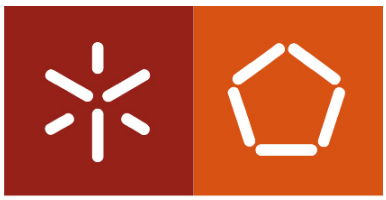

Escola de Engenharia
Universidade do Minho

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Trabalho Prático nº 1

Octávio Maia a71369
João Silva a72023
Rui Freitas a72399

6 de Abril de 2016

Conteúdo

Lista de Figuras

Resumo

Este relatório tem como objetivo descrever o processo de desenvolvimento, estruturação e tomadas de decisão resultantes da primeira fase do trabalho prático da Unidade Curricular de Processamento de Linguagens.

Nesta fase, foi decidido abordar o tema sobre o *Normalizador de ficheiros BibTeX*, cujo produto final consiste num interpretador de ficheiros *.bib*.

Palavras chave: BibTeX, HTML, FLEX, Estruturas dados, Expressões regulares.

Capítulo 1

Introdução

De modo a por em prática os conhecimentos adquiridos nas aulas da Unidade Curricular de Processamento de Linguagens, utilizamos a ferramenta FLEX[1], cuja utilização foi coberta extensivamente em ambas as aulas teóricas, bem como as práticas.

Neste relatório iremos expor os nossos métodos de desenvolvimento, bem como as decisões tomadas para concretizar a realização deste trabalho.

1.1 Descrição do Problema e Implementação

Após um debate entre os elementos do grupo sobre qual tema a desenvolver, ficou decidido desenvolver o *Normalizador de ficheiros BibTeX*[2], uma ferramenta de formatação utilizada em documentos LaTeX[3].

Este tem como objetivo o desenvolvimento o desenvolvimento de uma ferramenta que tenha como capacidade a análise de documentos *BibTeX* e a contagem das suas categorias, cuja contagem deverá ser exportada para um ficheiro *HTML*[4], bem como a normalização de texto e a construção de um grafo.

A nossa primeira tarefa teve como base a definição das várias expressões regulares que irão ser utilizadas no âmbito desta primeira fase do trabalho, de forma a filtrar as diversas informações dos ficheiros *.bib* fornecidos.

Após a finalização das expressões regulares tivemos como objetivo o desenvolvimento do código de forma a responder a todas as tarefas fornecidas.

[1]The Fast Lexical Analyzer, http://flex.sourceforge.net/
[2]http://www.bibtex.org/
[3]https://latex-project.org/intro.html
[4]HyperText Markup Language, https://www.w3.org/html/

Capítulo 2

Processamento do BibTeX

2.1 Contagem de categorias *BibTeX*

Nesta primeira alínea é pedido a criação de um ficheiro HTML que contenha a contagem das várias categorias presentes no documento *.bib* fornecido.

2.1.1 Expressões Regulares

As expressões regulares utilizadas nesta fase do exercício foram desenvolvidas para apenas trabalharmos com palavras reservadas do *BibTeX*, ou seja, todas a palavras que são antecedidas pelo caractere @ quando este se encontra no início da linha.

Além disso o grupo decidiu expandir a expressão regular para o caso do @ não ser o primeiro caractere da linha, mas poder ser antecedido por qualquer quantidade de espaços.

Uma vez que queríamos que todas as restantes expressões fossem ignoradas, inserimos a última expressão regular apresentada de forma a contabilizar todos esses casos.

Para contar com o descrito necessitamos de criar o estado de *reservada* onde apenas neste estado procuramos pela primeira palavra uma vez que anteriormente encontramos @ na forma descrita anteriormente, ou seja agora estamos perante uma "tag" do *BibTeX* que deveremos guardar.

```
%x reservada
%%
^[ ]*@ {BEGIN(reservada);}
<reservada>[A-Za-z]+  {char* aux = lowerCASE(yytext);addRegisto(aux);}
```

2.1.2 Estrutura de Dados

De forma a guardar a informação relativa a cada "tag" reservada bem como a quantidade de vezes que esta aparece no ficheiro de *input*, utilizamos uma simples estrutura com 2 "Arrays" em que num guardava a *string* correspondente à "tag" e no outro seria guardado o contador com o numero de vezes que esta aparecia no ficheiro, funcionado assim como um *Hash(String,Int)*.

