Processamento de Linguagens

1º Ficha para as Aulas Prática (ERs)
Resolução de alguns exercícios em Flex

Ano Lectivo de 19/20

Objectivos

A ficha prática na base destas notas auxiliares contém exercícios para serem resolvidos nas aulas teórico-práticas com vista a sedimentar os conhecimentos relativos a:

- Motivação para o uso de Expressões Regulares (ERs) como forma de especificar padrões a pesquisar em textos —recurso a utilitários de Linux que seguem essa abordagem;
- uso de Expressões Regulares para definir (gerar) Linguagens Regulares;
- uso de Expressões Regulares para desenvolver programas eficientes, baseados em algoritmos standard guiados por Autómatos Finitos Deterministas, para reconhecer Linguagens Regulares;
- uso de Autómatos Deterministas Reactivos, para processar Linguagens Regulares, isto é para desencadear Acções específicas ao reconhecer frases que derivam de Padrões (definidos com base em ERs) —princípio da Programação baseada em regras *Condição-Reacção*, **Sistemas de Produção**;
- geração automática de programas a partir de especificações formais;
- uso das ferramentas como o grep, o gawk e o flex, disponíveis em ambiente Linux, para processamento de linguagens regulares dentro de ficheiros de texto, nomeadamente para criação de *Filtros de Texto* em geral.

O presente documento pretende completar o que foi dito nas aulas TP apresentando a resolução de um ou dois e exercícios que envolvam os vários operadores estudados, disponíveis em Flex para trabalhar com ER, e também usem criação de **estados dependentes do contexto esquerdo** (designados em Flex por *Start Conditions (SC)*).

1 Desenvolvimento de Filtros de Texto com o Flex

Para introduzir a ferramenta de geração de programas FLex baseada em especificações com Expressões Regulares, e para ilustrar a importância do uso de autómatos deterministas reactivos como suporte à construção de programas eficientes, propõem-se alguns exercícios, para resolver dentro ou fora da aula, que visam a criação de programas autónomos para *filtrar textos* (FT).

1.1 Processador de Questionários

Suponha que ao fim de cada entrevista um Repórter produz um texto com as perguntas e respostas, distinguindo umas das outras porque as perguntas começam com 'EU:', no início da linha, e as respostas começam com 'ELE:', também no início da linha.
Nesse contexto, pretende-se desenvolver um FT para processar os questionários que:

a) simplesmente retire do texto original as tais marcas '`EU:`' e '`ELE:`', devolvendo todo o resto da entrevista sem qualquer alteração.
Melhore o filtro, de modo a tratar as marcas, quer estejam escritas em maiúsculas, quer em minúsculas;

b) substituia a marca 'EU' pela palavra 'Entrevistador' e a marca 'ELE' por 'Entrevistado';

c) substituia a marca 'EU' pelo nome do entrevistador (Er) e a marca 'ELE' pelo nome do entrevistado (Eo), supondo que no próprio texto encontrará as respectivas definições (ordem irrelevante) na forma '`EU=nomeEr.`' ou '`ELE=nomeEo.`'

Antes de resolver e usando a aproximação TDD[1] proposta em todas as aulas para desenvolver filtros ou processadores de linguagens, vai-se preparar 1 texto de entrada sofisticado para melhor se perceber o problema e seus requisitos e depois para testar a especificação Flex à medida que se escreve a mesma.

```
%---------------------------------------------------------------
EU= Andreia Vasques.
ele  = Pedro Rangel Henriques .

EU: como se chama?
ELE: eu chamo-me Pedro.
Eu: que graca eu tambem sou Pedro e ele é Xico.
ELE: eu ja me chamei Ricardo mas quando o conheci mudei para pedro.
Eu: e ele ali ao fundo é Xico.
ELE: muito prazer sr. Xico.
     entao vamos a isso?
eu: sim, vamos!
    onde nasceu o Pedro?
ele: eu nasci no Porto.
eU: que curioso eu também, ELe (o Xico) e que nasceu em Braga
eLe: ah, boa terra, terra de padres e igrejas...

%---------------------------------------------------------------
Ele=Ana.

EU: porque se interessa por essa area?
ele: nao sei! desde pequeno eu sempre tive um grande fascinio pela natureza.
Eu: ha quantos anos trabalha nisto?
ELE: ha cerca de 20 anos e ele (o Ze) sempre ao meu lado.
eU: ah, nao sabia que o Ze era o seu parceiro!...
EU: e entao quantos anos demoraram para acabar o livro?
ELE: sim, o Ze esteve a apoiar a minha obra estes 7 anos.
eu: muitos parabens pela obra e felicidades para a venda.
ele: obrigado! Deus a ouça!

%...............................................................
EU = Roberto da Silva.

EU: E então como chegou a esse estado?
eLe: Sinceramente não sei, aconteceu tudo tão rápido.
Eu: Mas disse-me aos berros e exasperado eLe: Coisas do além!!!.
```

