

Processamento de Linguagens e Compiladores
Trabalho Prático 1
LCC - Relatório de Desenvolvimento
Grupo 0

João Coutinho



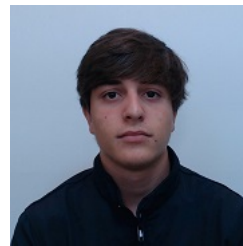
(A83545)

Eduardo Pereira



(A84098)

Tomás Valente



(A85172)

18 de novembro de 2020

Resumo

De maneira a ajudar um estudo sócio-linguístico da forma e do conteúdo dos comentários de uma dada notícia de Jornal, utilizamos a ferramenta *Flex* e desenvolvemos um programa que, através de expressões regulares, reconhece os padrões das *tags* de *HTML* de uma página online do jornal *Público* e converte a mesma num ficheiro *JSON*.

Conteúdo

1	Introdução	2
1.1	Estrutura do relatório	2
2	Análise e Especificação do Problema	3
2.1	Descrição e Enunciado	3
2.2	Requisitos	4
3	Funcionamento das nossas expressões regulares	5
3.1	Expressões Regulares	5
4	Codificação e Testes	6
4.1	Ferramentas utilizadas	6
4.1.1	MakeFile	6
4.1.2	<i>Flex</i> e expressões regulares	7
4.1.3	Stack e start conditions	7
4.2	Decisões e Problemas de implementação	8
4.3	Testes realizados e Resultados	8
5	Conclusão	9
A	Código do Programa	10

Capítulo 1

Introdução

Neste primeiro trabalho da unidade curricular de Processamento de Linguagens e Compiladores foram estabelecidos objetivos bem concretos com vista o desenvolvimento das nossas capacidades de manipulação das ferramentas do sistema operativo linux, mas essencialmente para aumentar a nossa capacidade e familiarização com a escrita de expressões regulares. Este tipo de expressões permite-nos automatizar Processadores de Linguagens Regulares que detetam determinados padrões em textos, guarda a informação delimitada por 2 desses padrões e exporta para um ficheiro novo a informação bem estruturada.

1.1 Estrutura do relatório

Capítulo 2 Capítulo onde faremos a especificação do problema e seus requisitos.

Secção 2.1 Descrição do problema e o seu respetivo enquadramento na área de estudo da unidade curricular.

Secção 2.2 Especificação dos Requisitos.

Capítulo 3 Explicação do funcionamento das nossas expressões regulares em detrimento do código *HTML*.

Secção 3.1 Tabela comparativa das tags *HTML* com as nossas expressões regulares, descrição e respetivas macros.

Capítulo 4 Codificação e Testes

Subcapítulo 4.1 Ferramentas utilizadas

Secção 4.1.1 Makefile utilizada

Secção 4.1.2 Flex e expressões regulares

Secção 4.1.3 Stack e start conditions

Secção 4.2 Decisões e Problemas de implementação

Secção 4.3 Testes realizados e Resultados

Capítulo 5 Conclusão relativa a todo o processo de resolução do problema.

Capítulo 2

Análise e Especificação do Problema

2.1 Descrição e Enunciado

2 Enunciados

Para sistematizar o trabalho que se pede em cada uma das propostas seguintes, considere que deve, em qualquer um dos casos, realizar a seguinte lista de tarefas:

1. Especificar os padrões de frases que quer encontrar no texto-fonte, através de ERs.
2. Identificar as acções semânticas a realizar como reacção ao reconhecimento de cada um desses padrões.
3. Identificar as Estruturas de Dados globais que possa eventualmente precisar para armazenar temporariamente a informação que vai extraindo do texto-fonte ou que vai construindo à medida que o processamento avança.
4. Desenvolver um Filtro de Texto para fazer o reconhecimento dos padrões identificados e proceder à transformação pretendida, com recurso ao Gerador FLex.

Para resolver os Enunciados 1 a 4, considere o seguinte contexto:

Para se fazer um estudo sócio-linguístico da forma e conteúdo dos comentários que uma dada notícia de Jornal suscitou, os dados relevantes à análise pretendida devem ser retirados automaticamente de cada ficheiro HTML extraído da versão online desse Jornal, devendo depois ser transformados no formato JSON a seguir mostrado.

