Universidade do Minho

Processamento de Linguagens e Compiladores

Trabalho Prático nº1:

Transformador Sol2NetLang

Grupo nº8

Carlos Ferreira
a87953

Daniel Ribeiro
a87994

Braga, Portugal
19 de novembro de 2020

Resumo

O seguinte relatório documenta, justifica, analisa e expõe todas as decisões tomadas ao longo do projeto "Transformador Sol2NetLang"realizado no âmbito da Unidade Curricular denominada Processamento de Linguagens e Compiladores no contexto do 3º ano do curso universitário Licenciatura em Ciências da Computação.

O seguinte trabalho era um dos 7 que nos foi apresentados na Unidade Curricular, visto que somos o grupo 8, usando a formula (NGr % 7) + 1 verificamos que nos calhou o trabalho nº2, este consistia em a partir de um ficheiro HTML contendo uma série de comentários retirados do jornal "Sol", utilizando o *Flex* (*Fast Lexical Analyzer Generator*) para filtrar o ficheiro através de expressões regulares e retirando assim a informação relevante e produzindo com ela um ficheiro no formato *JSON*.

Ao longo deste relatório vamos apresentar as nossas ideias de forma clara e objectiva, apoiando a sua explicação através de esquemas para uma melhor compreensão por parte do leitor.

Apesar de algumas dificuldades iniciais, conseguimos resolver o problema com elevado grau de sucesso e estamos bastante satisfeitos com o resultado.

Conteúdo

Capítulo 1

Introdução

1.1 Transformador Sol2NetLang

Vamos agora enquadrar e contextualizar o trabalho.

No capítulo 2 fazemos uma breve introdução ao enunciado, assim como os pontos importantes a retirar desse mesmo, os nossos pensamentos iniciais assim como estratégias deliberadas e adoptadas.

No capítulo 3 são explicados os métodos usados para filtrar assim como as estruturas de dados onde foram armazenados os comentários e ainda a forma como imprimimos para ficheiro.

No capítulo 4 concluímos o trabalho fazendo umas considerações finais á cerca do mesmo.

Capítulo 2

Análise do problema

2.1 Requisitos do Enunciado

No enunciado era nos informado que o ficheiro de output no formato *JSON* devia ter a seguinte estrutura:

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

      "replies": [ ]
     }, ......
]
```

Onde a informação a ser colocada nos campos seria:

id - O id único de cada comentário.

user - O utilizador que fez o comentário.

date -A data em que o user fez o comentário.

timestamp - A timestamp de cada comentário.

commentText - O texto do comentário.

likes - O número de likes do comentário.

hasReplies - Se o comentário possui respostas.

numberOfReplies - O número de respostas ao comentário.

replies - As respostas ao comentário, que também são comentários.

Percebemos logo que para cada linha de informação iríamos ter de construir uma expressão regular para a retirar do input no formato *HTML*.
Seguidamente fomos analisar os ficheiros de *HTML* fornecidos, reparamos que todos os comentários seguiam os mesmos padrões e possuíam "palavras-chave"onde a partir delas iríamos conseguir ir buscar a informação desejada.
Segue um excerto de um dos comentários:

```
<li class="post" id="post-4786087507"><div role="alert"></div>
<div class="post-body"><header class="comment__header"><span class="post-byline">
<span> <span class="author publisher-anchor-color">
<a data-action="profile" data-username="disqus_PRODHTFTl7"
href="https://disqus.com/by/disqus_PRODHTFTl7/"
rel="noopener noreferrer" target="_blank">Luiz Felipe</a>
 <a class="time-ago" data-role="relative-time"
 href="https://sol.sapo.pt/artigo/685414/
 marido-de-goucha-deixa-provocacao-a-joacine-katar-moreira-sera-que-e-xenofobia
 -#comment-4786087507" title="Thursday, February 6, 2020 5:02 PM">12 days ago</a>

<div class="post-body-inner"><div class="post-message-container"
data-role="message-container"><div class="publisher-anchor-color"
data-role="message-content"><div class="post-message" data-role="message"
dir="auto"><div><p>Qual dos dois é a esposa? será que não podiam passar
despercebidas/os??</p>
</div>
<span class="post-media"><ul data-role="post-media-list"></ul>
</span>
</div><span class="updatable count" data-role="likes">0</span>
</span>
</li>
</li>
<div class="children">
<ul data-role="children"></ul>
<div class="show-children-wrapper hidden"><a class="show-children"
data-action="show-children"
href="http://www4.di.uminho.pt/~prh/Sol/Sol_extraction_portuguese_comments_4.html#"
id="post-4786087507-show-children">Show more replies</a>
</div>
</li>
```

(Algumas partes menos essenciais do código foram cortadas para não ficar tão extenso, é possível ver o código completo nos apêndices em baixo)

A partir de uma análise cuidadosa deste excerto podemos identificar as tags e classes *HTML* importantes, sendo elas:

- < li class="post"id="post-4786087507"> onde "post-4786087507"representa o id do comentário.
- data-username= (...) >Luiz Felipe onde "Luiz Felipe"representa o user
- < a class="time-ago"(...) title="Thursday, February 6, 2020 5:02 PM"onde "Thursday, February 6, 2020 5:02 PM"> representa a data de publicação do comentário.
- < a class="time-ago"(...) > 12 days ago onde "12 days ago"representa a timestamp do comentário.
- < div class="post-message"(...) Qual dos dois é a esposa? será que não podiam passar despercebidas/os?? onde " Qual dos dois é a esposa? será que não podiam passar despercebidas/os?? "representa o texto do comentário
- data-role="likes"> 0 onde 0 representa o número de likes do comentário

Assim já sabemos onde ir retirar a informação, agora temos de criar uma estratégia para o mesmo, isso será abordado na próxima secção

2.2 Estratégia inicial adoptada

Depois de analisarmos os requisitos, deliberamos sobre a estratégia a adotar para o programa.

Nesta fase pensamos em diversas soluções, apontamos problemas a cada uma das mesmas, até que chegamos á solução que achamos mais adequada para a resolução deste problema, onde prosseguimos á sua implementação no capítulo seguinte.

Fizemos bastante pesquisas nesta etapa, para entender quais as melhores formas de ler de um ficheiro para posteriormente filtrar e guardar.

Assim a solução final chegou, utilizando uma struct do C para armazenar os dados, íamos lendo continuamente o ficheiro html onde filtrávamos com o Flex informação relevante, guardando então na struct, usando push e pop de start conditions á medida que precisávamos para filtrar, e com uma stack de apontadores para conseguirmos manipular as respostas aos comentários.

No final criávamos uma função que fosse capaz de pegar na nossa struct e imprimir para o formato desejado.

Capítulo 3

Soluções criadas

3.1 Estruturas de dados

Existe três coisas principais que temos de pensar neste ponto, como retirar a informação do html, como guardar essa mesma informação e no final como imprimir-la para ficheiro.
Comecemos pela forma de armazenar a informação, para isso criamos uma struct comentario em C:

```
typedef struct comentario {

    char* id;
    char* date;
    char* user;
    char* commentText;
    char* timestamp;
    int likes;
    int numberOfReplies;
    struct comentario *replies;
    struct comentario *next;

}*Comentario;
```

Podemos reparar que esta struct possui 2 apontadores chamados 'replies' e 'next', pois esta struct funciona como uma lista ligada, onde temos a referência da primeira resposta assim como a referência do próximo comentário na mesma profundidade. Não utilizamos uma variável 'hasReplies', pois basta verificar o valor da variável 'numberOfReplies'. Assim criamos uma função inicial criaComentário() para nos facilitar a criação de um objeto desta estrutura:

```
Comentario criaComentario (){

      Comentario h = NULL;
      h = malloc(sizeof(struct comentario));
      h->likes = 0;
      h->numberOfReplies = 0;
```

```
        h->replies = NULL;
        h->next = NULL;
        h->timestamp = "NA";
        h->date = "";
        h->user = "";
        h->id = "";
        h->commentText = "";

        return h;
}
```

3.2 Filtros de texto

Com as estruturas de dados criadas, passamos á filtragem do *HTML* para as preencher, começamos então por definir as várias tags que nos poderiam aparecer.
Para ficar um código mais limpo e organizado decidimos definir por keywords os textos para filtrar usando o flex, sendo eles:

```
    digito              [0-9]
    espaco              [\ \t\r\n]
    alpha               [a-zA-Z]
    acentos             \xc3[\x80-\xbf]
    letra               {alpha}|{acentos}

    username            ({letra}|{digito}|{espaco})+
    timestamp           ({alpha}|{digito}|{espaco})+
    data                title\=\".*(AM|PM)\"
    user                class\=\"post-body\".*data-username.*\"\>
    inicioComentario    class\=\"post\"{espaco}id\=\"post\-{digito}+\"
    fimComentario       class\=\"show-children\-wrapper[' ']hidden\"
    likes               data\-role\=\"likes\"\>{digito}+
    inicioTexto         class\=\"post-message\".*\<div\>
    inicioParagrafo     \<p\>
    fimParagrafo        \<\/p\>
```

Assim como as várias Start Conditions:

```
%x comentario user_SC texto tempo
```

Sem esquecer de ativar a opção da stack pois é essencial ao trabalho:

```
%option stack
```

A primeira expressão regular a ser encontrada é a que corresponde ao inicioComentário:

```
{inicioComentario} {
                        yy_push_state(comentario);
                        atual = criaComentario ();
                        if (ultimo != NULL){
                            ultimo -> next = atual;
                        }

                        push(atual,&posCabecaStack , MAXSIZE , stack);
                        if(iniciado == 0){
                            iniciado = 1;
                            lista = atual;
                        }
                        atual->id =  strdup(yytext+16);
}
```

Quando esta expressão é encontrada sem estarmos em nenhum estado (INITIAL STATE),indica-nos que encontrámos um comentário "principal"(profundidade 0).

Adicionamos o estado 'comentario' à stack, estado este que nos permite encontrar as informações relacionadas a este comentário.

Seguidamente criamos o comentário e guardamos no apontador 'atual' , que significa o comentario a ser atualmente analisado. O apontador ultimo representa o ultimo comentario que encontramos no INITIAL STATE, caso ultimo seja nulo quer dizer que o 'atual' é o primeiro comentario (de profundidade 0 , ou seja não é uma reply) a ser encontrado, caso o apontador 'último' for diferente de NULL, vamos colocar o apontador next dele a ser o 'atual', construindo assim a lista ligada.

Depois fazemos push do 'atual' para uma stack de Comentários que possui as seguintes funções básicas de manipulação:

```
int isempty(int  * posCabecaStack);
int isfull(int * posCabecaStack, int MAXSIZE);
Comentario peek(int * posCabecaStack , Comentario * stack);
Comentario pop(int * posCabecaStack , Comentario * stack);
void push(Comentario data , int * posCabecaStack , int  MAXSIZE , Comentario * stack);
Comentario paraFim (Comentario c);
```

(Para ver como foram definidas, pode dirigir-se ao apêndice 2)

Esta stack vai guardar os comentarios hierarquicamente, ou seja , o elemento no topo da stack será reply do elemento na posição imediatamente anterior na stack , e assim em diante.

O iniciado indica se a lista já foi construída, e usamos o apontador apenas para guardar a posição da cabeça da lista ligada;

No final preenchemos o id do comentário, usando a função strdup com o argumento yytext+16 pois queremos começar 16 caracteres á frente do que foi lido.

```
<comentario>{user} { yy_push_state(user_SC); }

<user_SC>{username}  {
                    atual -> user = strdup(yytext);
```

```
                    yy_pop_state();
}
```

Quando encontramos o user damos push do estado user_SC, nesse estado quando encontrarmos um username((letra—digito—espaco)+), guardamos-lo e damos pop deste estado.

