Processamento de Linguagens e Compiladores

Trabalho Prático 1

Relatório de Desenvolvimento

Gilberto Cunha
A89142

Tomás Carneiro
A82552

16 de novembro de 2020

Conteúdo

Capítulo 1

Introdução

Neste trabalho pretende-se utilizar expressões regulares recorrendo ao gerador `Flex` para o processamento de documentos do tipo `BibTeX`.

Este processamento tem como objetivo traduzir a informação formatada em `BibTex` para `html` e também para `dot`, permitindo a geração de grafos para uma visualização mais prática da informação presente no documento original.

Para simplificar o uso dos processadores de texto desenvolvidos, foi também criada uma `MakeFile`, que será posteriormente discutida, que permite executar as tarefas de pesquisa e listagem de autores, bem como a geração do ficheiro `html` e do grafo com comandos simples de `make`.

1.1 BibTeX

`BibTeX` é uma ferramenta de formatação de citações bibliográficas em documentos LaTeX, que permite a estruturação da bibliografia de um documento LaTeX.

Em documentos deste tipo, cada entrada na sua base de dados textual tem o seguinte aspeto:

```
@techreport{Dean:2004,
  author = {Dean, Mike and Schreijber, Guus},
  title = {OWL Web Ontology Language Reference},
  type = {Recommendation},
  year = 2004,
}
```

Cada referência, desta forma, consiste numa:

1) **Categoria**, identificada com um `@` (que no exemplo acima seria "techreport");

2) **Chave**, localizada imediatamente a seguir ("Dean:2004");

3) **Lista de identificadores**, podendo estes ser `author, title, type`, entre outros.

1.2 HTML

Documentos `html` são utilizados para a construção de páginas Web. Estes documentos estão geralmente estruturados da seguinte forma:

```
<!DOCTYPE html>
<html>
    <head>
    <title>Page Title</title>
    </head>
<body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
</body>
</html>
```

Para a tradução de `Flex` para `html` foi alterada a tag de `<title>` e criadas novas tags de headings (`<h1>, <h2>, ...`) e de listas não ordenadas `<ul>`, para colocar nessas tags informação do ficheiro `BibTex` que será descrita na secção 2.1.

1.3 Dot

Para a tarefa de geração de grafos foi utilizada a linguagem `dot` do `graphviz`. O seguinte código exemplifica a estrutura de um documento `dot` que gera um grafo não direcionado:

```
    graph G {
        layout = fdp;
        0 -- 1;
        0 -- 3;
        0 -- 4;
        0 [label="Bastian"];
        1 [label="Nuno"];
        3 [label="Pedro"];
        4 [label="Daniela"];
    }
```

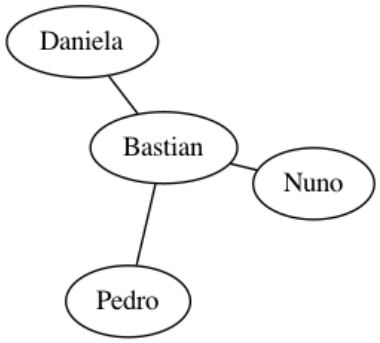

Figura 1.1: Grafo gerado pelo código `dot` à esquerda

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O problema subdivide-se em quatro secções:

1) **Contagem de Categorias**:
   Analisando o documento BibTeX, contar o número de ocorrências de uma determinada categoria, gerando posteriormente um documento `html` com os nomes das categorias encontradas e respetivas contagens.

2) **Filtragem da `chave`, `titulo` e `autores`**:
   Para cada categoria do documento, filtrar a respetiva `chave`, `titulo` e `autores`, incluindo o resultado no documento `html` gerado anteriormente.

3) **Criação de um índice de `autores`**:
   Para cada autor, mapeá-lo nas respetivas publicações, organizando-os de forma a permitir uma posterior pesquisa, utilizando uma ferramenta de procura em Linux.

4) **Construção de um grafo que mostre autores com publicações em comum**:
   Dado um autor, mostrar todos os autores com publicações em comum com o autor em causa, recorrendo à linguagem `dot` do `GraphViz` para desenho do respetivo grafo.

Capítulo 3

Concepção da Solução

3.1 Estruturas de Dados

Para o desenvolvimento do programa, optou-se pela utilização de listas ligadas para armazenamento de dados, uma vez que estas são de muito simples implementação, permitindo-nos focar mais na especificação `Flex` do que no código `C`. Apesar da sua ineficiência no acesso a elementos da lista, a grande maioria das operações usadas são inserções com comparação a elementos já pertencentes às listas, onde a desvantagem das listas ligadas não é tão notável.

Foram então criadas estruturas de listas ligadas onde são inseridos os padrões captados pelas Expressões Regulares do `Flex`. Após o armazenamento dos padrões de todo o ficheiro de input é utilizado um método `Show`, desenvolvido para cada estrutura, de modo a escrever esta informação armazenada na linguagem pretendida (`html` e `dot`).

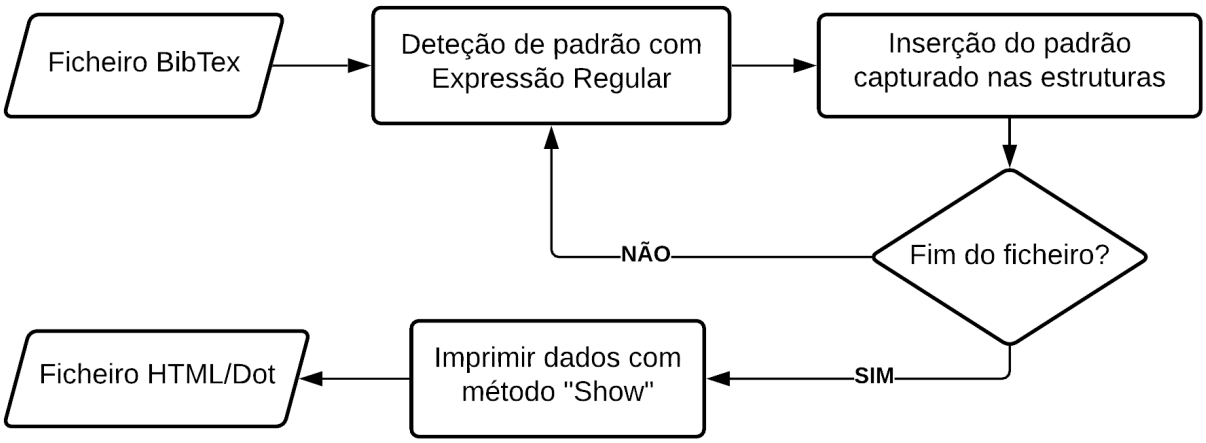

Figura 3.1: Diagrama de fluxo da aplicação das estruturas desenvolvidas em `C`

Para a contagem de categorias e filtragem da `chave`, `titulo` e `autores`, foram criadas três estruturas, sendo elas: `[struct categoria]`, `[struct projeto]` e `[struct autores]`, ilustradas no diagrama abaixo:

