

Processamento de Linguagens
Trabalho Prático 1b
Relatório de Desenvolvimento

Diogo Machado
(A75399)

Lisandra Silva
(A73559)

Rui Leite
(A75551)

23 de Abril de 2017

Resumo

Para a realização deste trabalho foram escolhidos dois dos seis temas propostos, sendo que o presente relatório visa mostrar o processo de desenvolvimento de filtros de texto para o Processador de Inglês corrente e o Normalizador de Autores em $\text{BIB}\text{T}_{\text{E}}\text{X}$, recorrendo ao Gerador FLEX .

O filtro para o Processador de Inglês tem como objetivo gerar um ficheiro resultado em que todas as contrações encontradas são expandidas e ainda criar um ficheiro HTML com todos os verbos não FLEX ionados que encontrar no texto.

Os filtros desenvolvidos para o Normalizador de Autores $\text{BIB}\text{T}_{\text{E}}\text{X}$ têm essencialmente dois objetivos: por um lado tornar os documentos mais uniformes, melhorando a apresentação visual do mesmo, por outro lado extrair informações do documento, como por exemplo saber para cada autor quais os autores com quem ele colaborou.

Conteúdo

Introdução	3
1 Processador de Inglês Corrente	4
1.1 Descrição do Problema	4
1.2 Implementação da Solução	4
1.2.1 Padrões de frases e Expressões Regulares	4
1.2.2 Estruturas de Dados	6
1.2.3 Filtros de Texto	6
1.3 Testes e Análise de Resultados	7
2 Normalizador de Autores em BibTeX	10
2.1 Descrição do Problema	10
2.2 Implementação da Solução	10
2.2.1 Padrões de frases e Expressões Regulares	10
2.2.2 Filtros de texto	12
2.3 Testes e Análise de Resultados	17
2.3.1 Filtro de texto <code>cleaner.1</code>	18
2.3.2 Filtro de texto <code>formatador.1</code>	18
2.3.3 Filtro de texto <code>normalizador.1</code>	19
2.3.4 Filtro de texto <code>processador.1</code>	19
Conclusão e aspetos a melhorar	20
A Código dos Filtros de Texto	21
A.1 Processador de Inglês corrente	21
A.1.1 <code>AlienaA.1</code>	21
A.1.2 <code>AlienaB.1</code>	23
A.2 Normalizador de autores em BibTeX	25
A.2.1 <code>cleaner.1</code>	25
A.2.2 <code>formatador.1</code>	26
A.2.3 <code>normalizador.1</code>	28

A.2.4	processador.1	31
B	Ficheiros de <i>input</i> e de <i>output</i>	35
B.1	Normalizador de Autores em BIB _T E _X	35
B.1.1	Teste.bib	35
B.2	Grafo	37

Introdução

Neste trabalho prático pretende-se aplicar os conceitos adquiridos na Unidade Curricular de Processamento de Linguagens associados à utilização de Expressões Regulares para descrever padrões de frases, de forma a tratar/retirar informação presente num ficheiro. Serão criados filtros de texto que identifiquem esses padrões e procedam à transformação pretendida, com recurso ao Gerador FLEX.

De entre os vários temas propostos para o desenvolvimento de filtros era sugerido que se escolhesse pelo menos um. Tomou-se a decisão de realizar dois dos temas propostos: Processador de Inglês Corrente e Normalizador de Autores em `BIBTeX`.

O Normalizador de Autores em `BIBTeX` foi escolhido como trabalho principal, contudo foram desenvolvidos filtros para tratar os problemas relacionados com o Processador de Inglês Corrente com o objetivo de melhorar os nossos conhecimentos no desenvolvimento de filtros de texto e ainda como valorização deste trabalho prático.

Estrutura do Relatório

O presente relatório encontra-se dividido em dois capítulos: o primeiro referente ao Processador de Inglês Corrente e o segundo ao Normalizador de Autores em `BIBTeX`. Em ambos os capítulos serão apresentados todos os passos seguidos desde a descrição do problema, das decisões tomadas para implementar a solução final e ainda exemplos de utilização e dos resultados obtidos. Em cada um deles será ainda especificado quais os padrões de frases que foram encontrados nos textos-fonte, quais as ações semânticas a realizar como reação ao reconhecimento de cada um desses padrões e ainda as estruturas de dados usadas para armazenar temporariamente a informação.

Capítulo 1

Processador de Inglês Corrente

1.1 Descrição do Problema

Para este problema pretende-se desenvolver filtros de texto em FLEX que leiam um texto corrente em inglês e que permitam:

- reproduzir o texto de entrada na saída, expandindo todas as contrações que encontrar.
- gerar em HTML uma lista ordenada de todos os verbos no infinitivo que encontrar no texto.

Por forma a evitar ambiguidades optou-se por, nas contrações em que surgiram dúvidas, expandir para um conjunto de possíveis palavras e deixar para tratamento manual. No ficheiro HTML gerado, para além dos verbos encontrados, é ainda apresentado o número de vezes que cada verbo ocorreu no ficheiro de *input*.

1.2 Implementação da Solução

Por forma a implementar aquilo a que nos propusemos, realizou-se uma análise ao ficheiro `FAQ-Phyton-EN.txt` disponibilizado com o objetivo de identificar padrões e construir expressões regulares eficazes para obtenção de informação importante. Para uma melhor organização, foram implementados 2 filtros de texto distintos que permitem obter aquilo que é solicitado em cada uma das duas alíneas.

1.2.1 Padrões de frases e Expressões Regulares

Após uma análise cuidada do ficheiro fornecido e a alguns documentos referentes à gramática inglesa foi possível observar um conjunto de contrações, possíveis de encontrar num texto redigido em Inglês. Isto permitiu-nos criar expressões regulares para poder interpretar a informação nele contido.

Grande parte das contrações que existem na língua inglesa não deixa ambiguidade à sua expansão. Para estas, apenas se criou um conjunto de Expressões Regulares para as identificar e fazer a respetiva expansão. De seguida são apresentadas todas as contrações que o filtro de texto é capaz de expandir.

```

"i aren't"          { printf("i am not");          }
"shan't"|"sha'n't" { printf("shall not");        }
"can't"             { printf("can not");          }
"won't"             { printf("will not");         }
"n't"               { printf(" not");            }
"'m"                { printf(" am");             }
"'re"               { printf(" are");            }
"where'd"           { printf("where did");        }
"y'all"             { printf("you all");          }
"y'all'd"           { printf("you all would");    }

```

Contudo, identificaram-se expressões para as quais é impossível garantir a sua correta expansão. Desta forma, foram consideradas três alternativas possíveis para a resolução deste problema:

- não expandir nada e deixar a contração para futuro tratamento manual;
- expandir usando a que achássemos mais frequente;
- expandir para as suas possibilidades e colocar uma etiqueta para posterior tratamento manual.

