Universidade do Minho

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Trabalho Prático nº1:
Publico2NetLang

João Nunes (a82300)

Luís Braga (a82088)

Shahzod Yusupov (a82617)

Braga, Portugal
28 de Junho de 2020

Resumo

O presente relatório foi elaborado no âmbito da Unidade Curricular de Processamento de Linguagens do 3ºano do Mestrado Integrado em Engenharia Informática. Este documento regista e fundamenta o trabalho e as decisões tomadas ao longo da elaboração do *Trabalho Prático nº1.*

O trabalho prático consiste, de forma sucinta, em utilizar o *Fast Lexical Analyzer Generator* (*Flex*) em conjunto com expressões regulares para resolver um dos problemas enunciados, mais especificamente, após deliberação do grupo de trabalho, o "*Transformador Publico2NetLang*". Este consiste em gerar, a partir de documentação HTML, documentação em formato json.

Os resultados atingidos vão de encontro com o esperado e resolvem o problema proposto de uma forma que o colectivo de trabalho pensa ser eficiente e robusta.

Conteúdo

1 Introdução **2**
1.1 Transformador Publico2NetLang 2

2 Análise e Especificação **3**
2.1 Enunciado 3
2.2 Descrição do problema 5
2.3 Estratégia de implementação 5

3 Concepção/desenho da Resolução **12**
3.1 Estruturas de dados 12

4 Codificação e Testes **15**
4.1 Alternativas, Decisões e Problemas de Implementação 15
4.2 Testes realizados e resultados 15

5 Conclusão **18**

A Código do filtro de texto (tp1.fl) **19**

Capítulo 1

Introdução

1.1 Transformador Publico2NetLang

O tema proposto enquadra-se na Unidade Curricular de Processamento de Linguagens do 3ºano do Mestrado Integrado em Engenharia Informática. Como tal, o tema abordado ao longo deste relatório consiste em utilizar *Fast Lexical Analyzer Generator* em conjunto com expressões regulares.

O grupo decidiu escolher o enunciado número quatro, de onde se pretendia processar um ficheiro *HTML* para um ficheiro *JSON*.

Estrutura do Relatório

No capítulo 2 introduz-se o que é dito no enunciado e descreve-se o problema. De seguida especifica-se a implementação da solução para o problema dado.

No capítulo 3 são explicadas as estruturas de dados usadas na resolução do problema e o que estas visam resolver.

No capítulo 4 faz-se referência às decisões mais importantes tomadas no decorrer do trabalho, às alternativas pelas quais se podia ter enveredado e imprevistos que surgiram durante a resolução do problema. Ou seja, conta-se, de uma forma geral, como foi o trabalho.

No capítulo 5 dá-se por concluído o relatório com um resumo do que foi dito.

Acrescentam-se ainda apêndices com o código do trabalho e a bibliografia.

Capítulo 2

Análise e Especificação

2.1 Enunciado

O problema consiste em gerar um *JSON* para onde se extraem os dados relevantes dos comentários de notícias publicadas no jornal *O Público*, que vêm no formato *HTML*. Como tal, o ficheiro *JSON* deverá possuir o seguinte formato.

```
"commentThread": [
    {
      "id": "STRING",
      "user": "STRING",
      "date": "STRING",
      "timestamp": NUMBER,
      "commentText": "STRING",
      "likes": NUMBER,
      "hasReplies": TRUE/FALSE,
      "numberOfReplies": NUMBER
      "replies": [ ]
     }, ...
]
```

Contudo, e embora o *output* seja de evidente importância, o ficheiro de *input HTML* também o é uma vez que são aplicadas as expressões regulares sobre este. Como tal, apresenta-se de seguida o formato do ficheiro de *input*.

```
[<h3 class="module__heading module__heading--major"><i aria-hidden="true" class="i-comment">
</i> 85 comentários</h3>]
<ol class="comments__list" id="approved-comments">
<li class="comment" data-comment-id="06de7129-6167-49cd-d330-08d743683e5c">
```

```
<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="yf8gxz-interchange" data-t="vch0wr-t" id="yf8gxz-interchange"
src="https://static.publicocdn.com/files/homepage/img/avatar_74x74.png"/>
</span>
</span>
<h5 class="comment__author">
<a href="/utilizador/perfil/ff148bf7-1a11-479e-a2ed-b66ab9855783"
rel="nofollow">PdellaF </a>
</h5>
<span class="comment__reputation comment__reputation-r1" title="Iniciante">
<i aria-hidden="true" class="i-check"></i>
</span>
<!--<span class="comment__location">Terra</span>-->
<time class="dateline comment__dateline" datetime="2019-10-03T21:11:55.99">
<a class="comment__permalink">03.10.2019 21:11</a>
</time>
</div>
<div class="comment__content">
<p>
Do assunto e de Justiça, Abrunhosa nada percebe.
Não sabe que o MP pronunciou nesta altura porque era esse o prazo?
Não sabe. tenho para mim que quando alguém, dando a sua opinião, inventa cabalas
e teorias da conspiração é porque é cognitivamente débil ou está a soldo de uma facção.
Rui Rio pegou numa câmara municipal corrupta, ineficaz e gerida para pagar
a clientelas partidárias e outros parasitas e transformou-a numa organização
equilibrada. naturalmente que isso não agradou a si nem aos seus amigos
que viviam parasitariamente da CMP.
</p>
</div>
</div>
```

Portanto, através da análise do ficheiro anterior é possível imediatamente verificar que existem algumas *tags html* importantes para o preenchimento do consequente ficheiro *JSON* como por exemplo a *tag <h5 class=”comment__author”>* que identifica posteriormente o nome do utilizador, sendo portanto evidente a existência de uma expressão regular para filtrar o nome do autor. Contudo, este mesmo processo será explicado posteriormente.

