Processamento de Linguagens I
LESI + LMCC (3º ano)

1ª Ficha Prática

Ano Lectivo de 05/06

1 Objectivos

Este ficha prática contém exercícios para serem resolvidos nas aulas teórico-práticas com vista a sedimentar os conhecimentos relativos a:

- 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 espec'ificas ao reconhecer frases que derivam de Padr oes (definidos com base em ERs) —princípio da Programação baseada em regras *Condição-Reação*;
- geração automática de programas a partir de especificaç oes formais;
- uso da ferramenta Flex, disponível em ambiente Linux, para geraç ao automática de processadores de linguagens regulares, nomeadamente para criaç ao de *Filtros de Texto* ou de *Analisadores Léxicos*.

2 Filtros de Texto

Para introduzir a ferramenta de geração de programas FLex baseada em especificacoes com Expressoes Regulares, e para sedimentar os conhecimentos que adquiriu sobre automatos deterministas reactivos como suporte a construcao 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.

Solução:

"ex2_1a1.l" 2a ≡

```
EU      EU:
ELE     ELE:

%%

^{EU}         { }
^{ELE}        { }

%%

int yywrap(void) { return 1;}

int main( int argc, char **argv ) {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

   yylex();
}

◇
```

Melhore o filtro, de modo a tratar as marcas, quer estejam escritas em maiúsculas, quer em minúsculas;

Solução

Aqui a única alteração que se pretende é ao nível das expressões regulares definidas. Ou seja,

"ex2_1a2.l" 2b ≡

```
EU      [Ee][Uu]:
ELE     [Ee][Ll][Ee]:

%%

^{EU}         { }
^{ELE}        { }

%%

int yywrap(void) { return 1;}

int main( int argc, char **argv ) {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

   yylex();
}
◇
```

Assim, com o seguinte teste:

"ex2_1.txt" 3a ≡

```
eU: Porque se interessa tanto por esta area?

ElE: Nao sei, desde pequeno tive um grande fascinio pela natureza.

Eu: Ha quantos anos trabalha nisto?

ELe: Cerca de 20 anos.
```
◇

O resultado de execução será:

```
Porque se interessa tanto por esta area?

Nao sei, desde pequeno tive um grande fascinio pela natureza.

Ha quantos anos trabalha nisto?

Cerca de 20 anos.
```

b) substitua a marca "EU:"pela palavra "Entrevistador"e a marca "ELE"por "Entrevistado";

"ex2_1b.l" 3b ≡

```
EU      [Ee][Uu]
ELE     [Ee][Ll][Ee]

%%

^{EU}           { printf("Entrevistador"); }
^{ELE}          { printf("Entrevistado"); }

%%

int yywrap(void) { return 1;}

int main( int argc, char **argv ) {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

   yylex();
}
```
◇

O resultado de execução obtido (com o exemplo anterior):

```
Entrevistador: Porque se interessa tanto por esta area?

Entrevistado: Nao sei, desde pequeno tive um grande fascinio pela natureza.

Entrevistado: Ha quantos anos trabalha nisto?

Entrevistado: Cerca de 20 anos.
```

c) substitua 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."[1]

Solução

"ex2_1c.l" 4 ≡

```
%{
    #include <strings.h>
    char *nomeEu;
    char *nomeEle;

%}

EU      [Ee][Uu]
ELE     [Ee][Ll][Ee]

%%

{EU}"="[^.]+"."        { yytext[yyleng-1]='\0';
                         nomeEu = (char*) strdup(&yytext[3]);   }
{ELE}"="[^.]+"."       { yytext[yyleng-1]='\0';
                         nomeEle = (char*) strdup(&yytext[4]);  }

^{EU}/":"              { printf("%s", nomeEu);        }
^{ELE}/":"             { printf("%s", nomeEle);       }


%%

int yywrap(void) { return 1; }

int main(int argc, char **argv ) {

               ++argv, --argc;  /* skip over program name */

               if ( argc > 0 )
                       yyin = fopen( argv[0], "r" );
               else
                       yyin = stdin;

               yylex();
               }
```

◇

Assim com o exemplo anterior, acrescentando apenas as duas definições para os nomes no ínicio do texto:

[1] Alínea proposta para pensar fora da aula.