Perante as hipóteses disponíveis optou-se, por quando a expansão da contração for ambígua, expandir para as suas possibilidades e colocar uma etiqueta para posterior tratamento manual, por exemplo:

```

"d"                 { printf(" [had,would]");      }
"ll"                { printf(" [shall,will]");     }
"it's"              { printf("it [has,is]");       }
"so's"              { printf("so [as,is]");        }
(?i:(i|you|she|he  { token = strtok(yytext,"");
|it|we|they))\'s    { printf("\%s [has,is,does]", yytext); }
([a-zA-Z]+\'[sS])  { printf(" [has,is,does,possesive case]"); }

```

Relativamente aos verbos no infinitivo que surgem ao longo do texto foi possível identificar os seguintes padrões:

- verbos que se encontram precedidos pela palavra *to*
- verbos antecidos por *can, could, shall, should, will, would, may e might*

- verbos precedidos por *do/does* ou *did*, i.e., frases que se encontram na forma interrogativa.

Deste modo, surgiram Expressões Regulares que identificam cada uma destas possíveis palavras precedidas por uma palavra, por exemplo:

```
(to|can|could|shall|should|will|would|may|might)\ [a-zA-Z]+
```

No caso dos verbos precedidos por *do/does* ou *did*, é necessário capturar as duas palavras imediatamente a seguir a cada um destes verbos, originando a seguinte Expressão Regular:

```
(?i:do|does|did)\ [a-zA-Z]+\ [a-zA-Z]+
```

Para garantir que a palavra que pretendemos capturar se trata de um verbo, foi feita a pesquisa dos verbos mais usados em na Língua Inglesa, armazenados no ficheiro *verbos.txt*. Todos esses verbos são guardados numa estrutura, antes de se iniciar o processamento do texto.

1.2.2 Estruturas de Dados

Para gerar o ficheiro HTML foi necessário armazenar os verbos encontrados numa estrutura. Então, decidiu-se que a melhor opção para armazenar esta informação seria uma árvore binária em que a chave é o verbo a armazenar (*char **) e o valor é o número de vezes que esse verbo foi encontrado no ficheiro de *input*. Esta estrutura foi implementada recorrendo ao uso da biblioteca *glib* e revelou-se eficaz no armazenamento de dados.

1.2.3 Filtros de Texto

Nesta subsecção são apresentadas as decisões tomadas para a implementação de cada um dos objetivos pedidos no enunciado do trabalho prático e também apresentados na secção 1.1 deste relatório.

A implementação dos filtros de texto fez-se com a construção dos seguintes ficheiros:

- **AlineaA.1** - para a primeira alínea
- **AlineaB.1** - para a segunda alínea

Alínea a) - Reproduzir o texto de entrada na saída expandindo todas as contrações que encontrar.

Para a implementação deste filtro, conforme especificado na subsecção 1.2.1, foram identificadas todas as possíveis padrões e quais as possíveis ações semânticas a realizar como reação ao reconhecimento de cada um desses padrões. Deste modo, o filtro de texto aplicará essas ações de acordo com o padrão identificado e é apresentando, à saída, o respetivo texto expandido. No fim do ficheiro é ainda apresentado o número de casos ambíguos encontrados por forma a instruir o utilizador de que terá que os tratar de forma manual.

Alínea b) - gerar em HTML uma lista ordenada de todos os verbos

Este filtro de texto foi implementado recorrendo às expressões regulares anteriormente explicitadas. De acordo com a Expressão Regular identificada foi aplicada a respetiva ação.

No caso de se tratar da palavra *to* ou de um dos auxiliares já referidos é feito o *parsing* das palavras capturadas e verificado se a 2^o palavra corresponde de facto a um verbo reconhecido. A ação a aplicar é a seguinte:

```
token = strtok(yytext, " ");
token = strtok(NULL, " ,.\n?");
verb = strdup(token);

if ((aux = g_tree_lookup(verbos, verb)) != NULL) {
    (*(int *)aux)++;
}
```

Caso se trate da palavra *do, does* ou *did*, a ação é semelhante, pois apenas é necessário verificar se a 3^o palavra é de facto um verbo. Vem então o seguinte:

```
token = strtok(yytext, " ");
token = strtok(NULL, " ");
token = strtok(NULL, " ,.\n?");
verb = strdup(token);

if ((aux = g_tree_lookup(verbos, verb)) != NULL) {
    (*(int *)aux)++;
}
```

Depois de percorrido todo o ficheiro de *input*, é feita a criação do ficheiro HTML: *index.html*. Este ficheiro é construído com auxílio da função `criar_HTML()`. Esta função irá percorrer a árvore onde estão armazenados os verbos encontrados, i.e., cujo número de ocorrências é maior do que zero, e para cada nodo dessa árvore irá imprimir o verbo e o número de vezes que este surgiu, recorrendo à função `print_pair_HTML2()`.

1.3 Testes e Análise de Resultados

De seguida são apresentados testes feitos e os respetivos resultados obtidos. Para o teste dos Filtros de Texto produzidos usaram-se os ficheiros fornecidos assim como outros criados pelo grupo de trabalho (em anexo) por forma a mostrar o funcionamento dos mesmos.

Para cada uma das alíneas é apresentado o ficheiro de *input* usado e o respetivo *output* obtido, bem como o regra da *Makefile* usada para a sua obtenção. Foi usado o ficheiro *Exemplo.txt* para a alínea a) e o ficheiro *Exemplo2.txt* para a alínea b).

Alínea a) - Reproduzir o texto de entrada na saída expandindo todas as contrações que encontrar.

EXEMPLO.TXT

You aren't invited.
You can't smoke in here.
We won't regret it.
She didn't remember me.
I'm older can than you.
They're happy to see me.
She'd gone to bed before he called.
They've asked me before.
I'll tell you later.
It's a small world after all.
The number is interpreted using Python's rules.

\$ make AlineaA FILE="Exemplo.txt"

Output

You are not to invited.
You can not to smoke in here.
We will not regret it.
She did not remember me.
I am older can than you.
They are happy to see me.
She [had,would] gone to bed before he called.
They have asked could me before.
I [shall,will] tell you later.
It [has,is,does] a small world after all.
The number is do you interpreted using [has,is,does,
possesive case]
rules.


Verifique casos de dúvida: 4 casos

Alínea b) - gerar em HTML uma lista ordenada de todos os verbos

EXEMPLO2.TXT

I should do my English homework.
You can play the tambourine.
Caroline bought some ingredients to bake a cake.
Alyson works hard in order to earn a lot of money.
Do you want to send this message by e-mail?
Did you finish doing your tests?
They might be interested.
They must study.
He will talk to me later.
I would make a cake for your birthday.
We shall sing a song.
I can't drive today.
You may be right.

\$ make AlineaB FILE="Exemplo2.txt"



Verbo	Número de ocorrências
be	2
do	1
drive	1
earn	1
finish	1
make	1
play	1
send	1
sing	1
talk	1
want	1




Figura 1.1: Página HTML gerada

Capítulo 2

Normalizador de Autores em BibTeX

2.1 Descrição do Problema

Neste problema pretende-se desenvolver filtros de texto em C FLEX que leiam um ficheiro BibTeX que permitam:

- converter todos os caracteres com acentos explícitos em caracteres portugueses
- reescrever todos os campos *author* e *editor*, mantendo os outros inalterados, seguindo sempre a mesma norma em que serão sempre usadas as chavetas
- no caso dos autores e editores, o apelido seja seguido por uma ”,” e as letras iniciais dos restantes nomes seguidas por um ”.”

