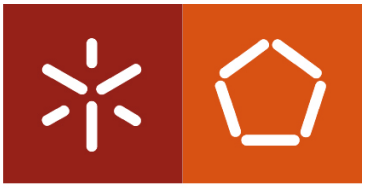

Escola de Engenharia

Universidade do Minho

Processamento de Linguagens

Primeiro Trabalho Prático

FLex

A78485 - Diogo Araújo, A78957 - Diogo Nogueira

1 de Abril de 2019

Resumo

Este primeiro trabalho prático surge no âmbito da Unidade Curricular de Processamento de Linguagens e destina-se essencialmente à utilização do sistema de gestão de filtros de texto em C, de nome FLex. Através dele, pretende-se escrever Expressões Regulares que depois permitam filtrar/transformar todo um ficheiro de texto .txt com base em regras de produção Condição-Ação.

Conteúdo

Capítulo 1

Processador Jornal Angolano

Neste capítulo irá ser apresentado, de um modo geral, aquilo que é o trabalho proposto em si e o problema que se pretende ver resolvido.

1.1 Enunciado

Jornal Angolano consiste num conjunto de notícias que se encontram parametrizadas pelo seu ID, data/autor, categoria, título e ainda uma coleção de Tags que ajudam a definir a notícia em si. Além desta base, existe também o texto correspondente a cada uma dessas. Este conjunto de informação é então universal a todas as notícias e é sobre este padrão que se pretende manipular o ficheiro .txt que agrupa o extrato da Folha 8 deste Jornal. Ao todo, o grupo preveu um total de aproximadamente 8000 notícias que terão de ser tratadas de igual modo para se poder criar depois os respetivos HTML.

1.2 Descrição do Problema

Como já foi referido numa fase inicial, todo este projeto baseia-se na interpretação, filtragem e posterior transformação de texto, proveniente de um ficheiro que aglomera todas as noticias de uma folha de um Jornal Angolano. Por ser um ficheiro que enumera aquilo que define uma notícia, é perfeitamente normal existirem informações que depois se tornam irrelevantes ao olhos de quem pretende manipular este conteúdo. Daí surgir a necessidade de interpretar primeiro. Observar aquilo que realmente importa e pensar também em termos de FLex e Expressões Regulares, tentando já facilitar a maneira como o algoritmo pode ser escrito. Ao fazer-se isto, consegue-se eliminar a informação desnecessária e delinear as alíneas de todo o projeto.

Perante esta descrição e nunca esquecendo os requisitos principais do processo de filtragem de informação, irá ser realizada uma análise do problema seguida das estruturas criadas para o efeito e do código FLex que vai de acordo com todas as decisões/implementações na qual o projeto se baseia.

Capítulo 2

Decisões e Implementação

2.1 Análise do Problema

Estando o problema descrito, é tempo de ser feita uma análise pormenorizada. Tomando uma notícia como referência, nota-se desde logo uma igualdade na disposição dos parâmetros de cada notícia. É precisamente esta igualdade que permite estruturar o código FLex para que exista um processo sequencial entre cada notícia.

Veja-se então um extrato daquilo que é considerado uma notícia. Note-se que esse mesmo extrato apenas apresenta a informação relevante que irá ser depois estudada e discutida.

```
<pub>

#TAG: tag:{Apreensão} tag:{Droga} tag:{EUA} ...

#ID:{post-1643 post type-post status-publish format...}

Lusofonia

100 milhões de dólares em droga

-----------------------------------------------------
PARTILHE VIA:
#DATE: [116eb] Redacção F8 | 30 de Setembro de 2014
[aaa15]

Um navio de carga com bandeira são-tomense que transportava mais ...

Etiquetas: ...

</pub>
```

O início de cada notícia é identificado pela tag <pub>

1. **Tags:** Começa-se com as várias tags da notícia. Cada uma destas tags tem também uma sintaxe que acaba por ser depois importante na implementação do código FLex. Aproveitando o destaque colorido a nível de código, entende-se então que a única informação que se precisa de filtrar é apenas a que se encontra cercada pelas chavetas. Este processo de filtragem deve então continuar até não existir mais tags a procurar.

