Processamento de Linguagens
(3º ano de Licenciatura em Engenharia Informática)

Trabalho Prático 1

Relatório de Desenvolvimento

André Gerlades (67673) Patrícia Barros (67665)
Sandra Ferreira (67709)

1 de Abril de 2015

Resumo

Este relatório descreve todo o processo de desenvolvimento e decisões tomadas para a realização do primeiro trabalho prático da Unidade Curricular de Processamento de Linguagens.
O problema a resolver consiste no desenvolvimento de um Filtro de Texto, utilizando Flex, para processar ficheiros XML com informações sobre fotografias e gerar um álbum HTML a partir delas.

Conteúdo

Capítulo 1

Introdução

A resolução deste trabalho prático passa pelo desenvolvimento de um Filtro de Texto em Flex para gerar ficheiros em HTML. Para isso utilizamos as técnicas leccionadas nas aulas da Unidade Curricular de Processamento de Linguagens. Pretendemos portanto com este trabalho aprimorar as nossas capacidades de escrever *Expressões Regulares (ER)* e também a nossa experiência na utilização da linguagem de programação C.
Neste relatório apresentamos todos os passos e decisões tomadas durante todo o processo, descrevemos as estruturas criadas para guardar o texto extraído pelo filtro e também uma descrição do produto final (em HTML) obtido com a utilização do filtro criado por nós.

Estrutura do Relatório

A elaboração deste documento teve por base a estrutura do relatório fornecida pelo docente.
O relatório encontra-se então estruturado da seguinte forma: possuí um primeiro capítulo que faz uma contextualização ao assunto tratado neste trabalho, seguindo-se uma introdução onde são apresentadas as metas de aprendizagem pertendidas.
Posteriormente é exposto o tema escolhido para desenvolver o trabalho e as tarefas que nele estão envolvidas.
Imediatamente após, são exibidas as expressões regulares definidas para extrair do ficheiro XML as informações para a construção da página HTML, mostrando também os estados da aplicação e os módulos desta.
Não menos importante, seguem-se os capitulos de apresentação das estruturas de dados usadas no desenvolvimento do trabalho e dos testes realizados à aplicação com os devidos resultados. Por último, faz-se uma análise crítica relativa quer ao desenvolvimento do projeto quer ao seu estado final e ainda é feita uma abordagem ao trabalho futuro.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Dos enunciados propostos o que escolhemos seguir foi o "Museu da Pessoa - tratamento de fotografias".
Neste enunciado foi-nos pedido que realizassemos um Filtro de Texto em Flex para, através de um ficheiro XML com informações (onde,quando,quem e facto) sobre fotografias que fazem parte do Museu da Pessoa, gerar um álbum em HTML em que sejam mostradas as fotografias em questão, por ordem cronológica, sendo que o título de cada fotografia será o campo "facto". Foi-nos ainda pedido que apresentássemos um índice no ínicio com o nome de todas as pessoas retratadas.

Capítulo 3

Concepção/desenho da Resolução

Para a resolução do problema foi necessário definirmos *Expressões Regulares*, estados no Flex e ainda estruturas de dados para guardar a informação filtrada. Nos próximos capítulos iremos descrever detalhadamente cada um destes pontos.

3.1 Expressões Regulares

As expressões regulares definidas visam extrair do ficheiro XML as informações necessárias para a construção da página HTML: foto, quando, onde, quem e facto. Após analisar cuidadosamente a estrutura dos ficheiros XML em questão chegamos às seguintes *Expressões Regulares*:
QUEM ≪"(?i:QUEM)≫"
FACTO ≪"(?i:FACTO)≫"
FOTO ≪"(?i:FOTO)(?i:FICHEIRO)"=¨
QUANDO ≪"(?i:QUANDO)(?i:DATA)"=¨

3.2 Estados da Aplicação

Estado QUEM
Quando é encontarada a palavra "QUEM"são guardadas as palavras seguintes, com o devido tratamento dado pela função trim definida, uma vez que estas serão referentes a quem pertence a foto.

```
"<"(?i:QUEM)">"  { BEGIN QUEM1; }
<QUEM1>"<"   { BEGIN INITIAL; }
<QUEM1>[^<]+ { quem = strdup(yytext); quem = trim(quem); }
```

Estado FACTO
Quando é identificada a expressão "FACTO"é guardada a informação que a esta se segue, através da função *strdup*.

```
"<"(?i:FACTO)">" { BEGIN FACTO1; }
<FACTO1>"<"   { BEGIN INITIAL; }
<FACTO1>[^<]+   { facto = strdup(yytext); }
```

