Processamento de Linguagens (3º ano do MiEI)

Trabalho Prático 1

GAWK

Gustavo Andrez (A27748)
Rogério Moreira (A74634)
Samuel Ferreira (A76507)

15 de Março de 2017

Resumo

O presente trabalho tem como objetivo aumentar a experiência no uso do ambiente Linux, aumentar capacidade de escrever Expressões Regulares (ER) e, a partir destas, desenvolver Processadores de Linguagens Regulares.
A ferramenta de processamento de texto utilizada é o GAWK. Todos os objetivos inicialmente propostos foram cumpridos.

Conteúdo

Capítulo 1

Introdução

Introdução

A diversidade e quantidade de informação nos dias de hoje é cada vez maior e mais dispersa, está em todo o lado em grandes quantidades. Posto isto, torna-se ainda mais necessária uma linguagem de programação como o AWK e uma ferramenta como o GAWK, que permitem filtrar, de maneira facilitada, a informação essencial num ficheiro em que os dados estejam dispersos. O trabalho apresentado na unidade curricular de *Processamento de Linguagens* tem como objetivo usar esta ferramenta para filtrar um conjunto de dados fornecido.

Seleção de enunciados

Os temas escolhidos pelo grupo foram:

Processador de transações da Via Verde - A partir de um ficheiro XML com o extrato mensal dos gastos associados a um dispositivo de via verde, por forma a apresentar ao utilizador/utente a informação mais relevante, é necessário filtrar um conjunto de dados disperso como é o caso do ficheiro enviado pela Via Verde, nomeadamente: calcular o número de entradas em cada dia do mês, escrever a lista de locais de saída, calcular o total gasto no mês, calcular o total gasto no mês apenas em parques.

Álbum Fotográfico em HTML - A partir de um ficheiro XML contendo meta-informações sobre as fotografias da coleção do núcleo português do Museu Pessoa que identifica para cada uma destas o ficheiro da foto, o local a data onde foi tirada e, entre outros parâmetros, os intervenientes da imagem. Pretende-se, a partir deste ficheiro, criar um álbum em HTML com as seguintes características: apresentação de uma lista de pessoas fotografadas, para cada elemento da lista permitir a visualização da respetiva foto.

Estrutura do Relatório

Inicialmente vai ser explicado o problema, as características dos dados fornecidos e os padrões de frase por nós encontrados e que levaram à resolução dos problemas inicialmente propostos.
Numa segunda fase apresentamos a conceção da nossa solução e o desenho da resolução, mostrando as estruturas de dados e as ações semânticas. Por fim, apresentamos a codificação em GAWK da solução mostrando as alternativas e as decisões tomadas para os problemas de implementação. Realizamos também alguns testes para verificar a solução encontrada. Em apêndice a este relatório está também todo o código desenvolvido para os dois temas escolhidos.

Capítulo 2

Análise e Especificação

2.1 Características dos Dados e Padrões de Frase

Processador de transações da Via Verde

O ficheiro xml fornecido contém informação, organizada por campos (tags) sinalizados entre os caracteres <e >, que se pode dividir em três partes:
Parte I

- Campos diversos relativos ao detentor da via verde em questão;
- Termina onde começa a primeira transação (<TRANSACCAO >);

Parte II

- Contém informação sobre um conjunto de transações registadas entre os dias 26 de Julho e 21 de Agosto de 2015;
- Os campos que definem cada transação encontram-se entre as tags <TRANSACCAO >e </TRANSACCAO>;

Parte III

- É identificado o total gasto e o valor de iva associado;

Campos que definem uma transação:

1. Data de Entrada - campo vazio em parques e scouts;
2. Hora de Entrada - campo vazio em parques e scouts;
3. Local de Entrada -campo vazio em transações relativas a parques ou scouts;
4. Data de Saída
5. Hora de Saída
6. Local de Saída
7. Custo - valor com iva incluído;
8. Desconto Aplicado - no ficheiro em causa, não há registo de aplicação de descontos(0,00);
9. Valor de Iva - igual a "23" em todos os registos do ficheiro;
10. Entidade Cobradora

