Processamento de Linguagens (3º ano de MIEI)

Trabalho Prático nº1

Relatório de Desenvolvimento

Luís Alves (a80165)

Rafaela Rodrigues (a80516)

31 de Março de 2019

Resumo

Este relatório inicia-se com uma breve contextualização, seguindo-se a descrição dos problemas propostos e o que deve ser desenvolvido para os solucionar. Segue-se a exposição da arquitetura da solução, sendo descritas as diversas estruturas de dados utilizadas e o modo de obtenção da informaçao pretendida. De seguida é explanado o processo de criação dos ficheiros normalizados e HTML, incluindo todas as decisões tomadas pelo grupo. Por fim, são apresentadas conclusões sobre o trabalho desenvolvido.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito do primeiro Trabalho Prático da Unidade Curricular de Processamento de Linguagens, que se insere no 2º semestre do 3º ano do primeiro ciclo de estudos do Mestrado Integrado em Engenharia Informática.

1.2 Objetivos e Trabalho Proposto

Pretende-se com este relatório formalizar toda o processo de tratamento de notícias de um jornal para vários fins.

Para este trabalho foi proposta a limpeza, normalização e criação de um ficheiro HTML por cada notícia do "Jornal Angolano - Folha 8, v2". Para além disso, também foi proposta a geração de um ficheiro HTML com todas as tags e respetivos *hyperlinks* para as notícias que as contenham.

1.3 Resumo do trabalho a desenvolver

Para atingir os objetivos deste trabalho, foi importante definir quais as informações que considerávamos pertinentes guardar e quais as estruturas para tal.

De seguida, foram estipuladas as expressões regulares e respetivas ações, que nos permitiu obter todos os dados pretendidos.

Por fim foram gerados todos os ficheiros, de acordo com o enunciado.

Capítulo 2

Análise e Especificação

2.1 Plano de implementação

Após a análise do problema, o grupo decidiu fazer o processamento de cada notícia de forma a guardar as informações pertinentes, e no fim deste, tratar os dados obtidos de acordo com o plano estabelecido (explicitado nos capítulos seguintes).

2.2 Informação Pretendida

Após a análise de algumas notícias do jornal, foi possível identificar os campos que consideramos essenciais para a caracterização de uma notícia, sendo estes:

- Tags
- Identificador (ID)
- Categorias
- Título
- Data de publicação
- Texto (notícia)

Assumindo que todas as notícias possuem IDs diferentes, este será usado como identificador inequívoco. Para além disso, todas as informações que não se enquadram em nos itens acima foram ignoradas.

Capítulo 3

Conceção/desenho da Resolução

3.1 Estruturas de Dados

Estando os dados pertinentes bem definidos, é necessário guardá-los para serem usados posteriormente. Para uma maior facilidade de armazenamento em estruturas de dados, recorremos à biblioteca GLib. Na tabela 3.1, encontram-se as estruturas de dados escolhidas para o efeito.

Estrutura	Utilização
char *	Utilizado para guardar o ID, título, a data e o texto
GSList	Uma lista para as tags e uma lista para as categorias de uma notícia
GHashTable	Hash que possui como chave o nome de uma tag e como chave a estrutura TagValues
TagValues	Estrutura que permite guardar múltiplas informações de uma determinada tag.

Tabela 3.1: Estruturas e respetiva utilização

Os tipos de dados char * permitem guardar as informações mais simples de cada noticia, sendo que o seu conteúdo é substituído aquando da utilização da mesma variável para uma nova notícia. Para tal, assumimos que existem sempre estes campos.

As GSLists possuem apenas as tags e categorias de uma notícia, ou seja, cada vez que passe uma nova pelo filtro, inicia-se uma nova lista. Estas permitem um acesso organizado a todas as tags e categorias de uma determinada notícia.