Foram consideradas a possível existência de 30 "tags" diferente, sendo assim alocadas no inicio do programa estaticamente 30 posições nos dois *arrrays*.

2.1.3 Ações executadas

A quando da inicialização do programa as estruturas de dados apresentadas em 2.1.2 de modo garantir a fiabilidade do programa.

Quando é encontrado um *match* na 1º expressão regular responsável declaramos que entramos no estado de palavra reservada.

Uma vez neste estado quando encontramos a 1ª palavra essa será a palavra reservada que queremos do *BibTeX*, ou seja adicionamos-la à estrutura de dados para isso recorrendo à função desenvolvida *addRegisto(char* nome)*.

O funcionamento da função *addRegisto* baseia-se em procurar no 1º *array* por uma *string* igual à que é passada, caso isso ocorra na posição correspondente do *array* com a contagem correspondente è adicionado 1, caso não seja encontrado nenhum "match" adicionamos à primeira posição livre o novo registo.

```
void addRegisto(char* nome){
    int found=0;
    int i;
    for(i=0;i<30 && nomesCategorias[i]!=NULL && found==0;i++){
        if(!strcmp(nomesCategorias[i],nome)){
            found=1;
            contCategorias[i]++;
        }
    }
    if(nomesCategorias[i]==NULL && found==0){
        nomesCategorias[i] = strdup(nome);
        contCategorias[i]++;
    }
}
```

Após o término do processamento do ficheiro de *input* é criado um ficheiro *index.html* que contem toda a informação presente na estrutura de dados referida no ficheiro de *input* em forma de lista, possível de ver num *browser*.

2.1.4 Execução do programa

Após a execução do programa é criado para esta parte o ficheiro *index.html*.

```
flex tp1.l
gcc -o tp1 lex.yy.c
./tp1 < lib.bib
more index.html
<html>
        <head>
                <title>EXERCICIO 1</title>
        </head>
        <body>
                <h2>Lista Exercico 1</h2>
                        <ul>
                        <li> string -> 31 </li>
                        <li> techreport -> 140 </li>
                        <li> inbook -> 3 </li>
                        <li> book -> 47 </li>
                        <li> phdthesis -> 21 </li>
                        <li> article -> 142 </li>
                        <li> inproceedings -> 209 </li>
                        <li> unpublished -> 15 </li>
                        <li> incollection -> 6 </li>
                        <li> manual -> 13 </li>
                        <li> misc -> 61 </li>
                        <li> proceedings -> 4 </li>
                        <li> mastersthesis -> 2 </li>
                        <li> proceeding -> 1 </li>
                        </ul>
        </body>
</html>
```

Figura 2.1: Execução da primeira alínea do projeto e respetivo output.

Lista Exercico 1

- string -> 31
- techreport -> 140
- inbook -> 3
- book -> 47
- phdthesis -> 21
- article -> 142
- inproceedings -> 209
- unpublished -> 15
- incollection -> 6
- manual -> 13
- misc -> 61
- proceedings -> 4
- mastersthesis -> 2
- proceeding -> 1

Figura 2.2: Página HTML gerada anteriormente.

Através da página HTML gerada podemos verificar as diversas categorias presentes no ficheiro *BibTeX* fornecido, bem como as vezes que estas aparecem.

2.2 Ferramenta de normalização

Na segunda alínea é pedido o desenvolvimento de uma ferramenta de normalização e a capacidade desta fazer *pretty-priting*.

2.2.1 Expressões Regulares

As expressões regulares utilizadas na segunda fase do exercício foram desenvolvidas de forma a coletar informação relativa aos autores de um dado livro, bem como o título do mesmo.

Foram então criados os seguintes campos para guardar informação:

- *char* a[500]* cujo objetivo é o armazenamento dos vários nomes que constituem um nome de um autor.
- *char lixo[5000]* cujo objetivo é guardar a informação não desejada, ou seja, toda a informação exclusive os campos *author* e *title*
- *int titleBOOL=0, authorBOOL=0* cujo objetivo é servir de *flag* para o caso de o campo *title* e o campo *author* terem sido encontrados.