2.2 Descrição do problema

Tal como foi dito anteriormente, este problema consiste na transformação dos dados relevantes de um ficheiro *HTML* para um ficheiro *JSON*. Os dados necessários encontravam-se dispersos, havendo bastante informação desprezável.
A partir de uma análise cuidadosa tanto do ficheiro de *input* como de *output* e de modo a também explicar o que é que cada campo significa no ficheiro de *output* segue-se uma explicação individual de cada campo, bem como uma breve explicação de onde são retirados os dados em cada campo.

- id - corresponde ao *data-comment-id* do ficheiro *HTML* sendo portanto o *id* do comentário;
- user - corresponde ao <h5 class="comment__author"> e é o nome do utilizador;
- date - corresponde ao atributo datetime da *tag HTML time* e representa data do comentário;
- timestamp - data produzida aquando da geração do *JSON* que é dada em *UNIX time* (número de millisegundos desde a meia noite de 1 de Janeiro de 1970);
- commentText - corresponde ao atributo class="comment__content"do ficheiro *HTML* e possui o texto do comentário;
- likes - número de *likes* do comentário e como este não se encontra no ficheiro *HTML* disponibilizado, encontrar-se-á sempre inicializado a zero;
- hasReplies - indica se o comentário possui ou não respostas;
- numberOfReplies - indicador com o número de respostas a um dado comentário;
- replies - sub-lista com os campos apresentados anteriormente. Portanto, podemos comcluir que esta estrutura JSON é recursiva.

2.3 Estratégia de implementação

Numa primeira fase o grupo de trabalho encarregou-se de definir as expressões regulares necessárias para filtrar os valores necessários para preencher o ficheiro *JSON*, bem como alguns auxiliares para efetuar o *cleaning* de espaços e entre outros.

```
space [\ \t \n]
decimal  [0-9]
alpha  [a-zA-Z]

comList \<ol\ +class\ *=\ *\"comments__list\"
endOL \<\/ol\>

com \<li\ +class\ *=\ *\"comment\"
endLI \<\/li\>
comListEnd  {space}*{endOL}
otherLI {space}*\<li
```

```
id data\-comment\-id=\"
idValue ({alpha}|{decimal}|\-)+

time \<time
datetime  datetime\ *=\ *\"
datetimeValue ({decimal}|[\-\:\.T])+

nome \<h5\ *class=\"comment__author\"\ *\>{space}*\<a[^\>]+\>
endAnchor \<\/a\>

commentText  \<div\ *class=\"comment__content\"\>{space}*\<p\>
endParagraph \<\/p\>
```

A primeira expressão regular *space* encarrega-se de filtrar os espaços, os *tabs* e as mudanças de linha. A expressão regular *decimal* possui o intuito de filtrar os dígitos de 0 até 9, o *alpha* por sua vez filtra também os caracteres de a até z, incluindo letras maiúsculas.

De seguida o *comList* é a expressão regular responsável por filtrar a classe que identifica o início da lista de comentários. E o *endOL* filtra o fim da lista de comentários.

A expressão regular *com* possui o intuito de filtrar o início do comentário, sendo que o *endLI* por sua vez filtrar o fim do comentário. A expresão regular *comListEnd* e *otherLI* possuem ambas o mesmo intuito de filtrar os espaços, tabs e mudanças de linha antes do fim da lista de comentários ou do início do comentário.

A *id* tal como o nome possui a finalidade de filtrar o identificador do comentário, depois a expressão regular *idValue* filtra o contéudo do identificador que poderá conter caracteres, dígitos sendo estes separados por "-".

A expressão regular *time* filtra a entrada da tag *time* do ficheiro *HTML*. O *datetime* identifica o início do conteúdo do atributo com o mesmo nome. Sendo que o valor do *datetime* poderá conter dígitos sendo estes separados por "-"ou *":"* e poderá também conter *"T"*.

A próxima expressão regular *nome* filtra o início da *tag HTML* responsável por conter o conteúdo relativo ao nome do utilizador que publicou o comentário. O *endAnchor* filtra o fecho da *tag* contendo o nome do utilizador que publicou o comentário.

O *commentText* filtra a tag da divisão, a classe e o início do parágrafo que contém o comentário do utilizador. O *endParagraph* identifica o fecho da *tag* que contém o texto do comentário do utilizador.

Tendo explicado cada uma das expressões regulares utilizadas no filtro de texto implementado pelo grupo, segue-se a explicação dos diversos contextos.

