TP1 - Processamento de Linguagens

Paulo Lima

2 de Abril de 2020

Resumo

O primeiro trabalho prático no âmbito da unidade curricular Processamento de Linguagens consistiu na elaboração de um projeto usando *Flex* que converte o conteúdo de um ficheiro *HTML* para o formato *JSON*.

Deste modo, no presente relatório, é explicada a forma como foi desenvolvido os filtros *Flex* que permitiram recolher os dados do ficheiro em *HTML* com vários comentários e gerar o ficheiro *JSON*, para cada pedaço de informação são capturados padrões presentes antes dos dados, de forma a conseguir imprimir corretamente todos os resultados para o ficheiro *JSON*.

Os principais objetivos deste trabalho prático foram aumentar a experiência de uso do ambiente Linux e de algumas ferramentas de apoio á programação e aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases. Para além disso, teve também como objetivo desenvolver, a partir de ERs, sistemática e automaticamente Processadores de Linguagens Regulares, que filtrem ou transformem textos com base no conceito de regras de produção Condiçao-Ação e utilizar o *Flex* para gerar filtros de texto em *C*.

Conteúdo

1 Introdução

Área: Processamento de Linguagens

1.1 Enquadramento e Contexto

O enunciado 2.4 Transformador Publico2NetLang contém diversos de comentários. Cada um desses comentários contém um id, nome de utilizador (user), data (date), marcação do tempo (timestamp), texto do comentário (commentText), e respostas a outros comentários (replies), sendo que a tag likes não foi encontrada nos ficheiros de teste e as tag hasReplies e numberOfReplies derivam da informação recolhida nas respostas a comentários.

Assim, o projeto consiste em extrair de um ficheiro input do Jornal O Publico em formato *HTML* toda a informação pretendida e organizá-la e apresentá-la no ficheiro *JSON* de forma a que possa ser facilmente consultada por qualquer pessoa.

No presente relatório é explicada a forma como cumprimos esse requisito e como geramos o respetivo ficheiro *JSON* com toda a informação, comentários e tags, de forma clara, percetível e apelativa.

1.2 Problema e Objetivo

Numa primeira fase, o problema passa por limpar e normalizar todos esses comentários contidos na notícia do Jornal. Em seguida, é necessário criar uma lista dos comentários encontradas no ficheiro *HTML*, com o respetivo número de respostas que cada comentário tem.

Após construída essa lista, que deverá ter como conteudo uma "ThreadSection", o objetivo é colocar cada um dos parametros da estrutura usada no formato pedido de modo a que, quando aberto o ficheiro, seja apresentado um novo ficheiro *JSON* contendo o "commentThread"com os diversos comentários e as suas tags.

Consequentemente, cada um comentários deverá também conter dentro da tag "replies"todas as possýeis respostas, de modo a perceber a ligação entre os vários comentários presentes no ficheiro final em formato *JSON*.

1.3 Decisões tomadas

Uma vez que o ficheiro *HTML* a analisar, é muito extenso, foi decidido recorrer à biblioteca GLib de modo a construir eficientemente as estruturas de dados com os tipos de dados apropriados e para, consequentemente, ser possível utilizar as funções que permitem manipular corretamente esses tipos de dados.

Nas estruturas de dados que foram construídas podem ser guardadas todas as tags presentes num comentário do Jornal O Publico.

1.4 Estrutura do Relatório

O presente relatório está dividido em 5 diferentes capítulos.

No capítulo 1, Introdução, é feito um enquadramento e contextualização do trabalho prático e, em seguida, é feita uma descrição do problema. Por fim, é evidenciado um pequeno ponto no que diz respeito às decisões tomadas.

No capítulo 2, Análise e Especificação, é feita uma descrição informal do problema e uma especificação dos requisitos necessários para uma correta resolução do problema, relevando quais serão as duas grandes tarefas do projeto.

Em seguida, no capítulo 3, Concepção e desenho da Resolução são expostas as estruturas de dados utilizadas e é feita uma descrição detalhada de todo o desenvolvimento do projeto até se obter a solução final. Para além disso, são apresentados máquinas de estados que representam as condições de contexto dos nossos filtros e as expressões regulares que desencadeiam as transições entre esses estados.

