

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

1	Introdução	2
2	Análise e Especificação	3
2.1	Descrição informal do problema	3
2.2	Especificação do Requisitos	3
2.2.1	Dados	3
2.2.2	Pedidos	3
2.2.3	Relações	3
3	Concepção/desenho da Resolução	4
3.1	Expressões Regulares	4
3.2	Estados da Aplicação	4
3.3	Módulos da Aplicação	4
3.4	Estruturas de Dados	5
4	Codificação e Testes	6
4.1	Alternativas, Decisões e Problemas de Implementação	6
4.2	Testes realizados e Resultados	6
5	Conclusão	7
A	Código do Programa	8

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 estruturada da seguinte forma: possui um primeiro capítulo que faz uma contextualização ao assunto tratado neste trabalho, seguindo-se uma introdução onde são apresentadas as metas de aprendizagem pretendidas.

Posteriormente é exposto o tema escolhido para a desenvolver o trabalho e as tarefas que nele estão envolvidas. redfalar do capítulo da especificação de requisitos 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 capítulos 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 realizássemos 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 início com o nome de todas as pessoas retratadas.

2.2 Especificação do Requisitos

2.2.1 Dados

2.2.2 Pedidos

2.2.3 Relações

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

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 da 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 guardar alguns dados, a estrutura 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 de quem se tratam as fotos.

```
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

4.2 Testes realizados e Resultados

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

Capítulo 5

Conclusão

Síntese do Documento.

Estado final do projecto; Análise crítica dos resultados.

Trabalho futuro.

Apêndice A

Código do Programa

```
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 struct data da;
13 struct listaLigada * dados = NULL ;
14 struct listaLigada * nodo;
15
16 int digito(char d){
17     int x = 0;
18     if (d >= '0' && d <= '9') x = 1;
19     return x;
20 }
21
22 /* Função que converte uma string num tipo de dados struct
    data */
23 struct data breakFoto (char * d){
24     struct data new;
25     char ano[5], mes[3], dia[3];
26     if (!digito(quando[0])) {
27         new.ano = 1;
28         new.mes = 1;
29         new.dia = 1;
30         return new;
31     }
32
```

```

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

```

```

76  "<"(?i:QUANDO)" "(?i:DATA)"=\"" { BEGIN QUANDO1; }
77 <QUANDO1>"\" {
    BEGIN INITIAL; }
78 <QUANDO1>[^\"]+ {
    quando = strdup(ytext);
79

```

da

=

```

breakF
(
quando
)
;
}

```

```

80
81  "<"(?i:FOTO)" "(?i:FICHEIRO)"=\"" { BEGIN FOTO1; }
82 <FOTO1>"\" { BEGIN INITIAL; }
83 <FOTO1>[^\"]+ { foto
    = strdup(ytext);
84

```

nodo

=

```

novoNo
(
da
,
foto
,
quem
,
facto
)
;

```

85

```

sortedIns
(&
dados
,
nodo
)
;

```

```

86 }
87
88 .|\n
           { ; }
89 %%
90
91 int yywrap()
92 {
93     return(1);
94 }
95
96 int main()
97 {
98     FILE * html;
99     html = fopen("new.html","w");
100     newHeader("Museu da Pessoa", html);
101     yylex();
102     insertImg(dados, html);
103     endHtml(html);
104     fclose(html);
105     return 0;
106 }

```

Bibliografia