Processamento de Linguagens e Compiladores (3º ano de LCC)

Trabalho Prático 1 - Enunciado 3

Relatório de Desenvolvimento do Grupo 2

Bruna Araújo	Daniel Ferreira	Ricardo Cruz
(a84408)	(a85670)	(a86789)

7 de dezembro de 2020

Resumo

Numa sociedade contemporânea observamos variadíssimas mudanças que afetam a vida e a interação de todos os indivíduos, nomeadamente o uso das vastas redes de comunicação digital que veiculam toda a espécie de notícias, informações, imagens... Estas têm influenciado os padrões das relações sociais pela sua vastidão, subjetividade e, por vezes, dissonância. A elaboração e implementação de mecanismos com capacidade para filtrar esta panóplia de notícias, oferecendo benefícios para todos, constitui-se como uma ferramenta fulcral para os utilizadores.

O presente trabalho surge com o intuito de explorar a construção de um filtro de texto e entender a sua importância para o processamento e recolha de todos os dados importantes para o estudo sócio linguístico pretendido.

Para o desenvolvimento desta investigação foi utilizado o gerador *FLex*, que nos permitiu retirar os dados de ficheiros *HTML* que contêm os comentários relativos a notícias do jornal *DailyExpress*.

O principal desafio foi definir as expressões regulares adequadas ao que pretendíamos.

Este estudo permitiu-nos trabalhar pela primeira vez com o gerador *FLex*, e perceber os benefícios para o utilizador.

Palavras-Chave: Filtro de texto, gerador *FLex*, sócio-linguístico, ficheiros *HTML*, expressões regulares, utilizador

Conteúdo

Capítulo 1

Introdução

1.1 Contextualização

Os ficheiros $HTML$ contêm centenas de comentários de vários utilizadores.
Cada comentário é composto por:

- **ID** do comentário
- **Username** de quem comenta
- **Data** do comentário
- **Timestamp** "NA"por default.
- **Texto** do comentário
- **Likes** número de likes por comentário
- **hasReplies** que indica se tem ou não respostas
- **NumberOfReplies** que mostra o número de respostas ao comentário.
- **Replies** lista de CommentThreads com as respostas ao comentário

1.2 Desafios

Numa primeira fase, tivemos de definir expressões regulares adequadas e verificar que o filtro apenas retinha a informação que interessava.
De seguida, conferimos se a informação filtrada foi colocada nos campos corretos da estrutura.
Por último, verificamos se os dados estavam a ser escritos corretamente no ficheiro *dados.json*.

1.3 Decisões

De forma a atribuir os tipos de dados (*GString **) adequados à estrutura, na nossa óptica, recorremos à biblioteca *Glib*.

Implementamos uma *CommentThread* principal (*CommentThread* fst) que contém todos os comentários de uma dada página **HTML**. Foi também implementada *CommentThread* secundária (*CommentThread* curr) que vai percorrer comentário a comentário, com vista a colocar comentários e respostas correspondentes de forma ordenada e correta.

Ambas as estruturas são inicializadas no *filtro.l*.

De modo a facilitar o filtro da informação pretendida, utilizamos *Start Conditions* (ID, USERNAME, DATE, TEXT, LIKES).

Por fim, temos uma variável *int resposta* que nos vai auxiliar a perceber se os dados que estamos a tratar pertencem a uma resposta ou a um comentário. Para isso, a função *giveThread()* indica-nos, com base nessa variável, se o conteúdo adicionado à estrutura principal é um comentário ou uma resposta.

Capítulo 2

Concepção/desenho da Resolução

2.1 Estruturas de Dados

No campo *hasReplies* consideramos apenas os valores "1"ou "0", caso um comentário tenha ou não reposta, respetivamente. Consideramos os *likes* e os *numberOfReplies* como inteiros.

Para as *replies* implementamos uma lista de *CommentThreads*.