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

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

Construa então um filtro de texto, recorrendo ao gerador FLex, que realize o processamento explicado, tendo em consideração que as respostas¹ que surjam a um dado comentário devem ser aninhadas, na forma de uma lista, dentro do campo "replies" do dito comentário, seguindo evidentemente o mesmo formato apresentado².

1 - Transformador Publico2NetLang

Concretize o enunciado genérico acima, considerando como base o ficheiro http://www4.di.uminho.pt/~prh/Publico/Publico_extraction_portuguese_comments_110.html que contém os comentários a uma notícia publicada no jornal O Público, o qual deve analisar com todo o cuidado. A ferramenta a desenvolver tem de funcionar bem para outros ficheiros recolhidos do mesmo jornal (na diretoria acima encontra mais exemplos para testar).

2.2 Requisitos

Conforme entendemos, é-nos requerido que construamos um filtro de texto capaz de processar e transformar o código-fonte de uma página de um Jornal online. O trabalho recai sob a criação de um programa em C que, recorrendo ao flex, transforme essa página *HTML* num ficheiro *JSON*. Posto isto, é claro que o objetivo final consiste na leitura clarificada do ficheiro *JSON* para a análise pretendida.

Capítulo 3

Funcionamento das nossas expressões regulares

3.1 Expressões Regulares

Para conseguirmos absorver um conjunto de letras ou números criámos 2 delimitadores para cada secção (seja ela user, id, date...) e absorver tudo o que se encontra entre eles. Para isso, precisamos de identificar no código fonte a tag que inicializa a secção (1º delimitador) e a tag que finda a mesma secção (2º delimitador).

HTML	DESCRIÇÃO	EXPRESSÕES REGULARES	MACRO
<ol(...)>	Início de uma lista de replies.	\<ol[^\>]+\>	INIT_ALL
	Fim de uma lista de replies.	\<\/ol\>	END_ALL
	Início de um comentário.	\	INIT_THR
	Fim de um comentário.	\<\/li\>	END_THR
<h5(...)>	Início da secção chamada de name.	\<h5[^\>]+\>	INIT_NAME
</h5>	Fim da secção name.	\<\/h5\>	END_NAME
<time(...)>	Início da secção time de um comentário.	\<time[^\>]+\>	INIT_TIME
</time>	Fim da secção time.	\<\/time\>	END_TIME
<a(...)>	Utilizado para ler a informação de uma secção time e h5.	\<a[^\>]+\>	INIT_INFO
	Fim da leitura de informação.	\<\/a\>	END_INFO
qualquerCharExceto"<"	Utilizado para percorrer a informação dentro de <a(...)>.	[^<]+	CHECK_INFO
<p> *	Início da secção de texto de um comentário.	\<p>[[:space:]]*	INIT_COMM
</p>	Fim da secção texto.	\<\/p\>	END_COMM
id="	Utilizada para encontrar o ID de um comentário.	id="	INIT_IDE
qualquerCharExceto" " "	Utilizada para ler um ID.	[^"]+	CHECK_IDE
>	Fim da procura de um ID.	\>	END_IDE