"ex2_1c.txt" 5 ≡

```
eu=Andreia.
eLe=Ana.

eU: Porque se interessa tanto por esta area?

ElE: Nao sei, desde pequeno tive um grande fascinio pela natureza.

Eu: Ha quantos anos trabalha nisto?

ELe: Cerca de 20 anos.
```
◇

O resultado obtido é:

```
Andreia: Porque se interessa tanto por esta area?
Ana: Nao sei, desde pequeno tive um grande fascinio pela natureza.
Andreia: Ha quantos anos trabalha nisto?
Ana: Cerca de 20 anos.
```

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 implicararia no seu filtro original.[2]

[2] Alínea proposta para pensar fora da aula.

a) e b) Solução

"ex2_2.l" 6a ≡

```
qq          \\qq
mtbom       \\mb
sse         \\sse
ft          \\FT
mente       \/
in        ~
letra       [a-zA-Z]

%%

{qq}            { printf("qualquer"); }
{mtbom}         { printf("muito bom"); }
{sse}           { printf("se e so se"); }
{ft}            { printf("filtro de texto"); }
{in}/{letra}+   { printf("in");        }
{letra}+{mente} { yytext[yyleng-1]='\0'; printf("%smente",yytext);     }

%%

int yywrap(void) { return 1;}

int main( int argc, char **argv ) {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

   yylex();
}
```

◇

Consideremos o seguinte exemplo:

"ex2_2.txt" 6b ≡

```
Suponha que criou esse costume e resolveu inserir nos seus textos as
ditas abreviaturas precedidas pelo caracter "\" - por exemplo:
1. \qq
2. \mb
3. \sse

Assim \qq abreviatura devera ser expandida e com ctz sera \mb nao
termos de nos preocupar com as abreviaturas mas \sse tivermos este
\FT nos nossos arquivos...

~feliz/ se nao o tivermos nao poderemos fazer abreviaturas nos
nossos documentos... Obvia/ teremos de saber utilizar lex.

```

◇

O resultado obtido é:

```
Suponha que criou esse costume e resolveu inserir nos seus textos as
ditas abreviaturas precedidas pelo caracter "\" - por exemplo:
1. qualquer
2. muito bom
3. se e so se

Assim qualquer abreviatura devera ser expandida e com ctz sera muito
bom nao termos de nos preocupar com as abreviaturas mas se e so se
tivermos este filtro de texto nos nossos arquivos...

infelizmente se nao o tivermos nao poderemos fazer abreviaturas nos
nossos documentos... Obviamente teremos de saber utilizar lex.
```

2.3 Normalizador de Emails

Os Emails escritos a moda PRH caracterizam-se por terem todas as palavras comecadas por letras minÃºsculas, à excepção dos nomes próprios e siglas.
Desenvolva, então:

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

b) complete a especificacao anterior de modo a que o seu *normalizador de emails prh* conte tambem todos os nomes próprios (palavras começadas por maiscúla) e siglas (palavras formadas so por maiscúlas, uma ou mais) encontradas no texto original.

Solução

a) e b)

"ex2_3.l" 7 ≡

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

int nProprios = 0, nSiglas = 0;
%}

ident        [a-z]+
nproprio     [A-Z][a-z]+
sigla        [A-Z]+
pontuacao    [.!?]

