

Processamento de Linguagens (3º Ano MIEI)

**Trabalho Prático 1a - GAWK**

Relatório de Desenvolvimento

Fábio Luís Baião da Silva  
(A75662)

João da Cunha Coelho  
(A74859)

Luís Miguel Moreira Fernandes  
(A74748)

15 de Março de 2017

## Resumo

A Via Verde é um sistema eletrónico de portagens usado em Portugal desde 1991. A *matéria-prima* para este projeto é um extrato mensal de um dos seus utilizadores, o qual será utilizado para responder a diversas *queries* usando **GAWK**, uma das ferramentas de processamento de texto mais eficientes do GNU/Linux. As expressões regulares são padrões aos quais o mecanismo de expressões regulares tenta corresponder no texto de entrada. Neste trabalho, a conjugação do **GAWK** com **Expressões Regulares** acarreta o objetivo de demonstrar a utilidade e o poder destas ferramentas na resolução de um problema do quotidiano.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Estrutura do Ficheiro XML</b>	<b>3</b>
<b>3</b>	<b>Apresentação da Solução de Leitura e Parser</b>	<b>4</b>
3.1	Record Separator (RS) . . . . .	4
3.2	Field Separator (FS) . . . . .	4
<b>4</b>	<b>Apresentação das Queries e Resoluções</b>	<b>5</b>
4.1	Queries de Resolução Obrigatória . . . . .	5
4.2	Queries Adicionais . . . . .	8
4.3	Outros esclarecimentos . . . . .	10
<b>5</b>	<b>Apresentação de Resultados</b>	<b>11</b>
5.1	Execução . . . . .	11
5.2	Exemplos . . . . .	11
<b>6</b>	<b>Conclusão</b>	<b>16</b>

# Capítulo 1

## Introdução

O primeiro contacto com as expressões regulares despertou grande curiosidade, dada a capacidade de analisar e interpretar um ficheiro de texto, que pode ser longo, em pouco tempo, recolhendo de uma só vez informação suficiente para responder a várias interrogações. Este trabalho permitirá comprovar essa utilidade e eficiência do **GAWK**, aplicando-o a um exemplo prático do dia-a-dia.

A escolha recaiu no tema da Via Verde devido à clareza do enunciado e à utilidade do tema, uma vez que este tipo de processador de transações pode ser aplicado nas mais diversas áreas.

Um dos principais objetivos na realização deste projeto é criar um processador de texto geral, permitindo pequenas alterações na estrutura dos ficheiros sem que comprometam os resultados apresentados.

## Capítulo 2

# Estrutura do Ficheiro XML

De seguida é sucintamente explicada a estrutura do ficheiro XML em análise.

**Cabeçalho com informação sobre o mês a que se refere o extrato, os dados do cliente, etc:**

```
<EXTRACTO id="011114056/08/2015">
<MES_EMISSAO>Ago-2015</MES_EMISSAO>
<CLIENTE id="514714936">
  <NIF>987653210</NIF>
  <NOME>PEDRO MANUEL RANGEL SANTOS HENRIQUES</NOME>
  <MORADA>RUA XXX</MORADA>
  <LOCALIDADE>BRAGA</LOCALIDADE>
  <CODIGO_POSTAL>4715-012 BRAGA</CODIGO_POSTAL>
</CLIENTE>
<IDENTIFICADOR id="28876820811">
<MATRICULA>00-LJ-11</MATRICULA>
<REF_PAGAMENTO>1234567</REF_PAGAMENTO>
```

Todas as transações de serviços oferecidos pela ViaVerde (portagens, parques de estacionamento, etc), sendo que cada transação tem campos como: data e hora de entrada e saída, local de entrada e saída, valor, etc:

```
<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>
...
```

**Rodapé com valores totais:**

```
<TOTAL>77,40</TOTAL>
</IDENTIFICADOR>
<TOTAL>77,40</TOTAL>
<TOTAL_IVA>14,49</TOTAL_IVA>
</EXTRACTO>
```

## Capítulo 3

# Apresentação da Solução de Leitura e Parser

```
RS = "<TRANSACCAO>";  
FS = "[<>]";
```

### 3.1 Record Separator (RS)

A opção passou por separar os *records* pela tag `<TRANSACAO>`.