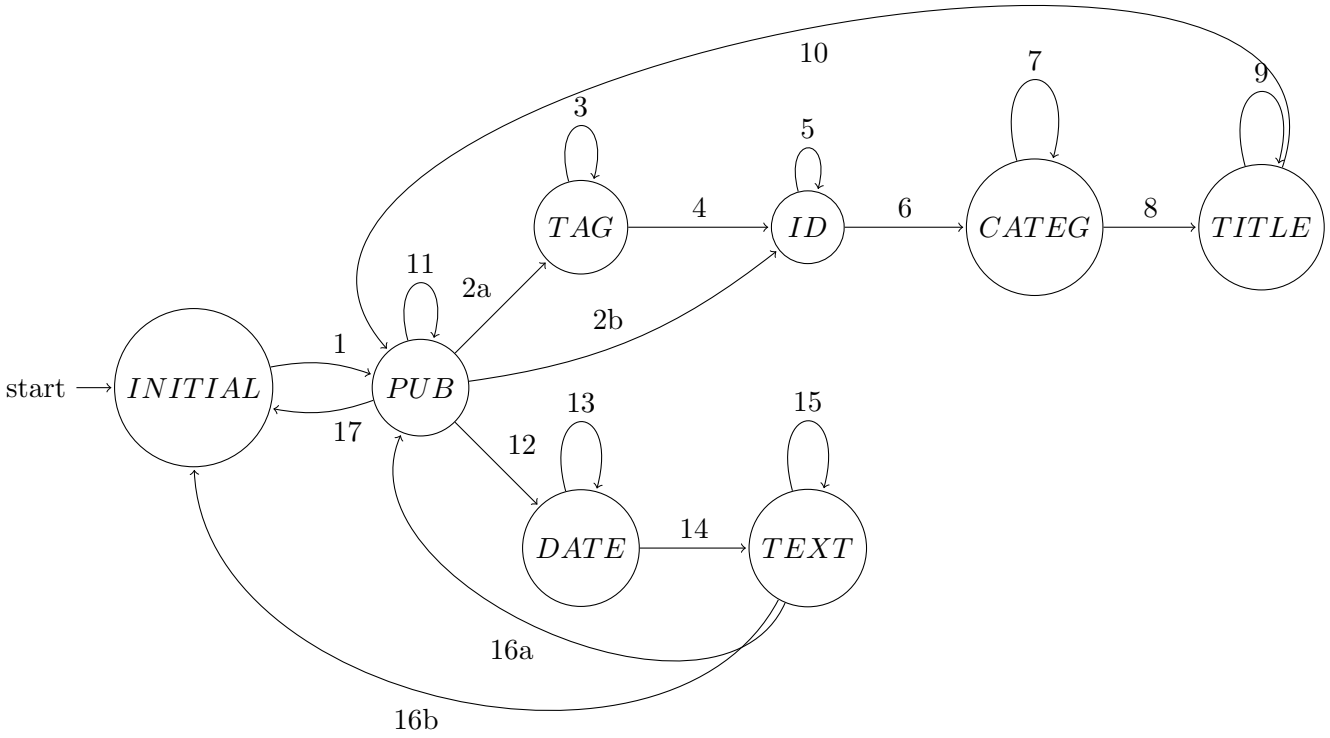
Relativamente à GHashTable, esta é partilhada por todas as notícias. Tendo como chave o nome de uma tag e como valor a estrutura TagValues, esta permitiu guardar informações sobre as tags que apareceram ao longo de todas as notícias.

Por fim, a estrutura TagValues possui um inteiro, que representa o número de ocorrências de uma determinada tag e uma lista com os ID de todas as notícias que contêm essa tag. Esta estrutura permitiu satisfazer um dos objetivos propostos com maior facilidade, como será aprofundado mais à frente.

3.2 Algoritmos/Resolução de implementação

Estando todas as estruturas definidas, seguimos para a criação de uma estratégia para o processamento de todas as notícias. Para tal, recorremos a expressões regulares que nos permitem selecionar informação baseada em *pattern matching*. Para além disso, usamos o FLex, uma ferramenta que gera programas que realizam *pattern matching* em textos.

Para uma melhor orientação no trabalho a realizar, desenvolvemos um autómato que nos permitiu compreender e definir todas as expressões necessárias.



Cada nodo do autómato representa a informação que desejamos obter e guardar. Para tal, foram criadas definições de contexto para cada um. Isto permitiu que, quando fosse encontrado um determinado padrão, entrassemos no contexto correto, facilitando assim a extração da informação.

Relativamente às setas, estas indicam qual o contexto seguinte, sendo que há a possibilidade de continuar no mesmo ou avançar para outro. Para além disso, cada seta possui um número que, para além de reperesentar o fluxo esperado, também representa a expressão regular que indica (ou não) a mudança de contexto.