Capítulo 4

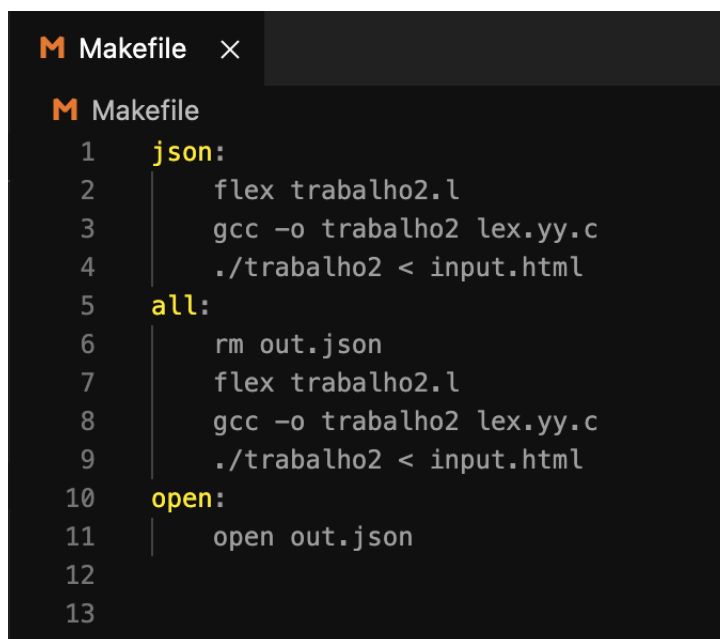
Codificação e Testes

4.1 Ferramentas utilizadas

4.1.1 MakeFile

Recorrendo à linguagem C e à ferramenta *Flex*, conseguimos retirar os dados relevantes à análise pretendida. Através de uma pequena *MakeFile*, simplificamos o processo, reduzindo o número de comandos a escrever no terminal.

- "*make json*" compila em flex e em c e cria o ficheiro "out.json".
- "*make all*" faz o mesmo que o "*make json*" mas com a particularidade que remove o ficheiro out.json criado anteriormente (é usado com mais frequência para obrigar a gerar o output de todas as vezes que testamos o programa).
- "*make open*" abre o ficheiro gerado.



```
M Makefile ×
M Makefile
1  json:
2      flex trabalho2.l
3      gcc -o trabalho2 lex.yy.c
4      ./trabalho2 < input.html
5  all:
6      rm out.json
7      flex trabalho2.l
8      gcc -o trabalho2 lex.yy.c
9      ./trabalho2 < input.html
10 open:
11     open out.json
12
13
```


4.1.2 *Flex* e expressões regulares

A ferramenta *flex* permite-nos reconhecer padrões dentro de ficheiros de texto. De uma forma mais rigorosa, conseguimos escrever expressões regulares que representam os padrões do texto que nos interessam e manipulá-los da forma que nos convier.

Na parte da criação dos nossos limitadores, primeiramente apenas nos preocupámos em defini-las de forma arbitrária, mais tarde percebemos que seria mais eficiente se separássemos as tags em tipos. As start conditions representam esses tipos. Em termos informáticos, é menos custoso no que diz respeito ao processamento, se utilizarmos determinados caracteres que representem "classes" de tags, assim conseguimos descomplicar as expressões regulares.

4.1.3 Stack e start conditions

Como o reconhecimento de tags e a sua posterior modelação tem algumas restrições bem definidas, decidimos usar o sistema de stack e as start conditions do *Flex*.

As start conditions permitem que exista um fluxo condicional de ações, ou seja permite-nos definir que determinadas tags só possam ser reconhecidas caso estejam dentro de outras. O sistema de stack traz-nos uma segurança de fluxo acrescida, pois permite que os nossos estados (start conditions) sejam rigorosamente controlados.

Como foi referido anteriormente agrupamos as tags por classes. Cada classe tem um caracter inicial que as distingue, utilizamos então esse caracter como uma expressão regular caraterístico de cada start condition e que permite que as tags sejam identificadas e associadas ao seu respetivo estado. Com o estado atribuído corretamente a cada expressão regular controlamos as suas substituições ordenadas através das funções do sistema stack: `yy_push_state(nome.do.estado)` e `yy_pop_state()`.

Definimos 6 estados representativos das classes de tags para o programa:

%s **ALL** Representa os comentários da notícia e consequentemente os comentários dos comentários.

%s **THR** Representa um comentário (Thread).

%s **IDE** Tags que contêm o ID de cada comentário.

%s **NAME** Tags que contêm o autor de cada comentário.

%s **TIME** Tags que contêm a data de cada comentário.

%s **INFO** Tags que contêm a informacao dentro das tags NAME/TIME.

%s **COMMENT** Tags para o conteudo do comentário.

4.2 Decisões e Problemas de implementação

Numa primeira fase tivemos alguma dificuldade em perceber o funcionamento do sistema de stack e start conditions, contudo percebemos rapidamente que se tratava quase como que de uma representação de um autómato determinista.