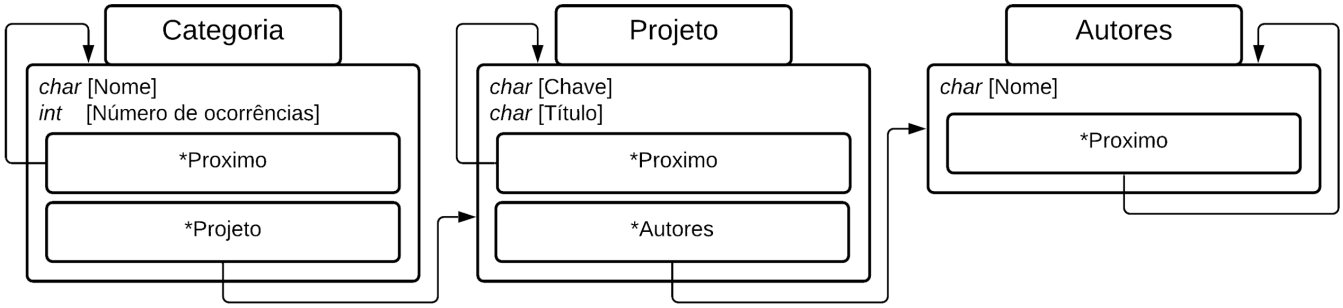

Figura 3.2: Estrutura desenvolvida para armazenar dados das categorias

De forma a criar um índice de autores, foram adicionadas duas estruturas: `[struct autor]` e `[struct publicacoes]`. Note-se que a `[struct autor]` não corresponde à estrutura previamente criada - `[struct autores]` - uma vez que a utilidade desta está em organizar cada autor de acordo com as publicações respetivas (e não organizar projetos de acordo com os seus autores). Podem observar-se abaixo:

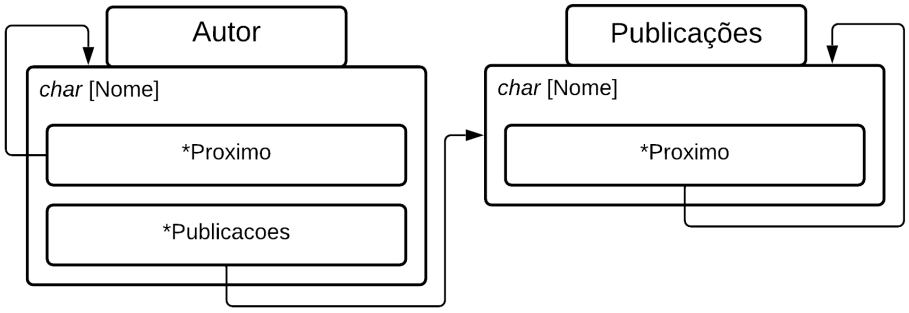

Figura 3.3: Estrutura desenvolvida para armazenar as publicações de cada autor

Por último, para gerarmos o nosso grafo dos autores, criámos as estruturas `[struct Grafo]` e `[struct Nodo]`. Novamente, podem ser visualizadas as respetivas representações:

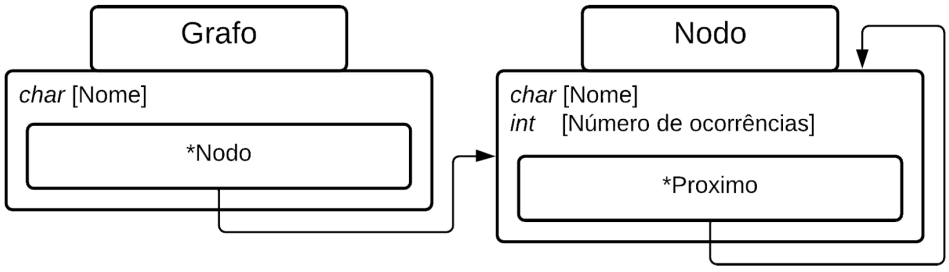

Figura 3.4: Estrutura desenvolvida para armazenar dados da geração do grafo

3.2 Estruturação do programa

O programa desenvolvido é constituído por dois filtros em cadeia, como ilutrado na seguinte figura:

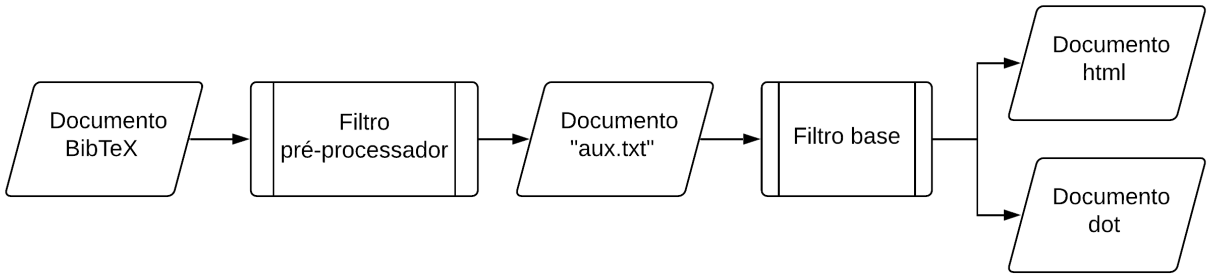

Figura 3.5: Fluxograma que ilustra o encadeamento dos filtros desenvolvidos

3.3 Filtro pré-processador

De forma a uniformizar um pouco mais o nosso ficheiro de input, e desse modo simplificar o nosso `filtro_base`, desenvolvemos um filtro pré-processador. Este novo filtro altera o formato do ficheiro original BibTeX, que é inicialmente:

```
@techreport{Dean:2004,
  author = {Dean, Mike and Schreijber, Guus},
  title = {OWL Web Ontology Language Reference},
  type = {Recommendation},
  year = 2004,
}
```

Para um novo formato, com o seguinte aspeto:

```
@techreport{Dean:2004,
AUTOR$$Mike Dean$$Guus Schreijber$},
TITULO$$OWL Web Ontology Language Reference$},
  type = {Recommendation},
  year = 2004,
}
```

Este novo filtro tem, como principais focos: marcar os inícios e fechos de títulos e autores com caracteres '$'; a separação de todos os nomes, anteriormente separados por 'and', recorrendo novamente à utilização do caracter '$'.

