Processamento de Linguagens - TP1a

Carlos Pereira (A61887)
João Barreira (A73831)
Rafael Costa (A61799)

Março 2017

1 Introdução

O presente trabalho consiste no desenvolvimento de filtros de texto recorrendo à ferramenta *GAWK*. Os diferentes filtros devem ser produzidos com o recurso a *Expressões Regulares* para detetar *padrões de frases*. Para aumentar estas capacidades, foram propostos quatro exercícios distintos. Neste trabalho, optou-se por resolver todos estes exercícios.

Ao longo deste relatório explicaremos, com detalhe, a resolução de cada exercício proposto. Para cada um é dada ênfase às *Expressões Regulares* e *Acções Semânticas*, bem como eventuais estruturas e variáveis auxiliares utilizadas. Cada secção é também acompanhada pelo código completo do filtro de texto e de um ou mais exemplos da sua execução.

2 Processador de transações da Via Verde

O ficheiro *viaverde.xml* contém o extrato mensal de um cliente do serviço da Via Verde, fazendo referência à data e aos locais de entrada e saída (e.g. estrada ou parque), ao valor pago e à matrícula do veículo, entre outros.

De acordo com estas informações, foi desenvolvido um processado de texto com o auxílio do *GAWK* para ler o ficheiro *XML* e extrair as meta-informações pedidas.

2.1 Número total de 'entradas' em cada dia do mês

Foi feito um processador que lê o ficheiro *XML* e apresenta o número total de entradas para cada um dos dias em cada mês.

2.1.1 Expressões Regulares e Ações Semânticas

- "[>-]" - Esta expressão é usada como **field separator**. Desta maneira, torna-se bastante simples extrair do ficheiro *XML* atributos de cada *tag*,

bem como separar os campos de cada uma das datas (mês e dia, nest caso).

- **/<data_entrada>[0-9]/** - Identifica todas as linhas com informação relativa à data de entrada. Através da utilização do *field separator* em questão, é-nos agora possível aceder ao dia e ao mês das datas de entrada (que estão nos campos 2 e 3, respetivamente).

2.1.2 Estrutura de Dados Globais

Neste exercício apenas utilizámos um array bidimensional para guardar o mês e o dia de cada uma das entradas registadas no ficheiro *XML*, bem como duas variáveis auxiliares *auxDay* e *auxMonth* que servem apenas para fazer com que o valor retirado dos campos 2 e 3 seja um inteiro (e não uma *string*).

2.1.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   FS = "[>-]";
}
/<data_entrada>[0-9]/ {
   auxDay = 0 + $2;
   auxMonth = 0 + $3;
   count[auxMonth][auxDay]++;
}
END {
   for (i = 1; i <= 12; i++) {
      if (i in count) {
         print "M s: " i;

         for (j = 1; j <= 31; j++) {
            if (j in count[i]) {
               print "\tDia: " j " - " count[i][j] " entradas.";
            }
         }
      }
   }
}
```

2.2 Lista dos locais de 'saída'

Fez-se um processador que lê o ficheiro *XML* e apresenta a lista dos locais de saída sem repetições e por ordem alfabética.

2.2.1 Expressões Regulares e Ações Semânticas

- **"[<>]"** - Esta expressão é usada como **field separator**. Desta maneira, torna-se bastante simples extrair do ficheiro *XML* atributos de cada *tag*.
- **/<saida>/ !($3 in list)** - Identifica todas as linhas com informação relativa à 'saída' e cujo local de saída (dado pelo campo 3) ainda não esteja no array 'list', que, no fim da execução, conterá o conjunto com os nomes de todos os locais de saída.

2.2.2 Estrutura de Dados Globais

Para este exercício apenas utilizámos um *array* que contém o nome de todos os locais de saída (ordenado alfabeticamente e sem repetições) presentes no ficheiro *XML*.

2.2.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   FS = "[><]";
}
/<saida>/ && !($3 in list) {
   list[$3] = $3;
}
END {
   print "Lista de saidas:"

   for (e in list) {
      print "\t" e;
   }
}
```

2.3 Total gasto no mês (total e apenas em 'parques')

Para responder às últimas duas alíneas deste exercício, foi desenvolvido um processador que lê o ficheiro *XML* e apresenta os valores mensais do total gasto e o total gasto apenas em parques.

