Università degli studi di Bergamo

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

CORSO DI

Linguaggi formali e compilatori

Formal languages and compilers

(COD.CORSO:38070)

Anno accademico:	**2017/2018**
Settore scientifico-disciplinare:	**ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI**
Dipartimento:	**Ingegneria gestionale, dell'informazione e della produzione**

Titolo: **Realizzazione di uno strumento per semplificare la scrittura di documenti LaTeX[1]**

Autori:

Davide Capelli **Matr# 1025005**
Marco Vitetta **Matr# 1048711**


Relazione
1 Introduzione
1.1 Obiettivi e funzionalità
1.2 Strumenti di sviluppo
1.3 Struttura del progetto
2 Template
2.1 Il tuo primo documento
2.2 Un esempio più complesso
2.3 Parole riservate
3 Definizione di un template personalizzato
3.1 Un esempio con template personalizzato
4 Conclusioni
5 Sitografia

1 Introduzione

LaTeX[1] è un linguaggio pensato per editare documenti di testo agevolmente perché permette di utilizzare alcune configurazioni predefinite degli attributi del testo senza doverne cambiare frequentemente l'assegnamento.

Le caratteristiche del testo, a partire dall'entità del contenuto (difatti si distinguono diverse tipologie di documenti che possono contenere, a loro volta, altre entità come le sezioni, i paragrafi, i sottoparagrafi, ecc.) fino ai colori, sono specificate con l'utilizzo di marcatori (in inglese markup o tag).
Un marcatore è identificato da un backslash seguito da una sequenza di caratteri (una parola, caratteri speciali, o più) racchiusi tra parentesi graffe ed è associato ad un insieme di attributi che definiscono la struttura del testo e ne influenzano l'apparenza grafica.
Gli attributi possono essere applicati da un solo marcatore o da una coppia, il primo dei quali è detto di apertura ed il secondo di chiusura.

La struttura descritta rende LaTeX[1] molto versatile ed apprezzato nell'ambito tecnico (specialmente in quello accademico) ma può essere anche difficile per gli utenti abituati agli editor di documenti tradizionali (basati sul paradigma WYSIWYG[2]) e novizi di LaTeX[1] (basato sul paradigma WYSIWYM[3]).

1.1 Obiettivi e funzionalità

La suddetta analisi ci ha portato a pensare un modo per semplificare la scrittura di documenti semplici in LaTeX[1], cercando di ridurre i marcatori da inserire, eliminare la necessità dei backslash e delle parentesi graffe, pur mantenendo la struttura generale del linguaggio originale.

L'obiettivo del progetto è definire un linguaggio che permetta di scrivere un documento LaTeX[1] in maniera più semplice e veloce con l'utilizzo di template predefiniti (che definiscono alcune classi di documenti standard di LaTeX[1]) o creati dall'utente.
Il documento scritto nel suddetto linguaggio potrà essere convertito in LaTeX[1] con un compilatore opportuno.

1.2 Strumenti di sviluppo

Il linguaggio di scrittura dei documenti è stato definito con ANTLR[4], uno strumento che permette di scrivere grammatiche EBNF (Extended Backus-Naur Form) e di generare automaticamente i rispettivi riconoscitori di linguaggi (parser).
Le peculiarità del linguaggio desiderato possono essere specificate all'interno della rispettiva grammatica senza che lo sviluppatore debba ritoccare il codice generato automaticamente dallo strumento perché i generatori di parser disaccoppiano la definizione del linguaggio dall'implementazione del riconoscitore.
La separazione netta tra il linguaggio definito ed il suo riconoscitore fa sì che per generare un parser è sufficiente conoscere il meta-linguaggio di definizione di ANTLR[4] ed il suo funzionamento.

ANTLR[4] permette di generare i parser in molteplici linguaggi detti target (alcuni dei quali sono C++, Java e Python) senza dover necessariamente conoscerli e questo rende lo strumento molto versatile e semplice da utilizzare pressoché in ogni contesto (applicativo, hardware e software).
Se non viene specificato il linguaggio target desiderato, ANTLR[4] genera un parser in Java (di default).