```
<comentario>{data} {
                    atual->date =  strdup(yytext+6);
                    yy_push_state(tempo);
}
```

Após o user vai-nos aparecer a data estando esta nas classes "time-ago", dentro do atributo title acabando sempre em AM ou PM, vamos ler-la e guarda-la (le-mos no yytext+6 para evitar o "title="), no final transitamos para o estado tempo:

```
    <tempo>{timestamp} {
                    atual -> timestamp = strdup(yytext);
                    yy_pop_state();
}
```

Este estado é logo seguido da data, apenas separado por '>' por isso apenas vamos ter de ler uma sequência composta por dígitos pois a timestamp acaba num símbolo também '<', letras e espaços, como assim foi definida a timestamp, guardamos-la e fazemos pop para voltar ao estado comentário.

Após a data, aparece-nos o texto do comentário, este causou-nos um pouco de confusão e complicação, pois estes comentários podem ser multi-linhas (possuir \n) assim como em alguns comentários ter certas tags html inseridas pelo autor para enfatizar o texto (por exemplo paragrafo, sublinhar, negrito, italiano..)

Começamos primeiro por encontrar a expressão regular que indica o início de um texto:

```
<comentario>{inicioTexto} {
                    yy_push_state(texto);
                    len = 0;
                    atual->commentText = malloc(1);
}
```

Como é possível observar, nós vamos para o estado "texto", colocamos a len = 0, sendo esta uma variável global que nos será útil para saber quantos caracteres tem o texto e assim alocar a quantidade de memória suficiente, e também fazemos uma alocação inicial de 1 byte de memória, apenas para inicializar e ser possível usar o realloc mais á frente:

```
<texto>{inicioParagrafo}.*|\n*{fimParagrafo} {
         len+= yyleng;
         if (yytext[0] == '\n')
                 {
         len-=1;
         atual->commentText= realloc(atual-> commentText, len * sizeof(char));
         strcat(atual->commentText,yytext+1);
           }
         else {
```

```
        atual->commentText= realloc(atual-> commentText, len * sizeof(char));
        strcat(atual->commentText,yytext);
        }
           }

<texto>\<\/div\> { yy_pop_state(); }
```

Para extrair o comentário, a nossa melhor opção foi ir lendo de parágrafo a parágrafo com o inicio e fim de paragrafo definido, dentro dessas tags podemos ter .* ou \n pois ou temos qualquer texto ou temos \n, não podendo estes ser juntos pois, como o algoritmo de flex é greedy ele iria ler o html desde o primeiro "<p>"até ao último "< /p>"depois de ler isto, caso a primeira coisa que lemos foi \n, deitamos fora pois caso imprimisse-mos este \n no ficheiro *json* iríamos ter um erro e este ficar desformatado.

Após isso damos realloc da memória necessária para guardar o parágrafo atual mais os anteriores e concatenamos o atual aos anteriores.

Finalmente quando acabarem os parágrafos, vamos encontrar uma tag que fecha uma divisão no ficheiro html, é aqui mesmo que acaba o texto do comentário.

```
    <comentario>{likes} {   atual -> likes =  atoi(yytext+18);}
```

O número de likes aparece logo de seguida e é bastante direto de o ler, fazemos atoi, que é para passar de string para int, e o +18 são o número de carateres que não nos interessam daquilo que filtramos.

```
    <comentario>{fimComentario}   {
            ultimo = pop(&posCabecaStack , stack);
            if (isempty(&posCabecaStack)){
         yy_pop_state();
    }
}
```

No estado comentário se encontrarmos um fimdeComentario, este pode ser o fim de uma reply ou o fim do comentário "principal", caso seja o fim de uma reply a stack não vai ficar vazia depois do pop pois ainda vão estar lá guardados os comentarios de profundidade menor. Cado a stack esteja vazia quer dizer que este fim é do comentário "principal", logo damos pop do estado 'comentario' , e voltamos para o INITIAL STATE onde vamos procurar os restantes comentários principais do html.

```
            <*>.|\n
```

Qualquer outra coisa em qualquer estado é ignorada.

```
<comentario>{inicioComentario} {
atual = criaComentario ();


ant = peek(&posCabecaStack , stack);
if (ant->replies == NULL){
ant-> replies = atual;
ant->numberOfReplies++;
```

```
}
else {
ant->numberOfReplies++;
ant = paraFim(ant->replies);
ant -> next = atual;

}


atual->id =  strdup(yytext+16);
push(atual,&posCabecaStack , MAXSIZE , stack);
}
```

A única condição que não vimos é quando se inicia um comentário, já no estado comentário, isto quer dizer que estamos perante uma resposta a um comentário e esta tem de ser tratada de forma diferente.
Criamos a estrutura e guardamos o apontador em 'atual', de seguida fazemos um peek(ver o elemento no topo da stack), guardando o apontador do resultado do peek em 'ant'(ou seja, 'atual' é reply do 'ant', pois 'ant' é o que está no topo da stack). Caso o apontador de replies for NULL, ou seja esta é a primeira reply que encontramos, o apontador de replies do 'ant' vai passar a apontar para o 'atual', e depois aumentamos o número de replies. Caso contrário, já existe uma lista de replies, logo, vamos para o fim dessa lista de replies e adicionamos lá o 'atual', aumentando o número de replies também.
Sem esquecer de guardar o id no final, assim como fazer push para a stack.

3.3 Imprimir para Ficheiro

Para imprimir a struct para ficheiro, escrevemos a seguinte função:

