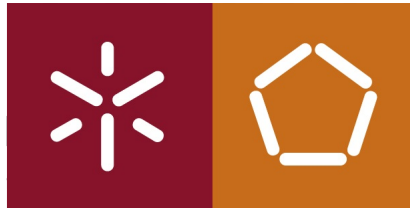


UNIVERSIDADE DO MINHO



PROCESSAMENTO DE LINGUAGENS

DEPARTAMENTO DE INFORMÁTICA

---

# Trabalho Prático 1

## Transformador Publico2NetLang

---

*Grupo 45:*

João Abreu

Hugo Matias

João Coutinho

*Número:*

A84802

A85370

A86272

5 de Abril de 2020

# Conteúdo

<b>1</b>	<b>Problema</b>	<b>1</b>
<b>2</b>	<b>Solução</b>	<b>2</b>
2.1	Estrutura . . . . .	2
2.2	Filtro de texto . . . . .	4
2.2.1	Início e fim de Comment Thread . . . . .	4
2.2.2	Estado Comum . . . . .	5
2.2.3	Estado P . . . . .	6
2.2.4	Estado R . . . . .	7
2.2.5	Estado T . . . . .	8
2.3	Problemas surgidos e respectivas soluções . . . . .	8
2.4	Output . . . . .	9

# 1. Problema

No âmbito da avaliação prática da cadeira de Processamento de Linguagens, foi escolhido como problema a resolver o transformador Publico2NetLang. Foi pedido que fosse desenhado um filtro de texto que, sendo-lhe fornecido um ficheiro HTML com comentários a um artigo de um jornal, produzisse um novo ficheiro no formato JSON com a seguinte estrutura:

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

O ficheiro HTML lido tem, em primeiro lugar, um campo delimitado por uma *tag* `<h3>` onde consta o número total de comentários ao artigo. De seguida, estão discriminados todos os comentários feitos ao artigo dentro de uma *tag* `<ol>` (lista ordenada). Dentro dessa lista, o início de um comentário é marcado pela *tag* `<li>` (item de uma lista). Dentro de um comentário encontra-se a informação relativa a esse comentário e também todas as respostas feitas a esse comentário. A sua informação encontra-se dentro de uma `<div>`, também ela dividida em duas `<div>`, uma com meta-informações e outra com o texto da resposta em si, e as respostas seguem a estrutura de um comentário e incluem-se dentro de uma *tag* `<ol>` pertencente ao respectivo comentário a que dão resposta.

## 2. Solução

### 2.1 Estrutura

Como primeiro passo para a resolução do problema, analisámos profundamente a construção do ficheiro HTML fornecido, que foi enunciada acima. Após uma compreensão generalizada da mesma, foi criada uma estrutura que corresponde aos comentários lidos do ficheiro HTML, onde é armazenada a informação que constará do ficheiro JSON:

```
typedef struct comment {
    char *id;
    char *user;
    char *date;
    char *timestamp;
    char *commentText;
    int likes;
    int hasReplies;
    int numberOfReplies;
    struct comment *next;
} * Comment;
```

Foi também desenhado um conjunto de funções que operam sobre a estrutura, nomeadamente funções de alocação e libertação de memória para a estrutura, *setters* para todos os campos da estrutura *Comment*, e funções de *print* de comentários e respostas.

```
Comment mkComment();
void unmkComment(Comment c);
void addComment(Comment comment, Comment reply);
void printfComment(Comment c, int nr_comment, int nr_comment_total);
void printfReply(Comment c, int f);
void setId(Comment c, char *id);
void setUser(Comment c, char *user);
void setDate(Comment c, char *timestamp);
void setTime(Comment c, char *timestamp);
void setCommentText(Comment c, char *commentText);
void setLikes(Comment c, int likes);
void setHasReplies(Comment c, int hasReplies);
void setNumberOfReplies(Comment c, int      numberOfReplies);
```

De seguida, começou a ser estruturado o filtro de texto a ser usado.

Também foi criada um ficheiro chamado *Array.c* onde é armazenado dinamicamente em memória o conteúdo dos comentários. Foi desenhado da seguinte forma:

```
typedef struct {
    char *array;
    int used;
    int size;
} Array;

void initArray(Array *a, int initialSize);
void insertArray(Array *a, char element);
void closeArray(Array *a);
char* getText(Array *a);
void freeArray(Array *a);
```

No fim de cada comentário o campo *array* da estrutura *Array* é colocado na estrutura *Comment* enunciada acima usando a função *strdup* de modo a não partilhar apontadores e manter o encapsulamento.

