Processamento de Linguagens - MIEI (3º ano)

Trabalho Prático nº1 – Parte A (GAWK)

Relatório de Desenvolvimento

Ana Isabel Castro (a55522)

Joana Miguel (a57127)

Lúcia Abreu (a71634)

15 de Março de 2017

Conteúdo

Capítulo 1

Introdução

Este primeiro trabalho da unidade curricular de Processamento de Linguagens do 3º ano do Mestrado Integrado em Engenharia Informática consiste no desenvolvimento e implementação de Processadores de Linguagens Regulares, que filtrem ou transformem textos, utilizando para isso o sistema de produção para filtragem de texto `GAWK`.

Os principais objetivos com este trabalho prático são:

- aumentar a experiência de uso do ambiente `Linux` e de algumas ferramentas de apoio à programação;
- aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases;
- desenvolver, a partir de Expressões Regulares (ERs), sistemática e automaticamente Processadores de Linguagens Regulares, que filtrem ou transformem texto.

Para tal, foram abordados dois problemas/enunciados:

- `Processador de transações da Via Verde`
- `Álbum Fotográfico em HTML`

Cujo objetivo é desenvolver um processador de texto que processe a informação contida nos ficheiros `XML` e:

- forneça informação e estatísticas sobre as transações realizadas na Via Verde;
- gerar uma página HTML com o albúm de fotografias dispondo-as de forma cronológica e listando todos os locais que foram fotografados.

Estrutura do Relatório

O Capítulo 2 refere-se ao **Processador de transações da Via Verde**, um dos enunciados selecionados para resolução. Este capítulo está dividido 2 sub-capítulos:

- A **Descrição do Problema** no qual se faz uma breve descrição da estrutura do ficheiro `XML` a extrair informação usando programas em `AWK`;
- A **Implementação** que corresponde ao Desenho da solução e Resultados e testes obtidos para cada uma das alíneas resolvidas.

O capítulo 3 corresponde ao segundo enunciado resolvido, **Àlbum Fotográfico em HTML**, que está dividido em 2 subcapítulos:

- A **Descrição do Problema** e
- A **Implementação**.

O capítulo 4 refere-se à **Conclusão** na qual se sumariza brevemente o que se concretizou com este trabalho.

Em **Apêndice** segue o código implementado para os dois enunciados resolvidos.

Capítulo 2

Processador de transações da Via Verde

2.1 Descrição do problema

A Via Verde envia a cada um dos seus utentes um extracto mensal no formato `XML`, como se pode obervar no ficheiro `viaverde.xml` (Apêndice B?)

Este ficheiro fornece informação sobre cada transação efetuada:

- qual a entrada e a saída (e respetivas datas e horas);
- a importância cobrada, o valor do desconto (se aplicado) e a taxa do IVA;
- o operador e o tipo (portagem ou parque de estacionamento);
- a data de débito e o número do cartão.

Tal pode ser observado neste pequeno "pedaço de código"retirado do `viaverde.xml`, e que corresponde a apenas uma transação.

```
<TRANSACCAO>
<DATA_ENTRADA>26-07-2015</DATA_ENTRADA>
<HORA_ENTRADA>11:33</HORA_ENTRADA>
<ENTRADA>Povoa N-S</ENTRADA>
<DATA_SAIDA>26-07-2015</DATA_SAIDA>
<HORA_SAIDA>11:42</HORA_SAIDA>
<SAIDA>Angeiras N-S</SAIDA>
<IMPORTANCIA>2,00</IMPORTANCIA>
<VALOR_DESCONTO>0,00</VALOR_DESCONTO>
<TAXA_IVA>23</TAXA_IVA>
<OPERADOR>I. de Portugal (N1)</OPERADOR>
<TIPO>Portagens</TIPO>
<DATA_DEBITO>05-08-2015</DATA_DEBITO>
<CARTAO>6749036</CARTAO>
</TRANSACCAO>
```

O restante ficheiro `XML` possui informação sobre:

- O **extracto**: (`version, encoding, id e mes_emissao`);
- O **cliente**: (`nome, morada, codigo postal`);
- Os **identificadores**: (`id, matricula, ref_pagamento`);
- O **valor**: (`total e total_iva`) geral e por identificador.

E é sobre estes dados que iremos processar este ficheiro, analisando-o com cuidado, e desenvolvendo programas em `AWK` que:

- Calcule o número de entradas em cada dia do mês
- Escreva a lista de locais de saída
- Calcule o total gasto no mês
- Calcule o total gasto no mês apenas em parques
- Escreva a lista das datas em que foram efetuados débitos
- Calcule a média dos pagamentos de cada mês
- Escreva a lista de operadores e respetivo número de ocorrências
- Calcule o período de tempo passado em parques de estacionamento
- Calcule o desconto total obtido no mês

Como ficheiros utilizados para a extração e processamento da informação utilizou-se para além do ficheiro `viaverde.xml` disponibilizado, o ficheiro `viaverde_identificadores.xml` constituído pelo conteúdo do ficheiro disponibilizado com a adição de transações de outro identificador de veículo.

2.2 Implementação

De modo a resolver as alíneas que se seguem sobre o Enunciado da Via Verde teve que se definir o *Register Separater*(RS) e o *Field Separator*(FS), de modo a extrair do ficheiro a informação pretendida.

O RS (separador de registo) por defeito é o *newline* (`\n`), algo que não se redefiniu visto que a informação relevante e distinta já se encontrava separada por linha. Logo, um registo é representado por um linha do ficheiro.

O FS (separador de campo), que divide um registo em campos, por defeito é o espaço. Redefiniu-se para `<` ou para `>`, visto que se trata de um ficheiro XML, onde temos tags que começam e acabam com `<` e `>`, respetivamente.

Para além disto a informação relevante a extrair se encontra entre tags, por exemplo

```
<IMPORTANCIA> 2,5 </IMPORTANCIA>
```

e desta forma ter-se-ia que ir ler o valor do campo 3, `$3`, para obter o valor da importância.

Todos os programas implementados `AWK` que irão ser abordados de seguida encontram-se em `Apêndice A.1`.

2.2.1 Alínea a) - Calcular o número de entradas em cada dia do mês

Desenho da solução

Para o cálculo do número de entradas em cada dia do mês, optou-se por calcular o número de entradas geral, de todos os identificadores da conta Via Verde. Apesar de um cliente poder ter várias matrículas associadas à sua conta, julgou-se não ser importante distinguir o número de entradas por identificador.

Para a obtenção do número de entradas em cada dia do mês, foi necessário utilizar uma expressão regular `/<DATA_ENTRADA>/` de forma a fazer *match* com a *tag* `<DATA_ENTRADA>` do ficheiro lido. Visto algumas datas de entrada serem `null`, foi necessário assegurar que o valor do campo, `$3`, apenas é guardado quando diferente de `null`.

Para tal, utilizou-se um *array* `entradasDia`, cujos índices são *strings* que correspondem às datas de entrada. Para cada *match*, incrementa-se o valor da posição do *array* `entradasDia` que corresponde à data de entrada. Data essa que corresponde ao valor do campo, `$3`.

No final, é executado o bloco `END` no qual se imprime para cada data de `entradasDia` o número de ocorrências.