11. Tipo - no ficheiro em causa, Portagens ou Parques de estacionamento;
12. Data de Débito
13. Cartão Associado - neste ficheiro existe apenas um cartão associado ao cliente;

Exemplo:

```
<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>
```

Álbum Fotográfico em HTML

O ficheiro xml fornecido contém informação, organizada por campos (tags) sinalizados entre os caracteres <e >, que se pode dividir em duas partes:

Parte I

- Informações sobre a versão e tipo do ficheiro XML;

Parte II

- Informações relativas às fotos a ser exibidas no álbum;
- Os campos que definem cada foto encontram-se entre as tags <foto ficheiro="nome_foto.ext">e </foto >.

Campos que definem uma fotografia: No ficheiro fornecido, todos os campos são opcionais, isto é, não é obrigatório todos os campos abaixo listados aparecem em todas as fotos:

- Onde (onde a foto foi tirada);
- Quando (quando foi tirada a foto);
- Quem (quem aparece na foto);
- Facto (descrição sobre a foto);
- Legenda (legenda associada à fotografia);

Exemplo:

```
<foto ficheiro="022-F-02.jpg">
     <onde>Casa Machado, Afurada</onde>
     <quando data="2000-09-12"/>
     <quem>Ana de Lourdes de Oliveira Chaminé e António Oliveira Machado</quem>
     <facto>António Machado e a sua esposa, dona Ana atrás do balc\~ao da taberna Casa Machado.</facto>
  </foto>
```

Capítulo 3

Decisões, Implementação e Ações Semânticas

Processador de transações da Via Verde

Como já mencionado anteriormente, o ficheiro XML referente à Via Verde está dividido por transações. Cada transação tem um conjunto de campos associado, cada um delimitado por tags e separado do seguinte por um "\n". Assim temos a seguinte numeração dos campos:

- Campo 1 ($1) - <DATA ENTRADA>
- Campo 2 ($2)- <HORA ENTRADA>
- Campo 3 ($3) - <ENTRADA>
- Campo 4 ($4)- <DATA SAIDA>
- Campo 5 ($5) - <HORA SAIDA>
- Campo 6 ($6) - <SAIDA>
- Campo 7 ($7)- <IMPORTANCIA>
- Campo 8 ($8) - <VALOR DESCONTO>
- Campo 9 ($9)- <TAXA IVA>
- Campo 10 ($10) - <OPERADOR>
- Campo 11 ($11)- <TIPO>
- Campo 12 ($12)- <DATA DEBITO>
- Campo 13 ($13)- <CARTAO>

Posto isto, para filtrar a informação de um campo em específico, acompanhado pelas suas tags (<tipo de dados >DADOS </tipo de dados >) , foram criadas funções auxiliares que recorrem à função split. Todas as funções auxiliares estão descritas no ficheiro gets.gawk. No ficheiro tp1.gawk os dados são obtidos através de funções da biblioteca gets.gawk, de maneira a cumprir os objetivos definidos mantendo o código curto e legível.
Optou-se por definir algumas funções auxiliares num ficheiro distinto, pelo que foi necessário importá-lo no código principal do programa.

```
@include "gets.gawk"
```

Inicialmente começou-se por definir os record e field separators. Por forma a que, durante a execução do programa, o valor do campo $1 corresponda ao primeiro campo útil (data de entrada). Feita esta decisão, definimos os separators da maneira a seguir apresentada.

```
BEGIN {
    RS="<TRANSACCAO>[ \t\n]*";
    FS="\n";
}
```

Assim, para trabalhar sobre cada registo ciclicamente, fez-se: A variável soma conta o número de transações. Alternativamente,no fim do programa, podia ser escrito o valor de NR-1.

```
     NR>1 {
    soma ++;
```

Para calcular o número de 'entradas' em cada dia do mês, como estrutura de dados, usámos um array cujos índices são strings (datas no formato DD-MM) e os elementos são inteiros. Para todas as transações, o elemento do array que corresponde à data de saída é incrementado.

```
     entradas[getDia($1)]++;
```