Posteriormente, no capítulo 4, Codificação e Testes, são apresentados alguns testes realizados e os resultados obtidos.

Por fim, no capítulo 5, Conclusão, termina-se o relatório com uma síntese e análise do que trabalho realizado.

2 Análise e Especificação

2.1 Descrição informal do problema

Dado um ficheiro *HTML* bastante extenso com várias comentários, é necessário conseguir extrair de cada comentário as suas tags e construir o ficheiro *JSON* a partir dos mesmos, indicando os identificadores (id) do respetivo comentário. Também é necessário extrair os próprios comentários de forma normalizada e a lista de todas as tags encontradas. Toda esta informação tem que ser retirada do extenso ficheiro e formatada para o ficheiro *JSON*.

2.2 Especificação dos Requisitos

2.2.1 Transformer

A etapa do transformer, escreve no descritor de saída todos os comentários, ordenados pela mesma ordem que a do ficheiro original. Mais precisamente, é contruída uma sequência de comentários no formato *JSON*, que após a leitura do ficheiro original são devidamente formatados e de seguida são enviados para o descritor de saída, de modo a obter o ficheiro transformado pretendido.

3 Concepção/desenho da Resolução

3.1 Estruturas de Dados

De forma a conseguir armazenar temporariamente a informação de um comentário foi necessário construir uma estrutura de dados que permitisse guardar em diferentes variáveis o id do comentário, o utilizador, a data, o timestamp, o texto do comentário, o numero de likes, se o comentário tem ou não respostas, o número de respostas, e uma lista com as respostas que este possa ter.

Para isso, foi contruída a seguinte estrutura de dados, **ThreadSection**, que tem todas as variáveis necessárias para guardar a informação de um comentário.

```
/* NetLang.h */
typedef struct threadSection * ThreadSection;
```

```
/* NetLang.c */
struct threadSection {
    GString * id;
    GString * user;
    GString * date;
    GString * timestamp;
    GString * commentText;
    gint likes;
    gboolean hasReplies;
    gint numberOfReplies;
    GSList * replies;
};
```

Tanto o *id*, como o *user*, a *date*, o *timestamp* e o *commentText* são strings, ou seja, simples conteúdo textual.

No caso das *replies*, uma vez que existe a necessidade de mais tarde ser necessário delimitar as mesmas, o tipo que as armazena é uma *GSList*, para que possam estar divididas e esses dados posteriormente serem mais facilmente tratados.

Para além da necessidade de uma estrutura que permitisse que as diferentes informações de um comentário estivesse bem organizado em diferentes variáveis, surgiu também a necessidade de armazenar todas estes comentários que iam sendo encontrados.

```
/* NetLang.c */
GSList * threadSList = NULL;
```

Deste modo, foi construída uma variável global do tipo *GSList* intitulada *threadSList*, com o objetivo de guardar todas as *ThreadSection* existentes num ficheiro *HTML* válido para o projeto.

A decisão de uma *GSList* deveu-se à necessidade de manter a ordenação dos comentários processados e das suas respostas.

3.2 Implementação

3.2.1 Transformer

Deste modo, foi criado um filtro de texto Flex, chamado transformer.l, que com base nas regras de produção Condição-Ação com expressões regulares vai construindo o resultado pretendido.

Impressão do Comentário

Analisando o input disponibilizado (Publico_extraction_portuguese_comments_4.html), é possível verificar que um comentário começa com o atributo $< li >$ onde está contido o (id), de seguida caso não hajam respostas, seguem-se o nome de utilizador (user) iniciado pelo atributo $< a >$, o (timestamp) com o atributo $< time >$, a data (date) iniciado também com $< a >$, e por fim o texto do comentário (commentText) que é iniciado com $< p >$.

Desta forma, é possível definir as condições de contexto necessárias: ID, USER, DATE, TIMESTAMP, COMMENT, TCOMMENT, TEXT e EMPTYTEXT, que auxliam no processamento de cada um dos elementos referidos anteriormente.

Deste modo, após estar presente uma ideia mais concreta da forma como se construiu as condições, o passo seguinte consistiu na decisão da forma como iria se representar e guardar cada uma dessas informações para posteriormente se extrair a informação do ficheiro da forma desejada. Para tal, foi utilizada a estrutura de dados **ThreadSection**, que foi anteriormente exposta e que se encontra no ficheiro *NetLang.c*. Esta estrutura de dados permite guardar os dados de cada comentário para depois serem impressos corretamente.