2. **Id:** O identificador da notícia vem logo após o conjunto das tags. Este identificador encontra-se agregado a uma String "post-"que acaba por ser necessária para solucionar um dos requisitos principais do projeto.A ideia é aproveitar toda a junção da String com o Id, uma vez que será esse o nome dado ao ficheiro html da notícia.

3. **Categoria:** Na linha seguinte ao terminar toda a informação anterior, está contida uma nova String, destacada do restante e que diz respeito à categoria.

4. **Título:** O título tem uma ideia similar à categoria, individualizando-se também do conteúdo posterior. Isso facilita porque depois se sabe que a String que representa o título, terá de ir até ao primeiro parágrafo.

5. **Autor/Data:** Antes da data surgem outras informações irrelevantes e que não são precisas filtrar para depois se gerar o HTML. Aquilo que realmente importa é o que vem após o caractér "]", sendo esse o ponto de referência para a filtragem desta informação.

6. **Texto:** Após se filtrarem todas as informações da notícia, vem o texto em si. Aquilo que define o seu fim é então a tag que sinaliza que a notícia terminou.

O fim de cada notícia é identificado pela tag </pub>

Foi sobre esta base de pensamento que as expressões regulares foram desenvolvidas. Na subsecção dedicada à Implementação do Código FLex irá ser debatida toda a temática da resolução dos requisitos e de que modo esta análise inicial contribui para mitigar tudo isso. Antes de se chegar a esse ponto, vão ser abordadas as estruturas que foram necessárias criar para ir fazendo todo o parse do ficheiro .txt.

2.2 Estruturas de Dados

As Estruturas de Dados surgem com a necessidade de produzir um parse de todas as notícias de maneira mais eficiente. Inicialmente, pensou-se em dividir o ficheiro .txt em vários, acabando por gerar tantos ficheiros quanto o número de notícias presentes na folha do Jornal. Isto pareceu facilitar, já que com isto se podia seguir o mesmo princípio abordado nas aulas práticas, poupando bastante código e diminuindo de alguma forma a dificuldade em desenvolver as expressões regulares. Mas, dada a extensa dimensão de notícias e tendo em conta o que se ia sofrer em termos de otimização, a criação de uma ou mais estruturas acabou por se tornar a melhor solução.

```
struct noticia {
    List tags;                  // Tags da noticia.
    char *id;                   // Id da noticia.
    char *categoria;            // Categoria da noticia.
    char *titulo;               // Titulo da noticia.
    char *autorData;            // Autor/Data da noticia.
    char *texto;                // Texto da noticia.
    struct noticia *proxima;    // Apontador para a pr xima noticia.
};
```

A ideia consistiu em criar uma estrutura que representasse cada notícia. Assim, a estrutura teria como variáveis os parâmetros que compõem a mesma, formando então uma espécie de Lista Ligada em que cada notícia apontasse para a seguinte. A grande diferença acaba por se concentrar nas *tags*. Como temos um conjunto delas, aprupamo-las numa **List**, permitindo depois iterar cada uma destas tags pertencentes a cada notícia. Com isto, existiu também a criação de uma mini estrutura para as tags, que nada mais é que uma lista ligada contendo as várias tags da notícia. (Esta implementação permite precisamente guardar/iterar as várias tags, para que depois sejam imprimidas no HTML da notícia em si).

Como o ficheiro do Jornal disponibilizado é linear de notícia a notícia, bastou declarar esta estrutura como variável global do código FLex. Baseando esta estrutura com o pensamento desenvolvido inicialmente, obteve-se um procedimento universal a todas as notícias:

(Ter em atenção que isto obrigou a criar uma variável global por cada parâmetro de notícia, bem como uma **Noticia** em si que depois é atualizada consoante o que se filtra.)

1. Sempre que se encontra a tag <pub>, inicializa-se a estrutura em si.
2. Obriga-se o FLex a procurar a primeira ocorrência de tag, filtrando apenas o texto da tag em si. Faz-se isto até à última tag, utilizando como ajuda uma espécie de ciclo entre condições FLex, que terminará quando na nova linha seguida de parágrafo for apresentado o Id;
3. Filtra-se a String "post-"que nos trará a informação do Id da notícia;
4. (...) Segue-se exatamente o mesmo princípio para a Categoria, Título, Data e Texto.
5. Ao encontrar a tag </pub>inicia-se o mesmo processo mas agora para a próxima notícia.