Foram definidos sete contextos que representam os sete estados diferentes do que o programa se pode encontrar.

```
%x COMLIST COM ID TIME DATETIME NAME COMMENTEXT
```

Ou seja, inicialmente mal se filtre o início da lista dos comentários é feito o push para a *stack* do contexto da lista de comentários.

```
{comList} {yy_push_state(COMLIST);}
```

Uma vez nesse contexto *comList*, é possível ou encontrar um comentário ou encontrar o fim da lista de comentários, caso se encontre o fim da lista de comentários é feita a verificação do nível de profundidade da lista de comentários, se for maior que zero elabora o print para o ficheiro de um *"]"* que dita o fim da lista, de seguida decrementa-se a profundidade e é feito o *pop state* que dita o *pop* do contexto da lista de comentários da *stack*, saíndo então deste contexto.

```
{endOL} {
        if(profundidade > 0)
           printFieldStringPreDef("],\n");
        --profundidade;
        yy_pop_state();
}
```

Caso seja feita a filtragem de um comentário, primeiro é verificada a profundidade, caso a profundidade seja maior que zero significa que é um *reply* a um *reply* e adiciona o comentário à estrutura (previamente discutida). Depois é feito o *print* para o *JSON* dos campos que podem ser imediatamente preenchidos que é o *timestamp* e o número de *likes*. Depois é também feito o *push state* e entra no contexto do comentário.

```
{com} {
      if(profundidade > 0){
         addComment(com);
         com = addNivelToComment(com);
      }
  printFieldStringPreDef("{\n");
  printTimestamp();
  printFieldStringPreDef("\"likes\": 0,\n");
  yy_push_state(COM);
 }
```

Prosseguindo para o contexto do comentário, é neste contexto que é feita a maior parte do encaminhamento para os contextos referentes ao preenchimento dos campos do ficheiro *JSON* de *output*.
Ou seja, se dentro do comentário é feita uma filtragem de uma lista de comentários, então significa que existe um *reply* a este comentário, pelo que preenche de imediato o campo *hasReplies* com *true* e prepara o campo *replies* para ser preenchido com os subsequentes *replies* ao comentário. De seguida incrementa a profundidade do comentário, e caso a profundidade seja 1 (devido ao *reply*) cria um nível para o comentário e faz o *push state* do contexto referente à lista de comentários.

```
{comList} {
    printFieldStringPreDef("\"hasReplies\": true,\n");
printFieldStringPreDef("\"replies\": [\n");
++profundidade;
if(profundidade == 1) com = createNivel(com);
yy_push_state(COMLIST);
}
```

Ainda dentro do comentário, é também possível filtrar o identificador do comentário, a *tag html* referente à data de publicação do comentário e o nome do autor do comentário, sendo que nestes três casos é necessário mudar de contexto para os contextos referentes a cada estado.

Caso seja feita a filtragem da *tag html* referente ao comentário através da expressão regular *commentText* então, cria-se o campo no ficheiro *JSON* referente ao texto do comentário e de seguida é feito mais um *push* do contexto referente ao texto do comentário.

```
{commentText} {
   printFieldStringPreDef("\"commentText\": \"");
   yy_push_state(COMMENTEXT);
 }
```

Continuando no contexto do comentário, após filtrar todos os aspetos relativos ao comentário, e os *respetivos push* para os contextos adequados, será filtrado o fim do comentário sendo este identificado pelo fecho da tag li. Sendo necessário também salvaguardar a filtragem dos espaços, *tabs* e mudanças de linha através da expressão regular *otherLI*, no outro caso é feito de maneira análoga mas recorrendo à expressão regular do *comListEnd*. Em ambos os casos é invocada a função *endComment*, que verifica o número de *replies* ao comentário, e caso este número seja maior que zero então imprime para o ficheiro *JSON* o número de comentários adequado. Caso contrário, altera os campos relativos aos *replies*, ou seja o *hasReplies* para falso, o *number of replies* para zero e imprime uma lista vazia para o campo *replies*.

```
if(nReplies > 0) {
   printFieldNum("numberOfReplies", nReplies, 1);
}
else{
   printFieldStringPreDef("\"hasReplies\": false,\n");
   printFieldNum("numberOfReplies", numberOfReplies(com), 0);
   printFieldStringPreDef("\"replies\": []\n");
}
```

No final desta função é feito um *pop* do estado relativo ao comentário da *stack*.

```
{endLI}/{comListEnd} {
        endComment();
        printFieldStringPreDef("}\n");
 }
{endLI}/{otherLI} {
        endComment();
        printFieldStringPreDef("},\n");
 }
```

Passando para os contextos relativos a cada um dos campos do comentário, no contexto *id* já foi identificado o campo do identificador, sendo apenas necessário filtrar o valor do identificador, como tal recorre-se à expressão regular *idValue*. De seguida é feito o *print* para o ficheiro do valor correspondente ao *yytext[0]* que corresponde ao valor do identificador que deu *match* com a expressão regular anterior. O *pop* deste contexto dá-se aquando é identificado o final do valor do identificador.

```
<ID>{
{idValue}        {printFieldString("id", yytext, 0);}
\"                       {yy_pop_state();}
}
```

No contexto referente à data do comentário, caso a expressão regular referente à classe *datetime* dê *match* então é feito o *push* para a *stack* do contexto referente ao filtro do valor da data de publicação do comentário. Após ser feito o *match* com o fim da *tag* referente ao tempo é feito o *pop* deste estado, voltando ao estado do comentário.

```
<TIME>{
{datetime}        {yy_push_state(DATETIME);}
\>                        {yy_pop_state();}
}
```

