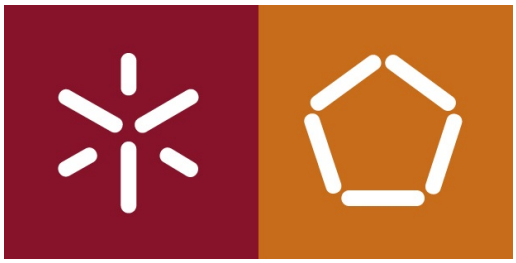


Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens e Compiladores

2017/2018 - 2ºsemestre

Trabalho Prático nº 2 - FLEX

Autores :
Diana Costa (A78985)

Marcos Pereira (A79116)

Vitor Castro (A77870)

Braga, 7 de Maio de 2018

Resumo

Perante a proposta de realizar um exercício sobre expressões regulares, filtros de texto e processadores de linguagens regulares utilizando FLEX, houve um impasse inicial, devido à necessidade de uma boa estruturação dos problemas. O nível de dificuldade e complexidade deste projeto, quando comparada com o primeiro exercício sobre GAWK, é muito superior, e levou a que o grupo se debatesse e fizesse um esforço conjunto para perceber como aplicar os conhecimentos. Tudo isto requeriu a escolha acertada de expressões regulares e respetivas ações semânticas, estruturas de dados globais para armazenar os dados ao longo do processamento, e uma análise aprofundada de extratos de texto (*datasets*), de forma a que a resolução do exercício fosse a mais clara e simples possível.

Depois de algum tempo e trabalho, o resultado encontrado foi satisfatório, e os objetivos e respostas às questões do enunciado proposto cumpridos.

Conteúdo

1 Introdução **3**

2 Preliminares **4**

3 BibTeXPro - Um processador de BibTex **5**
3.1 Análise dos extratos 5
3.2 Filtro de texto - Gerador FLEX 6
3.2.1 Alínea a - Contagem de categorias de referência e respetivo documento HTML 6
3.2.1.1 Estruturas de dados 6
3.2.1.2 Funções auxiliares 7
3.2.1.3 Ações semânticas e expressões regulares 7
3.2.1.4 Implementação 8
3.2.1.5 Como executar 9
3.2.2 Alínea b - Contagem de categorias de referência, identificação de chaves, autores e títulos de publicações, e respetivo documento HTML 10
3.2.2.1 Estruturas de dados 10
3.2.2.2 Funções auxiliares 11
3.2.2.3 Ações semânticas e expressões regulares 12
3.2.2.4 Implementação 14
3.2.2.5 Como executar 17
3.2.3 Alínea c - Índice de autores e respetivos registos para pesquisa em Linux . . 18
3.2.3.1 Estruturas de dados 18
3.2.3.2 Funções auxiliares 19
3.2.3.3 Ações semânticas e expressões regulares 20
3.2.3.4 Implementação 22
3.2.3.5 Como executar 25
3.2.4 Alínea d - Grafo de autores correlacionados por publicações para desenho em ferramentas de processamento Dot 26
3.2.4.1 Estruturas de dados 26
3.2.4.2 Funções auxiliares 27
3.2.4.3 Ações semânticas e expressões regulares 28
3.2.4.4 Implementação 30
3.2.4.5 Como executar 33
3.3 Análise de resultados 34
3.3.1 Alínea a 34
3.3.2 Alínea b 35
3.3.3 Alínea c 36
3.3.4 Alínea d 37

4 Conclusões e Sugestões **38**

1 Introdução

O objetivo deste trabalho prende-se com o aumento da experiência em ambiente *Linux*, em Expressões Regulares e utilização do *FLEX* para filtragem de texto. De entre os enunciados disponibilizados o grupo ficou com a tarefa de desenvolver o número 1, - **BibTeXPro - Um processador de BibTex**. Essencialmente, BibTeX é uma ferramenta de formatação de citações bibliográficas em documentos LaTeX, que tem como objetivo a separação de uma base de dados da bibliografia de um documento. Nela podem aparecer referências a teses de doutoramento, mestrado, livros, artigos, manuais, entre outros.

Assim, ao nível deste enunciado, era requerido que se procedesse inicialmente à contagem de categorias de referências, e que se adicionasse também, a cada categoria, a respetiva chave, autores e título de obras. Todo este esforço seria demonstrado, posteriormente, num documento HTML. Numa outra alínea, dentro do mesmo exercício, era desejado que se criasse um índice de autores, para mapear cada autor nos respetivos registos, de modo a que posteriormente uma ferramenta de procura do Linux pudesse fazer a pesquisa. Por fim, seria necessário elaborar um grafo que mostrasse, para um dado autor, todos os autores que publicam normalmente com o mesmo. Para esta fase, recorreu-se à linguagem Dot do GraphViz para gerar um ficheiro com o grafo obtido, para que se pudesse, em qualquer altura, usar uma ferramenta de processamento Dot para desenhar o dito grafo de associações de autores.

Em suma, a Secção 2 descreve os preliminares necessários ao projeto, enquanto que a Secção 3 descreve a resolução do primeiro enunciado do trabalho, desde a análise dos extratos(3.1), soluções(3.2.1 a 3.2.4) e análise consequente de resultados(3.3). O relatório termina com uma breve conclusão na Secção 4, onde é feito um balanço do trabalho realizado, tendo em conta as dificuldades ao longo do desenvolvimento do mesmo.

2 Preliminares

Para o desenvolvimento deste exercício prático não foi necessário o estudo de conteúdos adicionais aos lecionados nas aulas da unidade curricular.

A leitura e compreensão deste relatório são facilitadas através de conhecimentos de programação em FLEX e de expressões regulares, necessárias à resolução dos tópicos requeridos.

3 BibTeXPro - Um processador de BibTex

Este tema visa a análise de ficheiros em formato BibTeX, ferramenta de formatação de citações bibliográficas em documentos LaTeX, criada com o objetivo de facilitar a separação da base de dados da bibliografia consultada no fim de um documento LaTeX em edição. Para o presente exercício, foram fornecidos ao grupo dois *datasets* em formato BibTeX, "exemplo-latin1.bib" e "exemplo-utf8.bib", que possuem informação relativa a obras de carácter informático e científico, elaboradas por alguns célebres professores da Universidade do Minho, em conjunto com outros docentes e especialistas na área.

3.1 Análise dos extratos

Para compreensão total do exercício pedido, houve a necessidade de uma boa análise dos extratos. Assim, exceptuando-se alguns casos particulares, apresenta-se de seguida um excerto do texto-fonte, como representação ilustrativa e suporte para a estratégia adotada.

```
@inproceedings{AH97,
    title = "Dynamic Dictionary = cooperative information sources",
    author = "J.J. Almeida and P.R. Henriques",
    year = 1998,
    address = "Australia",
    url = "http://natura.di.uminho.pt/~jj/bib/agentes97.ps.gz",
    keyword ="dictionary, Agentes",
    booktitle = "Proc. II Conference on Knowledge-based Intelligent
                 Electronic Systems ({Kes98})",
    month = "April",
}

@inproceedings{museums98,
 author = {J.G. Rocha and M.R. Henriques and J.C. Ramalho and J.J.
           Almeida and J.L. Faria and P.R. Henriques},
 title = {Adapting Museum Structures for the Web: No Changes
          Needed!},
 booktitle = "Museums and the Web 1998",
 note = "Toronto - Canada",
 year = 1998,
}
```

Desta forma, verifica-se uma constante de categorias de referência, iniciadas com o caractere "@", que indicam todos os tipos de BibTeX existentes - article, book, manual, phdthesis, ... -, sendo que cada uma tem campos obrigatórios e opcionais, como o autor da obra, título, ano, escola, editora, entre outros. São estes campos que serão necessários analisar em prol da extração de informação necessária ao exercício. É de salientar que as informações constituintes dos campos podem aparecer entre { ou " " , o que será pertinente para a elaboração de expressões regulares, demonstradas nas secções seguintes.