O panorama passa por isto. Por ir analisando o ficheiro sequencialmente e atualizando os parâmetros da variável global que representa a **Noticia**. Quando se terminar de fazer toda a notícia, guarda-se essa **Noticia** que foi totalmente definida e preenchida, iniciando-se então uma nova iteração. Foi sobre este pensamento que o código FLex foi desenvolvido e idealizado. Atente-se mais pormenorizadamente na secção imediatamente abaixo.

2.3 Implementação Código FLex

Ficou claro que todo o processo de construção/processamento das várias expressões regulares é dependente da estrutura e do pensamento até então estudado e definido. O conceito de ir guardando as várias informações das várias notícias que da estrutura fazem parte, permitiu-se perceber que seria essencial criar várias *Start Conditions* que forçariam o FLex a tomar um caminho sequencial entre cada uma destas condições.

INICIO – TAG – ID – CATEGORIA – TITULO – DATA – TEXTO – FIM

Este foi o pensamento inicial. Filtrar à medida que se lê. Passar de uma *Start Condition* para a imediatamente a seguir. E sempre que se chega à última, voltar novamente à primeira para que o processo se repita para todas as notícias. Criar uma espécie de ciclo até não existir notícias para se poder filtrar.

2.3.1 Declarações

Nesta primeira secção ficaram declaradas as variáveis globais e os *includes* necessários para o código C.

```
%{
#include "estrutura.h"      /* Include do ficheiro .c da estrutura. */

char* tag;                  /* Tag atual da noticia. */
char* id;                   /* Id filtrado da noticia atual.*/
char * categoria;           /* Categoria filtrada da noticia atual. */
char * titulo;              /* Titulo filtrado da noticia atual. */
char * autorData;           /* Autor/Data filtrado da noticia atual. */
char * texto;               /* Texto filtrado da noticia atual. */
char * textoTemporario;     /* Buffer com cada caracter do texto filtrado. */
int tamanhoTexto;           /* Tamanho do Buffer temporario.*/

Noticia *noticia;           /* Noticia atual. */
HashNoticias noticias;      /* Conjunto de todas as noticias lidas.*/
%}

%x INICIO TAG MINITAG ID CATEGORIA TITULO AUTORDATA TEXTO FIM
```

Reparar que estão declaradas todas as variáveis globais que estão a ser atualizadas depois em cada *Start Condition*. Estas variáveis apenas servem para se conseguir atualizar os vários parâmetros da variável **noticia**.

2.3.2 Regras/ERs

A primeira *Start Condition* é aquela que verifica o início de uma notícia. Quando a *tag* <pub>é encontrada, inicializa-se a estrutura e começa-se a *Start Condition* seguinte.

```
<INICIO>{

    "<pub>" {
        noticia = inicializaNoticia();
        BEGIN TAG;
    }
}
```

Aqui surge a *Start Condition* TAG. Como existe um conjunto de Tags, teve de existir uma recursividade em termos de condições. Sempre que se deteta o início daquilo que cerca a tag, obriga-se o FLex a iniciar uma nova *Start Condition*, de nome MINITAG. É nesta condição que a filtragem acontece.

```
<TAG>{
    "tag:{" {
        BEGIN MINITAG;
    }
    "#ID" {
        BEGIN ID;
    }
}
```

Filtra-se todo o texto (exceto o carácter de terminação) até se encontrar esse mesmo carácter. Sempre que essa filtragem acontece, significa que uma das *tags* foi lida, obrigando a inserção na **List** e nova evocação da *Start Condition* TAG. Este processo até se encontrar a String "ID"que caracteriza o início do identificador da notícia (que pertencerá a uma nova *Start Condition*).

```
<MINITAG>{

    [^\}]+/[}] {
        tag = strdup(yytext);
        insereTags(noticia,tag);

        BEGIN TAG;
    }
}
```