É ainda solicitado a criação de um grafo com todos os autores que colaboram com um dado autor, usando para isso a linguagem DOTTY e o processador *dot*.

2.2 Implementação da Solução

Para implementar uma solução viável que permita responder ao que é solicitado, foi feita uma análise aos ficheiros disponibilizados: *exemplo-latin1.bib* e *exemplo-utf8.bib*. Constatamos que existe um conjunto de padrões que permite construir expressões regulares eficazes de forma a obter a informação pretendida. Foram então criados quatro filtros de texto, cada um com um propósito bem definido, por forma a obter um ficheiro com o qual seja possível trabalhar e retirar a informação necessária para a criação do grafo pedido.

2.2.1 Padrões de frases e Expressões Regulares

Analizados os ficheiros disponibilizados e alguma documentação relativa a sintaxe da formatação usada em documentos L^AT_EX, foi possível criar um conjunto de expressões regulares e respetivas ações para tornar o ficheiro de entrada mais fácil de interpretar.

Constatou-se que existiam algumas más formatações relativamente às diversas entradas bibliográficas:

1. existem espaços a mais ao longo do texto:

[]+

2. há nomes de autores e editores separados por mais que um *and*:

([]and){2}

3. há mudanças de linha no meio do conteúdo de diversos campos

[]*\n[]*

4. existem espaços a mais no fim do conteúdo de cada campo, que pode terminar com chaveta direita (}) ou aspa (")

[]*(\}|\") ,

5. existem campos de editor e autor vazios:

(?i:author|editor)[]*=[]*\{[]*\},\n

Para além disso e conforme é descrito no enunciado, identificaram-se algumas anomalias na forma como são apresentados os acentos. Por forma a fazer a conversão para o formato habitual da acentuação e também a tornar o ficheiro pronto a proceder às abreviações dos nomes dos autores e editores, foram consideradas as seguintes situações:

6. os acentos poderão surgir com a sintaxe do L^AT_EX, ou seja, com o “*escape*” do caractere do acento seguida da letra onde é aplicado, possivelmente entre chavetas (por exemplo:

{\ ^ a} = â):

\{?\^\a\}?

7. existem apelidos já abreviados, não necessitando de qualquer transformação, a menos que não tenha um espaço a separar mais do que um:

[A-Z]\.[]*

8. existem artigos em alguns dos nomes:

[](do|da|de|das|dos)[]

2.2.2 Filtros de texto

Para a aplicação das ações correspondentes a cada uma dos casos anteriormente explicitados, foram criados quatro filtros de texto, apresentados de seguida.

Filtro de texto cleaner.1

Os espaços em branco deram origem ao primeiro filtro de texto. Para conseguir proceder às normalizações apresentadas de seguida e conforme os padrões já apresentados, foi considerado um estado SUB, que é iniciado sempre que são encontradas as palavras *author* ou *editor*, possivelmente com maiúsculas, seguidas de um sinal de igual (=) e de uma chaveta direita (}), separados por zero ou mais espaços.

```
(?i:author)[ ]*=[ ]*(\{|\") [ ]*      { fprintf(yyout,"author = {");  
                                         BEGIN SUB;                               }  
  
(?i:editor)[ ]*=[ ]*(\{|\") [ ]*      { fprintf(yyout,"editor = {");  
                                         BEGIN SUB;                               }
```

Este filtro de texto recebe como argumento um ficheiro de *input* e escreve o *output* para um ficheiro auxiliar *limpo.txt*:

```
yyin = fopen(argv[1], "r");  
yyout = fopen("limpo.txt", "w+");
```

Foram, então, consideradas as seguintes ações para cada um dos padrões apresentados anteriormente:

- 1./4. Todos os espaços em branco são substituídos por um único espaço, incluindo os que antecedem uma chaveta direita (}) ou uma aspa ("):

```
<SUB>[ ]+          { fprintf(yyout," ");      }  
<SUB>[ ]*(\}|\") , { fprintf(yyout,"},");  
                  BEGIN INITIAL; }
```

2. Quando houver mais do que um “and” no estado SUB, estes são substituídos por um único:

```
<SUB>([ ]and){2}    { fprintf(yyout," and"); }
```

3. Todos os “new lines” encontrados no estado SUB são eliminados:

```
<SUB>[ ]*\n[ ]*      { fprintf(yyout, " ");      }
```

5. Os campos autor e editor que não contenham nenhum nome são eliminados:

```
(?i:author|editor)[ ]*=[ ]*\{[ ]*\},\n      {      ;      }
```

- tudo o resto é imprimido no ficheiro de *output*

```
<*>.\n      {      fprintf(yyout,"%s",yytext);      }
```

É iniciado o estado `INITIAL` sempre que, no estado `SUB`, for encontrada uma chaveta direita ou uma aspa, seguida de uma vírgula (`}`, ou `"`,).

```
<SUB>[ ]*(\}|\\"),      {      fprintf(yyout, "},");  
                          BEGIN INITIAL;                          }
```

Filtro de texto formatador.1

Depois de limpos todos os espaços em branco (incluindo *new lines*), chegou-se à conclusão que a próxima ação a tomar deve consistir, essencialmente, na formatação de todos os acentos. Para tal foi considerado o mesmo estado `SUB`, iniciado quando encontrado o campo *author* ou *editor* no texto e terminado quando o campo fecha com a chaveta direita seguida de virgula (`}`,).

```
(author|editor)[ ]=[ ]\{      {      fprintf(yyout,"%s",yytext);  
                                BEGIN SUB;                                }  
<SUB>\},      {      fprintf(yyout,"%s",yytext);  
                                BEGIN INITIAL;                                }
```

Este filtro de texto recebe como argumento o ficheiro de *input* do filtro anterior (`limpo.txt`) e escreve o *output* para um ficheiro auxiliar `formatado.txt`:

```
yyin = yyin = fopen(argv[1], "r");  
yyout = fopen("formatado.txt", "w+");
```

Foram, então, consideradas as seguintes ações:

6. todos os acentos com a sintaxe do \LaTeX são substituídos pelo respetivo caractere em português, assim como o *cê* de cedilha ($\text{\c{c}}$):

```
<SUB>\{?\\"'a\}?      {      fprintf(yyout, "á");      }  
<SUB>\{?\\"'e\}?      {      fprintf(yyout, "é");      }  
<SUB>\{?\\"'i\}?      {      fprintf(yyout, "í");      }  
<SUB>\{?\\"'o\}?      {      fprintf(yyout, "ó");      }  
<SUB>\{?\\"'u\}?      {      fprintf(yyout, "ú");      }  
  
<SUB>\{?\\"'A\}?      {      fprintf(yyout, "Á");      }
```

```

<SUB>\{?\\"'E\}?      {   fprintf(yyout, "É");           }
<SUB>\{?\\"'I\}?      {   fprintf(yyout, "Í");           }
<SUB>\{?\\"'O\}?      {   fprintf(yyout, "Ó");           }
<SUB>\{?\\"'U\}?      {   fprintf(yyout, "Ú");           }

<SUB>\{?\\"~a\}?      {   fprintf(yyout, "ã");           }
<SUB>\{?\\"~o\}?      {   fprintf(yyout, "õ");           }

<SUB>\{?\\"^a\}?      {   fprintf(yyout, "â");           }
<SUB>\{?\\"^e\}?      {   fprintf(yyout, "ê");           }
<SUB>\{?\\"^o\}?      {   fprintf(yyout, "ô");           }

<SUB>\{?\\"^A\}?      {   fprintf(yyout, "Â");           }
<SUB>\{?\\"^E\}?      {   fprintf(yyout, "Ê");           }
<SUB>\{?\\"^O\}?      {   fprintf(yyout, "Ô");           }

<SUB>\{?\\"c\{c\}\}\?  {   fprintf(yyout, "ç");           }

