Processamento de Linguagens e Compiladores (3º ano da Licenciatura em Ciências da Computação)

Trabalho Prático 1

Relatório de Desenvolvimento

Bruno Guedes (68707) Isac Meira (70069)
Rafael Silva (68685

2 de Novembro de 2014

Resumo

Neste relatório, encontra-se exaustivamente documentado todo o processo de concepção, testes e análise de um filtro de texto escrito em Flex, responsável pela geração automática de um ficheiro html, dado um ficheiro escrito correctamente no dialecto Enamex do XML, que resolve o problema proposto "2.2 Processamento de Entidades Nomeadas (Enamex)"

Conteúdo

Capítulo 1

Introdução

Enquadramento No âmbito da Unidade Curricular de Processamento de Linguagens e Compiladores, inserida no 1º semestre do 3º ano da Licenciatura em Ciências da Computação, lecionada pelo Sr. Professor Pedro Rangel Henriques, fomos desafiados a realizar um trabalho prático que tinha como por objectivos aumentar a experiência de uso do ambiente Linux, da linguagem Imperativa C, uma maior experência no uso de ferramentas de apoio à programação, bem como aumento da capacidade de escrever Expressões Regulares para descrição de padrões de frases, desenvolvimento de Processadores de Linguagens Regulares automáticos e utilização de filtros de texto, como o Flex.

Desta feita, foi-nos facultado pelo professor Pedro Rangel Henriques um guião com propostas de trabalhos a realizar, de entre os quais o grupo decidiu realizar "2.2 Processamento de Entidades Nomeadas (Enamex)", que tinha como por objectivo, dado um texto anotado com um dialecto XML chamado de Enamex, listar todas as pessoas , países, cidades e organizações, sem repetições, num ficheiro html dessas entidades nomeadas. Para o efeito, realizou-se um filtro de texto em Flex que recebia de input o ficheiro de texto e gerava automaticamente o ficheiro html. O filtro de texto Flex tinha associado a si, uma estrutura de dados "Lista"codificada em C, auxiliando assim a realização do ficheiro html.

Conteúdo do Trabalho Prático O trabalho prático possui um filtro de Texto codificado em Flex, que recebe de input um ficheiro de texto(.txt) com um texto devidamente anotado no dialecto Enamex do XML. O filtro de texto(codificado em Flex) possui expressões regulares, que quando lêm padrões como

```
<ENAMEX TYPE="PERSON"> texto </ENAMEX>,
<ENAMEX TYPE ="LOCATION" SUBTYPE="CITY"> texto </ENAMEX>
```

reagem adicionando o texto para a lista de strings correspondente à entidade nomeada, que a adiciona sem que haja repetições.
A lista de strings é programada em C e possui funções que permitem adicionar a si as palavras que pretendemos expôr no ficheiro html, bem como passar o seu conteúdo todo para um ficheiro html.

Resultados – pontos a evidenciar O resultado de passar o texto anotado no dialecto Enamex do XML pelo filtro de texto codificado em Flex, com o auxilio de uma estrutura de dados Lista, permite possuir-se de resultado um ficheiro .html com os nomes das pessoas, cidades, paises e organizações sem repetições e com um aspecto mais apelativo e sem que o restante conteúdo que não nos interessa saber apareça, no caso concreto da informação que pretendemos analisar.

Estrutura do Trabalho Prático O trabalho prático encontra-se organizado da seguinte forma:
O filtro de texto escrito em flex, encontra-se redigido num ficheiro .l que tem de nome proENAMEX, esse programa faz include de "List.h"para chamar a estrutura de dados Lista, desenvolvida em código C. A estrutura de dados Lista, possui um ficheiro header(.h) e o código C, sendo estes os ficheiros "List.h"e "List.c". O trabalho possui também um ficheiro makefile, que executando "make all", realiza toda a compilação do trabalho, restando então executar o programa gerado escrevendo na consola

```
./ENAMEX < "ficheiro".txt
```