De seguida, iremos aprofundar as expressões dos arcos. Para cada arco será explicado o que representa a alteração de contexto (ou não) e explicitar o algoritmo usado para obter os dados pretendidos. De ressalvar que o algoritmo apresentado não corresponde ao código produzido, pois foi modificado para uma melhor compreensão.

Arcos 1,2,4,6,8,10,12,16

Todos estes arcos representam uma transição entre contextos diferentes. Devido à sua simplicidade, estes podem ser generalizados. Quando é encontrada uma expressão que representa o início de um novo contexto, este é inicializado.

Normalmente, a mudança de contexto deve-se a um ou mais `\n`. Então, sabendo a ordem dos contextos, é fácil definir o momento da mudança.

Por exemplo, no arco 8 (Categoria -> Title), o algoritmo resultante é:

$$< CATEG > \texttt{\textbackslash n\textbackslash n} \quad \{BEGINTITLE;\}$$

Ou seja, quando no contexto CATEG aparecerem dois \n seguidos, entramos no contexto TITLE.

Relativamente ao arco 2a e 2b, existem duas possibilidades de mudança de contexto, dado que existem notícias que não possuem tags. Caso isso se verifique, passamos diretamente do contexto PUB para ID.

Arco 3

Este arco representa a expressão regular que nos permite obter todas as tags de uma notícia. O algoritmo utilizado foi o seguinte:

```
<TAG>tag:{[^}]+}    {
    retira chavetas finais da tag obtida;
    tag_name = yytext + tamanho(tag:{);

    se tag_name existe em hash:
        ocorrencias_da_tag++;
    senão:
        insere tag em hash;
        ocorrencias_da_tag = 1;

    adiciona tag_name a tags do post;
}
```

De uma forma resumida, a tag obtida é procurada na estrutura tagCount. Caso já exista essa tag na tabela de hash, é incrementado o valor do contador. Caso contrário, é alocada memória para uma nova estrutura tagValue e os valores iniciais são definidos. No fim, a tag é adicionada à lista de tags do post.

Arco 5

O arco 5 representa duas possíveis expressões.

Uma dela apanha o ID da notícia e a outra descarta todas as informações desnecessárias.

Começaremos por analisar o algoritmo do ID:

```
<ID>{post-{dig}+ {
    idPost = yytext + tamanho({);
    para cada tag do post:
        adiciona idPost;
}
```

Este algoritmo acede à estrutura tagCount e para cada uma das tags que se encontram na lista "tags"(obtidas anteriormente), procura a tag e adiciona o ID da notícia à lista "idPosts", da estrutura tagValues (chave).

Para descartar a informação desnecessária, é usada a expressão

$$< ID > . \quad \{;\}$$

.

Arco 7

O arco 7 também representa duas possíveis expressões, tal como o arco anterior. Uma dela obtém todas as categorias da notícia e a outra ignora todas as informações não pertinentes.
O algoritmo utilizado para as categorias foi:

```
<CATEG>[A-Z][^A-Z\n ]+ {
    categoria = yytext;
    adiciona categoria a categorias do post;
}
```

Este algoritmo apenas adiciona na lista "categories", a categoria que encontrou. Este processo é repetido para todas as categorias que encontrar.
Para descartar a informação desnecessária, é usada a expressão

$$< ID > .|\texttt{\textbackslash n} \quad \{;\}$$

Arco 9 e 13

Estes arcos utilizam expressões que permitem obter o título (arco 9) e a data (arco 11) da notícia, sendo o seus algoritmos:

```
<TITLE>[^\n]+ {title = yytext;}
<DATE>.+ {date = yytext;}
```

Estes algoritmos são muito simples, pois apenas copia o texto encontrado e guarda na variável title e date, respetivamente.

Arco 14

Apesar deste ser um arco de transição, a ação quando a expressão dá match limpa a variável que irá guardar o texto, pois esta poderá ter texto da notícia anterior. O algoritmo proposto é:

```
<DATE>\n {
liberta memória de text;
aloca memória para text;
BEGIN TEXT;
}
```

Por fim, é alocada nova memória para o texto que vai surgir de seguida.