```
void printaJson (Comentario l , FILE *yyout, int profundidade){

Comentario aux = l;
char arr[profundidade+1];
repeteN(arr,profundidade);
while(aux != NULL){

fprintf(yyout,"{\n\t\"id\": %s,\n",arr,arr,aux->id);
fprintf(yyout,"\t%s\"user\": \"%s\",\n",arr,aux->user);
fprintf(yyout,"\t%s\"date\": %s,\n",arr,aux->date);
fprintf(yyout,"\t%s\"timestamp\": \"%s\",\n",arr,aux->timestamp);
fprintf(yyout,"\t%s\"commentText\": \"%s\",\n",arr,aux->commentText);
fprintf(yyout,"\t%s\"likes\": %d,\n",arr,aux->likes);
fprintf(yyout,"\t%s\"hasReplies\": %s,\n",arr,(aux->numberOfReplies>0) ? "true" : "false");
fprintf(yyout,"\t%s\"numberOfReplies\": %d,\n",arr,aux->numberOfReplies);


Comentario repliesStruct = aux->replies;
fprintf(yyout,"%s\t\"replies\": [",arr);
```

```
if (repliesStruct == NULL){
fprintf(yyout,"]\n",arr);
} else {
fprintf(yyout,"\n");
printaJson(repliesStruct,yyout, profundidade +1);
fprintf(yyout,"\t%s]\n",arr);
}

if (aux -> next == NULL){
fprintf(yyout,"%s}\n",arr);
}
else {
fprintf(yyout,"%s},\n",arr);
}

aux = aux -> next;
}
}
```

E a sua auxiliar:

```
char* repeteN (char arr[], int profundidade){
for ( int i = 0; i < profundidade; i++){
arr[i] = '\t';
}
arr[profundidade] = '\0';
return arr;
}
```

Chamada da seguinte forma:

```
fprintf(yyout,"\"commentThread\": [\n");
printaJson(lista,yyout,1);
fprintf(yyout,"]");
```

A função printaJson pega no apontador l que é a cabeça da lista dos comentários, num apontador para ficheiro (yyout) que é o ficheiro de output e num terceiro argumento, a profundidade.
A função printaJson funciona como um ciclo que vai percorrendo a lista de apontadores enquanto o next for diferente de null, chamando sempre os replies recursivamente com profundidade incrementada, quando os possui.
A função auxiliar repeteN pega na profundidade e num array e vai preencher esse array com \t que vamos adicionar antes de cada print para ficar-mos com a identação correta.

3.4 Exemplo do funcionamento

O seguinte exemplo tem como objectivo clarificar o funcionamento principalmente da lógica dos estados e da stack de Comentários, devio a ser a parte mais complexa de entender.
Usando um ficheiro com a seguinte estrutura:

```
<li class="post" id="post-4786160376"><div role="alert"></div>
(...)<data-username="binopardal" (...)>Bino Pardal</a>
<a class="time-ago" (...) title="Thursday, February 6, 2020 5:54 PM">12 days ago</a>
(...)<div class="post-message" data-role="message"
dir="auto"><div><p>a miss piggy deve dormir entre os
dois...</p>
(...) data-role="likes">0</span>


<div class="children"><ul data-role="children"><li class="post"
id="post-4786946247"><div role="alert"></div>
(...)<data-username="disqus_derEqe29EO" (...)>Coniuratio Adversum Omnia</a>
<a class="time-ago" (...) title="Friday, February 7, 2020 7:36 AM">11 days ago</a>
(...)<div class="post-message" data-role="message"
dir="auto"><div><p>A tua mÃ£e faz de salchicha entre dois
rotos? Devias ter mais respeito por quem te mete o leitinho na mesa, imbecil!</p>
(...) data-role="likes">0</span>

<div class="children"><ul data-role="children"><li
class="post" id="post-4787032868"><div role="alert"></div>
(...) data-username="binopardal" (...)>Bino Pardal</a>
<a class="time-ago" (...) title="Friday, February 7, 2020 10:34 AM">11 days ago</a>
(...)<div class="post-message" data-role="message"
dir="auto"><div><p>a tua indignaÃ§Ã£o Ã© tanta que nem te percebo, fascista!!!</p>
<span class="post-media"><ul data-role="post-media-list"></ul>
(...)data-role="likes">0</span>


(...)<div class="show-children-wrapper hidden"><a
class="show-children" data-action="show-children"
href="#" id="post-4787032868-show-children">Show more replies</a>
(...)<div class="show-children-wrapper hidden"><a
class="show-children" data-action="show-children"
href="#" id="post-4786946247-show-children">Show more replies</a>
(...)<div class="show-children-wrapper hidden"><a
class="show-children" data-action="show-children"
href="#" id="post-4786160376-show-children">Show more replies</a>


<li class="post" id="post-4786949839"><div role="alert"></div>
(...) data-username="disqus_derEqe29EO" (...)>Coniuratio Adversum Omnia</a>
<a class="time-ago" (...) title="Friday, February 7, 2020 7:44 AM">11 days ago</a>
(...)<div class="post-message" data-role="message"
dir="auto"><div><p><b>Pensem no seguinte... <br>Ela
estÃ¡ a fazer campanha para um grupo de eleitorado muito especifico...
Tal como o Ventura o faz. Ela burra nÃ£o Ã© nada...</b>
</p>
<p><b>Quanto mais ela aparecer e quantos mais comentÃ¡rios a
```

```
mandarem-na daqui para fora mais pessoas desse seu eleitorado alvo vÃ£o pensar em
votar no partido que a apresentar a
eleÃ§Ãµes.</b>
</p>
<p><b>Quanto mais Ã³dio ou impropÃ©rios lanÃ§arem Ã 
Joacine mais estÃ£o a potenciar a sua eleiÃ§Ã£o. Pensem e sejamos todos um pouco mais
inteligentes...</b>
</p>
<p><b><u><i>Querem-se livrar dela? Deixem-na cair no esquecimento</i>
</u>
!!!!</b>
</p>
</div>
(...) data-role="likes">0</span>
(...)<div class="children"><ul data-role="children"></ul>
(...)<div class="show-children-wrapper hidden"><a class="show-children"
data-action="show-children" href="#" id="post-4786949839-show-children">
Show more replies</a>
(...)
```

neste exemplo temos 4 comentários, onde o 1 e 4 são "principais"(profundidade 0), onde o 1 tem como resposta o comentário 2, e o 2 tem como resposta o comentário 3, não havendo nenhuma resposta ao comentário 4.
Denotaremos por "Comentário x:"o início do comentário x, sendo este o caso chamado "inicioComentario"do flex:

```
inicioComentario    class\=\"post\"{espaco}id\=\"post\-{digito}+\"
```

E denotamos por "Fim x;"o fim do comentário x, sendo este o caso do flex chamado "fimComentario":

```
fimComentario       class\=\"show-children\-wrapper[' ']hidden\"
```

Assim criamos o seguinte esquema:

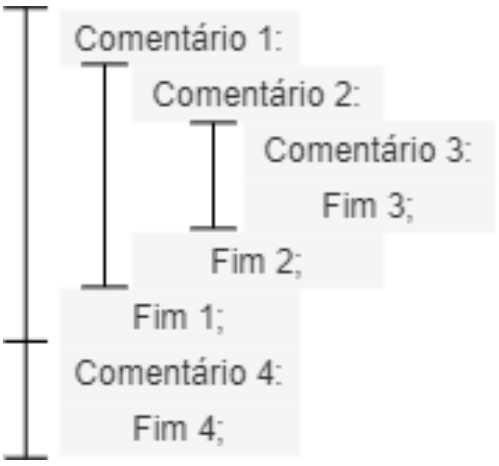
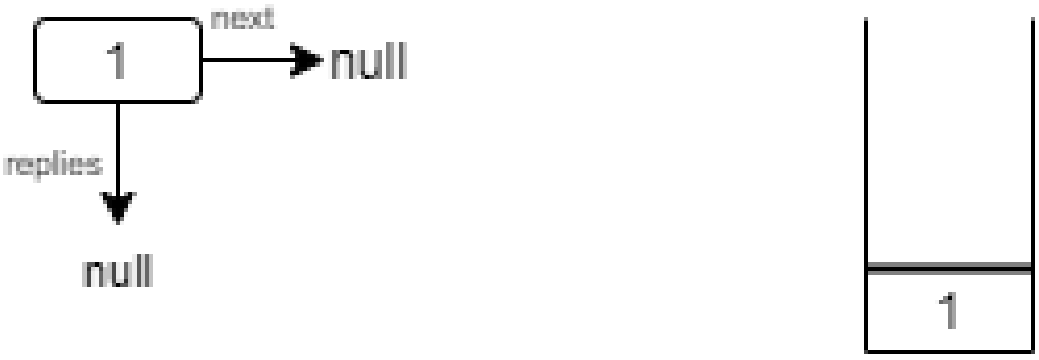


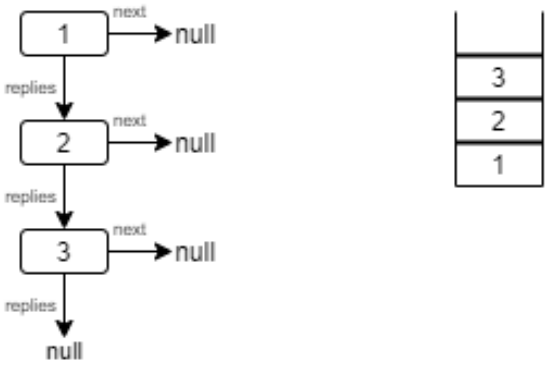
Figura 3.1: Comentários Encadilhados

A primeira coisa a ser filtrada vai ser o inicio do comentario 1, vamos assim para o caso do initialState inicioComentario, onde vamos criar uma nova struct comentario e colocar no apontador atual, dando push dela mesmo para a Stack de Comentários, assim como damos push da start Condition comentario, também colocamos o apontador lista a apontar para o atual, pois vai dar jeito para passar para a função de imprimir para ficheiro, após isso vamos ler e guardar o user, data, texto do comentário e assim, mas vamos ocultar essa parte neste exemplo.

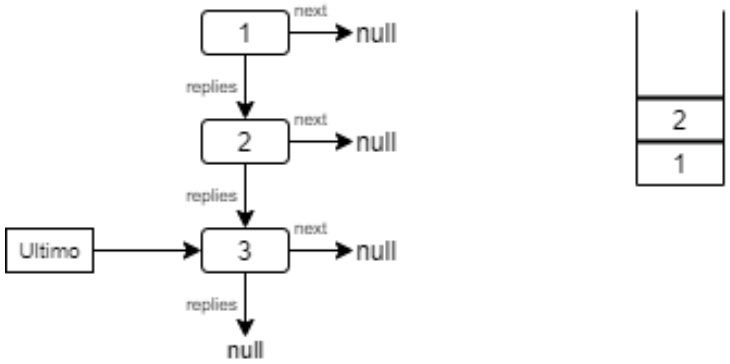


Após isso iremos encontrar o inicio do comentario 2, vamos assim para o caso <comentario>{inicioComentario} onde damos peek na Stack e vamos assim buscar o anterior ficando com o apontador do comentário 1, como o apontador replies do comentário 1 é null, vamos fazer esse apontador, apontar para o comentário 2. Após isso fazemos push do comentário 2 á Stack.

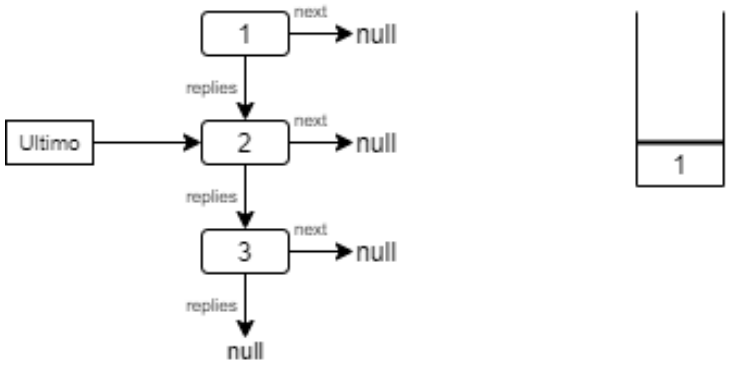