Testes e resultados

O filtro `viaverde_a.awk` foi testado com o ficheiro `viaverde.txt` e obteve-se os seguintes resultados:

```
Data                Nr Ocorrências
30-07-2015 :                3
18-08-2015 :                2
29-07-2015 :                2
11-08-2015 :                2
10-08-2015 :                7
26-07-2015 :                3
17-08-2015 :                4
06-08-2015 :                4
13-08-2015 :                5
21-08-2015 :                4
31-07-2015 :                1
```

2.2.2 Alínea b) - Escrever a lista de locais de saída

Desenho da solução

Para escrever a lista dos locais de saída, foi necessário utilizar uma expressão regular `/<SAIDA>/` de forma a fazer *match* com a *tag* `<SAIDA>` do ficheiro `viaverde.txt`.

Para cada um dos *match*, utilizou-se um *array* `saidas`, cujos índices são *strings* que correspondem ao local de saída, de forma a guardar essa informação. Os locais de saída correspondem ao valor do campo, `$3`.

Como em `AWK` é possível que os *arrays* possam ter *strings* como índices, no final, no bloco `END`, imprime-se cada índice do *array* `saidas`.

Testes e resultados

O filtro `viaverde_b.awk` foi testado com o ficheiro `viaverde.txt` e obteve-se os seguintes resultados:

```
Locais de saída:

        Braga Sul
        Maia PV
        Neiva S-N
        Angeiras N-S
        Freixieiro
        Valongo
        PQ A Sa Carn.I
        Ermesinde PV
```

```
Aeroporto
Maia II
EN107
EN 205 PV
Neiva N-S
Lipor
Ponte Pedra
PQ Av. Central
Custoias
Povoa S-N
Ferreiros
```

2.2.3 Alínea c) - Calcular o total gasto no mês

Para calcular o total gasto no mês, implementou-se dois programas em AWK.

- (i) Um lê o valor do total de cada identificador, e também o total de todos os identificadores;
- (ii) O outro simplesmente faz o incremento do valor da importância de cada transação por identificador de veículo, do cliente.

i) Desenho da solução

Para o cálculo do gasto no mês, optou-se por calcular o total gasto por identificador de veículo e o total por cliente, já que um cliente pode ter vários veículos associados à sua Via Verde.

Em vez de retornar como *output* o identificador do veículo (por exemplo: `id="28876820811"`), optou-se por imprimir a matrícula do carro, que o identifica também de forma única, contudo é mais legível e agradável à leitura humana. Para isso, utilizou-se a expressão regular `/<MATRICULA>/`, que quando faz *match* com uma linha do ficheiro lido, extrai o valor do campo `$3`, que corresponde à matrícula do veículo e guarda-o na variável `matricula`.

Para a obtenção do total gasto em cada identificador do veículo foi necessário utilizar uma expressão regular `/<TOTAL>/` de forma a fazer *match* com a *tag* `<TOTAL>` do ficheiro lido. Para cada *match*, executa-se as seguintes ações:

- ler o valor do campo `$3`, valor monetário que representa os gastos desse identificador ou o valor total gasto pelo cliente, dependendo se está dentro de uma *tag* `<IDENTIFICADOR>`.
- testar se a variável `id` é `null`:
  - sendo `null` é porque estamos na *tag* `<TOTAL>` correspondente ao total gasto pelo cliente, imprimindo como *output* o total mensal gasto pelo cliente.
  - senão é porque estamos na *tag* `<TOTAL>` correspondente a um identificador de um veículo do cliente, imprimindo como *output* o total mensal gasto pelo cliente com aquele veículo.

Para isto funcionar, teve-se que guardar o identificador do veículo na variável `id`, quando se encontrava no ficheiro a *tag* `<IDENTIFICADOR>`, e passá-la a `null` quando se encontrava a *tag* `</IDENTIFICADOR>` que delimita a informação correspondente a esse veículo.

i) Testes e resultados

O filtro `viaverde_c.awk` foi testado com o ficheiro `viaverde_identificadores.txt` e obteve-se os seguintes resultados:

```
Gasto Mensal (em euros)
00-LJ-11        77,40
45-NU-45        10,00
Total           87,40
```

ii) Desenho da solução

Alternativamente, decidiu-se calcular o gasto do mês por identificador e cliente, pela soma do valor das importâncias de cada transação, uma vez que as importâncias de cada transação já têm o IVA incluído e o desconto.

Para cada identificador de veículo leu-se a matrícula do veículo, e em cada transação desse veículo a importância gasta. Para isso, utilizou-se as seguintes expressões regulares `/<MATRICULA>/` e `/<IMPORTANCIA>/`, e executou-se uma acção que permitiu ler o valor do campo, `$3`, em ambas, de maneira a guardar o valor da matrícula na variável `matricula` e o valor da importância na variável `valor`.

Após ler e guardar esse valor na variável `valor`, executou-se o seguinte:

- split da variável `valor`, uma vez que para os valores monetários do ficheiro, a parte inteira é separada da parte decimal por uma vírgula e o `awk` não consegue realizar operações aritméticas com valores que tenham vírgulas. Teve-se que separar a variável valor com o separador vírgula, de modo a juntar os dois campos (parte inteira e parte decimal) por um ponto.
- incremento da variável `contaId` com o valor da variável `valor`.

Ao identificar-se no ficheiro a *tag* `</IDENTIFICADOR>` que corresponde ao fim da informação correspondente a um identificador de veículo, imprime-se para o `stdout` a variável `contaId` que corresponde ao total gasto para esse identificador, incrementa-se a variável `contaTotal`, que corresponde ao valor total gasto pelo cliente, com o valor da variável `contaId` e, por fim faz-se *reset* dessa variável de modo a ser reutilizada para o cálculo dos gastos de outro identificador de veículo.

No fim da leitura do ficheiro é executado o bloco `END` no qual se imprime para o `stdout` o valor total mensal gasto pelo cliente.

ii) Testes e resultados

O filtro `viaverde_c_2.awk` foi testado com o ficheiro `viaverde_identificadores.txt` e obteve-se os seguintes resultados:

```
Gasto Mensal (em euros)
00-LJ-11        77.4
45-NU-45        10
Total           87.4
```

2.2.4 Alínea d) - Calcular o total gasto no mês apenas em parques

Desenho da solução

Para cálculo do gasto mensal em parques decidiu-se apresentar os resultados, tal como na alínea anterior, por identificador de veículo (imprimindo a sua matrícula) e o total gasto pelo cliente.

Para cada identificador de veículo teve-se que ir buscar a matrícula do veículo, e em cada transação desse veículo a importância gasta. Para isso, utilizou-se as seguintes expressões regulares `/<MATRICULA>/` e `/<IMPORTANCIA>/`, e executou-se uma acção que permitiu ler o valor do campo, `$3`, em ambas, de maneira a guardar o valor da matrícula na variável `matricula` e o valor da importância na variável `valor`.

A expressão `/<TIPO>/ && $3 == "Parques de estacionamento"`, filtra todas as transações que forem sobre parques de estacionamento. Tratando-se de uma transação deste tipo é executa a seguinte ação:

- *split* da variável `valor`, e substituição da vírgula que separa a parte inteira do número da parte decimal por um ponto, de modo a poder realizar operações aritméticas com estes valores.
- incremento da variável `contaId` com o valor da variável `valor`.

Quando se encontra a *tag* `</IDENTIFICADOR>` que corresponde ao fim da informação correspondente a um identificador de veículo, imprime-se para o `stdout` a variável `contaId` que corresponde ao total gasto em parques para esse identificador, incrementa-se a variável `contaTotal`, que corresponde ao valor total gasto pelo cliente em parques, com o valor da variável `contaId` e, por fim faz-se reset dessa variável de modo a ser reutilizada para o cálculo dos gastos de outro identificador de veículo.