Estrutura do Relatório

O relatório é iniciado narrando de forma consica e precisa o enquadramento de toda a problemática explicada no relatório, a estrutura do trabalho e os resultados obtidos no desenvolvimento do trabalho prático.
Seguidamente segue-se uma análise e especificação do problema que irá ser tratado, falando nas especificidades do problema e nos seus requisitos.
É então apresentada a concepção da solução, evidenciando as estruturas de dados usadas, os algoritmos implementados, apresentando a sua codificação e as decisões tomadas para a sua implementação.
Por fim, encontra-se uma breve síntese do trabalho, narrando o aprendizado a elegância da solução e o poder da ferramenta flex na concepção de filtros de texto.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Foi-nos proposto que codificássemos um filtro de texto em flex, em que líamos um documento anotado no sistema ENAMEX, cujo objectivo consistia em três pontos:
Ponto 1: Listar todas as pessoas identificadas, sem repetições;
Ponto 2: Listar todos os países e cidades marcadas no documento;
Ponto 3: Listar todas as organizações;
Como resultado, teríamos que gerar um ficheiro em HTML que mostrasse os resultados obtidos para as especificações acima referidas.

2.2 Especificação dos Requisitos

2.2.1 Dados

No documento ENAMEX, são-nos dadas palavras marcadas ao longo de todo o texto. Essas marcações variam entre vários tipos. Temos por exemplo:
<ENAMEX TYPE="PERSON"> - Esta marcação identifica uma pessoa;
<ENAMEX TYPE="LOCATION" SUBTYPE="CITY"> - Esta marcação identifica uma cidade;
<TIMEX TYPE="DATE"> - Esta marcação identifica uma data;
<ENAMEX TYPE="?"> - Esta marcação identifica algo que está indefinido;
<ENAMEX TYPE="LOCATION"> - Esta marcação identifica uma localização sem subtipo definido;
<ENAMEX TYPE="LOCATION" SUBTYPE="COUNTRY"> - Esta marcação identifica um país;
<Numex> - Esta marcação identifica um determinado número;
<ENAMEX TYPE="ORGANIZATION"> - Esta marcação identifica uma organização;

2.2.2 Pedidos

Neste problema é nos pedido apenas para listar-mos num ficheiro gerado em HTML todas as pessoas (sem repetição), todos os países, cidades e todas as organizações.
Para tal, filtraremos com recurso a expressões regulares todos os marcadores presentes no documento que possam identificar algo de interesse para o problema.

2.2.3 Relações

A marcação ENAMEX que identifica uma pessoa é <ENAMEX TYPE="PERSON">.
Para marcar uma localização usamos <ENAMEX TYPE="LOCATION"... sendo que podemos ter dois subtipos em localização, sendo eles SUBTYPE="CITY"> e SUBTYPE="COUNTRY">. Por último, as organizações são marcadas por <ENAMEX TYPE="ORGANIZATION">.
O objectivo e o que explicaremos mais detalhadamente nos próximos capítulos baseia-se em guardar numa lista a string que segue cada uma das 4 marcações importantes para este problema, e quando estiver todo o texto do documento ENAMEX percorrido, aceder-lhes e transcrevê-las agrupadamente num ficheiro HTML.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

A fim de se conseguir listar as entidades nomeadas Pessoa, Cidade, País, e Organizações foi criada uma Lista Ligada para armazenar as entidades nomeadas, em que essas entidades eram inseridas na lista sem repetições, sendo a estrutura de dados a que se segue:

```
/**Ficheiro Header da implementação de Listas Ligadas para resolução do
*problema "2.2 Processamento de Entidades Nomeadas (Enamex)" */

#include <stdlib.h>
#include <stdio.h>

typedef struct node{
        char* text; //string de texto a armazenar
        struct node* next; //próxima string de texto
}Node, *List;

List init(char* t); //cria uma Lista com uma string t
int isEmpty(List l); // testa se a lista está vazia
int exist_Text(List l, char* t); //testa se a string t já
                // se encontra inserida na lista
List add(List l, char* t); //adiciona um texto t à lista.
FILE* writeList(FILE *f, List l); // escreve o conteudo da lista l no ficheiro f
char* trataString(char *s, char l); //transforma todos os caracteres
                // l da string s num espaço.
```

