Processamento de Linguagens – LEI

Exame de Recurso II

13 de Julho de 2015 (9h00)

Dispõe de **2:00 horas** para realizar este teste.

Questão 1: Expressões Regulares e Autómatos (4v = 1+1+1+1)

Responda às seguintes questões sobre ERs e autómatos:

a) Constrúa um autómato finito determinista (AFD) para a seguinte expressão regular:

```
[0-9]+(\.[0-9]+)?(E[0-9]+)?
```

b) Considere a seguinte expressão regular:

```
[a-zA-Z]*[^,]=
```

Diga, justificando, qual ou quais das seguintes frases fazem match com a expressão regular apresentada:

i) Butt=

ii) BotHEr,=

iii) Ample

iv) FIdDlE7h=

v) Brittle =

vi) Other.=

c) Especifique expressões regulares para definir o número de identificação em cada um dos dois tipos de cartão de crédito que se descrevem a seguir:

Mastercard: começam com 51, 52, 53, 54 ou 55 e têm no total 16 dígitos;

American Express: começam por 34 ou 37 e têm 15 dígitos.

d) Considere os seguintes exemplos de links HTML:

```
<a>This is a link</a>
<a href='http://regexone.com'>Link</a>
```

Escreva uma expressão regular para cada um dos tipos de link apresentados.

Questão 2: Filtros de Texto em Flex e GAWK (4v = 1+1+1+1)

Especifique filtros de texto com base em expressões regulares e regras de produção (padrão - ação) para resolver as seguintes alíneas:

a) Especifique em Flex um filtro que dado um texto de entrada contendo uma lista de palavras (sequências de letras podendo conter dígitos) seguidas por uma sigla em maiúsculas e entre parêntesis curvos, separadas por virgulas, produz um texto de saída em que cada palavra aparece em 1 linha começada pela sigla seguida por ":"e depois por uma lista com todos os caráteres da palavra dentro de parêntesis retos na forma minúscula e maiúscula. Todos os outros caráteres devem ser eliminados.
Exemplo: dado o seguinte texto de entrada

```
abril (ABR) , Janeiro (JAN), junHo (JUN) .
```

O resultado gerado na saída seria:

```
ABR: [aA][bB][rR][iI][lL]
JAN: [jJ][aA][nN][eE][iI][rR][oO]
JUN: [jJ][uU][nN][hH][oO]
```

b) Explique cuidadosamente o que faz a especificação Flex abaixo:

```
%{
#include <strings.h>
int c=1, ls=0, rs=0;
%}

%x CIT
%%
\\chapter\{[^}]+       { yytext=toupper(yytext+9); printf("%d. %s",c++,yytext); }
\\part\{[^}]+\}        { ; }
\\label\{              { ls++; ECHO; }
\\ref\{                { rs++; ECHO; }
\\cite\{               { printf("["); BEGIN CIT; }
<CIT>[a-zA-Z0-9]+      { printf("%s",geraChaves(yytext)); }
<CIT>\}                { printf("]"); BEGIN INITIAL; }
%%
int yywrap()
{ return(1); }

int main()
{ yylex();
   if (ls != rs) { printf("Aviso: ...\n"); }
  return 0; }
```

e, supondo que a função `'geraChaves()'` retorna sempre a string "KK", concretize a sua explicação, indicando qual o texto que seria produzido à saída ao ler o seguinte texto de entrada:

```
\part{Abertura} \label{cI}
\chapter{Inicio}
Aqui esta um exemplo interessante!
\Section{sec.1} \label{sum}
 \enum
   + primeiro caso: 2+3-1
 \ee
\part{Fecho}
\chapter{Conclusao} \label{sdois}
conclui-se que um  \Chapter{Exemplo} faz falta, bem como uma nova \section{Final}.
Este capítulo \ref{cI} fala de ... e por isso \cite{arl1} ou \cite{PRHas}.
Para terminar em beleza \cite{aa/1eum}.
```

c) Considere o seguinte excerto de um ficheiro de texto correspondente a uma listagem dos serviços académicos desta universidade:

```
62845 Abel Jonas Ferreira Faria
68700 Alexandra Cristina Tâmega Magalhães Meireles
48392 Ana Cristina Nunes Aires
63379 António Joaquim da Costa Martins Fernandes
65439 Bruno Filipe Fernandes Rebelo
68707 Bruno Moreira Guedes
58908 Carlos Filipe Borges Barreto
62123 Carlos da Cunha Martins
62122 David Carvalho Marques
...
```