No fim da leitura do ficheiro é executado o bloco `END` no qual se imprime para o `stdout` o valor total mensal gasto pelo cliente em parques.

Testes e resultados

O filtro `viaverde_d.awk` foi testado com o ficheiro `viaverde_identificadores.txt` e obteve-se os seguintes resultados:

```
Gasto Mensal Parques(em euros)
00-LJ-11        6.35
45-NU-45        1.95
Total           8.3
```

2.2.5 Alínea e) - Escrever a lista das datas em que foram efetuados débitos

Desenho da solução

Para a listagem das datas que em que foram efetuados débitos, optou-se por listar as datas de débito de todos os identificadores da conta Via Verde.

Para a obtenção das datas de débito foi preciso o uso da expressão regular `/<DATA_DEBITO>/` de forma a fazer *match* com a *tag* `<DATA_DEBITO>` do ficheiro lido.

Fazendo *match*, a seguinte acção é executada: guarda-se o valor do campo `$3`, correspondente à data de débito, num *array* auxiliar `datadeb`. Os índices deste *array* correspondem às datas de débito lidas.

No final, é executado o bloco `END` no qual se percorre o *array* `datadeb` e imprime para `stdout` as datas de débito (índices do *array*) presentes no ficheiro.

Testes e resultados

O filtro `viaverde_e.awk` foi testado com o ficheiro `viaverde.txt` e obteve-se os seguintes resultados:

```
Datas de débito:
07-08-2015
14-08-2015
05-08-2015
31-08-2015
21-08-2015
```

2.2.6 Alínea f) - Calcular a média dos pagamentos de cada mês

Desenho da solução

Para o cálculo da média de pagamento decidiu-se calcular:

- média de pagamento em parques de estacionamento por identificador
- média de pagamento em portagens por identificador
- média de pagamento por identificador
- média de pagamento total, isto é, média de pagamento do cliente.

De maneira a determinar estes valores, utilizou-se as seguintes variáveis auxiliares, inicializando-as a zero:

- **valorIdPortagens** - valor monetário gasto por um dado identificador de veículo em portagens.
- **valorIdParques** - valor monetário gasto por um dado identificador de veículo em parques de estacionamento.
- **valorTotalIdParques** - valor monetário gasto pelo cliente em parques de estacionamento.
- **valorTotalIdPortagens** - valor monetário gasto pelo cliente em parques de estacionamento.
- **nrOcorrenciasIdPortagens** - número de transações do tipo Portagem por identificador
- **nrOcorrenciasIdParques** - número de transações do tipo Parque de estacionamento por identificador
- **nrOcorrenciasTotalIdPortagens** - número de transações do tipo Portagem do cliente
- **nrOcorrenciasTotalIdParques** - número de transações do tipo Parques de estacionamento do cliente
- **nrOcorrenciasTotal** - número de transações por cliente.

Para cada identificador de veículo leu-se a matrícula do veículo, assim, como em cada transação desse veículo a importância gasta. Para tal, usou-se as seguintes expressões regulares `/<MATRICULA>/` e `/<IMPORTANCIA>/`, e executou-se uma acção que permitiu ler o valor do campo, `$3`, em ambas, de maneira a guardar o valor da matrícula na variável `matricula` e o valor da importância na variável `importancia`.

Seguidamente é executada a seguinte expressão `/<TIPO>/ && $3 =="Portagens"`, que sendo verdadeira é despoletada a seguinte ação:

- split da variável `importancia`, de modo a substituir a vírgula que separa a parte inteira da parte decimal do número, para depois se poder efectuar operações aritméticas com este valor.
- incremento da variável `valorIdPortangens` com o valor da variável `importancia`.
- incrementado o valor da variável `nrOcorrenciasIdPortagens`.

Também tenta-se procurar pelas linhas do ficheiro que sejam do do tipo parque de estacionamento com a seguinte expressão:

```
/<TIPO>/ && $3 == "Parques de estacionamento"
```

Havendo um *match*, o procedimento é praticamente igual ao procedimento anterior para as portagens, somando agora o `valorIdParques` com o valor da variável `importancia`, e incrementado o valor da variável `nrOcorrenciasIdParques`.

A identificação da *tag* `</IDENTIFICADOR>` conduz a ação de imprimir para o `sdtout` a média de pagamento do identificador do veículo para portagens e parques de estacionamento. Sendo actualizadas as variáveis correspondentes ao valor total gasto em portagens, valor total gasto em parques, valor total gasto em ambos, número de ocorrências de transações do tipo portagens, número de de ocorrências de transações do tipo parques de estacionamento, e por fim, número de transações de portagens e parques.

As variáveis usadas por identificador, como por exemplo

`valorIdPortagens e nrOcorrenciasIdPortagens`

assim como as correspondentes para os parques, são novamente inicializadas a zero, para reutilização no cálculo da média de pagamentos para outro identificador.

Após a leitura do ficheiro é executada a ação do bloco `END` na qual se imprime para `stdout` o valor médio gasto em portagens, parques de estacionamento e o valor médio global.

Testes e resultados

O filtro `viaverde_f.awk` foi testado com o ficheiro `viaverde_identificadores.txt` e obteve-se os seguintes resultados:

```
Media de Pagamento
                    Portagens(euros/mes)          Parques(euros/mes)
00-LJ-11              1.82                          3.17
45-NU-45              2.68                          1.95
Media Parcial         1.88                          2.77

Media Total: 1.94 euros/mes
```

2.2.7 Alínea g) - Escrever a lista de operadores e respetivo número de ocorrências

Desenho da solução

Para o cálculo do número de ocorrências de cada operador, optou-se por calcular o número de ocorrências do cliente.

Na determinação do número ocorrências de cada operador, foi necessário utilizar a expressão regular `/<OPERADOR>/` de forma a fazer *match* com a *tag* `<OPERADOR>` do ficheiro lido.

Para tal, utilizou-se um *array* `operadores`, cujos índices são *strings* que correspondem aos operadores. A cada *match*, incrementa-se o valor da posição do *array* operadores que corresponde ao operador (valor do campo `$3`).

No final, é executado o bloco `END` no qual se imprime para cada operador o número de ocorrências.

Testes e resultados

O filtro `viaverde_g.awk` foi testado com o ficheiro `viaverde.txt` e obteve-se os seguintes resultados:

```
Operador                                    Nr Ocorrencias
ANA - Aeroportos de Portugal. SA (AP)             1
BRAGAPARQUES (BP)                                 1
I. de Portugal (E1)                               12
Brisa Concessão Rodoviária (BR)                   6
I. de Portugal (P3)                               9
I. de Portugal (N1)                               12
```

2.2.8 Alínea h) - Período de tempo passado em parques de estacionamento

Desenho da solução

Para a determinação do tempo que cada veículo passou em parques de estacionamento, bem como o tempo total passado pelos veículos do cliente, foram necessárias inicializar a zero duas variáveis `tempoMinutosId` (minutos passados em parques por veículo) e `tempoMinutosTotal` (minutos passados em parques pelos veículos do cliente).

Para cada identificador de veículo teve-se que ler a matrícula do veículo, assim, como em cada transação desse veículo, a hora de entrada e a hora de saída. De seguida, caso a expressão `/<TIPO>/ && $3 =="Parques de estacionamento"` faça *match*, a seguinte ação é executada:

- realiza-se um *split* sobre o valor da variável `horaentrada` e `horasaida` de forma a separar em dois campos as horas e minutos, de maneira a calcular os minutos de entrada e os minutos de saída.
- obtém-se a duração em minutos do tempo passado em parques, subtraindo aos minutos de saída os minutos de entrada.