[1] Do inglês *Test Driven Development*.

```
ElE: Coitado de si!
ElE: Desculpe mas agora vou mesmo ter de me ir embora. Até mais ver...
```

Resolução das alíneas a) e c) usando duas SC para captar os nomes dos entrevistador (`DEFeu`) e do entrevistado (`DEFele`):

```
%{
#include <string.h>
#include <stdlib.h>

char *eu;
char *ele;
int i;
%}
%option noyywrap

/* declara 2 Start-Condictions para captar os dois nomes */
%x DEFeu DEFele

%%
^(?i:eu)[ ]*\=[ ]*      { BEGIN DEFeu; }
                        /* quando encontra o contexto esquerdo apropriado, ativa a SC */
                        /* e muda para um estado especifico */
<DEFeu>\..*\n           { BEGIN INITIAL; }
                        /* quando encontra o fim deste estado, volta ao estado Incial */
<DEFeu>[^.]+            { eu = strdup(yytext); }
                        /* dentro do estado especifico capta todos os carateres e guarda-os */

^(?i:ele)[ ]*\=[ ]*     { BEGIN DEFele; }
<DEFele>\..*\n          { BEGIN INITIAL; }
<DEFele>[^.]+           { ele = strdup(yytext); }

^(?i:eu)[ ]*\:     { printf("%s:", eu); }
^(?i:ele)[ ]*\:         { printf("%s:", ele); }
%%

int main()
{
    eu = ele = "";
     yylex();
return(0);
}
```

O filtro, gerado pelo Flex a partir desta especificação, substitui cada ocorrência do texto 'EU:' no inicio de uma linha precisamente pelo texto (nome completo) que apareça depois de 'EU=' (também no início de uma linha e com zero ou mais espaços antes e depois do '='). Considerando o texto de entrada acima o resultado produzido seria

```
%-------------------------------------------------------------

Andreia Vasques: como se chama?
Pedro Rangel Henriques : eu chamo-me Pedro.
Andreia Vasques: que graca eu tambem sou Pedro e ele é Xico.
Pedro Rangel Henriques : eu ja me chamei Ricardo mas quando o conheci mudei para pedro.
```

```
Andreia Vasques: e ele ali ao fundo é Xico.
Pedro Rangel Henriques : muito prazer sr. Xico.
     entao vamos a isso?
Andreia Vasques: sim, vamos!
    onde nasceu o Pedro?
Pedro Rangel Henriques : eu nasci no Porto.
Andreia Vasques: que curioso eu também, ELe (o Xico) e que nasceu em Braga
Pedro Rangel Henriques : ah, boa terra, terra de padres e igrejas...

%---------------------------------------------------------------

Andreia Vasques: porque se interessa por essa area?
Ana: nao sei! desde pequeno eu sempre tive um grande fascinio pela natureza.
Andreia Vasques: ha quantos anos trabalha nisto?
Ana: ha cerca de 20 anos e ele (o Ze) sempre ao meu lado.
Andreia Vasques: ah, nao sabia que o Ze era o seu parceiro!...
Andreia Vasques: e entao quantos anos demoraram para acabar o livro?
Ana: sim, o Ze esteve a apoiar a minha obra estes 7 anos.
Andreia Vasques: muitos parabens pela obra e felicidades para a venda.
Ana: obrigado! Deus a ouça!

%...............................................................

Roberto da Silva: E então como chegou a esse estado?
Ana: Sinceramente não sei, aconteceu tudo tão rápido.
Roberto da Silva: Mas disse-me aos berros e exasperado eLe: Coisas do além!!!.
Ana: Coitado de si!
Ana: Desculpe mas agora vou mesmo ter de me ir embora. Até mais ver...
```

1.2 Documento anotado em XML

Como sabe um Documento XML é um texto vulgar semeado de *anotações*, ou *marcas*, que são identificadores especiais (designados por *elementos*) intercalados entre os caracteres '<' e '>'. Os referidos *elementos* são palavras, ou seja, sequências de 1 ou mais letras maiúsculas ou minúsculas.

Num documento XML *bem formado*, a cada *marca de abertura* corresponderá uma *marca de fecho*, que tem o mesmo identificador, mas que começa por '</' terminando na mesma em '>'.

