Laboratorio di Automi e Linguaggi Formali
Seconda Esercitazione

Davide Bresolin

a.a. 2020/2021

1 Introduzione

Oggi ci sono migliaia di linguaggi informatici disponibili: linguaggi di programmazione, di markup, di query, eccetera, e ne vengono pubblicati di nuovi ogni anno. Ogni informatico, ad un certo punto della sua carriera, si ritrova a dover definire un nuovo linguaggio per qualche scopo particolare. In questo tutorial vedremo come utilizzare alcuni concetti di base visti al corso di Automi e Linguaggi Formali ed il generatore di parser ANTLR v4 per creare un parser per un frammento del linguaggio di programmazione imperativa Pascal.

2 Impostazione dell'ambiente di lavoro

Il file `laboratorio_antlr.zip` contiene la grammatica che definisce i costrutti di base del Pascal, il codice del syntax checker ed un `Makefile` per compilare il codice. Per installare ANTLR v4 ed il runtime C++ su una distribuzione Linux basata su Debian come Ubuntu e simili, è sufficiente installare i pacchetti

```
g++ make antlr4 libantlr4-runtime-dev
```

Per l'installazione e l'uso di ANTLR e del runtime C++ su altre architetture si rimanda al sito ufficiale `http://www.antlr.org/`.

3 Creazione del file con la grammatica

Vediamo ora come possiamo definire la grammatica per il Pascal, partendo da un esempio di programma:

```
program valid;
var
    x: integer;
    y: integer;
begin
    x := 0;
    y := 1;
    if x <= y then
    begin
        writeln(y);
    end;
    writeln(x);
end.
```

Il programma qui sopra mostra la struttura di base del linguaggio:

- ogni programma inizia con la parola chiave `program` seguita dal nome del programma
- le variabili vanno dichiarate in anticipo nella sezione `var`
- `x` e `y` sono variabili di tipo `integer`
- il codice principale del programma inizia con `begin` e termina con `end.`
- l'istruzione di assegnamento è identificata dal simbolo `:=`
- la procedura `writeln` stampa un valore sullo schermo

- il costrutto `if`, e l'uso di `begin` e `end` per separare i blocchi di codice
- il punto e virgola separa le istruzioni

Il file `pascal.g4` contiene la grammatica che definisce la sintassi del linguaggio:

```
grammar pascal;

start     : 'program' ID ';' 'var' decl_list main_code EOF ;

decl_list : decl | decl decl_list ;
decl      : ID ':' 'integer' ';' ;

main_code : 'begin' st_list 'end' '.' ;
code_block: statement | 'begin' st_list 'end' ;
st_list   : statement ';' | statement ';' st_list  ;

statement : assign | branch | out ;

assign    : ID ':=' expr ;
out       : 'writeln' '(' expr ')' ;
branch    : 'if' relation 'then' code_block ;
expr      : NUMBER | ID ;
relation  : expr LT expr | expr LEQ expr | expr EQ expr
          | expr NEQ expr | expr GEQ expr | expr GT expr ;

EQ        : '=' ;
LT        : '<' ;
LEQ       : '<=' ;
GT        : '>' ;
GEQ       : '>=' ;
NEQ       : '<>' ;
ID        : [a-z]+ ;
NUMBER    : [0-9]+ ;
R_COMMENT : '(*' .*? '*)' -> skip ;     // .*? matches anything until the first */
C_COMMENT : '{' .*? '}' -> skip ;       // .*? matches anything until the first }
LINE_COMMENT : '//' ~[\r\n]* -> skip ;  // ~[\r\n]* matches anything but \r and \n
WS        : [ \n\t\r]+ -> skip;
ErrorChar : . ;
```

4 Generazione del Parser e del Lexer

Dopo aver creato il file con la grammatica possiamo utilizzare il comando `antlr4` per creare automaticamente il codice C++ necessario per fare il parsing dei programmi Pascal. ANTLR permette di generare codice per diversi linguaggi di target: Java, Pyhton, C++, C#, Swift e Go. In questo tutorial ci utilizzeremo il linguaggio C++. Il linguaggio target va specificato con l'opzione `-Dlanguage`:

```
dbresoli@t68:~/antlr4/pascal$ antlr4 -Dlanguage=Cpp pascal.g4
```