Por sua vez, a identificação da *tag* `<\IDENTIFICADOR>` conduz ao calculo e à impressão para o `stdout` do tempo em horas e minutos passados em parques para um dado veículo, incrementando-se de seguida o valor da variável `tempoMinutosTotal` com o valor correspondente ao tempo em minutos passado por um veículo, e por fim faz-se *reset* à variável `tempoMinutosId`, para posterior reutilização.

Finalmente, no bloco `END` é calculado e enviado para `stdout` o tempo total gasto em parques pelo cliente.

Testes e resultados

O filtro `viaverde_h.awk` foi testado com o ficheiro `viaverde_identificadores.txt` e obteve-se os seguintes resultados:

```
Parques
00-LJ-11        2 horas e 51 minutos
45-NU-45        1 horas e 1 minutos
Total           3 horas e 52 minutos
```

2.2.9 Alínea i) - Desconto total obtido no mês

Desenho da solução

No cálculo do desconto total obtido no mês, optou-se por calcular o total de desconto do cliente.

Para a obtenção do desconto total mensal foi preciso utilizar a espressão regular `/<VALOR_DESCONTO>/` de modo a fazer *match* com a *tag* `<VALOR_DESCONTO>` do ficheiro lido. Ao fazer *macth* é executada a seguinte ação:

- ler o valor do campo, `$3`, valor monetário que representa o desconto obtido pelo cliente numa transação e guardá-lo na variável `valor`.
- *split* da variável `valor`, e substituição da vírgula que separa a parte inteira do número da parte decimal por um ponto, de modo a poder realizar operações aritméticas com estes valores.
- incremento da variável conta com o valor da variável valor

No fim da leitura do ficheiro é executado o bloco `END` no qual se imprime para o `stdout` o valor da variável `conta`, ou seja, o desconto total obtido no mês.

Testes e resultados

O filtro `viaverde_i.awk` foi testado com o ficheiro `viaverde_identificadores.txt` e obteve-se os seguintes resultados:

```
Valor total do desconto (euros):
0.75 euros
```

Capítulo 3

Albúm Fotográfico em HTML

3.1 Descrição do problema

O núcleo português do Museu da Pessoa (http://npmp.epl.di.uminho.pt/), dispõe de uma enorme colecção de fotografias (umas em formato `JPEG` outras em `PNG`). A informação das fotografia encontra-se anotada num ficheiro `XML` que, disponibiliza a respetiva meta-informação. Este ficheiro apresenta, para cada fotografia, informações como:

- identificação do **ficheiro** da foto
- o local **onde** a foto foi tirada
- **quem** se encontra na fotografia
- entre outras informações

Como exemplo foi disponibilizado o ficheiro legenda.xml.
Os ficheiros `XML` com a meta informação relativa às fotografias segue uma estrutura deste tipo:

- Começa com um cabeçalho:

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <!DOCTYPE fotos SYSTEM "Users/sony/mp/fotos.dtd">
  ```

  Este cabeçalho apenas fornece indicações no inicio do ficheiro sobre a versão do `XML` utilizada e que se está a utilizar a codificação `UTF-8`.

- Para cada fotografia apresenta a respetiva informação, de forma geral :