O mesmo irá acontecer ao encontrar-mos o inicio do comentário 3, vamos para o <comentario>{inicioComentario} damos peek e ficamos com o 2 , como os replies do 2 são null, vamos fazer os replies do 2 apontar para o 3 e adicionamos o 3 á stack.



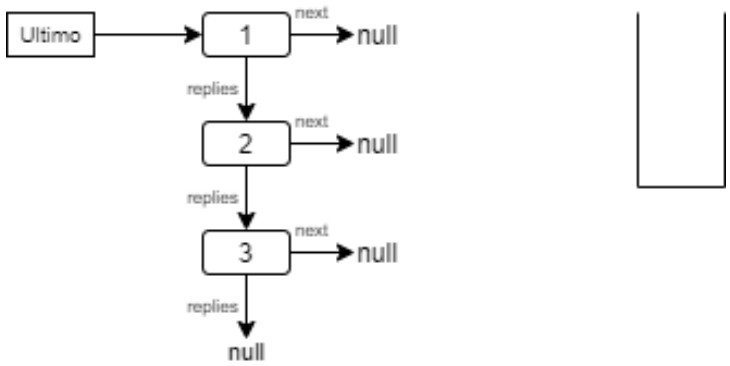
Seguidamente vamos encontrar o caso <comentario>{fimComentario} relativo ao comentário 3, aqui damos pop da Stack de Comentários e vamos colocar o apontador ultimo a ser comentario que acabou de sair da Stack, caso a Stack fique vazia, também damos pop da Start Condition de modo a ficarmos no Initial State, não é o caso desta vez.



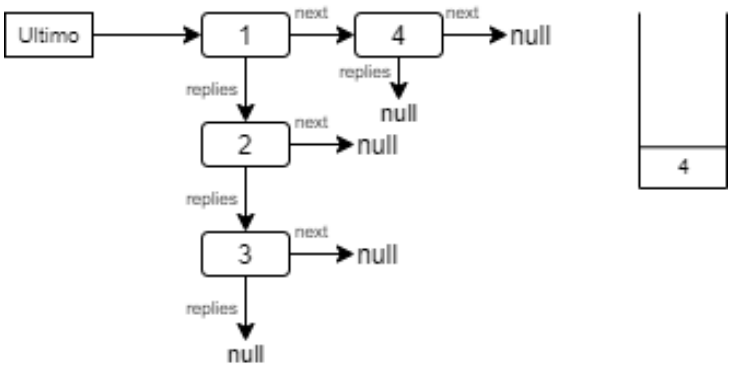
Seguidamente o Fim 2 é análogo.



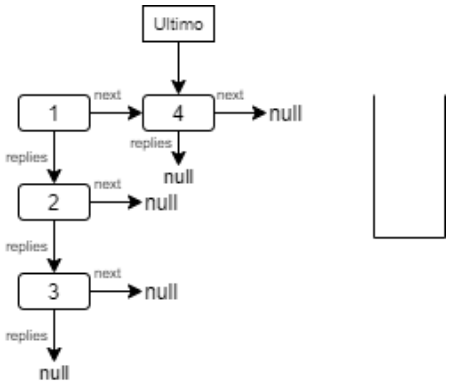
O fim da 1 também é análogo mas desta vez como a Stack fica vazia por isso damos pop da Start Condition.



Seguidamente chegamos ao inicio do comentário 4, como estamos no Initial State, ativamos o caso {inicioComentario}, criamos então a nossa estrutura, e desta vez vamos ligar ao next do último que deu pop, que neste caso é o comentário 1, por isso o último->next vai ser o atual (4) damos também push para a Stack do apontador Comentário e do estado Comentário.



No final vamos para o caso <comentario>{fimComentario} relativo ao comentário 4 e damos pop do mesmo da stack assim como pop do estado, finalizando assim a filtragem.



Usando então o programa da seguinte forma:

```
daniel@daniel-Aspire-A515-52G:~/Desktop/plc20TP1Gr $ make
daniel@daniel-Aspire-A515-52G:~/Desktop/plc20TP1Gr08$ ./tp1 exemplo.html out
```

Ficamos com o seguinte output no ficheiro "out.json"

```
{
"commentThread": [
{
    "id": "post-4786160376",
    "user": "Bino Pardal",
    "date": "Thursday, February 6, 2020 5:54 PM",
    "timestamp": "12 days ago",
    "commentText": "<p>a miss piggy deve dormir entre os dois...</p>",
    "likes": 0,
    "hasReplies": true,
    "numberOfReplies": 1,
    "replies": [
```

```
{
        "id": "post-4786946247",
        "user": "Coniuratio Adversum Omnia",
        "date": "Friday, February 7, 2020 7:36 AM",
        "timestamp": "11 days ago",
        "commentText": "<p>A tua mÃ£e faz de salchicha entre
        dois rotos? Devias ter mais respeito por quem te
        mete o leitinho na mesa, imbecil!</p>",
        "likes": 0,
        "hasReplies": true,
        "numberOfReplies": 1,
        "replies": [
{
            "id": "post-4787032868",
            "user": "Bino Pardal",
            "date": "Friday, February 7, 2020 10:34 AM",
            "timestamp": "11 days ago",
            "commentText": "<p>a tua indignaÃ§Ã£o Ã© tanta
            que nem te percebo, fascista!!!</p>Q",
            "likes": 0,
            "hasReplies": false,
            "numberOfReplies": 0,
            "replies": []
}
        ]
    }
    ]
},
{
    "id": "post-4786949839",
    "user": "Coniuratio Adversum Omnia",
    "date": "Friday, February 7, 2020 7:44 AM",
    "timestamp": "11 days ago",
    "commentText": "<p><b>Pensem no seguinte... <br>Ela
    estÃ¡ a fazer campanha para um grupo de eleitorado muito
    especifico... Tal como o Ventura o faz. Ela burra nÃ£o
    Ã© nada...</b></p><p><b>Quanto mais ela aparecer e
    quantos mais comentÃ¡rios a mandarem-na daqui para fora
    mais pessoas desse seu eleitorado alvo vÃ£o pensar em
    votar no partido que a apresentar a
    eleÃ§Ãµes.</b></p><p><b>Quanto mais Ã³dio ou
    impropÃ©rios lanÃ§arem Ã  Joacine mais estÃ£o a
    potenciar a sua eleiÃ§Ã£o. Pensem e sejamos todos um
    pouco mais inteligentes...</b></p><p><b><u><i>Querem-se
    livrar dela? Deixem-na cair no esquecimento</i></p>",
    "likes": 0,
    "hasReplies": false,
    "numberOfReplies": 0,
```

```
    "replies": []
    }
]
}
```

Sendo este o output pretendido para este exemplo.
Esperemos que com este exemplo tenha ficado clara a lógica do programa, passamos assim para a utilização do mesmo.

3.5 Utilização do Programa

O nosso Programa é de fácil utilização, usando o comando make no terminal:

```
$ make
```

Compilamos o nosso programa, após isso apenas temos de o executar da seguinte forma:

```
$ ./tp1 exemplo.html out
```

Substituindo exemplo pelo ficheiro *HTML* dos comentários a ser filtrado, e out pelo nome do ficheiro json onde quer guardar o resultado.
No final para eliminar-mos os ficheiros resultantes da execução *flex* e do *gcc* podemos executar o comando:

```
$ make clean
```

Capítulo 4

Conclusão

Este trabalho teve um elevado grau de importância pois ajudou-nos bastante a consolidar a matéria lecionada nas aulas sobre *FLex* e expressões regulares, com aplicações reais e úteis.

Apesar de nos deparar-mos com diversas dificuldades, o grupo esteve á altura para tais, como por exemplo os replies encadilhados que exigiu de nós montar uma stack e fazer uma lógica recursiva para ser possível a filtragem e posterior armazenamento de tais.

Estamos assim satisfeitos com o resultado deste trabalho, conseguimos filtrar correctamente os ficheiros, com uma solução clara e elegante satisfazendo todos os requesitos pedidos.

Finalizamos este relatório com certeza de termos acrescentado conhecimentos úteis e importantes para o nosso futuro, não só como filtragem em *FLex* mas também de estruturas de dados em C, trabalho e organização em grupo assim como escrita de documentos em *Latex*, algo que não tinhamos muita experiência prévia.

Apêndice A

Código do filtro (tp1.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "funcoes.h"

int len = 0;
int indice = 0;
int iniciado = 0;

extern FILE *yyin, *yyout;

Comentario lista = NULL;
Comentario ant = NULL;
Comentario atual = NULL;
Comentario ultimo = NULL;

int MAXSIZE = 20;
Comentario stack [20];
int posCabecaStack = -1;

%}

digito  [0-9]
espaco [\ \t\r\n]
alpha                 [a-zA-Z]
acentos            \xc3[\x80-\xbf]
letra                 {alpha}|{acentos}

username  ({letra}|{digito}|{espaco})+
timestamp ({alpha}|{digito}|{espaco})+
data             title\=\".*(AM|PM)\"
user             class\=\"post-body\".*data-username.*\"\>
inicioComentario class\=\"post\"{espaco}id\=\"post\-{digito}+\"
```

```
fimComentario    class\=\"show-children\-wrapper[' ']hidden\"
likes            data\-role\=\"likes\"\>{digito}+
inicioTexto      class\=\"post-message\".*\<div\>
inicioParagrafo  \<p\>
fimParagrafo     \<\/p\>

%option stack
%x comentario user_SC texto tempo

%%


{inicioComentario} {
yy_push_state(comentario);
atual = criaComentario ();
if (ultimo != NULL){
ultimo -> next = atual;
}

                                              push(atual,&posCabecaStack , MAXSIZE , stack);
if(iniciado == 0){
iniciado = 1;
lista = atual;
}
atual->id =  strdup(yytext+16);



}

<comentario>{fimComentario}   {


ultimo = pop(&posCabecaStack , stack);
if (isempty(&posCabecaStack)){
yy_pop_state();
}
}


<comentario>{inicioComentario} {




atual = criaComentario ();
```

```
ant = peek(&posCabecaStack , stack);
if (ant->replies == NULL){
ant-> replies = atual;
ant->numberOfReplies++;

}
else {
ant->numberOfReplies++;
ant = paraFim(ant->replies);
ant -> next = atual;

}


atual->id =  strdup(yytext+16);
push(atual,&posCabecaStack , MAXSIZE , stack);



}




<comentario>{data} {
atual->date =  strdup(yytext+6);
yy_push_state(tempo);
}

<comentario>{user} {                        yy_push_state(user_SC); }


<comentario>{likes} {                       atual -> likes =  atoi(yytext+18);}

<comentario>{inicioTexto} {                 yy_push_state(texto);
                                            len = 0;
atual->commentText = malloc(1);}


<tempo>{timestamp} {
atual -> timestamp = strdup(yytext);
yy_pop_state();
}



<user_SC>{username}  {
atual -> user = strdup(yytext);
```

```
yy_pop_state();

}


<texto>{inicioParagrafo}.*|\n*{fimParagrafo} {
len+= yyleng;
if (yytext[0] == '\n'){
len-=1;
atual->commentText= realloc(atual-> commentText, len * sizeof(char));
strcat(atual->commentText,yytext+1);
}
else {
atual->commentText= realloc(atual-> commentText, len * sizeof(char));
strcat(atual->commentText,yytext);
}


}

<texto>\<\/div\> {                          yy_pop_state(); }


<*>.|\n

%%


int yywrap()
{
    return 1;
}


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

if ((yyin = fopen(argv[1],"r") )== NULL ) {
    printf("Não consegui ler '%s'\n", argv[1]);
    return 0;
}

if ((yyout = fopen(strcat(argv[2],".json"),"w")) == NULL ) {
    printf("Não consegui escrever '%s'\n", argv[2]);
return 0;
}

yylex();
```

```
fprintf(yyout,"{\n\"commentThread\": [\n");
printaJson(lista,yyout,1);
fprintf(yyout,"]\n}");
}
```