3.2 Filtro de texto - Gerador FLEX

3.2.1 Alínea a - Contagem de categorias de referência e respetivo documento HTML

Neste exercício, era pedido que fosse elaborada uma página HTML com o nome das categorias encontradas no ficheiro fornecido pela equipa docente, juntamente com a respetiva contagem das mesmas. Estas categorias - article, book, manual, phdthesis, ... - aparecem ao longo do ficheiro, referindo-se a citações, na seguinte forma:

```
@inproceedings{museums98,
      author = {J.G. Rocha and M.R. Henriques and J.C. Ramalho and
                J.J. Almeida and J.L. Faria and P.R. Henriques},
      title = {Adapting Museum Structures for the Web: No Changes
              Needed!},
      booktitle = "Museums and the Web 1998",
      note = "Toronto - Canada",
      year = 1998,
}

@article{Ramalho98,
    author = "J.C. Ramalho and J.J. Almeida and P.R. Henriques",
    title = "Algebraic specification of documents",
    year = 1998,
    number = "199",
    pages = "231--247",
    journal = "Theoretical Computer Science",
    url="http://natura.di.uminho.pt/~jj/bib/amilp95.ps.gz",
    docpage="http://www.di.uminho.pt/~jcr/projectos/david/ARTIGOS/
             AMiLP95/amilp95.html",
    keyword ="PDavid, Camila, SGML",
}
```

Assim, nas próximas secções explicar-se-á o algoritmo empregue, assim como as estruturas de dados utilizadas, funções auxiliares, expressões regulares e ações semânticas.

3.2.1.1 Estruturas de dados

Para este problema, foram usadas duas estruturas de dados. A primeira é um array de strings, onde se encontram guardadas as categorias existentes possíveis de aparecer no ficheiro em análise para futura comparação (serve para listar as categorias que se quer contar). A segunda constitui um array de inteiros, com o objetivo de armazenar as contagens de cada categoria que for detectada.

```
/* Data structures for counting category occurences */
const char* categories[] = {"article", "book", "booklet", "inbook",
                            "incollection", "inproceedings", "manual",
                            "mastersthesis", "misc", "phdthesis",
                            "proceedings", "techreport", "unpublished"
                           };

int counters[NUM_OF_CATEGORIES];
```

3.2.1.2 Funções auxiliares

A função abaixo é responsável por, sempre que recebe uma nova categoria do ficheiro, a comparar com as categorias já existentes, e, em caso positivo, incrementar o contador dessa categoria, armazenado no array *counters[]*.

```
void onCategoryDetection(char* line) {}
```

Função auxiliar da anterior, que prepara o texto recebido (yytext) para uma string, de modo a poder ser posteriormente comparada com as strings do array *categories[]* (categorias existentes).

```
void trimStringEnds(char* string, char* empty) {}
```

3.2.1.3 Ações semânticas e expressões regulares

Um arroba no início da linha especifica a categoria da referência, que termina numa chaveta. Assim sendo, a expressão regular representada abaixo captura qualquer sequência de caracteres (exceto nova linha) entre um arroba no início da linha e uma chaveta. Ações Semânticas: ao encontrar um match para esta expressão regular, é desejado que se extraia no que está entre o arroba e a chaveta (fazer trim de yytext, que foi o texto capturado), verificar se é uma das categorias de que estamos à procura, e se sim, aumentar o contador dessa categoria.

```
^@.+\{      { onCategoryDetection(yytext); }
```

A última expressão captura todos os caracteres que não foram capturados pela expressão regular anterior. Ações Semânticas: nenhuma; quer-se prevenir o ECHO por defeito de cada um destes caracteres, para não poluir o output.

```
.|\n        { /* Ignore all other characters. */ }
```

3.2.1.4 Implementação

```
%{
  /* For tolower() */
  #include <ctype.h>
  /* For strcpy() */
  #include <string.h>

  /* Update this if you change categories[] */
  #define NUM_OF_CATEGORIES 13
  /* Update this if you need more bibtex entries
     (We could use a dynamic array) */
  #define MAX_ENTRIES 1024
  /* Update this if bibtex fields can be bigger
     (We could use a better strcpy) */
  #define MAX_FIELD_LENGTH 1024

  // Data structures for counting category occurences
  const char* categories[] = {"article", "book", "booklet",
                              "inbook", "incollection", "inproceedings",
                              "manual", "mastersthesis", "misc", "phdthesis",
                              "proceedings", "techreport", "unpublished"};
  int counters[NUM_OF_CATEGORIES];


  void trimStringEnds(char* string, char* empty) {
    char* copy = string;
    copy++; // Trim first character
    strcpy(empty, copy);
    empty[strlen(empty) - 1] = '\0'; // Last string character becomes \0
  }

  void onCategoryDetection(char* line) {

    char* category = malloc(MAX_FIELD_LENGTH * sizeof(*category));
    trimStringEnds(line, category);

    // Category to lowercase
    char* c2 = category;
    for ( ; *c2; ++c2) *c2 = tolower(*c2);

    // Loop over categories, and if there is a match,
    // increase the respective counter
    int i;
    for (i = 0; i < NUM_OF_CATEGORIES; i++) {
      if (strcmp(category, categories[i]) == 0) {
        counters[i]++;
        break;
      }
    }
  }

%}
```

```
%%

^@.+\{       { onCategoryDetection(yytext); }
.|\n         { /* Ignore all other characters. */ }

%%

int main() {

  yylex();

  printf("<!DOCTYPE␣html>\n");
  printf("<html>\n");
  printf("<head>\n");
  printf("<title>The␣best␣page␣in␣the␣world</title>\n");
  printf("<meta␣charset=\"utf-8\"/>\n");
  printf("</head>\n");
  printf("<body␣style=\"margin:␣50px\">\n");
  printf("<h1>Category␣Counter</h1>\n");
  printf("<table>\n");

  // Print categories and counters
  int i;
  for(i = 0; i < NUM_OF_CATEGORIES; i++) {
    printf("<tr>\n");
    printf("<td>%s</td>\n", categories[i]);
    printf("<td>%d</td>\n", counters[i]);
    printf("</tr>\n");
  }

  printf("</table>");
  printf("</body>\n");
  printf("</html>\n");
}
```

3.2.1.5 Como executar

Para o ficheiro e1p1.l:

```
flex e1p1.l
gcc -o e1p1 lex.yy.c -lfl
./e1p1 < exemplo-utf8.bib > result.html
```

3.2.2 Alínea b - Contagem de categorias de referência, identificação de chaves, autores e títulos de publicações, e respetivo documento HTML

Nesta alínea, era desejado que se completasse o exercício anterior, e que se acrescentasse à página HTML, para além do nome das categorias encontradas no *dataset* e as respetivas contagens, as chaves de cada categoria(ID), seguido dos autores da citação e do título da mesma. Desta forma, seria necessário captar:

```
@inproceedings{museums98,
      author = {J.G. Rocha and M.R. Henriques and J.C. Ramalho and
                J.J. Almeida and J.L. Faria and P.R. Henriques},
      title = {Adapting Museum Structures for the Web:  No Changes
               {Needed!},
      booktitle = "Museums and the Web 1998",
      note = "Toronto - Canada",
      year = 1998,
}

@article{Ramalho98,
    author = "J.C. Ramalho and J.J. Almeida and P.R. Henriques",
    title = "Algebraic specification of documents",
    year = 1998,
    number = "199",
    pages = "231--247",
    journal = "Theoretical Computer Science",
    url="http://natura.di.uminho.pt/~jj/bib/amilp95.ps.gz",
    docpage="http://www.di.uminho.pt/~jcr/projectos/david/ARTIGOS/
             AMiLP95/amilp95.html",
    keyword ="PDavid, Camila, SGML",
}
```

Assim, nas próximas secções explicar-se-á o algoritmo empregue, assim como as estruturas de dados utilizadas, funções auxiliares, expressões regulares e ações semânticas.

3.2.2.1 Estruturas de dados

Neste problema, foram usadas cinco estruturas de dados. As duas primeiras, já explicadas anteriormente, são um array de categorias possíveis de aparecer no ficheiro, e um array de inteiros, que guardará as contagens de cada categoria detetada. As três últimas aparecem como novas ao problema, e servirão para armazenar os id's ou chaves presentes no *dataset*, e os respetivos títulos de citações e autores filtrados. A variável entryCounter é usada para guardar os id's, autores e títulos no índice correto.

```
/* Data structures for counting category occurences */
const char* categories[] = {"article", "book", "booklet", "inbook",
                            "incollection", "inproceedings", "manual",
                            "masterthesis", "misc", "phdthesis",
                            "proceedings", "techreport", "unpublished"
                           };

int counters[NUM_OF_CATEGORIES];
/* Data structures for filtering IDs, authors, and titles */
char* ids[MAX_ENTRIES];
char* authors[MAX_ENTRIES];
char* titles[MAX_ENTRIES];
```

```
/* State variable - what entry index are we currently on?
   Starts at -1 because we increment right away,
   we can't increment at the end of onCategoryDetection
   because other functions run after it that depend on the counter */
int entryCounter = -1;
```