2.3.1 Expressões Regulares e Ações Semânticas

- **"[<>]"** - Esta expressão é usada como **field separator**. Desta maneira, torna-se bastante simples extrair do ficheiro *XML* atributos de cada *tag*.
- **/<importancia>/** - Identifica todas as linhas com informação relativa ao valor pago (i.e. importância).

- **/<valor_desconto>/** - Identifica todas as linhas com informação relativa ao desconto. O valor final será calculado subtraindo à importância o valor do desconto.
- **/<tipo>parque/** - Identifica todas as linhas com informação relativa ao 'tipo', selecionando apenas aquelas que são do tipo 'parque'.

2.3.2 Estruturas de Dados Globais

Foram utilizadas as variáveis *imp*, *desc*, *total* e *totalP* para guardar, respetivamente, a importância (custo), o valor do desconto, o gasto total e o gasto em 'parques'.

2.3.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   FS = "[><]";
}
/<importancia>/ {
   sub(",", ".", $3);
   imp = $3;
}
/<valor_desconto>/ {
   sub(",", ".", $3);
   desc = $3;
   total += imp - desc;
}
/<tipo>parque/ {
   totalP += imp - desc;
}
END {
   print "Total gasto: " total;
   print "Total gasto em parques: " totalP;
}
```

3 Album fotográfico em HTML

O principal objetivo deste exercício consiste em extrair toda a meta-informação de um ficheiro, em formato *XML*, e gerar uma página *HTML* com essa meta-informação. Esse ficheiro, chamado *lengenda.xml*, possui informação acerca de um conjunto de ficheiros de fotografias e, para cada uma destas, o nome das pessoas envolvidas e o local onde foram tiradas. A página *HTML* deve apresentar todas a fotos listadas no ficheiro *XML*, bem como as pessoas envolvidas em cada uma. Para além disso, deve ser gerada a lista de todos

os locais envolvidos, sem repetições. De referir que os ficheiros das fotografias encontram-se em `http://npmp.epl.di.uminho.pt/images/`.

Optou-se por gerar a página *HTML* de duas maneiras distintas: através do comando *IMG* e através de uma ancora como hiper-link que associe o nome do fotografado ao ficheiro. Apresentam-se, nas secções seguintes, os detalhes de cada uma destas implementações.

3.1 Página *HTML* com comandos *IMG*

Esta implementação gera uma página *HTML* com o seguinte formato:

```
<LI><b>NomeDaPessoa</b></LI>
    <center><img src="NomeDoFicheiro.jpg"/></center>
```

3.1.1 Expressões Regulares e Acções Semânticas

- **"[<>/"]"** - Esta expressão é usada como *FIELD SEPARATOR*. Desta maneira, torna-se bastante simples extrair do ficheiro *XML* atributos de cada *tag*, bem como nomes de ficheiros entre aspas.
- **/<foto /** - Identifica todas as linhas com informação relativa a uma foto. Como se pode verificar, na definição do *FIELD SEPARATOR*, o nome do ficheiro de uma foto é armazenado no campo número três. Basta armazenar-se o valor desse campo numa variável auxiliar.
- **/<quem>/** - A identificação dos sujeitos envolvidos numa foto é armazenada no campo número três. Usa-se uma função auxiliar para limpar todos os espaços em branco tanto no início como fim da descrição. No final, é acrescentada a informação de uma imagem em formato *HTML* a um ficheiro chamado *index.html*.
- **/<onde>/ !($3 = removeSpaces($3)) in locals)** - Filtra todas as linhas que possuam informações do local onde foi tirada uma foto. Uma linha é selecionada se contiver informação de um local que ainda não tenha sido armazenado. Caso estas condições se verifiquem, o local (presente no campo número três) é guardado num *array*.

3.1.2 Estruturas de Dados Globais

As estruturas globais envolvidas neste exercício são as seguintes: um *array* *locals* para guardar todos os locais (sem repetições) onde foram tiradas as fotos e quatro variáveis globais. Três das variáveis (*fmtLI*, *fmtI* e *end*) possuem valores constantes relativos a código *HTML*. A quarta variável (*image*) serve para guardar o ficheiro da última foto lida.

3.1.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   fmtLI = "<li><b>%s</b></li>\n";
   fmtI = "<center><img src=
           \"http://npmp.epl.di.uminho.pt/images/%s\"/>
           </center>\n"
   FS = "[<>\"]";
   end = "</body></html>";
   print "<html><head><meta charset='UTF-8'/>
          </head><body>" > "index.html";
}

/<foto / {
   file = $3;
}

/<quem>/ {
   $3 = removeSpaces($3);

   printf(fmtLI, $3) > "index.html";
   printf(fmtI, file) > "index.html";
}

/<onde>/ && !(($3 = removeSpaces($3)) in locals) {
   locals[$3] = 0;
}

END {
   print "<p></p>\n<b>Locais:</b>\n" > "index.html";

   for (i in locals) {
      printf("<li>%s</li>\n", i) > "index.html";
   }

   print end > "index.html";
}

function removeSpaces(str) {
   sub("^ +", "", str);
   sub(" +$", "", str);

   return str;
}
```