Posto isto, estão disponíveis todas as ferramentas necessárias para começar a construir o filtro *Flex*. Tal como constatado ao analisar o ficheiro input, um comentário começa por um $< li >$. Logo, no filtro foi colocada essa condição e, quando aparece essa expressão é contruída uma instância de **ThreadSection**.

```
/* NetLang.c */
thread_section = initThread();
```

Consequentemente, ao longo da construção do filtro, foram criadas ações que visam preencher corretamente a estrutura de dados **ThreadSection** (variável thread_section). Uma vez que o funcionamento e o raciocínio por detrás de cada uma das ações é semelhante vão ser demonstradas apenas duas situações.

No caso do (id), por exemplo, o input é lido e limpo, de forma a facilitar a inserção do elemento na instância *threadSection*, após este adição o estado é alterado para *INITIAL*, de modo a que o filtro volte a poder dar match com qualquer uma das condições de contexto existentes. A regra Condição-Ação exposta é a seguinte:

```
<ID>.*                                                  {
                                                            thread_section = initThread();
                                                            yytext = cleanString(yytext,
                                                                yyleng, '\"');
                                                            setId(thread_section, yytext);
                                                            idBuffer = g_string_new(yytext);
                                                            BEGIN INITIAL;
                                                        }

<INITIAL>.*"<li class=\"comment\" data-comment-id=\"" {
                                                            li_open++;
                                                            BEGIN ID;
                                                        }
```

Por exemplo, no caso do conteúdo do comentário, sempre que aparece este encontra um $< p >$ o programa passa a estar no estado COMMENT que passa depois para o estado TCOMMENT, o estado do correto "parse"da informação. Assim, sempre que entramos nesse estado é inicializado um *commentBuffer*, que vai acumulando a conteúdo de texto presente no input. Caso apareça o carácter de mudança de linha significa que a informação da data tem que ser adicionada ao buffer. Quando o seguinte match é encontrado ".*≪/p¿".*"significa que o texto já terminou e, portanto, pode ser adicionado à estrutura de dados. O estado do programa passa a estar na nossa condição de contexto INITIAL.

```
<INITIAL>.*"<p>"[\n\r\t]*    {
                                commentBuffer = g_string_new("");
                                BEGIN COMMENT;
                             }

<TCOMMENT>.*"</p>".*         {
                                setComment(thread_section, commentBuffer->str);
                                _tempList = g_slist_append (_tempList,
                                    (ThreadSection)newThreadSection(thread_section));
                                BEGIN INITIAL;
                             }
<TCOMMENT>."\n"*             {
                                if(yytext[0] == '\"') {
                                   g_string_append_c(commentBuffer,'\\');
                                   g_string_append_c(commentBuffer,yytext[0]);
                                }
                                else if(yytext[0] == '\n' || yytext[0] == '\r' ||
                                    yytext[0] == '\t'){
                                   g_string_append_c(commentBuffer, ' ');
                                }
                                else {
                                   g_string_append_c(commentBuffer,yytext[0]);
                                }
                             }

<COMMENT>[ ]*                { BEGIN TCOMMENT; }
```

Por fim, quando a referência é fechada $< /li >$, o comentário é adicionado á lista de comentários. Caso seja detatado o fim do ficheiro a lista é impressa através da função printJson.

```
<INITIAL>.*"</html>"         { printJson(); }
```

3.2.2 Makefile

Para uma compilação fácil e simples do programa foi criada uma *Makefile*. Desta forma, esta *Makefile* visa compilar o filtro *Flex* e, em seguida, compilar o código **C** resultado do *Flex*. No projeto, é também necessário utilizar o package da biblioteca *GLib* e os includes criados no processo.

Na diretoria principal do trabalho, que contém a src, há uma segunda *Makefile*. Essa *Makefile* é responsável por correr a outra *Makefile*, gerando o executável final *publico2NetLang*.