3.2.2.2 Funções auxiliares

A função abaixo é responsável por, sempre que recebe uma nova categoria do ficheiro, a comparar com as categorias já existentes, e, em caso positivo, incrementar o contador dessa categoria, armazenado no array *counters[]*.

```
void onCategoryDetection(char* line) {}
```

A função *onIDDetection()* tem como objetivo alocar espaço para uma chave recebida, e inseri-la no array *ids[]*.

```
void onIDDetection(char* line) {}
```

A próxima função é responsável por alocar espaço para um título recebido, preparar essa string e inseri-la no array *titles[]*.

```
void onTitleDetection(char* line) {}
```

Função que é responsável por alocar espaço para um autor detetado, preparar essa string e inseri-la no array *authors[]*.

```
void onAuthorDetection(char* line) {}
```

Função auxiliar de todas as anteriores (exceto da *onIDDetection()*), que prepara o texto recebido (yytext) para uma string, de modo a poder ser posteriormente comparada com as strings do array *categories[]* (categorias existentes), ou para ser simplesmente armazenada no array respetivo a informação que lhe é pertinente.

```
void trimStringEnds(char* string, char* empty) {}
```

3.2.2.3 Ações semânticas e expressões regulares

Com esta expressão regular capturam-se as linhas que começam com "@string". Ações Semânticas: ignorar estas linhas, para que o programa não comece o processo de registo de uma nova entrada.

```
^@string\{                    { /* Ignore @string */ }
```

Um arroba no início da linha especifica a categoria da referência, que termina numa chaveta. Assim sendo, a expressão regular representada abaixo captura qualquer sequência de caracteres (exceto nova linha) entre um arroba no início da linha e uma chaveta. Ações Semânticas: ao encontrar um match para esta expressão regular, é desejado que se extraia no que está entre o arroba e a chaveta (fazer trim de yytext, que foi o texto capturado), verificar se é uma das categorias de que se está à procura, e se sim, aumentar o contador dessa categoria. Para além disto, e em acréscimo ao que foi feito na alínea a, quer-se agora entrar no contexto ID, que deve capturar a ID da entrada que se acabou de detetar.

```
^@.+\{                        { onCategoryDetection(yytext); BEGIN ID; }
```

Esta expressão captura qualquer sequência de caracteres até uma vírgula, dentro do contexto ID. Ações Semânticas: capturar a ID da entrada pela qual se está a passar. No final, volta-se ao contexto inicial.

```
<ID>[^,]+                     { onIDDetection(yytext); BEGIN INITIAL; }
```

A expressão seguinte captura qualquer linha começada por um número arbitrário de espaços, seguida de "author"em upper ou lower case e um "=", com um número também arbitrário de espaços antes e depois do igual. É necessário lidar com author em upper ou lower case porque há uma entrada no ficheiro exemplo-utf8.bib (só uma) que tem estes campos em maiúsculas. Ações Semânticas: quer-se detetar o campo author e entrar no contexto que o captura.

```
^[ ]*(author|AUTHOR)[ ]*=[ ]*              BEGIN AUTHOR;
```

Esta expressão faz o mesmo que a anterior, mas em vez de capturar o autor, captura o título. Ações Semânticas: Mais uma vez, deseja-se detetar o campo e entrar no contexto que o captura.

```
^[ ]*(title|TITLE)[ ]*=[ ]*                BEGIN TITLE;
```

A próxima expressão é a mais complicada do ficheiro. Dentro do contexto AUTHOR, quer-se capturar o valor do campo, que pode ser delimitado por chavetas ({}) ou aspas (¨¨), e que pelo meio pode ter "newlines"e mais um nível de profundidade de chavetas ({}), desde que sejam fechadas.

```
[{"]   (   \{[^{}"]*\}   |   [^{}"]   )*   [}"]
1              2              3      4     1
```

1. Indica que o campo pode ser delimitado por chavetas ou aspas.
2. Indica que o campo pode conter chavetas fechadas com qualquer coisa lá dentro que não seja chavetas ou aspas. Provavelmente também não devem haver "newlines"dentro destas chavetas, mas isso não entra em conflito com os objetivos do grupo, e não é trabalho do mesmo validar o ficheiro.
3. Indica que o campo pode conter quaisquer caracteres que não chavetas ou aspas, incluindo "newlines".
4. Indica que 2. ou 3. podem ser repetidos 0 ou mais vezes.

Ações Semânticas: guardar o valor do campo autor da entrada em que se está, e de seguida voltar ao contexto inicial.

```
<AUTHOR>[{"](\{[^{}"]*\}|[^{}"])*[}"]{ onAuthorDetection(yytext);
                                       BEGIN INITIAL;
                                     }
```

Esta expressão é igual à anterior, mas acionada no contexto TITLE em vez de AUTHOR. Ações Semânticas: guardar o título da entrada pela qual se está a passar, e de seguida voltar ao contexto inicial.

```
<TITLE>[{"](\{[^{}"]*\}|[^{}"])*[}"] { onTitleDetection(yytext);
                                       BEGIN INITIAL;
                                     }
```

A última expressão captura todos os caracteres que não foram capturados pelas expressão regular anteriores. Ações Semânticas: nenhuma; quer-se prevenir o ECHO por defeito de cada um destes caracteres, para não poluir o output.

```
.|\n         { /* Ignore all other characters. */ }
```

3.2.2.4 Implementação

```
%{
  /* For tolower() */
  #include <ctype.h>
  /* For strcpy() */
  #include <string.h>

  /* Update this if you change categories[] */
  #define NUM_OF_CATEGORIES 13
  /* Update this if you need more bibtex entries
     (We could use a dynamic array) */
  #define MAX_ENTRIES 1024
  /* Update this if bibtex fields can be bigger
     (We could use a better strcpy) */
  #define MAX_FIELD_LENGTH 1024

  // Data structures for counting category occurences
  const char* categories[] = {"article", "book", "booklet",
                              "inbook", "incollection", "inproceedings",
                              "manual", "mastersthesis", "misc", "phdthesis",
                              "proceedings", "techreport", "unpublished"};

  int counters[NUM_OF_CATEGORIES];

  // Data structures for filtering IDs, authors, and titles
  char* ids[MAX_ENTRIES];
  char* authors[MAX_ENTRIES];
  char* titles[MAX_ENTRIES];

  /* State variable - what entry index are we currently on?
     Starts at -1 because we increment right away,
     we can't increment at the end of onCategoryDetection
     because other functions run after it that depend on the counter
  */
  int entryCounter = -1;



  void trimStringEnds(char* string, char* empty) {
    char* copy = string;
    copy++; // Trim first character
    strcpy(empty, copy);
    empty[strlen(empty) - 1] = '\0'; // Last string character becomes \0
  }



  void onCategoryDetection(char* line) {

    char* category = malloc(MAX_FIELD_LENGTH * sizeof(*category));
    trimStringEnds(line, category);
```

```
    // Category to lowercase
    char* c2 = category;
    for ( ; *c2; ++c2) *c2 = tolower(*c2);

    // Loop over categories, and if there is a match,
    // increase the respective counter
    int matched = 0;
    int i;
    for (i = 0; i < NUM_OF_CATEGORIES; i++) {
      if (strcmp(category, categories[i]) == 0) {
        counters[i]++;
        matched = 1;
        break;
      }
    }
    if (matched == 1) {
      // Increment counter, because there was a match!
      // We'll be storing an entry.
      entryCounter++;
    }
  }


  void onIDDetection(char* line) {
    char *id = malloc(MAX_FIELD_LENGTH * sizeof(*id));
    strcpy(id, line);
    ids[entryCounter] = id;
  }


  void onTitleDetection(char* line) {
    char *title = malloc(MAX_FIELD_LENGTH * sizeof(*title));
    trimStringEnds(line, title);
    titles[entryCounter] = title;
  }


  void onAuthorDetection(char* line) {
    char *author = malloc(MAX_FIELD_LENGTH * sizeof(*author));
    trimStringEnds(line, author);
    authors[entryCounter] = author;
  }
%}

%x ID AUTHOR TITLE

%%

^@string\{                      { /* Ignore @string */ }
^@.+\{                          { onCategoryDetection(yytext); BEGIN ID; }
<ID>[^,]+                       { onIDDetection(yytext); BEGIN INITIAL; }
^[ ]*(author|AUTHOR)[ ]*=[ ]*   BEGIN AUTHOR;
^[ ]*(title|TITLE)[ ]*=[ ]*     BEGIN TITLE;
```

```
<AUTHOR>[{"](\{[^{}"]*\}|[^{}"])*[}"]    { onAuthorDetection(yytext);
                                           BEGIN INITIAL;
                                         }
<TITLE>[{"](\{[^{}"]*\}|[^{}"])*[}"]     { onTitleDetection(yytext);
                                           BEGIN INITIAL;
                                         }
.|\n                                     { /* Ignore all other characters. */ }

%%

int main() {
  yylex();

  printf("<!DOCTYPE␣html>\n");
  printf("<html>\n");
  printf("<head>\n");
  printf("<title>The␣best␣page␣in␣the␣world</title>\n");
  printf("<meta␣charset=\"utf-8\"/>\n");
  printf("</head>\n");
  printf("<body␣style=\"margin:␣50px\">\n");
  printf("<h1>Category␣Counter</h1>\n");
  printf("<table>\n");

  // Print categories and counters
  int i;
  for(i = 0; i < NUM_OF_CATEGORIES; i++) {
    printf("<tr>\n");
    printf("<td>%s</td>\n", categories[i]);
    printf("<td>%d</td>\n", counters[i]);
    printf("</tr>\n");
  }
  printf("</table>");

  // Print IDs, titles, and authors
  printf("<br>\n");
  printf("<br>\n");
  printf("<table>\n");
  printf("<td>ID</td>\n");
  printf("<td>Title</td>\n");
  printf("<td>Author</td>\n");

  int j;
  for(j = 0; j <= entryCounter; j++) {
    printf("<tr>\n");
    printf("<td>%s</td>\n", ids[j]);
    printf("<td>%s</td>\n", titles[j]);
    printf("<td>%s</td>\n", authors[j]);
    printf("</tr>\n");
  }

  printf("</table>");
  printf("</body>\n");
  printf("</html>\n");
}
```