Note-se que, em ambos os nomes dos autores no exemplo acima, o apelido e primeiro nome estavam separados por vírgulas, e no novo formato, passam a estar trocados, sem a vírgula. Para isto, criámos uma função `swap_comma`, que irá efetuar esta formatação. Esta convenção foi adotada para nos permitir identificar autores cuja ocorrência nas publicações era referida ambiguamente com recurso aos dois formatos ([`apelido , nome`] e [`nome apelido`]), fazendo com que estes nomes sejam agora lidos como o mesmo, e não dois diferentes.

Neste filtro, são utilizadas apenas duas *Start Conditions*:

```
%x AUTOR TITULO
```

Que são inicializadas da seguinte forma:

```
1  ^\ +(?i:author)\ *\=\ *(\{|\")\ *        { printf ("AUTOR$$");  BEGIN AUTOR;}
2  ^\ +(?i:title)\ *\=\ *(\{|\")\ *         { printf ("TITULO$$"); BEGIN TITULO;}
```

É fácil de reparar que as Expressões Regulares que identificam tanto o título como o autor, são fundamentalmente a mesma.

Quando forem detetadas formatações de LaTeX, estão são ignoradas:

```
1  <*>\\text(sc|it|bf)         { ; }
2  <*>\\(underline|emph)       { ; }
```

Caso seja detetado um título, o seu identificador original é substituído por `"TITULO$$"` e a sua *Start Condition* é inicializada:

```
1  <TITULO>(\}|\")\,                        { printf ("$},"); BEGIN INITIAL; }
2  <TITULO>[\n\r\t\{\}]                     { ; }
3  <TITULO>.                                { ECHO; }
```

A marca que sinaliza o final de um título é sempre (},) ou (",). Caso o final seja detetado, este será substituído por ($},), e a *Start Condition* INITIAL será inicializada.

Caso sejam detetados caracteres que sejam quebras de linha, *tabs*, *carriage return* ou chavetas, estes são ignorados.

Todo o texto que não esteja inserido nestes casos é simplesmente devolvido utilizando o comando `ECHO`.

Caso um autor seja detetado, da mesma forma, o seu identificador irá ser substituído por `"AUTOR$$"`, e a sua *Start Condition* inicializada.

Detetando uma separacão de nomes, sinalizada sempre por 'and', a nossa *Start Condition* executa os seguintes comandos:

```
1  <AUTOR>((\ |\n|\r|\t)+and(\ |\n|\r|\t)+|\ and\ and\ )   {
2      swap_comma (autor, aux);
3      acrescentaNodo (&autores, autor);
4      printf ("%s$$", autor);
5      memset(autor, 0, strlen(autor));
6  }
```

Na linha 2, a função `swap_comma` supra-mencionada é executada.

Na linha 3, o nome do autor já formatado é acrescentado à `struct nodo` (ver Figura 3.4), utilizando a função `acrescentaNodo`. O aspeto final desta lista será uma lista de autores, todos com nomes diferentes, com o número de ocorrências respetivamente associado.

De seguida, na linha 4, o nome do autor é devolvido, seguido pelos caracteres '$$'. A utilização de dois caracteres '$' visa uniformizar a estrutura da expressão regular utilizada na deteção dos nomes, uma vez que nos permite, desta forma, diferenciar **qualquer** autor pelos limites `"$nome do autor$"`.

Na última linha, todo o conteúdo da `string autor` é eliminado, de forma a permitir acrescentar os nomes

seguintes.

No caso em que encontremos o final da lista de autores, marcada por (}",) ou por (",), a *Start Condition* tem o seguinte funcionamento:

```
1 <AUTOR>\ *(\}|\")\,     {
2     swap_comma (autor, aux);
3     acrescentaNodo (&autores, autor);
4     printf ("%s$},", autor); BEGIN INITIAL;
5     memset(autor, 0, strlen(autor));
6 }
```

Esta secção de código executa fundamentalmente as mesmas instruções do que a anterior, no entanto, o nome que é captado corresponde ao último presente nesta lista de autores. Podemos então observar na linha 4 que o texto devolvido corresponderá ao nome do autor seguido de "$},", estando, desta forma, de acordo com a convenção escolhida anteriormente.
Uma vez que ocorre o término desta *Start Condition*, voltamos ao nosso estado inicial do programa, predefinido pela *Start Condition* INITIAL.

No decorrer do desenvolvimento deste filtro, fomo-nos deparando com diferentes casos específicos em que a formatação do ficheiro "exemplo-utf8.bib" implicava a implementação de expressões regulares adicionais, objetivadas para a resolução destas excepções. A *Start Condition* AUTOR, nestes casos, tem o seguinte funcionamento:

```
1 <AUTOR>(M?rio\ B?ron|Mario\ B?ron|Mario\ Ber?n) { strcat (autor, "Mario Beron"); }
2 <AUTOR>(\n|\r|\t)+\ *                           { strcat (autor, " "); }
3 <AUTOR>\ *\,\ *                                 { strcat (autor, ","); }
4 <AUTOR>?                                        { strcat (autor, "?"); }
5 <AUTOR>(\{|\}|\\\'|\\\~|\\\^|\\\")              { ; }
6 <AUTOR>\ +                                      { strcat (autor, " "); }
```

Finalmente, caso nenhuma das Expressões Regulares mencionadas acima seja detetada, isto significa que o filtro está a ler **um nome**:

```
1 <AUTOR>.                                        { strcat (autor, yytext); }
```

Fora destas *Start Conditions*, não pretendemos fazer alterações ao documento original, por isso, a seguinte instrução faz "ECHO" do texto restante:

```
1 (.|\n)                                    { ECHO; }
```

3.4 Filtro Base

Para o filtro base, é necessário detetar e armazenar os campos Categoria, Autores e Titulo. Para atingir este objetivo, criaram-se as seguintes *Start Conditions*:

```
1 %x CATEGORIA CHAVE DENTRO AUTORES TITULO
```

Cada CATEGORIA está identificada por um '@' no início e uma chaveta '{' no final, seguida de uma chave.