Estado QUANDO
Aquando da análise léxica, se for detetada a palavra "QUANDO"a data a esta referente é guardada. Posteriormente é chamada a função breakFoto que recebe como parâmetro a string guardada e converte-a num tipo de dados *struct data*.

```
"<"(?i:QUANDO)" "(?i:DATA)"=\"" { BEGIN QUANDO1; }
<QUANDO1>"\"" { BEGIN INITIAL; }
<QUANDO1>[^\"]+  { quando = strdup(yytext);
da = breakFoto(quando); }
```

Estado FOTO
Encontrada a palavra "FOTO"é criado um novo nodo na lista ligada que contêm a informação referentes a outras fotos. Depois de inserido o novo nodo, a lista é ordenada pela data das fotos.

```
"<"(?i:FOTO)" "(?i:FICHEIRO)"=\"" { BEGIN FOTO1; }
<FOTO1>"\""   { BEGIN INITIAL; }
<FOTO1>[^\"]+ { foto = strdup(yytext);
nodo = novoNodo(da, foto, quem, facto);
sortedInsert(&dados, nodo);
}
```

3.3 Módulos da Aplicação

makefile Ficheiro com a configuração de compilação.
parser.l Ficheiro com código necessário para fazer o processamento dos ficheiros XML que contêm a informação.
listaligada.h Ficheiro que contém o código necessário à implementação de uma lista ligada e das funções necessárias para a sua manipulação.
dados.h Ficheiro onde se encontra a estrutura definida para guardar a data das fotos.
html.h Ficheiro com o código que gera as páginas em HTML.

3.4 Estruturas de Dados

Dado ser necessário guradar alguns dados, a estutura de dados escolhida foi a lista ligada. Esta estrutura possui um campo para guardar a data da fotografia,

o facto, breve descrição da foto, o nome do ficheiro da foto e por último, um campo quem, que se destina a guardar os nomes das pessoas retratadas.

```
1  struct data {
2          int ano;
3          int mes;
4          int dia:
5  };
6
7  struct listaLigada {
8          struct data datay;
9          char *nome;
10         char *quem;
11         char *fato;
12         struct listaLigada * next;
13 };
```

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

No decorrer do desenvolvimento do projeto verificamos que nem todas as fotos eram possuidoras do campo "FACTO", campo este que deveria ser o cabeçalho das fotos. Embora tenha sido pedido no enunciado que utilizassemos este como cabeçalho o grupo decidiu que seria melhor se ao invés se colocasse o campo "QUEM"como cabeçalho.

4.2 Testes realizados e Resultados

São apresentadas de seguida, imagens que mostram o resultado do filtro de texto elaborado pelo grupo para os diversos casos.

Figura 4.1: Resultado do filtro de testo para o caso do António Machado

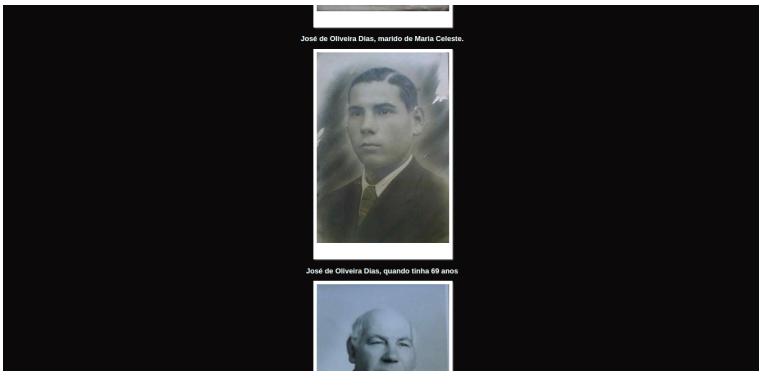

Figura 4.2: Resultado do filtro de testo para o caso do José de Oliveira Dias

Figura 4.3: Resultado do filtro de testo para o caso da Maria Celeste

Figura 4.4: Resultado do filtro de testo para o caso Taberna do Fausto

Capítulo 5

Conclusão

A realização deste projeto foi fundamental para se consolidar a matéria leccionada quer nas aulas práticas quer nas aula teóricas, uma vez que as técnicas de utilização de expressões regulares aí aprendidas facilitou a implementação do problema. Estas técnicas pertimitiram recolher de forma eficiente e clara as informações pertendidas.
Foi ainda possível ao grupo aprender a trabalhar com o básico da linguagem HTML que nunca tinha sido utilizada por qualquer dos elementos. O projeto "Museu da Pessoa - tratamento de fotografias"foi muito enriquecedor, dado que obrigou a dedicar um porção de tempo e a refinar as qualidades do grupo na utilização do Flex e na construção das expressões regulares.