3.2.2.5 Como executar

Para o ficheiro e1p2.l:

```
flex e1p2.l
gcc -o e1p2 lex.yy.c -lfl
./e1p2 < exemplo-utf8.bib > result.html
```

3.2.3 Alínea c - Índice de autores e respetivos registos para pesquisa em Linux

Neste problema, pedia-se que se criasse um índice de autores, que mapeasse cada autor nos respetivos registos, de modo a que posteriormente uma ferramenta de Linux possa fazer a pesquisa. Desta forma, seria necessário captar a seguinte informação:

```
@inproceedings{museums98,
      author = {J.G. Rocha and M.R. Henriques and J.C. Ramalho and
                J.J. Almeida and J.L. Faria and P.R. Henriques},
      title = {Adapting Museum Structures for the Web:  No ChangesNeeded!},
      booktitle = "Museums and the Web 1998",
      note = "Toronto - Canada",
      year = 1998,
}
```

Assim, nas próximas secções explicar-se-á o algoritmo empregue, assim como as estruturas de dados utilizadas, funções auxiliares, expressões regulares e ações semânticas.

3.2.3.1 Estruturas de dados

De uma maneira simples, o programa realizado pelo grupo passa pelas entradas uma a uma, e vai guardando a informação dos autores, título, e ID à medida que os atravessa. Ao chegar uma nova entrada, esta informação é toda guardada, e é dado reset às variáveis que guardam os valores da entrada atual. Desta forma, utilizaram-se as seguintes variáveis e estruturas de dados globais:

- char* currentTitle - Guarda o título da entrada atual
- char* currentID - Guarda a ID da entrada atual.
- char* currentAuthors[MAX_AUTHORS] - Armazena a lista dos autores da entrada atual.
- int currentAuthorsLength = 0 - Guarda o tamanho da lista dos autores da entrada atual.
- int startedWritingCurrentAuthor = 0 - Uma flag que indica se já se começou a guardar os caracteres do autor atual. Se a flag se encontra a zero, sabe-se que é necessário alocar espaço para uma nova string que será colocada no array de autores
- char* allAuthors[MAX_AUTHORS] - Guarda os nomes de todos os autores encontrados. Juntamente com o array allWorks, guarda os resultados que o grupo pretende obter.
- int allAuthorsLength = 0 - Armazena o tamanho do array allAuthors.
- char* allWorks[MAX_AUTHORS] - Guarda os trabalhos de cada autor em allAuthors no índice respetivo, no formato "título por extenso (ID), outro título (outra ID), etc."

Em resumo...

```
char* currentTitle;
char* currentID;
char* currentAuthors[MAX_AUTHORS];
int currentAuthorsLength = 0;
int startedWritingCurrentAuthor = 0;
char* allAuthors[MAX_AUTHORS];
int allAuthorsLength = 0;
char* allWorks[MAX_AUTHORS];
```

3.2.3.2 Funções auxiliares

A próxima função tem como propósito alocar espaço para guardar o título de uma citação, e posicionar o apontador *currentTitle* no título atual.

```
void storeTitle(char* line) {}
```

Função auxiliar da anterior, que prepara o texto recebido (yytext) para uma string, de modo a poder ser armazenada no array respetivo a informação que lhe é pertinente.

```
void trimStringEnds(char* string, char* empty) {}
```

A função seguinte é responsável por alocar espaço para guardar o ID de uma citação, para associar posteriormente a um título. Posiciona também o apontador *currentID* no ID atual.

```
void storeID(char* line) {}
```

Uma vez que é necessário fazer a separação de caracteres para extrair os autores de um mesmo campo (o campo author pode conter mais do que um autor, sendo estes separados pela palavra "and"), a função *oneCharOfAuthor()* surge para extrair, caractere a caracter, os autores, alocando espaço para os mesmos, para armazenamento no array *currentAuthors[]*, caso o autor ainda não tenha começado a ser escrito. Caso contrário, apenas concatena o caractere à palavra (autor) já existente.

```
void oneCharOfAuthor(char* string) {}
```

Função auxiliar da anterior, que concatena, no final de cada palavra (autor), os caracteres constituintes do seu nome. No fim, adiciona um '\0'.

```
void appendCharToString(char* string, char c) {}
```

A função abaixo é chamada quando é detetado outro autor, para saber, na função *oneCharOfAuthor()* que será necessário alocar espaço para o novo autor, e que o tamanho do array *currentAuthors[]* crescerá uma unidade.

```
void anotherAuthor() {}
```

A função *storeData()* é das mais extensas, e tem como responsabilidade guardar a informação nos respetivos arrays. Assim, no fim da análise de cada citação, para cada autor que esteja no array de autores temporários(*currentAuthors[]*), inicialmente vai verificar se este já se encontra no array final, *allAuthors[]*. Em caso positivo, extrai o índice em que ele está e usa-o para armazenar os trabalhos do autor em causa. Em caso negativo, a função cria uma entrada neste último array, e armazena a informação pretendida. Os trabalhos de um autor são separados por vírgulas.

```
void storeData() {}
```

3.2.3.3 Ações semânticas e expressões regulares

Com esta expressão regular capturam-se as linhas que começam com "@string". Ações Semânticas: ignorar estas linhas, para que o programa não comece o processo de registo de uma nova entrada.

```
^@string\{                    { /* Ignore @string */ }
```

Um arroba no início da linha especifica a categoria da referência, que termina numa chaveta. Assim sendo, a expressão regular representada abaixo captura qualquer sequência de caracteres (exceto nova linha) entre um arroba no início da linha e uma chaveta. Ações Semânticas: ao encontrar um match para esta expressão regular, é desejado que se extraia no que está entre o arroba e a chaveta (fazer trim de yytext, que foi o texto capturado), e que se armazene a informação.

```
^@.+\{                        { storeData(); BEGIN ID; }
```

Esta expressão captura qualquer sequência de caracteres até uma vírgula, dentro do contexto ID. Ações Semânticas: capturar a ID da entrada pela qual se está a passar. No final, volta-se ao contexto inicial.

```
<ID>[^,]+                     { storeID(yytext); BEGIN INITIAL; }
```

A expressão representada abaixo captura qualquer linha começada por um número arbitrário de espaços, seguida de "author"em upper ou lower case e um "=", com um número também arbitrário de espaços antes e depois do igual. É necessário lidar com author em upper ou lower case porque há uma entrada no ficheiro exemplo-utf8.bib (só uma) que tem estes campos em maiúsculas.

```
^[ ]*(author|AUTHOR)[ ]*=[ ]*[{"]*          BEGIN AUTHOR;
```

Esta expressão faz o mesmo que a anterior, mas em vez de capturar o autor, captura o título. Ações Semânticas: Mais uma vez, deseja-se detetar o campo e entrar no contexto que o captura.

```
^[ ]*(title|TITLE)[ ]*=[ ]*                BEGIN TITLE;
```