Todos os caracteres que se encontrarem entre o '@' e a chaveta '{' são guardados para uma variável auxiliar `char *nome`. Esta *Start Condition* pode então descrita com as seguintes instruções `Flex`:

```
\@                          { BEGIN CATEGORIA; }
<CATEGORIA>[^\{]+           { nome = strdup (str_to_lower (yytext)); }
<CATEGORIA>\{               { BEGIN CHAVE; }
```

Repare-se que é feita a conversão do nome da categoria para minúsculas, usando a função `str_to_lower`. Esta conversão para minúsculas teve de ser feita, uma vez que existiam categorias com o mesmo nome que ao serem armazenadas não eram identificadas como tal, pois diferiam na capitalização das suas letras (ex: inproceedings e InProceedings).

Uma vez dentro da *Start Condition* `CHAVE`, após sair da `CATEGORIA`, são guardados todos os caracteres numa variável auxiliar `char *chave` até ser encontrado o caracter de término ','.

```
<CHAVE>[^\,]+           { chave = strdup (yytext); }
<CHAVE>\,               { BEGIN DENTRO; }
```

Quando se sai da *Start Condition* `CHAVE`, é inicializada uma nova *Start Condition* denominada `DENTRO`, que é utilizada apenas para sabermos que nos encontramos entre a `CHAVE` e o término de cada uma das referências `BibTex` (tudo que se encontre nos campos `authors, title, type, etc` do exemplo apresentado na sec. 3.3). Nesta *Start Condition* é-nos então possível detetar o início das restantes *Start Conditions* `TITULO` e `AUTORES`:

```
<DENTRO>AUTOR\$             { BEGIN AUTORES; }
<DENTRO>TITULO\$            { BEGIN TITULO; }
```

Note-se que a maneira como se deteta o começo dos autores e do título depende do filtro pré-processador descrito acima: quando é encontrado o padrão `AUTORES$` entra-se na *Start Condition* `AUTORES` e quando é encontrado o padrão `TITULO$` entra-se na *Start Condition* `TITULO`.

A *Start Condition* `TITULO` deteta todos os caracteres que se encontrem entre os caracteres '$' (no início e no fim). Quando forem encontrados os caracteres de término '},' como exige o filtro pré-processador, então regressamos à *Start Condition* `DENTRO`. O título é também guardado numa variável auxiliar `char *titulo`:

```
<TITULO>\$[^\$]+\$      {
    yytext[strlen(yytext)-1] = '\0';
    titulo = strdup (yytext+1);
    BEGIN DENTRO;
}
```

É importante reparar que são eliminados o primeiro e o último caracteres do padrão detetado relativo ao título, para não serem guardados os caracteres '$' usados para delimitação no filtro pré-processador.

Uma vez dentro da *Start Condition* `AUTORES`, analogamente ao referido acima para o título, vão ser captados todos os caracteres diferentes de '$' e removidos o primeiro e último caracteres. Este nome é também acrescentado a uma lista ligada auxiliar de autores `a`, utilizando a função `acrescentaLStr`.

```
<AUTORES>\},                          { BEGIN DENTRO; }
<AUTORES>\$[^\$]+\$                   {
    yytext[strlen(yytext)-1] = '\0';
    acrescentaLStr (&a, yytext+1);
}
```

Por fim, quando uma dada referência BibTex terminar, isto é, quando saírmos da nossa *Start Condition* `DENTRO` ao encontrar o caracter mudança de linha seguido de um número arbitrário de espaços e uma chaveta '}', sabemos que todos os dados foram adquiridos e que podemos finalmente guardá-los nas estruturas descritas na secção 3.1.

```
<DENTRO>\n\ *\}                       {
    acrescentaProj (&p, chave, titulo, a);
    acrescentaCat (&l, nome, p);
    acrescentaGrafo (grafo, a);
    while (a != NULL) {
        acrescentaAut (&autores, a->nome, titulo);
        a = a->prox;
    }
    BEGIN INITIAL; p = NULL;
}
```

No código acima apresentado, é inicializado na 2ª linha um novo projeto, utilizando a chave, título e autores guardados nas variáveis auxiliares `chave`, `titulo` e `a`. Este projeto é posteriormente utilizado para inserção na sua respetiva categoria na linha 3 (estrutura da Figura 3.2). Na linha 4 é feita a inserção dos nomes da lista de autores `a` caso o nome do autor de input ocorra nesta lista, utilizando a função auxiliar `acrescentaGrafo` (estrutura da Figura 3.4). Por fim, é também acrescentado o título deste projeto a cada um dos autores pertencentes ao mesmo (estrutura da Figura 3.3).

Qualquer outro padrão que não se enquadre em todos os outros previamente descritos neste filtro base é ignorado. Esta condição é válida para todas as *Start Conditions* definidas:

```
<*>(.|\n)                     { ; }
```

A impressão de todos os dados armazenados é feita na função `main` deste filtro (ver Apêndice A.2), onde é gerado o ficheiro `dot` do grafo, criada uma imagem a partir desse ficheiro e posteriormente colocada também essa imagem no ficheiro `html`. Esta impressão é feita com três funções auxiliares definidas em `C` (ver apêndice A.4): `ShowCat`, `ShowAut` e `ShowGraph`.

3.5 MakeFile

Para permitir o uso das ferramentas desenvolvidas de uma maneira mais simples, foi criada uma `MakeFile` com três comandos principais:

- `make find_author name="[nome_autor]" file="[nome_fich]"`: que encontra o autor "[nome_autor]" e todas as suas publicações no ficheiro "[nome_ficheiro]"
- `make list_authors file="[nome_fich]"`: lista todos os autores presentes no ficheiro "[nome_fich]"

- `make run name="[nome_autor]" file="[nome_fich]"`: gera o ficheiro `html` para o ficheiro original "[nome_fich]" e o ficheiro `dot` para o autor "[nome_autor]" e ficheiro original "[nome_fich]"

3.6 Testes e exemplos

Os testes abaixo demonstrados foram feitos utilizando, como ficheiro de input, o exemplo presente no Apêndice B:

- Lista de autores presentes no ficheiro `"exemplo.bib"`:

```
$ make list_authors file="exemplo.bib"

                            LISTA DE AUTORES

Gilberto Cunha              Tomás Carneiro                  P.R. Henriques
Pedro Moura
```

- Lista de publicacões pelo autor "Pedro Moura":

```
$ make find_author name="Pedro Moura" file="exemplo.bib"

Pedro Moura - 2 Publicações
        - Como lecionar PLC
        - Projeto de PLC
```

- Ficheiro `html` de saída, relativo às categorias presentes no `exemplo.bib`:

```
$ make run name="P.R. Henriques" file="exemplo.bib"
```

