Processamento de Linguagens e Compiladores
LCC (3ºano) + MiEFis (4ºano)

1º Ficha para as Aulas Prática

Ano Lectivo de 19/20

Objectivos

Este ficha prática 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.

1 Pesquisa de Padrões em Texto usando o comando GREP

Na shell do seu sistema operativo Linux (abrindo um terminal de trabalho), invoque o comando grep como se mostra a seguir (note que PADRAO representa a expressão regular, ER, que vai variando conforme se indica abaixo); o ficheiro ao qual se aplica o comando grep, '`index.html`', é um ficheiro de texto puro anotado segundo a sintaxe da linguagem de markup HTML:

```
grep -P PADRAO index.html
```

Para ver o que faz este comando e perceber o significado e efeito das expressões regulares de procura, faça PADRAO tomar os valores definidos pelas ER seguintes:

```
"pedro"
"[Pp]edro"
"^[Pp]edro"
"[Pp]edro manuel"
```

```
"[Pp]edro manuel?"
"[Pp]edro (manuel)?"
"[Pp]edro (R|r)[a-z]+"
"[Pp]edro | (R|r)[a-zA-Z]+"
"[Pp]edro[ ]+(?i:rangel)"
"[Pp]edro H"
"[Pp]edro.+H"
"[Pp]edro[^H]+H"
```

1.1 Definição de ERs para pesquisar com o comando GREP

Depois de perceber o objetivo e funcionamento do Grep e o papel das ERs na definição dos padrões de pesquisa, tente por si escrever as ER convenientes para procurar no ficheiro de teste '`index.html`':

a) todas as linhas com a marca concreta '`HTML`' e só essas linhas; incluia também a marca de fecho '`/HTML`'; note que a marca '`HTML`', '`Html`' ou '`html`' devem ser ambas encontradas.

b) todas as linhas com qualquer marca do dialeto '`HTML`' (palavra entre os sinais '$<$' e '$>$'); altere a sua procura de modo a encontrar também as marcas que contêm atributos além do nome do elemento. Note que deve ser mantida a preocupação introduzida na linha anterior: as marcas HTML tanto podem ser escritas em maiúsculas ou minúsculas!

c) todas as linhas que correspondem a cabeçalhos de nível 1 (marcada com a tag '$< H1 >$'); todas as linhas que correspondem a cabeçalhos de qualquer nível (marcada com a tag '$< Hn >$' em que n é um natural).

d) todas as linhas que comecem por uma âncora (identificadas em `HTML` pela marca '$< A$').

e) todas as linhas que contenham números; todas as linhas terminadas por números; imponha que os números tenham 2 ou mais dígitos.

f) todos os comentários do ficheiro dado (em `HTML` os comentários são qualquer texto entre '$<!--$' e '$-->$').

1.2 Pesquisa de um Padrão em vários Ficheiros com o comando GREP

Após ter percebido como se usa o Grep para procurar um determinado padrão de carateres em um ficheiro de texto, experimente agora o uso desta ferramenta para procurar padrões em todos os ficheiros de Assembly da máquina de stack virtual VM, ficheiros de texto com a extensão '`*.vm`'

a) todas as linhas onde apareçam instruções de carregar valores para a stack (mnemónicas começadas por '`PUSH`'.

b) todas as linhas que contenham *comentários* em Assembly os quais são qualquer texto desde '`//`' até ao fim da linha .

c) todas as *labels* que em Assembly são formadas por uma palavra (*case-insensitive*) no início de uma linha seguida por ':'. Note que as palavras podem conter algarismos pelo meio.

2 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).

2.1 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 e a marca 'ELE' pelo nome do entrevistado, supondo que no início encontrará as respectivas definições (ordem irrelevante) na forma 'EU=nome.' ou 'ELE=nome.'

2.2 Expansor de Abreviaturas

Quando se retiram apontamentos, ou de uma forma geral, se tem de escrever muito depressa, é hábito usar abreviaturas que correspondam a uma ou mais palavras vulgares e longas.
Suponha que criou esse costume e resolveu inserir nos seus textos as ditas abreviaturas (2 ou mais letras) precedidas pelo carácter '\'. Por exemplo: '\qq' (qualquer), ou '\mb' (muito bom), ou ainda '\sse' (se e só se).
Desenvolva, então:

a) um FT que lhe devolva o texto original mas com todas as abreviaturas (que definiu à partida) devidamente expandidas;

b) melhore o seu filtro de modo a contemplar ainda o tratamento do carácter '/' no fim de uma palavra, representando o sufixo 'mente', e o carácter '~' no início de uma palavra, representando o prefixo 'in'. Uma palavra pode conter ambos os caracteres, um no início e outro no fim (pense na abreviatura da palavra 'infelizmente');

c) outra melhoria que poderia introduzir no seu filtro era contemplar a possibilidade de definir abreviaturas dentro do próprio texto, na forma '\def:abrev=termo-expandido;'. Pense como o fazer e nas implicações que tal requisito teria no seu filtro original.[1]

2.3 Somador de Números

Construa um Filtro de Texto que adicione todos os números dum texto e que, no final, imprima a sua soma (no ficheiro de saída não deve aparecer nenhum caracter do texto de entrada).
Evolua o seu processador no sentido de:

a) Escrever a soma sempre que encontre o carácter '='.

b) Só comece a somar quando detectar o carácter '+' e deixe de somar quando este carácter voltar a aparecer.

[1] Alínea proposta para pensar fora da aula.