Arco 15

Este arco representa a expressão que coleta todo o corpo da notícia (texto) e a expressão que ignora informação desnecessária . Para a primeira expressão, foi definida a seguinte expressão regular e algoritmo:

```
<TEXT>.*|\n {
tamanho = tamanho(text) + yyleng + 1;
```

```
se tamanho > tamanhoDoTexto:
    aloca dobro da memória;
    copia conteúdo antigo;
    concatena com yytext;

senão:
    concatena com yytext;
}
```

Este algoritmo permite uma gestão mais eficiente de memória, ao duplicar o espaço alocado sempre que este se revela insuficiente.
Para ignorar a informação desnecessária, o algoritmo é igual ao utilizado nos arcos anteriores para o mesmo fim.

Arco 16

O arco 16 corresponde à expressão que indica que o texto terminou.
O fim deste pode ser identificado pela frase "Partilhe este artigo"ou "Etiquetas". Quando estes padrões são encontrados, inicia-se o proceso de normalização e geração de ficheiros HTML e transita-se para o contexto PUB.
No entanto, existem notícias que não possuem nenhuma destas frases. Para este caso, ao encontrar o símbolo de fim de publicação ($< /pub >$), passamos diretamente para o contexto INITIAL.
O algoritmo é:

```
<TEXT>\nPartilhe" "este" "Artigo\n { formatPrintNorm();formatPrintHTML();BEGIN PUB;}
<TEXT>\nEtiquetas:     { formatPrintNorm();formatPrintHTML();BEGIN PUB;}
<TEXT>\<\/pub\>{formatPrintNorm();formatPrintHTML();BEGIN INITIAL;}
```

Arco 17

Este arco representa o fim de uma notícia, entrando-se no contexto INITIAL.
Voltar para esse contexto permite-nos iniciar o processamento de uma nova noticia, até não haver mais notícias.
A expressão regular e o algoritmo são:

$$< PUB >< /pub > \quad \{BEGININITIAL\}$$

Finda a análise de todas as expressões regulares e respetivas ações que nos permitiram tratar das notícias, passaremos para a criação dos ficheiros normalizados e HTML.

3.3 Normalização da notícia

Para a normalização da notícia, assumimos que uma notícia normalizada possui o formato do exemplo fornecido no enunciado, ou seja, cada informação recolhida (título, data, texto, etc) foi incluida dentro de tags iniciais e finais, como por exemplo:

```
<title>Zenú: a corrupção mata</title>
```

Para além disso, cada tag e as categorias da notícia também estão entre tags XML, tal como em:

```
<tags>
<tag>regime</tag> <tag>fundo soberano</tag> <tag>Zenú</tag> <tag>Saúde</tag>
<tag>Nepotismo</tag> <tag>MPLA</tag> <tag>Corrupção</tag> <tag>Angola</tag>
</tags>
```

Estando definida a estrutura, apenas foi necessário escrever para os ficheiros a informação da maneira correta.

Como foi dito anteriormente, o processo de criação dos ficheiros normalizados ocorre quando é encontrado o padrão correspondente à expressão regular que passa do contexto TEXT para PUB.

Cabe à função formatPrintNorm() escrever para o ficheiro <idPost>.norm toda a notícia, conforme a estrutura definida acima.

É apresentado um exemplo de um ficheiro com uma notícia normalizada em 4.1.

3.4 Ficheiro HTML

Um dos exercícios propostos era a criação de um ficheiro HTML por cada notícia.

Para tal, seguiu-se uma estratégia semelhante à usada para gerar ficheiros normalizados. No entanto, os ficheiros HTML possuem uma linguagem própria relativamente às tags. Por exemplo, para defnir uma lista de itens, usa-se a tag <li> (ex: Item).

Tendo esta informação em consideração, criamos para todos as notícias o respetivo ficheiro HTML, através da função formatPrintHTML().

3.5 Índice de tags

O último exercício proposto era a criação de um ficheiro HTML que apresentasse todas as tags encontradas ao longo das notícias e um hiperlink para a respetiva notícia.

Para tal, no fim do processamento de todas as notícias, é chamada a função generateIndex(), que percorre a hash tagCount, e para cada chave (tag) gera uma lista no ficheiro em que cada item corresponde ao id da notícia. Estes ids foram guardados ao longo do processamento, como referido anteriormente.

Estando todos os ficheiros HTML criados, já é possível consultar as notícias por tag. Para além disso, em frente a cada tag aparece o seu número de ocorrências.