## 2.2 Filtro de texto

### 2.2.1 Início e fim de Comment Thread

```
\[\<h3.+c    {      // Primeira linha do ficheiro HTML
                  printf("{\n    \"commentThread\": [");
                  yytext[yytext-2] = '\0';
                  nr_comment_total = atoi(yytext+97);
                  }

\</o1>      {      // Última linha do ficheiro HTML
                  printf("\n    ]\n}");
                  }
```

Inicialmente, o começo e fim da *Comment Thread* eram definidas na função *main* do respectivo programa *FLex*. Porém, ao longo do desenvolvimento e solidificação da solução, foram implementadas duas expressões regulares: uma que indica o início da *Comment Thread* e outra que indica o fim. É na do início que é contado também o número total de comentários.

### 2.2.2 Estado Comum

```
<*>{
    data\~comment~-id=\"[^\"]+
                                {
                                setId(piece,yytext+17);
                                }

    rel=\"nofollow\"\\>[^\<]+
                                {
                                yytext[yytext-1] = '\\0';
                                setUser(piece,yytext+15);
                                }

    \\<h5[ ]class=\"comment__author\"\\>[^\<]+
                                {
                                setUser(piece,\"Conta desactivada por ..\");
                                }

    \\<a[ ]class=\"comment__permalink\"\\>[^\<]+
                                {
                                char* token = strtok(yytext+30, " ");
                                setDate(piece,strdup(token));
                                token = strtok(NULL, "-");
                                setTime(piece,strdup(token));
                                }

    \\<p\\>
                                {
                                initArray(&comment,13);
                                yy_push_state(T);
                                }
}
```

As expressões regulares apresentadas em cima foram usadas para retirar os dados de cada comentário, bem como das suas respectivas respostas. Encontra-se também a inicialização de uma estrutura *Array* onde será armazenado o conteúdo da mensagem do comentário.

### 2.2.3 Estado P

```
<P>{
    \<ol      {
                                yy_push_state(R);
                                }
    \<\/li>    {
                                printfComment(com,nr_comment,nr_comment_total);
                                unmkComment(com);
                                yy_pop_state();
                                }
}
```

Como foi identificado acima na enunciação do problema, uma lista de comentários está delimitado pela *tag* *<ol>*, sendo um comentário compreendido pela *tag* *<li>*. Com isto em mente, foi desenhado o estado P. Este estado é usado para identificar o início de uma lista de respostas a um comentário ao artigo e para identificar o fim de um comentário ao artigo. No primeiro caso coloca a máquina no estado R em que serão tratadas as respostas ao comentário identificado, e no segundo, após ter sido processado anteriormente todo o comentário e terem sido preenchidos correctamente todos os seus campos, imprime esse mesmo comentário e prepara a leitura do comentário seguinte.



### 2.2.4 Estado R

```
<R>{
  \</ol>      {
                yy_pop_state();
              }
  \<li        {
                piece = mkComment();
                nr_comment++;
              }
  \</li>      {
                addComment(com,piece);
              }
}
```

O estado R foi desenhado para lidar com as respostas aos comentários ao artigo. Tem, portanto, três utilizações bem definidas: identificar o fim de uma lista de respostas a um comentário ao artigo, identificar o início de uma resposta a um comentário ao artigo, e o fim da mesma. No primeiro caso, findado o propósito do estado, é simplesmente retirado da máquina. No segundo, prepara um comentário que será posteriormente definido. Finalmente, no terceiro, adiciona o comentário anteriormente encontrado à lista de respostas a que pertence.

### 2.2.5 Estado T

```
<T>{
    [ ][ ][ ]+    {}
    [ ][ ]        {
                        insertArray(&comment,' ');
                    }
    [\n\t\r]+     {}
    \"            {
                        insertArray(&comment,'\\');
                        insertArray(&comment,'\"');
                    }
    \\<\/p>        {
                        closeArray(&comment);
                        setCommentText(piece,getText(&comment));
                        freeArray(&comment);
                        yy_pop_state();
                    }
    .              {
                        insertArray(&comment,yytext[0]);
                    }
}
```

O estado T foi desenhado com a intenção de "limpar" alguns caracteres brancos cuja remoção faz sentido, apesar de pertencerem aos comentários a tratar. A inclusão destes espaços, tabulações, e termos de linha desformatariam aquilo que, na nossa óptica, seria o cumprimento mais correcto da estrutura pedida no ficheiro JSON. Desta forma, a máquina é colocada neste estado sempre que se encontra o fim de um comentário, procedendo-se assim à sua "limpeza" para ser posteriormente impresso devidamente. É também aqui que preenchemos o campo do comentário da estrutura de dados principal *Comment*.