3.2 Algoritmos

Os Algoritmos usados na resolução do problema foram os seguintes:

```
/** Implementação dos algoritmos das Listas Ligadas para manipulação
*das mesmas futuramente no filto de texto Flex.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "List.h"

/** Dada uma string de entrada t, cria uma lista com a
*string t
*@param string t
*@return Lista com string t
*/
List init(char* t){
        List new;
        if((new = (List)malloc(sizeof(Node))) == NULL) //testa a alocação de memória
                return NULL;

        new->text = strdup(t); //aloca memória e faz o strcpy para new->text
        new->next = NULL;

        return new;
}
/** Testa se a lista l está vazia ou não.
*@param Lista l
*@return TRUE(!=0) ou FALSE(==0)
*/
int isEmpty(List l){
        return(l == NULL);
}

/**Testa se a string t já se encontra inserida na lista l.
*@param Lista l e string t
@return TRUE(!=0) ou FALSE(==0)
*/
int exist_Text(List l, char* t){
        List aux = l;
        //enquanto houverem string's na lista testa se alguma é
        //igual à string t dada de parametro.
        while(!isEmpty(aux)){
                if(strcmp(aux->text, t) == 0)
```

```
                        return 1;

                aux = aux->next;
        }
        //não encontrou nenhuma string igual.
        return 0;
}

/**Dada uma string s e um caracter n, substitui todas as entradas
*do caracter n na string s pelo caracter ' '.
*Cria uma string auxiliar com o tamanho da string s e efectua as
*alterações nessa lista auxiliar que é retornada.
*@param String s e caracter n
*@return String com as alterações.
*/
char* trataString(char*s, char n){
        char *s2;
        //aloca-lhe o tamanho da string s
        s2 = (char*) malloc((strlen(s)+1) * sizeof(char));
        int i, j=0;

        for(i=0; s[i]!='\0'; i++){
                if(s[i] == n)
                        s2[j++] = ' '; //substitui as entradas de n por ' '
                else
                        s2[j++] = s[i];
        }
        s2[j] = '\0'; //acrescenta o caracter terminador à String.

        return s2;
}

/**Dada uma lista de entrada l e uma String t, trata a string para
*lhe retirar as quebras de linha por espaços, testando de seguida se a
*string já trabalhada já se encontra na lista, se tal não acontecer
*insere a string t manipulada na lista l.
@param Lista l, String t
@return Lista l
*/
List add(List l, char* t){
        t = trataString(t, '\n');//substitui '\n' por ' '
        if(!exist_Text(l, t)){
                //insere a strinf na lista l
                List new = init(t);
                new->next = l;
                l = new;
```

```
        }

        return l;
}

/**Dado um ficheiro f e uma Lista l, Lista todos os elementos no ficheiro f
*usando para isso nos fprintf (escrita em ficheiro) código especifíco de html.
<ul> começa a listagem
<li> itens da listagem
*@param Ficheiro f e Lista l
*@return Ficheiro f
*/
FILE* writeList(FILE *f, List l){
        List aux = l;
        fprintf(f,"<ul>\n");

        while(!isEmpty(aux)){
                fprintf(f,"<p><li>%s</li></p>\n", aux->text);
                aux = aux->next;
        }

        fprintf(f,"</ul>\n");

        return f;
}
```

Capítulo 4

Codificação e Testes

4.1 Decisões

Na codificação de expressões regulares para identificação das entidades nomeadas, o grupo decidiu usar TAG's, assim quando detectada uma TAG de abertura, como por exemplo,

```
<ENAMEX TYPE="PERSON">
```

a partir daí conseguiriamos filtrar todo o texto útil sem termos que manipular a string yytext com o texto que queriamos inserir na lista, pois possuimos uma expressão regular que lê tudo excepto <, que é precisamente o texto delimitado entre:

```
<ENAMEX TYPE="PERSON"> e </ENAMEX>
```

