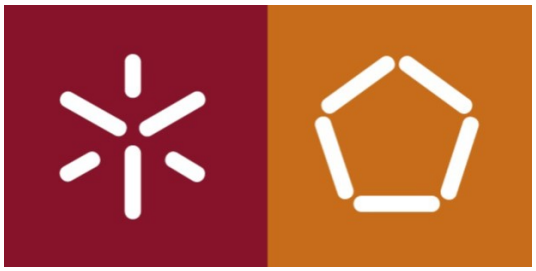

Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Relatório do Trabalho Prático 2

75248 Ana Sofia Gomes Marques
A65277 Flávio Manuel Machado Martins

Grupo 27

26 Junho 2020

Conteúdo

1 Introdução

Este projeto, desenvolvido no ambito da unidade curricular de Processamento de Linguagens, tem como principal objetivo desenvolver um processador de linguagem segundo o método da tradução dirigada pela sintaxe, suportado numa gramática tradutora(GT). Este será desenvolvido com a utilização do par de geradores de compiladores flex/yacc.

Para realizar este projeto iremos ter como vase uma gramática independente de contexto(GIC) que satifaz a condição LR() para criar a Linguagem de Domínio Específico(DSL).

O nosso grupo vai desenvolver o enunciado 4 proposto, este consiste na escrita duma gramática capaz de cubrir um subconjunto da linguagem TOML, pretendemos iniciar com um conjunto bastante robusto, embora não cubra inicialmente toda a linguagem TOML, deve ser desenvolvido com intenção de que possa eventualmente se tornar um tradutor completo de TOML para json.

Após o desenvolvimento da Gramática Independente de Contexto iremos desenvolver uma especificação em flex que permite reconhecer os símbolos terminais da linguagem desenvolvida e uma série de acções semânticas para cada produção concebendo assim uma Gramática Tradutora (GT).

Então iremos criar um ficheiro de input capaz de cobrir o subconjunto selecionado e realizar os testes utilizando algum validador de json para confirmar a escrita dum ficheiro json válido.

O presente relatório está divido em várias secções, especificando com detalhe as decisões tomadas para cada uma das fases do projeto acima referidas. No final serão apresentado alguns exemplos de teste assim como uma conclusão face à aprendizagem e dificuldades obtidas ao longo do desenvolvimento do projeto.

2 Contextualização

TOML é um formato de arquivo de configuração criado por forma a ser mais legível usando uma sintaxe mínima. O nome é um acrônimo para "Tom's Obvious, Minimal Language".
O formato foi desenvolvido para mapear de forma inequívoca para tabelas hash.

Em baixo é apresentada uma tabela que compara vários formatos de configuração, sendo nomeadamente do nosso interesse os formatos TOML e JSON.

Format Comparison

Format	Formal Standard	Flexible Standard	Strongly Typed	Easy Implementation	Human Readable	Allows comments
JSON	Yes	No	Yes	Yes	Yes	No
YAML	Yes	No	Yes	No	Yes	Yes
TOML	Yes	No	Yes	Yes	Yes	Yes
INI	No	Yes	No	Yes	Yes	Yes

Figura 1: Comparação entre TOML e outros formatos.

De seguida iremos descrever o subconjunto de toml sobre o qual este projeto foi desenvolvido.

O toml é contituido por tabelas e pares chave/valor.

Para representar um comentário é utilizado um # seguido do comentário.

Os pares chave/value são encontrados na forma chave = value. A chave, sinal de igual e valor devem estar na mesma linha, embora alguns valores possam ser divididos em várias linhas.

Tabelas são colecções de pares chave/valor ou tabelas. Podemos encontrar as tabelas em linhas sozinhos e entre parênteses retos. Podem ainda haver tabelas aninhadas, isto é tabelas dentro de tabelas.
Uma maneira de representar tabelas aninhadas é colocando o nome da tabela separados por pontos, por exemplo para uma tabela2 dentro duma tabela1: [tabela1.tabela2]
Na sepração descrita anteriormente é possível que existam espaços antes e após os ., por exemplo [tabela1 . tabela2 . tabela3] é aceite, assim como ["a". "b".c.d], sendo que, para a existência da tabela d, a c pode não existir previamente, não fazendo ["a". "b".c.d] errado a inexistência das tabelas anteriores.
Uma chave vazia deve estar não vazia, mas uma chave entre aspas vazia é permitida, apesar de este comportamento ser desencorajado de ser utilizado.