Com o *Record Separator* definido desta forma sabe-se que o primeiro *record* corresponde ao cabeçalho, enquanto que os restantes irão corresponder às transações a analisar. A última transação contém, ainda, o rodapé.

Desta forma assegura-se que, ao processar cada transação, todos os dados que se encontrem nesse *record* pertencem à mesma transação.

### 3.2 Field Separator (FS)

Aqui foi tomada a decisão de escolher como *field separator* os caracteres `<e >`. Cada transação é definida por vários segmentos com o seguinte formato:

```
<...>...</...>
```

Com este *field separator*, os dados relevantes para análise encontrar-se-ão sempre no segundo e terceiro campo (tag e valor, respetivamente) de cada linha. Desta forma, para se obter o valor de determinada tag, basta iterar pelos campos referentes à tag em cada linha, obtendo-se a seguinte função:

```
function getValueOf (tag){  
    for (i=2; i < NF; i+=4){  
        if ($i == tag)  
            return $(i+1);  
    }  
    return null;  
}
```

Com esta função a obtenção do valor de determinada tag não depende da ordem que esta aparece no ficheiro XML.

## Capítulo 4

# Apresentação das Queries e Resoluções

Neste capítulo serão explicadas as queries realizadas. Na primeira secção apresentar-se-á as queries obrigatórias sendo, de seguida, apresentadas as queries criadas adicionalmente pelo grupo.

### 4.1 Queries de Resolução Obrigatória

A escolha dos **RS** e **FS** permite-nos encontrar, para cada transação, o valor do campo que pretendemos, invocando somente a função *getValueOf*, sendo o argumento esse mesmo campo.

a) Calcular o número de 'entradas' em cada dia do mês

Para além do número de entradas, extendemos o âmbito da *query* de modo a distinguir as entradas nos diferentes serviços - portagens ou parques de estacionamento.

O objetivo da inversão da data é permitir, posteriormente, a ordenação. A função *inverter* usa a função *split* para dividir a data em dia, mês e ano, com o separador -.

```
NR > 1 {
    tipo = getValueOf("TIPO");
    data = getValueOf("DATA_ENTRADA");
    if (tipo != null && data != null && data != "null"){
        invData = inverter(data);
        nEntradas[tipo][invData]++;
    }
}
```

Para a impressão dos resultados foi criada uma tabela *html*, porém aqui mostrar-se-á o processo de impressão no terminal para que seja mais fácil explicar o processo.

Começou-se por separar os resultados por tipo. Em seguida, como se pretendia a impressão ordenada das datas, utilizou-se a função *asorti* para criar um array com as datas ordenadas de forma crescente, iterando-o depois no momento de imprimir o valor do número de entradas.

O resultado que é apresentado é a data original (é necessário inverter já que foi invertido anteriormente para se proceder à ordenação) e o número de entradas correspondente a essa data (este valor é obtido no array original, já que estes valores foram perdidos no array ordenado).

```
END {
    for (i in nEntradas){
        print i;
```

```

        n = asorti(nEntradas[i], ordenado);
        for (j=1; j <= n; j++){
            data = ordenado[j];
            print inverter(data), nEntradas[i][data];
        }
    }
}

```

b) Escrever a lista de locais de saída

Para esta *query* adicionou-se uma funcionalidade que permite saber, a cada local, o número de vezes que este foi utilizado como local de saída, bem como o tipo de serviço. O valor de **saidas[tipo][saida]** representa o número de vezes que a saída *saida* foi usada para o serviço *tipo*.

```

NR > 1 {
    tipo = getValueOf("TIPO");
    saida = getValueOf("SAIDA");
    if (tipo != null && saida != null){
        saidas[tipo][saida]++;
    }
}

```

Uma vez que os dados são ordenados a partir do número de saídas, o processo para a apresentação dos resultados nesta *query* é mais complexo, já que para cada número de saída é necessário descobrir no array original qual a saída correspondente. Quando essa saída é encontrada, esta é apagada do array (com recurso à função `delete`) para que, caso existam várias saídas com o mesmo número de ocorrências, estas não se repetiam nem haja omissões.