Para registar os locais de 'saída' de cada transação, como estrutura de dados, usámos um array cujos índices são strings (correspondentes aos locais de saída) e os elementos são inteiros. Para todas as transações, o elemento do array que corresponde ao local de saída é colocado com valor 1. Usando os locais de saída como índices garantimos que não haverá repetições neste array.

```
     saida = getSaida($6);
    if(!saidas[saida])
        saidas[arr[3]]=1;
```

Para calcular o total gasto no mês, foi criada a variável 'preco' para armazenar o custo da transação (campo 7) e a variável 'desconto' para registar o desconto correspondente (campo 8). A diferença destes valores é adicionada a uma terceira variável 'totalGasto', que irá guardar o somatório dos valores pagos em todas as transações.

```
     preco = getImportancia($7);
    desconto = getDesconto($8);
    totalGasto += (preco-desconto);
```

Para responder às questões relativas a parques de estacionamento, foi aplicada uma condição "if" que testa se a transação corrente corresponde a um parque de estacionamento, tendo sido as instruções relativas a estas questões colocadas dentro da condição. Assim, apenas se o tipo de transação corresponder a 'Parques de estacionamento' (campo 11) , serão executadas as instruções.

```
     tipo=getTipo($11);
    if( tipo ~ /Parques de estacionamento/ ){
```

Para calcular o total gasto no mês apenas em parques, foi criada a variável 'totalParques' que irá guardar o somatório do valor efetivamente pago em cada transação do tipo 'parques de estacionamento'. A diferença entre as variáveis 'preço' e 'desconto' será adicionada à variável 'totalParques'.

```
     totalParques += (preco-desconto);
```

Para cada transação, o operador do parque estacionamento (campo 10) é armazenado temporariamente na variável 'operador' e no array 'locaisParques'. da mesma forma, a data de saida (campo 4) é armazenado na variável 'dataSaida' e no array 'datasParques'. Para controlo dos índices dos arrays 'locaisParques' e 'datasParques' foi criada a variável 'contaParques', que é incrementada a cada nova transação, tendo o mesmo índice de cada array informação relativa à mesma transação.

```
     contaParques++;
        operador=getOperador($10);
        locaisParques[contaParques]=operador;
        dataSaida=getDataSaida($4);
        datasParques[contaParques]=dataSaida;
}
```

Foi criada a variável 'cartao' que irá guardar o número do cartão associado à transação. Para calcular o número de transações associados a cada cartão, foi criado o array 'cartoes' ao qual, a cada transação, é incrementado o elemento do índice 'cartao'.

```
     cartao=getCartao($13);
    cartoes[cartao]++;
}
```

No final da execução do programa, são imprimidos no terminal os resultados acumulados das operações aplicadas aos diversos campos das transações.

```
     END {
    print toupper("Total de transações: ") soma;
    print "TOTAL GASTO: " totalGasto;
    print "TOTAL GASTO EM PARQUES DE ESTACIONAMENTO: " totalParques;

    print toupper("numero de 'entradas' em cada dia do mes:");
    for( i in entradas )
        print "\tdia " i " : " entradas[i];


    print "\nLOCAIS DE SAIDA:";
    for( i in saidas )
        print "\t" i;

    print "\nCARTÕES UTILIZADOS:";
    for( i in cartoes )
        print "\tCartão " i " : " cartoes[i];

    print "\nDATAS E LOCAIS DE ESTACIONAMENTO:"
    for( i in locaisParques )
        print "\t" datasParques[i] " - " locaisParques[i];

}
```

Álbum Fotográfico em HTML

À semelhança do exercício anterior, neste exercício os dados também estão separados por linhas, delimitadas pela tag foto. Cada foto tem um conjunto de atributos a si associado. Assim temos a seguinte numeração dos campos:

- Campo 1 - Onde
- Campo 2 - Quando
- Campo 3 - Quem
- Campo 4 - Facto
- Campo 5 - Legenda

Para filtrar um dado em específico, acompanhado pelas suas tags (<tipo de dados >DADOS </tipo de dados >) foram criadas funções auxiliares que estão descritas no ficheiro funcs.gawk. No ficheiro tp1FOTOS.gawk os dados são extraídos através de funções da biblioteca gets.gawk, de maneira a cumprir os objetivos definidos mantendo o código curto e legível.