Com todas as tags já filtradas, é hora de percorrer a *Start Condition* ID. Sabe-se que é a junção de duas Strings e que tudo o que está para lá do hífen se encontra no formato numérico (vários números por Id, daí a expressão + que simboliza precisamente esta repetição).

```
<ID>{
```

```
    "post-"[0-9]+   {
        id = strdup(yytext);
        setId(noticia,id);

        BEGIN CATEGORIA;
    }
}
```

Após o Id do *post* existe um tanto de informação desnecessária que só termina quando é feito um novo parágrafo. Na linha que precede essa mesma mudança de linha, encontramos uma única String. Essa é a String que representa a Categoria da notícia. Como sabemos que essa informação se encontra em formato alfanumérico, tomou-se a decisão de definir inicialmente no ficheiro .pl o tipo *Portugues* que englobaria todas as letras, incluindo as exceções latinas, como acentos, tis ou cedilhas.

```
<CATEGORIA>{

    ^{Portugues}+ {
        categoria = strdup(yytext);
        setCategoria(noticia,categoria);

        BEGIN TITULO;
    }
}
```

Recorre-se ao padrão FLex, uma vez que se tem a certeza que a Categoria começa sempre no início da linha.

O procedimento é o mesmo para o Título, já que é igualmente uma String individualizada e a começar logo no início da próxima linha. Repare-se que ambos os parâmetros usam o **Portugues** por nós definidos. Como feito dito, esta definição exclui a mudança de linha, o que garante que nada do que se encontra para baixo do ficheiro é filtrado.

```
<TITULO>{

    ^{Portugues}+ {
        titulo = strdup(yytext);
        setTitulo(noticia,titulo);

        BEGIN AUTORDATA;
    }
}
```

Entre o Título e o Autor/Data existe outro tanto de informação inútil. Sabemos que este parâmetro tem uma sintaxe específica e que está representado em apenas uma linha como acontece com a Categoria e o Título. Por não se estar a utilizar a definição de **Portugues**, é necessário filtrar apenas todos os caracteres exceto a mudança de linha.

```
<AUTORDATA>{
```

```
    "#DATE: "[^\n]+   {
        autorData = strdup(yytext+15);
        setAutorData(noticia,autorData);

        BEGIN TEXTO;
        texto = (char*) malloc(100);
        tamanhoTexto = 100;
    }
}
```

Dado que a informação filtrada para o texto seria por norma maior, foi necessária a criação de um *buffer* de caracteres, de forma a guardar e re-alocar espaço de memória sempre que o FLex lê um novo caráter utilizando a ER `.|\n` que como visto em baixo, copia a informação filtrada pelo FLex para o texto temporário usando a função C `strcat`. Dessa forma, a variável `textoTemporario` funciona como um intermediário para a cópia para a String `texto` que irá ficar com o texto final de cada notícia. De seguida, foi pensado que o critério de paragem seria quando o FLex encontrasse o literal `"Etiquetas"`, que pelo nosso pensamento seria o fim do texto dado que achamos desnecessária essa informação ser passada para a normalização do artigo.

(Notar que o alocamento de memória para o buffer textttttexto é feito antes de se entrar na próxima *Start Condition*, para garantir que ao entrar no entrar no texto em si, existe memória para se ler desde logo o primeiro cáracter.)

```
<TEXTO>{

    .|\n {
        textoTemporario = realloc(texto, ++tamanhoTexto);
        strcat(textoTemporario,yytext);
        texto=textoTemporario;
    }
    "Etiquetas" {
        setTexto(post,texto);
        BEGIN FIM;
    }
}
```

Quando o texto da notícia é, por fim, filtrado, tem de existir um critério de paragem que detete este fim e guarde finalmente tudo aquilo que estava a ser definido na nossa estrutura. Este critério tem também de garantir que um novo ciclo é iniciado e que por isso a *Start Condition* inicial é começada.

```
<FIM>{

    "</pub>" {
        insereNoticia(noticias,noticia);
        BEGIN INICIO;
    }
}
```

2.3.3 Rotinas em C

Para a concretização correta do enunciado pedido, foi obviamente necessário toda uma assistência com a linguagem C de modo a suportar diversos requisitos necessários que o FLex não suporta de raiz e/ou se tornasse mais dispendioso criar a algoritmia para o mesmo.
Foi para isso criado dois tipos de ficheiros C que englobavam todas as funções relativas ao projeto, focando-se em guardar todas as informações filtradas pelo FLex ao longo que lia a notícia e assim após ter sido feito o *parse* do ficheiro `.txt` em si, existir a impressão dessas informações filtradas e aglomeradas, agora de forma única dada a ligação entre os dados e o ID de cada notícia.