3.2 Página *HTML* com âncoras

Esta implementação é muito semelhante à implementação com o comando *IMG* mas, acrescenta uma nova funcionalidade. O conjunto de pessoas envolvidas no album é apresentado sem repetições. Cada pessoa tem a si associada uma página *HTML* que dispõe todos as fotos em que está presente. De seguida faremos referência apenas às expressões regulares e acções semânticas que divirgem da implementação com *IMG*.

3.2.1 Expressões Regulares e Acções Semânticas

A única expressão regular que é distinta das expressões regulares da outra implementação é a seguinte: **/<quem>/ (length($3) <200)**. Esta expressão filtra todas as linhas com a *tag quem* em que o nome da pessoa envolvida não ultrapasse os duzentos carateres (de modo a evitar erros ao guardar ficheiros com um nome demasiado extenso).

3.2.2 Estruturas de Dados Globais

Tal como a implementação com *IMG* são utilizadas um conjunto de variáveis constantes que contêm código *HTML*, uma variável para guardar o nome do ficheiro de uma foto e um *array* de locais. Para além destas estruturas é utilzado um *array* de duas dimensões *album[quem][images]*, que associa a uma pessoa um conjunto de fotos onde está presente.

3.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   enc = "<html> <head> <meta charset='UTF-8'/>
          </head> <body>"
   fmtHREF = "<p><a href=\"%s.html\"> %s </a></p>\n";
   fmtIMG = "<li><center><img
             src=\"http://npmp.epl.di.uminho.pt/images/%s\"/>
             </center></li>\n"
   FS = "[<>\"]";
   end = "</body></html>";
   print enc > "index.html";
}


/<foto / {
   image = $3;
}

/<quem>/ && (length($3) < 200) {
```

```
   $3 = removeSpaces($3);
   array[$3][image] = 0;
}

/<onde>/ && !(($3 = removeSpaces($3)) in locals) {
   locals[$3] = 0;
}

END {
   for (i in array) {
      aux = removeInvalidChars(i);

      printf(fmtHREF, aux, i) > "index.html";

      print enc > aux".html";

      for (j in array[i]) {
         printf(fmtIMG, j) > aux".html";
      }

      print end > aux".html";
   }

   print "<p></p>\n<b>Locais:</b>\n" > "index.html";

   for (x in locals) {
      printf("<li>%s</li>\n", x) > "index.html";
   }

   print end > "index.html";
}

function removeSpaces(str) {
   sub("^ +", "", str);
   sub(" +$", "", str);

   return str;
}


function removeInvalidChars(str) {
   sub("[:\\t]", "", str);
   return str;
}
```

4 Autores musicais

Existe uma diretoria com vários ficheiros de extensão '*.lyr*', que contêm a letra de canções precedida de 2 ou mais campos de meta-informação (1 por linha). Esta informção pode ser: título da canção, autor da letra, cantor, etc. A letra da música e a meta-informação estão separadas por uma linha em branco.

Foi feito um Processador de Texto com o auxílio do *GAWK* para ler todos os ficheiros '*.lyr*' e obter informações sobre os cantores, autores e títulos de canções.

4.1 Total de *cantores* e a lista com os nomes

Aqui foi feito um processador que recebe ficheiros de letra de música como *input* e escreve o total e os nomes de todos os cantores.

4.1.1 Expressões Regulares

Após a análise de alguns possíveis ficheiros de entrada, verificou-se que as linhas de informações relativas aos nomes dos cantores começam por *"singer:"*, onde os nomes estão separados por espaços em branco, dois pontos, vírgulas ou por pontos e vírgulas, daí os termos utilizado como *Field Separator*.

A informação que queremos retirar está em linhas que começam por *"singer:"*.

4.1.2 Ações Semânticas

Sempre que o processador encontra um registo iniciado por *"singer:"*, percorrem-se todos os campos desse registo. Para cada um destes campos, retiram-se os carateres desnecessários. Depois, caso o registo não exista no *array* dos nomes dos cantores, é adicionado a esse *array* e o contador de todos os cantores é incrementado.

No fim, antes de se imprimirem os resultados, o *array* com os nomes dos cantores é ordenado.

4.1.3 Estruturas de Dados Globais

Foram utilizadas duas estruturas de dados globais: um *array singers*, em que o índice são os nomes de todos os cantores e uma variável *total* que é utilizada como contador de cantores.

4.1.4 Filtro de Texto