De forma a representar a informação referenciada anteriormente, foi criado o estado *author*, a *categoria* e o *title*.

```
@[A-Za-z]+\{[A-Za-z0-9\:\+]+,
<categoria>[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*[\"\{]
<author>[ ]+[Aa][Nn][Dd][ ]+
<author>[ ]+[Aa][Nn][Dd][ ]*\n?[ ]*
<author>[ ]*\n[ ]*[Aa][Nn][Dd][ ]+
<author>\{[^ ,]+[^ \"\}]\}
<author>[^ ,]+([^ \"\},\n]|[^ \"\}],)
<author>[ ]*[\"\}],?\n
<author>.|\n
<categoria>[ \n\t][Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*\"
<categoria>[ \n\t][Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*\{
<title>[^=]*\",?
<title>[^=]*\},?
<categoria>[ ]*=[ ]*\"
<categoria>\",
<categoria>.|\n
<categoria>\}\n
.|\n
```

2.2.2 Estrutura de Dados

Nesta fase sendo mais complexa foram criados os seguintes campos para guardar informação para auxiliar a execução pretendida

- *char* a[500]* cujo objetivo é o armazenamento dos vários nomes que constituem os nomes dos autores.
- *int n* é a variável auxiliar que permite contar o número de nomes que um autor tem, por exemplo para "Jose Antonio" o valor de n seria 2.
- *int numeroAutores* variável que guarda o numero total de autores que participaram num livro.
- *int numeroNomesAutores[100]* é o *array* onde é guardado o numero de nomes correspondentes aos autores em a, de modo a iterar de nome completo em nome completo.
- *int bool=0* São guardados os nomes normalizados dos autores de uma publicação.

2.2.3 Ações executadas

- `@[A-Za-z]+\{[A-Za-z0-9:\+]+`

  Expressão regular que marca o inicio de um registo de um dado bibliográfico, onde se encontram os campos dos registos que queremos processar.

- `<categoria>[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*[\"\{]`

  Expressão regular que marca o inicio de um registo de um conjunto de autores, para isso faz reset as variáveis e estruturas de dados para este novo registo de autores.

- `<author>[ ]+[Aa][Nn][Dd][ ]+`
  `<author>[ ]+[Aa][Nn][Dd][ ]*\n?[ ]*`
  `<author>[ ]*\n[ ]*[Aa][Nn][Dd][ ]+`

  Expressões regulares que marcam a delimitação dos nomes dos distintos autores de uma obra, registando no array que representa o numero de nomes que o autor em analise tem. Indicamos também que a obra tem mais que um autor, e o numero de nomes do próximo autor será inicializado a 0.

- `<author>\{[^ ,]+[^ \"\}]\}`
  `<author>[^ ,]+([^ \"\},\n]|[^ \"\}],)`
  `<author>[ ]*[\"\}],?`

  Expressões regulares que marcam o registo de um nome presente num autor, guardando-os no array auxiliar a onde os nomes de todos os autores são guardados.

- `<author>[ ]*[\"\}],?`

  Expressões regulares onde encontramos o final do campo correspondente aos autores da obra em análise. Após passamos para o contexto de *categoria* e depois registamos no ficheiro do *grafo* todas as ligações presentes na obra em analise, para isso recorremos à função *void registaAuthors()*.

- `<categoria>[ \n\t][Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*\"`
  `<categoria>[ \n\t][Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*\{`

  Expressão regular que marca o início do registo de um titulo, iniciando assim um novo contexto.

2.2.4 Execução do programa

De modo a executar o programa mais facilmente, procedemos à criação de uma simplex *makefile*, de modo a poupar repetições de comandos *(flex, gcc e execução)*.

```
@Misc{CH05a,
author = { D. Cruz and
        P. Henriques },
 title = {LISS - Language of Integers, Sequences and Sets},
    howpublished = {Talk to the gEPL, Dep. Inform\'atica / Univ. Minho},
    year = 2005,
    month = {Oct.},
    annote = {compilacao, ga, geracao codigo, VM"
    }
```

Figura 2.3: Output da segunda alínea.