%x IFrase

◇
```

File defined by 7, 8a.

"ex2_3.l" 8a ≡

```
%%

{nproprio}        { nProprios++; ECHO;       }
{sigla}           { nSiglas++; ECHO;         }
{pontuacao}" "*\n* { ECHO; BEGIN IFrase;        }

<IFrase>{nproprio} { nProprios++; ECHO; BEGIN INITIAL;  }
<IFrase>{sigla}    { nSiglas++; ECHO; BEGIN INITIAL;    }
<IFrase>{ident}    { yytext[0] = toupper(yytext[0]);
                     printf("%s", yytext);
                     BEGIN INITIAL;          }

%%

int yywrap(void) { return 1;}

int main(int argc, char **argv ) {

                ++argv, --argc;  /* skip over program name */

                if ( argc > 0 )
                        yyin = fopen( argv[0], "r" );
                else
                        yyin = stdin;

           // enter in state that match with IFRASE
           BEGIN IFrase;

           printf("\n\t\t\t ------ Resultado do Filtro de Texto: ------\n\n");
                yylex();
           printf("\n\t\t\t ------ Fim de Filtro de Texto. ------ \n\n");

           printf("Numero de siglas: %d\n", nSiglas);
                printf("Numero de nomes proprios: %d\n\n",nProprios);

                return 0;
                }
```

◇

File defined by 7, 8a.

Consideremos o seguinte exemplo:

"ex2_3.txt" 8b ≡

```
qualquer delas me agrada mas tendo dar uma resp, acho que o esquema
que prefiro e PAG 5. mas talvez com a caixa de letras do PAG 6 e a
cor azul final (no PAG 5 esta muito escuro, quase parece preto).

dp podemos falar com Joao para acertar a versao final, mas iria tv
mais para os tons de azul.
```

◇

O resultado obtido é:

```
                        ------ Resultado do Filtro de Texto: ------

    Qualquer delas me agrada mas tendo dar uma resp, acho que o esquema que prefiro e PAG 5.
    Mas talvez com a caixa de letras do PAG 6 e a cor azul final (no PAG 5 esta muito
    escuro, quase parece preto).

    Dp podemos falar com Joao para acertar a versao final, mas iria tv mais para os
    tons de azul.

                        ------ Fim de Filtro de Texto. ------

    Numero de siglas: 3
    Numero de nomes proprios: 1
```

3 Analisadores Léxicos

No contexto do desenvolvimento de Compiladores, ou mais genericamente de Processadores de Linguagens, o primeiro nível, ou tarefa a implementar, é a **análise léxica** que tem por missão ler o texto fonte (que se pretende *transformar*, caso seja uma *frase válida* da linguagem em causa) e converter todas as palavras correctas em símbolos terminais dessa linguagem.
Com esse fim em vista, propõe-se para esta aula o recurso à ferramenta Flex para gerar um **Analisador Léxico (AL)** a partir da descrição dos símbolos terminais de uma linguagem e sua associação aos respectivos códigos internos.

3.1 Máquina de Venda de Chocolates

Considere uma situação em que se pretende simular o funcionamento de uma máquina de vender chocolates. Dado o stock no início do dia (nome, preço e quantidade de cada produto disponível), a quantia inicial de trocos e os registos das vendas diárias (nome do chocolate escolhido e a quantia introduzida), o objectivo é calcular *a evolução do stock ao longo do dia e o dinheiro acumulado.* A animação pretendida deverá mostrar através de desenhos o estado da máquina (stock existente e dinheiro ganho) após cada movimento.
No contexto desta aula o que se pretende é: que defina uma linguagem para descrever o estado inicial da máquina e os registos de vendas efectuadas durante o dia; e que desenvolva um AL para reconhecer todos os símbolos terminais dessa linguagem e devolver os respectivos códigos.
Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.

Consideremos a seguinte gramática para a linguagem pretendida:

```
G1 = (T,N,S,P)

T = { '{', '}', ';', ',' '(', ')', '-', string, int, real,
      STOCK, TROCOS, VENDAS }

N = { MVC, Stock, Descricao, Trocos, Moeda, Vendas, Nome, Preco,
      Quantidade, IdMoeda, QuantiaI }