Outro aspeto a considerar é que, já que se pretende apresentar os número de saída por ordem descendente, é necessário iterar do fim até ao início do array ordenado.

```

END {
    for (i in saidas){
        n = asort(saidas[i], ordenado);
        for (j=n; j > 0; j--){
            numeroSaidas = ordenado[j];
            for (k in saidas[i]){
                if (saidas[i][k] == numeroSaidas){
                    print k, numeroSaidas;
                    delete saidas[i][k];
                }
            }
        }
    }
}

```

Nota: foi criada, adicionalmente, uma *query* idêntica para os locais de entrada, sendo a única alteração o valor da tag pretendida para "ENTRADA".

```

NR > 1 {
    tipo = getValueOf("TIPO");
    entrada = getValueOf("ENTRADA");
    if (tipo != null && entrada != null){
        entradas[tipo][entrada]++;
    }
}

```



```

}

END    {
    for (i in entradas){
        n = asort(entradas[i], ordenado);
        for (j=n; j > 0; j--){
            numeroEntradas = ordenado[j];
            for (k in entradas[i]){
                if (entradas[i][k] == numeroEntradas){
                    print k, numeroEntradas;
                    delete entradas[i][k];
                }
            }
        }
    }
}

```

c) Calcular o total gasto no mês

O total gasto no mês é calculado a partir da soma dos valores das importâncias e a subtração dos descontos. É necessário substituir a 'vírgula', presente nos valores, por 'ponto' já que, para realizar as operações aritméticas, o separador decimal é o 'ponto'.

```

BEGIN    {
    total = 0;
}

NR > 1 {
    imp = getValueOf("IMPORTANCIA");
    desc = getValueOf("VALOR_DESCONTO");
    if (imp != null && desc != null){
        gsub(",", ".", imp);
        gsub(",", ".", desc);
        total += imp - desc;
    }
}

END    {
    printf ("TOTAL %.2f", total);
}

```

Para além do total, foi também calculado o valor do iva e o total sem iva. Uma vez que o valor do iva de cada transação está em percentagem é necessário dividir o valor por 100, para poder efetuar os cálculos corretamente.

```

BEGIN    {
    semIvaTotal = 0;
    ivaTotal = 0;
}

NR > 1 {
    imp = getValueOf("IMPORTANCIA");
    desc = getValueOf("VALOR_DESCONTO");
    ivaPerc = getValueOf("TAXA_IVA");
    if (ivaPerc != null && imp != null && desc != null){
        gsub(",", ".", imp);

```

```

        gsub(",", ".", desc);

        semIva = imp / (1 + ivaPerc/100) - desc;

        semIvaTotal += semIva;
        ivaTotal += semIva * (ivaPerc/100);
    }
}

END    {
    printf ("Total sem IVA %.2f", semIvaTotal);
    printf ("IVA %.2f", ivaTotal);
}

```

d) Calcular o total gasto no mês apenas em parques

Nesta *query* decidiu-se generalizar, apresentando os gastos mensais para todos os tipos existentes no extrato. Tal como no cálculo do total gasto no mês, também aqui os "totais" são calculados somando as importâncias e subtraindo os descontos.

```

NR > 1 {
    tipo = getValueOf("TIPO");
    imp = getValueOf("IMPORTANCIA");
    desc = getValueOf("VALOR_DESCONTO");
    if (tipo != null && imp != null) {
        gsub(",", ".", imp);
        gsub(",", ".", desc);
        tipos[tipo] += imp - desc;
    }
}

```

Para apresentar os resultados, e uma vez que se pretende ordenar pelos valores por ordem decrescente, é necessário descobrir os tipos que correspondem a cada valor, já que ao aplicar a função "asort" os índices (que, aqui, são os tipos) são substituídos por números. A função "delete" serve o mesmo propósito explicado numa das queries referidas anteriormente.

```

END    {
    n = asort(tipos, ordenado);
    for (i=n; i > 0; i--){
        valor = ordenado[i];
        for (j in tipos){
            if (tipos[j] == valor){
                printf ("%s %.2f", j, valor);
                delete tipos[j];
            }
        }
    }
}