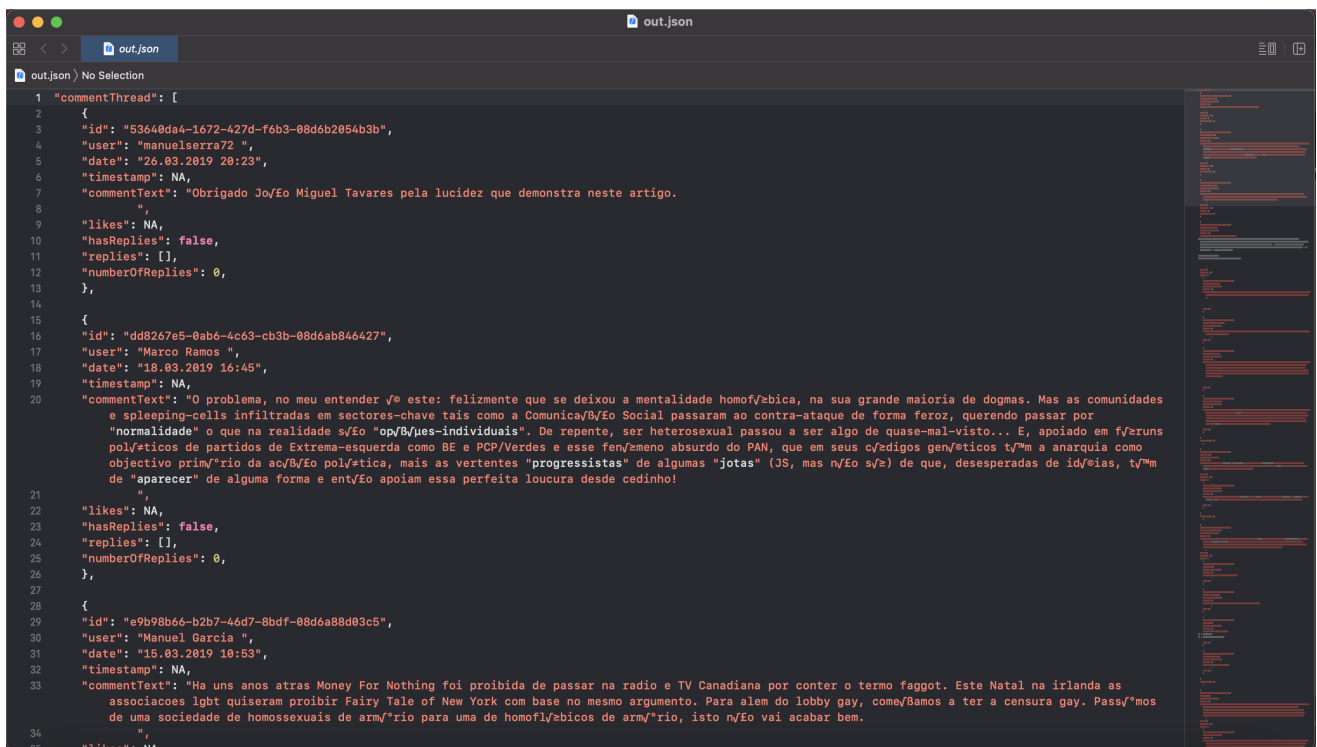
Em termos de expressões regulares não tivemos grandes dificuldades na sua representação, das vezes em que tínhamos erros de compilação ou resultados errados no output devia-se a problemas no fluxo dos estados. Em algumas ocasiões não iniciávamos determinado estado com o comando BEGIN(estado) ou, na maioria das vezes, não garantíamos que determinado padrão reconhecido tivesse o seu estado na stack para conseguir entrar no fluxo.