S = MVC
```

```
P = {

MVC        ---> '(' STOCK '{' Stock '}' ';'
                    TROCOS '{' Trocos '}' ';'
                    VENDAS '{' Vendas '}'
                ')'
Stock      ---> Descricao
                Stock ',' Descricao
Descricao  ---> Nome '-' Preco '-' Quantidade
Trocos     ---> Moeda
              | Trocos ',' Moeda
Moeda      ---> IdMoeda '-' Quantidade
Vendas     ---> Nome '-' QuantiaI
Nome       ---> string
Preco      ---> real
Quantidade ---> int
IdMoeda    ---> real
QuantiaI   ---> real

 }
```

Consideremos agora um exemplo específico para esta linguagem, obedecendo às regras estabelecidas por esta gramática.

"ex3_1.txt" 10 ≡

```
(
  /- registo de stock
  STOCK  { twix - 2.0 - 5, mars -3.5 - 7, dove -1.5- 10, kitkat -3.25- 9 } ;

  /- registo de trocos -/
  TROCOS { 0.1 - 13, 0.2 - 8, 0.5 - 15, 1. - 20, 2. - 10 };

  /- registo de vendas
  -/
  VENDAS { dove - 1.0, mars - 0.7, kitkat - 0.8, kitkat - 0.9,
    mars - 0.5, dove - 1., twix - 7. }
)
```

◇

Solução

"ex3_1.l" 11 ≡

```
%{
#define STOCK 1
#define TROCOS 2
#define VENDAS 3
#define STRING 4
#define INT 5
#define REAL 6

%}

stock         [Ss][Tt][oO][cC][kK]
trocos        [tT][rR][oO][cC][oO][sS]
vendas        [vV][eE][nN][dD][aA][sS]
string        [a-zA-Z]+
int           [1-9][0-9]*
real          [0-9]+"."[0-9]*
separadores   [{}(),;-]
comentarios "/-"[^-]*"-/"|"/-"[^\n]*

%%

{stock}                 { return(STOCK);         }
{trocos}                { return(TROCOS);        }
{vendas}                { return(VENDAS);        }
{int}                   { return(INT);           }
{real}                  { return(REAL);          }
{string}                { return(STRING);        }
{separadores}           { return(yytext[0]);     }
{comentarios}|[ \t]     { ; }
.                       { printf("Unrecognized symbol"); }

%%

int yywrap(void) { return 1;}

int main()
          {
           int s;
           while (s=yylex()) { printf("%d ",s);  }
           return 0;
          }


◇
```

Resultado obtido (utilizando o exemplo acima):

```
40
1 123 4 45 6 45 5 44 4 45 6 45 5 44 4 45 6 45 5 44 4 45 6 45 5 125 59
2 123 6 45 5 44 6 45 5 44 6 45 5 44 6 45 5 44 6 45 5 125 59
3 123 4 45 6 44 4 45 6 44 4 45 6 44 4 45 6 44
4 45 6 44 4 45 6 44 4 45 6 125
41
```

3.2 Anuário dos Medicamentos brancos

Considere agora uma outra situação em que, para auxiliar o Instituto Farmacêutico do Ministério da Saúde na gestão do novo lote de medicamentos brancos, se pretende criar um sistema de consulta a esses medicamentos acessível a qualquer farmácia via um browser HTML. Esse sistema deve mostrar a informação agrupada por: classe de medicamentos no Symposium Terapêutico (uma página por classe, com os medicamentos ordenados alfabeticamente); ou por fabricante (uma página única, com os medicamentos agrupados por fabricante).
Sobre cada medicamento é fornecida a seguinte documentação: nome, código, classe, composição química, preço recomendado, fabricantes disponíveis e lista de medicamentos de marca equivalentes (respectivo nome e fabricante).
No contexto desta aula o que se pretende é: que defina uma linguagem para descrever a informação envolvida no lote de medicamentos a considerar (essa linguagem terá que permitir definir inicialmente o ano a que o Symposium Terapêutico diz respeito e a lista das classes de medicamentos); e que desenvolva um AL para reconhecer todos os símbolos terminais dessa linguagem e devolver os respectivos códigos.
Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.

Consideremos a seguinte gramática:

```
G2 = (T,N,S,P)

T = {SYMPOSIUM, ano, string, int, real, '[',']','{','}',',','-'}

N = {SymposiumT, Classes, Medicamentos, Medicamento, Fabs, Fab,
     MedEq, MedEq1, Ano, Nome, NomeC, Comp, NFab, Pr, Cod}

S = SymposiumT

P = {

SymposiumT      ---> SYMPOSIUM ':' Ano
                     '[' Classes ']'
                     '[' Medicamentos ']'
Classes         ---> Nome
                   | Classes ',' Nome
Medicamentos    ---> Medicamento
                   | Medicamentos ',' Medicamento
Medicamento     ---> '(' Nome ',' Cod ',' NomeC ',' Comp ',' Pr ','
                         '{' Fabs '}' ',' '{'  MedEq '}'
                     ')'
Fabs            ---> NFab
                   | Fabs ',' NFab
MedEq           ---> MedEq1
                   | MedEq ',' MedEq1
MedEq1          ---> Nome '-' NFab
Ano             ---> ano
Nome            ---> string
NomeC           ---> string
Comp            ---> string
NFab            ---> string
Pr              ---> real
Cod             ---> int
}
```

Um exemplo para esta gramática poderia ser:

"ex3_2.txt" 13 ≡

```
Symposium: 2006
                [Analgesico,Antibiotico]
                [
                  /- Medicamento 1 -/
                 (Moment,1,Analgesico,Paracetemol,4.5,{Roche},{Qualquer-Bial});

                  /- Medicamento 2 -/
                 (Antigripine,2,Antibiotico,Acetalilico,6.,{Fabt,Faba},{Aga-Fabc,Agb-Fabh})
                ]
```

◇

Soluçã

"ex3_2.l" 14 ≡

```
%{
#define SYMP 1
#define STRING 2
#define ANO 3
#define INT 4
#define REAL 5
%}

symposium    [Ss][Yy][mM][pP][oO][sS][iI][uU][Mm]
string       [A-Z][a-z]+
ano          [0-2][0-9][0-9][0-9]
codigo       [1-9][0-9]*
preco        [0-9]+"."[0-9]*
sinais       [\[\](),;{}:-]
comentarios  "/-"[^-]*"-/"|"/-"[^\n]*

%%

{symposium}            { return(SYMP);                 }
{string}               { return(STRING);               }
{ano}                  { return(ANO);                  }
{codigo}               { return(INT);                  }
{preco}                { return(REAL);                 }
{sinais}               { return(yytext[0]);            }
{comentarios}|[ \t]    { ;                             }
.                      { printf("Unrecognized symbol"); }

%%

int yywrap(void) { return 1;}

int main() {
            int s;
            while (s=yylex()) { printf("%d ", s); }
            return 0;
          }
```

◇

O resultado seria (com o exemplo acima):

```
1 58 3
91 2 44 2 93
91
40 2 44 4 44 2 44 2 44 5 44 123 2 125 44 123 2 45 2 125 41 59
40 2 44 4 44 2 44 2 44 5 44 123 2 44 2 125 44 123 2 45 2 44 2 45 2 125 41
93
```

3.3 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 ">".
Dentro de cada *marca de abertura*, além do identificador do elemento, ainda podem aparecer triplos formados por um outro identificador (de atributo), pelo sinal "="e pelo respectivo valor que é qualquer texto entre aspas.
Cada fragmento do documento (texto livre) entre marcas deve ser considerado em bloco como sendo o símbolo PCDATA.
Desenvolva então um AL que receba um documento XML e devolva todos os símbolos terminais encontrados, a seguir resumidos: "<", ">", "</", "=", identificador (qualquer palavra formada por letras), valor, PCDATA.

Como complemento a este exercício[3], 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[4];

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

Na sequência do exercício anterior consideremos o seguinte exemplo em XML:

"ex3_3.txt" 15 ≡