```

## 4.2 Queries Adicionais

a) Gasto diário nos vários tipos de serviço disponibilizados pela Via Verde

Mais uma vez, inverte-se a data para permitir, posteriormente, a ordenação. Em cada transação calcula-se o valor gasto nessa transação, somando-o ao gasto, para a data e tipo dessa transação, já calculado.

```

NR > 1 {
    tipo = getValueOf("TIPO");
    data = getValueOf("DATA_ENTRADA");
    imp = getValueOf("IMPORTANCIA");
    desc = getValueOf("VALOR_DESCONTO");
    if (tipo != null && data != null && data != "null" && imp != null && desc != null){
        gsub(",", ".", imp);
        gsub(",", ".", desc);
        invData = inverter(data);
        gastoD[invData][tipo] += imp - desc;
    }
}

END {
    n = asorti(gastoD, ordenado);
    for (j=1; j <= n; j++){
        data = ordenado[j];
        for (k in gastoD[data]){
            printf ("%s %s %.2f", inverter(data), k, gastoD[data][k]);
            delete gastoD[i][k];
        }
    }
}

```

b) Débitos realizados por dia nos serviços Via Verde

Tal como nas outras queries, para apresentar os resultados ordenados pela data é necessário invertê-la. Neste caso, as datas utilizadas são as datas referentes à tag DATA\_DEBITO. O valor debitado é calculado, tal como nas outras queries, subtraindo o valor do desconto à importância.

```

NR > 1 {
    dataDebito = getValueOf("DATA_DEBITO");
    imp = getValueOf("IMPORTANCIA");
    desc = getValueOf("VALOR_DESCONTO");
    if (dataDebito != null && imp != null && desc != null){
        gsub(",", ".", imp);
        gsub(",", ".", desc);

        invData = inverter(dataDebito);
        debitos[invData] += imp - desc;
    }
}

END {
    n = asorti(debitos, ordenado);
    for (i=1; i <= n; i++){
        dataDebito = ordenado[i];
        printf ("%s %.f", inverter(dataDebito), debitos[dataDebito]);
    }
}

```

c) Dados do Cliente

Para terminar, foi criada uma query que permite apresentar os dados do cliente. Uma vez que estes dados se encontram no cabeçalho esta query é executada apenas uma vez no *number of records* = 1. Os dados do cliente

encontram-se entre as tags <CLIENTE> e <CLIENTE>. Assim o algoritmo a seguir apresentado imprime todos os dados do cliente que se encontrem entre estas duas tags.

Desta forma o número de dados pode variar de extrato para extrato, sendo os dados presentes nesses extratos sempre apresentados, independentemente do seu número.

```
NR == 1 {
    for (i=1; i < NF; i++){
        if ($i ~ /CLIENTE/){
            for (i += 2; $i !~ /CLIENTE/; i+=4){
                gsub("_", " ", $i);
                print $i, $(i+1);
            }
            break;
        }
    }
}
```

## 4.3 Outros esclarecimentos

Para a explicação do relatório optou-se por mostrar as queries separadamente. No entanto, no projeto criado, as queries encontram-se todas no mesmo ficheiro. Assim sendo, existem algumas alterações, como por exemplo, a invocação da função `getValueOf` apenas é realizada uma vez para cada tag. Também a função `gsub` é invocada apenas uma vez para cada valor (importância e desconto), não estando, portanto, no corpo de cada `if`.

Outro aspeto omitido na explicação das queries foi o facto de a apresentação dos dados ter sido feita em *HTML*. Cada query é apresentada em páginas HTML diferentes, tal como a página principal e a página dos dados do cliente. O grupo decidiu, então, criar o ficheiro HTML, correspondente à página principal, com o nome do mês a que se refere o extrato, sendo que todas as queries estarão numa pasta com o mesmo nome. Com isto, é possível processar vários ficheiros, com extratos referentes a meses diferentes, sem que nenhum dos ficheiros HTML seja perdido. O HTML que contém os dados do cliente é, no entanto, atualizado de acordo com o último XML processado.

O mês de emissão e a criação das pastas é efetuada da seguinte forma:

```
NR == 1 {
    extrato = getMesEmissao();

    system("mkdir -p " extrato);
}

function getMesEmissao(){
    for (i=1; i < NF; i++){
        if ($i == "MES_EMISSAO")
            return $(i+1)
    }
}
```

## Capítulo 5

# Apresentação de Resultados

Neste capítulo serão apresentados exemplos de utilização.