  ```
  <foto ficheiro= NOME >
   <onde> ONDE </onde>
   <quando data="yyyy-mm-dd"/>
   <quem> QUEM </quem>
  </foto>
  ```

- Sendo que, opcionalmente, algumas fotos ainda possam conter a seguinte informação:

  `<legenda> pequena legenda da foto </legenda>`

  `<facto> alguma descrição </facto>`

Pretende-se desenvolver processadores de texto com o `GAWK` para ler ficheiros deste tipo de forma a:

1 Gerar um ficheiro `HTML` que corresponda ao álbum com todas as fotografias. Considerou-se interessante apresentar o álbum de fotografias dispondo-as de forma cronológica

2 Listar, em repetições, todos os locais fotografados

A solução destes dois problemas foi desenvolvida no mesmo ficheiro `.GAWK`, gerando-se o ficheiro com a informação da página `HTML` e, também, imprimindo no *standart output* a lista dos locais fotografados.

No entanto, no relatório optou-se por descrever os dois problemas e as respetivas soluções separadamente, de forma independente.

3.2 Implementação

3.2.1 Gerar o albúm fotográfico em HTML

Pretende-se gerar uma página `HTML` a partir de ficheiros `XML` do tipo especificado. Para tal, começou-se por elaborar um esboço da página que se pretende gerar, de forma a representá-la esquematicamente. Na Figura 3.1 encontra-se representado o esquema projetado da página `HTML` a ser criada.

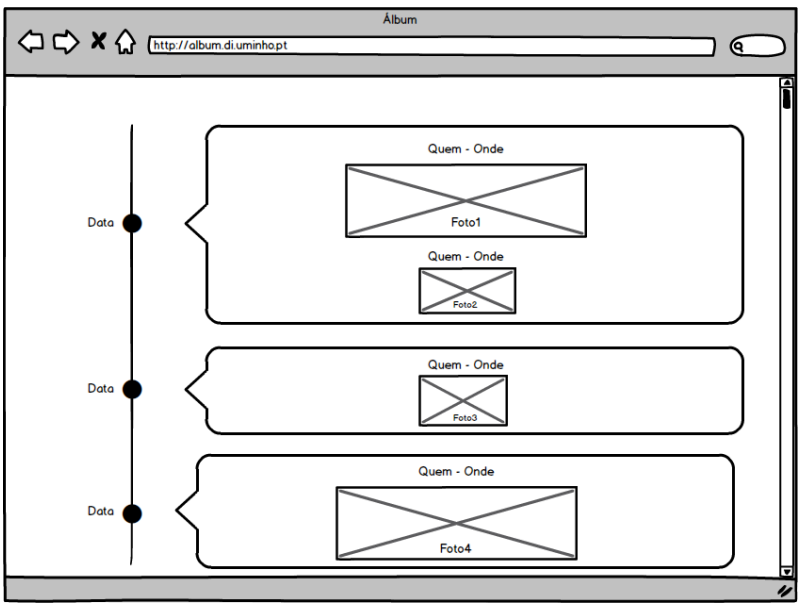

Figura 3.1: Esquematização da página `HTML` com o álbum fotográfico dispondo as fotografias de forma cronológica

Portanto, desta forma percebe-se que tipo de informação se requer para obter a página. Esta informação tem que ser identificada para que possa ser extraída do ficheiro fonte. Identificou-se no ficheiro fonte o padrão e a forma onde se encontra a informação que se pretende extrair:

- **nome da fotografia:** identificação da fotografia , no ficheiro fonte:

  Inicia-se: `<foto`

  Termina: `</foto>`

- **data:** data da fotografia , no ficheiro fonte:

  Inicia-se: `<quando data ="`

  termina: `</quando>`

- **onde:** local onde foi tirada a fotografia, no ficheiro fonte:

  Inicia-se: `<onde>`

  Termina: `</onde>`

- **quem:** identifica quem se encontra na fotografia:

  Inicia-se: `<quem>`

  Termina: `</quem>`

Assim, facilmente se observa de que forma se encontra a informação que se pretende colocar na página `HTML` mostrando o álbum fotográfico.

Desenho da solução

Abordaram-se vários pontos importantes no sentido de obter e definir a solução pretendida para gerar a página `HTML` do álbum cronológico. Seguem de seguida a descrição destes pontos.

Definir as variáveis associadas ao GAWK

Identificada a informação que se pretende extrair definiram-se as variáveis, que permitem ao `GAWK` "partir"o ficheiro `XML` em registos e campos.

- **NS** *(register separator)*: Analisando a forma como se encontra estruturado o ficheiro decidiu-se que é apropriado cada linha do próprio ficheiro ser definida como um registo, pois, cada linha contém um tipo de informação importante. Isto permite isolar em cada registo apenas um tipo de informação importante. Desta forma, cada registo contém informação que é passível de ser extraída (como o nome da foto, quando ou onde), e portanto, é sobre cada registo que deve ser feita a extração dos valores da informação.

  Como o `'\n'` já é o separador de registo por defeito não se especificou esta informação.

- **FS** *(register separator):* Analisando a estrutura de cada registo, que segue, de forma geral a seguinte forma:

  `<Elemento> informação </Elemento>\n`

  Decidiu-se utilizar como separador de registo `'<'` ou `'>'`. Estes caracteres permitem identificam o inicio e o fim de cada registo tornando-se por isso apropriados. Além disso, cada registo começa por `'<'` e segue-se a identificação do tipo de elemento que se encontra no registo até encontrar o próximo `'>'` e de imediato vem a **informação** que interessa extrair, que vai desde `'>'` até `'<'`, ficando perfeitamente isolada.

  Especificou-se o separador de registo do ficheiro `GAWK`, no `BEGIN`, para apenas ser executado uma vez e ser válido para todo o ficheiro desde o inicio:

  FS = `"[><]"`

  Que permite definir através da expressão regular `[><]` que será ou `"<"` ou `">"`.

Desta forma consegue-se, perfeitamente, identificar a informação que se pretende a partir do campo 1 e ter acesso à informação correspondente através do campo 2, de forma genérica, tem-se:

- **campo 1:** `Elemento` (que poderá ser por exemplo: "quem"ou "onde".
- **campo 2:** `informação` (que poderá ser por exemplo: "Taberna do Fausto, Afurada"ou "Fausto Ferreira Gomes").
- **campo 3:** `/Elemento`

Nota: nem sempre esta estrutura foi encontrada exactamente desta forma para todos os registos. Por exemplo, quando o `Elemento = "foto"`, existe apenas um campo com toda a informação.

Especificar as expressões Regulares

A etapa seguinte consistiu em especificar os padrões através de expressões regulares que permitam capturar os campos pretendidos no texto-fonte.

Como se pretende obter o álbum de todas as fotografias então as ER utilizadas têm como objetivo apenas identificar os registos onde se encontram as informações pretendidas, para cada uma das fotografias.

Apresentam-se as especificações das expressões regulares:

- `/<foto /` permite encontrar os registos com o elemento `foto` o que possibilita identificar o nome das fotografias
- `/<onde/` permite encontrar os registos com o elemento `onde` e desta forma aceder à informação sobre o local onde a fotografia foi fotografada
- `/<quando /` permite encontrar os registos com o elemento `quando` possibilitando aceder à informação da data da fotografia

- `/<quem>/` permite encontrar os registos com o elemento `quem`, permitindo aceder à informação de quem se encontra na fotografia

Estas ER permitem identificar os registos com informação relevante. Não se filtrou nenhuma fotografia, apenas se identificaram todos os registos que apresentam dados que importam. Desta forma, filtram-se os registos que têm interesse para o álbum fotográfico cronológico.

Identificar as ações semânticas e Estruturas de Dados globais

Através das ER definidas é possível ter acesso aos registos que contêm a informação (num dos seus campos) que realmente importa! Acedendo a esses registos é então possível manipulá-los de forma a extrair o campo pretendido. Para tal, associado a cada ER definiu-se a respetiva acção semântica que é efectuada sempre que a ER correspondente se verificar.

A ideia consiste em passar pelo ficheiro-fonte e, para cada foto, juntar toda a informação correspondente de forma a dispo-la como pretendido no álbum a gerar.

Para cada foto, as respetivas informações encontram-se de forma sequencial e, portanto, garante-se que cada foto passa pelo conjunto das quatro ERs especificadas de forma sequencial.

Assim, definiu-se um array que se dominou de `datas` em que o índice corresponde a uma data e cada posição contém uma *string* que foi sendo concatenada. Nesta **string** encontra-se a informação de todas as fotografias associadas à data definida pelo respetivo índice. Desta forma, na *string* a informação de uma foto é separada da informação de outra foto através do caracter `'@'`. Para cada foto as suas informações são separadas pelo caracter `';'`.

Apresenta-se, de forma esquemática como se estrutura a variável global definida `data`, exemplificando para duas fotos fotografadas a `"2001-03-23"`. Assim o elemento do array `datas` na posição `"2001-03-23"` contém:

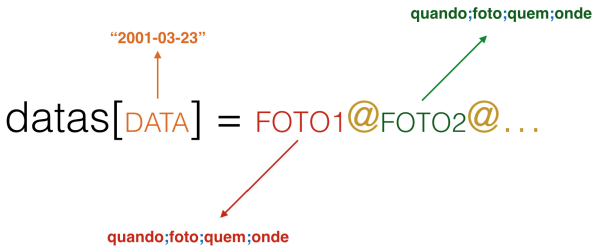

Figura 3.2: Esquematização da variável global `datas`

Portanto, para cada ER definiram-se as seguintes acções :

- `/<foto /` Sabe-se que este registo contém a informação do nome da fotografia no campo 2, neste formato `ficheiro = "nome da fotografia"`, portanto: - Fez-se um *split* para extrair e obter apenas o nome da fotografia, que se guarda temporariamente na variável global `nomeFoto`
- `/<onde/` Este registo contém a informação do local onde se fotografou a foto no campo 3. Apesar do campo 3 apenas conter o sitio da fotografia existiu a necessidade de normalizar esta *string* retirando espaços desnecessários através da função definida `normalizarTexto`.

- `/<quando /` Este registo contém a informação da data da fotografia no campo 2, neste formato `data = "yyyy-mm-dd"`, portanto:
  - Fez-se um *split* para extrair e obter apenas o valor da data, correspondente a: `"yyyy-mm-dd"`. Guarda-se este valor temporariamente na variável global `quando`
- `/<quem>/` Este registo contém a *string* com a informação de quem se encontra na fotografia, no campo 3. Este campo também é normalizado através da função definida `normalizarTexto` para retirar espaços extra. Neste ponto, para cada fotografia, já se recolheram todas as informações necessárias nas variáveis globais e portanto: - Todos os campos são concatenados e separados por ';' formando uma *string* que é guardada temporariamente na variável `conteudoFoto` - Esta *string* vai ser guardada no array datas na posição do array com o valor da variável global `quando` - A *string* da variável `conteudoFoto` é concatenada com a informação de outras fotos que eventualmente existam na mesma posição do array `datas`, sendo separadas por `'@'`.

No final, ou seja no `END`, toda a informação está contida no array `datas`. Este array é ordenado pelo índice de forma a que fique ordenado cronologicamente com a data mais antiga a corresponder à primeira posição do array. O array é percorrido e a informação vai sendo escrita, de acordo com os *templates* definidos em variáveis globais, para o ficheiro `HTML` que se denominou "`index.html`"

O código do ficheiro `AWK` final obtido pode ser consultado em anexo.

Testes e resultados

Para executar o ficheiro de `AWK` definido e explicado anteriormente, foi executado o comando: `gawk -f album.awk legenda.xml`.

A execução deste comando, gera o ficheiro `index.html`. Neste ficheiro `HTML` encontra-se a informação definida no ficheiro `AWK` e produzida pela execução do comando. A página produzida após a execução do comando com o ficheiro `legendas.xml` pode ser consultada na Figura 3.3.

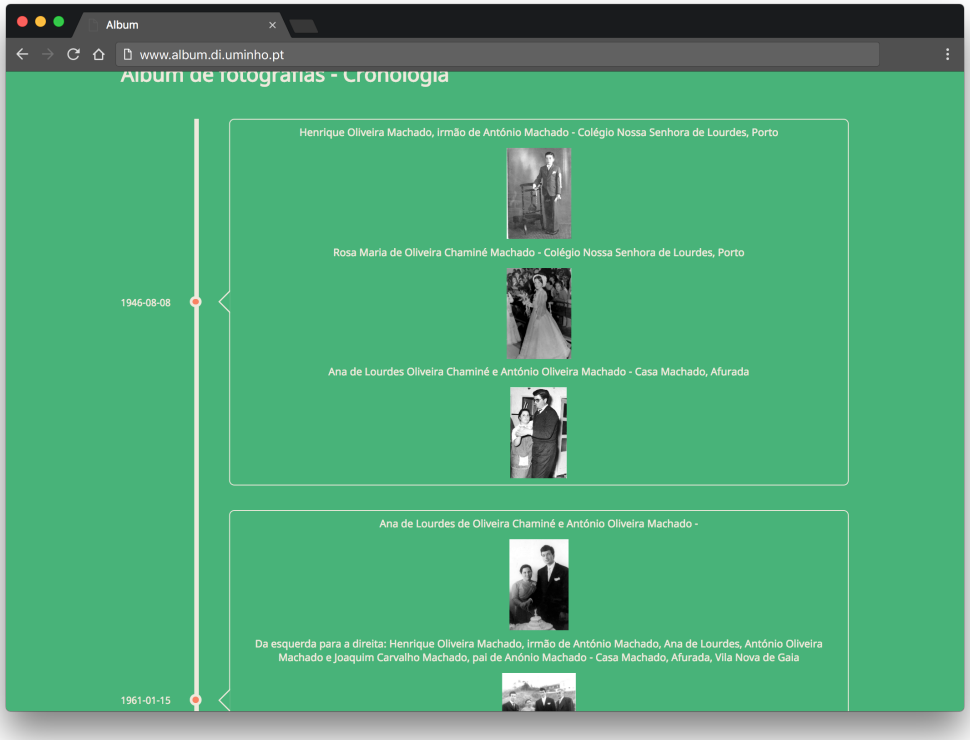

Figura 3.3: Página `HTML` gerada com o álbum fotográfico dispondo as fotografias por ordem cronológica

Nota: Para efeito de embelezamento da página foram adicionados estilos que modificam a aparência da página. Utilizou-se `CSS` para complementar o `HTML` já definido e consegui-se dar a ideia de cronologia. Estes estilos encontram-se definidos no cabeçalho da página.

Esta página encontra-se idêntica ao rascunho definido que se seguiu e ambicionou obter. As datas estão ordenadas de forma cronológica (das mais antigas para as mais recentes). Em cada data estão agrupadas as fotos datadas desse dia. Cada foto aparece ilustrada na página e contém informação de quem está associado à foto e onde esta foi tirada.

3.2.2 Listar a lista de locais fotografados (sem repetições)

Neste problema pretende-se obter a lista de todos os locais fotografados sem que existam repetições.

Desenho da solução

O ficheiro-fonte fornecido como exemplo "legenda.xml"é o mesmo que utilizado para gerar a página html no ponto anterior e, portanto, apresenta a mesma estrutura. Contudo, neste problema, apenas se pretende aceder à informação do local onde foi tirada a fotografia.

Definir as variáveis associadas ao `GAWK` e Especificar as expressões Regulares

Da mesma forma que no problema anterior definiu-se:

- **NR:** como `'\n'`
- **FR:** como `[><]`

E por isso, cada registo e cada campo mantêm a mesma definição.

Especificar as Expressões Regulares

Neste caso, pretende apenas ter acesso a registos que contenham o elemento "onde", ou seja, a registo que sigam a seguinte estrutura:

`<onde> informação </onde>`

Desta forma, especificou-se somente a expressão regular correspondente:

- `/<onde`: permite encontrar os registos com o elemento "onde" e, assim, aceder à informação sobre o local onde a fotografia foi fotografada.

Identificar as ações semânticas e Estruturas de Dados globais

Definiu-se um array global que se denominou de *locais* em que cada posição corresponde a um local onde se fotografou uma fotografia.

Então, para a ER especificada, definiram-se as seguintes ações semânticas:

- `/<onde`: Um registo que concorde com esta ER, contém a informação do local onde a fotografia foi tirada, nomeadamente, no campo 3.
  - A *string* contida no campo 3 foi normalizada com o objetivo de retirar espaços desnecessários através da função definida `normalizarTexto`. Aqui, esta normalização torna-se essencial, uma vez que, não se requer que não exista qualquer tipo de ambiguidades nas *strings* que identificam os locais fotografados para evitar repetições. Por exemplo, dois locais iguais que apenas defiram num caracter espaço no inicio ou no fim da *string* iriam definir diferentes locais correspondendo a duas entradas distintas na lista final, originando repetidos.
  - Após normalizar o campo 3, o resultado é adicionado ao array `locais`. Em cada foto, acede-se ao índice do local e é reescrita a informação do local. Esta unicidade de índices nos *arrays* em `AWK`, garante que cada local apenas aparece uma vez, assegurando-se assim que não ocorrem repetidos.

No final, o array locais é iterado e é impressa para o *standart output* a lista dos locais onde as fotografias foram tiradas.

O código do ficheiro `awk` final obtido pode ser consultado em anexo.

Testes e resultados

Para além da página `HTML` anteriormente mostrada, após a execução do comando `awk` com o ficheiro de `awk` definido, aplicado a um ficheiro com a estrutura definida, o *output* da ferramenta de `AWK` apresenta uma lista dos locais de todas as fotos sem repetições.

Quando aplicado ao ficheiro `legendas.xml`, produz a lista dos locais fotografados existentes neste ficheiro que se apresenta de seguida:

```
Lista de locais fotografados:
        Igreja Santa Marinha Afurada
        Casa Machado, Rua 27 de Fevereiro, Afurada
        Casa Machado
        Casa Machado, Afurada, Vila Nova de Gaia
        Taberna do Fausto, Afurada
        Colégio Nossa Senhora de Lourdes, Porto
        Taberna do Fausto, na Afurada
        Casa Machado, Afurada
        Taberna Neca Chaminé, Afurada
```

Capítulo 4

Conclusão

Neste trabalho realizaram-se dois enunciados: o da Via Verde e o do Álbum Fotográfico. Estes problemas consistiram em implementar filtros `AWK` para tratar de ficheiros de texto comuns que facilmente se encontram no dia a dia e que contêm muita informação. Aplicar os filtros nestes ficheiros permite extrair apenas a informação pertinente que se pretende obter, podendo mesmo, agrupá-la de diferentes formas de acordo com o pretendido. Tornando-se assim uma ferramenta muito poderosa que permite manipular ficheiros de grande dimensão.

A resolução do enunciado da Via Verde permitiu a sedimentação dos conteúdos lecionados nas aulas. Apesar de se tratar de um ficheiro com um conteúdo aparentemente simples, este permitiu a extração de variada informação, para além dos quatro tópicos requeridos no enunciado, e além disso decidiu-se extrair-se informação por cliente e por identificador de veículo do cliente em algumas das alíneas resolvidas. No decorrer do desenvolvimento deste enunciado desenvolveu-se a capacidade de escrever expressões regulares para a descrição de padrões de frases, usando-as num sistema de produção para filtragem de texto `AWK`.

Na resolução do trabalho do Álbum Fotográfico optou-se por seguir um desafio um pouco diferente do pedido no enunciado que se considerou mais interessante. Gerou-se uma página HTML que possibilita ver o álbum fotográfico de forma cronológico possibilitando ordenadas todas as fotos pela respetiva data!

O foco central com a resolução deste trabalho prático é a consolidação dos conhecimentos adquiridos sobre `AWK`, logo conclui-se que os objectivos foram plenamente cumpridos com a resolução de dois enunciados dos quatro propostos.

Apêndice A

Código do Processador de Texto

A.1 Processador de transações da Via Verde

A.1.1 Alínea a) - Calcular o número de entradas em cada dia do mês