2.3 Construção de grafo das ligações presentes no ficheiro e de um autor

Na última alínea é pedida a construção de um grafo para um determinado autor, que mostre todos os autores que publicaram artigos com esse autor, recorrendo ao *GraphViz2* e à linguagem *Dot.*

2.3.1 Expressões Regulares

```
@[A-Za-z]+\{[A-Za-z0-9:\+]+,
<categoria>[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*[\"\{]
<author>[ ]+[Aa][Nn][Dd][ ]+
<author>[ ]+[Aa][Nn][Dd][ ]*\n?[ ]*
<author>[ ]*\n[ ]*[Aa][Nn][Dd][ ]+
<author>\{[^ ,]+[^ \"\}]\}
<author>[^ ,]+([^ \"\},\n]|[^ \"\}],)
<author>[ ]*[\"\}],?
<author>.|\n
<categoria>.|\n
<categoria>\}\n
.|\n
```

2.3.2 Estrutura de Dados

Nesta fase foram usadas as variáveis disponíveis na alínea B bem como as variáveis seguintes:

- *int nParceiros* cujo objetivo é guardar a informação relativa ao numero de parceiros do autor que é passado como autor a procurar.
- *authorParceria[100]* cujo objetivo é guardar os nomes de dos autores que participaram em alguma obra com o autor em causa.

2.3.3 Ações executadas

- `@[A-Za-z]+\{[A-Za-z0-9:\+]+`

  Expressão regular que marca o inicio de um registo de um dado bibliográfico, onde se encontram os campos dos registos que queremos processar.

- `<categoria>[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*[\"\{]`

  Expressão regular que marca o início de um registo de um conjunto de autores, para isso faz reset as variáveis e estruturas de dados para este novo registo de autores.

- `<author>[ ]+[Aa][Nn][Dd][ ]+`
  `<author>[ ]+[Aa][Nn][Dd][ ]*\n?[ ]*`

`<author>[ ]*\n[ ]*[Aa][Nn][Dd][ ]+`

Expressões regulares que marcam a delimitação dos nomes dos distintos autores de uma obra, registando no array que representa o numero de nomes que o autor em analise tem. Indicamos também que a obra tem mais que um autor, e o numero de nomes do próximo autor será inicializado a 0.

- ```
  <author>\{[^ ,]+[^ \"\}]\}
  <author>[^ ,]+([^ \"\},\n]|[^ \"\}],)
  <author>[ ]*[\"\}],?
  ```

  Expressões regulares que marcam o registo de um nome presente num autor guardando-os no array auxiliar *a* onde os nomes de todos os autores são guardados.

- `<author>[ ]*[\"\}],?`

  Expressões regulares onde encontramos o final do campo correspondente aos autores da obra em análise. Após passamos para o contexto de *categoria* e depois registamos no ficheiro do *grafo* todas as ligações presentes na obra em analise, para isso recorremos à função *void registaAuthors()*.

2.3.4 Execução do programa

A execução deste programa vai gerar dois ficheiros *.dot*. Um deles com todas as ligações presente, e outro apenas com as ligações dos autores escolhidos.

Estes ficheiros serão depois processados pelo programa *dot* de modo a obtermos uma representação visual do grafo das ligações presentes no ficheiro *.bib*.

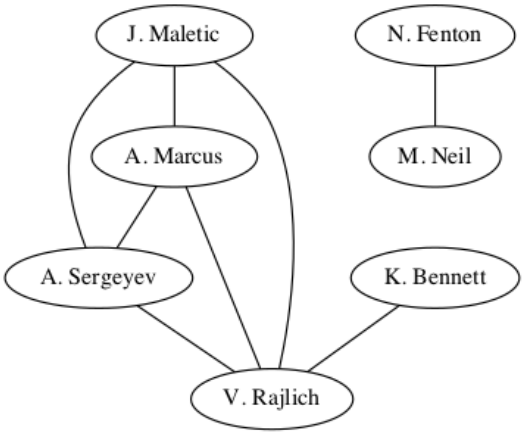

Figura 2.4: Excerto do grafo produzido pela terceira alínea.
  ```

Capítulo 3

Testes realizados

3.1 Alínea A

Para a realização desta alínea realizaram-se várias tentativas até encontrar a expressão regular que permite o correto processamento de texto.

De modo a facilitar a execução do programa criou-se uma *makefile* que faz a compilação do código, bem como a execução do mesmo.

