Progetto di Linguaggi e Compilatori 1 – Parte 2
A.A 2015-16

GRUPPO 7

Graca Sanela 114201
Markovski Vlatko 112122
Michelli Luca 109532

Esercizio 1

Punto a)
Abbiamo fatto la seguente espressione in sintassi flex per la grammatica:

Ein=((("Michelli \ Luca")($printable#[\;\,\$\|\?\#\~\{\}\(\)\^\/])+\) ((("Comini \ Marco") ($printable#[\;\,\$\|\?\#\~\{\}\(\)\^\/])+\) (("Graca \ Sanela")("Markovski \ Vlatko")?)?|("Markovski \ Vlatko")?)|(("Graca \ Sanela")($printable#[\;\,\$\|\?\#\~\{\}\(\)\^\/])+\) ("Markovski \ Vlatko")?)| ("Markovski \ Vlatko")))|(("Comini \ Marco")($printable#[\;\,\$\|\?\#\~\{\}\(\)\^\/])+\) ((("Graca \ Sanela")($printable#[\;\,\$\|\?\#\~\{\}\(\)\^\/])+\) ("Markovski \ Vlatko")?))|("Markovski \ Vlatko")))|(("Graca \ Sanela")($printable#[\;\,\$\|\?\#\~\{\}\(\)\^\/])+\) ("Markovski \ Vlatko")?)| (("Markovski \ Vlatko")))

Il problema è che non siamo riusciti a scrivere Eout per questa espressione, perché ci sarebbero troppe combinazioni da considerare. Quindi abbiamo scritto una espressione più generica che però non accetta tutte le combinazoni di quella Ein scritta sopra ({s} indica tutte le possibili stringhe):
Ein1=({s})(\")(Michelli)" "(Luca)\"{s}(", ")*\"(Comini)" "(Marco)\"{s}", "" "*\"(Graca)" "(Sanela)\"{s}", "" "\"(Markovski)" "(Vlatko)\"){s}
Eout1=1\2\7\5\2\6\5\1\2\9\5\2\8\5\1\2\11\5\2\10\5\1\2\4\5\2\3\2\1

Per le espressioni che abbiamo scritto, noi assumiamo che nella stringa in input solo i nostri nomi siano racchiusi tra doppi apici.

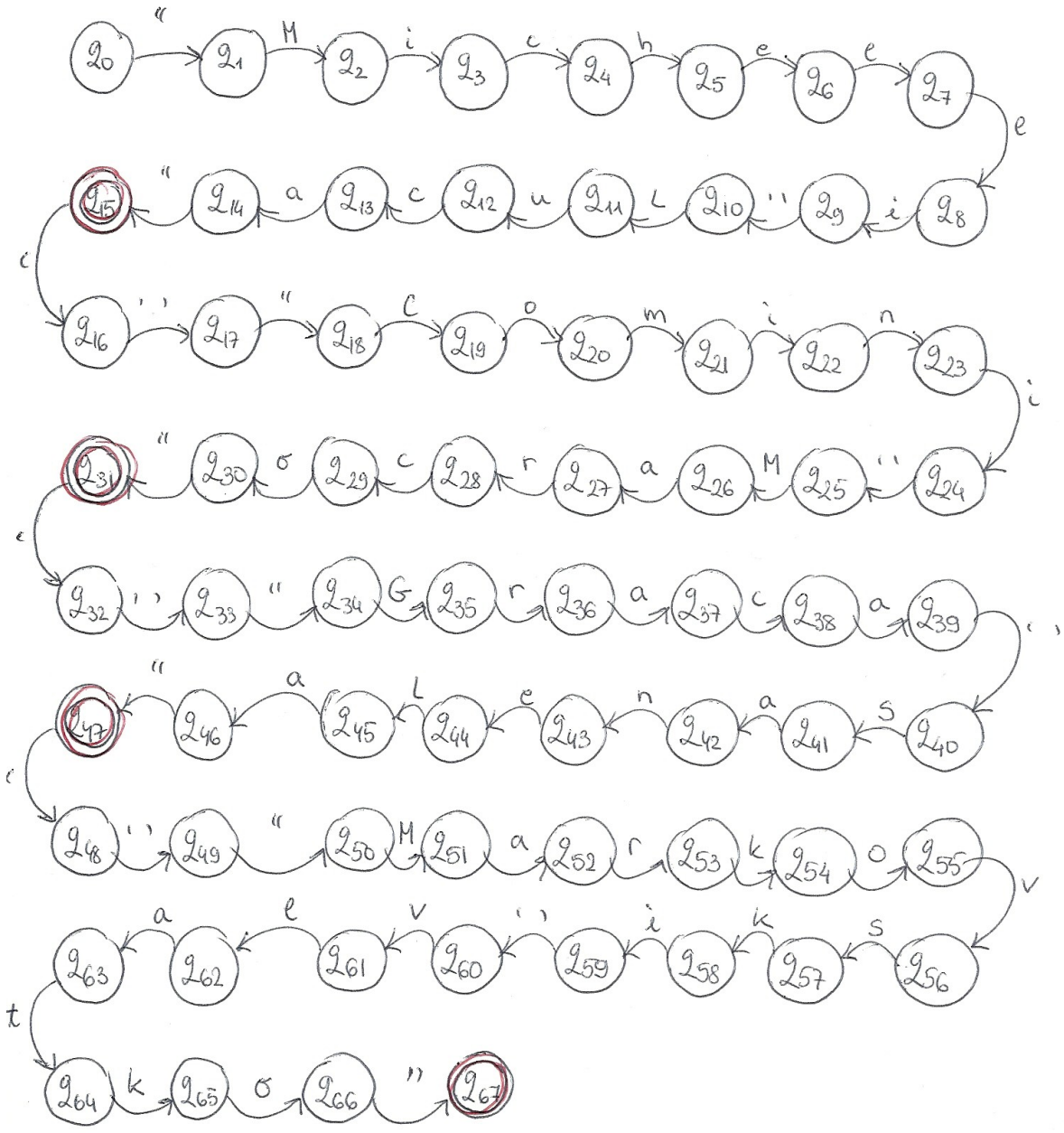
Punto b)
Noi non siamo riusciti a rappresentare tutte le possibili combinazioni di stringhe in un unico automa; quindi abbiamo disegnato tre automi (sono gli autom1 1-2-3 allegati qui sotto) che possono riconoscere le nostre stringhe in tutti i modi possibili. Le combinazioni di stringhe sono specificate nelle immagini degli automi.