Um boolean é true ou false.

Uma data pode ter vários formatos como por exemplo:

- 1979-05-27T07:32:00Z
- 1979-05-26T15:32:00+08:00
- 1979-05-27T07:32:00

- 1979-05-27

Um time pode ter dois formatos que podemos ver de seguida:

- 21:30:00
- 21:30:00:9999

As strings podem ser encontradas como um seqência de letras, números, '_' e'-' ou alternativamente qualquer sequência de caracteres desde que delimitada por ' ou ".

Existem ainda string multi linhas, estas podem ocuprar várias linhas e são delimitadas em no inicio e final por três ' ou ".

Números inteiros são encontrados iniciando opcionalmente por + ou -, e ou é 0 ou o primeiro número varia entre 1 e 9, podendo ser seguido de uma sequência de inteiros e podendo conter '_' no meio para facilitação da leitura.

Podemos encontrar ainda floats, que são um inteiro seguido duma parte decimal ou exponencial. Por exemplo:

- -10,1
- 10e-2

3 Gramática

3.1 Gramática Independente de Contexto

G = (
T = { '#' , '=', '[', ']', ',', '.', '{', '}', string, inumber, fnumber, TRUE, FALSE, date, times, triplequote }
NT = { Toml, AtributeList, Atribute, Key, Value, Section, Array, ValueList, Multiline }
S = Toml
P = {
Toml -> AtributeList ;
AtributeList -> AtributeList Atribute | Atribute
Atribute -> Key '=' Value | '[' Section ']'
Section -> Section '.' string | string
Key -> string
Value -> string
| inumber
| fnumber
| TRUE
| FALSE
| date
| time
| Array
| multiline Multiline multiline
Array -> '[' ']' | '[' ValueList ']'
ValueList -> ValueList ',' Value | Value
Multiline -> Multiline string | string
}
)

3.2 Expressões Regulares

3.2.1 Símbolos

A expressão regular dos simbolos é, naturalmente, o próprio símbolo.

Símbolo	Expressão Regular
#	\#
=	\=
[	\[
]	\]
,	\,
.	\.
{	\{
}	\}

3.2.2 Palavras Reservadas

Palavra Reservada	Expressão Regular
TRUE	true
FALSE	false

3.2.3 Expressões

De seguida desenvolvemos as expressões regulares para as expressões:

- string: Uma string pode ser uma sequência de letras,números, '-' e '_', ou uma qualquer sequência de caracteres desde que delimitada por aspas ou apostrofos

```
([a-zA-Z0-9\-\_]+|(\"|\')(.*)(\"|\'))
```

- inumber: Um número inteiro, opcialmente pode ser iniciado por - ou + e para melhor leitura pode ter '_', não pode começar por 0.

```
(0|(-|\+)?[1-9]((_?)[0-9])*)
```

- fnumber: Um número decimal, do tipo 0,1 ou 1e-1.

```
(-|\+)?[0-9]+((.[0-9]+)|(e(\+|-)?[0-9]+))
```

- date: uma data, que pode incluir o tempo.

```
[0-9]{4}(-[0-9]{2}){2}(T(([0-2][0-9](:[0-6][0-9]){2})(Z|(\+[0-1][0-9]:([03]0|45))?))((.|-|\+
```

- time: tempo.

```
[0-9]{2}(:[0-9]{2}){2}(.[0-9]+)?
```

- triplequote: sequência de três aspas ou apostrofos.

```
([a-zA-Z0-9\-\_]+|(\"|\')(.*)(\"|\'))
```

3.3 Acções Semânticas

Após a criação da gramática da linguagem foram definidas as acções semânticas para cada produção de modo a converter o ficheiro toml processado para json.
Uma forma de realizar esta conversão é pensar no ficheiro toml como uma lista de atributos, esta lista é sequêncial na ordem que irá aparecer após conversão para json, como tal pode ser percorrida e escrita desde que seja feita alguma reorganização. Para tal são utilizadas algumas funções bem conhecidas de c e ainda algumas criadas propositadamente para este projeto.

```
Toml
    : AtributeList  { $1[strlen($1)-1] = '\0'; fprintf(yyout,"{\n%s%s\n}\n", $1, finalclose()); }
    ;
```