```
1 p1:
2         flex tp1.l
3         gcc -o tp1 lex.yy.c
4         ./tp1 < lib.bib
5         more index.html
```

A página HTML resultante desta invocação pode ser verificada na imagem 2.2

3.2 Alínea B

Para a realização desta alínea foi criada a *makefile* proposta a seguir que faz a compilação do código, bem como a execução do mesmo.

```
1 p2:
2         flex tp2.l
3         gcc -o tp2 lex.yy.c
4         ./tp2 < lib.bib
```

Após a execução deste comando, será impresso no terminal o resultado do *pretty-printing* resultante.

3.3 Alínea C

Para a realização desta alínea foi criada a seguinte *makefile* que faz a remoção dos grafos já existentes, caso existam.

Após remover os ficheiros faz a compilação do código

```
p3:
        if [ -f "graf.dot" ]; then rm graf.dot; fi;
        if [ -f "grafAuthor.dot" ]; then rm grafAuthor.dot; fi;
        flex tp3.l
        gcc -o tp3 lex.yy.c
        ./tp3 ${ARGS} < lib.bib
```

De modo a produzir o grafo relativo a um autor em específico, neste exemplo será o autor P. Henriques, deverá ser executado o comando *make p3 ARGS="P. Henriques"*.

Caso executemos apenas *make p3* irá ser produzido um grafo geral, ou seja, irá representar as relações de todos os autores.

```
graph:
        if [ -f "graf.dot" ]; then dot -Tpng graf.dot -o graf.png;
           fi;
        if [ -f "grafAuthor.dot" ]; then dot -Tpng grafAuthor.dot -
           o grafAuthor.png; fi;
```

Após correr o comando mencionado anteriormente (*make p3* ou *make p3 ARGS="..."*) podemos executar o comando *make graph* de forma a gerar a imagem em formato PNG.

Capítulo 4

Conclusão

Neste projeto, o principal objetivo consistiu na aplicação da linguagem *FLex*, e a construção de *ER's* funcionais e eficazes para o propósito das mesmas.

Este conhecimento foi posto em prática, recorrendo ao tema posposto sobre *BibTeX*, sendo dividido em 3 partes uma de identificação de expressões reservadas, uma segunda parte de normalização do ficheiro *BibTex* e por fim a construção de um grafo em *Dot* para representar as ligações dos autores em grafos. Isto levou o grupo a realizar um estudo prévio sobre o que iríamos abordar, de forma a garantir que tudo fosse efetuado da forma correta.

Após o início do projeto, podemos afirmar que este fluiu de forma natural graças aos diversos conhecimentos adquiridos em ambas aulas práticas, bem como nas teóricas.

Devido a este grau de familiaridade com a linguagem *C* e a linguagem de programação onde o programa *Flex* se encontra embebido, podemos afirmar que desenvolvemos um projeto que cumpre os requerimentos propostos e cuja implementação foi feita de forma correta.

Assim sendo, o grupo foi capaz de atingir os objetivos propostos, o que nos deixou satisfeitos com o produto final desenvolvido.

Capítulo 5

Anexos

5.1 Ficheiro BibTeX

Para a realização deste projeto, utilizamos o ficheiro *BibTeX* disponibilizado no endereço `http://di.uminho.pt/~prh/lp.bib`

Este ficheiro irá seguir a tipologia proposta pelo seguinte excerto do ficheiro *.bib*:

```
@techreport{BW83a,
    author = "Manfred Broy and Martin Wirsing",
    title = "Generalized Heterogeneous Algebras and Partial
       Interpretations",
    year = 1983,
    month = Feb,
    institution = "Institut fur Informatik, TUM",
    note = "(draft version)",
    annote = "espec algebrica"
    }

@inbook{Val90a,
    author = "Jos\'e M. Valen\c{c}a",
    title = "Processos, {O}bjectos e {C}omunica\c{c}\~ao
                ({O}p\c{c}\~ao I - {MCC})",
    chapter = 2,
    year = 1990,
    month = Oct,
    publisher = gdcc,
    address = um,
    annote = "programacao oobjectos, proc comunicantes, espec
       formal"
    }
```

5.2 Código Flex

De seguida apresentamos uma cópia do ficheiro *.l* desenvolvido em *Flex*.