### 5.1 Execução

Para criar os resultados de um ficheiro XML com um extrato - *extrato.xml*, é necessário executar o seguinte comando na bash:

```
./viaverde extrato.xml
```

Depois disso são criados vários ficheiros HTML, sendo recomendado abrir o ficheiro **Mes-Ano.html** que corresponde ao menu principal.

### 5.2 Exemplos

Abrindo o menu principal aparece o seguinte resultado:

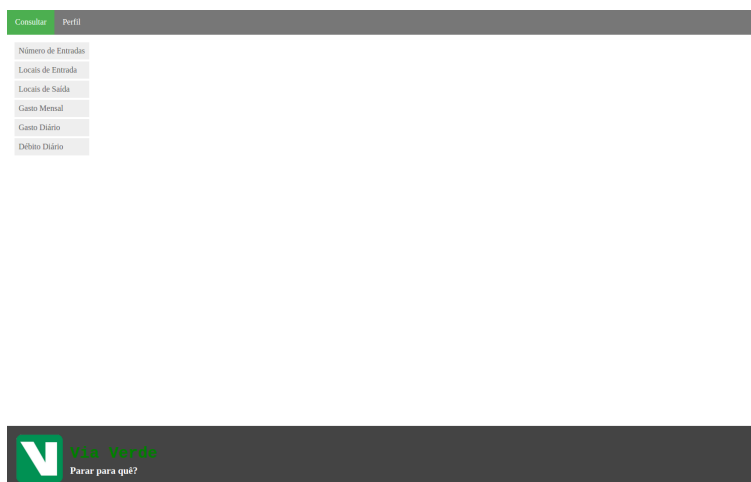



Figura 5.1: Menu Principal.


Utilizando o menu de navegação superior para clicar em Cliente, obtém-se a página relativa aos dados do cliente. A partir do menu lateral, na secção de Consultas, é possível consultar as queries realizadas.

Consultar

Perfil



NIF	987653210
NOME	PEDRO MANUEL RANGEL SANTOS HENRIQUES
MORADA	RUA XXX
LOCALIDADE	BRAGA
CODIGO POSTAL	4715-012 BRAGA



Via Verde  
Parar para quê?

Figura 5.2: Dados do Cliente.

Consultar

Perfil

Número de Entradas

Locais de Entrada

Locais de Saída


Gasto Mensal

Gasto Diário

Débito Diário

Número de entradas em cada dia, nos vários tipos de serviço disponibilizados pela Via Verde

Portagens	
26-07-2015	3
29-07-2015	2
30-07-2015	2
31-07-2015	1
06-08-2015	3
10-08-2015	7
11-08-2015	2
13-08-2015	5
17-08-2015	4
18-08-2015	2
21-08-2015	4
Parques de estacionamento	
30-07-2015	1
06-08-2015	1
















Via Verde  
Parar para quê?

Figura 5.3: Número de entradas em cada dia.

Consultar	Perfil
Número de Entradas	Locais de entrada e respetivo número de visitas, nos vários tipos de serviço disponibilizados pela Via Verde
<b>Locais de Entrada</b>	
Locais de Saída	
Gasto Mensal	
Gasto Diário	
Débito Diário	

Portagens		
	EN 205 PV	6
	Ferreiros	6
	Povoa N-S	5
	Angelas S-N	5
	Braga Sul	2
	Aeroporto	2
	Via Norte E/O	2
	Maia PV	1
	Via Norte O/E	1
	Valongo	1
	EN14 E/O	1
	Ermesinde PV	1
	Maia II	1
















**Via Verde**  
 Parar para quê?

Figura 5.4: Locais de entrada e número de visitas.

Consultar	Perfil
Número de Entradas	Locais de saída e respetivo número de visitas, nos vários tipos de serviço disponibilizados pela Via Verde
Locais de Entrada	
<b>Locais de Saída</b>	
Gasto Mensal	
Gasto Diário	
Débito Diário	

Portagens		
	EN 205 PV	6
	Ferreiros	6
	Angelas N-S	5
	Povoa S-N	5
	Ponte Pedra	3
	Braga Sul	2
	Frelxheiro	2
	Maia PV	1
	Neiva S-N	1
	Valongo	1
	Ermesinde PV	1
	Aeroporto	1
	Maia II	1



**Via Verde**  
 Parar para quê?

Figura 5.5: Locais de saída e número de visitas.