```
BEGIN {
   FS = " *[:;,] *"
}

/singer:/ {
```

```
    for (i = 2; i <= NF; i++) {
        sub("[ ?()]+$", "", $i);

        if (!($i in singers) && ($i != "")) {
          count++;
          singers[$i] = $i;
        }
    }
}

END {
    n = asort(singers);

    for (i = 1; i <= n; i++) {
       print singers[i];
    }

    print "Total: " count;
}
```

4.2 Todas as canções do mesmo *autor*

Fez-se um processador que recebe ficheiros de letra de música como *input* e calcula o número de canções de cada autor.

4.2.1 Expressões Regulares

Verificou-se que as linhas de informações relativas aos nomes dos autores começam por *"author:"*. Tal como no caso dos cantores, os nomes estão separados por espaços em branco, dois pontos, vírgulas ou por pontos e vírgulas, daí os termos utilizado como *Field Separator*.

As informações que pretendemos são retiradas de registos que começam por *"author:"*.

4.2.2 Ações Semânticas

Ao encontrar um registo iniciado por *"author:"*, o processador percorre todos os campos desse registo. Para cada um destes campos, retiram-se os carateres desnecessários. Depois, caso o registo não seja vazio, ou seja, conhecem-se os autores da música, a posição correspondente ao nome do autor no *array* de autores é incrementado. Caso contrário, incrementa-se a posição correspondente ao "Autor Desconhecido".

No fim, imprimem-se os resultados da seguinte forma: nome do autor seguido do número de músicas associadas a ele.

4.2.3 Estruturas de Dados Globais

Temos como estrutura global um *array authors*, em que o índice é o nome de um autor e o valor correspondente ao número de músicas associado a esse autor.

4.2.4 Filtro de Texto

```
BEGIN {
   FS = " *[:;,] *"
}

/author:/ {
   for (i = 2; i <= NF; i++) {
      sub("[ ?()\\t]+$", "", $i);

      if ($i != "") {
         songs[$i]++;
      }
      else {
         songs["Autor desconhecido"]++;
      }

   }
}

END {
   for (i in songs) {
      print i " - " songs[i];
   }
}
```

4.3 Escrever o nome de cada *autor* seguido do título das suas canções

O processador recebe ficheiros de letra de música como *input* e escreve, para cada autor, o título das suas canções.

4.3.1 Expressões Regulares

Aqui, as linhas de informação relativas ao nome de uma canção começam por *"title:"* e as dos nomes dos autores começam por *"author:"*.

Da mesma maneira que se fez nos exercícios anteriores, os nomes estão separados por espaços em branco, dois pontos, vírgulas ou por pontos e vírgulas, daí os termos utilizado como *Field Separator*.

As informações que pretendemos são retiradas de registos que começam por *"title:"* e por *"author:"*.

4.3.2 Ações Semânticas

O processador, a cada registo que comece por *"title:"*, atribui esse campo à variável global *song*. Aqui fica guardado o título da canção. Depois retira os carateres que são considerados desnecessários.

Aos registos que sejam iniciados em *author:*, começa a percorrê-lo a partir do seu segundo campos (visto que ser *author:*). Cada um destes campos é o nome de um autor. Depois, a cada um destes que não seja vazio, retiram-se os carateres desnecessários e é adicionado à variável global *songs*, na linha correspondente ao nome do autor, o conteúdo na variável *song* que se traduz no título da canção.

Caso não se tenha o nome do cantor, ou seja, o campo é vazio, adiciona-se o título da canção à linha do "Autor Desconhecido".

No fim, para cada uma das linhas da matriz *songs*, imprime-se para cada linha, o seu índice e os conteúdos de cada uma das suas colunas.

4.3.3 Estruturas de Dados Globais

Utilizaram-se 2 variáveis globais: a variável *song* que guarda o título de uma canção e uma matriz *songs*. Nesta matriz, as linhas correspondem a nome de autores e as respetivas colunas correspondem aos títulos das canções associadas ao autor dessa linha.

4.3.4 Filtro de Texto