```
BEGIN                       { FS="[><]"; print"Data\t\tNr␣Ocorrencias"}
/<DATA_ENTRADA>/ && $3!="null" { entradasDia[$3]++; }
END { for(dia in entradasDia)  { print dia ":\t\t" entradasDia[dia]}}
```

A.1.2 Alínea b) - Escrever a lista de locais de saída

```
BEGIN       { FS = "[><]" ; print "Locais␣de␣saida:␣\n" }
/<SAIDA>/   { saidas[$3] }
END         { for(i in saidas) print "\t" i }
```

A.1.3 Alínea c) - Calcular o total gasto no mês

Versão i)

```
BEGIN               { FS = "[><]" ; print "Gasto␣Mensal␣(em␣euros)";
   }
/<IDENTIFICADOR/    { split($2, a, "␣"); id = a[2]; }
/<MATRICULA>/       { matricula = $3; }
/<\/IDENTIFICADOR>/ { id = null; }
/<TOTAL>/           { valor = $3;
                      if (id==null)
                          print  "Total\t\t\t" valor;
                      else
                          print matricula "\t\t" valor; }
END{ }
```

Versão ii)

```
BEGIN                { FS = "[><]"; print "Gasto␣Mensal␣(em␣euros)";
                       contaId = 0; contaTotal = 0; }
/<MATRICULA>/        { matricula = $3; }
/<IMPORTANCIA>/      { valor = $3;
                       split(valor, a, ",");
                       valor = a[1] "." a[2];
                       contaId += valor; }
/<\/IDENTIFICADOR>/ { print matricula "␣\t\t" contaId;
                       contaTotal += contaId;
                       contaId = 0; }
END                  { print "Total\t\t\t" contaTotal; }
```

A.1.4 Alínea d) - Calcular o total gasto no mês apenas em parques

```
BEGIN                { FS = "[><]"; print "Gasto␣Mensal␣Parques(em␣
   euros)";
                       contaId = 0; contaTotal = 0; }
/<MATRICULA>/        { matricula = $3; }
/<IMPORTANCIA>/      { valor = $3; }
/<TIPO>/ && $3 =="Parques␣de␣estacionamento" { split(valor, a, ",");
                                               valor = a[1] "." a[2];
                                               contaId += valor; }
/<\/IDENTIFICADOR>/ { print matricula "␣\t\t" contaId;
                       contaTotal += contaId; contaId = 0; }
END                  { print "Total␣\t\t\t" contaTotal; }
```

A.1.5 Alínea e) - Lista das datas em que foram efetuados débitos

```
BEGIN              { FS = "[><]" ; print "Datas␣de␣debito:" }
/<DATA_DEBITO>/    { datadeb[$3] }
END                { for(i in datadeb) print i }
```

A.1.6 Alínea f) - Calcular a média dos pagamentos de cada mês

```
BEGIN { FS = "[><]";
        valorIdPortagens = 0;
        valorIdParques = 0;
        valorTotalIdParques = 0;
        valorTotalIdPortagens = 0
        nrOcorrenciasIdPortagens = 0;
        nrOcorrenciasTotalIdPortagens = 0;
        nrOcorrenciasIdParques = 0;
        nrOcorrenciasTotalIdParques=0;
        nrOcorrenciasTotal = 0;
        print "Media de Pagamento";
        print"\t\t\t\t\Portagens(euros/mes)\t\tParques(euros/mes)"; }
/<MATRICULA>/   { matricula = $3; }
/<IMPORTANCIA>/ { importancia = $3; }
/<TIPO>/ && $3 == "Portagens" { split(importancia, a,",");
                                importancia = a[1] "." a[2];
                                valorIdPortagens += importancia;
                                nrOcorrenciasIdPortagens++; }

/<TIPO>/ && $3 == "Parques de estacionamento" {
                                split(importancia, a,",");
                                importancia = a[1] "." a[2];
                                valorIdParques += importancia;
                                nrOcorrenciasIdParques++; }

/<\/IDENTIFICADOR>/ { printf matricula "\t\t\t %.2f \t\t\t\t %.2f\n",
                      (valorIdPortagens/nrOcorrenciasIdPortagens),
                      (valorIdParques/nrOcorrenciasIdParques);

        valorTotalIdPortagens += valorIdPortagens;
        nrOcorrenciasTotalIdPortagens += nrOcorrenciasIdPortagens;

        valorTotalIdParques += valorIdParques;
        nrOcorrenciasTotalIdParques += nrOcorrenciasIdParques;

        valorTotal += valorIdPortagens+valorIdParques;
        nrOcorrenciasTotal += nrOcorrenciasIdPortagens +
           nrOcorrenciasIdParques;

        valorIdPortagens = 0;
        valorIdParques = 0;
        nrOcorrenciasIdPortagens = 0;
        nrOcorrenciasIdParques = 0; }
```

```
43
44 END     { printf "Media␣Parcial\t\t\t␣%.2f␣\t\t\t\t␣%.2f\n",
45           (valorTotalIdPortagens/nrOcorrenciasTotalIdPortagens),
46           (valorTotalIdParques/nrOcorrenciasTotalIdParques);
47           printf "\nMedia␣Total:␣%.2f␣euros/mes\n",
48           (valorTotal/nrOcorrenciasTotal); }
```

A.1.7 Alínea g) - Escrever a lista de operadores e respetivo número de ocorrências

```
1 BEGIN         {FS = "[><]" ; print "Operador\t\t\t\t\tNr␣Ocorrencias"}
2 /<OPERADOR>/ { operadores[$3]++;}
3 END  { for(op in operadores) {print op "\t\t\t\t\t" operadores[op] }}
```

A.1.8 Alínea h) - Período de tempo passado em parques de estacionamento