Per utilizzare ANTLR[4] è possibile scaricare gratuitamente l'eseguibile oppure installare uno dei plugin disponibili per gli IDE più diffusi tra i quali Eclipse, IntelliJ e VisualStudio.

1.3 Struttura del progetto

Il progetto è costituito dalla grammatica del linguaggio definita con ANTLR[4] ed un programma di test scritto in Java per verificare il funzionamento del compilatore generato automaticamente; entrambi i codici sono stati realizzati con Eclipse e resi disponibili al seguente indirizzo[5].

La grammatica[6] è rappresentata da un file testuale con estensione *.g4* e dev'essere inserita all'interno di un progetto ANTLR[4] creato con Eclipse.
Il programma di test[7], invece, può essere importato all'interno di un progetto Java (nel seguito chiamato *FLC - Tester proj*[7]) utilizzando l'apposito wizard di Eclipse.

Per eseguire il programma di test è sufficiente compilare la grammatica con ANTLR[4] e copiare i file generati dallo strumento all'interno della cartella *generated* del progetto *FLC - Tester proj*[7].

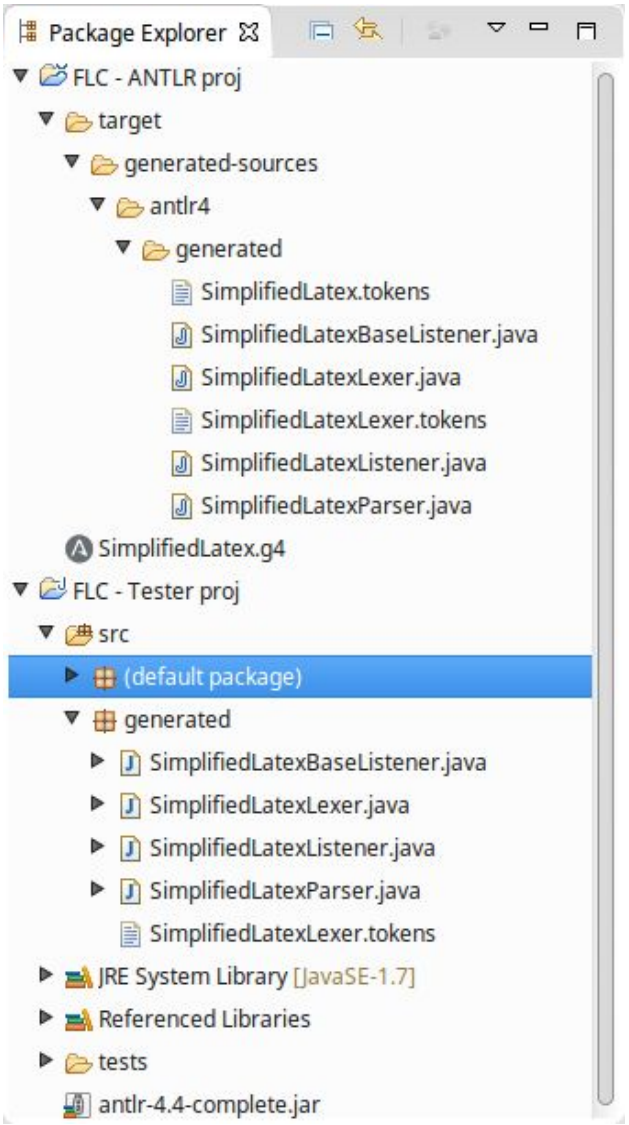

Fig. 1 - Albero completo affinché il programma di test possa essere eseguito

2 Template

Il linguaggio definito nell'ambito del progetto si fonda sul concetto di template, ossia una struttura di documento alla quale sono associate, eventualmente, una o più regole che permettono la generazione automatica di codice LaTeX[1].

Alcuni documenti standard di LaTeX[1] sono predefiniti, sotto forma di template, all'interno del linguaggio realizzato e sono: gli articoli (identificati dalla parola riservata *article*), i curriculum (identificati dalla parola riservata *cv*), le lettere (identificate dalla parola riservata *letter*) e le tesi (identificate dalla parola riservata *thesis*).