E, os restantes atributos são *GString* porque em *C* não há *strings*.

```
typedef struct commentThread {
    GString * id;
    GString * user;
    GString * date;
    GString * timestamp; //NA
    GString * text;
    int likes;
    int hasReplies;
    int numberOfReplies;
    CommentThread replies[];
}*CommentThread;
```

Figura 2.1: Estrutura de dados *CommentThread*

2.2 Expressões Regulares

O método que decidimos utilizar foi, ao encontrar uma *Tag* que nos interessava, recorremos às *Start Conditions* para mudar de estado no filtro. De seguida, ignorámos tudo aquilo que não nos interessava para, no fim, estar no filtro, apenas, a informação necessária ao preenchimento da estrutura.
Preenchido o campo da estrutura, correspondente à *Tag*, fizemos *BEGIN INITIAL* para o filtro voltar ao estado inicial e procurar a restante informação.

2.2.1 *Tag Única*

Ao analisar o código *HTML* reparamos que há, sempre, uma *tag* única em cada ficheiro. Por isso, definimos uma expressão regular para, ao ler essa *tag* (aria-label=”List of Comments”), ser inicializada a estrutura principal (*CommentThread* fst).

```
(aria)\-(label)\=\"(List)[' '](of)[' '](Comments)\"                    { fst = newCommentThread(); }
```

Figura 2.2: *Tag* principal

2.2.2 *Tag ID*

A *tag* com a informação do *ID* (data-message-id). Ao encontrar este padrão, é iniciada a *Start Condition ID*.

```
(data)\-(message)\-(id)              { curr = addnewComment(giveThread()); BEGIN ID; }
<ID>\>                               { BEGIN INITIAL; }
<ID>[ ]                              { ; }
<ID>\"                               { ; }
<ID>\=                               { ; }
<ID>[\t\n\r]+                        { ; }
<ID>data\-spot\-im\-class            { ; }
<ID>message\-view                    { ; }
<ID>.                                { setID(curr,yytext); }
```

Figura 2.3: *Tag ID*

2.2.3 *Tag Username*

A *tag* com a informação do *Username* (spcv_username...). Ao encontrar este padrão, é iniciada a *Start Condition USERNAME*.

```
(spcv)\_(username)\"\>                    { ; }
(spcv)\_(username)\"[^>]*\>               { BEGIN USERNAME; }
<USERNAME>\<\/(span)\>                   { BEGIN INITIAL; }
<USERNAME>\n                             { ; }
<USERNAME>.                              { setUser(curr,yytext); }
```

Figura 2.4: *Tag Username*

2.2.4 *Tag DATE*

A *tag* com a informação do *DATE* (<time...). Ao encontrar este padrão, é iniciada a *Start Condition DATE*.

```
\<(time)[^>]*\>      { BEGIN DATE; }
<DATE>\<             { BEGIN INITIAL; }
<DATE>.              { setDate(curr,yytext); }
```

Figura 2.5: *Tag DATE*

2.2.5 *TIMESTAMP*

De acordo com o enunciado, o campo *TIMESTAMP* da estrutura é preenchido, sempre, com "NA".

2.2.6 *Tag DELETED*

A *tag* com a informação do *DELETED* (class=”spcv_is-deleted”). Ao encontrar este padrão, é iniciada a *Start Condition DELETED*.

```
(class)\=\"(spcv)\_(is)\-(deleted)\"\>                { BEGIN DELETED; }
<DELETED>\<(span)\>                                  { ; }
<DELETED>[\n\r]*                                     { ; }
<DELETED>[ ][ ]+                                     { ; }
<DELETED>\<\/(span)\>                                { BEGIN INITIAL; }
<DELETED>.                                           { setText(curr,yytext); }
```

