

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

**Trabalho Prático 1 - Enunciado 3**

Relatório de Desenvolvimento do Grupo 2

Bruna Araújo  
(a84408)

Daniel Ferreira  
(a85670)

Ricardo Cruz  
(a86789)

13 de novembro de 2020

## Resumo

Este trabalho consiste na construção de um filtro de texto, recorrendo ao gerador *FLex*, que vai servir para o processamento e recolha de todos os dados importantes para o estudo sócio-linguístico pretendido. Dados esses, foram retirados de ficheiros *HTML* que contêm os comentários relativos a notícias do jornal *DailyExpress*.

Os principais desafios foram definir as expressões regulares adequadas ao que pretendíamos e trabalhar, pela primeira vez, com *FLex*.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Contextualização . . . . .	2
1.2	Desafios . . . . .	2
1.3	Decisões . . . . .	3
<b>2</b>	<b>Concepção/desenho da Resolução</b>	<b>4</b>
2.1	Estruturas de Dados . . . . .	4
2.2	Expressões Regulares . . . . .	5
2.2.1	<i>Tag Única</i> . . . . .	5
2.2.2	<i>Tag ID</i> . . . . .	5
2.2.3	<i>Tag Username</i> . . . . .	6
2.2.4	<i>Tag DATE</i> . . . . .	6
2.2.5	<i>TIMESTAMP</i> . . . . .	6
2.2.6	<i>Tag TEXT</i> . . . . .	7
2.2.7	<i>Tag LIKES</i> . . . . .	7
2.2.8	<i>Tag Replies e Tag Final</i> . . . . .	8
<b>3</b>	<b>Testes</b>	<b>9</b>
3.1	Testes realizados e Resultados . . . . .	9
<b>4</b>	<b>Conclusão</b>	<b>10</b>

# 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, recorreremos 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>\n { ; }
<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 TEXT

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

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

Figura 2.6: Tag TEXT

### 2.2.7 Tag LIKES

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

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

Figura 2.7: Tag LIKES

### 2.2.8 *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".

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

Figura 2.8: *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 não contém texto devido ao facto de o comentário ter sido apagado.

```
{
  "id" : "sp_9LMINbK9_1165460_c_cfjLPd"
  "user" : ""
  "data" : ""
  "timestamp" : "NA"
  "comment" : ""
  "Nº likes" : "0"
  "Has replies" : "1"
  "Nº respostas" : "2"
  "reply" : [
    {
      "id" : "sp_9LMINbK9_1165460_c_cfjLPd_r_ZUxhy4"
      "user" : "kimboy1"
      "data" : "14ª Aug"
      "timestamp" : "NA"
      "comment" : "It's the melt down of it's users that makes me howl with laughter."
      "Nº likes" : "5"
      "has replies" : "0"
      "Nº respostas" : "0"
      "reply" : [ ]
    },
    {
      "id" : "sp_9LMINbK9_1165460_c_cfjLPd_r_EKPQAq"
      "user" : "CJjj1"
      "data" : "14ª Aug"
      "timestamp" : "NA"
      "comment" : "Only because of the people that use it"
      "Nº likes" : "6"
      "has replies" : "0"
      "Nº respostas" : "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*. Para concluir, tendo em conta a análise feita aos resultados obtidos, cumprimos os objetivos estipulados.