```
1 BEGIN { FS = "[><]";
2         print "Parques␣";
3         tempoMinutosId = 0;
4         tempoMinutosTotal = 0; }
5
6 /<MATRICULA>/     { matricula = $3; }
7 /<HORA_ENTRADA>/ { horaentrada = $3; }
8 /<HORA_SAIDA>/   { horasaida = $3; }
9
10 /<TIPO>/ && $3 =="Parques␣de␣estacionamento" {
11                    split(horaentrada, a, ":");
12                    minutosEntrada = a[1]*60 + a[2];
13                    split(horasaida, b, ":");
14                    minutosSaida = b[1]*60 + b[2];
15                    tempoMinutosId+=(minutosSaida - minutosEntrada);}
16
17 /<\/IDENTIFICADOR>/  { print matricula "\t\t" int(tempoMinutosId/60)
18                        "␣horas␣e␣" (tempoMinutosId%60) "␣minutos";
19                        tempoMinutosTotal += tempoMinutosId;
20                        tempoMinutosId = 0; }
21
22 END { print "Total\t\t\t" int(tempoMinutosTotal/60) "␣horas␣e␣"
23       (tempoMinutosTotal%60) "␣minutos"; }
```

A.1.9 Alínea i) - Desconto total obtido no mês

```
1 BEGIN { FS = "[><]";
2         print "Valor␣total␣do␣desconto␣(euros):" ; conta = 0;}
3
4 /<VALOR_DESCONTO>/  { valor = $3;
5                       split(valor, a, ",");
6                       valor = a[1] "." a[2];
7                       conta += valor;}
8
9 END                 { print conta "␣euros"}
```

A.2 Álbum Fotográfico em HTML

```
BEGIN {
        FS = "[><]";
        path = "http://npmp.epl.di.uminho.pt/images/galleryInterviews
           /"

        templateDataInicio = "<li class='work'> <input class='radio'
           id='work5' name='works' type='radio' checked> <div class='
           relative'> <span class='date'>%s</span> <span class='
           circle'></span> </div> <div class='content'>";
        templateDataFim = "</div> </li>";

        templateFotoF = "<p>%s - %s</p> <img src='%s' alt=''/>";

        cabecalho = "<!DOCTYPE html> <html> <head> <title>Album</
           title> <style>@import url(http://fonts.googleapis.com/css?
           family=Noto+Sans); body { max-width: 1200px; margin: 0
           auto; padding: 0 5%; font-size: 100%; font-family: 'Noto
           Sans', sans-serif; color: #eee9dc; background: #48b379; }
           h2 { margin: 3em 0 0 0; font-size: 1.5em; letter-spacing:
           2px; text-transform: uppercase; } /*
           ---------------------------------- * timeline *
           ----------------------------------*/ #timeline { list-
           style: none; margin: 50px 0 30px 120px; padding-left: 30px
           ; border-left: 8px solid #eee9dc; } #timeline li { margin:
           40px 0; position: relative; } #timeline p { margin: 0 0
           15px; } .date { margin-top: -10px; top: 50%; left: -158px;
           font-size: 0.95em; line-height: 20px; position: absolute;
           } .circle { margin-top: -10px; top: 50%; left: -44px;
           width: 10px; height: 10px; background: #48b379; border: 5
           px solid #eee9dc; border-radius: 50%; display: block;
           position: absolute; } .content { max-height: 20px; padding
           : 10px 20px 0; border-color: transparent; border-width: 2
           px; border-style: solid; border-radius: 0.5em; position:
           relative; } .content:before, .content:after { content: '';
           width: 0; height: 0; border: solid transparent; position:
           absolute; pointer-events: none; right: 100%; } .content:
           before { border-right-color: inherit; border-width: 20px;
           top: 50%; margin-top: -20px; } .content:after { border-
           right-color: #48b379; border-width: 17px; top: 50%; margin
           -top: -17px; } .content p { max-height: 0; color:
           transparent; text-align: center; word-break: break-word;
           hyphens: auto; overflow: hidden; } label { font-size: 1.3
```

```
em; position: absolute; z-index: 100; cursor: pointer; top
: 20px; transition: transform 0.2s linear; } .radio { 
display: none; } .radio + .relative label { cursor: auto; 
transform: translateX(42px); }
```

```
11      .radio + .relative .circle { background: #f98262; } .radio ~ .
        content { max-height: inherit; border-color: #eee9dc; margin-
        right: 20px; transform: translateX(20px); transition: max-
        height 0.4s linear, border-color 0.5s linear, transform 0.2s
        linear; } .radio ~ .content p { max-height: 200px; color: #
        eee9dc; transition: color 0.3s linear 0.3s; } .radio ~ .
        content img { max-width: 100%; max-height: 150px; display:
        block; margin-left: auto; margin-right: auto; padding-bottom:
        10px; color: #eee9dc; transition: color 0.3s linear 0.3s; } /*
         ------------------------------------ * mobile phones (
        vertical version only) * ------------------------------------
         */ @media screen and (max-width: 767px) { #timeline { margin-
        left: 0; padding-left: 0; border-left: none; } #timeline li {
        margin: 50px 0; } label { width: 85%; font-size: 1.1em; white-
        space: nowrap; text-overflow: ellipsis; overflow: hidden;
        display: block; transform: translateX(18px); } .content {
        padding-top: 45px; border-color: #eee9dc; } .content:before, .
        content:after { border: solid transparent; bottom: 100%; } .
        content:before { border-bottom-color: inherit; border-width:
        17px; top: -16px; left: 50px; margin-left: -17px; } .content:
        after { border-bottom-color: #48b379; border-width: 20px; top:
         -20px; left: 50px; margin-left: -20px; } .content p { font-
        size: 0.9em; line-height: 1.4; } .circle, .date { display:
        none; } }</style> </head> <body> <h1>Album de fotografias -
        Cronologia</h1> <ul id='timeline'>";
12
13  print cabecalho > "index.html";
14  end = "</ul></body></html>";
15  }
16
17  /<foto / {split($2, a, "\"");
18                                  nomeFoto = a[2] }
19
20  /<onde/         {       ondeNormalizado = normalizarTexto($3); }
21
22  /<quando /      {       split($2, dataFoto, "\"")
23                          quando = dataFoto[2] }
24
25  /<quem>/         {
26                          if (quando != null) {
```

```
27                                          nomeNormalizado =
                                               normalizarTexto($3);
28
29                                          if (ondeNormalizado != null)
                                               locais[ondeNormalizado] =
                                               ondeNormalizado;
30
31                                          conteudoFoto = quando ";"
                                               nomeFoto ";"
                                               nomeNormalizado ";"
                                               ondeNormalizado
32
33                                          if(datas[quando] != null)
                                               datas[quando] = datas[
                                               quando] "@" conteudoFoto;
34                                          else datas[quando] =
                                               conteudoFoto;
35                                  }
36
37                                  quando = null;
38 }
39
40 END{
41          asort(datas);
42
43          for(data in datas) {
44
45                  split(datas[data], fotos, "@");
46
47                  for (i in fotos) {
48                          split(fotos[i], campos, ";");
49
50                          quando = campos[1];
51                          nomeFoto = campos[2];
52                          quem = campos[3];
53                          onde = campos[4];
54
55                          if (i == 1) printf(templateDataInicio, quando
                               ) > "index.html";
56
57                          printf(templateFotoF, quem, onde, path
                               nomeFoto) > "index.html"
58                  }
59
60                  print templateDataFim > "index.html";
```

```
61          }
62
63          print "Lista␣de␣locais␣fotografados:"
64          for (local in locais) print "\t" local
65
66          print end > "index.html";
67  }
68
69
70  function normalizarTexto(textoComEspacos) {
71          tamanho = split(textoComEspacos, textoSemEspacos, "␣");
72
73      for (i = 1; i <= tamanho; i++) {
74          if (i == 1) result = textoSemEspacos[i];
75          else result = result "␣" textoSemEspacos[i];
76      }
77
78      return result
79  }
```