```

7. sempre que a seguir a uma letra seguida de um ponto, i.e., uma apelido já abreviado, não exista um espaço, é inserido um, a menos que se encontre no final do campo *author/editor*, i.e., seguido de uma chaveta direita e de uma vírgula:

```

<SUB>[A-Z]\.\},      {   fprintf(yyout,"%c.}",yytext[0]);
                        BEGIN INITIAL;                               }

<SUB>[A-Z]\.[ ]*     {   fprintf(yyout,"%c. ",yytext[0]);      }

```

8. todos os artigos que separem nomes ou apelidos são retirados e é colocado um espaço no seu lugar:

```

<SUB>[ ](do|da|de|das|dos)[ ]   {   fprintf(yyout, " ");   }

```

- tudo o resto é imprimido no ficheiro de *output*

```

<*>.\|\\n           {   fprintf(yyout,"%s",yytext);      }

```


Filtro de texto normalizador.1

Depois de obtido o ficheiro com a formatação pretendida procedeu-se à normalização dos nomes dos autores e editores. Para tal recorreram-se aos estados LINHA e COMPRESSAO:

- LINHA - uma vez que se pretende que apenas os autores e os editores sigam a norma apresentada no enunciado procedeu-se à criação do estado LINHA. Este é iniciado sempre que é encontrada a expressão regular "author = { | editor = {". No estado LINHA vai-se proceder à análise e transformação dos nomes de cada linha, isto é, até encontrar o caractere "}" que indica o fim da linha.

Para se proceder à normalização dos nomes, no estado LINHA, sempre que se encontra um nome, que pode ou não ser seguido por um ponto, este é guardado num *array* de *strings*. Caso seja encontrada a palavra "and" ou "}" então o último nome guardado no *array* é imprimido no ficheiro seguido de uma vírgula (,), percorrendo-se posteriormente o *array* de strings e imprimindo apenas o primeiro caractere de cada nome seguido de um ponto (.).

No caso de o nome encontrado ser seguido de uma virgula significa que o nome já se encontra na forma pretendida e então bastará comprimir todos os nomes que forem encontrados de seguida, sendo iniciado para tal o estado COMPRESSAO.

Foram ainda tratados alguns casos especiais, como exemplo: quando o nome de um conjunto de autores está na forma "First-Name Last-Name et al." ou quando nomes que não correspondem a pessoas, tais como "Iberâmia 2014", e que não devem sofrer qualquer alteração.

Estando no estado COMPRESSAO e sendo encontrado o caractere "}" é iniciado o estado INITIAL, pois significa que a linha do autor ou editor já terminou.

- COMPRESSAO - neste estado, sempre que é encontrada uma palavra, ela é comprimida para a sua inicial, seguida de um ponto (.). São também tratados casos especiais, como exemplo: quando a inicial no nome é um caractere com um acento este é substituído pelo respetivo caractere sem qualquer acentuação.

Neste estado, quando é encontrada a expressão regular []and[] é iniciado o estado LINHA, para se proceder à análise do nome seguinte. No caso de se encontrar o caractere "}" então significa que já não existem mais nomes para analisar e é iniciado o estado INITIAL.

Este filtro de texto recebe como argumento o ficheiro de *input* do filtro anterior (*formatado.txt*) e escreve o *output* para um ficheiro auxiliar *normalizado.txt*:

```
yyin = yyin = fopen(argv[1], "r");  
yyout = fopen("normalizado.txt", "w+");
```

Filtro de texto `processador.1`

O objetivo agora será construir um grafo que tenha todos os autores que colaboram com um determinado autor, tal como é proposto no enunciado, usando para tal a linguagem `Dotty` e o processador `dot`.

Nesta fase é suposto já termos um ficheiro formatado e com todos os nomes normalizados segundo o formato pretendido. Assim, o processo de encontrar os colaboradores de um determinado autor passa por analisar todas as linhas do campo *author* e, caso essa linha contenha o autor para o qual se procuram os colaboradores, então todos os outros autores são armazenados numa estrutura de dados. A estrutura de dados escolhida para o armazenamento dos nomes encontrados foi uma `GTree` (da `Glib`), pois além de ser uma estrutura eficiente de procura também garante que não sejam armazenados nomes repetidos.

Para a implementação do `processador.1` decidiu-se que seria conveniente ter um estado `NOME` que é iniciado sempre que, estando no estado `INITIAL`, é encontrada a expressão regular: `author\ =\ {`.

Este filtro de texto recebe como argumento o ficheiro de *input* do filtro anterior (`normalizado.txt`) e escreve o *output* para um ficheiro que contém a informação suficiente para ser gerado um grafo: `grafo.dot`:

```
yyin = yyin = fopen(argv[1], "r");
yyout = fopen("grafo.dot", "w+");
```

De seguida apresentam-se as expressões regulares esperadas no estado `NOME` e qual o respetivo processamento da informação filtrada.

1. `<NOME>[A-Z][a-zA-Zá-úãõÁ-ÚâêôÂÊÔç]+\,`
sempre que são encontradas uma ou mais palavras antecidas de uma vírgula, as mesmas são guardadas num `char **nomes`. Esta expressão permite guardar o apelido de um determinado nome.
2. `<NOME>\ ?[a-zA-Zá-úãõÁ-ÚâêôÂÊÔç]+\.`
esta expressão surge depois da anterior e permite capturar as contrações dos nomes na forma “X.”, por exemplo. Permite ainda capturar situações excecionais tais como “et” e “al.”. Também as expressões encontradas aqui são armazenadas no “`char **nomes`”.
3. `<NOME>\ [0-9]+`
também se forem encontrados números, os mesmos são armazenados no mesmo `char **nomes`.
4. `<NOME>\ and\`
esta expressão regular vem antes de todas as outras e quando é encontrada significa que se chegou ao fim da filtragem de um nome. Assim, procede-se à concatenação de todos os nomes armazenados no `char **nomes` numa única string `char *nomeTodo`. De

seguida essa *string*, que contém o nome todo, é armazenada num *array* de *strings* `char **nomesLinha`.