Deste feita quando encontra a expressão regular

```
</ENAMEX>
```

o filtro de texto flex, volta a procurar uma TAG de abertura que coincida com o ficheiro de texto, para assim filtrar a sua informação relevante.
O grupo decidiu também que a melhor estrutura de dados a ser usada, era a Lista Ligada, por motivos obvios e simples. Não sabemos à partida quantos elementos iremos ter que colocar para armazenar todas as strings que pretendemos, pelo que precisavamos de uma estrutura com memória dinâmica para albergar os textos filtrados.

4.2 Testes realizados e Resultados

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

INPUT

```
<ENAMEX TYPE="PERSON"> Bento de Castro Abreu </ENAMEX>, que
emigrou para o <ENAMEX TYPE="LOCATION" SUBTYPE="CITY"> Rio de Janeiro </ENAMEX> em
<TIMEX TYPE="DATE"> 1/2/1895 </TIMEX>,
com 26 anos de idade, e referido no seu passaporte como proprietario, sobrinho de
outro <ENAMEX TYPE="?"> Brasileiro </ENAMEX>
<ENAMEX TYPE="PERSON"> Fernando de Castro Abreu e Magalh~
aes </ENAMEX>,
que apoiou financeiramente a construcão da casa do
<ENAMEX TYPE="LOCATION"> Santo Novo </ENAMEX>.
<ENAMEX TYPE="?">Foi Visconde e Marquˆ
es de Paranagua</ENAMEX> <ENAMEX TYPE="PERSON">Francisco
Vilela Barbosa</ENAMEX> que nasceu
no <ENAMEX TYPE="LOCATION" SUBTYPE="COUNTRY">Brasil</ENAMEX>, em
<ENAMEX TYPE="LOCATION" SUBTYPE="CITY">Parati</ENAMEX>
em <ENAMEX TYPE="?">Novembro</ENAMEX> de <NUMEX>1769</NUMEX> e ali
morreu em <TIMEX TYPE="DATE">Setembro de 1846</TIMEX>, filho de
<ENAMEX TYPE="PERSON">Francisco Vilela Barbosa</ENAMEX>,
natural de <ENAMEX TYPE="LOCATION" SUBTYPE="CITY">Braga</ENAMEX>
(<ENAMEX TYPE="LOCATION" SUBTYPE="Country">Portugal</ENAMEX>) e de
<ENAMEX TYPE="PERSON">D. Ana Maria da Conceicão</ENAMEX>.
Formado em matematica pela <ENAMEX TYPE="ORGANIZATION">Universidade
de Coimbra</ENAMEX>, em <Numex>1789</NuMEX>
assentou praca na armada e quando 2 o tenente prestou relevantes
servicos no cerco de <ENAMEX TYPE="LOCATION" SUBTYPE="CITY">Tunis</ENAMEX>
e na perseguicão aos pirata argelinos do
<ENAMEX TYPE="?">Mediterrˆ
aneo</ENAMEX>.
Nomeado lente da <ENAMEX TYPE="ORGANIZATION">Academia da Marinha</ENAMEX>,
passou em <Numex>1801</NuMEX> para o
<ENAMEX TYPE="ORGANIZATION">Real Corpo de Engenheiros</ENAMEX>
e em <Numex>1810</NuMEX> foi promovido a <ENAMEX TYPE="?">Major</ENAMEX>
e reformado mais tarde no
posto de <ENAMEX TYPE="?">Brigadeiro</Enamex>.
```