Figura 2.6: *Tag Deleted*

2.2.7 *Tag TEXT*

A *tag* com a informação do *TEXT* (data-spot-im-class=”message-text”). Ao encontrar este padrão, é iniciada a *Start Condition TEXT*.

```
(data)\-(spot)\-im\-(class)\=\"(message)\-(text)\"    { BEGIN TEXT;}
<TEXT>\<\/(div)\>                                    { BEGIN INITIAL; }
<TEXT>\<\/*strong\>                                  { ; }
<TEXT>\>                                             { ; }
<TEXT>[\t\n\r]+                                      { setText(curr," "); }
<TEXT>[ ]{2}                                         { ; }
<TEXT>\<[^>]*\>                                      { ; }
<TEXT>.                                              { setText(curr,yytext); }
```

Figura 2.7: *Tag TEXT*

2.2.8 *Tag LIKES*

A *tag* com a informação do *LIKES* ("spcv_number-of-votes"). Ao encontrar este padrão, é iniciada a *Start Condition LIKES*.

```
\"(spcv)\_(number)\-(of)\-(votes)\"      { BEGIN LIKES; }
<LIKES>\<\/(span)\>                      { curr = getCurrentReply(fst); BEGIN INITIAL; }
<LIKES>\>                                { ; }
<LIKES>[0-9]+                            { setLikes(curr,yytext);}
<LIKES>.                                 { ; }

.|\n                                     { ; }
```

Figura 2.8: *Tag LIKES*

2.2.9 *Tag Replies e Tag Final*

A *tag* (<ul class="spcsv_children-list">) indica-nos que vai haver uma ou mais respostas ao comentário que estamos a analisar e, por isso, actualizamos o valor da *flag resposta* para 1, de modo a sabermos que vamos passar a trabalhar com respostas a um comentário.
Na *tag* (< /ul>) temos dois casos possíveis:

1. Pode estar a ser fechada uma lista de respostas, por isso, actualizamos o valor da resposta para 0 e fazemos *curr = getCurrentReply(fst)* para regressar a um comentário e procurarmos pelo próximo.

2. Pode estar a ser fechada a *tag única*, acima descrita. Dessa forma, sabemos que não há mais repostas nem comentários a analisar e, por isso, ao fazermos *curr = getCurrentReply(fst)* não vai haver nenhum comentário anterior para "regressar".

```
\<(ul)[' '](class)\=\"(spcv)\_(children)\-(list)\"\>      { resposta = 1; }
\<\/(ul)\>                                               { resposta = 0; curr = getCurrentReply(fst); }
```

Figura 2.9: *Tag Replies e Tag Final*

Capítulo 3

Testes

3.1 Testes realizados e Resultados

Observação: Perante os resultados obtidos, o comentário principal da Figura 3.1 contém um comentário "This message is deleted."pois o comentário foi apagado pelo sistema.

```
[

{
"id" : "sp_9LMINbK9_1165460_c_cfjlPd",
"user" : "",
"data" : "",
"timestamp" : "NA",
"comment" : "This message was deleted.",
"numberOfLikes" : 0,
"hasReplies" : 1,
"numberOfreplies" : 2,
"reply" : [
{
"id" : "sp_9LMINbK9_1165460_c_cfjlPd_r_ZUxhy4",
"user" : "kimboy1",
"data" : "14Â Aug",
"timestamp" : "NA",
"comment" : "It's the melt down of it's users that makes me howl with laughter.",
"numberOfLikes" : 5,
"hasReplies" : 0,
"numberOfreplies" : 0,
"reply" : [ ]


},

{
"id" : "sp_9LMINbK9_1165460_c_cfjlPd_r_EKPQAq",
"user" : "CJjj1",
"data" : "14Â Aug",
"timestamp" : "NA",
"comment" : "Only because of the people that use it",
"numberOfLikes" : 6,
"hasReplies" : 0,
"numberOfreplies" : 0,
"reply" : [ ]


}
]


},
```


Figura 3.1: Parte do ficheiro de dados *output* do ficheiro `DailyExpress_extraction_english_comments_14.html`

Capítulo 4

Conclusão

Após a realização deste trabalho, conseguimos desenvolver as nossas capacidades de trabalhar com expressões regulares e com o *FLex*.