A acção semântica associada à produção Toml apenas faz alguma formatação e o processamento final do json. Esta começa por eliminar a virgula desnecessária que se encontra no final do resultado vindo em atributeList, e recorre à função finalclose que trata da formatação final do json após processamento de todo o toml de entrada, mais concretamente fecha as secções que ainda estejam abertas.

```
AtributeList
    : AtributeList Atribute  {
                             char* aux = strdup($2); char* s; char* d;
                             for (s=d=aux;*d=*s;d+=(*s++!='\t'));
                             /* remove all '\t' */
                             if( aux[0] == '}' ) {
                             /*  if Atribute starts with '}' => new section */
                                 $1[strlen($1)-1] = '\0';
                                 /* new section => no ',' at the end of previous line */
                             }
                             asprintf(&$$,"%s\n%s",$1,$2); }
| Atribute      { asprintf(&$$,"%s",$1); }
;
```

Na AtributeList é simplesmente necessário processar sequêncialmente os atributos que são encontrados, quando encontramos um atributo sozinho limitamo-nos a fazer a sua escrita, quando encontramos um atributo que ocorre após uma lista de atributos, fazemos a escrita do atributo após a lista sendo que é realizada uma correcção da lista até ao momento, isto é, caso encontremos o final duma secção eliminamos a ',' escrita no final da lista, visto que esta não será necessária.

```
Atribute
    : Key '=' Value  { printKeyVal(&$$, $1, $3); }
    | '[' Section ']'  { sectionUpdate($2,&$$); }
    ;
```

Nos atributos que encontramos do tipo par chave valor apenas é necessário fazer a sua escrita, recorrendo a uma função criada para o efeito, printKeyVal, que faz uma escrita adequado ao formato de json, no caso de encontrarmos uma Section é necessário chamar uma função alternativa. sectionUpdate, que além de realizar a escrita necessária actualiza a secção em que o processamento se encontra, sendo que a escrita é feita de forma fechar as secções anterior caso necessário antes de iniciar a encontrada.

```
Section
    : Section '.' string { asprintf(&$$,"%s %s", $1, $3); }
    | string { asprintf(&$$,"%s", $1); }
    ;
```

A Section apenas é formatada para o formato que é esperado na função sectionUpdate.

```
Key
    : string  { asprintf(&$$,"%s", $1); }
    ;

Value
    : string { asprintf(&$$,"\"%s\"",$1); }
    | inumber { asprintf(&$$,"%i",$1); }
    | fnumber { asprintf(&$$,"%lf",$1); }
    | TRUE { asprintf(&$$,"true");  }
    | FALSE { asprintf(&$$,"false"); }
    | date { asprintf(&$$,"\"%s\"",$1);  }
    | time { asprintf(&$$,"\"%s\"",$1);  }
    | Array { asprintf(&$$,"%s",$1); }
    | triplequote Multiline triplequote { asprintf(&$$,"\"\%s\"",$2); }
    ;
```

Ao encontrar uma Key ou value a única acção necessária é a de atribuição da chave recebida à Key, ou a atribuição do valor a Value.

```
Array
    : '[' ']'  { asprintf(&$$,"[ ]");  }
    | '[' ValueList ']'{ asprintf(&$$,"[\n%s\n%s]",$2,tabs(lvl)); }
    ;

ValueList
    : ValueList ',' Value  { asprintf(&$$,"%s,\n%s%s",$1,tabs(lvl+1),$3); }
    | Value   { asprintf(&$$,"%s%s",tabs(lvl+1),$1); }
    ;
```

No caso de encontrar um array como value, prodemos facilmente processar a lista de valores na produção(ValueList) e passar os mesmos para o Array onde se faz a escrita dele e de todos os valores devidamente formatado para json.

```
Multiline
    : Multiline string  { asprintf(&$$,"%s\\n%s",$1,$2); }
    | string { asprintf(&$$,"%s",$1); }
    ;
```

As strings em múltiplas linhas não são compatíveis com json, pelo que devem ser formatadas para ocuparem apenas uma utilizando $\backslash n$ como separador das várias linhas.