O campo author pode conter mais do que um autor, sendo estes separados pela palavra "and". Enquanto se "apanha"um campo author, se for encontrada uma chaveta, entra-se no contexto AUTHOR_BRACKET. O contexto AUTHOR_BRACKET é necessário, uma vez que se se continuasse dentro do contexto AUTHOR, daria-se o campo por terminado quando encontrássemos a respetiva chaveta a fechar (}), o que faria com que o programa deixasse de funcionar corretamente. Este contexto é, sobretudo, pertinente para o caso em que o campo autor se encontra na forma author ={{projecto Camila}}.

```
<AUTHOR>{
  \{                        { oneCharOfAuthor("{"); BEGIN AUTHOR_BRACKET; }
```

Enquanto se detecta um campo author, capturam-se todos os caracteres que façam parte do nome (todos menos "abre chaveta"e "nova linha", sendo incluído também o \r. Ação semântica: Adicionar o caractere atual à string que contém o nome do autor que se está a guardar.

```
  [^{\n\r]                { oneCharOfAuthor(yytext); }
```

Quando se está a "apanhar"um campo author, é desejado que se detecte a palavra "and"rodeada de espaços ou newlines, o que significa que, de seguida irá aparecer o nome de um outro autor que faz parte da mesma obra. Ação semântica: Guardar o autor anterior, e preparar o array de autores da obra atual com uma string vazia para começar a receber o nome do próximo.

```
  [ \n\r]+and[ \n\r]+    { anotherAuthor(); }
```

Enquanto se detecta um campo author, se for encontrado um fecho de chaveta ou uma aspa, dá-se como terminado o campo e volta-se ao contexto inicial.

```
  [}"],                  { BEGIN INITIAL; }
```

Esta expressão ignora newlines no meio de nomes de autores, para que não apareçam nomes duplicados com newlines no meio.

```
  \n                { /* Ignore newlines in the middle of author names */ }
}
```

Dentro do contexto AUTHOR_BRACKET, todos os caracteres menos chaveta a fechar (}) são adicionados ao nome do autor. Assume-se que os conteúdos de um bracket num campo autor não podem conter newlines.

```
<AUTHOR_BRACKET>{
  [^}]            { oneCharOfAuthor(yytext); }
```

Dentro do mesmo contexto, uma chaveta a fechar significa que se volta ao contexto AUTHOR.

```
  \}              { oneCharOfAuthor("}"); BEGIN AUTHOR; }
}
```

Já no contexto TITLE, quer-se guardar o título que se detetou e voltar ao contexto inicial.

```
  <TITLE>[{"](\{[^{}"]*\}|[^{}"])*[}"]   { storeTitle(yytext); BEGIN INITIAL; }
}
```

A última expressão captura todos os caracteres que não foram capturados pelas expressão regular anteriores. Ações Semânticas: nenhuma; quer-se prevenir o ECHO por defeito de cada um destes caracteres, para não poluir o output.

```
.|\n          { /* Ignore all other characters. */ }
```

3.2.3.4 Implementação

```
%{
  /* For tolower() */
  #include <ctype.h>
  /* For strcpy() */
  #include <string.h>

  /* Update this if you need more bibtex entries
     (We could use a dynamic array) */
  #define MAX_ENTRIES 10000
  /*  Update this if you need to allow more authors in total */
  #define MAX_AUTHORS 10000
  /* Update this if bibtex fields can be bigger
     (We could use a better strcpy) */
  #define MAX_FIELD_LENGTH 2048

  // Globals
  char* currentTitle;
  char* currentID;
  char* currentAuthors[MAX_AUTHORS];
  int currentAuthorsLength = 0;
  int startedWritingCurrentAuthor = 0;
  char* allAuthors[MAX_AUTHORS];
  int allAuthorsLength = 0;
  char* allWorks[MAX_AUTHORS];


  void trimStringEnds(char* string, char* empty) {
    char* copy = string;
    copy++; // Trim first character
    strcpy(empty, copy);
    empty[strlen(empty) - 1] = '\0'; // Last string character becomes \0
  }

  void appendCharToString(char* string, char c) {
    string[strlen(string) + 1] = '\0';
    string[strlen(string)] = c;
  }

  void storeTitle(char* line) {
    char *title = malloc(MAX_FIELD_LENGTH * sizeof(*title));
    trimStringEnds(line, title);
    currentTitle = title;
  }

  void storeID(char* line) {
    // Unlike title, ID doesn't have to be trimmed
    char *id = malloc(MAX_FIELD_LENGTH * sizeof(*id));
    strcpy(id, line);
    currentID = id;
  }
```

```
void oneCharOfAuthor(char* str) {

  if (!startedWritingCurrentAuthor) {
    char* author = malloc(MAX_FIELD_LENGTH * sizeof(*author));
    currentAuthors[currentAuthorsLength] = author;
    startedWritingCurrentAuthor = 1;
  }
  appendCharToString(currentAuthors[currentAuthorsLength], str[0]);
}

void anotherAuthor() {
  startedWritingCurrentAuthor = 0;
  currentAuthorsLength++;
}

void storeData() {

  // Check if there is data to store
  // (This could be called at the start of the file)
  if (currentAuthorsLength == 0) {
    return;
  }

  // Stopped writing an author, increment
  currentAuthorsLength++;

  // For each current author
  int i;
  for (i = 0; i < currentAuthorsLength; i++) {

    // Find author in all authors array
    int found = 0;
    int authorIndex = -1;
    int j;
    for (j = 0; j < allAuthorsLength; j++) {

      // If found, get the index
      if (strcmp(currentAuthors[i], allAuthors[j]) == 0) {
        authorIndex = j;
        found = 1;
        break;
      }
    }

    // If not found, append and index = allAuthorsLength
    if (found == 0) {
      char *s = malloc(MAX_FIELD_LENGTH * sizeof(*s));
      strcpy(s, currentAuthors[i]);
      allAuthors[allAuthorsLength] = s;
      authorIndex = allAuthorsLength;
      allAuthorsLength++;
    }
```

```
      // Add ID and Title string to the string in allWorks
      // in the index of that author
      if (found == 0) {
        char* n = malloc(MAX_FIELD_LENGTH * sizeof(*n));
        allWorks[authorIndex] = n;
      }

      strcat(allWorks[authorIndex], currentTitle);
      strcat(allWorks[authorIndex], "␣(");
      strcat(allWorks[authorIndex], currentID);
      strcat(allWorks[authorIndex], "),␣");
    }

    // Reset needed variables
    currentAuthorsLength = 0;
    startedWritingCurrentAuthor = 0;
  }
%}

%x ID AUTHOR AUTHOR_BRACKET TITLE

%%

^@string\{                              { /* Ignore @string */ }
^@.+\{                                  { storeData(); BEGIN ID; }
<ID>[^,]+                               { storeID(yytext); BEGIN INITIAL; }
^[ ]*(author|AUTHOR)[ ]*=[ ]*[{"]*      BEGIN AUTHOR;
^[ ]*(title|TITLE)[ ]*=[ ]*             BEGIN TITLE;
<AUTHOR>\{                              { oneCharOfAuthor("{");
                                          BEGIN AUTHOR_BRACKET;
                                        }
<AUTHOR>[^{\n\r]                        { oneCharOfAuthor(yytext); }
<AUTHOR>[}"],                           BEGIN INITIAL;
<AUTHOR>[ \n\r]+and[ \n\r]+             { anotherAuthor(); }
<AUTHOR>\n                              { /* Ignore newlines in the middle
                                             of author names */
                                        }
<AUTHOR_BRACKET>[^}]                    { oneCharOfAuthor(yytext); }
<AUTHOR_BRACKET>\}                      { oneCharOfAuthor("}");
                                          BEGIN AUTHOR;
                                        }
<TITLE>[{"](\{[^{}"]*\}|[^{}"])*[}"]    { storeTitle(yytext);
                                          BEGIN INITIAL;
                                        }
.|\n                                    { /* Ignore all other characters. */ }

%%

int main() {

  yylex();
```

```
  /* One last storeData();
     Needed because storeData is called at the start
     of a new entry, because we can't rely on a specific
     field to be the last of an entry
  */
  storeData();

  // allAuthorsLength is actually allAuthorsMaxIndex.
  // We should fix that in the future!
  allAuthorsLength--;

  printf("Authors␣and␣Works\n");

  int j;
  for(j = 0; j <= allAuthorsLength; j++) {
    printf("\n%s:␣", allAuthors[j]);
    printf("%s\n", allWorks[j]);
  }
}
```

3.2.3.5 Como executar

Para o ficheiro e1p3.l:

```
flex e1p3.l
gcc -o e1p3 lex.yy.c -lfl
./e1p3 < exemplo-utf8.bib > result.txt
```

3.2.4 Alínea d - Grafo de autores correlacionados por publicações para desenho em ferramentas de processamento Dot

Neste exercício, era pedido que fosse elaborado um grafo que, para um dado autor introduzido pelo utilizador, mostrasse todos os autores que publicam normalmente com o primeiro. Assim, e antes da construção do grafo, era essencialmente necessário que se processasse o documento de modo a retirar todos os autores que aparecessem dentro do campo "author"de uma citação. A título de exemplo, se fosse escolhido o autor **"J.L.Faria"**, todos os que surgissem no respetivo campo com ele teriam de ser armazenados para posterior desenho do grafo.

```
@inproceedings{museums98,
      author = {J.G. Rocha and M.R. Henriques and J.C. Ramalho and
                J.J. Almeida and J.L. Faria and P.R. Henriques},
      title = {Adapting Museum Structures for the Web: No Changes
             Needed!},
      booktitle = "Museums and the Web 1998",
      note = "Toronto - Canada",
      year = 1998,
}
```

Assim, nas próximas secções explicar-se-á o algoritmo empregue, assim como as estruturas de dados utilizadas, funções auxiliares, expressões regulares e ações semânticas.

