Processamento de Linguagens
MiEI (3ºano)

Trabalho Prático nº 1 – parte B (FLex)

Ano lectivo 16/17

1 Objectivos e Organização

Este trabalho prático tem como principais **objectivos**:

- aumentar a experiência de uso do ambiente Linux e de algumas ferramentas de apoio à programação;
- aumentar a capacidade de escrever *Expressões Regulares (ER)* para descrição de *padrões de frases*;
- desenvolver, a partir de ERs, sistemática e automaticamente *Processadores de Linguagens Regulares*, que filtrem ou transformem textos com base no conceito de regras de produção ¡Condição-Ação¿;
- utilizar o para gerar *filtros de texto em C* FLex.

Aprecia-se a imaginação/criatividade dos grupos ao incluir outros processamentos além dos mínimos pedidos.
A entrega deve ser feita através de um link a disponibilizar pela equipa docente via Bb **até 3ªfeira dia 18 de Abril**. Não faça um ficheiro comprido zip nem envie vários ficheiros; deve submeter apenas 1 ficheiro, o PDF do relatório, com toda a informação incluída conforme se explica abaixo.
O programa desenvolvido será apresentado aos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo, em data a marcar.
O **relatório** a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da solução e sua implementação (incluir a especificação FLex), deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LaTeX.

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.

2.1 Processador de Inglês corrente

Neste projeto pretende-se que leiam um texto corrente em inglês que faça uso das habituais contrações do tipo *I'm, I'll, W're, can't, it's*[1] e:

a) reproduza o texto de entrada na saída expandindo todas as contrações que encontrar.

b) gere em HTML uma lista ordenada de todos os verbos (no infinitivo não flexionado) que encontrar no texto, recordando que essa forma verbal em inglês é precedida pela palavra *to*, sendo a mesma palavra retirada se o dito verbo for antecedido por *can, could, shall, should, will, would, may, might*. Considere também a forma interrogativa em que o verbo no infinitivo é precedido por *do/does* ou *did*.

Nota: O ficheiro anexo 'FAQ-Python-EN.txt' pode ser usado para experiência ou testes, mas não deve limitar-se a esse.

2.2 Normalizador de Autores em BibTex

Em BibTeX a lista de autores ou editores de uma publicação pode ser escrita de várias formas como se exemplifica a seguir

```
author    = {Ricardo Martini and Cristiana Araújo and Pedro Rangel Henriques},
editor    = {Giovani Librelotto},
author = {Javier Azcurra and Mario Ber{\'o}n and Pedro Henriques and Maria Jo{\~a}o Varanda Pereira},
author    = {Oliveira, Nuno and Henriques, Pedro Rangel and da Cruz, Daniela and Pereira, Maria João Varanda},
author = {Vilas Boas, Ismael and Oliveira, Nuno and Rangel Henriques, Pedro},
author="Martini, Ricardo G. and Ara{\'u}jo, Cristiana and Almeida, Jos{\'e} Jo{\~a}o and Henriques, Pedro",
editor="Rocha, {\'A}lvaro and Correia, Maria Ana  and Adeli, Hojjat and Mendon{\c{c}}a Teixeira, Marcelo",
author = {Berón, M. and Montejano, G. and Riesco, D. and Henriques, P.R. and Debnath, N.},
```

Neste projeto pretende-se que comece por converter todos os carateres com acentos explícitos em caracteres portugueses e depois reescreva todos os campos `author` e `editor`, mantendo os outros inalterados, seguindo sempre a mesma norma em que serão sempre usadas as chavetas (troque os casos em que aparecem aspas) e aparece sempre o Apelido seguido por uma ","e as Letras Iniciais dos restantes nomes seguidas por um "."
Assim no exemplo anterior teríamos:

```
author = {Martini, R. and Araújo, C. and Henriques, P. R.},
editor = {Librelotto, G.},
author = {Azcurra, J. and Berón, M. and Henriques, P. R. and Pereira, M. J. V.},
author = {Oliveira, N. and Henriques, P. R. and da Cruz, D. and Pereira, M. J. V.},
author = {Vilas Boas, I. and Oliveira, N. and Rangel Henriques, P.},
author = {Martini, R. G. and Araújo, C. and Almeida, J. J. and Henriques, P.},
editor = {Rocha, A. and Correia, M. A.  and Adeli, H. and Mendonça Teixeira, M.},
author = {Berón, M. and Montejano, G. and Riesco, D. and Henriques, P. R. and Debnath, N.},
```

Use a linguagem `Dotty` e o processador `dot` para criar um grafo com todos os autores que colaboram com um dado autor.
Nota: O ficheiro anexo 'exemplos.bib' pode ser usado para experiência ou testes, mas não deve limitar-se a esse.