Devido ao facto de os ficheiros *HTML* do jornal O Publico não estarem formatados de forma equivalente ao necessário em *JSON* é preciso executar um pequeno script que converta os ficheiros no formato mais indicado (*UTF-8*).

```
$ script.sh [pathOrigin] [pathDestiny]
```

Assim, para correr o programa basta escrever make no terminal, na diretoria do trabalho e executar o executável gerado *public2NetLang*, com o ficheiro input desejado.

```
$ public2NetLang < [pathTofile] > [destinePath][destiny_filename].json
```

De modo a completar a documentação existe também um *README* com uma ajuda adicional ao funcionamento do programa.

A *Makefile* também possui a opção de clean, que limpa o projeto removendo os binários.

3.3 Outros extras

De forma a conseguir facilitar o funcionamento do programa, foram criadas duas *Makefiles*, além disto, todo o texto *JSON* produzido segue uma identação ideal para a leitura e compreensão dos comentários e das suas respostas.

Adicionalmente, foi incluída na entrega final na diretoria *results*, os exemplos finais em formato *JSON*. O filtro criado para o efeito contabiliza o número de ocorrências de comentários em cadeia, isto é entre $< li >$ e $< /li >$, de forma a identificar os comentários que são respostas a outros. Esta contagem foi feita através do número de $< li >$ abertos e do número de $< /li >$ fechados identificando assim um comentário.

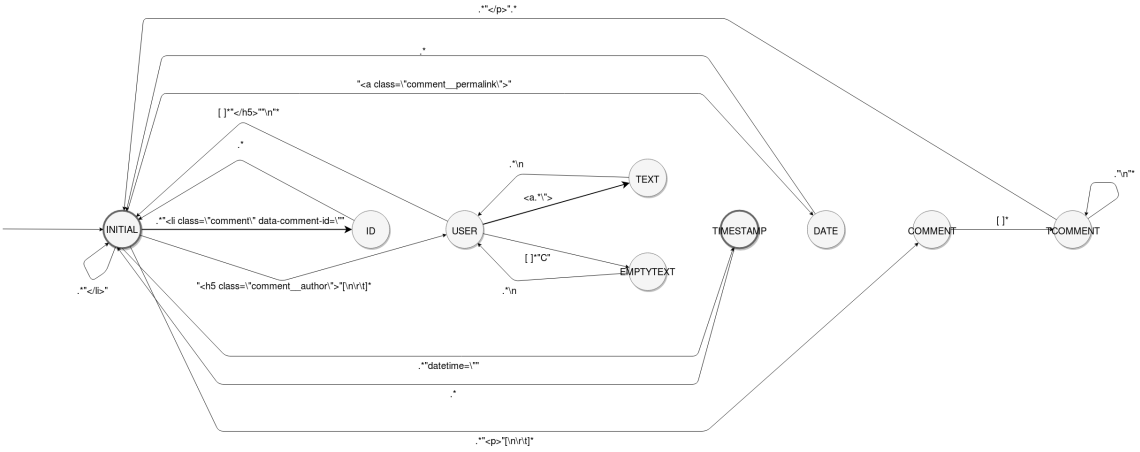

Figura 1: Autómato Transformer

4 Codificação e Testes

4.1 Testes realizados e Resultados

4.1.1 Impressão do Comentário

Dado o seguinte comentário:

```
<li class="comment" data-comment-id="40458a7c-d75f-4bd8-4565-08d7471e83ae">
<div class="comment__inner">
<div class="comment__meta">
<span class="avatar comment__avatar">
<span class="avatar__pad">
<img alt=""
    data-interchange="https://static.publicocdn.com/files/homepage/img/avatar_74x74.png,
    small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, retina]"
    data-resize="vk4i45-interchange" data-t="n17kfk-t" id="vk4i45-interchange"
    src="Publico_extraction_portuguese_comments_4_files/avatar_74x74.png">
</span>
</span>
<h5 class="comment__author">
<a
    href="https://natura.di.uminho.pt/utilizador/perfil/0a3f5416-92c0-4a59-84ad-9eaabff0dc23"
    rel="nofollow">Clarisse Vilanova </a>
</h5>
<span class="comment__reputation comment__reputation-r2" title="Experiente"><i
    aria-hidden="true" class="i-check"></i></span>
<!--<span class="comment__location">Sao Pedro de Aguiao</span>-->
<time class="dateline comment__dateline" datetime="2019-10-03T06:34:14.24">
<a class="comment__permalink">03.10.2019 06:34</a>
</time>
</div>
<div class="comment__content">
<p>
              Depois de se ter transformado num musico mediocre, Abrunhosa infecta de
                  mediocridade tambem as suas opinioes.
          </p>
</div>
</div>
<form class="form comments__form expanded" data-abide="" data-t="15omu8-t"
    style="display:none"></form>
</li>
```