## 2.3 Problemas surgidos e respectivas soluções

### Perguntas

1. No fim de cada bloco JSON de um comentário coloca-se uma vírgula no fim. Como não colocar no último?
2. Como saber o tamanho certo do *buffer* para armazenar o comentário?
3. Como se sabe quantos *likes* uma resposta tem?
4. Como contar o número de respostas a respostas de um comentário principal?

### Respostas

1. Reparámos que na estrutura do ficheiro HTML está presente o número total de comentários. Por isso, apenas tivemos de os ir contando e saberíamos que no último não precisaríamos de escrever uma vírgula.
2. Como foi referido anteriormente, criámos um ficheiro que contém uma *struct* e funções que tratam dinamicamente do tamanho de uma *string* chamado *Array.c*. Não é 100% eficaz pois o seu algoritmo baseia-se na duplicação do tamanho quando este *buffer* fica preenchido, de modo que pode alocar espaço de que não necessitará mais tarde.

3. Como não encontramos nenhuma referência a número de *likes* no ficheiro HTML, optámos por escolher como valor padrão 0.
4. Do nosso ponto de vista, as respostas referem-se apenas ao comentário principal, de modo que as repostas têm sempre os campos *numberOfReplies* e *hasReplies* iguais a 0. Aachamos que esta é a escolha correcta pois, seguindo a lógica da estruturação dos comentários ao artigo e das suas respostas, as respostas às respostas teriam de estar contidas numa *tag <ol>* dentro da *tag <li>* referente à resposta ao comentário ao artigo. Como isto não acontece, assumimos que existe apenas um nível de profundidade nesta estrutura de dados de respostas, que é a lista das respostas aos comentários ao artigo.

## 2.4 Output

```
{
  "commentThread": [
    {
      "id": "06de7129-6167-49cd-d338-88d743683e5c",
      "user": "Pedralar",
      "date": "03.10.2019",
      "timestamp": "21:11",
      "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 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.",
      "likes": 0,
      "hasReplies": 0,
      "numberOfReplies": 0,
      "replies": []
    },
    {
      "id": "2c5940ee-754e-41f7-d893-88d748126a85",
      "user": "PEDROA Santos",
      "date": "03.10.2019",
      "timestamp": "19:39",
      "commentText": "Como vamos de Salgado, Bava, Granadeiro e Marquês? Isso deve perguntar a Costa. Por outro lado referir que Rio esvaziou a cidade do Porto é desonesto. Sei que alguma gente se refere a Rio como \"contabilista, contra as artes e o Fcp\". Esse é um discurso patético. Há que reconhecer valor no que fez pela cidade, e que o eleitorado sempre reconheceu e reforçou. Nunca existiram familygates, protecção a grupos económicos, pelo contrário.",
      "likes": 0,
      "hasReplies": 0,
      "numberOfReplies": 0,
      "replies": []
    },
    {
      "id": "7e2b2bb1-7760-463c-c70e-88d747278411",
      "user": "José Carvalho",
      "date": "03.10.2019",
      "timestamp": "10:44",
      "commentText": "Que pobreza de argumentos, Pedro Abrunhosa. Que pobreza mesmo! Lembra o Coliseu e as correntes do \"daqui não saio\" (será que já saiu?). Não lembra nem ao diabo (olá!), dizer que Rui Rio está a tirar a massa crítica do PSD. Confesso que prefiro Pacheco Pereira a Miguel Relvas e companhia. Mas afinal Abrunhosa defende o passado PSD? Ou defende o PS com maioria absoluta? E contra a esquerda do PSD? bom ter um jornal que estrategicamente o deixa vir aqui dizer umas coisitas, não é? bom sentido de oportunidade. Ainda vou ver por aqui o Francisco Assis... Tema? Aquil vai. O PS deve juntar-se à direita democrática, se e apenas se não obtiver uma maioria absoluta, imprescindível para o avanço social de Portugal e contra os perigos totalitários. Pedro Abrunhosa já ganhou o dia",
      "likes": 0,
      "hasReplies": 0,
      "numberOfReplies": 0,
      "replies": []
    },
    {
      "id": "48458a7c-d75f-4bd8-4565-88d7471e83ae",
      "user": "Clarisse Vilanova",
      "date": "03.10.2019",
      "timestamp": "06:34",
      "commentText": "Depois de se ter transformado num músico medíocre, Abrunhosa infecta de mediocridade também as suas opiniões.",
      "likes": 0,
      "hasReplies": 0,
      "numberOfReplies": 0,
      "replies": []
    }
  ]
}
```

Figura 2.1: Parte inicial do output resultante.