4 Implementação

```
extern int asprintf();

char* tabs(int i);
char* closelvls(int newlvl);
char* finalclose();
void printKeyVal(char** res, char* key, char* value);
void sectionUpdate(char* newSection, char** res);

int lvl = 1;
const char* section;
```

Como foi possível perceber ao longo das secções anteriores não existe necessidade de uma grande estrutura para este processamento, o ficheiro toml vai sendo processado e uma string, com recurso à função externa aspritnf, vai sendo escrita em formato json. Para uma escrita adequada apenas é necessário manter o nivel da secção, e referência ao nome da secção. Foram desenvolvidas funções auxiliares que resumimos em seguida.
A função tabs apenas devolve uma char* com a impressão das tags, permitindo escrever as linhas num formato mais agradavel de ler. A função closelvls recebe o novo nivel, após uma alteração de secção e devolve um char* onde fechou os conjuntos necessários no json. A função finalclose faz uma escrita final no ficheiro, mais concretamente fecha os conjuntos ainda abertos. A função printKeyVal faz a impressão dum par Key,Value em res. Por fim a função sectionUpdate recebe uma nova secção e escreve as linhas necessárias para fazer actualização do conjunto no ficheiro json, esta funçao escreve no json a abertura do conjunto, fecha os conjuntos por fechar, caso necessário, e ainda actualiza no programa a secção e o nível.

5 Utilização

5.1 Input Exemplo

```
title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
date = 2010-04-23
date1 = 1979-05-27T07:32:00Z  # UTC time, following RFC 3339/ISO 8601 spec
date2 = 1979-05-27T15:32:00+08:00  # with RFC 3339/ISO 8601 offset
date3 = 1979-05-27T07:32:00 # without offset
date4 = 1979-05-27  # without offset or time
time = 21:30:00
"time." = 21:30:00:9999
trueBool = true
falseBool = false
teste_ = underscore

[database]
server = "192.168.1.1"
ports = [ 8001, 8001, 8002 ]
connection_max = 5000
enabled = true

[servers]
[servers.alpha]
ip = "10.0.0.1"
dc = "eqdc10"

[servers . beta]
ip = "10.0.0.2"
dc = "eqdc10"
[servers.beta.lvl3.lvl4]
lvl3Accepted = true

# Line breaks are OK when inside arrays
hosts = [
"alpha",
"omega"
]

"" = emptyKey
'n' = singlequotekey
"empty List" = [ ]

array4 = [ [ 1.2, 2.4 ], [3,6] ]

multilineStr = """ multi
line
string
"""
```

```
multilineStr2 = ''' multi
line
string
'''

multilineStr3 = '"' multi
line
string
"'"

integer = 10
intNeg = -10
intWith_ = 100_000_000
float = 10.2
floatExp = 10e-3
zero = 0

key = end
[ serversnew ]
asdas = "adas"
```

Este é um exemplo que demonstra grande parte da linguagem aceite. Este ficheiro foi usado para vários testes e o seu resultado foi confirmado como sendo um ficheiro json válido.

5.2 Opções aceites

Para correr o programa existem 3 opções:

- toml2json: Nesta opção corremos a aplicação sem argumentos, e podemos fazer alguns testes escrevendo diretamente o toml e obtendo no terminal o resultado
- toml2json ficheiroEntrada: Nesta opção corremos a aplicação com um argumento, o ficheiro toml de origem e obtemos no terminal o resultado
- toml2json ficheiroEntrada ficheiroSaida: Nesta opção corremos a aplicação com dois argumento, o ficheiro toml de origem e o nome do ficheiro json destino que vamos criar, se necessário, e escrever o resultado.

Caso seja inserido um input inválido é dada a seguinte mensagem:
Número de argumentos inválido
Pequenos testes imediatos: toml2json
Visualização imediata de ficheiro: toml2json ficheiroEntrada
Conversão para ficheiro: toml2json ficheiroEntrada ficheiroSaida
Em caso de falha na abertura de algum dos ficheiros, é devolvido um erro indicando qual ficheiro falhou.

5.3 Output

Por fim iremos analisar o resultado da aplicação aplicada ao ficheiro descrita em 6.1.

5.3.1 String

Nesta secção iremos demostrar uma lista com input e output de strings.

String em Toml	Resultado em json
name = "Tom Preston-Werner" ⠿ = emptyKey 'n' = singlequotekey	"name": "Tom Preston-Werner", "⠿ : "emptyKey", "n": "singlequotekey",
multilineStr3 = '"' multi line string "'	"multilineStr3": "multi\nline\nstring",

5.3.2 Dates

Date em Toml	Resultado em json
date1 = 1979-05-27T07:32:00Z	"date1": "1979-05-27T07:32:00Z",
date2 = 1979-05-27T15:32:00+08:00	"date2": "1979-05-27T15:32:00+08:00",
date3 = 1979-05-27T07:32:00	"date3": "1979-05-27T07:32:00",