O texto (qualquer sequência de carateres, eventualmente vazia) entre as duas marcas, de abertura e de fecho, diz-se o *texto anotado*; a anotação que se lhe aplica é definida, ou descrita, pelo nome escolhido para o *elemento*.

Além disso, dentro de cada *marca de abertura*, além do *elemento*, que identifica a marca, ainda podem aparecer *atributos* que são triplos formados por um outro identificador (nome do atributo, formado por letras maiúsculas ou minúsculas), pelo sinal "=" e pelo respectivo valor que é qualquer texto entre aspas.

Para clarificar a explicação anterior, mostra-se abaixo um exemplo de um pequeno documento XML.

```
<REGISTO   NUM="BRG-2020.10" DATA = "20200312">
Este Registo foi lavrado em Braga no mês de março a pedido do Pai.
  <Nome>
    <NProp>Rui</NProp><Apel>Costa e Sousa</Apel>
  </Nome>
Identificada a criança, assenta-se o local e hora do nascimento.
  <Nascim>
```

```
    O bebé nasceu de parto natural em
      <NLoc>Hospital de St. Maria, Porto</NLoc> às <NHora>12:14</NHora> horas.
    </Nascim>
  Nada mais havendo a registar, encerra-se o registo,
  <ASSINA Funcao="Escrivao"> <nome>Bento de Lancastre</nome></ASSINA>
  </REGISTO>
```

Começar por escrever e analisar um texto de entrada sofisticado, como o que se mostra acima, ajuda a melhor perceber o problema e seus requisitos e é muito útil para testar a especificação Flex à medida que se escreve a mesma, seguindo-se assim a aproximação TDD[2] proposta em todas as aulas para desenvolver filtros ou processadores de linguagens.
Desenvolva um ou mais filtros de texto (FT) que receba um documento XML e:

a) devolva o texto original, após ter retirado todas as marcas.

b) verifique se o texto *está balanceado*. Para isso conte o número de *marcas de abertura* e o número de *marcas de fecho*, indicando *erro* sempre que se verifique estão desequilibradas (mais marcas de abertura que marcas de fecho, ou vice-versa).

c) verifique a concordância entre as *marcas de abertura* e as *marcas de fecho*, isto é, garanta que as marcas se fecham por ordem inversa que se abrem tomando em atenção o *elemento* de cada marca. Note que o validador pedido só deve produzir uma mensagem para o 1º erro detetado; caso o texto de entrada esteja bem formado, deve terminar silenciosamente[3].

d) escreva, por cada marca de abertura, o respetivo texto anotado.

e) (esta alínea fica para os alunos resolveram sozinhos usando as Start-Conditions da alínea anterior) escreva, por cada marca de abertura, todos os *atributos* (par '*nome-valor*') que essa marca tenha, separando o *nome do atributo* do respetivo *valor* depois de retiradas as aspas.

Começando por tratar das 2 primeiras alíneas, é mesmo muito fácil e simples escrever uma especificação Flex que detete e retire todas as marcas XML do texto de entrada, contando as marcas à medida que são retiradas e deixando passar para a saída todos os restantes carateres fora das Marcas.
Observe-se então a especificação seguinte

```
%{
#include <stdio.h>
#include <string.h>

int abre=0, fecho=0;
%}
%option noyywrap

%%
\<\/[^>]*\>    { fecho++; }
               /* retira a Marca de Fecho e conta os fechos */
\<[^>]+\>      { abre++; }
               /* retira a Marca de Abertura e conta as aberturas */
               /* todos os restantes carateres fora das marcas são copiadas para a saída */
%%

int main() {
```

[2] Do inglês *Test Driven Development*.
[3] À boa moda das ferramentas Linux.

```
    yylex();
         if (abre != fecho) { printf("ERRO! texto desbalanceado\n"); }
    return 0;
}
```

O filtro, gerado pelo Flex a partir desta especificação, produz, conforme pedido, a saída que se mostra abaixo (note-se que no fim não surge nenhuma mensagem de erro porque o número de marcas de abertura e de fecho está balanceado):

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    Este Registo foi lavrado em Braga no mês de março a pedido do Pai.

        RuiCosta e Sousa

    Identificada a criança, assenta-se o local e hora do nascimento.

      O bebé nasceu de parto natural em
        Hospital de St. Maria, Porto às 12:14 horas.