RESULTADO

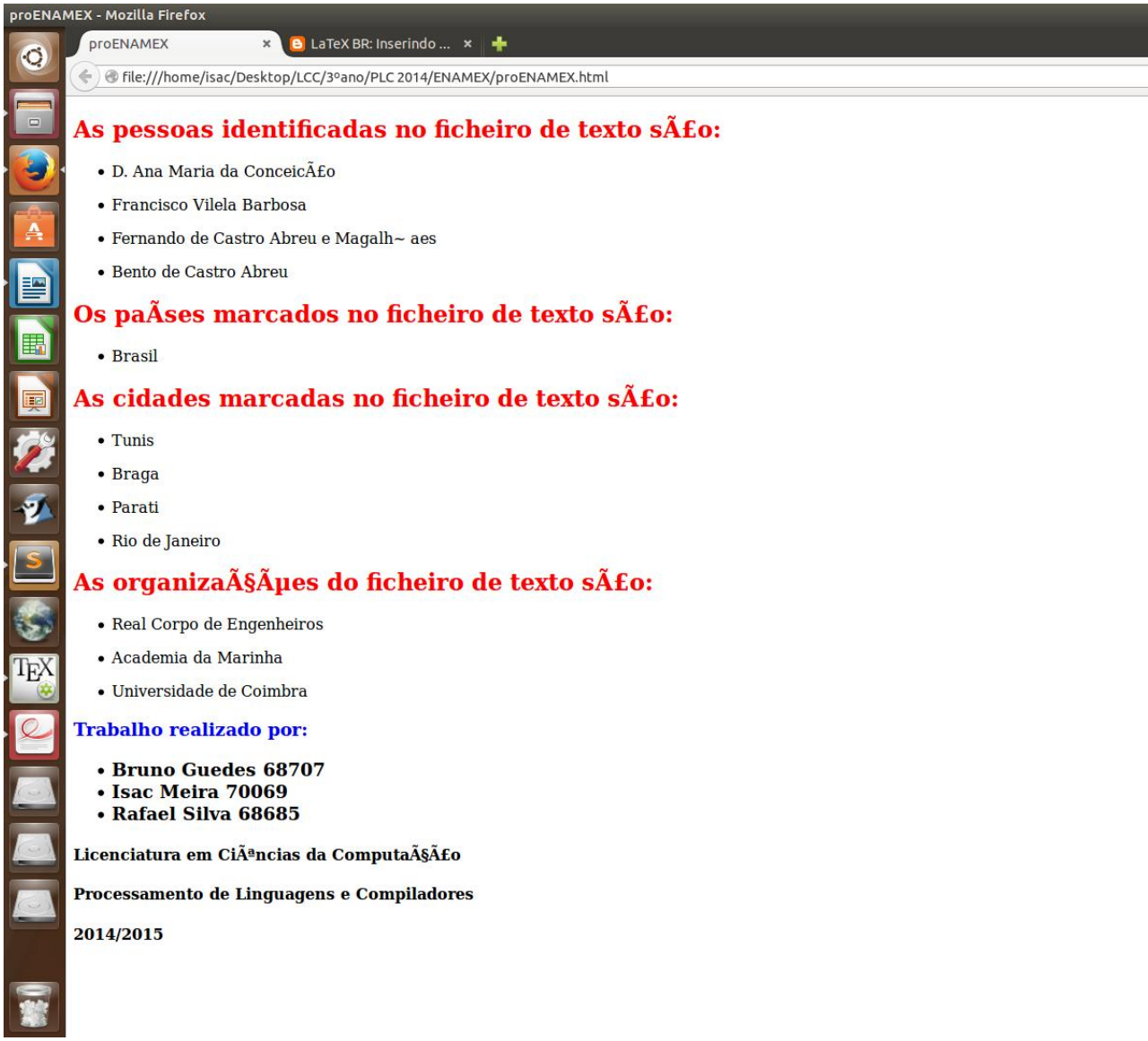

Figura 4.1: Resultado do teste

Capítulo 5

Conclusão

Com a leitura deste relatório, o leitor pode-se aperceber que o uso de filtros de texto, nomeadamente o flex, neste caso concreto, é uma ferramente muito poderosa e útil para se conseguir filtrar conteúdo.
Podemos aperceber-nos também, como a correcta manipulação de Expressões Regulares na linguagem flex, permite arquitecturar soluções simples e elegantes que seriam impossíveis nos paradigmas de programação a que nos encontrámos habituados a utilizar. De salientar a forma como o flex funciona, sendo o seu Automáto Finito Reactivo o responsavel para que quando a Expressão Regular é satisfeita, conseguirmos executar a acção que pretendemos.
Desta feita, com o uso de um filtro de texto escrito em Flex e uma estrutura de dados Lista, conseguimos realizar uma solução simples e eficiente, que dado um ficheiro escrito num dialecto Enamex, gera automaticamente um ficheiro html, esteticamente bem concebido e com a informação organizada simples e intuitiva.

Apêndice A

Código do Programa

proComent.l :

```
%{
#include "List.h"
List person = NULL;
List organization = NULL;
List country = NULL;
List city = NULL;

%}

%x PERSON
%x ORGANIZATION
%x CITY
%x COUNTRY

%%
\<ENAMEX" "+TYPE\=\"PERSON\"\>        {BEGIN PERSON;}
<PERSON>[^<]+        {person = add(person, yytext);}
<PERSON>\<\/ENAMEX\>        {BEGIN INITIAL;}
\<ENAMEX" "+TYPE\=\"ORGANIZATION\"\>        {BEGIN ORGANIZATION;}
<ORGANIZATION>[^<]+        {organization = add(organization, yytext);}
<ORGANIZATION>\<\/ENAMEX\>        {BEGIN INITIAL;}