3.2.4.1 Estruturas de dados

Para este problema, foram usadas três estruturas de dados. A primeira é uma string simples para posterior armazenamento do input proveniente do utilizador, que será o autor desejado. A segunda é um array de strings, onde serão guardados os autores já processados que estejam relacionados com o principal, e que será utilizada para produzir o grafo. Por fim, tem-se outro array de strings, com o objetivo de estrutura de armazenamento intermédia, usada aquando do processamento de dados. Usaram-se também três variáveis, de modo a monitorizar o tamanho dos arrays, e controlar a escrita de autores, tal como na alínea c.

```
/* string to save the input (desired author) */
char authorIN[50];
/* Data structure for the main author's correlated authors */
char* allAuthors[MAX_AUTHORS];
/* Data structure to save temporarily the authors under analysis */
char* currentAuthors[MAX_AUTHORS];
int startedWritingCurrentAuthor = 0;
int currentAuthorsLength = 0;
int allAuthorsLength = 0;
```

3.2.4.2 Funções auxiliares

Uma vez que é necessário fazer a separação de caracteres para extrair os autores de um mesmo campo (o campo author pode conter mais do que um autor, sendo estes separados pela palavra "and"), a função *oneCharOfAuthor()* surge para extrair, caractere a caracter, os autores, alocando espaço para os mesmos, para armazenamento no array *currentAuthors[]*, caso o autor ainda não tenha começado a ser escrito. Caso contrário, apenas concatena o caractere à palavra (autor) já existente.

```
void oneCharOfAuthor(char* string) {}
```

Função auxiliar da anterior, que concatena, no final de cada palavra (autor), os caracteres constituintes do seu nome. No fim, adiciona um '\0'.

```
void appendCharToString(char* string, char c) {}
```

A função abaixo é chamada quando é detetado outro autor, para saber, na função *oneCharOfAuthor()* que será necessário alocar espaço para o novo autor, e que o tamanho do array *currentAuthors[]* crescerá uma unidade.

```
void anotherAuthor() {}
```

A função *checkAuthors()* é a mais importante, no sentido em que se fará todo o processamento essencial ao programa. De cada vez que se finalize a análise de um registo, esta função vai aceder ao array *currentAuthors[]*(que possui os autores desse registo/citação), com o intuito de analisar se o autor desejado (input) se encontra lá. Em caso negativo, passar-se-á para o próximo registo sem se efetuar nenhuma ação; em caso positivo, vão-se adicionar os autores ao array definitivo *allAuthors[]*, com exceção do próprio autor. Para este efeito, primeiro verifica-se se os autores a adicionar já não se encontram no array definitivo, para evitar repetições. Por fim, e caso não estejam no array, serão adicionados ao mesmo.

```
void checkAuthors() {}
```

3.2.4.3 Ações semânticas e expressões regulares

A expressão representada abaixo captura qualquer linha começada por um número arbitrário de espaços, seguida de "author"em upper ou lower case e um "=", com um número também arbitrário de espaços antes e depois do igual. É necessário lidar com author em upper ou lower case porque há uma entrada no ficheiro exemplo-utf8.bib (só uma) que tem estes campos em maiúsculas.

```
^[ ]*(author|AUTHOR)[ ]*=[ ]*[{"]*          BEGIN AUTHOR;
```

O campo author pode conter mais do que um autor, sendo estes separados pela palavra "and". Enquanto se "apanha"um campo author, se for encontrada uma chaveta, entra-se no contexto AUTHOR_BRACKET. O contexto AUTHOR_BRACKET é necessário, uma vez que se se continuasse dentro do contexto AUTHOR, daria-se o campo por terminado quando encontrássemos a respetiva chaveta a fechar (}), o que faria com que o programa deixasse de funcionar corretamente. Este contexto é, sobretudo, pertinente para o caso em que o campo autor se encontra na forma author ={{projecto Camila}}.

```
<AUTHOR>{
  \{                        { oneCharOfAuthor("{"); BEGIN AUTHOR_BRACKET; }
```

Enquanto se detecta um campo author, capturam-se todos os caracteres que façam parte do nome (todos menos "abre chaveta"e "nova linha", sendo incluído também o \r. Ação semântica: Adicionar o caractere atual à string que contém o nome do autor que se está a guardar.

```
  [^{\n\r]                  { oneCharOfAuthor(yytext); }
```

Quando se está a "apanhar"um campo author, é desejado que se detecte a palavra "and"rodeada de espaços ou newlines, o que significa que, de seguida irá aparecer o nome de um outro autor que faz parte da mesma obra. Ação semântica: Guardar o autor anterior, e preparar o array de autores da obra atual com uma string vazia para começar a receber o nome do próximo.

```
  [ \n\r]+and[ \n\r]+       { anotherAuthor(); }
```

Enquanto se detecta um campo author, se for encontrado um fecho de chaveta ou uma aspa, dá-se como terminado o campo e volta-se ao contexto inicial.

```
  [}"],                     { checkAuthors(); BEGIN INITIAL; }
```

Esta expressão ignora newlines no meio de nomes de autores, para que não apareçam nomes duplicados com newlines no meio.

```
  .|\n                  { /* Ignore newlines in the middle of author names */ }
}
```

Dentro do contexto AUTHOR_BRACKET, todos os caracteres menos chaveta a fechar (}) são adicionados ao nome do autor. Assume-se que os conteúdos de um bracket num campo autor não podem conter newlines.

```
<AUTHOR_BRACKET>{
  [^}]            { oneCharOfAuthor(yytext); }
```

Dentro do contexto AUTHOR_BRACKET, uma chaveta a fechar significa que se volta ao contexto AUTHOR.

```
  \}              { oneCharOfAuthor("}"); BEGIN AUTHOR; }
}
```

Por fim, já fora dos contextos AUTHOR e AUTHOR_BRACKET, a expressão abaixo ignora todos os outros caracteres não apanhados. Ações Semânticas: nenhuma; quer-se prevenir o ECHO por defeito de cada um destes caracteres, para não poluir o output.

```
.|\n              { /* Ignore all other characters. */ }
```