2.3 Processador de DTDs

Um DTD[2] é um ficheiro em notação própria que define cada dialeto XML[3] concreto, isto é, define as etiquetas (ou *tags*) de todos os *elementos* que podem ser usados para marcar os textos desse dialeto e define também a estrutura da anotação (ou seja, a forma como os elementos se podem combinar).
O extrato que se segue é um exemplo de DTD, neste caso o que é usado para identificar as fotografias dos entrevistados no âmbito do Museu da Pessoa:

[1] Recorra à sua gramática de Inglês para relembrar os outros casos.
[2] Document Type Definition
[3] eXtensible Markup Language

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- edited with XML Spy v4.1 U (http://www.xmlspy.com) by siglab (University of Minho) -->
<!ELEMENT fotos (foto+)>
<!ELEMENT foto (quem, onde?, quando?, facto?, legenda?,autor?)>
   <!ATTLIST foto ficheiro CDATA #REQUIRED>
<!ELEMENT quem (#PCDATA)>
<!ELEMENT quando (#PCDATA)>
   <!ATTLIST quando data CDATA #IMPLIED>
.......
```

Pretende-se que escreva um processador para ler um DTD e gerar a lista (por ordem alfabética) de todas as etiquetas que se podem usar[4] e para cada uma os seus atributos, caso existam[5], e as etiquetas das quais depende, as quais aparecem do lado direito dentro de parêntesis, ou seja, que podem ser usadas dentro desse elemento[6].
Para clarificar, recorde o ficheiro 'legenda.xml' escrito de acordo com o DTD acima e que foi utilizado no TP anterior, do qual se lista um extrato abaixo.

```
<fotos>
<foto ficheiro="022-F-01.jpg">
<quando data="1961-01-15"/>
<quem>Ana de Lourdes de Oliveira Chaminé e António Oliveira Machado</quem>
<facto> Os noivos cortam o bolo de casamento</facto>
</foto>
</fotos>
```

Use a linguagem `Dotty` e o processador `dot` para criar um grafo com todos as etiquetas de que depende uma dada etiqueta.
Nota: O ficheiro anexo 'legenda.dtd' pode ser usado para experiência ou testes, mas não deve limitar-se a esse.

2.4 Processador de Named Entities

O texto mostrado abaixo é um exemplo (por razões óbvias apenas se nostra um fragmento) de documento anotado num dialeto XML chamado Enamex, o qual permite identificar com as etiquetas `ENAMEX` e `TIMEX` as *Entidades* referidas num texto através de nomes próprios (do tipo: *Pessoa*, *País*, *Cidade*) ou *Datas*.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<ENAMEX> Vivia </ENAMEX> este herói no <ENAMEX TYPE="CITY"> Rio de
Janeiro </ENAMEX>, <ENAMEX TYPE="COUNTRY">Brasil</ENAMEX> e era
professor não se sabe de que doutrinas no ano de
<TIMEX TYPE="DATE"> 1710 </TIMEX>, quando os franceses comandados por
<ENAMEX TYPE="PERSON"> Duclerc </ENAMEX>, atacaram a cidade.
O governador portou-se mal, e permanecia numa deplorável inacção, quando
<ENAMEX TYPE="PERSON"> Bento do Amaral </ENAMEX> à frente dos seus
estudantes e de paisanos armados, saiu a tomar o passo aos invasores,
repelindo-os energicamente, e dando lugar a que o ataque se malograsse
e os agressores ficassem prisioneiros.
<ENAMEX> Não </ENAMEX> tardou <ENAMEX TYPE="PERSON"> Dugua-Trouin </ENAMEX>
a vir tomar a desforra.
</document>
```

Escreva um processador que analise um ficheiro *Enamex* como o referido acima e produza um Índice em HTML com as Pessoas, Países e Cidades referidas.
Para situar cronologicamente a narrativa, indique o ano menor citado e o ano mais atual.
Use o `Google Maps` para localizar as cidades referidas.
Nota: O ficheiro anexo 'exemploEnamex.xml' pode ser usado para experiência ou testes, mas não deve limitar-se a esse.