5. `<NOME>\ }`,

sempre que se encontrada esta expressão regular significa que chegamos ao final da linha e, por isso, devemos proceder ao seu processamento. Para tal, implementamos um ciclo que verifica se o nome do autor recebido como argumento se encontra no `char **nomesLinha` e, em caso afirmativo, percorrem-se todos os nomes armazenados no `char **nomeLinha` e armazenam-se na `GTree nomes`, que contém os colaboradores do nome recebido.

Por último, depois de ter sido feita a análise léxica do ficheiro e de os nomes pretendidos estarem armazenados na `GTree nomes`, procede-se à elaboração do ficheiro que irá permitir a construção do grafo. Para tal, implementou-se uma função designada por `geraGrafo()` que percorre todos os nodos da `GTree` e imprime num ficheiro a informação armazenada de acordo a linguagem Dotty por forma a permitir a geração do grafo.

2.3 Testes e Análise de Resultados

Para cada um dos filtros desenvolvidos e já apresentados neste relatório, foi passado como argumento um ficheiro de teste e analisado o *output* correspondente. O ficheiro de *output* do filtro de texto `cleaner.1` é usado como ficheiro de *input* no filtro de texto `formatador.1`, e assim sucessivamente até se obter o ficheiro final.

NOTA: Dado o tamanho dos ficheiros, decidiu-se incluir nesta apresentação de resultados apenas parte dos ficheiros. O ficheiro de *input* inicial pode ser consultado no anexo B.1.1.

TESTE.BIB

```
author={F. M\'ario Martins and J.J. Almeida and P.R. Henriques},
author ={{projecto Camila}},
editor = {L.S. Barbosa and J.J. Almeida and J.N. Oliveira and Luís Neves},
author = "J.J. Almeida and Barbosa, L.S. and Neves, F.L. and Oliveira, J.N.",
editor = "De Giusti, A. and Diaz, J. and Pesado, P.",
author={J.C. Ramalho and and J.J. Almeida and P.R. Henriques},
author = "Alberto Manuel Sim{\~o}es and J.J. Almeida",
editor = "Sely Costa et al.",
author = {Alberto Manuel Sim\~oes and J.J. Almeida
and Lu{\'}s Cabral},
editor = {IBERAMIA 2004},
author = {Leonor Barroca and Pedro Rangel Henriques},
editor = {},
```

2.3.1 Filtro de texto cleaner.1

LIMPO.TXT

```
author = {F. M\'ario Martins and J.J. Almeida and P.R. Henriques},
author = {projecto Camila},
editor = {L.S. Barbosa and J.J. Almeida and J.N. Oliveira and Lu\'is Neves},
author = {J.J. Almeida and Barbosa, L.S. and Neves, F.L. and Oliveira, J.N.},
editor = {De Giusti, A. and Diaz, J. and Pesado, P.},
author = {J.C. Ramalho and J.J. Almeida and P.R. Henriques},
author = {Alberto Manuel Sim\~oes and J.J. Almeida},
editor = {Sely Costa et al.},
author = {Alberto Manuel Sim\~oes and J.J. Almeida and Lu\{'i}s Cabral},
editor = {IBERAMIA 2004},
author = {Leonor Barroca and Pedro Rangel Henriques},
```

2.3.2 Filtro de texto formatador.1

FORMATADO.TXT

```
author = {F. M\'ario Martins and J. J. Almeida and P. R. Henriques},
author = {projecto Camila},
editor = {L. S. Barbosa and J. J. Almeida and J. N. Oliveira and Lu\'is Neves},
author = {J. J. Almeida and Barbosa, L. S. and Neves, F. L. and Oliveira, J.
↪ N.},
editor = {De Giusti, A. and Diaz, J. and Pesado, P.},
author = {J. C. Ramalho and J. J. Almeida and P. R. Henriques},
author = {Alberto Manuel Sim\~oes and J. J. Almeida},
editor = {Sely Costa et al.},
author = {Alberto Manuel Sim\~oes and J. J. Almeida and Lu\'is Cabral},
editor = {IBERAMIA 2004},
author = {Leonor Barroca and Pedro Rangel Henriques},
```

2.3.3 Filtro de texto normalizador.1

NORMALIZADO.TXT

```
author = {Martins, F. M. and Almeida, J. J. and Henriques, P. R.},
author = {Camila, p.},
editor = {Barbosa, L. S. and Almeida, J. J. and Oliveira, J. N. and Neves,
↪ L.},
author = {Almeida, J. J. and Barbosa, L. S. and Neves, F. L. and Oliveira, J.
↪ N.},
editor = {De Giusti, A. and Diaz, J. and Pesado, P.},
author = {Ramalho, J. C. and Almeida, J. J. and Henriques, P. R.},
author = {Simões, A. M. and Almeida, J. J.},
editor = {Costa, S. et al.},
author = {Simões, A. M. and Almeida, J. J. and Cabral, L.},
editor = {IBERAMIA 2004},
author = {Barroca, L. and Henriques, P. R.},
```

2.3.4 Filtro de texto processador.1

O ficheiro de *output* pode ser consultado no anexo B.2.

Conclusão e aspetos a melhorar

A realização deste trabalho permitiu-nos não só consolidar os conhecimentos em FLEX mas também concluir que esta é uma ferramenta muito eficaz para filtrar e tratar informação.

Na realização do trabalho sobre a Normalização de Inglês Corrente, as principais dificuldades prenderam-se em saber como expandir as formas verbais no caso de haver mais do que uma possibilidade, e portanto decidimos apresentar as várias opções de expansão deixando ao critério do utilizador escolher manualmente a opção mais adequada. Outra dificuldade encontrada prendeu-se com o fasto de filtrar os verbos e distinguir os mesmo de outras palavras que não fossem verbos. Para resolver este problema a alternativa que nos pareceu mais adequada passou por armazenar numa estrutura de dados os verbos mais utilizados na língua inglesa e, sempre que fosse encontrada uma palavra que pudesse ser um verbo, verificar se a mesma se encontrava na "lista de verbos". Tal poderia ser um aspeto a melhorar uma vez que poderão existir verbos não reconhecidos.

Na realização do trabalho sobre o Normalizador de Autores em BibTEX constatámos que teria sido possível alcançar os objetivos pretendidos fazendo menos vezes a leitura e tratamento do ficheiro de entrada, no entanto concluímos que para tal o uso de expressões regulares seria absurdamente simplificado, na medida em que poderíamos apenas filtrar as linhas dos autores e editores e fazer a manipulação da informação com código C. No entanto, concluímos que seria mais produtivo no âmbito da disciplina filtrar mais vezes o ficheiro e tirar o maior partido das Expressões Regulares, reduzindo a implementação de código C.

Em ambos os trabalhos identificamos a necessidade de fazer uma escolha adequada da estrutura de dados na qual foram armazenados os dados, já que esta escolha influencia a complexidade e eficácia do programa. Além disso, também concluímos que a experiência em manipular expressões regulares foi crescendo ao longo do desenvolvimento do trabalho e permitiu-nos ir simplificando a implementação dos objetivos que haviam sido definidos.