Figura 3.6: *Tab* Categorias do ficheiro `html`

Figura 3.7: *Tab* Autores do ficheiro `html`

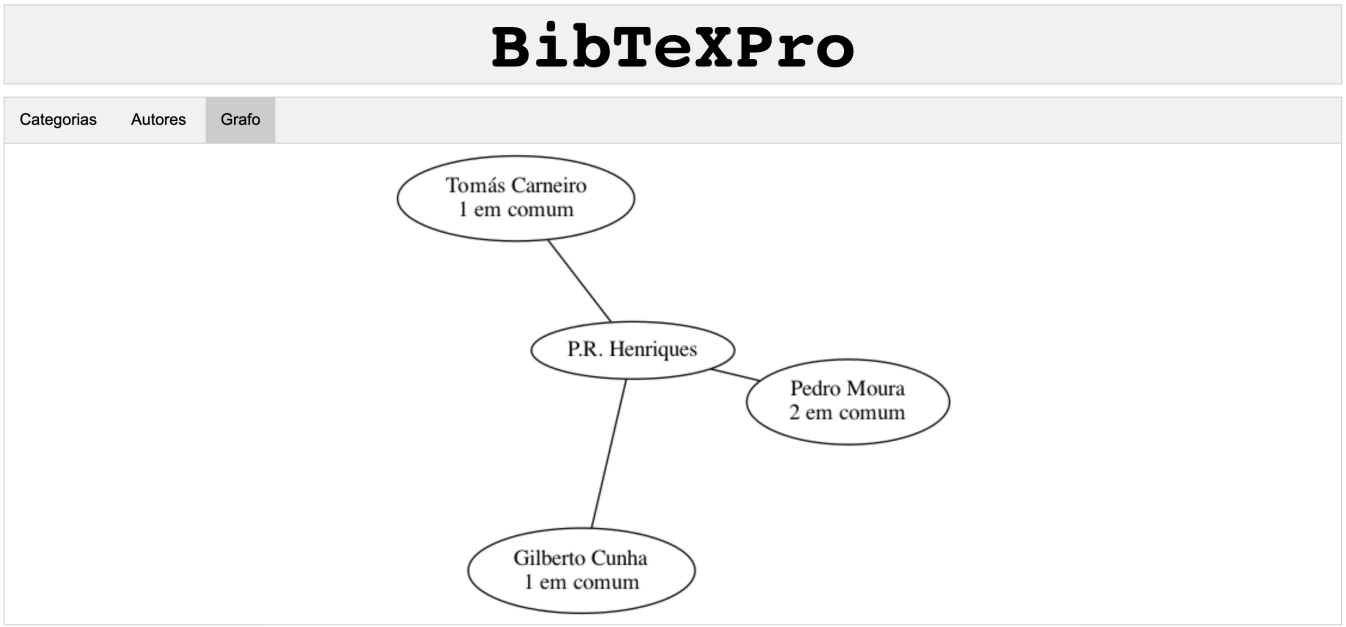

Figura 3.8: *Tab* Grafo do ficheiro `html`

- Grafo representativo dos autores com publicações em comum com "P.R. Henriques"

  ```
  $ make run name="P.R. Henriques" file="exemplo.bib"
  ```

  Após executar este comando `make`, são gerados os ficheiros `graph.dot` e `graph.png`:

```
graph G {
        layout = fdp;
        0 -- 1;
        0 -- 2;
        0 -- 3;
        0 [label="P.R. Henriques"];
        1 [label="Pedro Moura
        2 em comum"];
        2 [label="Gilberto Cunha
        1 em comum"];
        3 [label="Tomás Carneiro
        1 em comum"];
}
```

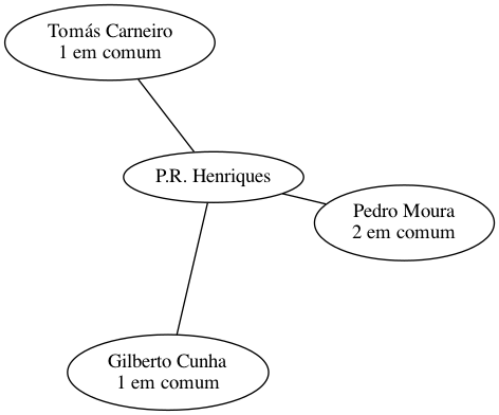

Figura 3.9: Grafo do código `dot` (à esquerda) gerado a partir do documento `exemplo.bib`

Capítulo 4

Conclusão

Elaborar o processador `BibTexPro` permitiu-nos verificar que a tradução intra e extra linguagens tem uma vasta utilidade e aplicações, desde a simplificação de visualização de dados, como tratado neste trabalho, ao desenvolvimento de compiladores. É também de notar a enorme simplicidade da ferramenta `Flex` para criar este tipo de processadores textuais.

Utilizando a ferramenta `Flex` conseguiu-se, para qualquer ficheiro BibTex seguindo correta e consistentemente todas as normas de formatação, que programa desenvolvido funcionasse devidamente e apresentasse ainda uma gama acrescida de formatações, não inteiramente corretas em BibTex, que são contempladas.

Foram ainda desenvolvidos melhoramentos ao programa proposto através do uso de uma `MakeFile`, permitindo-nos não só executar estas tarefas através de um terminal Linux com um único comando, mas também a execução da tarefa de listagem de todos os autores de um documento BibTex.

A maior dificuldade a elaborar este trabalho foi aceitar a inevitável limitação do mesmo face a erros de formatação do ficheiro original. Apesar de querermos que este funcione para qualquer caso (mesmos nomes escritos/abreviados de diferentes maneiras, caracteres desconhecidos, etc...), deve sempre haver um conjunto de regras de formatação que devem ser obedecidas para um programa funcionar corretamente.

Com os conhecimentos resultantes do estudo de processadores de texto, conseguiu-se identificar possíveis aplicações de grande utilidade e interesse com estas ferramentas. Uma possível aplicação discutida seria a utilização destes processadores para a tradução entre diferentes versões da mesma linguagem, permitindo que programas desenvolvidos em versões já deprecadas consigam ser automatica e continuamente atualizados.