O output textual obtido é o seguinte:

```
{
"id": "40458a7c-d75f-4bd8-4565-08d7471e83ae",
"user": "Clarisse Vilanova ",
"date": "03.10.2019 06:34",
"timestamp": "2019-10-03T06:34:14.24",
"commentText": "Depois de se ter transformado num musico mediocre,
    Abrunhosa infecta de mediocridade tambem as suas opinioes. ",
"likes": "0",
"hasReplies": false,
"numberOfReplies": "0",
"replies": []
},
```

O output JSON do código acima apresentado é o presente na Figura[2].

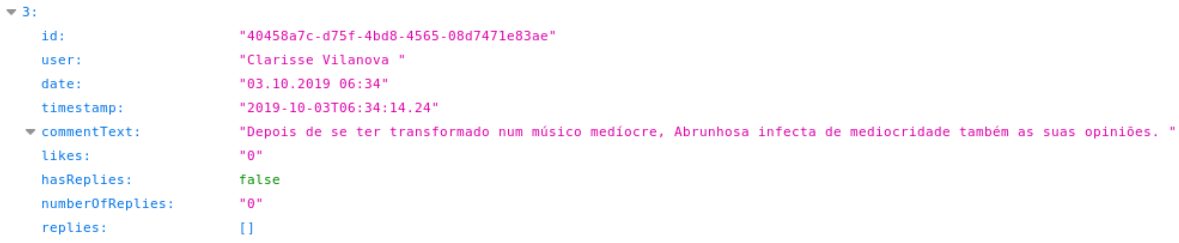

Figura 2: Exemplo de Comentário

4.1.2 Impressão da Lista de Comentários

Dada o seguinte sequência de comentários:

```
<li class="comment" data-comment-id="87499609-7d5b-406e-276e-08d74633975a">
<div class="comment__inner">
<div class="comment__meta">
<span class="avatar comment__avatar">
<span class="avatar__pad">
<img alt=""
    data-interchange="https://static.publicocdn.com/files/homepage/img/avatar_74x74.png,
    small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, retina]"
    data-resize="yqjuh2-interchange" data-t="5ppm38-t" id="yqjuh2-interchange"
    src="Publico_extraction_portuguese_comments_4_files/avatar_74x74.png">
</span>
</span>
<h5 class="comment__author">
<a
    href="https://natura.di.uminho.pt/utilizador/perfil/a2b439dd-2620-4a5b-91b2-7a70bd276049"
    rel="nofollow">Antonio Paulo </a>
</h5>
<span class="comment__reputation comment__reputation-r1" title="Iniciante"><i
    aria-hidden="true" class="i-check"></i></span>
<!--<span class="comment__location"></span>-->
<time class="dateline comment__dateline" datetime="2019-10-02T08:41:38.033">
<a class="comment__permalink">02.10.2019 08:41</a>
</time>
</div>
<div class="comment__content">
<p>
              Touche. Com Rui Rio, Portugal voltaria as corridas de
calhambeque tal como aconteceu em tempos a magnifica cidade do Porto....
          </p>
</div>
</div>
<ol class="comments__list">
<li class="comment" data-comment-id="65fa2e28-76cb-44e2-3b07-08d7471dd7c5">
<div class="comment__inner">
<div class="comment__meta">
<span class="avatar comment__avatar">
<span class="avatar__pad">
<img alt=""
    data-interchange="https://static.publicocdn.com/files/homepage/img/avatar_74x74.png,
    small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, retina]"
    data-resize="q8o5lx-interchange" data-t="kgmjrx-t" id="q8o5lx-interchange"
    src="Publico_extraction_portuguese_comments_4_files/avatar_74x74.png">
</span>
</span>
<h5 class="comment__author">
<a
    href="https://natura.di.uminho.pt/utilizador/perfil/183a23c5-8011-4d66-ada8-271e2d2c08ce"
    rel="nofollow">Paulo Valente </a>
</h5>
<span class="comment__reputation comment__reputation-r1" title="Iniciante"><i
    aria-hidden="true" class="i-check"></i></span>
<!--<span class="comment__location"></span>-->
<time class="dateline comment__dateline" datetime="2019-10-02T11:08:49.893">
<a class="comment__permalink">02.10.2019 11:08</a>
</time>
</div>
```

```
<div class="comment__content">
<p>
                      Que posso dizer? Tenho saudades das corridas de
calhambeques, que tal como hoje encheram de turistas a cidade do Porto.
                  </p>
</div>
</div>
</li>
</ol>
<form class="form comments__form expanded" data-abide="" data-t="cui7d9-t"
    style="display:none"></form>
</li>
</ol>
</body></html>
```