```
BEGIN {
   FS = " *[:;,] *"
}

/title: / {
   song = $2;
   sub("\\(\\?\\)", "", song);
   sub("^[ *=]+", "", song);
}

/author:/ {
   for (i = 2; i <= NF; i++) {
      sub("[ ?()\\t]+$", "", $i);

      if ($i != "") {
         authors[$i][song] = song;
      }
      else {
```

```
            authors["Autor desconhecido"][song] = song;
        }

    }
}

END {
    for (a in authors) {
        printf("%s: ", a);

        flag = 0;

        for (j in authors[a]) {
            if (flag == 0) {
                printf("%s", j);
            }
            else {
                printf(", %s", j);
            }

            flag++;
        }

        printf("\n");
    }
}
```

5 Dicionauro

Uma diretoria designada por *Dicionauro* contém um conjunto de ficheiros com a extensão '*.txt*' com inúmeras entradas em Português de termos de um *Thesaurus*. Cada termo é iniciado pela sigla '*PT*' e tem a si associado uma ou mais categorias. Um termo pode possuir um conjunto de definições. Uma definição é uma linha iniciada pela sigla '*Def*'.

Descrevem-se, de seguida, os exercícios propostos que têm como base a diretoria *Dicionauro*.

5.1 Criação de uma página *HTML* com todos os termos e suas respetivas categorias e definições

A estrutura da página *HTML* criada neste exercício é a seguinte:

```
<p></p><b>Termo</b>
<p></p><b>Categorias:</b>
<li>Categoria 1</li>
```

```
<li>Categoria 2</li>
            .
            .
            .