```
%{
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char*  nomesCategorias[30]; //guardar categorias
int contCategorias[30]; //guargar o numero de vezes de cada categoria correspondente    posi   o do
      array seguinte

char* a[500];
int n=0;
int tot=0;
int numeroAutores=0;
int numeroNomesAutores[100];
char lixo[5000];

int bool=0;
int titleBOOL=0;
int authorBOOL=0;

char* new[1000];


int nParceiros=0;
char* authorParceria[100];
char* authorProcura;


char* lowerCASE(char * str){
        int i;
        for(i = 0; str[i]!='\0'; i++){
                str[i] = tolower(str[i]);
        }
        return str;
}
void addRegisto(char* nome){
        int found=0;
        int i;
        for(i=0;i<30 && nomesCategorias[i]!=NULL && found==0;i++){
                if(!strcmp(nomesCategorias[i],nome)){
                        found=1;
                        contCategorias[i]++;
                }
        }
        if(nomesCategorias[i]==NULL && found==0){
                nomesCategorias[i] = strdup(nome);
                contCategorias[i]++;
        }
}
char* removeVirgula(char* name){
        if(name[strlen(name)-1]==','){
                name[strlen(name)-1]='\0';
        }
        return name;
}
char* retiraEnter(char* lixo){
        int i = 0;
        while(lixo[0]=='\n'){lixo++;}
        i=strlen(lixo);
        while(lixo[i-1]=='\n'){lixo[i-1]='\0';i--;}

        return lixo;
}
void registaAuthors(){
        char** aux = a;
        char* autor;
        int i,j;
        char buf[100];

        FILE* fp = fopen("graf.dot","a+");
        FILE* fpAuthor = fopen("grafAuthor.dot","a+");

        numeroNomesAutores[numeroAutores]=n;
        for(i=0; i<=numeroAutores;i++){
                if(numeroAutores==0){
                        //0 ou 1 autores
                        if(tot==0){
                                printf("author = { },\n");
                        }else{
                                if(tot==1){
                                        printf("author = { %s },\n",aux[0]);
                                        sprintf(buf,"%s",aux[0]);
                                }else{
                                        if(aux[0][strlen(aux[0])-1]==','){
                                                printf("author = { %c. %s },\n",aux[1][0],
```