4.3 Testes realizados e Resultados

Após direcionarmos o nosso ficheiro de teste como input para o nosso programa, obtemos o ficheiro de output *JSON* (escrevendo no terminal *"make all"*). Ao abrirmos o ficheiro (ou escrevendo no terminal *"make open"*) obtivemos o seguinte resultado:

Exemplo de uma página do Jornal *Público* que demos como input do nosso programa:

http://www4.di.uminho.pt/prh/Publico/Publico_extraction_portuguese_comments33.html.



```
1 "commentThread": [
2   {
3     "id": "53640da4-1672-427d-f6b3-08d6b2054b3b",
4     "user": "manuelsera72 ",
5     "date": "26.03.2019 20:23",
6     "timestamp": NA,
7     "commentText": "Obrigado Jo/Jo Miguel Tavares pela lucidez que demonstra neste artigo.
8   },
9   {
10    "likes": NA,
11    "hasReplies": false,
12    "replies": [],
13    "numberOfReplies": 0,
14  },
15  {
16    "id": "dd8267e5-0ab6-4c63-cb3b-08d6ab846427",
17    "user": "Marco Ramos ",
18    "date": "18.03.2019 16:45",
19    "timestamp": NA,
20    "commentText": "O problema, no meu entender /e este: felizmente que se deixou a mentalidade homof/zbica, na sua grande maioria de dogmas. Mas as comunidades
    e s/sleeping-cells infiltradas em sectores-chave tais como a Comunica/B/fo Social passaram ao contra-ataque de forma feroz, querendo passar por
    "normalidade" o que na realidade s/fo "op/B/ues-individuais". De repente, ser heterossexual passou a ser algo de quase-mal-visto... E, apoiado em f/ziruns
    pol/fticos de partidos de Extrema-esquerda como BE e PCP/Verdes e esse fen/mento absurdo do PAN, que em seus c/edigos gen/eticos tv/mm a anarquia como
    objectivo prim/rio da ac/B/fo pol/ftica, mais as vertentes "progressistas" de algumas "jotas" (JS, mas n/fo s/ze) de que, desesperadas de id/eias, tv/mm
    de "aparecer" de alguma forma e ent/fo apoiam essa perfeita loucura desde cedinho!
21  },
22  {
23    "likes": NA,
24    "hasReplies": false,
25    "replies": [],
26    "numberOfReplies": 0,
27  },
28  {
29    "id": "e9b98b66-b2b7-46d7-8bdf-08d6a88d03c5",
30    "user": "Manuel Garcia ",
31    "date": "15.03.2019 10:53",
32    "timestamp": NA,
33    "commentText": "Ha uns anos atras Money For Nothing foi proibida de passar na radio e TV Canadiana por conter o termo faggot. Este Natal na Irlanda as
    associacoes lgbt quiseram proibir Fairy Tale of New York com base no mesmo argumento. Para alem do lobby gay, come/amos a ter a censura gay. Pass/amos
    de uma sociedade de homossexuais de arm/rio para uma de homofl/zbicos de arm/rio, isto n/fo vai acabar bem.
34  },
35  {
36    "likes": NA,
```

Capítulo 5

Conclusão

Consumada a realização do primeiro trabalho prático podemos seguramente afirmar que não era uma tarefa facilmente exequível, relativamente à parte mais teórica e matemática das expressões regulares. Felizmente o nosso curso deu-nos uma bagagem forte que pudemos agora por à prova em situações concretas. Como estamos habituados na nossa área, qualquer nova ferramenta ou linguagem que necessitemos de utilizar exige prática. No caso do *Flex*, fruto de não existir muita documentação online, a tarefa parecia-nos um pouco mais complicada mas graças às aulas e ao manual do próprio flex julgamos que conseguimos orientar e estruturar as nossas implementações em código.

Em suma, é opinião unânime dos membros do nosso grupo que este primeiro trabalho prático foi um catalisador do interesse na área do Processamento de Linguagens, para além de todos os benefícios que nos trouxe em termos de expansão de conhecimento e desenvolvimento da componente prática.