3.2.4.4 Implementação

```
%{
  /* For tolower() */
  #include <ctype.h>
  /* For strcpy() */
  #include <string.h>
  #include <stdio.h>

  /* Update this if you need to allow more authors in total */
  #define MAX_AUTHORS 10000
  /* Update this if bibtex fields can be bigger */
  #define MAX_FIELD_LENGTH 2048

  // Globals
  /* string to save the input (desired author) */
  char authorIN[50];
  /* Data structure for the main author's correlated authors */
  char* allAuthors[MAX_AUTHORS];
  /* Data structure to save temporarily the authors under analysis */
  char* currentAuthors[MAX_AUTHORS];
  int startedWritingCurrentAuthor = 0;
  int currentAuthorsLength = 0;
  int allAuthorsLength = 0;



  void appendCharToString(char* string, char c) {
    string[strlen(string) + 1] = '\0';
    string[strlen(string)] = c;
  }

  void oneCharOfAuthor(char* string) {

    char str = tolower(string[0]);
    if (!startedWritingCurrentAuthor) {
      char* author = malloc(MAX_FIELD_LENGTH * sizeof(*author));
      currentAuthors[currentAuthorsLength] = author;
      startedWritingCurrentAuthor = 1;
    }
    appendCharToString(currentAuthors[currentAuthorsLength], str);
  }



  void anotherAuthor() {
    startedWritingCurrentAuthor = 0;
    currentAuthorsLength++;
  }
```

```
/*
  NOTE: if we do not have removed the duplicates in this
        function it was ok because, in dot language, in a
        graph, a node is connected only to another node,
        and if there is a lot of occurrences of one of the
        nodes, it draws lots of arrows -> useful to check
        who is the author that correlates more with the input
        author.
*/
void checkAuthors() {
    int found = 0;
    int foundFinal = 0;
    int authorIndex = -1;
    int i;


    // Stopped writing authors, increment
    currentAuthorsLength++;


    // Find authorIN in currentAuthors array
    for (i = 0; i < currentAuthorsLength; i++) {

      // If found, get the index
      if (strcmp(currentAuthors[i], authorIN) == 0) {
        authorIndex = i;
        found = 1;
        break;
      }
    }


    //if found, add to allAuthors array all the correlated authors
    //(except himself). It is also necessary to check if the
    //correlated author is not already in the allAuthors array
    //(avoiding repeated correlated authors)
    if(found) {

      for(i = 0; i < currentAuthorsLength; i++) {

        //do not add himself
        if(i != authorIndex) {
          int j;

          for(j = 0; j < allAuthorsLength; j++) {
            if (strcmp(currentAuthors[i], allAuthors[j]) == 0) {
              foundFinal = 1;
              break;
            }
          }

          //if correlated author is not in allAuthors array, add to it
          if(!foundFinal) {
            char *s = malloc(MAX_FIELD_LENGTH * sizeof(*s));
```

```
                strcpy(s, currentAuthors[i]);
                allAuthors[allAuthorsLength] = s;
                allAuthorsLength++;
              }

            }

          }

        }

  // Reset needed variables
  currentAuthorsLength = 0;
  startedWritingCurrentAuthor = 0;
  }
\%}

%x AUTHOR AUTHOR_BRACKET

%%

^[ ]*(author|AUTHOR)[ ]*=[ ]*[{"]*        BEGIN AUTHOR;
<AUTHOR>{
  \{                    { oneCharOfAuthor("{"); BEGIN AUTHOR_BRACKET; }
  [^{\n\r]              { oneCharOfAuthor(yytext); }
  [ \n\r]+and[ \n\r]+   { anotherAuthor(); }
  [}"],                 { checkAuthors(); BEGIN INITIAL; }
  .|\n              { /* Ignore newlines in the middle of author names */ }
}
<AUTHOR_BRACKET>{
  [^}]          { oneCharOfAuthor(yytext); }
  \}            { oneCharOfAuthor("}"); BEGIN AUTHOR; }
}
.|\n            { /* Ignore all other characters. */ }

%%

/*
Example: ./e1p4 < exemplo-utf8.bib > results.txt -author "J.J. Almeida"

argv[0] = ./exec
argv[1] = -autor
argv[2] = "<authorIn>"
*/
int main(int argc, char** argv) {

  if(argc >= 3) {
    strcpy(authorIN, argv[2]);

    //authorIN to lowercase
    char* a = authorIN;
    for ( ; *a; ++a) *a = tolower(*a);

    yylex();
```

```
    allAuthorsLength--;

    //added ratio, node and color parameters just so the graph
    //looked pretty
    printf("digraph G {\n");
    printf("  ratio = fill;\n");
    printf("  node [style=filled];\n");

    //to print the quotation marks, it is necessary to escape them!
    int i;
    for(i = 0; i <= allAuthorsLength; i++) {
      printf("  \"%s\"", authorIN);
      printf(" -> ");
      printf("\"%s\"", allAuthors[i]);
      printf(" [color=\"0.650 0.700 0.700\"];\n");
    }

    printf("}");
  } else {
    fprintf(stderr, "Argumentos insuficientes!\n");
  }
}
```

3.2.4.5 Como executar

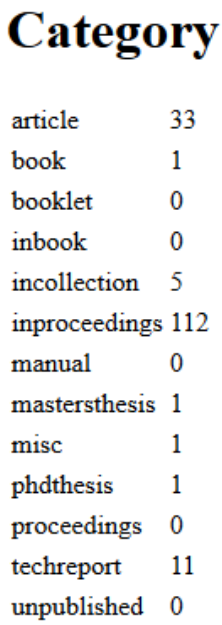
Para o ficheiro e1p4.l:

```
flex e1p4.l
gcc -o e1p4 lex.yy.c -lfl
./e1p4 < exemplo-utf8.bib > result.txt -autor="<nomeAutor>"
```

3.3 Análise de resultados

3.3.1 Alínea a

Apresenta-se, de seguida, o resultado do programa, onde se verificam o nome das categorias e respetivas contagens.



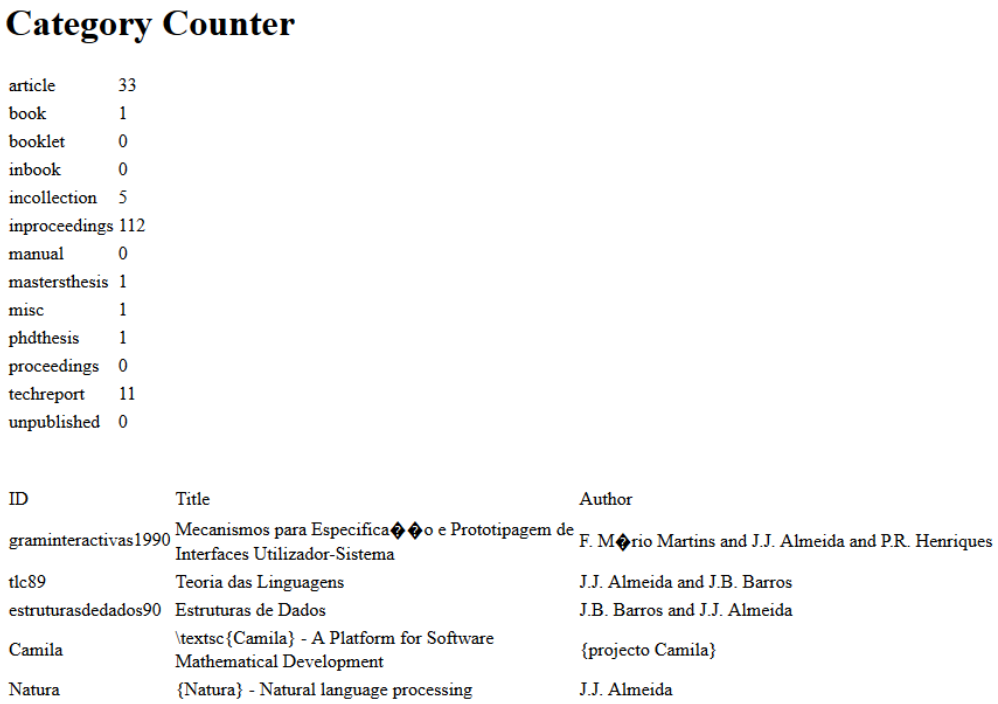
Category Counter

article	33
book	1
booklet	0
inbook	0
incollection	5
inproceedings	112
manual	0
mastersthesis	1
misc	1
phdthesis	1
proceedings	0
techreport	11
unpublished	0

Figura 1: Resultado da execução do programa numa página html

3.3.2 Alínea b

Mostra-se, de seguida, o resultado do programa, onde se verificam o nome das categorias, respetivas contagens, e para cada chave, os autores e título da citação.