```
                                                removeVirgula(aux[0]));
85                                              sprintf(buf,"%c. %s",aux[1][0],removeVirgula(aux[0])
                                                    );
86                                      }else{
87                                              printf("author = { %c. %s },\n",aux[0][0],aux[
                                                    numeroNomesAutores[i]-1]);
88                                              sprintf(buf,"%c. %s",aux[0][0],aux[
                                                    numeroNomesAutores[i]-1]);
89                                      }
90                              }
91                      }
92              }else{
93                      if(i==0){
94                              if(numeroNomesAutores[i]==1){
95                                      printf("author = { \n\t%s and\n",aux[0]);
96                                      sprintf(buf,"%s",aux[0]);
97                              }else{
98                                      if(aux[0][strlen(aux[0])-1]==','){
99                                              printf("author = { \n\t%c. %s and\n",aux[1][0],
                                                    removeVirgula(aux[0]));
100                                             sprintf(buf,"%c. %s",aux[1][0],removeVirgula(aux[0])
                                                    );
101                                     }else{
102                                             printf("author = { \n\t%c. %s and\n",aux[0][0],aux[
                                                    numeroNomesAutores[i]-1]);
103                                             sprintf(buf,"%c. %s",aux[0][0],aux[
                                                    numeroNomesAutores[i]-1]);
104                                     }
105                             }
106
107                     }else{
108                             if(i==numeroAutores){
109                                     if(numeroNomesAutores[i]==1){
110                                             printf("\t%s },\n",aux[0]);
111                                             sprintf(buf,"%s",aux[0]);
112                                     }else{
113                                             if(aux[0][strlen(aux[0])-1]==','){
114                                                     printf("\t%c. %s },\n",aux[1][0],
                                                            removeVirgula(aux[0]));
115                                                     sprintf(buf,"%c. %s",aux[1][0],removeVirgula
                                                            (aux[0]));
116                                             }else{
117                                                     printf("\t%c. %s },\n",aux[0][0],aux[
                                                            numeroNomesAutores[i]-1]);
118                                                     sprintf(buf,"%c. %s",aux[0][0],aux[
                                                            numeroNomesAutores[i]-1]);
119                                             }
120                                     }
121
122                                     }else{
123                                     if(numeroNomesAutores[i]==1){
124                                             printf("\t%s and\n",aux[0]);
125                                             sprintf(buf,"%s",aux[0]);
126                                     }else{
127                                             if(aux[0][strlen(aux[0])-1]==','){
128                                                     printf("\t%c. %s and\n",aux[1][0],
                                                            removeVirgula(aux[0]));
129                                                     sprintf(buf,"%c. %s",aux[1][0],removeVirgula
                                                            (aux[0]));
130                                             }else{
131                                                     printf("\t%c. %s and\n",aux[0][0],aux[
                                                            numeroNomesAutores[i]-1]);
132                                                     sprintf(buf,"%c. %s",aux[0][0],aux[
                                                            numeroNomesAutores[i]-1]);
133                                             }
134                                     }
135
136                                     }
137                             }
138                             new[i]=strdup(buf);
139                             aux=aux+numeroNomesAutores[i];
140                     }
141             }
142
143             for(i=0;i<=numeroAutores;i++){
144                     for(j=0;j<=numeroAutores;j++){
145                             if(i!=j){
146                             fprintf(fp,"\t\"%s\" -- \"%s\";\n",new[i],new[j]);
147                             if(!strcmp(new[i],authorProcura)){
148                                     fprintf(fpAuthor,"\t\"%s\" -- \"%s\";\n",new[i],new[j]);
149                                     authorParceria[nParceiros]=strdup(new[j]);
150                                     nParceiros++;
151                             }
152                     }
153
154                     }
155             }
156             fclose(fp);
157             fclose(fpAuthor);
158     }
159
160     %}
161     %x reservada categoria author title
```

```
%%
^[ ]*@          {BEGIN(reservada);}
<reservada>[A-Za-z]+   {
        printf("\n@@%s",yytext);
        char* aux = lowerCASE(yytext);
        addRegisto(aux);
}

<reservada>\{[^,]+, {
        printf("%s\n",yytext);
        BEGIN(categoria);
        titleBOOL=0;
        authorBOOL=0;
        bool=0;
        lixo[0]='\0';
}

<categoria>[ \n\t][Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*[\"\{]    {
        BEGIN(author);
        numeroAutores=0;
        tot=0;n=0;
        authorBOOL=1;
}

<author>[ ]+[Aa][Nn][Dd][ ]+     {
        numeroNomesAutores[numeroAutores]=n;
        numeroAutores++;
        n=0;
}

<author>[ ]+[Aa][Nn][Dd][ ]*\n?[ ]*              {
        numeroNomesAutores[numeroAutores]=n;
        numeroAutores++;
        n=0;
}

<author>[ ]*\n[ ]*[Aa][Nn][Dd][ ]+               {
        numeroNomesAutores[numeroAutores]=n;
        numeroAutores++;
        n=0;
}

<author>\{[^ ,]+[^ \"\}]\},? {
        a[tot]=strdup(yytext);
        n++;tot++;
}

<author>[^ ,]+([^ \"\},\n]|[^ \"\}],) {
        a[tot]=strdup(yytext);
        n++;tot++;
}

<author>[ ]*[\"\}],?\n  {
        BEGIN(categoria);
        registaAuthors();
        //registaAuthorsGraf();
}

<author>.|\n     {;}
<categoria>[ \n\t][Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*\"    {
        BEGIN(title);
        if(authorBOOL){
                printf("title = {");
        }else{
                strcat(lixo,"title = {");
        }
        titleBOOL=1;
}