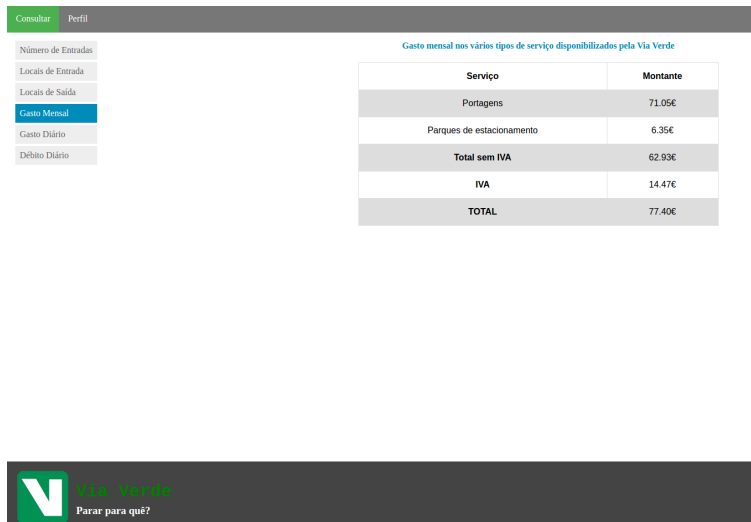


Figura 5.6: Gastos mensais.

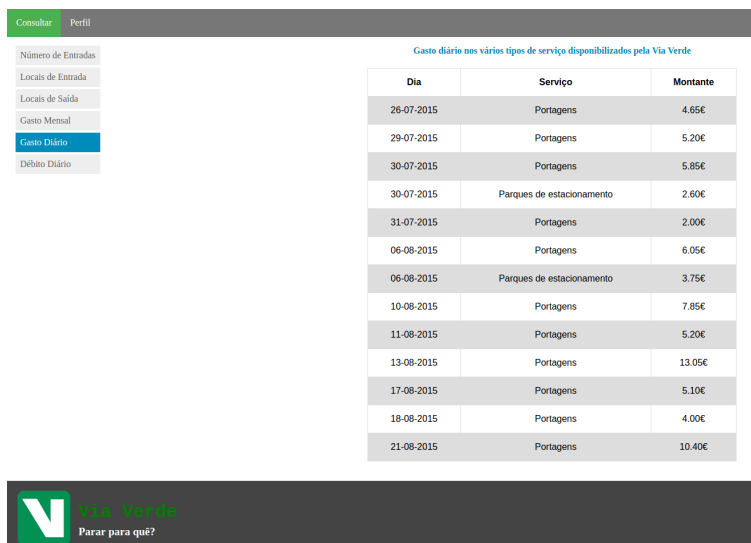


Figura 5.7: Gastos diários.



Consultar	Perfil
Número de Entradas	
Locais de Entrada	
Locais de Saída	
Gasto Mensal	
Gasto Diário	
<b>Débito Diário</b>	

Débitos realizados por dia nos serviços da Via Verde	
Dia	Montante
05-08-2015	4.65€
07-08-2015	15.65€
14-08-2015	13.60€
21-08-2015	23.80€
31-08-2015	19.70€


	<b>Via Verde</b> Parar para quê?
---	-------------------------------------

Figura 5.8: Débitos diários.

## Capítulo 6

# Conclusão

Concluído o projeto, destaca-se a facilidade que o filtro de texto permite na recolha de informações do ficheiro. Após a escolha dos **RS** e **FS** apropriados, tornou-se simples responder às *queries* que nos iam surgindo ao longo do desenvolvimento do projeto. Esta mesma escolha conduziu a uma divisão por campos que, aliada à forma como se procedeu à iteração por estes mesmos campos, acabou por reduzir a necessidade de expressões regulares na filtragem do texto.

Por outro lado, sublinha-se a criação de uma página *HTML* para a apresentação dos dados, pois tratou-se da primeira experiência do grupo com *HTML* e com resultados, a nosso ver, bastante apelativos.

Quanto a trabalho futuro, a interpretação de um ficheiro com extratos de vários meses seria o passo seguinte lógico deste tipo de projeto.