Capítulo 4

Codificação e Testes

4.1 Decisões e Problemas de Implementação

O grupo utilizou como base o exemplo fornecido no enunciado. No entanto, ao longo do trabalho, apercebemo-nos que nem todas as notícias possuem aquele formato.

O principal problema encontrado foi o facto de haver notícias com Ids repetidos.

Para as notícias com ids repetidos, os ficheiros são reescritos, permanecendo a última notícia com o Id

Para além disso, descobrimos que existem tags que a meio possuem `\n`, não compactuando com o padrão geral e, por esse motivo, a tag é ignorada.

A nível da implementação, não foi criada uma expressão regular que apanhasse letras acentuadas, sendo que nessas palavras só são apanhadas as letras até à letra com acentuação.

Um outro problema, relacionado com a existência de notícias repetidas, passou pela impossibilidade de manter a memória sob controlo, uma vez que apenas o conteúdo maior (texto) é limpo regularmente. Isto deveu-se a vários erros que foram surgindos aquando de tentativas de limpar memória, devido à existência de notícias repetidas.

4.2 Testes realizados e Resultados

De seguida, apresenta-se o exemplo de uma notícia presente no ficheiro original, no formato normalizado e HTML.

Por fim, segue-se um exemplo da página HTML com as tags e respetivos hiperlinks.

Estes exemplos representam e correspondem às expetativas do grupo.

```
<pub id="post-11179">
    <title>Requiem pela "Livraria LELLO"</title>
    <author_date>Redacção F8 — 7 de Outubro de 2015</author_date>
    <tags>
        <tag>património</tag> <tag>livraria</tag> <tag>lello</tag> <tag>história</tag> <tag>fim</tag>
    </tags>
    <categories>
        <category>Destaque</category> <category>Cultura</category>
    </categories>
    <text>


A famosa e histórica livraria Lello, da baixa de Luanda, vendeu ao desbarato os últimos livros que
restavam antes de encerrar definitivamente.

S egundo os funcionários, o espaço ou edifício, ou parte dele, foi vendido. A livraria acabou. Os
funcionários vão para casa.

Não sabemos se os funcionários dessa livraria histórica se vão embora com ou sem indemnização. O
certo é que, no quadro dos novos hábitos "que estamos com eles", mais um espaço de cultura cedeu ao
poder dos dólares e foi destruído.

Nesta imponente empreitada de demolição do centro histórico de Luanda, assistimos impotentes a um
espectáculo desolador e imparável, dum lado, a destruição quase total do legado histórico da nossa
capital, do outro, os discursos de todos, dizemos bem, TODOS SEM EXCEPÇÃO, dirigentes deste nosso
querido país a proclamar aos azimutes e ao povo de Angola e ao mundo que é preciso salvaguardar os
buelens arquitectónicos que nos foram legados pela história de Angola.

A hipocrisia em todo o seu esplendor! Ricardo Manuel deve-se ter voltado mais de uma vez na sua
tumba.

    </text>
</pub>
```

Figura 4.1: Exemplo de ficheiro normalizado

11179

Requiem pela "Livraria LELLO"

Author - Date: Redacção F8 — 7 de Outubro de 2015

Tags:

- património
- livraria
- lello
- história
- fim

Categoria:

- Destaque
- Cultura

A famosa e histórica livraria Lello, da baixa de Luanda, vendeu ao desbarato os últimos livros que restavam antes de encerrar definitivamente. S egundo os funcionários, o espaço ou edifício, ou parte dele, foi vendido. A livraria acabou. Os funcionários vão para casa. Não sabemos se os funcionários dessa livraria histórica se vão embora com ou sem indemnização. O certo é que, no quadro dos novos hábitos "que estamos com eles", mais um espaço de cultura cedeu ao poder dos dólares e foi destruído. Nesta imponente empreitada de demolição do centro histórico de Luanda, assistimos impotentes a um espectáculo desolador e imparável, dum lado, a destruição quase total do legado histórico da nossa capital, do outro, os discursos de todos, dizemos bem, TODOS SEM EXCEPÇÃO, dirigentes deste nosso querido país a proclamar aos azimutes e ao povo de Angola e ao mundo que é preciso salvaguardar os buelens arquitectónicos que nos foram legados pela história de Angola. A hipocrisia em todo o seu esplendor! Ricardo Manuel deve-se ter voltado mais de uma vez na sua tumba.