<categoria>[ \n\t][Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*\{    {
        BEGIN(title);
        if(authorBOOL){
                printf("title = {");
        }else{
                strcat(lixo,"title = {");
        }
        titleBOOL=1;
}

<title>[^=]*\",? {
        BEGIN(categoria);
        if(yytext[strlen(yytext)-1]==','){
                yytext[strlen(yytext)-2]='}';
        } else{
                yytext[strlen(yytext)-1]='}';
        }
        if(authorBOOL){
                printf("%s\n",yytext);
        }else{
                strcat(yytext,"\n");
                strcat(lixo,yytext);
        }
}
```

```
255
256 <title>[^=]*[^\n]\},? {
257         BEGIN(categoria);
258         if(authorBOOL){
259                 printf("%s",yytext);
260         }else{
261                 strcat(yytext,"\n");
262                 strcat(lixo,yytext);
263         }
264 }
265
266 <categoria>[ ]*=[ ]*\"          {
267         if(titleBOOL && authorBOOL){
268                 printf(" = {");
269         }else{
270                 strcat(lixo," = {");
271         }
272 }
273
274 <categoria>\",          {
275         if(titleBOOL && authorBOOL){
276                 printf("},");
277         }else{
278                 strcat(lixo,"},");
279         }
280 }
281
282 <categoria>\"\n         {
283         if(titleBOOL && authorBOOL){
284                 printf("}");
285         }else{
286                 strcat(lixo,"}");
287         }
288 }
289
290 <categoria>.|\n {
291         if(titleBOOL && authorBOOL ){
292                 if(!bool){
293                         printf("%s%s",retiraEnter(lixo),yytext);
294                         bool=1;
295                 }else{
296                         printf("%s",yytext);
297                 }
298          }else{
299                 strcat(lixo,yytext);
300         }
301 }
302
303 <categoria>\}\n {
304         BEGIN(INITIAL);
305         if(!(titleBOOL && authorBOOL)){
306                 printf("%s",lixo);
307         }
308         printf("} \n");
309 }
310
311 .|\n {;}
312
313 %%
314 void initArrayCategorias(){
315         int i;
316         for(i=0;i<30;i++){
317                 nomesCategorias[i]=NULL;
318                 contCategorias[i]=0;
319         }
320 }
321
322 void gerarHTML(){
323         int i;
324         FILE* fp =fopen("index.html","w");
325
326         fprintf(fp, "<html>\n\t<head>\n\t\t<title>EXERCICIO 1</title>\n\t</head>\n\t<body>\n \t\t<h2
              >Lista Exercico 1</h2>\n\t\t\t<ul>\n");
327
328         for(i=0;i<30 && nomesCategorias[i]!=NULL ;i++){
329
330                 fprintf(fp, "\t\t\t<li> %s -> %d </li>\n", nomesCategorias[i],contCategorias[i]);
331         }
332
333         fprintf(fp, "\t\t\t</ul>\n\t</body>\n</html>\n");
334         fclose(fp);
335 }
336
337 int yywrap() {return 1;}
338 int main(int argc, char** argv){
339         initArrayCategorias();
340
341         if(argc<=1){
342                 authorProcura=strdup("");
343         }else{
344                 authorProcura=argv[1];
345         }
346
```

```
347         FILE* fp = fopen("graf.dot","w");
348         fprintf(fp,"strict graph authors {\n");
349         fclose(fp);
350 
351         FILE* fpAuthor = fopen("grafAuthor.dot","w");
352         fprintf(fpAuthor,"strict graph authors {\n");
353         fclose(fpAuthor);
354 
355 
356         yylex();
357 
358         fp = fopen("graf.dot","a+");
359         fprintf(fp,"}");
360         fclose(fp);
361 
362         fpAuthor = fopen("grafAuthor.dot","a+");
363         fprintf(fpAuthor,"}");
364         fclose(fpAuthor);
365 
366         gerarHTML();
367 
368         system("dot -Tpng graf.dot -o graf.png");
369         system("dot -Tpng grafAuthor.dot -o grafAuthor.png");
370 
371         return 0;
372 }
```