5.3.3 Números

Número em Toml	Resultado em json
intNeg = -10	"intNeg": -10,
intWith_100_00_00	"intWith" : 100000000
float = 10.2	"float": 10.200000,
floatExp = 10e-3	"floatExp": 0.010000,

5.3.4 Array

Toml	Resultado em json
ports = [8001, 8001, 8002]	"ports": [8001, 8001, 8002],
array4 = [[1.2, 2.4], [3,6]]	"array4": [[1.200000, 2.400000], [3.000000]],

5.3.5 Booleans

Toml	Resultado em json
trueBool = true	"trueBool": true,
falseBool = false	"falseBool": false,

5.3.6 Sections

Por fim demonstramos a parte mais complexa desenvolvida.

Começando pelo final, olhemos ao exemplo seguinte: Input:

```
...
key = end
[ serversnew ]
asdas = "adas"
```

Temos o correspondente em json:

```
                    "key": "end"
                }
            }
        }
    },
    "serversnew": {
        "asdas": "adas"
    }
}
```

Aqui de imediato se verifica como o final do ficheiro json é corretamente escrito, além de que o inicio duma tabela de nível 1 faz com que as tabelas anteriores sejam corretamente fechadas.

De seguida resta demonstrar como as tabelas novas são abertas.

Olhando ao exemplo seguinte iremos ver o correto comportamento para os casos de tabelas aninhadas mesmo quando a anterior ainda não foi aberta.

Input:

```
[servers]
[servers.alpha]
    ip = "10.0.0.1"
    dc = "eqdc10"

[servers . beta]
    ip = "10.0.0.2"
    dc = "eqdc10"
[servers.beta.lvl3.lvl4]
    lvl3Accepted = true
```

Resultado:

```
    "servers": {
        "alpha": {
            "ip": "10.0.0.1",
            "dc": "eqdc10"
        },
        "beta": {
            "ip": "10.0.0.2",
            "dc": "eqdc10",
            "lvl3": {
            "lvl4": {
                "lvl3Accepted": true,
```

6 Conclusão

Relativamente ao desenvolvimento e conclusão deste projeto o grupo considera que este foi desenvolvido com sucesso, não tendo encontrado grandes dificuldades na sua execução.
Embora o problema posso parecer complicado o bom desenvolvimento inicial da gramática independente de contexto permitiu que a realização das tarefas propostas dosse relativamente simples. As maiores dificuldades acabaram por ser em decidir como gerir as secçoes pois existe mais do que uma possibilidade de como estas são encontradas e mais que uma opção a tomar, e em dar um retoques finais no output final de forma a que ele devolvesse um ficheiro json válido, mas, com a gestão das secções e verificando um correto fecho e inicio das mesmas conseguimos formatar o resultado para um json válido e capaz de cobrir um alargado subconjunto de toml.
Além disso acreditamos que o projeto desenvolvido pode ser incrementado e eventualmente se tornar num tradutor de TOML para JSON bastante válido e completo. A realização deste projeto permitiu um enriquecimento do conhecimento dos membros do grupo sobre expressões regulares, GICs, acções semânticas e modo de funcionamento de um compilador simples.
Como trabalho futuro consideramos talvez o enriquecimento do projeto com mais funcionalidades, como por exemplo:

- Formatação das datas no ficheiro original para um formato único em json, isto pode ser feito com base na utilização que será dada ao json
- Verificação de erros nas secções aninhas.
- Implementação de ERs e processamento de mais values
- Processamento de mais formatos para secções

7 Referências

Stephen C. Johnson. (2017). Yacc: Yet Another Compiler-Compiler.
[online] Available at:
http://epaperpress.com/lexandyacc/download/yacc.pdf
[Accessed 7 Jun. 2017].

Tom Niemann. (2017). LEX YACC TUTORIAL
[online] Available at:
http://epaperpress.com/lexandyacc/download/LexAndYaccTutorial.pdf
[Accessed 9 Jun. 2017].