Apêndice A

Código dos Filtros de Texto

A.1 Processador de Inglês corrente

A.1.1 AlienaA.1

```
%{  
    #include <string.h>  
    #define RED    "\x1B[31m"  
    #define RESET "\x1B[0m"  
  
    int duvidas = 0;  
%}  
  
%option noyywrap  
  
%%  
    char* token;  
  
"i aren't"                { printf("i am not"); }  
  
"shan't"|"sha'n't"        { printf("shall not"); }  
"can't"                   { printf("can not"); }  
"won't"                   { printf("will not"); }  
"n't"                     { printf(" not"); }  
  
" 'm"                      { printf(" am"); }  
  
" 're"                     { printf(" are"); }  
  
"where'd"                 { printf("where did"); }  
  
"y'all"                   { printf("you all"); }
```

```

"y'all'd"          { printf("you all would"); }

"how'd"            { printf("how did"); }
"how'd'y"          { printf("how do you"); }
"'d've"            { printf(" would have"); }
"'d"               { printf(" [had,would]"); duvidas++; }
"'ve"              { printf(" have"); }

"how'll"           { printf("how will"); }
"'ll"              { printf(" [shall,will]"); duvidas++; }

"let's"            { printf("let us"); }
"it's"             { printf("it [has,is]"); duvidas++; }
"so's"             { printf("so [as,is]"); }

(?i:(i|you|she|he|it|we|they))\'s { token = strtok(yytext,"");
                                   printf("%s [has,is,does]", yytext);
                                   duvidas++; }

([a-zA-Z]+\'[sS]) { printf(" [has,is,does,possesive
→ case]");
                   duvidas++; }

"o'clock"          { printf("of the clock"); }
%%

int main(int argc, char **argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
    }
    yylex();
    printf("%s-----\n", RED);
    printf("\tVerifique casos de dúvida: %d casos\n", duvidas);
    printf("-----%s\n", RESET);
    return 0;
}

```


A.1.2 AlienaB.1

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

    GTree * verbos;
}%

%option noyywrap

%%
    char * verb;
    char * token;
    gpointer aux = NULL;

(to|can|could|shall|should|will|would|may|might)\ [a-zA-Z]+ {
    token = strtok(yytext, " ");
    token = strtok(NULL, " ,.\n?");
    verb = strdup(token);
    if ((aux = g_tree_lookup(verbos, verb)) != NULL) {
        (*(int *)aux)++;
    }
}

(?i:do|does|did)\ [a-zA-Z]+\ [a-zA-Z]+ {
    token = strtok(yytext, " ");
    token = strtok(NULL, " ");
    token = strtok(NULL, " ,.\n?");
    verb = strdup(token);
    if ((aux = g_tree_lookup(verbos, verb)) != NULL) {
        (*(int *)aux)++;
    }
}

(to|can|could|shall|should|will|would|may|might)(\ not)?\ [a-zA-Z]+ {
    token = strtok(yytext, " ");
    token = strtok(NULL, " ");
    token = strtok(NULL, " ,.\n?");
    verb = strdup(token);
    if ((aux = g_tree_lookup(verbos, verb)) != NULL) {
        (*(int *)aux)++;
    }
}
```

```

}

(to|can|could|shall|should|will|would|may|might)(n)?(\\'t)?\\ [a-zA-Z]+ {
    token = strtok(yytext, " ");
    token = strtok(NULL, " ,.\\n?");
    verb = strdup(token);
    if ((aux = g_tree_lookup(verbos, verb)) != NULL) {
        (*(int *)aux)++;
    }
}

.|\\n { ;}
%%

gboolean print_pair_HTML2(gpointer k, gpointer v, gpointer d) {
    if ((* (int *)v) > 0) {
        fprintf(yyout, "<tr><td>%s</td><td>%d</td></tr>", (char*)k, (*(int
            ↪ *)v));
    }
    return FALSE;
}

void criar_HTML() {
    char *title = "Verbos encontrados";
    fprintf(yyout, "<html> <head> <meta charset = 'UTF-8' /> <center> <title> %s
        ↪ </title> </head>", title);
    fprintf(yyout, "<body> <h1> %s </h1>", title);
    fprintf(yyout, "<body background=\\\"https://previews.123rf.com/images/
        etiamos/etiamos1206/etiamos120600134/14180784-Background-made-of-
        papers-with-colorful-letters-Stock-Vector-school-
        background-doodle.jpg\\\">");
    fprintf(yyout, "<table border=\\\"1\\\" width=\\\"500\\\" height=\\\"150\\\"
        ↪ bordercolor=\\\"green\\\" align=\\\"center\\\">");
    fprintf(yyout, "<tr><th>Verbo</th><th>Número de ocorrências</th></tr>");
    g_tree_foreach(verbos, print_pair_HTML2, NULL);
    fprintf(yyout, "</center></body></html>");
}

int compareInt (gconstpointer a, gconstpointer b) {
    int *aa = (int *)a;
    int *bb = (int *)b;
    int r;
    if (*aa == *bb) {
        return 0;
    } else {

```

```

        return (*aa - *bb);
    }
}

int main (int argc, char* argv[]) {
    verbos = g_tree_new((GCompareFunc)compareInt);
    char * verbo;
    char * buffer;
    size_t len;
    FILE * vrb = fopen("verbos.txt", "r");
    int read;
    while((read = getline(&buffer, &len, vrb)) != -1){
        buffer[strlen(buffer) - 1] = '\0';
        verbo = strdup(buffer);
        gint* j = g_slice_alloc(sizeof(gint));
        *j = 0;
        g_tree_insert(verbos, verbo, j);
    }
    yyin = fopen(argv[1], "r");
    yylex();
    yyout = fopen("index.html", "w+");
    criar_HTML();
    return 0;
}

```

A.2 Normalizador de autores em BibTeX

A.2.1 cleaner.l

```

%{
    #include <string.h>
}%

%option noyywrap
%x SUB

%%
<INITIAL>(?!:author|editor)[ ]*=[ ]*\{[ ]*\},\n          {      ;      }

<INITIAL>(?!:author)[ ]*=[ ]*(\{|\\") [ ]*      {
    fprintf(yyout, "author = {");
    BEGIN SUB;
}

```

```

<INITIAL>(?!:editor)[ ]*=[ ]*(\{|\\") [ ]*    {
    fprintf(yyout, "editor = {");
    BEGIN SUB;
}

<SUB>[ ]*(\\}|\\"),          {    fprintf(yyout, "},");
                                BEGIN INITIAL;          }

<SUB>([ ]and){2}             {    fprintf(yyout, " and");          }

<SUB>[ ]*\n[ ]*              {    fprintf(yyout, " ");            }

<SUB>[ ]+                    {    fprintf(yyout, " ");            }

<*>.|\\n                     {    fprintf(yyout, "%s", yytext);    }
%%

int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
    }
    yyout = fopen("limpo.txt", "w+");
    yylex();
    fclose(yyout);
    return 0;
}

```

A.2.2 formatador.l

```

%{
    #include <string.h>
}%

%option noyywrap
%x SUB

%%

<INITIAL>(author|editor)[ ]=[ ]\\{    {
    fprintf(yyout, "%s", yytext);
    BEGIN SUB;
}