Escreva scripts em GAWK para responder às seguintes questões:

i) Qual a distribuição de alunos por *classe* de ano de inscrição? (considere que o ano de inscrição corresponde à classe milhares do número de aluno, por exemplo a Alexandra e o Bruno Guedes deveriam ser contabilizados na classe "68");

ii) Qual a distribuição de nomes nesta listagem? (por exemplo: Quantos Brunos? Quantos Farias? Quantos Guedes? etc.)

Questão 3: Desenho/especificação de uma Linguagem (5v = 3+1+1)

Pretende-se construir um parser que reconheça listas heterogéneas. Uma lista heterogénea é uma lista em que os elementos são números inteiros, booleanos, palavras ou sublistas do mesmo tipo. Eis alguns exemplos:

- `[]`
- `[1,2,3,4]`
- `[1, ameixa, 2, banana, 3, uva, 4, laranja, TRUE]`
- `[3, [p1, animal, 10], [p2, FALSE, desporto, 5], [p3, FALSE, legumes, 10], perguntas]`

Baseando-se nesta descrição responda às seguintes alíneas:

a) Escreva uma gramática independente de contexto (GIC) para a linguagem apresentada;

b) Especifique o analisador léxico usando a notação do *flex*;

c) Apresente a árvore de derivação para o último exemplo, construída segundo a gramática que especificou.

Questão 4: Gramáticas e Parsing (5v = 1+2+1+1)

Considere a gramática independente de contexto, GIC, abaixo apresentada, atendendo a que os símbolos terminais T e não-terminais NT são definidos antes do conjunto de produções P, sendo A o seu axioma ou símbolo inicial.

```
T  = { abre, fecha, info }
NT = { A }
P = {
      p1: A --> abre info A A fecha
      p2: A --> abre fecha
    }
```

Baseando-se nesta descrição responda às seguintes alíneas:

a) Desenhe o autómato LR(0) completo para a gramática especificada. É visível algum conflito? Se sim, quais?

b) Escreva uma gramática tradutora (GT) em notação do *yacc* e tendo apenas as acções semânticas necessárias para contabilizar o número de elementos de informação presentes em cada frase (info); acrescente acções semânticas à gramática anterior para contar o número de blocos 'A' reconhecidos.

c) A Gramática apresentada é LL(1)? Justifique a sua resposta e em caso negativo transforme-a numa que o seja.

d) Escreva um Parser Recursivo Descendente (RD) para a gramática transformada na alínea anterior.

Questão 5: Compilação (2v)

Supondo que no seu programa em LPIS surge a seguinte instrução condicional

```
if (a[b] > 3) then { d= 2* c; }
```

e assumindo que os endereços de `a, b, c, d` são respetivamente `0, 10, 11, 12`
diga **justificando (passo a passo)** qual dos fragmentos de código Assembly da VM (abaixo) seria gerado

```
   (a)          (b)          (c)
se: PUSHG 0   se: PUSHI 0   PUSHGP
 PUSHI 10      PUSHG 10     PUSHI 0
 PADD          PADD         PADD
 PUSHI 3       LOADN        PUSHG 10
 SUP           PUSHI 3      LOADN
 JZ fse        SUP          PUSHI 3
 NOP           JZ  se       SUP
 PUSHI 2       PUSHI 12     JZ  fse
 PUSHG 11      PUSHI 2      PUSHI 2
 MUL           PUSHG 11     PUSHG 11
 STOREG 12     MUL          MUL
 JUMP se       STOREN       STOREG 12
fse: NOP      fse: NOP     fse: NOP
```