<p></p><b>Definicoes:</b>
<li>Definicao 1</li>
<li>Definicao 2</li>
            .
            .
            .
```

Ou seja, na leitura de todos os ficheiros presentes na diretoria *Dicionauro* não se devem apresentar termos repetidos, mas sim acrescentar a um termo que já exista novas categorias e defenções.

5.1.1 Expressões Regulares e Acções Semânticas

- **"[>!<]*pt(_br)?(_pt)?"** - Expressão regular atribuída ao *FIELD SEPARATOR*. Ao inspecionarmos vários ficheiros da diretoria em questão, verificamos que nem todos seguem o mesmo padrão. A maior parte dos ficheiros apresenta as linhas respetivas a um termo iniciadas pela sigla '*PT*'. No entanto, verificamos que um pequeno número de ficheiros possuía a sigla '*PT*' precedida ou sucedida pelo seguinte conjunto de carateres: *<, >, ! e .* Para além disso, uma pequena minoria apresentava distinções para termos em portugês e em brasileiro, ou seja, '*PT_PT*' e '*PT_BR*'. A expressão regular descrita permite tratar cada um destes diferentes tipos de termos.

- **/^pt/ || /^[><!]*pt/** - Filtra todos as linhas correspondentes a um termo. Tal como foi, falado acima, existem vários formatos para uma linha com um termo. Todos os espaços em branco (tanto no início como no final) de um termo são extraídos, bem como todos os *tabs* que possua. Depois dessa extração, o termo é guardado numa variável auxiliar.

- **/[<>!?]*def/** - Esta expressão permite selecionar todas as linhas que contenham uma definição. Novamente a sintaxe de uma linha que contém uma definição pode ser constituída por um conjunto diferente de carateres. Como o *FIELD SEPARATOR* foi definido de modo a selecionar todos os termos de um ficheiro, é necessário recorrer-se ao uso da função *split* para se extrair a definição propriamente dita. Basta usar-se esta função para o campo número zero (todo o registo) e fazer-se o *split* pela sequência de carateres "*def*".

- **/[<>!?-]*catgra/** - Seleciona todas as linhas que contêm uma categoria. Tal como para uma definição, é realizado um *split* no campo número zero, mas desta vez pela sequência de carateres "*catgra*".

5.1.2 Estruturas de Dados Globais

Este exercício recorre ao uso de cinco variáveis auxiliares, de valor constante, para armazenarem meramente código *HTML*. São utilizadas mais três variáveis (*entry*, *def* e *catg*) para guardarem, respetivamente, os valores de um termo, definição e uma categoria. A estrutura mais relevante para este exercício consiste num *array* de três dimensões *entries[entry][catg][def]*. Cada termo deste *array* tem a si associado um conjunto de categorias. Por sua vez, cada categoria tem a si associada um conjunto de definições. Esta estruturação é útil em casos em que uma nova categoria é definida para um mesmo termo noutro ficheiro. De modo a facilitar a compreensão desta estrutura ilustraremos o seguinte exemplo: suponha-se que existem três ficheiros distintos ("A", "B"e "C"). Ambos os ficheiros possuem o mesmo termo ("*corpo*") e um conjunto de definições distintas para este. Suponha-se, também, que no ficheiro "A"a categoria do termo é "*nl*"(tal como no ficheiro "B") e que no ficheiro "C"não há nenhuma informação relativa à categoria do termo. Sendo assim o *array* possui as seguintes configurações: *entries["corpo"]["nl"]["conjunto de definições do ficheiro "A"e "B]* e *entries["corpo"][""]["conjunto de definições do ficheiro "C"]*.

5.1.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   fmtLI = "<li styl, =\"margin-left:50px\">%s</li>\n";
   fmtE = "<p></p><b style=\"font-size:150%\">%s</b>\n";
   fmtCD = "<p></p><b style=\"margin-left:25px\">%s</b>\n";
   FS = "^[> !<]*pt(_br)?(_pt)?";
   end = "</body></html>";
   print "<html><head><meta charset='UTF-8'/>
          </head><body>" > "index.html";
}


/^pt/ || /^[>< !]*pt/ {
   entry = removeWhiteSpaces($2);
}

/[> <!?]*def/ {
   split($0, target, "def");
   def = removeWhiteSpaces(target[2]);
   entries[entry][catg][def] = def;
}

/[> <!?-]*catgra/ {
   split($0, target, "catgra");
   catg = removeWhiteSpaces(target[2]);
```

```
}

END {
   for (i in entries) {
      if (length(i) > 1) {
      printf(fmtE, i) > "index.html";

      printf(fmtCD, "Categorias:") > "index.html";

      for (c in entries[i]) {
         if (length(c) > 1) {
            printf(fmtLI, c) > "index.html";
         }
      }

      printf(fmtCD, "D e f i n i   e s :") > "index.html";

      for (j in entries[i]) {
         for (w in entries[i][j]) {
               if (length(w) > 1) {
                  printf(fmtLI, w) > "index.html";
               }
            }
         }
      }
   }

   print end > "index.html";
}

function removeWhiteSpaces(str) {
   sub("\\t", "", str);
   sub("^ +", "", str);
   sub(" +$", "", str);

   return str;
}
```

5.2 Listagem de todos os diferentes domínios e respetivo número de entradas

5.2.1 Expressões Regulares e Acções Semânticas

5.2.2 Estruturas de Dados Globais

5.2.3 Filtro de Texto

```
BEGIN {
   IGNORECASE = 1;
   FS = "^[&> !<]*dom +";
}

/^[&>< !]*dom / && length($2) > 0 {
   dom[norm($2)]++;
}

END {
   for (i in dom) {
      print i ": " dom[i];
   }
}

function norm(str) {
   sub("^ +", "", str);
   sub(" +$", "", str);
   return tolower(str);
}
```

6 Conclusão

Com a realização deste trabalho, pudemos pôr em prática os conceitos adquiridos nas aulas teórico-práticas sobre a ferramenta *GAWK*.

Percebemos a importância do uso das expressões regulares e ações semânticas, de forma a podermos pegar num texto de um qualquer tipo, obter as informações que necessitamos e, com elas, chegar a um conjunto de conclusões.

Tomámos consciência de que esta tarefa nem sempre é fácil (devido à complexidade dos textos em questão), o que faz com que a definição das expressões regulares para conseguirmos obter as informações que necessitamos nem sempre seja uma tarefa trivial. Um bom exemplo deste tipo de dificuldade, foi a nossa tentativa de separar as pessoas no ficheiro *xml* do exercício 2 relativo ao álbum de fotografias.

Assim sendo, em retrospetiva, achamos que a realização deste primeiro trabalho prático foi enriquecedora e nos deu uma boa base relativamente ao processamento de ficheiros textuais.