<SUB>\\{?\\\'a\\}?                {    fprintf(yyout, "á");          }
<SUB>\\{?\\\'e\\}?                {    fprintf(yyout, "é");          }
<SUB>\\{?\\\'i\\}?                {    fprintf(yyout, "í");          }

```

```

<SUB>\{?\\"'o\}?      {    fprintf(yyout, "ó");      }
<SUB>\{?\\"'u\}?      {    fprintf(yyout, "ú");      }

<SUB>\{?\\"'A\}?      {    fprintf(yyout, "Á");      }
<SUB>\{?\\"'E\}?      {    fprintf(yyout, "É");      }
<SUB>\{?\\"'I\}?      {    fprintf(yyout, "Í");      }
<SUB>\{?\\"'O\}?      {    fprintf(yyout, "Ó");      }
<SUB>\{?\\"'U\}?      {    fprintf(yyout, "Ú");      }

<SUB>\{?\\"~a\}?      {    fprintf(yyout, "ã");      }
<SUB>\{?\\"~o\}?      {    fprintf(yyout, "õ");      }

<SUB>\{?\\"^a\}?      {    fprintf(yyout, "â");      }
<SUB>\{?\\"^e\}?      {    fprintf(yyout, "ê");      }
<SUB>\{?\\"^o\}?      {    fprintf(yyout, "ô");      }

<SUB>\{?\\"^A\}?      {    fprintf(yyout, "Â");      }
<SUB>\{?\\"^E\}?      {    fprintf(yyout, "Ê");      }
<SUB>\{?\\"^O\}?      {    fprintf(yyout, "Ô");      }

<SUB>\{?\\"c\{c\}\}\}? {    fprintf(yyout, "ç");      }

<SUB>[A-Z]\.\.,      {    fprintf(yyout, "%c.},", yytext[0]);
                        BEGIN INITIAL;      }

<SUB>[A-Z]\.[ ]*      {    fprintf(yyout, "%c. ", yytext[0]);      }

<SUB>\},              {    fprintf(yyout, "%s", yytext);
                        BEGIN INITIAL;      }

<SUB>[ ](do|da|de|das|dos)[ ]      {    fprintf(yyout, " ");      }

<*>.\|\\n            {    fprintf(yyout, "%s", yytext);      }
%%

int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
    }
    yyout = fopen("formatado.txt", "w+");
    yylex();
    fclose(yyout);
    return 0;
}

```

A.2.3 normalizador.l

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

%}

%option noyywrap
%x LINHA COMPRESSAO

%%
    char * token;
    char * nome[128];
    int i=0;

<INITIAL>author\ =\ \{ |
editor\ =\ \{      {
    fprintf(yyout,"%s",yytext);
    BEGIN LINHA;
}

<LINHA>[0-9]+\ and\      {
    for(int j=0; j<i ; j++) {
        fprintf(yyout,"%s",nome[j]);
    }
    fprintf(yyout,"%s",yytext);
    i=0;
}

<LINHA>[0-9]+\},      {
    for(int j=0; j<i ; j++) {
        fprintf(yyout,"%s",nome[j]);
    }
    fprintf(yyout,"%s",yytext);
    i=0;
    BEGIN INITIAL;
}

<LINHA>\ *and\ *      {
    nome[i-1][strlen(nome[i-1])-1] = '\0';
    fprintf(yyout,"%s",nome[i-1]);
    for(int j=0; j<i-1 ; j++) {
        fprintf(yyout," %c.",nome[j][0]);
    }
    fprintf(yyout," and ");
}
```

```

        i=0;
    }

<LINHA>et\ al\.\ and\ {
    nome[i-1][strlen(nome[i-1])-1] = '\0';
    fprintf(yyout,"%s",nome[i-1]);
    for(int j=0; j<i-1 ; j++) {
        fprintf(yyout," %c.",nome[j][0]);
    }
    fprintf(yyout," %s",yytext);
    i=0;
}

<LINHA>et\ al\.\}, {
    nome[i-1][strlen(nome[i-1])-1] = '\0';
    fprintf(yyout,"%s",nome[i-1]);
    for(int j=0; j<i-1 ; j++) {
        fprintf(yyout," %c.",nome[j][0]);
    }
    fprintf(yyout," %s",yytext);
    i=0;
    BEGIN INITIAL;
}

<LINHA>[A-Z]\.\ {
    nome[i] = strdup(yytext);
    i++;
}

<LINHA>[a-zA-Zá-úãõÁ-ÚâêôÂÊÔÇ]+\ {
    nome[i] = strdup(yytext);
    i++;
}

<LINHA>[a-zA-Zá-úãõÁ-ÚâêôÂÊÔÇ]+\, {
    for(int j=0; j<i ; j++) {
        fprintf(yyout,"%s",nome[j]);
    }
    fprintf(yyout,"%s",yytext);
    BEGIN COMPRESSAO;
}

<LINHA>[a-zA-Zá-úãõÁ-ÚâêôÂÊÔÇ]+\}, {
    token = strtok(yytext,"}");

```

```

    nome[i] = strdup(token);
    fprintf(yyout, "%s", nome[i]);
    for(int j=0; j<i ; j++) {
        fprintf(yyout, " %c.", nome[j][0]);
    }
    fprintf(yyout, "},");
    i=0;
    BEGIN INITIAL;
}

<COMPRESSAO>[A-Z]\.    {
    fprintf(yyout, "%c.", yytext[0]);
}

<COMPRESSAO>Á[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Ê[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Í[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Ô[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Ú[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Â[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Ê[a-z]+    {
    fprintf(yyout, "A.");
}

<COMPRESSAO>Û[a-z]+    {
    fprintf(yyout, "A.");
}

```



```

<COMPRESSAO>[a-zA-Zá-úãõÁ-ÚâêôÊËÔç]+ {
    fprintf(yyout,"%c.",yytext[0]);
}

<COMPRESSAO>\ and\ {
    fprintf(yyout,"%s",yytext);
    i=0;
    BEGIN LINHA;
}

<COMPRESSAO>\}, {
    fprintf(yyout,"%s",yytext);
    i=0;
    BEGIN INITIAL;
}

<*>.\|\\n { fprintf(yyout,"%s",yytext); }
%%

int main(int argc, char** argv) {
    if(argc==2) {
        yyin = fopen(argv[1],"r");
    }
    yyout = fopen("normalizado.txt","w+");
    yylex();
    fclose(yyout);
    return 0;
}

```

A.2.4 processador.l

```

%{
    #include <string.h>
    #include <glib.h>
    #include <unistd.h>
    #include <stdio.h>

    GTree * nomesTree;
    char * recebido;
    int counter = 1;
    gpointer aux = NULL;
}%

%option noyywrap