Apêndice B

Código do header (funcoes.h)

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct comentario {

    char* id;
    char* date;
    char* user;
    char* commentText;
    char* timestamp;
    int likes;
    int numberOfReplies;
    struct comentario *replies;
    struct comentario *next;

}*Comentario;

Comentario criaComentario ();
void printaJson (Comentario l , FILE *yyout, int profundidade);
int isempty(int  * posCabecaStack);
int isfull(int * posCabecaStack, int MAXSIZE);
Comentario peek(int * posCabecaStack , Comentario * stack);
Comentario pop(int * posCabecaStack , Comentario * stack);
void push(Comentario data , int * posCabecaStack , int  MAXSIZE , Comentario * stack);
Comentario paraFim (Comentario c);
```

Apêndice C

Código das funções C (funcoes.c)

```
#include "funcoes.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


Comentario criaComentario (){

  Comentario h = NULL;
  h = malloc(sizeof(struct comentario));
  h->likes = 0;
  h->numberOfReplies = 0;
  h->replies = NULL;
  h->next = NULL;
  h->timestamp = "NA";
  h->date = "";
  h->user = "";
  h->id = "";
  h->commentText = "";

  return h;
}

char* repeteN (char arr[], int profundidade){



for ( int i = 0; i < profundidade; i++){
arr[i] = '\t';
}

arr[profundidade] = '\0';
return arr;
}
```

```
void printaJson (Comentario l , FILE *yyout, int profundidade){

Comentario aux = l;
char arr[profundidade+1];
repeteN(arr,profundidade);
while(aux != NULL){

fprintf(yyout,"{\n\t%s\"id\": %s,\n",arr,aux->id);
fprintf(yyout,"\t%s\"user\": \"%s\",\n",arr,aux->user);
fprintf(yyout,"\t%s\"date\": %s,\n",arr,aux->date);
fprintf(yyout,"\t%s\"timestamp\": \"%s\",\n",arr,aux->timestamp);
fprintf(yyout,"\t%s\"commentText\": \"%s\",\n",arr,aux->commentText);
fprintf(yyout,"\t%s\"likes\": %d,\n",arr,aux->likes);
fprintf(yyout,"\t%s\"hasReplies\": %s,\n",arr,(aux->numberOfReplies>0) ? "true" : "false");
fprintf(yyout,"\t%s\"numberOfReplies\": %d,\n",arr,aux->numberOfReplies);


Comentario repliesStruct = aux->replies;
fprintf(yyout,"%s\t\"replies\": [",arr);
if (repliesStruct == NULL){
fprintf(yyout,"]\n",arr);
} else {
fprintf(yyout,"\n");
printaJson(repliesStruct,yyout, profundidade +1);
fprintf(yyout,"\t%s]\n",arr);
}

if (aux -> next == NULL){
fprintf(yyout,"%s}\n",arr);
}
else {
fprintf(yyout,"%s},\n",arr);
}

aux = aux -> next;
}
}

int isempty(int  * posCabecaStack) {

   if((*posCabecaStack) == -1)
      return 1;
   else
      return 0;
}

int isfull(int * posCabecaStack , int MAXSIZE) {
```

```
   if((*posCabecaStack) == MAXSIZE)
      return 1;
   else
      return 0;
}

Comentario peek(int * posCabecaStack ,  Comentario stack[]) {
   return stack[(*posCabecaStack)];
}

Comentario pop(int * posCabecaStack , Comentario stack[]) {
   Comentario data;

   if(!isempty(posCabecaStack)) {
      data = stack[(*posCabecaStack)];
      (*posCabecaStack) = (*posCabecaStack) - 1;
      return data;
   }
}

void push (Comentario novo , int * posCabecaStack , int MAXSIZE , Comentario stack[]) {

   if(!isfull(posCabecaStack , MAXSIZE)) {
      (*posCabecaStack) = (*posCabecaStack) + 1;
      stack[(*posCabecaStack)] = novo;
   }
}


Comentario paraFim (Comentario c){
    Comentario ant = c;
while(c -> next != NULL){
c = c -> next;
ant = c;

}


    return ant;
}
```

Bibliografia

[1] Enunciado do trabalho prático,
https://elearning.uminho.pt/bbcswebdav/pid-1042824-dt-content-rid-3961864_1/courses/ 2021.850507_1/plc20tp1.pdf

[2] Template de um relatório em Latex fornecido pelo docente responsável pela U.C.,
https://elearning.uminho.pt/bbcswebdav/pid-1021529-dt-content-rid-3863379_1/courses/ 2021.850507_1/relprojLayout.pdf

[3] Ficheiros de comentários html,
http://www4.di.uminho.pt/ prh/Sol/

[4] Copiar conteúdo de um ficheiro para outro com lex, exemplo muito útil,
https://www.geeksforgeeks.org/lex-program-to-copy-the-content-of-one-file-to-another file/

[5] Regex101 usado para testar expressões e debuging,
https://www.regex101.com/

[6] Overleaf usado para escrever o latex,
https://www.overleaf.com/

[7] Draw.io usado para criar os diagramas,
https://app.diagrams.net/