Código do Programa

A Ficheiro Flex

```
%{
#include "y.tab.h"
%}
%option yylineno
%x MULTINE
%%
[ \t\n\r]  ;
#[^\n]+ ;

[\=\[\]\,\.\{\}] return yytext[0];

    /* Boolean */
true  return TRUE;
false return FALSE;

    /*  Integers can start with a +, a - or nothing.
        Leading zeros are not allowed.
        Underscores accepted for better readability */
(0|(-|\+)?[1-9]((_?)[0-9])*)    { char* s; char* d; for (s=d=yytext;*d=*s;d+=(*s++!='_'));
                                  // remove os '_' caso existam
                                  yylval.ivalue = atoi(yytext);
                                  return inumber; }

    /* Floats are an integer followed by a fractional and/or an exponent part. */
(-|\+)?[0-9]+((.[0-9]+)|(e(\+|-)?[0-9]+)) { yylval.fvalue = atof(yytext);
                                            return fnumber; }

    /* DateTime
    #offset datetime
    odt1 = 1979-05-27T07:32:00Z
    odt2 = 1979-05-27T00:32:00-07:00
    odt3 = 1979-05-27T00:32:00.999999-07:00
    # local datetime
    ldt1 = 1979-05-27T07:32:00
    ldt2 = 1979-05-27T00:32:00.999999
    # local date
```

```
    ld1 = 1979-05-27
    */
[0-9]{4}(-[0-9]{2}){2}(T(([0-2][0-9](:[0-6][0-9]){2})(Z|(\+[0-1][0-9]:
([03]0|45))?))((.|-|\+)[0-9]+((-|\+|:)[0-9]{2}(:[0-9]{2})?)?)?)? {
                                    yytext[yyleng] = '\0';
                                    yylval.svalue = strdup(yytext);
                                    return date; }

    /* Time */
[0-9]{2}(:[0-9]{2}){2}(.[0-9]+)?    { yytext[yyleng] = '\0';
                                      yylval.svalue = strdup(yytext);
                                      return time; }

     /* multi line strings:
     """
     string
     newline
     """ or ''' str ''' */
(\"|\'){3} { return triplequote; }



    /* abc-d_e | ("abcde" | 'abcde' | 'abc" | "abc') */
([a-zA-Z0-9\-\_]+|(\"|\')(.*)(\"|\'))   {  if( (yytext[0] =='\"') || (yytext[0] =='\'')) {
                                               yytext[yyleng-1] = '\0';
                                               yylval.svalue = strdup(yytext+1);
                                           } else {
                                               yytext[yyleng] = '\0';
                                               yylval.svalue = strdup(yytext);
                                           }
                                           return string;
                                           }

.   return ERRO;
```

B Ficheiro Yacc

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

extern int yylex();
extern int yylineno;
extern char *yytext;
extern FILE* yyin;
extern FILE* yyout;
int yyerror();
```

```
int erroSem(char*);

extern int asprintf();

char* tabs(int i);
char* closelvls(int newlvl);
char* finalclose();
void printKeyVal(char** res, char* key, char* value);
void sectionUpdate(char* newSection, char** res);

int lvl = 1;
const char* section;
%}
%union{
    int ivalue;
    float fvalue;
    char* svalue;
}

%token ERRO string inumber fnumber TRUE FALSE date time triplequote
%type <ivalue> inumber
%type <fvalue> fnumber
%type <svalue> string date time AtributeList Atribute Key Section Value Array ValueList Multiline

%%
Toml        /* remove last ','     print with proper { }     finalclose write close for open sections
    : AtributeList  { $1[strlen($1)-1] = '\0'; fprintf(yyout,"{\n%s%s\n}\n", $1, finalclose()); }
    ;

AtributeList
    : AtributeList Atribute  {
                            char* aux = strdup($2); char* s; char* d;
                            for (s=d=aux;*d=*s;d+=(*s++!='\t'));
                            /* remove all '\t' */
                            if( aux[0] == '}' ) {
                            /*  if Atribute starts with '}' => new section */
                                $1[strlen($1)-1] = '\0';
                                /* new section => no ',' at the end of previous line */
                            }
                            asprintf(&$$,"%s\n%s",$1,$2); }
| Atribute      { asprintf(&$$,"%s",$1); }
;

Atribute
    : Key '=' Value  { printKeyVal(&$$, $1, $3); }
    | '[' Section ']'  { sectionUpdate($2,&$$); }
    ;

Section
    : Section '.' string { asprintf(&$$,"%s %s", $1, $3); }
    | string { asprintf(&$$,"%s", $1); }
    ;
```

```
Key
    : string  { asprintf(&$$,"%s", $1); }
    ;