Negli automi 1 e 3, Tutti i caratteri che non vengono processati (ossia non sono specificati nell'automa) finiscono in uno stato pozzo)

Affinché l'automa 3 prenda in input tutte le stringhe, anche se non compaiono simultaneamente, basta far sì che i caratteri che non vengono processati, invece di finire in uno stato pozzo finiscano nello stato iniziale della stringa che si sta leggendo (in questo caso l'automa accetta tutto).

AUTOMA DETERMINISTICO CHE ACCETTA LE SOLE STRINGHE

- "Michelli Luca"
- "Michelli Luca", "Comini Marco"
- "Michelli Luca", "Comini Marco", "Graca Samela"
- "Michelli Luca", "Comini Marco", "Graca Samela", "Markovski Vlatko"



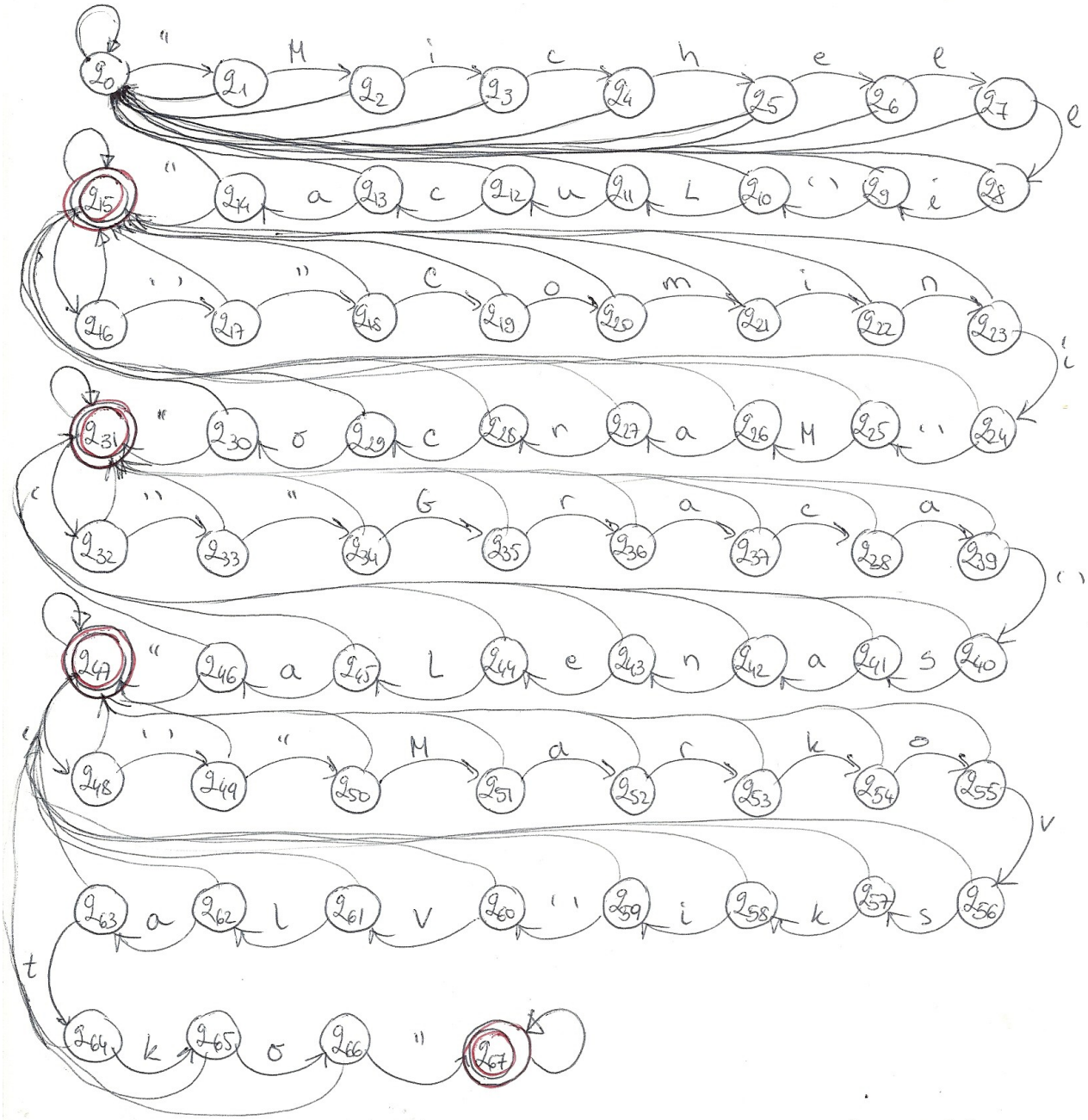
Tutti i caratteri che non vengono processati (ossia non sono specificati nell'automa) finiscono in uno stato pozzo.

AUTOMA 1

AUTOMA DETERMINISTICO CHE ACCETTA LE STRINGHE

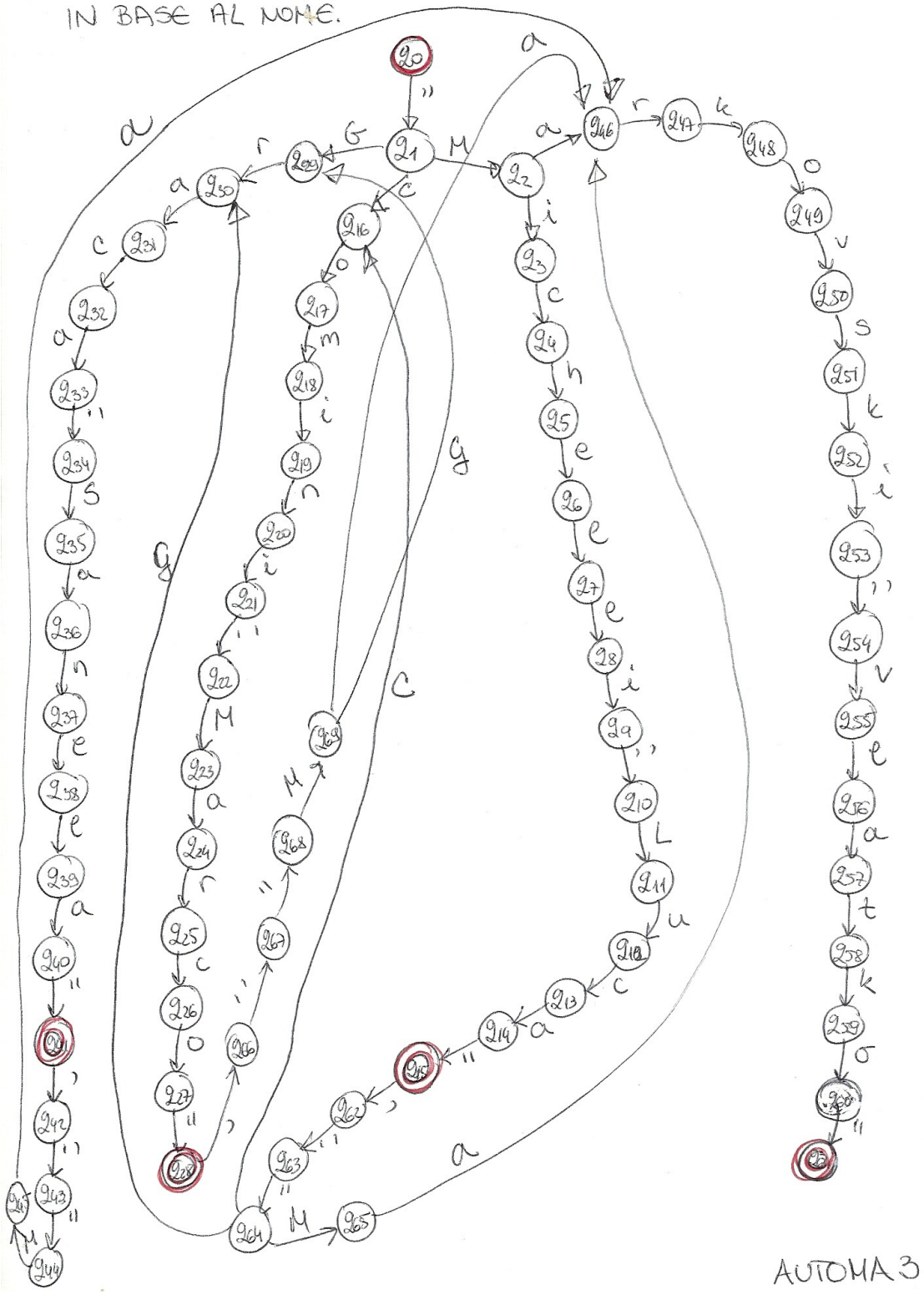
- "Michelli Luca"
- "Michelli Luca", "Comini Marco"
- "Michelli Luca", "Comini Marco", "Graca Sanela"
- "Michelli Luca", "Comini Marco", "Graca Sanela", "Markovski Vlatko"