A filtragem propriamente dita do campo referente à data de publicação do comentário é feita através da expressão regular *datetime*, para o ficheiro é impresso o resultado do *match* do *yytext[0]*. É feito o *pop* deste contexto quando se dá *match* com o final da data de publicação.

```
<DATETIME>{
{datetimeValue}        {printFieldString("date", yytext, 0);}
\"                              {yy_pop_state();}
}
```

No que toca ao preenchimento do campo *nome* no ficheiro de *output* este é feito no contexto referente ao *name*. Uma vez neste contexto, é feito o *match* de qualquer caractere excepto o fecho da *tag* referente ao nome do utilizador, sendo depois feito o *print* para o ficheiro o resultado do *match* do **yytext*. Após ser feito o *match* do fecho da *tag* relativo ao nome do utilizador é feito o *pop* deste estado.

```
<NAME>{
.+/\<\/a\>           {printFieldString("user", yytext, 0);}
{endAnchor}         {yy_pop_state();}
}
```

Relativamente ao texto do comentário, existem três filtros relativamente a possíveis ocorrências de caracteres especiais, como por exemplo a mudança de linha onde é feito o *print* para o ficheiro *JSON* de um espaço ao invés da mudança de linha, até porque o formato *JSON* não aceita *strings* multi-linha. Caso seja feito *match* de uma aspa, então para o ficheiro *JSON* é impresso o contéudo dentro do *fprintf*. Qualquer outro caracter é impresso para o ficheiro. No final quando é efetuado o filtro do fecho da *tag* o consequente *match* com esta expressão leva a um *pop* deste contexto.

```
<COMMENTEXT>{
{endParagraph} {printf(fptr, "\",\n"); yy_pop_state();}
\"             {fprintf(fptr, "\\\"");}
\n             {fprintf(fptr, " ");}
.              {fprintf(fptr, "%s", yytext);}
}
```

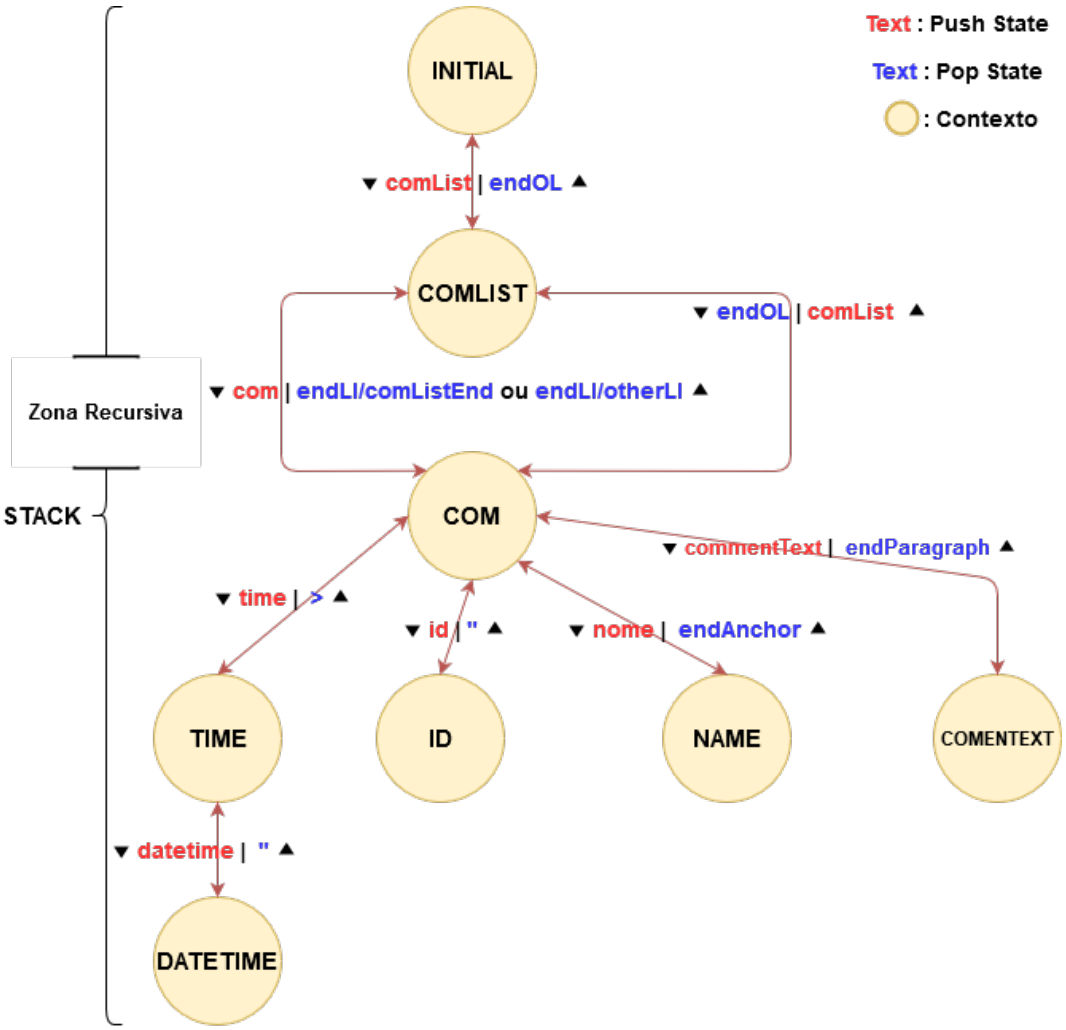

Figura 2.1: Representação da transição de contextos.