\<ENAMEX" "+TYPE\=\"LOCATION\"" "+SUBTYPE\=\"COUNTRY\"\>        {BEGIN COUNTRY;}
<COUNTRY>[^<]+        {country = add(country, yytext);}
<COUNTRY>\<\/ENAMEX\>        {BEGIN INITIAL;}
\<ENAMEX" "+TYPE\=\"LOCATION\"" "+SUBTYPE\=\"CITY\"\>        {BEGIN CITY;}
<CITY>[^<]+                {city = add(city, yytext);}
<CITY>\<\/ENAMEX\>        {BEGIN INITIAL;}
.|\n         {;}
```

```
%%

int yywrap(){
        return (1);
}

int main(){
yylex();

FILE *f;

f = fopen("proENAMEX.html", "w");
fprintf(f,"<html><title>proENAMEX</title><body><img src=\"uminho.jpg\"");
fprintf(f,"align=\"right\"><p><b>");
fprintf(f,"<h2><font color=\"red\">As pessoas identificadas no ficheiro");
fprintf(f,"de texto são:</font></h2></b> </p>");
f = writeList(f, person);
fprintf(f,"<p><b><h2><font color=\"red\">Os países marcados no ficheiro");
fprintf(f,"de texto são:</font></h2></b> </p>");
f=writeList(f,country);
fprintf(f,"<p><b><h2><font color=\"red\">As cidades marcadas no ficheiro");
fprintf(f,"de texto são:</font></h2></b> </p>");
f=writeList(f,city);
fprintf(f, "<p><b><h2><font color=\"red\">As organizações do ficheiro");
fprintf(f,"de texto são:</font></h2></b> </p>");
f=writeList(f,organization);
fprintf(f,"<p><h3><font color=\"blue\">Trabalho realizado por:");
fprintf(f,"</p></font><ul><li>Bruno Guedes 68707");
fprintf(f,"</li><li>Isac Meira 70069</li><li>Rafael Silva 68685");
fprintf(f,"</li></ul><p><h4>Licenciatura em Ciências da Computação</h4></p><p>");
fprintf(f,"<h4>Processamento de Linguagens e Compiladores</h4>");
fprintf(f,"</p><p><h4>2014/2015</h4></p> </body></html>");
fclose(f);
return 0;
}
```