Dentro al progetto *FLC - Tester proj*[7] è anche possibile definire dei template personalizzati ed aggiungerli alla classe del compilatore (ossia *SimplifiedLatexTranslator*[8]) affinché vengano riconosciute ed applicate le regole definite al loro interno.
La specifica di un template personalizzato verrà affrontata nel dettaglio in seguito.

I paragrafi successivi affrontano gradualmente la spiegazione della sintassi del linguaggio definito attraverso due esempi di complessità crescente.

2.1 Il tuo primo documento

Il primo esempio permette di valutare la semplicità del linguaggio definito mediante la scrittura del documento più semplice che può essere realizzato.

L'esempio riportato è il primo caso di test[9] definito per verificare il funzionamento dello strumento ed è disponibile nella sottocartella *tests* all'interno del progetto *FLC - Tester proj*[7].

```
template article
     Alura world!
end
```

Listato 1 - Primo esempio

Dall'esempio si evince che la stesura di un testo semplice con il linguaggio definito richiede solo l'indicazione della tipologia di documento desiderata.

Le parole riservate utilizzate nell'esempio sono:

- *template* che apre il documento insieme al nome del template utilizzato ed è una regola sintattica del linguaggio;
- *article* che definisce il template utilizzato (i template predefiniti hanno lo stesso nome della tipologia di documento LaTeX[1]);
- *end* che indica la fine del documento.

Il codice LaTeX[1] generato dallo strumento a partire dal **Listato 1** è il seguente:

```
\documentclass{article}
\begin{document}
Alura world!

\end{document}
```

Listato 2 - Codice LaTeX[1] generato a partire dal Listato 1

2.2 Un esempio più complesso

Il secondo esempio[10] introduttivo aggiunge alcuni costrutti base del linguaggio definito, quali l'indicazione dell'autore e del titolo (entrambe opzionali e inseribili nell'ordine che si preferisce) e la possibilità di aggiungere i commenti (che integrano il documento ma non verranno mostrati una volta compilato il codice LaTeX[1] generato).

```
template  article
author    Pippo
title     Pluto
    % questo è un commento
    Alura world
end
```

Listato 3 - Secondo esempio

I costrutti del linguaggio introdotti nel secondo esempio sono:

- *author* che è seguito dal nome dell'autore (una parola alfanumerica che può contenere anche caratteri speciali) ed è una regola sintattica del linguaggio;
- *title* che è seguito dal titolo del documento (una parola alfanumerica che può contenere anche caratteri speciali) ed è una regola sintattica del linguaggio.

Il codice LaTeX[1] generato dallo strumento a partire dal **Listato 3** è il seguente:

```
\documentclass{article}
\begin{document}
\author{Pippo}
\title{Pluto}
\maketitle
% questo è un commento
    Alura world

\end{document}
```

Listato 4 - Codice LaTeX[1] generato a partire dal Listato 3

2.3 Parole riservate

La tabella seguente riassume le parole riservate del linguaggio ed il rispettivo utilizzo.

Parola riservata	Utilizzo
article	dichiara che il documento è un articolo ed è un template predefinito
author	è seguita dal nome dell'autore ed è una regola sintattica del linguaggio
cv	dichiara che il documento è un curriculum vitae ed è un template predefinito
end	termina il documento
letter	dichiara che il documento è una lettera ed è un template predefinito
rule	è seguita dal nome della regola definita dall'utente che deve essere applicata
template	è la regola sintattica che deve essere inserita per prima in qualsiasi documento (insieme al nome del template da utilizzare)
thesis	dichiara che il documento è una tesi ed è un template predefinito
title	è seguita dal titolo del documento ed è una regola sintattica del linguaggio

Tabella 1 - Parole riservate del linguaggio

3 Definizione di un template personalizzato

Lo strumento realizzato può essere adattato alle proprie esigenze con la definizione dei template personalizzati.
I template personalizzati permettono di definire le proprie regole di traduzione ed aggiungerle a quelle predefinite (ossia le regole *author* e *title*) per far sì che lo strumento generi il codice LaTeX[1] desiderato.