De forma a aumentar a robustez do programa foram acrescentadas algumas verificações como averiguar se um ficheiro existe e se possuímos permissão de leitura sobre o mesmo. No fim do ficheiro também é feita a conversão do ficheiro de *WINDOWS-1252* ou *CP-1252* para *UTF-8*.

Para facilitar a utilização e explicar algumas coisas a quem utiliza também foi elaborado um pequeno menu *help*.

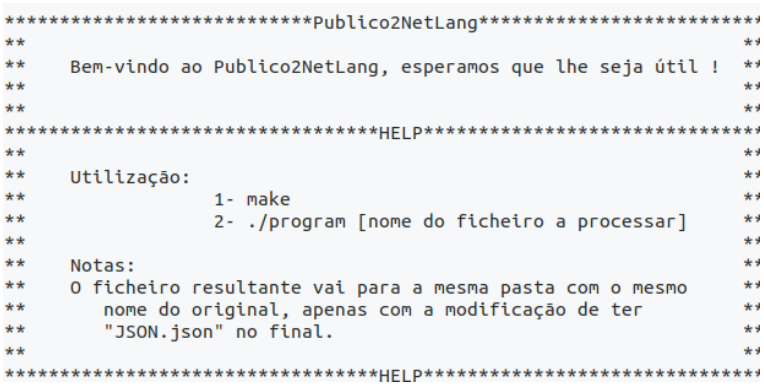

Figura 2.2: Menu de ajuda.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de dados

Considerando apenas os exemplos dados, poder-se-ia concluir que apenas existiriam comentários e respostas a esses comentários. Contudo, são impostos requisitos sobre a solução, que passam por recorrer a recursividade. Quer isto dizer que não só comentários poderão ter respostas, mas também que respostas poderão ter respostas.

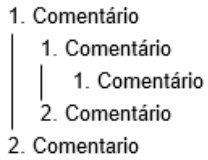

Figura 3.1: Comentários recursivos.

Com isto surge um problema na contagem do número de respostas a um comentário, que num caso simples poderia ter-se meramente recorrido a um ao tipo primitivo *int*. Isto impõe uma solução que passa pelo recurso a recursividade visto que a estrutura HTML dada possui propriedades recursivas na forma em como representa respostas a um comentário, que podem ser muito sucintamente representadas da seguinte forma:

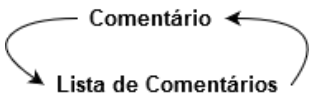

Figura 3.2: Representação da recursividade da estrutura.

Para o efeito, era necessário criar uma estrutura que respondesse às exigências impostas pelo problema. Começou-se por criar uma estrutura que representasse uma lista de comentários. A essa estrutura chamou-se *nivel*, pois as listas de comentários estão dispostas por níveis ou profundidades em relação ao primeiro comentário. Um esquema da estrutura pode ser visto de seguida.

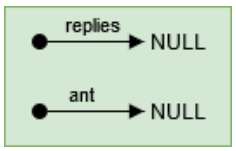

Figura 3.3: Representação da estrutura nivel.

A estrutura possui um campo chamado *ant* que aponta para o *nivel* anterior de forma podermos retroceder de *nivel* rapidamente. Possui ainda outro campo chamado *replies*, que se resume a uma estrutura próxima da definição de lista ligada que representa os comentários presentes numa lista de comentários (*nivel*). Esta estrutura foi denominada de *comentario* e foi esquematizada na seguinte figura.

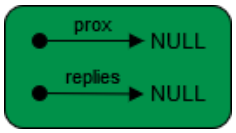

Figura 3.4: Representação estrutura comentario.

Conjugando as duas estruturas obteve-se a estrutura abaixo. Esta apenas contém apontadores e resolve o problema da contagem de comentários para o caso em que vários níveis (listas de comentários) aninhados.

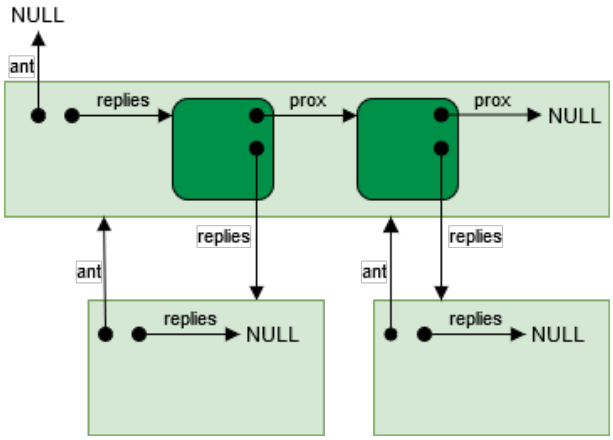

Figura 3.5: Representação da estrutura criada para o trabalho.