La sintassi dell'opzione è case-sensitive: è importante fare attenzione alla '`C`' maiuscola. In caso di errore si riceve un messaggio di errore simile al seguente.

```
error(31):  ANTLR cannot generate cpp code as of version 4.7.1
```

L'esecuzione corretta di `antlr4` crea i seguenti file:

```
pascalBaseListener.h
pascalLexer.cpp
pascalLexer.tokens
pascalParser.cpp
```

```
pascalLexer.h
pascalListener.cpp
pascalParser.h
pascalBaseListener.cpp
pascal.interp
pascalLexer.interp
pascalListener.h
pascal.tokens
```

5 Creazione di un Syntax Checker

Per testare il corretto funzionamento della grammatica di Pascal possiamo scrivere un semplice programma che esegue queste semplici operazioni:

1. legge un file con un programma scritto in Pascal
2. utilizza la classe `pascalLexer` per suddividere il file in *token*
3. utilizza la classe `pascalParser` per creare un *albero sintattico* che rappresenta la struttura del testo
4. controlla il numero di errori di sintassi che il parser ha generato nella costruzione dell'albero e lo riporta all'utente.

Il codice del programma si trova nel file `syncheck.cpp` ed è riportato qui sotto:

```
#include <iostream>
#include <fstream>
#include <string>
#include "antlr4-runtime.h"
#include "pascalLexer.h"
#include "pascalParser.h"

using namespace std;
using namespace antlr4;

int main(int argc, char* argv[]) {
    if(argc != 2) {
        cout << "Usage: syncheck filename.pas" << endl;
        return 1;
    }
    ifstream pascalFile(argv[1]);
    if(pascalFile.fail()){
        // File open error
        cout << "Error while reading file " << argv[1] << endl;
        return 1;
    }
    ANTLRInputStream input(pascalFile);
    pascalLexer lexer(&input);
    CommonTokenStream tokens(&lexer);
    pascalParser parser(&tokens);
    tree::ParseTree *tree = parser.start();
    int errors = parser.getNumberOfSyntaxErrors();
    if(errors > 0) {
        cout << errors << " syntax errors found." << endl;
        return 1;
    }
    cout << "No syntax errors found." << endl;
    return 0;
}
```

Per facilitare la compilazione dei programmi lo schema di codice contiene un `Makefile` che richiama ANTLR per generare il codice C++ a partire dalla grammatica del linguaggio, compila i diversi file `.cpp` contenuti nella cartella ed infine esegue il linking con la libreria `antlr4-runtime`. Per compilare il syntax checker è sufficiente utilizzare il comando `make`:

```
dbresoli@t68:~/pascal$ make syncheck
```

o più semplicemente

```
dbresoli@t68:~/pascal$ make
```

La compilazione di `syncheck.cpp` crea l'eseguibile `syncheck` che può essere utilizzato per controllare la sintassi dei programmi Pascal. La directory `tests` contiene alcuni programmi di esempio che si possono usare per testare il programma:

```
dbresoli@t68:~/pascal$ ./syncheck tests/valid.pas
No syntax errors found.
```

Oltre al target `syncheck` che compila il controllore sintattico, il `Makefile` mette a disposizione anche il target `make clean` che elimina gli eseguibili ed i file temporanei creati da ANTLR e dall compilatore C++.

Riferimenti

Per maggiori informazioni su ANTLR potete far riferimenti ai siti web:

- `http://www.antlr.org/` sito ufficiale di ANTLR v4
- `https://github.com/antlr/antlr4/tree/master/runtime/Cpp` repository github con il codice e la documentazione del runtime C++ per ANTLR
- `https://tomassetti.me/antlr-mega-tutorial/` un tutorial molto esteso sull'uso di ANTLR con molti esempi d'uso in Javascript, Python, Java e C#