[4] No exemplo incompleto serão 'fotos', 'foto', 'quem', 'quando'.
[5] Note que no exemplo mostrado, 'foto' tem 1 atributo de nome 'ficheiro' e 'quando' tem também um atributo de nome 'data'.
[6] No exemplo, 'fotos' depende de uma etiqueta 'foto' que pode ocorrer 1 ou mais vezes; 'foto' depende de 6 outras etiquetas, 5 delas opcionais; mas já 'quem' e 'quando' não dependem de nenhuma pois só podem ter texto dentro do elemento.

2.5 Preprocessador para notação musical Abc

A notação musical textual Abc permite definir músicas e gerar partituras (abcm2ps, abc2svg) ou gerar MIDI (abc2midi). (Relembre essa notação com alguns exemplos de letras de músicas usados nas aulas).

Pretendemos equipar Abc com uma funcionalidade semelhante aos "defines"do C. Sugere-se a seguinte sintaxe:

- Para definir uma abreviatura a1 usar §`a1={valor 1}`
- Seguidamente, no interior de uma frase, §`a1` será substituída por `valor 1`.
- Se numa frase surgir `{valor 1}`→§`a1`, será substituída por `valor 1` e §`a1` fica com `valor 1`
- §`3{cd e2|}` repete 3 vezes o seu argumento (= `cd e2|cd e2|cd e2|`)
- Adicionalmente crie um conjunto de abreviaturas predefinidas e que sejam expandidas usando a mesma notação, e que facilitem a escrita (sugestão abreviaturas para drum-patterns ou para guitar-chords).

Nota: O ficheiro anexo 'exemploAbc.abc' pode ser usado para experiência ou testes; sugere-se ainda a consulta da documentação Abc em `https://sourceforge.net/projects/abcplus/files/Abcplus/abcplus_en-2017-02-22.zip`

Para mais esclarecimentos contactar `jj@di.uminho.pt`.

2.6 Inline Templates em linguagem C

Um template contêm tipicamente texto com variáveis ou expressões.

Neste enunciado propomos uma síntaxe para templates embebidos em C (tempc) e pede-se um programa flex que, dado um ficheiro tempc, gere C válido correspondente.

Exemplo de ficheiro tempc:

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

Fli={{<li> [% ele %] < /li>
}}

Fhtml={{<html>
 <head><title>[% tit %]</title></head>
<body>
 <h1>[% tit %]</h1>
 <ul>[% MAP Fli comp items %]</ul>
</body>
</html>
}}

int main(){
  char * a[]={"expressões regulares","parsers","compiladores"};
  printf("%s\n",Fhtml("Conteudo programático", 3, a));
}
```

Note que cada template:

1. está contido entre `{{...}}`, e vai dar origem a uma função.
2. pode conter variáveis (seus parâmetros implícitos) representadas na forma `[% id %]`; essas variáveis dão origem a parâmetros da função do tipo string.
3. pode conter MAPs que aplicam uma função 'f' a uma lista 'l' de comprimento 'c' denotando-se por `[% MAP f c l %]` em que 'f' é o nome de uma função (ou de outro template); 'c' e 'l' são parâmteros do template que contém o MAP.

Se for possível, arrange modo de incluir nos templates expressões C (que retornem um valor do tipo string) e que sejam executadas na hora do processamento de modo a usar o respetivo valor na expansão do dito template.
Exemplo de código C que poderia ser gerado a partir do exemplo anterior:

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

char* Fli(char* ele){
  char BUF[10000]; int j=0;
  j+=sprintf(BUF+j," <li>%s</li>\n",ele);
  strdup(BUF);
}

char* Fhtml(char * tit, int comp, char* items[]){
  char BUF[10000]; int j=0;
  j+=sprintf(BUF+j,"<html>\n");
  j+=sprintf(BUF+j," <head><title>%s</title></head>\n",tit);
  j+=sprintf(BUF+j,"<body>\n");
  j+=sprintf(BUF+j," <h1>%s</h1>",tit);
  j+=sprintf(BUF+j," <ul>");
  for(int i=0 ; i < comp; i++){ j+=sprintf(BUF+j,"%s", Fli(items[i])); };
  j+=sprintf(BUF+j,"</ul>");
  j+=sprintf(BUF+j,"</body>");
  j+=sprintf(BUF+j,"</html>");
  strdup(BUF);
}

int main(){
  char * a[]={"expressões regulares","parsers","compiladores"};
  printf("%s\n",Fhtml("Conteudo programático", 3, a));
}
```