    Nada mais havendo a registar, encerra-se o registo,
     Bento de Lancastre
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Para resolver a alínea c) deve construir-se um novo filtro muito parecido com o anterior, mas agora será necessário: (1) usar uma stack para empilhar todos os elementos das marcas de abertura e depois (quando aparecer uma marca de fecho) comparar o elemento no topo da stack com o elemento dessa marca de fecho assinalando erro se não forem iguais[4]; e (2) deitar fora, eliminar da saída, todos os restantes carateres.
Abaixo lista-se a nova especificação para gerar o verificador pedido:

```
%{
#include <stdio.h>
#include <string.h>
#define  max 20

char * stack[max];
int sp=0;
char *fecha,*abre;
%}
%option noyywrap

%%
\<\/[a-zA-Z]+    { fecha = strdup(yytext+2); abre = stack[--sp];
                   if (strcmp(fecha,abre))
                      {printf("ERRO! esperava fechar %s e fechou %s\n",abre,fecha);exit(1);} }
                 /* retira da stack a última Marca Aberta e verifica se concorda com a Marca de F
\<[a-zA-Z]+      { stack[sp++] = strdup(yytext+1); }
                 /* empilha a Marca de Abertura */
.|\n             { ; } /* todos os restantes carateres dentro ou fora das marcas são eliminados *
%%
```

[4]Note que o processamento deve ser interrompido mal se detete o primeiro erro, para evitar mensagens de erro em cascata.

```
int main() {
    yylex();
    return 0;
}
```

Por fim, para resolver a alínea d) é preciso usar uma *Start-Condition* para se conseguir identificar e guardar o texto que surja após uma marca de abertura e até se encontrar uma marca de fecho. Tal mudança de contexto, ou de estado de reconhecimento, é forçosa porque o texto anotado entre marcas é exatamente igual ao texto não anotado, daí que surja a necessidade de se ter ERs que só serão ativadas quando se reconhecer que se está num novo estado.
Dessa forma (recorrendo a SC) a solução fica elegante e clara, como se lista abaixo.

```
%{
#include <stdio.h>
#include <string.h>
#define  max 20
%}
%option noyywrap
%option stack

%x TXT
%%
\<[a-zA-Z]+[^>]*\>          { yy_push_state(TXT);}
                        /* quando encontra uma Marca de Abertura ativa um novo Estado */
<TXT>\<\/[a-zA-Z]+\>        { yy_pop_state(); }
                        /* ao encontrar uma Marca de Fecho retorna ao Estado anterior */
<TXT>\<[a-zA-Z]+[^>]*\>     { yy_push_state(TXT);}
<TXT>[^<]*                  { printf("Texto anotado: [%s]\n",yytext); }
.|\n                    /* todos os restantes carateres dentro ou fora das marcas são eliminados */
%%

int main() {
    yylex();
    return 0;
}
```

Note que neste exemplo como podem aparecer marcas dentro de marcas, o processo de tratamento do texto anotado é recursivo (conforme se vai ver, tal situação ocorre em imensas situações reais). Assim sendo, em vez de usar a instrução `BEGIN` para comutar entre dois estados (*Start-Conditions*) vai ser necessário recorrer a uma **stack de estados**[5] para retornar ao estado anterior pela ordem inversa com que os estados se abrem até regressar ao estado inicial.
Desta forma, será necessário usar as instruções especiais `yy_push_state(novoEstado)` e `yy_pop_state()`.

O filtro, gerado pelo Flex a partir desta última especificação, produz, conforme pretendido, a saída que se mostra abaixo:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Texto anotado: [
    Este Registo foi lavrado em Braga no mês de março a pedido do Pai.
      ]
Texto anotado: [
        ]
Texto anotado: [Rui]
```

[5] Ver a sua declaração no preâmbulo da especificação.

```
Texto anotado: [Costa e Sousa]
Texto anotado: [
      ]
Texto anotado: [
    Identificada a criança, assenta-se o local e hora do nascimento.
      ]
Texto anotado: [
      O bebé nasceu de parto natural em
        ]
Texto anotado: [Hospital de St. Maria, Porto]
Texto anotado: [ às ]
Texto anotado: [12:14]
Texto anotado: [ horas.
      ]
Texto anotado: [
    Nada mais havendo a registar, encerra-se o registo,
    ]
Texto anotado: [ ]
Texto anotado: [Bento de Lancastre]
Texto anotado: [
    ]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Note-se que apesar de aparentar um formato de saída estranho (possivelmente errado) está correto e resulta dos `NewLine` que surgem na entrada, com frequência, imediatamente após a marca de abertura ou mesmo dentro do próprio texto anotado; isto também explica os textos vazios ('`[ ]`'). Resolvida a alínea d) e observado o texto de saída é normal que se queira melhorar a solução[6] porque apesar de se identificarem as diferentes partes do texto de entrada marcadas, não se sabe qual a respetiva anotação.
Ou seja, em vez do texto de saída acima, seria interessante e desejável ter algo como se ilustra a seguir.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Texto anotado (REGISTO NUM="BRG202010" DATA="20200312")[
    Este Registo foi lavrado em Braga no mês de março a pedido do Pai.