Apêndice A

Código do Programa

```
%{
#include <stdio.h>
#include <strings.h>
#include <stdlib.h>

/* Declaracoes C diversas */

int flag=0;
int reply = 1;
int isReply;

FILE * fp;

char* openChav = "\t{\n";
char* openChavReply = "\t\t{\n";
char* closeChav = "\t},\n\n";
char* closeChavReply = "\t\t},\n\n";

%}

%option stack

%s COMMENT TIME NAME INFO ALL THR IDE

%%

<INITIAL>{INIT_ALL}          { BEGIN ALL ; yy_push_state(ALL) ; fprintf(fp, "\"commentThread\": [\n") ;
                               reply = 1; isReply = 0;}

<ALL>{INIT_THR}              { yy_push_state(THR); if(isReply == 1){ fprintf(fp,"%s", openChavReply); }
                               else {fprintf(fp, "%s", openChav); } ; reply++ ; }

<ALL>{END_ALL}               { BEGIN ALL ; fprintf(fp, "\t]\n") ; reply-- ; if(isReply != 0){
                               fprintf(fp, "\t\"numberOfReplies\": %d,\n\t%s",reply,closeChav) ; }
                               isReply = 0 ; reply = 1 ; }

<THR>{INIT_NAME}             { yy_push_state(NAME) ; }
<NAME>{INIT_INFO}           { yy_push_state(INFO) ; flag = 0 ; }
<NAME>{END_NAME}            { yy_pop_state(); }
<THR>{INIT_TIME}            { yy_push_state(TIME) ; }
<TIME>{INIT_INFO}           { yy_push_state(INFO) ; flag = 1 ; }
<TIME>{END_TIME}            { yy_pop_state(); }
<THR>{INIT_COMM}            { yy_push_state(COMMENT) ; flag = 2 ; }

<COMMENT>{CHECK_INFO}       { if(isReply == 1){ fprintf(fp,"\t\t\"commentText\": \"%s\", \n\t\t\"likes\": NA,\n", yytext); }
                               else {fprintf(fp, "\t\t\"commentText\": \"%s\", \n\t\t\"likes\": NA,\n", yytext); }}

<COMMENT>{END_COMM}         { yy_pop_state() ; }
```

```

<THR>{INIT_IDE}          { yy_push_state(IDE) ; }

<IDE>{CHECK_IDE}          { if(isReply == 1){ fprintf(fp, "\t\t\"id\": \"%s\", \n", yytext); }
                           else {fprintf(fp, "\t\t\"id\": \"%s\", \n", yytext); }}

<IDE>{END_IDE}            { yy_pop_state();}

<THR>{END_THR}            { yy_pop_state(); if(isReply == 1){ fprintf(fp, "%s", closeChavReply); }
                           else {fprintf(fp, "\t\t\"hasReplies\": false, \n\t\t\"replies\": [], \n\t\t\"numberOfReplies\": 0, \n %s",
                           closeChav); } ;}

<THR>{INIT_ALL}           { yy_push_state(ALL); fprintf(fp, "\t\t\"hasReplies\": true, \n") ;
                           fprintf(fp, "\t\t\"replies\": [\n") ; isReply = 1; reply = 1; }

<INFO>{CHECK_INFO}        {if(flag == 0) { if(isReply == 1){ fprintf(fp, "\t\t\t\"user\": \"%s\", \n", yytext); }
                           else {fprintf(fp, "\t\t\t\"user\": \"%s\", \n", yytext); }}
                           else{if(flag == 1){ if(isReply == 1){
                               fprintf(fp, "\t\t\t\"date\": \"%s\", \n\t\t\t\t\"timestamp\": NA, \n", yytext); }
                               else {fprintf(fp, "\t\t\t\"date\": \"%s\", \n\t\t\t\t\"timestamp\": NA, \n", yytext); }}}}}

<INFO>{END_INFO}          { yy_pop_state(); }

%%
int yywrap(){
    return(1);
}

int main(){

    fp = fopen("out.json", "w");

    yylex();
    return 0;
}

```

Macros que definimos no nosso ficheiro C para representar as expressões regulares:

INIT_ALL	\<ol[~>]+>
END_ALL	\<\/ol\>
INIT_NAME	\<h5[~>]+\>
END_NAME	\<\/h5\>
INIT_TIME	\<time[~>]+\>
END_TIME	\<\/time\>
INIT_INFO	\<a[~>]+\>
END_INFO	\<\/a\>
CHECK_INFO	[^\\<]+
INIT_COMM	\<p\>[[[:space:]]]*
END_COMM	\<\/p\>
INIT_THR	\<li
END_THR	\<\/li\>
INIT_IDE	t\~id\="
CHECK_IDE	[^"]+
END_IDE	\"">