Optou-se por definir algumas funções auxiliares num ficheiro distinto, pelo que foi necessário importá-lo no código principal do programa.

```
@include "funcs.gawk"
```

Inicialmente começou-se por definir os record e field separators.
O field separator foi definido desta maneira devido ao facto que existirem caracteres "\n"no conteúdo dos campos. Caso contrário o field separator poderia ser apenas "\n".

```
BEGIN {
RS="<foto ";
FS=">[ ]*\n";
```

Foram criadas variáveis para guardar os títulos a apresentar nas páginas HtML de maneira a que estes sejam mais facilmente personalizáveis.

```
headerTitle="Processamento de Linguagens 16/17 TP1 Grupo XX";
bodyTitle="Álbum Fotográfico em HTML";
headerFormat= "<html><head><meta charset = 'UTF-8'/><title>%s</title></head><body><center><h1>%s<
item="<center><li><b><a href=\"%s\">%s</a></b></li></center>\n";
desc="<center><p>%s , %s</p><p>%s</p><img src=\"fotos/%s\" width=500><p>%s</p></center>\n";
```

A instrução seguinte escreve o cabeçalho da página index.html para um ficheiro com o mesmo nome.

```
printf headerFormat , headerTitle , bodyTitle > "index.html";
}
```

Uma das dificuldades encontradas nesta parte foi o facto de a informação do ficheiro nem sempre ter a mesma sintaxe. Como mencionado anteriormente, os campos que compõem cada transação podem diferir em quantidade e ordem. Por este motivo, o programa foi concebido para avaliar cada FIELD (campo) constituinte de cada RECORD (transação) através de uma expressão regular, de maneira a identificar que informação é que consta no campo em questão.
Assim, para trabalhar sobre cada registo ciclicamente, fez-se:

```
NR>1 {
```

Foram criadas variáveis para armazenar a informação sobre os campos necessários para a análise de cada fotografia. Dado que estas variáveis são reutilizadas em cada iteração, há a necessidade de, no início de cada iteração, inicializá-las com uma string vazia para que não seja transportada informação de uma iteração para a seguinte.

```
onde=quem=facto=quando=foto=legenda="";
```

É guardada em duas variáveis a designação da fotografia em questão (nome_foto.ext)

```
foto=foto2=getCampo(\1);
```

Através do uso da função sub, remove-se a extensão do ficheiro de modo a ficar apenas com o nome da fotografia na variável 'foto2'.

```
sub( /\.[a-zA-Z]+/ ,"",foto2);
```

São percorridos todos os campos do registo atual e, para cada campo, são testadas as condições necessárias para identificar que tipo de informação contém o campo em questão para que se possa processar essa informação da maneira adequada, uma vez que a ordem e quantidade de campos variam no ficheiro em questão.

```
for( i=2 ; i<=NF ; i++ ){
      if($i~/<quem/){
        quem=trim( getCampoDefault($i) );
}
```

Adicionalmente, as strings relativas ao campo 'onde' são armazenadas no array 'locais' criado para o efeito. Antes de serem inseridas no array, é feita a verificação se já existe essa informação de modo a que não haja duplicação de locais.

```
if($i~/<onde/){
     onde=trim( getCampoDefault($i) );
     if( pertence( locais , onde ) == 0 )
        locais[locLength++]=onde;
}
if($i~/<quando/){
    quando= getCampo($i);
}
if($i~/<facto/){
    facto=trim( getCampoDefault($i) );
}
if($i~/<legenda/){
    legenda=trim( getCampoDefault($i) );
}
}
```

Nesta altura seguem-se as instruções que escrevem a informação relativa a uma página html que apresentará informação sobre a foto correspondente à iteração atual.

```
printf headerFormat ,onde , quem > foto2".html";
        printf desc, onde , quando , facto , foto , legenda > foto2".html";
        printf item , foto2".html" , quem > "index.html";
}
```

No final da execução do programa, depois de o array 'locais' estar devidamente preenchido, o programa imprime os elementos deste array no terminal e imprime uma string para o ficheiro 'index.html' que corresponde ao fecho de algumas tags HTML.

```
END {
        for(x in locais)
            print locais[x] ;
        print "</ul></body></html>" > "index.html";
    }
```

3.1 Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos.