```

```

%x NOME

%%
char * nome[20];
char * nomeTodo;
char * nomesLinha[128];
int i = 0;
int n = 0;

<INITIAL>author\ =\ \{    {
    BEGIN NOME;
}

<NOME>\ and\    {
    nomeTodo = strdup(nome[0]);
    for (int j = 1; j < i ; j++) {
        strcat(nomeTodo,nome[j]);
    }
    nomesLinha[n] = strdup(nomeTodo);
    n++;
    i = 0;
}

<NOME>\},    {
    nomeTodo = strdup(nome[0]);
    for (int j = 1; j < i; j++) {
        strcat(nomeTodo,nome[j]);
    }
    nomesLinha[n] = strdup(nomeTodo);
    i = 0;
    for (int j = 0; j <= n; j++) {
        if (strcmp(nomesLinha[j],recebido) == 0) {
            for (int k = 0; k <= n; k++) {
                if (k != j) {
                    if ((aux = g_tree_lookup(nomesTree,nomesLinha[k])) !=
                        NULL) {
                        (*(int *)aux)++;
                    } else {
                        int *j = (int *)malloc(sizeof(int));
                        *j = 1;
                        g_tree_insert(nomesTree, nomesLinha[k], j);
                    }
                }
            }
        }
    }
    j = n+1; //Para sair do ciclo

```

```

    }
}
n = 0;
BEGIN INITIAL;
}

<NOME>[A-Z][a-zA-Zá-úãõÁ-ÚäêôÂÊÔç ]+\, {
    nome[i] = strdup(yytext);
    i++;
}

<NOME>\ ?[a-zA-Zá-úãõÁ-ÚäêôÂÊÔç ]+\.? {
    nome[i] = strdup(yytext);
    i++;
}

<NOME>\ [0-9]+\ {
    nome[i] = strdup(yytext);
    i++;
}

<*>.\|\\n { ; }
%%

gboolean print_name(gpointer k, gpointer v, gpointer d) {
    fprintf(yyout, "\tColaborador%d [label=\"%s\"];\n", counter, k);
    int vezes = *(int *)v;
    if (vezes > 1) {
        fprintf(yyout, "\tAutor -> Colaborador%d [label=\"Colaborou %d  

        ↳ vezes\"];\n", counter, vezes);
    } else {
        fprintf(yyout, "\tAutor -> Colaborador%d [label=\"Colaborou %d  

        ↳ vez\"];\n", counter, vezes);
    }
    counter++;
    return FALSE;
}

void geraGrafo() {
    fprintf(yyout, "digraph Colaboradores {\n");
    fprintf(yyout, "\t\tAutor [shape=polygon, sides=6, peripheries=2,  

    ↳ color=lightblue, label=\"%s\"];\n", recebido);
    g_tree_foreach(nomesTree, print_name, NULL);
    fprintf(yyout, "}");
}

```

```

}

int main(int argc, char** argv) {
    if (argc > 2) {
        yyin = fopen(argv[1], "r");
        recebido = strdup(argv[2]);
        for(int i=3; i<argc; i++) {
            strcat(recebido, " ");
            strcat(recebido, strdup(argv[i]));
        }
    }
    nomesTree = g_tree_new((GCompareFunc)strcmp);
    yylex();
    yyout = fopen("grafo.dot", "w+");
    geraGrafo();
    return 0;
}

```

Apêndice B

Ficheiros de *input* e de *output*

B.1 Normalizador de Autores em BibT_EX

B.1.1 Teste.bib

```
@string{ um="Universidade do Minho" }
@string{ umdi="Universidade do Minho, Departamento de Informática" }

@inproceedings{graminteractivas1990,
  author={F. M\ 'ario Martins and J.J. Almeida and P.R. Henriques},
  note = {(Gramáticas Interactivas guardadas)},
  booktitle={3$º$ Encontro Português de Computação Gráfica},
}

@techreport{Camila,
  author ={{projecto Camila}},
  editor ={L.S. Barbosa and J.J. Almeida and J.N. Oliveira and Luís Neves},
  institution = umdi,
}

@inproceedings{ABN097a,
  author = "J.J. Almeida and Barbosa, L.S. and Neves, F.L. and Oliveira,
    ↪ J.N.",
  editor = "De Giusti, A. and Diaz, J. and Pesado, P.",
  year = 1997,
  address = "La Plata, Argentina"
}

@inproceedings{museums98,
  author={J.C. Ramalho and and J.J. Almeida and P.R. Henriques},
  booktitle = "Museums and the Web 1998",
  year= 1998
```

}

```
@InProceedings{elpub2003,  
  author = "Alberto Manuel Simões and J.J. Almeida",  
  title= "Music publishing",  
  note = {Guimarães, Jun. 2003},  
  editor ="Sely Costa et al.",  
}
```

```
@inproceedings{linguateca,  
  author = {Alberto Manuel Simões and J.J. Almeida  
    and Luís Cabral},  
  year = 2004,  
  editor = {IBERAMIA 2004}  
}
```

```
@Article{BH98,  
  author = {Leonor Barroca and Pedro Rangel Henriques},  
  editor = {},  
}
```

B.2 Grafo

GRAFO.DOT

```
digraph Colaboradores {
  Autor [shape=polygon, sides=6, peripheries=2, color=lightblue, label="Almeida, J. J."];
  Colaborador1 [label="Barbosa, L. S."];
  Autor -> Colaborador1 [label="Colaborou 1 vez"];
  Colaborador2 [label="Cabral, L."];
  Autor -> Colaborador2 [label="Colaborou 1 vez"];
  Colaborador3 [label="Henriques, P. R."];
  Autor -> Colaborador3 [label="Colaborou 2 vezes"];
  Colaborador4 [label="Martins, F. M."];
  Autor -> Colaborador4 [label="Colaborou 1 vez"];
  Colaborador5 [label="Neves, F. L."];
  Autor -> Colaborador5 [label="Colaborou 1 vez"];
  Colaborador6 [label="Oliveira, J. N."];
  Autor -> Colaborador6 [label="Colaborou 1 vez"];
  Colaborador7 [label="Ramalho, J. C."];
  Autor -> Colaborador7 [label="Colaborou 1 vez"];
  Colaborador8 [label="Simões, A. M."];
  Autor -> Colaborador8 [label="Colaborou 2 vezes"];
}
```

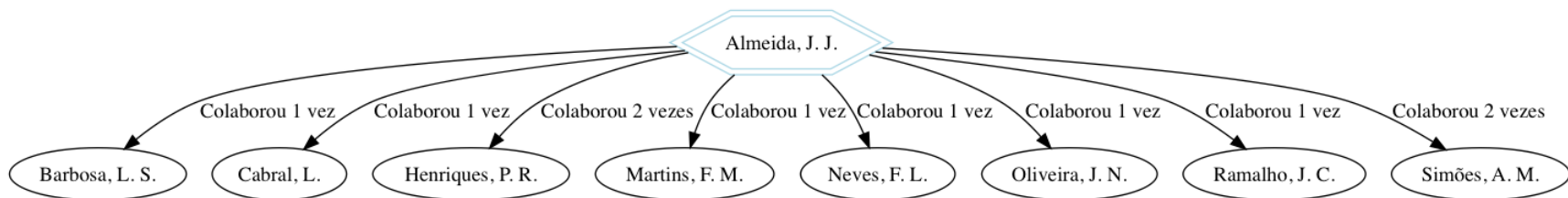


Figura B.1: Grafo gerado pelo DOTTY