O resultado visual resultante é o apresentado na Figura [3].

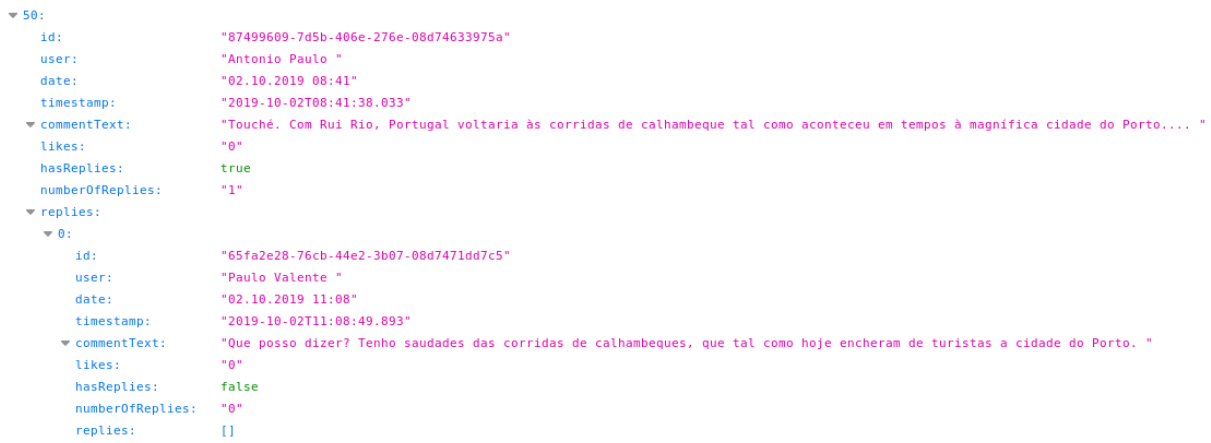

Figura 3: Exemplo de Respostas

5 Conclusão

Através do desenvolvimento deste projeto consegui melhorar os meus conhecimentos relativamente à construção de filtros *Flex*, de expressões regulares e de regras Condição-Ação. A utilidade destes filtros foi realmente notória, uma vez que foi possível a partir de um ficheiro com diversos comentários em formato *HTML*, gerar um ficheiro *JSON* com toda essa informação de forma organizada e apelativa, o que revela que a utilização destes filtros faz realmente a diferença quando o objetivo é apresentar informação, analisar dados, recolher dados, entre muitas outras utilidades.

Abrindo o ficheiro *JSON* no browser, é possível visualizar a todos os comentários de uma forma simples e gráficamente clara. Assim, é bastante mais fácil de identificar comentários e todos os seus parâmetros.

Assim, através deste ficheiro, é possível visualizar alguns comentários e observar as respostas feitas a esse mesmo comentário.

Com este extenso ficheiro seria possível realizar muitos outros filtros que permitiram que os dados presentes no Jornal fossem analisados e apresentados de várias outras formas, fornecendo mais funcionalidades ao utilizador. Por exemplo capturar a informação da localização do utilizador que em certos casos estava disponível para ser processada.

Concluindo, através do formato especificado nos ficheiro *HTML* fornecidos, foi possível cumprir todos os requisitos propostos e construir processador de texto que facilita a perceção da secção de comentários do jornal O Publico.