Apêndice A

Código do programa

A.1 Código Flex do filtro pré-processador

```
%{
#include <stdio.h>
#include <string.h>
#include "funcs.h"

LNodo autores = NULL;
char autor[100];
char aux[100];
%}
%option noyywrap

%x AUTOR TITULO

%%
^\ +(?i:author)\ *\=\ *(\{|\")\ *        { printf ("AUTOR$$"); BEGIN AUTOR; }
^\ +(?i:title)\ *\=\ *(\{|\")\ *         { printf ("TITULO$$"); BEGIN TITULO; }
<*>\\text(sc|it|bf)                      { ; }
<*>\\(underline|emph)                    { ; }

<AUTOR>([\ \n\r\t]+and[\ \n\r\t]+|(\ and)+\ )   {
    swap_comma (autor, aux);
    acrescentaNodo (&autores, autor);
    printf ("%s$$", autor);
    memset(autor, 0, strlen(autor));
}
<AUTOR>\ *(\}|\")\,                          {
    swap_comma (autor, aux);
    acrescentaNodo (&autores, autor);
    printf ("%s$},", autor); BEGIN INITIAL;
    memset(autor, 0, strlen(autor));
}
<AUTOR>(Mrio\ Bron|Mario\ Bron|Mario\ Bern) { strcat (autor, "Mario Beron"); }
```

```
<AUTOR>(\n|\r|\t)+\ *                          { strcat (autor, " "); }
<AUTOR>\ *\,\ *                                { strcat (autor, ","); }
<AUTOR>                                       { strcat (autor, "?"); }
<AUTOR>(\{|\}|\\\'|\\\~|\\\^|\\\")             { ; }
<AUTOR>\ +                                     { strcat (autor, " "); }
<AUTOR>.                                       { strcat (autor, yytext); }

<TITULO>(\}|\")\,                        { printf ("$},"); BEGIN INITIAL; }
<TITULO>[\n\r\t\{\}]                     { ; }
<TITULO>.                                { ECHO; }

(.|\n)                                   { ECHO; }
%%

int main () {
    FILE *f = fopen ("lista_autores.txt", "w");
    yylex();
    ShowAuthorTable(f, autores);
    fclose (f);
    return 0;
}
```

A.2 Código Flex do filtro base

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

FILE *file;
LCat l = NULL; LProj p = NULL; LStr a = NULL;
LAut autores = NULL; Graph grafo = NULL;
char *nome, *chave, *titulo;
%}

%option noyywrap
%x CATEGORIA DENTRO CHAVE AUTORES TITULO

%%
\@                                  { BEGIN CATEGORIA; }
<CATEGORIA>[^\{]+                   { nome = strdup (str_to_lower (yytext)); }
<CATEGORIA>\{                       { BEGIN CHAVE; }
<DENTRO>[\n\r]\ *\}                 {
    acrescentaProj (&p, chave, titulo, a);
    acrescentaCat (&l, nome, p);
    acrescentaGrafo (grafo, a);
```

```
24      while (a != NULL) {
25          acrescentaAut (&autores, a->nome, titulo);
26          a = a->prox;
27      }
28
29      BEGIN INITIAL; p = NULL;
30  }
31
32  <CHAVE>[^\,]+                       { chave = strdup (yytext); }
33  <CHAVE>\,                           { BEGIN DENTRO; }
34
35  <DENTRO>AUTOR\$                     { BEGIN AUTORES; }
36  <AUTORES>\}\,                       { BEGIN DENTRO; }
37  <AUTORES>\$[^\$]+\$                 {
38      yytext[strlen(yytext)-1] = '\0';
39      acrescentaLStr (&a, yytext+1);
40  }
41
42  <DENTRO>TITULO\$                    { BEGIN TITULO; }
43  <TITULO>\$[^\$]+\$                  {
44      yytext[strlen(yytext)-1] = '\0';
45      titulo = strdup (yytext+1);
46      BEGIN DENTRO;
47  }
48
49  <*>(.|\n)                           { ; }
50  %%
51
52  int main (int argc, char **argv) {
53      initGraph (&grafo, argv[1]);
54      yylex();
55
56      if (strcmp(argv[2], "html") == 0) {
57          printHTMLstart ();
58
59          ShowGraph (&grafo, "graph.dot");
60          system("dot -Kfdp -Tpng -Goverlap=false -Gsplines=true graph.dot > graph.png");
61
62          printf ("<div id=\"Categorias\" class=\"tabcontent\">\n");
63          ShowCat (&l);
64          printf ("</div>\n");
65          printf ("<div id=\"Autores\" class=\"tabcontent\">\n");
66          ShowAut (&autores);
67          printf ("</div>\n");
68          printf ("<div id=\"Grafo\" class=\"tabcontent\">\n");
69          printf ("<img src=\"graph.png\" alt=\"Grafo dos autores em comum de %s\"", argv[1]);
70          printf ("width=\"600\" class=\"center\">\n");
71          printf ("</div>\n");
```

```
72
73          printHTMLend ();
74      }
75      else if (strcmp(argv[2], "index") == 0) {
76          FILE *f = fopen ("author.txt", "w");
77          while (autores != NULL && strcmp(autores->nome, argv[1]) != 0) autores = autores->prox
78          if (autores != NULL) {
79              autores->prox = NULL;
80              ShowAutF (&autores, f);
81          }
82          else fprintf (f, "\nERROR: This author can't be found in the specified file\n");
83          fclose (f);
84      }
85      return 0;
86  }
```

A.3 Código MakeFile

```
1   run: name_filter
2           @flex filtrobase.l
3           @gcc lex.yy.c funcs.c
4           @./a.out "$(name)" "html" < aux.txt > saida.html
5
6   find_author: name_filter
7           @flex filtrobase.l
8           @gcc lex.yy.c funcs.c
9           @./a.out "$(name)" "index" < aux.txt > aux2.txt
10          @rm aux.txt
11          @rm aux2.txt
12          @cat author.txt
13          @rm author.txt
14
15  name_filter:
16          @flex name_filter.l
17          @gcc lex.yy.c funcs.c
18          @./a.out < "$(file)" > aux.txt
19
20  list_authors:
21          @flex name_filter.l
22          @gcc lex.yy.c funcs.c
23          @./a.out < $(file)$ > aux.txt
24          @cat lista_autores.txt
25
26  clean:
27          @rm a.out
28          @rm lex.yy.c
29          @rm aux.txt
```