Texto anotado (Nome)[

Texto anotado (NProp)[Rui](NProp)
Texto anotado (Apel)[Costa e Sousa](Apel)
      ](Nome)
    Identificada a criança, assenta-se o local e hora do nascimento.

Texto anotado (Nascim)[
      O bebé nasceu de parto natural em

Texto anotado (NLoc)[Hospital de St. Maria, Porto](NLoc) às
Texto anotado (NHora)[12:14](NHora) horas.
      ](Nascim)
    Nada mais havendo a registar, encerra-se o registo,

Texto anotado (ASSINA Funcao="Escrivao")[
```

[6] Recorde que o ciclo 'resolver-testar-otimizar' é uma prática corrente e muito desejável em Programação (dir-se-ia mesmo, em Engenharia em geral).

```
Texto anotado (nome)[Bento de Lancastre](nome)](ASSINA)
    ](REGISTO)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

No qual se lè na mesma os vários fragmentos de texto anotados, mas à volta de cada fragmento aparece a respetiva marca que o anota.
Atente-se no exemplo abaixo[7] onde se mostra outro tipo frequente de texto de entrada anotado e logo a seguir a saída desejada, para que melhor se perceba a otimização pretendida.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Aqui temos um exemplo de um texto corrido que só será anotado pontualmente
e não na forma do registo anterior ou de muitos textos que estão todos dentro de uma mesma marca
Neste o autor <NE type="PER">Fulano de Tal</NE> só vai anotar os termos
que sejam Nomes de Entidades como pessoas <NE type="PER">Cicrano Beltrano</NE>
ou organizações como a <NE type="ORG">Universidade do Minho</NE> ou
então nomes de locais como <NE type="LOC">Braga</NE> ou
<NE type="LOC">Viana do Catelo</NE>.
E pronto, fim da historia escrita em <NE type="LOC">Esposende</NE>.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Texto anotado (NE type="PER")[Fulano de Tal](NE)
Texto anotado (NE type="PER")[Cicrano Beltrano](NE)
Texto anotado (NE type="ORG")[Universidade do Minho](NE)
Texto anotado (NE type="LOC")[Braga](NE)
Texto anotado (NE type="LOC")[Viana do Catelo](NE)
Texto anotado (NE type="LOC")[Esposende](NE)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Para produzir uma nova saída como explicado e ilustrado acima, basta fazer um pequeno ajuste na especificação Flex, lista atrás, do filtro pretendido.
Mostra-se abaixo a nova especificação que gera o filtro para produzir a otimização requerida, especificação essa que apenas acrescenta à inicial operadores e técnicas Flex já conhecidas de exercícios anteriores.

```
%{
#include <stdio.h>
#include <string.h>
#define  max 20
%}
%option noyywrap
%option stack

%x TXT
%%
\<[a-zA-Z]+[^>]*\>    { yytext[yyleng-1]='\0'; printf("\nTexto anotado (%s)[",yytext+1);
                        yy_push_state(TXT);}
                      /* quando encontra uma Marca de Abertura ativa um novo Estado */
<TXT>\<\/[a-zA-Z]+\>  { yytext[yyleng-1]='\0'; printf("](%s)",yytext+2);
                        yy_pop_state(); }
                      /* ao encontrar uma Marca de Fecho retorna ao Estado anterior */
<TXT>\<[a-zA-Z]+[^>]*\>    { yytext[yyleng-1]='\0'; printf("\nTexto anotado (%s)[",yytext+1);
                             yy_push_state(TXT);}
<TXT>[^<]*            { printf("%s",yytext); }
<*>.|\n               /* todos os restantes carateres dentro ou fora das marcas são eliminados */
```

[7] Lembre-se da importância do recurso a TDD na área de Processamento de Linguagens.

```
%%

int main() {
    yylex();
    return 0;
}
```

Nesta especificação Flex importa apenas chamar a atenção para o uso do *especificador de contexto*, ou *de estado*, '`<*>`' que significa que a ER seguinte deve ser reconhecida em qualquer estado, independentemente da SC ativa.