O exemplo acima representa um comentários deste tipo:

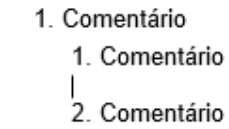

Figura 3.6: Esquema do comentário representado na figura 3.5.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Como não poderia deixar de ser: inerente à produção de um trabalho é a tomada de decisões e, infelizmente, o aparecimento de problemas. Aqui falar-se-ão das decisões mais importantes tomadas ao longo do desenvolvimento do projecto.

A primeira decisão tomada foi a escolha do enunciado. O grupo de trabalho escolheu o enunciado "*Publico2NetLang*", pois não só parecia o projecto mais próximo da realidade, mas também devido ao trabalho que os elementos têm feito na área das tecnologias WEB, o que permite um conforto e familiaridade com o formato JSON.

Numa fase inicial surgiram várias dúvidas no que toca à possibilidade *input*, ou seja, quão aninhados poderiam ser os comentários. Foi-nos esclarecido que os comentários poderiam estar aninhados indefinidamente. Com isto e como foi explicado na secção "*Estruturas de dados*", surgiu um problema na contagem do número de replies (campo "*numberOfReplies*") o que fez com que fosse necessária a criação de uma estrutura auxiliar.

Tendo em conta que no formato JSON a ordem dos campos não é relevante para o seu acesso, o grupo de trabalho enveredou por utilizar o mínimo de memória. Isto em conjunto com o *stumbling block* descrito no parágrafo anterior deu origem a uma estrutura composta apenas por apontadores que manteria registo das respostas a um comentários. Sendo que os restantes dados são imediatamente escritos logo que encontrados.

A partir daí o trabalho fluiu com naturalidade apenas com algumas dúvidas no que toca a REGEX o que é normal pois o grupo de trabalho não têm experiência na matéria.

No que toca à estrutura de dados houveram alternativas. Por exemplo, uma estrutura singular que tivesse um apontador para uma estrutura igual e para a anterior, mas optou-se por duas structs também por ser mais compreensível.

Também existiria a opção de guardar os dados em memória e só depois escrevê-los, mas mais uma vez, o formato JSON não necessita que campos estejam ordenados e que isso implicava guardar os dados todos de um comentários e de todas as suas respostas, o que, se um comentário tivesse muitas repostas, levaria um excesso no uso de memória.

4.2 Testes realizados e resultados

No que concerne ao *teste* da solução concebida pelo grupo, após contactar um dos docentes da cadeira foi possível obter extratos *html* do Público para testar o filtro de texto desenvolvido.

De forma a avaliar o desempenho do filtro de texto concebido, foi medido o tempo desde o início da filtragem,

ou seja, o *yylex* até ao fim da escrita para o ficheiro *JSON*.
O primeiro ficheiro possui cerca de 85 comentários no total, e segue a estrutura *html* apresentada anteriormente. Como tal, para este ficheiro foi possível obter o seguinte tempo de execução do programa.

```
joao@joao-VirtualBox:~/Uni/PL/TP/PL/TP1/src$ ./program ../dados/Publico_extracti
on_portuguese_comments_4.html
Programa demorou: 0.005542s
```

Figura 4.1: Primeiro teste efetuado.

Como é possível observar, o tempo de resposta por parte do programa é de cerca de 0.005 segundos para fazer a filtragem de 85 comentários. Foi também possível gerar o seguinte *output* para esse ficheiro de *input*.

```
"commentThread": [
        {
        "timestamp": 1586083396,
        "likes": 0,
        "id": "06de7129-6167-49cd-d330-08d743683e5c",
        "user": "PdellaF ",
        "date": "2019-10-03T21:11:55.99",
        "commentText": "Do assunto e de Justiça, Abrunhosa nada percebe.
        Não sabe que o MP pronunciou nesta altura porque era esse o prazo?
        Não sabe. tenho para mim que quando alguém, dando a sua opinião, inventa cabalas e teoria
        da conspiração é porque é cognitivamente débil ou está a soldo de uma facção.
        Rui Rio pegou numa câmara municipal corrupta, ineficaz e gerida para pagar a clientelas
        partidárias e outros parasitas e transformou-a
        numa organização equilibrada.
        naturalmente que isso não agradou a si nem aos seus amigos que viviam parasitariamente da
        "hasReplies": false,
        "numberOfReplies": 0,
        "replies": []
        },
        (...)
```

Foi também possível obter um segundo extrato, desta vez um ficheiro um pouco maior que o primeiro, com cerca de 88 comentários, de onde se obteve o seguinte tempo de execução do programa.

```
joao@joao-VirtualBox:~/Uni/PL/TP/PL/TP1/src$ ./program ../dados/Publico_extracti
on_portuguese_comments_9.html
Programa demorou: 0.007182s
```

Figura 4.2: Segundo teste efetuado.