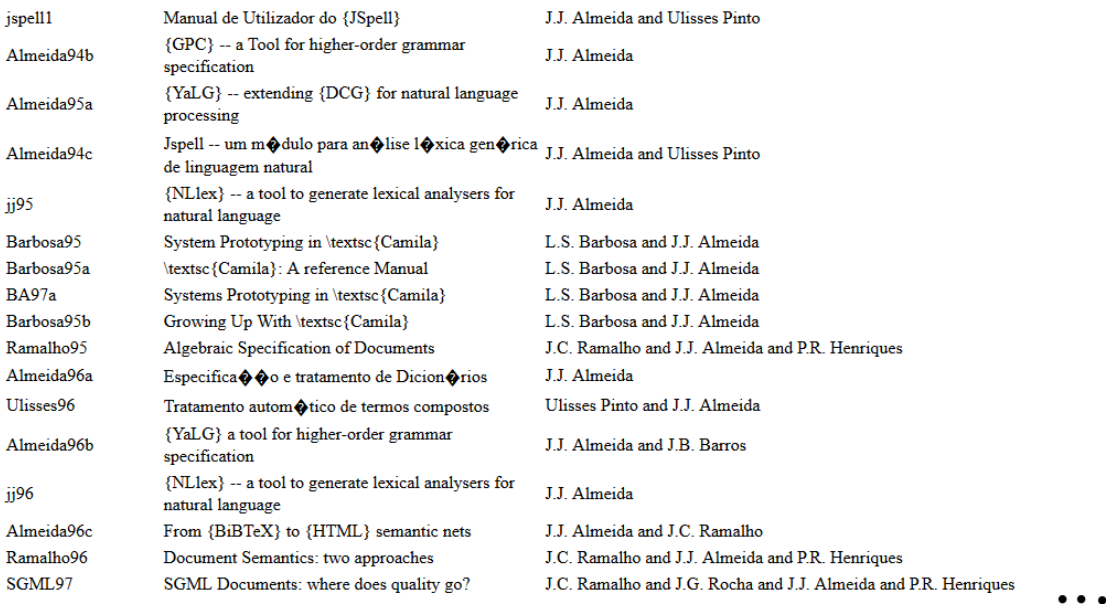


Category Counter

article	33
book	1
booklet	0
inbook	0
incollection	5
inproceedings	112
manual	0
mastersthesis	1
misc	1
phdthesis	1
proceedings	0
techreport	11
unpublished	0

ID	Title	Author
graminteractivas1990	Mecanismos para Especifica��o e Prototipagem de Interfaces Utilizador-Sistema	F. M�rio Martins and J.J. Almeida and P.R. Henriques
tlc89	Teoria das Linguagens	J.J. Almeida and J.B. Barros
estruturasdedados90	Estruturas de Dados	J.B. Barros and J.J. Almeida
Camila	\textsc{Camila} - A Platform for Software Mathematical Development	{projecto Camila}
Natura	{Natura} - Natural language processing	J.J. Almeida

Figura 2: Resultado da execução do programa numa página html



jspell1	Manual de Utilizador do {JSpell}	J.J. Almeida and Ulisses Pinto
Almeida94b	{GPC} -- a Tool for higher-order grammar specification	J.J. Almeida
Almeida95a	{YaLG} -- extending {DCG} for natural language processing	J.J. Almeida
Almeida94c	Jspell -- um m�dulo para an�lise l�xica gen�rica de linguagem natural	J.J. Almeida and Ulisses Pinto
jj95	{NLlex} -- a tool to generate lexical analysers for natural language	J.J. Almeida
Barbosa95	System Prototyping in \textsc{Camila}	L.S. Barbosa and J.J. Almeida
Barbosa95a	\textsc{Camila}: A reference Manual	L.S. Barbosa and J.J. Almeida
BA97a	Systems Prototyping in \textsc{Camila}	L.S. Barbosa and J.J. Almeida
Barbosa95b	Growing Up With \textsc{Camila}	L.S. Barbosa and J.J. Almeida
Ramalho95	Algebraic Specification of Documents	J.C. Ramalho and J.J. Almeida and P.R. Henriques
Almeida96a	Especifica��o e tratamento de Dicion�rios	J.J. Almeida
Ulisses96	Tratamento autom�tico de termos compostos	Ulisses Pinto and J.J. Almeida
Almeida96b	{YaLG} a tool for higher-order grammar specification	J.J. Almeida and J.B. Barros
jj96	{NLlex} -- a tool to generate lexical analysers for natural language	J.J. Almeida
Almeida96c	From {BiBTeX} to {HTML} semantic nets	J.J. Almeida and J.C. Ramalho
Ramalho96	Document Semantics: two approaches	J.C. Ramalho and J.J. Almeida and P.R. Henriques
SGML97	SGML Documents: where does quality go?	J.C. Ramalho and J.G. Rocha and J.J. Almeida and P.R. Henriques

Figura 3: Resultado da execução do programa numa página html - continuação

3.3.3 Alínea c

Apresenta-se o resultado do programa, onde se verificam o nome dos autores e as entradas em que aparecem.

```
Henriques, Pedro Rangel: Influence of domain-specific notation to program
understanding (kosar09),

Bruno Defude: A Query-by-Example Approach for XML Querying (FCHGD09a),
GuessXQ, an inference Web-engine for querying XML Documents (FCHGD09b),

Paulo Jorge Oliveria: SMARTCLEAN: uma ferramenta para a limpeza
incremental de dados (ORH09a),

Maria de F□tima Rodrigues: SMARTCLEAN: uma ferramenta para a limpeza
incremental de dados (ORH09a),

Rafael Teod□sio Pereira: Uma metodologia para Consultas aos Bancos de
Dados do NCBI (LPH09a),

Bastian Cramer: {VisualLISA}: A Domain Specific Visual Language for
Attribute Grammars (OPHCC09), VisualLISA: A Visual Environment to Develop
Attribute Grammars (OPHCC2010),

Nuno Oliveira : Comparison of XAML and C\# Forms using Cognitive
Dimensions Framework (MKCHCPO09),

Pedro Henriques : Assessing Databases in .Net: comparing approaches (
CH09d),

Matej Crepinsek: Comparing General-Purpose and Domain-Specific Languages:
An Empirical Study (KOMPCCH2010),

Danielada Cruz: Comparing General-Purpose and Domain-Specific Languages:
An Empirical Study (KOMPCCH2010),
```

Figura 4: Resultado da execução do programa

3.3.4 Alínea d

Demonstra-se abaixo o ficheiro resultante para o input author = "P.R. Henriques", juntamente com o desenho do grafo já processado a partir de uma ferramenta de Graphviz online[1]. É de notar que não seriam necessários os parâmetros ratio, node e color, adicionados apenas por uma questão de estética.

```
digraph G {
  ratio = fill;
  node [style=filled];
  "p.r. henriques" -> "f. mario martins" [color="0.650 0.700 0.700"];
  "p.r. henriques" -> "j.j. almeida" [color="0.650 0.700 0.700"];
  "p.r. henriques" -> "j.c. ramalho" [color="0.650 0.700 0.700"];
  "p.r. henriques" -> "j.g. rocha" [color="0.650 0.700 0.700"];
  "p.r. henriques" -> "m.r. henriques" [color="0.650 0.700 0.700"];
}
```

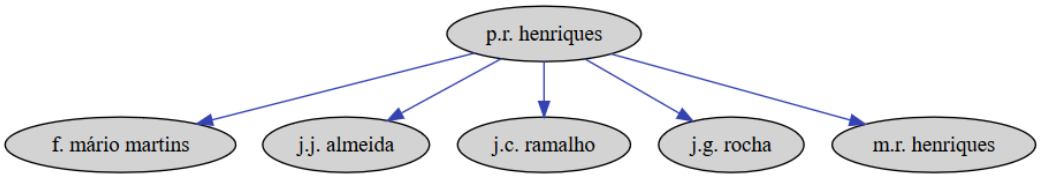


Figura 5: Resultado do grafo para o autor "P.R. Henriques"

Apresenta-se, também, o output do programa quando não é inserido um autor para análise.

```
Diana@DESKTOP-E77INGS /home/pl
$ ./e1p4 < exemplo-utf8.bib > result.txt
Argumentos insuficientes!
```

Figura 6: Resultado do programa na falta de inserção de um autor

4 Conclusões e Sugestões

A resolução deste segundo exercício prático foi bastante importante e enriquecedora, pois permitiu aos membros do grupo perceber e interiorizar melhor os conceitos abordados nas aulas práticas da Unidade Curricular de Processamento de Linguagens. Deste modo, foram adquiridos conhecimentos mais sólidos acerca de expressões regulares e ações semânticas, e da utilidade que o gerador *FLEX* pode ter num contexto real. De facto, foi possível perceber que a exigência de programar algo equivalente ao desenvolvido em Java (linguagens de maior domínio do grupo), por exemplo, seria muito maior. Assim, o grupo ficou satisfeito por ter desenvolvido este trabalho, de complexidade bastante superior ao primeiro exercício da unidade curricular, em GAWK.

No que toca ao enunciado 1, em geral, uma das dificuldades residiu, em primeira instância, na escolha das expressões regulares, de modo a prever todos os casos de exceção, tornando os programas mais robustos. Ainda assim, o grupo considera que o maior obstáculo, e fase mais dispendiosa de tempo, consistiu na elaboração de algoritmos que processassem ou transformassem os *datasets* nos resultados desejados.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu tirar partido dos conhecimentos adquiridos neste projeto, sentido-se capaz de, num contexto futuro, aplicar os conceitos subjacentes de forma eficaz.

Referências

[1] *WebGraphviz is Graphviz in the Browser*. URL: http://www.webgraphviz.com/. (accessed: 04.05.2018).