As nossas maiores dificuldades foram perceber qual era o padrão e a sua expressão regular correspondente a cada campo da estrutura a preencher, perceber o melhor algoritmo para percorrer comentários e respostas e a forma como iríamos alocar memória para criar e preencher novas *CommentThreads*.

Um dos aspetos, que poderíamos ter melhorado era a eficiência do nosso algoritmo para ficheiros *HTML* de grandes dimensões.

Para concluir, tendo em conta a análise feita aos resultados obtidos, os objetivos estipulados foram cumpridos.

Capítulo 5

Ficheiro filtro.l

```
%{
/* Declaracoes C diversas */
#include "commentThread.h"
#include <stdio.h>
#include <glib.h>

CommentThread fst;
CommentThread curr;
int resposta;

CommentThread giveThread(){
    if(resposta == 0){
        return fst;
    }else{
        return curr;
    }
}

%}
%x ID USERNAME DATE TEXT LIKES DELETED
%%

(aria)\-(label)\=\"(List)[' ](of)[' '](Comments)\"    { fst = newCommentThread(); }
\<(ul)[' '](class)\=\"(spcv)\_(children)\-(list)\"\>  { resposta = 1; }
\<\/(ul)\>                                  { resposta = 0; curr = getCurrentReply(fst); }

(data)\-(message)\-(id)         { curr = addnewComment(giveThread()); BEGIN ID; }
<ID>\>                          { BEGIN INITIAL; }
<ID>[ ]                         { ; }
<ID>\"                          { ; }
<ID>\=                          { ; }
<ID>[\t\n\r]+                   { ; }
<ID>data\-spot\-im\-class       { ; }
<ID>message\-view               { ; }
<ID>.                           { setID(curr,yytext); }


```

```
(spcv)\_(username)\"\>          { ; }
(spcv)\_(username)\"[^>]*\>     { BEGIN USERNAME; }
<USERNAME>\<\/(span)\>          { BEGIN INITIAL; }
<USERNAME>\n                   { ; }
<USERNAME>.                    { setUser(curr,yytext); }

\<(time)[^>]*\>       { BEGIN DATE; }
<DATE>\<              { BEGIN INITIAL; }
<DATE>.               { setDate(curr,yytext); }

(class)\=\"(spcv)\_(is)\-(deleted)\"\>                    { BEGIN DELETED; }
<DELETED>\<(span)\>                                       { ; }
<DELETED>[\n\r]*                                          { ; }
<DELETED>[ ][ ]+                                          { ; }
<DELETED>\<\/(span)\>                                     { BEGIN INITIAL; }
<DELETED>.                                                { setText(curr,yytext); }

(data)\-(spot)\-im\-(class)\=\"(message)\-(text)\"        { BEGIN TEXT;}
<TEXT>\<\/(div)\>                                         { BEGIN INITIAL; }
<TEXT>\<\/*strong\>                                       { ; }
<TEXT>\>                                                  { ; }
<TEXT>[\t\n\r]+                                           { setText(curr," "); }
<TEXT>[ ]{2}                                              { ; }
<TEXT>\<[^>]*\>                                           { ; }
<TEXT>.                                                   { setText(curr,yytext); }

\"(spcv)\_(number)\-(of)\-(votes)\"    { BEGIN LIKES; }
<LIKES>\<\/(span)\>                    { curr = getCurrentReply(fst); BEGIN INITIAL; }
<LIKES>\>                              { ; }
<LIKES>[0-9]+                          { setLikes(curr,yytext);}
<LIKES>.                               { ; }

.|\n                                   { ; }

%%

int yywrap(){
  return(1);
}

int main(){
  yylex();

  openFile("dados.json");

  formatToJsonHead(fst);

  return 0;
}
```