Como é possível observar o tempo de execução apenas subiu ligeiramente para um ficheiro também ligeiramente maior. No que concerne ao *output* resultante, foi gerado o seguinte ficheiro *JSON*.

```
{"commentThread": [
        {
        "timestamp": 1586084082,
        "likes": 0,
```

```
"id": "073bdffa-d851-426c-9685-08d730042b61",
"user": "João ",
"date": "2019-09-04T02:10:52.88",
"commentText": " Como já se disse em baixo, e só é visível para assinantes e os que têm
a versão em papel, o artigo acaba como um \"as despesas
da viagem foram pagas pela ILGA-Europa\".
Depois não se admirem de os jornais serem acusados de terem uma agenda própria
e serem parciais. É que em casos como este, são mesmo. ",
"hasReplies": false,
"numberOfReplies": 0,
"replies": []
},
(...)
```


De modo a testar com uma carga de trabalho maior, o grupo decidiu criar um ficheiro *HTML* com cerca de 200 *megabytes* de tamanho, com cerca de três milhões de linhas. Após exectuar o programa com este ficheiro de *input* foi possível obter o seguinte tempo de execução.

```
joao@joao-VirtualBox:~/Uni/PL/TP/PL/TP1/src$ ./program ../dados/Publico_200MB.ht
ml
Programa demorou: 10.672668s
```

Figura 4.3: Terceiro teste efetuado.

Portanto para um ficheiro com um tamanho grande o programa demorou cerca de 10.7 segundos para filtrar o texto e construir o ficheiro de *output JSON*. Sendo, portanto, evidente a robustez da solução desenvolvida pelo grupo.

O ficheiro de input é uma repetição sucessiva do ficheiro utilizado para o primeiro teste, sendo que o *output* é portanto semelhante ao *output* obtido no primeiro teste.

Capítulo 5

Conclusão

Ao longo da concepção deste primeiro trabalho prático o grupo deparou-se com alguns *stumbling blocks*, como por exemplo a ocorrência de aspas no *commentText*. Contudo, após alguma reflexão e pesquisa, foi sempre possível ultrapassar os entraves que surgiram.

O trabalho revelou-se como sendo crucial para consolidar os conhecimentos no que toca à utilização e criação das expressões regulares bem como um melhor entendimento da linguagem de filtragem de texto *FLex*.

Como tal o grupo de trabalho faz uma avaliação positiva da solução concebida para esta primeira fase, sendo que o grupo considera que esta solução atende e responde a todos os tópicos necessários e efectua uma filtragem correta e clara.