2.4 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 XML*) intercalados entre os caracteres '<' e '>'. 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 '>'.
Desenvolva um filtro de texto (FT) que receba um documento XML e:

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

b) conte o número de *marcas de abertura* e o número de *marcas de fecho*, indicando *erro* sempre que se verifique um desequilíbrio entre ambas[2];

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[3]. No fim produza uma listagem, ordenada alfabeticamente, de todos os elementos distintos encontrados.

2.5 Normalizador de Emails

Os Emails escritos à moda PRH caracterizam-se por terem todas as palavras começadas por letras minúsculas, à exceção dos nomes próprios e siglas.
Desenvolva, então:

a) um FT que normalize o texto, *capitalizando* (escrevendo a letra inicial em maiúsculas) todas as palavras no início de cada frase. Além da primeira palavra do texto, uma frase começa depois de um '.', '?' ou '!', seguido de zero ou mais espaços, eventualmente um ou mais fim-de-linha;

b) complete a especificação anterior de modo a que o seu *normalizador de emails prh* conte também todos os nomes próprios (palavras começadas por maiúsculas) e siglas (palavras formadas só por maiúsculas, uma ou mais) encontradas no texto original.

[2] Aqui apenas se pede que detecte o erro por contagem e não atendendo ao *identificador do elemento* em causa em cada marca.

[3] Mas agora tomando em atenção o *identificador do elemento* em causa em cada marca.

3 Desenvolvimento de Filtros de Texto com o Gawk

Para introduzir o Sistema de Produção[4] AWK (uma *linguagem de programação* para deteção de padrões e processamento de texto, mais o respetivo *interpretador*) —criado em 1977 pelos cientistas Alfred Aho, Peter J. Weinberger e Brian Kernighan no Bell Labs— execute os comandos abaixo e analise com cuidado o seu resultado:

```
gawk '{ print $1 }' utilizadores.txt
gawk '{ print $1 }' pltesteER.txt
gawk '{ print $2 }' pltesteER.txt
gawk -F: '{ print $1 }' utilizadores.txt
gawk -F: '{ print $3 }' utilizadores.txt
gawk -F: '$1=="prh" { print $1 "->" $6 }' utilizadores.txt
gawk -F: '$1=="prh"||$1=="jcr" { print }' utilizadores.txt
```

e depois disso execute repetidamente o comando:

```
gawk -f FILE.gawk utilizadores.txt
```

para os ficheiros (`FILE`) '`expusers1`' e '`expusers2`', observando com cuidado o resultado produzido em cada caso, sendo:

```
expusers1.gawk
    BEGIN             { FS=":"; conta=0 }
    NR==1             { print "A processar o ficheiro: " FILENAME }
    NR>=1 && NR<=10   { print $1 }
                      { conta+=NF }
    END               { print conta " - " NR  }

expusers2.gawk
    BEGIN              { FS=":"; conta=0 }
    /rita/             { conta++; print $1 " -> " $6 }
    /prh|jcr/          { conta++; print $1 " -> " $6 }
    /uucp/,/rpm/       { conta++; print }
    /x.*sbin/ && $3>40 { conta++; print "Em sbin: " $0 }
    $1 ~ "nuno"        { conta++; print $1 " =» " $NF }
    END                { print conta " - " NR " = " conta/NR }
```

3.1 Um Filtro de Texto inicial com o Gawk

Com base nos exemplos anteriores, escreva um ficheiro '`exppltesteER.gawk`' para processar o texto '`pltesteER.txt`' conforme solicitado nas alíneas seguintes:

a) contar todas as linhas começadas pela palavra '`linha`'.

b) contar todas as linhas que contenham marcas HTML (só de abertura e depois de abertura e fecho), imprimindo a respetiva linha.

c) detetar todas as linhas que tenham ancoras HTML (ou sejam marcas '<`A`' com atributo '`HREF`') e imprimir o respetivo URL.

d) detetar todas as linhas que tenham ancoras HTML e imprimir o *texto ancorado*.

[4]Sistema baseado num conjunto de regras 'Condição-Ação'.

3.2 Processar os Inscritos numa atividade desportiva com o Gawk

Faça um exercício semelhante ao anterior, construindo agora um ficheiro '`expinscr.gawk`' para processar o texto '`inscritos.txt`' conforme solicitado nas alíneas seguintes:

a) imprimir o nome e o email dos concorrentes inscritos entre a 5º e a 15º posições.

b) imprimir o nome dos concorrentes que se inscrevem como 'Individuais' e são de 'Valongo'.

c) imprimir o telemóvel e a prova em que está inscrito cada concorrente cujo nome seja 'Paulo' ou 'Ricardo', desde que seja da Vodafone.

d) imprimir os 20 primeiros registos com os nomes convertidos para minúsculas.

3.3 Processar as Pessoas listadas nos Róis de Confessados com o Gawk

Construa agora um ficheiro '`expprocessos.gawk`' para processar o texto '`processos.txt`' com o intuito de calcular frequências de alguns elementos (a ideia é utilizar arrays associativos para o efeito) conforme solicitado a seguir

a) Calcula a frequência de processos por ano (primeiro elemento da data);

b) Calcula a frequência de nomes (considera um nome uma palavra e propaga o cálculo por todos os campos que contenham nomes);

c) Calcula a frequência dos vários tipos de relação: irmão, sobrinho, etc.

3.4 Processar um Arquivo Musical com o Gawk

De modo a reforçar toda a matéria vista até aqui, Construa agora um ficheiro '`exparqson.gawk`' para processar o texto '`arq-son.txt`' de modo a:

a) Quais os títulos das canções alentejanas?

b) Quantas músicas estão catalogadas por distrito/área geográfica?

c) Quais os títulos das músicas com áudio disponível em MP3?

d) Quais os títulos das músicas e quantas são que têm a palavra 'Jesus' no título?