List.h :

```
/**Ficheiro Header da implementação de Listas Ligadas para resolução do
*problema "2.2 Processamento de Entidades Nomeadas (Enamex)" */

#include <stdlib.h>
#include <stdio.h>

typedef struct node{
        char* text; //string de texto a armazenar
        struct node* next; //próxima string de texto
```

```
}Node, *List;

List init(char* t); //cria uma Lista com uma string t
int isEmpty(List l); // testa se a lista está vazia
int exist_Text(List l, char* t); //testa se a string t já
                // se encontra inserida na lista
List add(List l, char* t); //adiciona um texto t à lista.
FILE* writeList(FILE *f, List l); // escreve o conteudo da lista l no ficheiro f
char* trataString(char *s, char l); //transforma todos os caracteres
                // l da string s num espaço.
```

List.c :

```
/** Implementação dos algoritmos das Listas Ligadas para manipulação
*das mesmas futuramente no filto de texto Flex.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "List.h"

/** Dada uma string de entrada t, cria uma lista com a
*string t
*@param string t
*@return Lista com string t
*/
List init(char* t){
        List new;
        if((new = (List)malloc(sizeof(Node))) == NULL) //testa a alocação de memória
                return NULL;

        new->text = strdup(t); //aloca memória e faz o strcpy para new->text
        new->next = NULL;

        return new;
}
/** Testa se a lista l está vazia ou não.
*@param Lista l
*@return TRUE(!=0) ou FALSE(==0)
*/
int isEmpty(List l){
        return(l == NULL);
}

/**Testa se a string t já se encontra inserida na lista l.
*@param Lista l e string t
```

```
@return TRUE(!=0) ou FALSE(==0)
*/
int exist_Text(List l, char* t){
        List aux = l;
        //enquanto houverem string's na lista testa se alguma é
        //igual à string t dada de parametro.
        while(!isEmpty(aux)){
                if(strcmp(aux->text, t) == 0)
                        return 1;

                aux = aux->next;
        }
        //não encontrou nenhuma string igual.
        return 0;
}

/**Dada uma string s e um caracter n, substitui todas as entradas
*do caracter n na string s pelo caracter ' '.
*Cria uma string auxiliar com o tamanho da string s e efectua as
*alterações nessa lista auxiliar que é retornada.
*@param String s e caracter n
*@return String com as alterações.
*/
char* trataString(char*s, char n){
        char *s2;
        //aloca-lhe o tamanho da string s
        s2 = (char*) malloc((strlen(s)+1) * sizeof(char));
        int i, j=0;

        for(i=0; s[i]!='\0'; i++){
                if(s[i] == n)
                        s2[j++] = ' '; //substitui as entradas de n por ' '
                else
                        s2[j++] = s[i];
        }
        s2[j] = '\0'; //acrescenta o caracter terminador à String.

        return s2;
}

/**Dada uma lista de entrada l e uma String t, trata a string para
*lhe retirar as quebras de linha por espaços, testando de seguida se a
*string já trabalhada já se encontra na lista, se tal não acontecer
*insere a string t manipulada na lista l.
@param Lista l, String t
@return Lista l
```

```
*/
List add(List l, char* t){
        t = trataString(t, '\n');//substitui '\n' por ' '
        if(!exist_Text(l, t)){
                //insere a strinf na lista l
                List new = init(t);
                new->next = l;
                l = new;
        }

        return l;
}

/**Dado um ficheiro f e uma Lista l, Lista todos os elementos no ficheiro f
*usando para isso nos fprintf (escrita em ficheiro) código especifíco de html.
<ul> começa a listagem
<li> itens da listagem
*@param Ficheiro f e Lista l
*@return Ficheiro f
*/
FILE* writeList(FILE *f, List l){
        List aux = l;
        fprintf(f,"<ul>\n");

        while(!isEmpty(aux)){
                fprintf(f,"<p><li>%s</li></p>\n", aux->text);
                aux = aux->next;
        }

        fprintf(f,"</ul>\n");

        return f;
}
```