Para isso houve a criação dum ficheiro `estruturas.c` que observa e implementa um número de métodos que foram necessários ao longo da filtragem FLex. Como visto em baixo temos 3 tipos de funções principais:

1. Inicializar estrutura(s) do trabalho.
2. Definir/Preencher os parâmetros de cada **Noticia** à medida que o FLex encontra os mesmos.
3. Funções finais chamadas no fim da filtração e processamento do ficheiro `.txt` sob forma de assim imprimir para ficheiros `*.html` toda a informação que tinha sido obtida e que agora estava numa estrutura em C.

```
// Funcoes que inicializam a estrutura fulcral do trabalho.
Noticia* inicializaNoticia();
void inicializaNoticias(HashNoticias noticias);
void insereNoticia(HashNoticias noticias, Noticia *noticia);

// Funcoes que definem os parametros de toda a noticia.
void insereTags(Noticia *noticia, char *tag);
void setId(Noticia *noticia, char *id);
void setCategoria(Noticia *noticia, char *categoria);
void setTitulo(Noticia *noticia, char *titulo);
void setAutorData(Noticia *noticia, char *autorData);
void setTexto(Noticia *noticia, char *texto);

// Funcoes que tratam do parse do ficheiro .txt e posterior .html
void parsingFich(Noticia *noticia, int *tamanho);
void parseTotal(HashNoticias noticias);
```

Esta parte final sobre a criação do HTML é que gera em si alguma rotina e algoritmia em C. Através da informação já guardada na estrutura relativa a todos os parâmetros necessários foi utilizado intensivamente a função pré-definida em C `fprintf` que escreve e cria os diversos ficheiros HTML sobre cada notícia com toda a sua informação associada e normalizada em modo HTML.
Para a parte das *tags* foi utilizado um pensamento similar a este sendo que sempre que se encontrava as *tags* dessa **Noticia**, criava-se os ficheiros HTML de cada uma e colocava-se automaticamente um link para o ficheiro HTML dessa notícia de forma assim a criar um índice de notícias que tinham no seu conteúdo aquela *tag*.

Capítulo 3

Considerações Finais e Trabalho Futuro

A ideia por detrás da realização de todo o projeto foi, desde o início, solucionar os requisitos de uma forma sequencial. Após se conseguir fazer o *parse* de todo o Jornal, guardando as várias informações na estrutura em si, esperava-se conseguir gerar todos os ficheiros HTML de cada **notícia**, criando depois para cada *tag*, a lista de notícias que a incluem num ficheiro HTML, acabando por fazer uma contagem da quantidade de vezes que cada *tag* estava a ser usada.

O grande desafio deste relatório foi precisamente estabelecer uma correta correspondência entre o código C e FLex, de modo a se conseguir gerar então os vários HTML. A dificuldade resultante disto foi toda a gestão de memória alocada aquando o processamento das diversas *strings* que não nos conseguiu processar todo o ficheiro *raw* fornecido pelos docentes e dessa forma a impossibilidade de filtrar e processar todas as notícias como era suposto.

As *tags* e todos os seus ficheiros HTML foram criados e por consequência, cada uma tem os diversos *links* para as notícias que continham as mesmas. Como dificuldade final foi a criação dum índice desses mesmos ficheiros HTML das tags e a contagem das vezes que cada tag única aparecia durante o processamento das notícias todas.

Como trabalho futuro, tem de existir uma melhor gestão e pensamento sobre como interligar o código C e todas as suas limitações ao nível baixo de memória que utiliza e dessa forma escrever melhor `ERs` que filtrem e processem o ficheiro de forma adequada, limpa e com a alocação da memória em mente de forma a conseguir utilizar a ferramenta poderosa FLex com rigor e usufruir dela toda a capacidade inerente que nos oferece.