Figura 4.2: Exemplo da página HTML

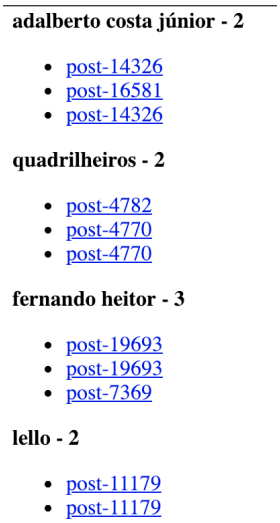


Figura 4.3: Exemplo de página de tags e respetivos hiperlinks

Capítulo 5

Conclusão

Com a realização deste trabalho foi possível apronfudar os conhecimentos em FLex que havíamos adquirido ao longo das aulas teóricas e práticas. Foi um desafio mais complexo, mas que consideramos que foi superado.

Quanto aos pontos fracos do trabalho realizado, consideramos que poderíamos lidar de uma forma mais elegante com a existência de IDs repetidos, ao invés de reescrever o ficheiro já existente. Para além disso, também poderíamos ter tido em consideração as palavras acentuadas, mas isso seria trabalho extra acessório e que não acrescenta muito em termos de conteúdo.

Qunto aos seus pontos fortes, pensamos que cumprimos na totalidade com os requisitos definidos no enunciado, de forma não muito complexa.