ANCHE SE NON COMPAIONO SIMULTANEAMENTE (nel senso che possono esserci altri nomi, cognomi o qualunque altro carattere)



AUTOMA 2

AUTOMA DETERMINISTICO CHE ACCETTA I NOMI E I COGNOMI DEI CANDIDATI, OLTRE CHE MARCO COMINI.
PUÒ ACCETTARE SOLO UNA COPPIA NOME-COGNOME, MA ANCHE DUE, TRE O TUTTE E QUATTRO.
L'UNICO VINCOLO È CHE VENGANO PROCESSATI ALFABETICAMENTE IN BASE AL NOME.



AUTOMA 3

Esercizio 2

```
procedura principale:
ENT IMAX        alloco una quantita' di spazio pari a IMAX
RETP

g:
ENT 0           non ci sono parameti
RETF

ins:
ENT 0
LDA 0 5         carico indirizzo di k
IND             e il suo valore
LDC int 0
EQU int         calcolo k=0
FJP e           valuto se fare then o else
LDA 1 4         carico indirizzo di v
LDA 0 5         carico indirizzo di k
IND             e il suo valore
IXA 1           ottengo v[k]
LDA 0 4         carico indirizzo di x
IND             e il suo valore
STO             memorizzo in v[k]
UJP e1
e:
LDA 2 4         carico indirizzo di mm
LDA 0 5         carico indirizzo di k
IND             e il suo valore
LDC int 1
ADD int         calcolo k+1
IXA 1           ottengo mm[k+1]
LDA 0 4         carico indirizzo di k
IND             e il suo valore
LEQ int         calcolo mm[k+1]<=x
FJP e2          valuto se fare then o else
LDA 1 4         carico indirizzo di v
LDA 0 5         carico indirizzo di k
IND             e il suo valore
IXA 1           ottengo v[k]
LDA 0 4         carico indirizzo di x
IND             e il suo valore
STO             memorizzo in v[k]
UJP e1
e2:
LDA 1 4         carico indirizzo di v
LDA 0 5         carico indirizzo di k
IND             e il suo valore
IXA 1           ottengo v[k]
LDA 1 4         carico indirizzo di v
LDA 0 5         carico indirizzo di k
LDC int 1
```

```
SUB int         calcolo k-1
IXA 1           ottengp v[k-1]
STO             memorizzo su v[k]
MST 1           praparo la chiamata a ins
LDA 0 4         carico indirizzo di x
IND             e il suo valore
LDC int 2
SUB int         calcolo x-2
LDA 0 5         carico indirizzo di k
IND             il suo valore
LDC int 1
SUB int         calcolo k-1
CUP 2 ins       chiamo ins(X-2, K-1)
e1:
RETP

sort:
ENT 1           alloco spazio 1
LDA 0 7         carico indirizzo di temp(serve per trasformare il for in while)
MST 1           preparo la chiamata a g
LDA 0 5         carico indirizzo di n
IND             e il suo valore
LDC int 5
DIV int         faccio n/5
CUP 1 g         chiamo g(n/5)
STO             memorizzo in temp
LDA 0 6         carico indirizzo di i
LDC int 1
STO             memorizzo in i
UJP guard
body:
MST 1 ins       preparo chiamata a ins
LDA 0 4         carico indirizzo di v
LDA 0 6         carico indirizzo di i
IND             e il suo valore
IXA 1           ottengo v[1]
LDC int 3
SUB int         calcolo v[1]-3
LDA 0 6         carico indirizzo d i
IND             e salvo il suo valore
CUP 2 ins       chiamo ins(v[1]-3,i)
LDA 0 6         carico indirizzo di i
LDA 0 6         carico indirizzo di i
IND             e il suo valore
LDC int 1
ADD int         calcolo i+1
STO             e lo memorizzo in i
guard:
LDA 0 6         carico indirizzo di i
IND             e il suo valore
LDA 0 7         carico indirizzo di temp