All'interno del progetto *FLC - Tester proj*[7] sono definite le classi Java *Template*[11] e *Rule*[12] che modellano, rispettivamente, i template e le regole definite al loro interno.

Per realizzare un template personalizzato è necessario estendere la classe *Template*, aggiungere le proprie regole di traduzione nel costruttore della classe derivata ed implementare opportunamente il metodo *translate* (secondo il criterio di analisi del parametro di ingresso *token* e di applicazione delle regole personalizzate).
Il metodo *translate* deve essere in grado di riconoscere la regola da applicare e deve richiamare i rispettivi metodi *enter* ed *exit*.

La classe *Rule*[12] permette di definire le regole personalizzate attraverso i metodi *enter* ed *exit*, da implementare, che generano la traduzione dei parametri passati alla regola (analoghi agli argomenti delle funzioni matematiche o dei linguaggi di programmazione).

3.1 Un esempio con un template personalizzato

Il compilatore *SimplifiedLatexTranslator*[8] include il template personalizzato *ladox* (classe *LadoxTemplate*[13]) che genera automaticamente la struttura tipica della documentazione dei codici sorgenti di un programma (Javadoc o Doxygen).

Il template *ladox* include solo la regola *sign* (classe *SignRule*[14]) che genera automaticamente la documentazione della funzione (o metodo) che segue la regola.

L'esempio seguente è il terzo caso di test[15] allegato al progetto.

```
template ladox
    % da notare che ho dovuto usare la parola modello...
    Questo è un modello LaDoX e, se funziona,
    genera la javadoc per il seguente metodo:
    rule sign HashMap<Funghi,int> raccogli(Bosco b)
end
```

Listato 5 - Terzo esempio

Il codice LaTeX[1] generato dallo strumento a partire dal **Listato 5** è il seguente:

```
\documentclass{ladox}
\begin{document}
% da notare che ho dovuto usare la parola modello...
    Questo è un modello LaDoX e, se funziona,
    genera la javadoc per il seguente metodo:
@param b \\
@return HashMap<Funghi,int> \\
HashMap<Funghi,int> raccogli(Bosco b)
\end{document}
```

Listato 6 - Codice LaTeX[1] generato a partire dal Listato 5

4 Conclusioni

Lo strumento realizzato permette di generare codice LaTeX[1] sulla base di alcune strutture di documenti predefinite (template) e richiede di inserire un minor numero di caratteri e marcatori.

Il compilatore può anche essere migliorato con l'aggiunta di altri marcatori tipici di LaTeX[1] (e di regole più compatte che integrino più marcatori) e con la definizione di template personalizzati che semplificano la scrittura di documenti sulla base delle proprie esigenze.

Il progetto aveva la finalità di rendere più semplice e veloce la scrittura di documenti LaTeX[1] e l'obiettivo è stato raggiunto.

5 Sitografia

[1] LaTeX - https://it.wikipedia.org/wiki/LaTeX

[2] WYSIWYG - https://it.wikipedia.org/wiki/WYSIWYG

[3] WYSIWYM - https://it.wikipedia.org/wiki/WYSIWYM

[4] ANTLR - http://www.antlr.org/

[5] Progetto completo - https://github.com/ErVito/FLC2017-18

[6] Grammatica - https://github.com/ErVito/FLC2017-18/blob/master/SimplifiedLatex.g4

[7] Progetto di test - https://github.com/ErVito/FLC2017-18/tree/master/FLC - Tester proj

[8] Compilatore - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/src/SimplifiedLatexTranslator.java

[9] Primo caso di test - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/tests/test1

[10] Secondo test - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/tests/test2

[11] *Template* - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/src/Template.java

[12] *Rule* - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/src/Rule.java

[13] *LadoxTemplate* - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/src/LadoxTemplate.java

[14] *SignRule* - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/src/SignRule.java

[15] Terzo caso di test - https://github.com/ErVito/FLC2017-18/blob/master/FLC - Tester proj/tests/test3