Apêndice A

Código do Programa

```
    %{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gmodule.h>
 // Hash table com tag as key e e número e TagValue
GHashTable * tagCount = NULL;
 // Lista com tags de cada artigo
GSList * tags = NULL;
 // Lista com categorias de cada artigo
GSList * categories = NULL;
char * idPost = NULL;
char * title = NULL;
char * date = NULL;
char * text = NULL;

typedef struct tagValue {
int count;
GSList * idPosts;
} * TagValue;
int textSize = 0;

void printCategoriaInNorm(void * tag, void * data) {
fprintf((FILE *) data, "<category>%s</category> ", (char *) tag);
}

void printTagInNorm(void * tag, void * data) {
fprintf((FILE *) data, "<tag>%s</tag> ", (char *) tag);
}

void printTaginHTML(void * tag, void * data) {
fprintf((FILE *) data, "<li>%s</li>\n",(char * )tag);
}
```

```
void printCategoriesInHTML(void * category, void* data){
fprintf((FILE *)data, "<li>%s</li>\n",(char *)category);
}

void htmlInit(FILE * f) {
fprintf(f,"<html><head><meta charset='UTF-8'/> </head><body>");
}

void formatPrintNorm() {
char * fName = malloc(sizeof(char)*512);
sprintf(fName,"%s.norm",idPost);
FILE * normalized= fopen(fName,"w");
fprintf(normalized,"<pub id=\"%s\">\n",idPost);
fprintf(normalized,"\t<title>%s</title>\n",title);
fprintf(normalized,"\t<author_date>%s</author_date>\n",date);
fprintf(normalized,"\t<tags>\n\t\t");
g_slist_foreach(tags,printTagInNorm,normalized);
fprintf(normalized,"\n\t</tags>\n");
fprintf(normalized,"\t<categories>\n\t\t");
g_slist_foreach(categories,printCategoriaInNorm,normalized);
fprintf(normalized,"\n\t</categories>\n");
fprintf(normalized,"\t<text>\n");
fprintf(normalized,"%s\n",text);
fprintf(normalized,"\t</text>\n");
fprintf(normalized,"</pub>\n");
fclose(normalized);
free(fName);
}
void formatPrintHTML(){
// now we create the file with html
char * fileName = malloc(sizeof(char)*512);
sprintf(fileName,"%s.html",idPost);
FILE * postHtml = fopen(fileName,"w");
htmlInit(postHtml);
fprintf(postHtml,"<h1>%s</h1>\n",idPost + 5);
fprintf(postHtml,"<h2>%s</h2>\n",title);
fprintf(postHtml,"<b>Author - Date: </b>%s\n",date);
fprintf(postHtml,"<p></p><b>Tags:</b><ul>\n");
g_slist_foreach(tags,printTaginHTML,postHtml);
fprintf(postHtml,"</ul>\n");
fprintf(postHtml,"<p></p><b>Categoria:</b><ul>\n");
g_slist_foreach(categories,printCategoriesInHTML,postHtml);
fprintf(postHtml,"</ul>\n");
fprintf(postHtml,"<p>%s</p>",text);

fclose(postHtml);
free(fileName);
```

```
}

void freeString(void * string) {
free((char *)string);
}

void addIdsToHash(void * tag, void * id) {
TagValue value = g_hash_table_lookup(tagCount,(char *) tag);
if (value)
value->idPosts = g_slist_prepend(value->idPosts,(char *)id);
}

%}

%x PUB TAG ID CATEG TITLE DATE TEXT
dig [0-9]


%%
\<pub\>{ BEGIN PUB; }

<PUB>\<\/pub\>{ BEGIN INITIAL; }
<PUB>#TAG:  { tags = NULL; BEGIN TAG; }
<PUB>#DATE:" "\[[^\]]+\]" " { BEGIN DATE; }
<PUB>#ID: { idPost = NULL; BEGIN ID;}
<PUB>.|\n {;}

<TAG>#ID: { BEGIN ID; }
<TAG>tag:\{[^}]+\} {
yytext[yyleng-1]=0;
char* tag_name = strdup(yytext+5);
TagValue value = g_hash_table_lookup(tagCount,tag_name);
if (value) { // if already exists, increase count
value->count = (value->count)+1;
}
else { // else create new entry in hash count
value = (TagValue) malloc(sizeof(struct tagValue));
value->count = 1;
value->idPosts = NULL;
g_hash_table_insert(tagCount,tag_name,value);
}
tags = g_slist_prepend(tags,tag_name);
}
<TAG>.|\n  {;}

<ID>\}\n { categories = NULL; BEGIN CATEG; }
<ID>\{post-{dig}+ {
idPost=strdup(yytext+1);
```

```
// add ID to each tag
g_slist_foreach(tags,addIdsToHash,idPost);}
<ID>. {;}

<CATEG>\n\n { BEGIN TITLE; }
<CATEG>[A-Z][^A-Z\n ]+ {
char * category = strdup(yytext);
categories = g_slist_prepend(categories,category);
}
<CATEG>.|\n  {;}

<TITLE>\n {BEGIN PUB;}
<TITLE>[^\n]+ {title = strdup(yytext);}

<DATE>\n {
if (text) {
free(text);
}
text = malloc(sizeof(char)*1024);
text[0] = '\0';
textSize = 1024;
BEGIN TEXT;
}
<DATE>.+ {date = strdup(yytext);}

<TEXT>\nPartilhe" "este" "Artigo\n { formatPrintNorm();formatPrintHTML();BEGIN PUB;}
<TEXT>\nEtiquetas:     { formatPrintNorm();formatPrintHTML();BEGIN PUB;}
<TEXT>\<\/pub\>{formatPrintNorm();formatPrintHTML();BEGIN INITIAL;}
<TEXT>^\[.*\] {;}
<TEXT>.*|\n {
int size = strlen(text)+yyleng+1;
if (size > textSize) {
char * newText = malloc(sizeof(char)*textSize*2);
strcpy(newText,text);
strcat(newText,yytext);
free(text);
text = newText;
textSize *= 2;
}
else {
strcat(text,yytext);
}
}
.|\n  {;}
%%

int yywrap() {
return 1;
```

```
}

void printPostIdInTagIndex(void * id, void * file) {
fprintf((FILE *)file,"<li><a href=\"%s.html\">%s</a></li>\n",(char *)id, (char *)id);
}

void printTagIndex(void * key, void * value, void * file) {
fprintf((FILE *)file,"<p><b>%s - %d</b></p>\n<ul>",(char *) key, ((TagValue) value)->count);
g_slist_foreach(((TagValue) value)->idPosts,printPostIdInTagIndex,file);
fprintf((FILE *)file,"</ul>\n");
}

void generateIndex() {
FILE * tagsIndexFile = fopen("tags.html","w");
g_hash_table_foreach(tagCount, printTagIndex, tagsIndexFile);
}

int main(int argc, char * argv[]) {
tagCount = g_hash_table_new(g_str_hash,g_str_equal);
yylex();
generateIndex();
return 0;
}
```