Processador de transações da Via Verde

```
rgllm:via-verde rgllm$ gawk -f tp1.gawk viaverde.xml
TOTAL DE TRANSAÇÕES: 41
TOTAL GASTO: 77.4
TOTAL GASTO EM PARQUES DE ESTACIONAMENTO: 6.35
NUMERO DE 'ENTRADAS' EM CADA DIA DO MES:
        dia 31-07 : 1
        dia 29-07 : 2
        dia 17-08 : 4
        dia 13-08 : 5
        dia 06-08 : 4
        dia 18-08 : 2
        dia 21-08 : 4
        dia nao especificado : 4
        dia 26-07 : 3
        dia 10-08 : 7
        dia 11-08 : 2
        dia 30-07 : 3

LOCAIS DE SAIDA:
        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

CARTÕES UTILIZADOS:
        Cartão 6749036 : 41

DATAS E LOCAIS DE ESTACIONAMENTO:
        30-07-2015 - BRAGAPARQUES (BP)
        06-08-2015 - ANA - Aeroportos de Portugal. SA (AP)
```

Ao analisar um ficheiro XML com a estrutura da via verde são obtidos alguns dados interessantes para o utilizador recorrendo ao programa desenvolvido. O programa calcula o número total de transações, o total gasto e o total gasto só em parques de estacionamento, para além de indicar o número de entradas em cada mês, os locais de saída, os cartões utilizados e as datas e os locais de estacionamento.

Álbum Fotográfico em HTML

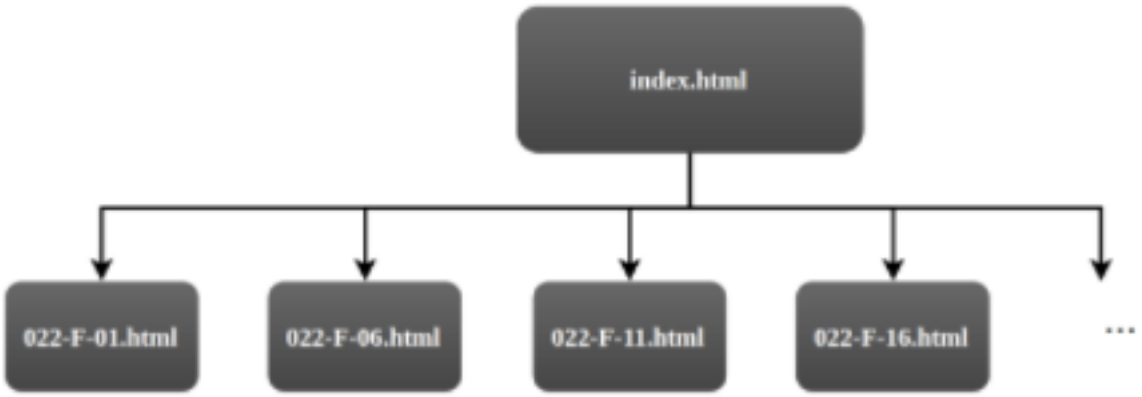
```
rgllm:galeria rgllm$ gawk -f tp1FOTOS.gawk legenda.xml
Casa Machado, Afurada
Colégio Nossa Senhora de Lourdes, Porto
Na praia de Nazaré
Igreja Santa Marinha Afurada
Taberna Neca Chaminé, Afurada
Taberna do Fausto, na Afurada
Monte de Santa Luzia, Viana do Castelo
```

Álbum Fotográfico em HTML

Eugénia Maria de Oliveira Chaminé Machado e uma Irmã do Colégio Nossa Senhora de Lourdes, no Porto.

Colégio Nossa Senhora de Lourdes, Porto , 1969-06-05

Primeira comunhão de Eugénia Maria de Oliveira Chaminé Machado.



Quando o programa é executado, os resultados são apresentados de duas maneiras distintas:

- No terminal/bash são listados todos os locais fotografados, sem repetições;
- Por outro lado são também gerados os ficheiros index.html e um ficheiro nome-da-foto.html para cada uma das fotos. O ficheiro index.html apresenta uma lista das fotos disponíveis com o respetivo apontador para o ficheiro da foto, onde são mostrados todos os dados relativo aquela foto.

