguito dal carattere di End-of-File (EOF). Nel nostro caso:

      ```
      ...
      // Parser rules (questo è un commento)
      parse_all:
          srule EOF;

      srule
          : 'a' srule 'b'
          | 'a' 'b'
      ;
      ...
      ```

      Le parti sinistre della grammatica vengono solitamente scritte in minuscolo e possono avere qualsiasi nome. In questo caso **srule** corrisponde all'assioma della nostra grammatica.

      Il nome **parse_all** è quello che viene dato di norma (ma è anch'esso arbitrario) alla prima regola della grammatica, ed è anche colui che darà il nome alla funzione per effettuare il parsing (vedi sezione 2 -> "Creare un oggetto Parser").

      Si noti infine che i token 'a' e 'b' sono stati definiti direttamente all'interno delle regole per il parser. Questo è possibile quando abbiamo a che fare con singoli caratteri. Per token più complessi è necessario definire anche le regole per il lexer.
    - **Regole per il Lexer**: definizione dei token. Queste regole vanno inserite sempre dopo quelle per il parser. Token più complessi possono essere definiti tramite espressioni regolari. Se ad esempio al posto del carattere 'a' volessimo un numero (intero) qualsiasi allora dovremmo definire un token in questo modo: `NUM: [0-9]+;` , dove *NUM* è il token che va sostituito ad 'a' nelle regole per il parser (es: `NUM srule 'b'`). Inoltre è necessario definire anche un token che equivalga a tutto quello che il Lexer deve segnalare come "token sconosciuto":

      ```
      ...
      // Lexer rules
      UNKNOWN_KEYWORD: .;
      ```

      il '.' sta ad indicare "tutto tranne i token già definiti". **NOTA**: Ai token può essere assegnato qualsiasi nominativo.

2. Creare un file `Main.java` . Qui potremo analizzare i risultati prodotti dal Lexer e dal Parser e manipolarli per i nostri scopi (es: stampare i token analizzati, stampare l'albero sintattico, stampare messaggi di errore):
    - Leggere l'input da file:

      ```
      FileInputStream input_stream = new FileInputStream(args[0]);
      CharStream input = CharStreams.fromStream(input_stream);
      ```
    - Creare un oggetto Lexer:

      ```
      MyGrammarLexer lexer = new MyGrammarLexer(input);
      CommonTokenStream tokens = new CommonTokenStream(lexer);
      ```

      La classe **MyGrammarLexer** è il lexer che viene generato da Antlr, secondo le regole definite in *MyGrammar.g4*. Il nome di questa classe sarà diverso a seconda del nome che viene dato alla grammatica.
    - Stampare i token:

```
tokens.fill();
for(Token token : tokens.getTokens()){
    System.out.println(token.toString());
}
```

- Creare un oggetto Parser:

```
MyGrammarParser parser = new MyGrammarParser(tokens);
ParseTree tree = parser.parse_all();
```

La classe **MyGrammarParser** è il parser che viene generato da Antlr, secondo le regole definite in *MyGrammar.g4*. Il nome di questa classe sarà diverso a seconda del nome che viene dato alla grammatica. La classe **ParseTree** è propria di Antlr ed èdefinita nella documentazione ufficiale. Si noti il metodo ***parse_all()*** che prende, come detto in precedenza, il nome dalla prima regola definita nella grammatica.

- Controllare risultato del parsing e in caso positivo stampare il parse tree:

```
if(parser.getNumberOfSyntaxErrors() == 0){
    System.out.println("Parsing result = SUCCESS\n");
    // Print the parsing tree
    System.out.println(tree.toStringTree(parser));
}
```

Il metodo ***getNumberOfSyntaxErrors()*** è proprio di Antlr e ritorna il numero di errori sintattici riscontrati durante il processo di parsing.

ALTRI LINGUAGGI DI PROGRAMMAZIONE: Questi passi possono essere effettuati, nello stesso modo, anche negli altri linguaggi di programmazione supportati da Antlr (Java, C#, Python2|3, JavaScript, Go, C++, Swift)

Compilazione ed esecuzione

- **OPERAZIONI PRELIMINARI**:
  - Copiare il file `antlr-4.8-complete.jar` in `/usr/local/lib` e aggiungerlo al classpath, eseguendo i seguenti comandi:

```
sudo cp antlr-4.8-complete.jar /usr/local/lib/
export CLASSPATH=".:/usr/local/lib/antlr-4.8-complete.jar:$CLASSPATH"
```

  - Per semplificare l'uso del tool aggiungiamo i seguenti alias:

```
alias antlr4="java -jar /usr/local/lib/antlr-4.8-complete.jar"
alias grun="java org.antlr.v4.gui.TestRig"
```

1. Generare il Lexer ed il Parser tramite le regole appena definite: **`antlr4 MyGrammar.g4`** . Questo genererà i file:
2. Compilare i file generati: **`javac *.java`**
3. Eseguire passando come parametro di input il file a cui applicare il parsing: **`java Main input-file`**

ALTRI LINGUAGGI DI PROGRAMMAZIONE: Se vogliamo generare l'analizzatore sintattico in un linguaggio diverso da Java aggiungere al passo 1 l'opzione `-Dlanguage=your-language` ed eseguire i passi 2 e 3 secondo le procedure del linguaggio usato.