```
30          @rm lista_autores.txt
31          @rm graph.dot
32          @rm graph.png
33          @rm saida.htlm
```

A.4 Código C

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4  #include <stdlib.h>
5  #include "funcs.h"
6
7  void swap_comma (char nome[], char a[]) {
8      int i;
9      for (i=0; nome[i]!='\0' && nome[i]!=','; ++i);
10     if (i != strlen(nome)) {
11         strcat (a, nome+i+1);
12         strcat (a, " ");
13         nome[i] = '\0';
14         strcat (a, nome);
15         strcpy (nome, a);
16     }
17     memset(a, 0, strlen(a));
18 }
19
20 char *str_to_lower (char *s) {
21     for (int i=0; i<strlen(s); ++i) s[i] = tolower(s[i]);
22
23     return s;
24 }
25
26 void ShowLStr (LStr *l) {
27     while ((*l) != NULL && (*l)->prox != NULL) {
28         printf ("%s -> ", (*l)->nome);
29         l = &((*l)->prox);
30     }
31     if ((*l) != NULL) printf ("%s\n", (*l)->nome);
32 }
33
34 void acrescentaLStr (LStr *l, char *s) {
35     while ((*l) != NULL) l = &((*l)->prox);
36
37     (*l) = malloc (sizeof (struct slist));
38     (*l)->nome = strdup (s);
39     (*l)->prox = NULL;
40 }
```

```
41
42  void ShowProj (LProj *p) {
43      while ((*p) != NULL) {
44          printf ("Chave: %s\n", (*p)->chave);
45          printf ("Título: %s\n", (*p)->titulo);
46          printf ("Autores: ");
47          ShowLStr (&((*p)->autores));
48          printf ("\n");
49          p = &((*p)->prox);
50      }
51  }
52
53  void copiaLStr (LStr *source, LStr *dest) {
54      *dest = NULL;
55      while (*source != NULL) {
56          (*dest) = malloc (sizeof (struct slist));
57          (*dest)->nome = (*source)->nome;
58          (*dest)->prox = NULL;
59          source = &((*source)->prox);
60          dest = &((*dest)->prox);
61      }
62      *dest = NULL;
63  }
64
65  void acrescentaProj (LProj *p, char *chave, char *titulo, LStr autores) {
66      while (*p != NULL) p = &((*p)->prox);
67
68      (*p) = malloc (sizeof (struct projeto));
69      (*p)->chave = strdup (chave);
70      (*p)->titulo = strdup (titulo);
71      (*p)->autores = autores;
72      (*p)->prox = NULL;
73  }
74
75  void acrescentaNodo(LNodo *n, char *nome) {
76      while (*n != NULL && strcmp((*n)->nome, nome) != 0) n = &((*n)->prox);
77
78      if(*n == NULL){
79          (*n) = malloc(sizeof(struct nodo));
80          (*n)->nome = strdup(nome);
81          (*n)->num_ocorr = 0;
82          (*n)->prox = NULL;
83      }
84      (*n)->num_ocorr++;
85  }
86
87  void initGraph (Graph *g, char *nome) {
88      *g = malloc (sizeof (struct agraph));
```

```
89      (*g)->nome = strdup (nome);
90      (*g)->autores = NULL;
91  }
92
93  void acrescentaAut (LAut *a, char *name, char *pub) {
94      while (*a != NULL && strcmp((*a)->nome, name) != 0) a = &((*a)->prox);
95
96      if (*a == NULL) {
97          (*a) = malloc (sizeof (struct autor));
98          (*a)->nome = strdup (name);
99          (*a)->prox = NULL;
100     }
101     acrescentaLStr (&((*a)->public), pub);
102 }
103
104 void copiaLProj (LProj *source, LProj *dest) {
105     while (*dest != NULL) dest = &((*dest)->prox);
106     while (*source != NULL) {
107         *dest = malloc (sizeof (struct projeto));
108         (*dest)->chave = (*source)->chave;
109         (*dest)->titulo = (*source)->titulo;
110         copiaLStr (&((*source)->autores), &((*dest)->autores));
111         (*dest)->prox = NULL;
112         source = &((*source)->prox);
113         dest = &((*dest)->prox);
114     }
115     *dest = NULL;
116 }
117
118 void acrescentaCat (LCat *l, char *s, LProj p) {
119     while (*l != NULL && strcmp((*l)->nome, s) < 0)
120         l = &((*l)->prox);
121
122     if (*l == NULL) {
123         *l = malloc (sizeof (struct categoria));
124         (*l)->nome = strdup (s);
125         (*l)->num_ocorr = 1;
126         copiaLProj (&p, &((*l)->projeto));
127         (*l)->prox = NULL;
128     }
129     else if (strcmp ((*l)->nome, s) == 0) {
130         (*l)->num_ocorr++;
131         copiaLProj (&p, &((*l)->projeto));
132     }
133     else {
134         LCat new = malloc (sizeof (struct categoria));
135         new->nome = strdup (s);
136         new->num_ocorr = 1;
```

```
137         copiaLProj (&p, &(new->projeto));
138         new->prox = *l;
139         *l = new;
140     }
141 }
142 
143 void ShowCat (LCat *l) {
144     while (*l != NULL) {
145         printf("\t\t<h1> <b> Categoria %s</b> - %d ", (*l)->nome, (*l)->num_ocorr);
146         if ((*l)->num_ocorr > 1) printf ("ocorrências </h1>\n");
147         else printf ("ocorrência </h1>\n");
148 
149         LProj *sitio = &((*l)->projeto);
150         printf ("<ol>\n");
151         while (*sitio != NULL) {
152             printf ("\t\t\t<li style=\"font-size:1.5vw\"> <b> Título: %s </b> </li>\n", (*siti
153             printf ("\t\t\t\t<ul> \t\t<b>Chave:</b> %s </ul>\n", (*sitio)->chave);
154             printf ("\t\t\t\t<ul> \t\t<b>Autores:</b> ");
155 
156             LStr *sitio2 = &((*sitio)->autores);
157             while (*sitio2 != NULL && (*sitio2)->prox != NULL) {
158                 printf ("%s and ", (*sitio2)->nome);
159                 sitio2 = &((*sitio2)->prox);
160             }
161             if ((*sitio2) != NULL) printf ("%s", (*sitio2)->nome);
162             printf (" </ul>\n\n");
163             sitio = &((*sitio)->prox);
164         }
165         printf ("</ol>\n");
166 
167         l = &((*l)->prox);
168     }
169 }
170 
171 int contaPubs (LStr *a) {
172     int r = 0;
173 
174     while (*a != NULL) {
175         a = &((*a)->prox);
176         r += 1;
177     }
178 
179     return r;
180 }
181 
182 void ShowAut (LAut *a) {
183     int num;
184     printf ("\n");
```

```
185     while ((*a) != NULL) {
186         num = contaPubs (&((*a)->public));
187         printf ("<h1> %s - %d ", (*a)->nome, contaPubs (&((*a)->public)));