Capítulo 4

Conclusão

Durante todo o desenvolvimento dos dois programas, tentámos que o código se aplicasse a qualquer ficheiro com o mesmo formato e fosse o mais compacto possível. Contudo, os programas desenvolvidos não cobrem algumas exceções e por se tratar de um trabalho académico não têm algumas verificações que porventura num programa em produção seriam necessárias.
No geral, os objetivos esperados para o programa foram alcançados e obtemos uma solução sólida para o problema apresentado. Destacamos ainda alguns pontos:

- No trabalho do álbum em HTML foi gerado um ficheiro para cada foto e um ficheiro central com apontadores para estes em alternativa a haver apenas um ficheiro com toda a informação das fotos.
- No trabalho da via verde foram filtrados mais dados do que aqueles inicialmente previstos no guião. Dos quais destacamos: listagem das datas e locais de estacionamento, os cartões utilizados e o total de transações.
- Destacamos também que funções auxiliares para cada um dos projetos são codificadas em ficheiros aparte (ficheiro gets.gawk e ficheiro funcs.gawk) para garantir a uniformização do código.

Como trabalho futuro sugerimos:

- Mais verificações para garantir que não ocorrem erros de leitura;
- Tratamento das legendas das fotos com o parâmetro “alt”.
- Navegação entre fotografias.

Apêndice A

Código GAWK

Lista-se a seguir o código desenvolvido.
Processador de transações da Via Verde
Ficheiro gets.gawk

```
# getCampoStandard( campo ) -> devolve a apenas a informação útil de uma string com um campo ( incluindo tags
# usada para obter as informações delimitadas por > e <
function getCampoStandard( campo ){
    split( campo , arr , /[><]/ );
    return arr[3];
}
# getMonetario( campo ) -> devolve a apenas a informação útil de uma string com o campo 8 ou 9 ( incluindo ta
# usada para obter valores numericos que vão ser usados para cálculos
function getMonetario( campo ){
    split(campo,arr,/[><]/);
    sub(",",".",arr[3]);
    return arr[3];
}
# getDia( campo ) -> devolve a apenas a informação útil de uma string com o campo 2 de uma transacao ( inclui
# usada para obter uma string com a data no formato DD-MM
function getDia( campo2 ){
    split(campo2,arr,">");
    split(arr[2],arr,"<");
    split(arr[1],arr,"-");
    if(arr[1]!="null")
    #          dia - mes
        dia=arr[1]"-"arr[2];
    else dia="nao especificado";
    return dia;
}
function getDataSaida( campo5 ){return getCampoStandard( campo5 );}
function getSaida( campo7 ){return getCampoStandard( campo7 );}
# getTipo -> Portagens ou Parques de estacionamento
function getTipo( campo12 ){return getCampoStandard( campo12 );}
function getCartao( campo14 ){return getCampoStandard( campo14 );}
# getOperador -> "nome" do parque de estacionamento
function getOperador( campo11 ){return getCampoStandard( campo11 );}
function getImportancia( campo8 ){return getMonetario( campo8 );}
function getDesconto( campo9 ){return getMonetario( campo9 );}
```