Apêndice A

Código do filtro de texto (tp1.fl)

```
%{
/* Declaracoes C diversas */
#include <comentarios.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
int fileno(FILE *stream);

long profundidade = 0;
Nivel com = NULL;

void printTimestamp();
void endComment();
void printNTimes(int times, char print);
void printFieldStringPreDef(char* string);
void printFieldString(char* field, char* yytext, int ultimo);
void printFieldNum(char* field, long num, int ultimo);
%}
%option stack noinput nounput noyy_top_state
/* Abreviaturas de ER */

/* Gerais */
space [\ \t \n]
decimal  [0-9]
alpha  [a-zA-Z]

comList \<ol\ +class\ *=\ *\"comments__list\"
endOL \<\/ol\>

com \<li\ +class\ *=\ *\"comment\"
endLI \<\/li\>
comListEnd  {space}*{endOL}
otherLI {space}*\<li
```

```
id data\-comment\-id=\"
idValue ({alpha}|{decimal}|\-)+

time \<time
datetime  datetime\ *=\ *\"
datetimeValue ({decimal}|[\-\:\.T])+

nome \<h5\ *class=\"comment__author\"\ *\>{space}*\<a[^\>]+\>
endAnchor \<\/a\>

commentText  \<div\ *class=\"comment__content\"\>{space}*\<p\>
endParagraph \<\/p\>

%x COMLIST COM ID TIME DATETIME NAME COMMENTEXT

%%
{comList} {yy_push_state(COMLIST);}

<COMLIST>{
{com} {
if(profundidade > 0){ addComment(com); com = addNivelToComment(com);}
printFieldStringPreDef("{\n");
printTimestamp();
printFieldStringPreDef("\"likes\": 0,\n");
yy_push_state(COM);
}
{endOL} {
if(profundidade > 0) printFieldStringPreDef("],\n");
--profundidade;
yy_pop_state();
}
}

<COM>{
{comList} {
printFieldStringPreDef("\"hasReplies\": true,\n");
printFieldStringPreDef("\"replies\": [\n");
++profundidade;
if(profundidade == 1) com = createNivel(com);
yy_push_state(COMLIST);
}
{id} {yy_push_state(ID);}
{time} {yy_push_state(TIME);}
{endLI}/{comListEnd} {
endComment();
printFieldStringPreDef("}\n");
}
```

```
{endLI}/{otherLI} {
endComment();
printFieldStringPreDef("},\n");
}
{nome} {yy_push_state(NAME);}
{commentText} {
printFieldStringPreDef("\"commentText\": \"");
yy_push_state(COMMENTEXT);
}
}

<ID>{
{idValue} {printFieldString("id", yytext, 0);}
\" {yy_pop_state();}
}

<TIME>{
{datetime} {yy_push_state(DATETIME);}
\>{yy_pop_state();}
}

<DATETIME>{
{datetimeValue} {printFieldString("date", yytext, 0);}
\" {yy_pop_state();}
}

<NAME>{
.+/\<\/a\>{printFieldString("user", yytext, 0);}
{endAnchor} {yy_pop_state();}
}

<COMMENTEXT>{
{endParagraph} {fprintf(yyout, "\",\n"); yy_pop_state();}
\" {fprintf(yyout, "\\\"");}
\n   {fprintf(yyout, " ");}
. {fprintf(yyout, "%s", yytext);}
}

<*>.|\n {}
%%

int yywrap()
{ return(1); }

void help(){

printf("\n");
printf("**************************Publico2NetLang************************\n");
```

```
printf("**                                                                  **\n");
printf("**    Bem-vindo ao Publico2NetLang, esperamos que lhe seja útil !  **\n");
printf("**                                                                  **\n");
printf("**                                                                  **\n");
printf("**********************************HELP**********************************\n");
printf("**                                                                  **\n");
printf("**    Utilização:                                                   **\n");
printf("**                1- make                                           **\n");
printf("**                2- ./program [nome do ficheiro a processar]       **\n");
printf("**                                                                  **\n");
printf("**    Notas:                                                        **\n");
printf("**    O ficheiro resultante vai para a mesma pasta com o mesmo      **\n");
printf("**       nome do original, apenas com a modificação de ter          **\n");
printf("**       \"JSON.json\" no final.                                      **\n");
printf("**                                                                  **\n");
printf("**********************************HELP**********************************\n");

}
int contador = 0;
void endComment(){
long nReplies = numberOfReplies(com);

if(nReplies > 0) {
printFieldNum("numberOfReplies", nReplies, 1);
}
else{
// para não imprimir como string
printFieldStringPreDef("\"hasReplies\": false,\n");
printFieldNum("numberOfReplies", numberOfReplies(com), 0);
printFieldStringPreDef("\"replies\": []\n");
}

if(profundidade > 0) com = getAnt(com);
else {freeAll(com); com = NULL;}

yy_pop_state();
}

void printNTimes(int times, char print){
for(int i = -1; i < times; i++)
fprintf(yyout, "%c", print);
}

void printFieldStringPreDef(char* string){
printNTimes(profundidade, '\t');
fprintf(yyout, "%s", string);
}
```

```
void printFieldString(char* field, char* yytext, int ultimo){
printNTimes(profundidade, '\t');
if(ultimo == 1) fprintf(yyout, "\"%s\": \"%s\"\n", field, yytext);
else fprintf(yyout, "\"%s\": \"%s\",\n", field, yytext);
}

void printFieldNum(char* field, long num, int ultimo){
printNTimes(profundidade, '\t');
if (ultimo == 1) fprintf(yyout, "\"%s\": %ld\n", field, num);
else fprintf(yyout, "\"%s\": %ld,\n", field, num);
}

void printTimestamp(){
time_t t = time(NULL);
printFieldNum("timestamp", t, 0);
}

int main(int argc, char* argv[]){

if(argc < 2){
help();
return 0;
}

if(access(argv[1], F_OK) != -1){

if(access(argv[1], R_OK) != -1){

// abrir o ficheiro temporário para escrever
yyout = fopen("TEMP.json", "w");

// escrever nome da collection
fprintf(yyout, "{\"commentThread\": [\n");

// abrir o ficheiro a ler
yyin = fopen(argv[1], "r");

clock_t start = clock();
// inicializar a leitura
yylex();

// fechar a collection
fprintf(yyout, "]}\n");

fclose(yyin);
fclose(yyout);

int len = strlen(argv[1]);
```

```
// definir nome do ficheiro final
char CP1252toUTF8[45+len];
char nome[len+10];
strcpy(nome, argv[1]);
nome[len-5] = '\0';
strcat(nome, "JSON.json");

// Como o ficheiro de input (yyin) tem o encoding CP1252 tem se passar para UTF8
sprintf(CP1252toUTF8, "iconv -f cp1252 -t utf8 TEMP.json > \"%s\"", nome);
system(CP1252toUTF8);
system("rm TEMP.json");
clock_t end = clock();
float seconds = (float)(end - start) / CLOCKS_PER_SEC;
printf("Programa demorou: %fs\n", seconds);
}
else{
printf("Não possui permissão de leitura sobre o ficheiro fornecido!\n");
}
}
else{
printf("O ficheiro dado como argumento não existe !\n");
}

return 0;
}
```

Bibliografia

[1] Enunciado do trabalho prático,
https://natura.di.uminho.pt/$\sim jj/pl-20/TP1/pl19tp1.pdf$

[2] C File Handling,
https://www.programiz.com/c-programming/c-file-input-output

[3] Windows-1252,
https://en.wikipedia.org/wiki/Windows-1252

[4] Quora - Lex functions and variables,
https://www.quora.com/What-is-the-function-of-yylex-yyin-yyout-and-fclose-yyout-in-LEX

[5] Quora - Lex functions and variables,
https://www.quora.com/What-is-the-function-of-yylex-yyin-yyout-and-fclose-yyout-in-LEX

[6] Flex Options,
http://dinosaur.compilertools.net/flex/flex_17.html

[7] Flex Pratice,
https://www.epaperpress.com/lexandyacc/prl.html

[8] UTF-8 encoding table and Unicode characters,
https://www.utf8-chartable.de/unicode-utf8-table.pl