IND             e il so valore
```

```
GTR int         faccio i>temp (la guardia va negata)
FJP body
RETP

alt:
ENT 0           non ci sono parametri
LDA 0 4         carico indirizzo di i
IND             e il suo valore
LDC int 0
EQU int         calcolo i=0
FJP e           valuto se fare then o else
LDA 0 0         carico indirizzo di ritorno di alt
LDA 0 5         carico indirizzo di x
IND             e il suo valore
STO             ritorno il risultato
UJP e1          valuto se fare then o else
e:
LDA 0 4         carico indirizzo di i
ODD int         calcolo odd(i)
FJP e2
LDA 0 0         carico indirizzo del valore di ritorno
MST 1           praparo la chiamata ricorsiva
LDA 0 4         carico indirizzo di i
IND             e il suo valore
LDC int i
SUB int         calcolo i-1
MST 1           preparo chiamata a g
LDA 0 5         carico indirizzo di x
IND             e il suo valore
CUP 1 f         chiamo f(x)
CUP 2 alt       calcolo alt(i-1,f(x))
STO             ritorno il risultato
UJP e1
e2:
LDA 0 0         carico indirizzo del valore di ritorno
MST 1           preparo la chiamata ricorsiva
LDA 0 4         carico indirizzo di i
IND             e il suo valore
LDC int i
SUB int         calcolo i-1
MST 1           preparo chiamata a g
LDA 0 5         carico indirizzo di x
IND             e il suo valore
CUP 1 g         chiamo g(x)
CUP 2 alt       chiamo alt(i-1,g(x))
STO             ritorno il risultato
e1:
RETF

f:
ENT 0           non ci sono parametri
LDA 0 0         carico indirizzo del valore di ritorno
```

```
LDC int 1
LDC int 1
LDA 0 4          carico indirizzo di x
IND              e il suo valore
DIV int          calcolo 1/x
ADD int          calcolo 1+1/x
STO              ritorno il risultato
RETF
```

Esercizio 3

Noi abbiamo usato i tool *flex* e *bison* per generare la grammatica richiesta. Abbiamo progettato un semplice *lexer* che si occupa di riconoscere le varie componenti delle sezioni (come sono definite nell'esercizio) e restituisce i token corrispondenti al Parser.

Il parser costruisce una struttura dati che conterrà i valori associati a ogni variabile. Abbiamo implementato questa struttura con liste contenenti etichette-valori. In questa struttura non compaiono i commenti. Infatti, ogni volta che viene riconosciuto un commento, il parser lo inserisce in una lista, associandolo al nome della sezione in cui si trovava. In questo modo il pretty-printer è in grado di riscrivere i commenti nella giusta posizione.

Abbiamo però riscontrato dei problemi: prima di tutto, il nostro parser non è completo, perché è in grado di riconoscere tutta la sezione (distinguendo commenti e assegnamenti) ma, per qualche motivo, non è in grado di riconoscere la fine di una sezione, e quindi di fatto va in errore ogniqualvolta una sezione termina.

Inoltre, non siamo riusciti a implementare il pretty-printer. Abbiamo comunque creato la lista per contenere i commenti con la loro posizione, e la nostra idea era quella di scrivere una funzione pretty-printer che, stampando correttamente i nomi delle sezioni e delle loro variabili, avrebbe aggiunto dopo il nome della sezione tutti i commenti che erano stati trovati in questa sezione, uno dopo l'altro.