Ficheiro tp1.gawk

```
@include "gets.gawk"
BEGIN {
    RS="<TRANSACCAO>[ \t\n]*";
    FS="\n";
}
NR>1 {
    soma ++;
    # calcular o numero de 'entradas' em cada dia do mes
    entradas[getDia($1)]++;
    # escrever a lista de locais de 'saida' ( sem repeticoes )
    saida = getSaida($6);
    if(!saidas[saida])
        saidas[arr[3]]=1;
    # calcular o total gasto no mes
    preco = getImportancia($7);
    desconto = getDesconto($8);
    totalGasto += (preco-desconto);
    # INICIO - calcular o total gasto no mes apenas em 'parques'
    tipo=getTipo($11);
    if( tipo ~ /Parques de estacionamento/ ){
        totalParques += (preco-desconto);
        # INICIO - escrever a lista de parques de estacionamento e as datas em que estes foram utilizados
        contaParques++;
        operador=getOperador($10);
        locaisParques[contaParques]=operador;
        dataSaida=getDataSaida($4);
        datasParques[contaParques]=dataSaida;
        # FIM - escrever a lista de parques de estacionamento e as datas em que estes foram utilizados
    }
    # FIM - calcular o total gasto no mes apenas em 'parques'
    # calcular o numero de transacoes associadas cada cartao
    cartao=getCartao($13);
    cartoes[cartao]++;
}
END {
    print toupper("Total de transações: ") soma;
    print "TOTAL GASTO: " totalGasto;
    print "TOTAL GASTO EM PARQUES DE ESTACIONAMENTO: " totalParques;

    print toupper("numero de 'entradas' em cada dia do mes:");
    for( i in entradas )
        print "\tdia " i " : " entradas[i];


    print "\nLOCAIS DE SAIDA:";
    for( i in saidas )
        print "\t" i;

    print "\nCARTÕES UTILIZADOS:";
    for( i in cartoes )
        print "\tCartão " i " : " cartoes[i];

    print "\nDATAS E LOCAIS DE ESTACIONAMENTO:"
```

```
    for( i in locaisParques )
        print "\t" datasParques[i] " - " locaisParques[i];

}
```

Álbum Fotográfico em HTML

Ficheiro funcs.gawk

```
# trim( s ) -> remove espaços em branco nas extremidades de uma string
function trim( s ) {
    sub(/^[ \t\r\n]+/, "", s);
    sub(/[ \t\r\n]+$/, "", s);
    return s;
}

# pertence( arr , s ) -> retorna 1 se existir o valor s no array arr; retorna 0 caso contrário
function pertence( arr , s ){
    for( x in arr ){
        if( arr[x] == s )
            return 1;
    }
    return 0;
}

# getCampoDefault( campo ) -> devolve a apenas a informação útil de uma string com um campo ( incluindo tags
# usada para obter as informações delimitadas por > e <
function getCampoDefault( campo ){
    split( campo , arr , "[><]" );
    return arr[3];
}

# getCampo( campo ) -> devolve a apenas a informação útil de uma string com um campo ( incluindo tags )
# usada para obter as informações delimitadas por aspas
function getCampo( campo ){
    split( campo , arr , "\"" );
    return arr[2];
}
```

Ficheiro tp1FOTOS.gawk

```
@include "funcs.gawk"
BEGIN {
        RS="<foto ";
        FS=">[ ]*\n";
        headerTitle="Processamento de Linguagens 16/17 TP1 Grupo XX";
        bodyTitle="Álbum Fotográfico em HTML";
        headerFormat= "<html><head><meta charset = 'UTF-8'/><title>%s</title></head><body><center><h1>%s</h1>
        item="<center><li><b><a href=\"%s\">%s</a></b></li></center>\n";
        desc="<center><p>%s , %s</p><p>%s</p><img src=\"fotos/%s\" width=500><p>%s</p></center>\n";
        printf headerFormat , headerTitle , bodyTitle > "index.html";
    }
NR>1 {
        onde=quem=facto=quando=foto=legenda="";
        foto=foto2=getCampo($1);
        sub( /\.[a-zA-Z]+/ ,"",foto2);
        for( i=2 ; i<=NF ; i++ ){
            if($i~/<quem/){
                 quem=trim( getCampoDefault($i) );
            }
            if($i~/<onde/){
                 onde=trim( getCampoDefault($i) );
                 if( pertence( locais , onde ) == 0 )
                    locais[locLength++]=onde;
            }
            if($i~/<quando/){
                 quando= getCampo($i);
            }
            if($i~/<facto/){
                facto=trim( getCampoDefault($i) );
            }
            if($i~/<legenda/){
                legenda=trim( getCampoDefault($i) );
            }
        }
        printf headerFormat ,onde , quem > foto2".html";
        printf desc, onde , quando , facto , foto , legenda > foto2".html";
        printf item , foto2".html" , quem > "index.html";
    }
END {
        for(x in locais)
            print locais[x] ;
        print "</ul></body></html>" > "index.html";
    }
```