Value
    : string { asprintf(&$$,"\"%s\"",$1); }
    | inumber { asprintf(&$$,"%i",$1); }
    | fnumber { asprintf(&$$,"%lf",$1); }
    | TRUE { asprintf(&$$,"true");  }
    | FALSE { asprintf(&$$,"false"); }
    | date { asprintf(&$$,"\"%s\"",$1);  }
    | time { asprintf(&$$,"\"%s\"",$1);  }
    | Array { asprintf(&$$,"%s",$1); }
    | triplequote Multiline triplequote { asprintf(&$$,"\"\%s\"",$2); }
    ;

Array
    : '[' ']'  { asprintf(&$$,"[ ]");  }
    | '[' ValueList ']'{ asprintf(&$$,"[\n%s\n%s]",$2,tabs(lvl)); }
    ;

ValueList
    : ValueList ',' Value  { asprintf(&$$,"%s,\n%s%s",$1,tabs(lvl+1),$3); }
    | Value   { asprintf(&$$,"%s%s",tabs(lvl+1),$1); }
    ;

Multiline
    : Multiline string  { asprintf(&$$,"%s\\n%s",$1,$2); }
    | string { asprintf(&$$,"%s",$1); }
    ;

%%
int main(int argc, char *argv[]){
    if(argc==1) {
        yyparse();
        return 0;
    } else if (argc==2) {
        yyin = fopen(argv[1],"r");
        if (yyin == NULL) {
            printf("Erro na abertura do ficheiro de entrada\n");
            return 1;
        } else {
            yyparse();
            return 0;
        }
    } else if (argc==3) {
        yyin = fopen(argv[1],"r");
        if (yyin==NULL) {
            printf("Erro na abertura do ficheiro de entrada\n");
            return 2;
        }
```

```
        yyout = fopen(argv[2],"w");
        if(yyout == NULL) {
            printf("Erro na abertura do ficheiro de escrita\n");
            return 3;
        }
        yyparse();
        return 0;
    } else {
        printf("Número de argumentos inválido\n");
        printf("Pequenos testes imediatos:  toml2json\n");
        printf("Visualização imediata de ficheiro:  toml2json ficheiroEntrada\n");
        printf("Conversão para ficheiro:  toml2json ficheiroEntrada ficheiroSaida\n");
        return 4;
    }
    return 0;
}

int erroSem(char *s) {
    printf("Erro Semântico na linha: %d, %s...\n", yylineno, s);
    return 0;
}

int yyerror(){
    printf("Erro Sintático ou Léxico na linha: %d, com o texto: %s\n", yylineno, yytext);;
    return 0;
}

char* tabs(int i) {
    char* tabs = malloc(sizeof('\t')*i);
    for(int j=0;j<i;j++) {
        tabs[j] = '\t';
    }
    tabs[i] = '\0';
    return tabs;
}

void printKeyVal(char** res, char* key, char* value) {
    asprintf(res,"%s\"%s\": %s,", tabs(lvl),key, value);
}

void sectionUpdate(char* newSection, char** res) {
    int newlvl = 1;
    const char s[2] = " ";
    char* token;
    char* newSectionTemp;
    token = strtok(newSection,s);
    char* aux = "";
    while(token != NULL) {
        newlvl++;
        newSectionTemp = strdup(token);
        token = strtok(NULL, s);
        if((newlvl>lvl) && (token != NULL))
```

```
        asprintf(&aux,"%s\"%s\": { \n",tabs(newlvl-1),newSectionTemp);
        /* New section opens more then one section */
    }
    if (newlvl == lvl) {
        section = strdup(newSectionTemp);
        asprintf(res,"%s%s\"%s\": { ",closelvls(newlvl),tabs(newlvl-1),section);
    }
    else if(newlvl > lvl) {
        lvl = newlvl;
        asprintf(res,"%s%s\"%s\": { ",aux,tabs(lvl-1),newSectionTemp);
        section = strdup(newSection);
    } else {
        section = strdup(newSectionTemp);
        asprintf(res,"%s%s\"%s\": { ",closelvls(newlvl),tabs(newlvl-1),section);
        lvl = newlvl;
    }
}

char* closelvls(int newlvl) {
    char* close = "";
    int x = lvl;
    while(x>newlvl) {
        x--;
        asprintf(&close,"%s%s}\n",close,tabs(x));
    }
    asprintf(&close,"%s%s},\n",close,tabs(x-1));
    return close;
}

char* finalclose() {
    char* close = "";
    for(int x=lvl; x>1; x--) {
        asprintf(&close,"%s\n%s}",close,tabs(x-1));
    }
    return close;
}
```