Apêndice A

Código Flex

```
1  %{
2  /* Declaracoes C diversas */
3  #include <stdio.h>
4  #include <string.h>
5  #include "data.h"
6  #include "listaligada.h"
7  #include "html.h"
8  char * foto;
9  char * facto;
10 char * quem;
11 char * quando;
12 int k = 0;
13 struct data da;
14 struct listaLigada * dados;
15 struct listaLigada * nodo;
16
17 /* FunÃ§Ã£o que verifica se um determinado char Ã© um digito*/
18 int digito(char d){
19         int x = 0;
20         if (d >= '0' && d <= '9') x = 1;
21         return x;
22 }
23
24 /* FunÃ§ao que converte uma string num tipo de dados struct
       data */
25 struct data breakFoto (char * d){
26         struct data new;
27         char ano[5], mes[3], dia[3];
28         if(!digito(quando[0])) {
29                 new.ano = 1;
30                 new.mes = 1;
31                 new.dia = 1;
32                 return new;
```

```
33          }
34
35          int i = 0, j = 0;
36          for(i=0; i < 4; i++) ano[i] = d[j++];
37          ano[i] = '\0';
38          j++;
39          for(i=0; i < 2 ; i++) mes[i] = d[j++];
40          j++;
41          mes[i] = '\0';
42          for(i=0; i < 2 ; i++) dia[i] = d[j++];
43          dia[i] = '\0';
44          new.ano = atoi(ano);
45          new.mes = atoi(mes);
46          new.dia = atoi(dia);
47          return new;
48  }
49
50  /* FunÃ§Ã£o para remover tabs e espaÃ§os seguidos de uma
        string */
51  char * trim(char * q){
52          int i, j;
53          for(i=0; q[i] != '\0'; i++){
54                  if(q[i] == '\n' || q[i] == '\t') q[i] = ' ';
55                  if((q[i] == ' ') && (q[i+1] == ' ')){
56                          for(j=i; q[j] != '\0'; j++) q[j] = q[j
                                +1];
57                  }
58          }
59          return q;
60  }
61
62  %}
63
64
65  %x QUEM1
66  %x FACTO1
67  %x FOTO1
68  %x QUANDO1
69
70  %%
71  "<"(?i:QUEM)">"                                                 {
        BEGIN QUEM1; }
72  <QUEM1>"<"
                { BEGIN INITIAL; }
73  <QUEM1>[^<]+                                                    { quem
        = strdup(yytext); quem = trim(quem); k++; }
74
75  "<"(?i:FACTO)">"                                                {
        BEGIN FACTO1; }
```

```
76 <FACTO1>"<"
                { BEGIN INITIAL; }
77 <FACTO1>[^<]+                                                  {
       facto = strdup(yytext); }
78
79 "<"(?i:QUANDO)" "(?i:DATA)"=\""            { BEGIN QUANDO1; }
80 <QUANDO1>"\""                                                  {
       BEGIN INITIAL; }
81 <QUANDO1>[^\"]+                                                {
       quando = strdup(yytext);
82                                                                                                  da
                                                                                                        =
                                                                                                        breakFoto
                                                                                                        (
                                                                                                        quando
                                                                                                        )
                                                                                                        ;
                                                                                                        }
83
84 "<"(?i:FOTO)" "(?i:FICHEIRO)"=\""          { BEGIN FOTO1; }
85 <FOTO1>"\""
                { BEGIN INITIAL; }
86 <FOTO1>[^\"]+                                                { foto
        = strdup(yytext); }
87
88 "</foto>"
                {       nodo = novoNodo(da, foto, quem, facto);
89                                                                                                      sortedI
                                                                                                        (&
                                                                                                        dad
                                                                                                        ,
                                                                                                        nod
                                                                                                        )
                                                                                                        ;
                                                                                                        }
90
91
92 .|\n
                { ; }
93 %%
94
95 int yywrap()
96 {
```

```
97          return(1);
98  }
99
100 int main()
101 {
102         FILE * html;
103         html = fopen("museu.html","w");
104         newHeader("Museu da Pessoa", html);
105         yylex();
106         insertImg(dados, html);
107         endHtml(html);
108         fclose(html);
109         return 0;
110 }
```