188         if (num > 1) printf ("Publicações\n </h1>\n");
189         else printf ("Publicação\n </h1>\n");
190 
191         LStr *sitio = &((*a)->public);
192         printf ("<ul>\n");
193         while (*sitio != NULL) {
194             printf ("<li>\t%s \n </li>", (*sitio)->nome);
195             sitio = &((*sitio)->prox);
196         }
197         printf ("</ul>\n");
198         a = &((*a)->prox);
199     }
200 }
201 
202 void ShowAutF (LAut *a, FILE *f) {
203     int num;
204     fprintf (f, "\n");
205     while ((*a) != NULL) {
206         num = contaPubs (&((*a)->public));
207         fprintf (f, "%s - %d", (*a)->nome, num);
208         if (num > 1) fprintf (f, "Publicações\n");
209         else fprintf (f, "Publicação\n");
210 
211         LStr *sitio = &((*a)->public);
212         while (*sitio != NULL) {
213             fprintf (f, "\t- %s \n", (*sitio)->nome);
214             sitio = &((*sitio)->prox);
215         }
216         fprintf (f, "\n");
217         a = &((*a)->prox);
218     }
219 }
220 
221 void ShowGraph (Graph *grafo, char *path) {
222     LNodo *aux = &((*grafo)->autores);
223     FILE *file = fopen(path, "w");
224     fprintf (file, "graph G {\n");
225     fprintf (file, "\tlayout = fdp;\n");
226     int num = 1;
227     while (*aux != NULL){
228         fprintf (file,"\t%d -- %d;\n", 0, num++);
229         aux = &((*aux)->prox);
230     }
231     aux = &((*grafo)->autores);
232     fprintf (file,"\t%d [label=\"%s\"];\n", 0,(*grafo)->nome);
```

```
233     num = 1;
234     while (*aux != NULL){
235         fprintf (file, "\t%d [label=\"%s\n%d em comum\"];\n",
236         num++,(*aux)->nome, (*aux)->num_ocorr);
237         aux = &((*aux)->prox);
238     }
239     fprintf (file,"}");
240     fclose(file);
241 }
242 
243 void acrescentaGrafo (Graph g, LStr auts) {
244     LStr *sitio = &auts;
245     while (*sitio != NULL && strcmp((*sitio)->nome, g->nome) != 0) {
246         sitio = &((*sitio)->prox);
247     }
248     if (*sitio != NULL) {
249         sitio = &auts;
250         while(*sitio!=NULL) {
251             if (strcmp((*sitio)->nome, g->nome) != 0) {
252                 acrescentaNodo (&(g->autores), (*sitio)->nome);
253             }
254             sitio = &((*sitio)->prox);
255         }
256     }
257 }
258 
259 void ShowAuthorTable (FILE *f, LNodo autores) {
260     int num_nomes, num_chars;
261     fprintf (f, "\n\n");
262     for (int i=0; i<8; ++i) fprintf (f, "     ");
263     fprintf (f, "LISTA DE AUTORES\n\n");
264     while (autores != NULL) {
265         for (num_nomes=0; num_nomes<3 && autores != NULL; ++num_nomes) {
266             for (int i=0; autores->nome[i]!='\0'; ++i) {
267                 fprintf (f, "%c", autores->nome[i]);
268             }
269             for (num_chars=strlen(autores->nome); num_chars<40; ++num_chars) {
270                 fprintf (f, " ");
271             }
272             autores = autores->prox;
273         }
274         fprintf (f, "\n");
275     }
276 }
277 
278 void printHTMLstart () {
279     printf ("<!DOCTYPE html>\n<html>\n<head>\n");
280     printf ("<title> Trabalho 1 </title>\n");
```

```
281     printf ("<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">\n");
282     printf ("<style>\nbody {\nfont-family: Arial;\n}\n");printf (".tab {\noverflow: hidden;\nb
283     printf ("img {\ndisplay: block;\nmargin-left: auto;\nmargin-right: auto;\n}\n");
284     printf (".tab button {\nbackground-color: inherit;\nborder: none;\noutline: none;\ncursor
285     printf ("padding: 14px 16px;\ntransition: 0.3s;\nfont-size: 17px;\n}\n");
286     printf (".tab button:hover {\nbackground-color: #ddd;\n}\n");
287     printf (".tab button.active {\nbackground-color: #ccc;\n}\n");
288     printf (".tabcontent {\ndisplay: none;\npadding: 6px 12px;\nborder: 1px solid #ccc;\nborde
289     printf ("</style>\n</head>\n<body>\n");
290     printf ("<pre class=\"tab\" style=\"text-align:center;\"> <h style=\"font-size:5vw\"> <b>
291     printf ("<div class=\"tab\">\n");
292     printf ("<button class=\"tablinks\" onclick=\"openCity(event, 'Categorias')\">Categorias</
293     printf ("<button class=\"tablinks\" onclick=\"openCity(event, 'Autores')\">Autores</buttor
294     printf ("<button class=\"tablinks\" onclick=\"openCity(event, 'Grafo')\">Grafo</button>\n<
295 }
296
297 void printHTMLend () {
298     printf ("<script>\nfunction openCity(evt, cityName) {\n");
299     printf ("var i, tabcontent, tablinks;\n");
300     printf ("tabcontent = document.getElementsByClassName(\"tabcontent\");\n");
301     printf ("for (i = 0; i < tabcontent.length; i++) {
302     \ntabcontent[i].style.display = \"none\";\n}");
303     printf ("tablinks = document.getElementsByClassName(\"tablinks\");\n");
304     printf ("for (i = 0; i < tablinks.length; i++) {\n");
305     printf ("tablinks[i].className = tablinks[i].className.replace(\"
306     active\", \"\");\n}");
307     printf ("document.getElementById(cityName).style.display = \"block\";\n");
308     printf ("evt.currentTarget.className += \" active\";\n}\n</script>\n");
309     printf ("</body>\n</html>");
310 }
```

Apêndice B

Ficheiro de exemplo com formato BibTeX

```
@techreport{chave1,
 author={Cunha, Gilberto and Tomás Carneiro},
 title = {Estruturas de Dados},
 institution = umdi,
 type = "Texto didactico",
 year = 1990,
 keyword = "algoritmos",
}

@InProceedings{chave2,
  author = "P.R. Henriques and Moura, Pedro",
  title = "Como lecionar PLC",
  booktitle = "SGML/XML'97 Conference",
  year =  1997,
  address = "Washington D.C. - USA",
  month =  "Dec.",
  keyword = "PDavid, SGML, Semantics",
}

@inproceedings{chave3,
 author={Gilberto Cunha and Carneiro, Tomás and Pedro Rangel Henriques and Moura, Pedro},
 title={Projeto de PLC},
 booktitle = "Museums and the Web 1998",
 note = "Toronto - Canada",
 year= 1998,
}
```