Apêndice B

Código da Lista Ligada

```
1 #include "listaligada.h"
2
3
4 void sortedInsert(struct listaLigada** head_ref, struct
      listaLigada* new_node)
5 {
6     struct listaLigada* atual;
7
8     if (*head_ref == NULL || (comparaDatas((*head_ref)->datay,
          new_node->datay) == 1))
9     {
10        new_node->next = *head_ref;
11        *head_ref = new_node;
12    }
13    else
14    {
15        /* Encontrar o nodo antes de inserir */
16        atual = *head_ref;
17        while (atual->next != NULL &&
18               (comparaDatas(atual->next->datay, new_node->
                   datay) != 1))
19        {
20            atual = atual->next;
21        }
22        new_node->next = atual->next;
23        atual->next = new_node;
24    }
25 }
26
27 /*FunÃ§Ã£o que cria um novo nodo de listaligada */
28 struct listaLigada *novoNodo(struct data d, char * n, char * q
      , char * f){
```

```
29      struct listaLigada * new = (struct listaLigada *) malloc(
            sizeof(struct listaLigada));
30          new->datay = d;
31          new->nome = n;
32          new->quem = q;
33          new->fato = f;
34          new->next = NULL;
35
36          return new;
37  }
38
39
40  /* FunÃ§Ã£o que compara duas datas, retorna 1 se a data1 for
        mais recente **/
41  int comparaDatas(struct data data1, struct data data2){
42          if(data1.ano > data2.ano)return 1;
43          else if(data1.ano < data2.ano) return -1;
44          else {
45                  if(data1.mes > data2.mes)return 1;
46                  else if(data1.mes < data2.mes)return -1;
47                  else {
48                          if(data1.dia > data2.dia)return 1;
49                          else if(data1.dia <= data2.dia) return
                                -1;
50                  }
51      }
52      return -1;
53  }
54
55  /* FunÃ§Ã£o para utilizaÃ§Ã£o do qsort */
56  int compare (const void * a, const void * b)
57  {
58      return strcmp (*(const char **) a, *(const char **) b);
59  }
60
61  /* FunÃ§Ã£o que retira os nomes da lista ligada, ordena e
        guarda num array de strings */
62  char ** getNames(struct listaLigada* l, int k){
63          int maior = 0;
64          int i,h;
65          struct listaLigada * n = l;
66          while(n != NULL){
67                  if(strlen(n->quem) > maior) maior = strlen(n->
                        quem);
68                  n = n->next;
69          }
70          char ** new;
71          new = malloc(k * maior * sizeof(char*));
72          for(i = 0; i < k; i++) {
```

```
73                  new[i] = malloc(maior* sizeof(l->quem));
74                  strcpy(new[i],l->quem);
75                  if(new[i][0] == ' ') {
76                          for(h = 0; h < strlen(new[i]); h++)
                                new[i][h] = new[i][h+1];
77                  }
78                  l = l->next;
79          }
80          qsort(new, k, sizeof(const char *), compare);
81          return new;
82  }
```

Apêndice C

Código para gerar o HTML

```
1 #include "html.h"
2
3 void newHeader(char *tit, FILE *fp) {
4         fputs("<!DOCTYPE html>\n",fp);
5         fputs("<html>\n",fp);
6         fputs("<meta charset=utf-8>\n",fp);
7         fputs("<link rel=stylesheet type=text/css href=mystyle
              .css>\n",fp);
8         fprintf(fp,"<body bgcolor= #0C090A>\n");
9         fprintf(fp, "<title>%s</title>\n", tit);
10        fputs("<font color=##95B9C7><font face=arial><font
              size=5><h1><center>Museu da Pessoa</center></h1></
              font></font></font>\n",fp);
11        fputs("</head>\n",fp);
12 }
13
14 void endHtml(FILE * fp) {
15        fputs("</html>\n",fp);
16 }
17
18 void addImg(char * nomeImg, char * descricao, FILE *fp){
19        fputs("<body>\n",fp);
20        fprintf(fp, "<center><div class=polaroid><font face=
              arial><font size=2><font color=#F0FFFF><h2>%s</h2
              ></font></font></font>\n", descricao);
21        fprintf(fp, "<img src=\"%s\" alt=\"%s\" style=\"width
              :354px;height:508px\"><br></div></center>\n",
              nomeImg, descricao);
22        fputs("</body>\n",fp);
23 }
24
25 void insertImg(struct listaLigada * a, FILE *fp) {
26        struct listaLigada* temp = a;
```

```
27      while(temp != NULL)
28      {
29          addImg(temp->nome, temp->quem, fp);
30          temp = temp->next;
31      }
32  }
```