```
<Medicamentos>
        <Codigo Numero="1">some text</Codigo>
        <Classe Nome="Analgesico"></Classe>
        <Medicaequiv>
            <Med Nome="GenericoA" Fabricante="Bial"></Med>
        </Medicaequiv>
</Medicamentos>
```

◇

Para resolver a primeira parte do exercício (construção do AL) consideremos então a seguinte gramática que descreve um documento XML como o exemplificado acima:

```
G3 = (T,N,S,P)

T = {id, val, pcdata, '<','/','>','='}

N = {DocXML, MarcAbre, MarcFech, Conteud, Compon, Elem, Atribs, Atrib,
     IdEle, IdAtr}

S = DocXML

P = {

DocXML      ---> MarcAbre Conteud MarcFech
MarcAbre    ---> '<' Elem '>'
MarcFech    ---> '<''/' IdEle '>'
```

[3] Alíneas propostas para pensar fora da aula.
[4] Aqui apenas se pede que detecte o erro por contagem e não atendendo ao *identificador do elemento* em causa em cada marca.
[5] Mas agora tomando em atenção o *identificador do elemento* em causa em cada marca.

```
Elem        ---> IdEle Atribs
Atribs      --->
               | Atribs Atrib
Atrib       ---> IdAtr '=' val
Conteud     ---> Compon
               | Conteud Compon
Compon      ---> MarcAbre
               | MarcFech
               | pcdata
IdEle       ---> id
IdAtr       ---> id

}
```

Solução (Parte 1)

"ex3_3.l" 16 ≡

```
%{
#define BEA 1
#define BEF 2
#define BD  3
#define IG  4
#define ID  5
#define VAL 6
#define PCDATA 7
%}

id           [A-Z][a-z]+
string       \"[^"]*\"

%x FimMarca
%%

{id}              { return(ID); }
{string}          { return(VAL); }
\=                { return(IG); }
\/                   { return(BEF);  }
\<                   { return(BEA); }
\>/[ \t\n]*\<           { return(BD); }
\>                   { BEGIN FimMarca; return(BD); }
<FimMarca>\<         { unput(yytext[0]);
           BEGIN INITIAL; return(PCDATA);  }
<FimMarca>.       { ;  }

%%
◇
```

File defined by 16, 17a.

"ex3_3.l" 17a ≡

```
int yywrap(void) { return 1;}

int main() {
            int s;
            while (s=yylex()) { printf("%d ",s); }
            return 0;
          }

◇
```

File defined by 16, 17a.

Considerando o exemplo anterior, o resultado obtido é:

```
1 5 3
        1 5  5 4 6 3 7 1 2 5 3
        1 5  5 4 6 3 1 2 5 3
        1 5 3
            1 5  5 4 6  5 4 6 3 1 2 5 3
        1 2 5 3
1 2 5 3
```

Solução para as alíneas a, b e c

Solução a)

"ex3_3a.l" 17b ≡

```
%%

"<""/"?[^>]*">"  { ; }
[ \t]     { ; }

%%

int yywrap(void) { return 1;}

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

◇
```

Solução b)

"ex3_3b.l" 18 ≡

```
%{
int contaA=0, contaF=0;
%}

%%

"<""/"      { contaF++; ECHO; }
"<"         { contaA++; ECHO; }
%%

int yywrap(void) { return 1;}

int main() {
            int s;
            yylex();
            if (contaA != contaF) {
      printf("\n\nERRO: marcas mal emparelhadas!!! \n ");
      printf("N. Marcas Abertas: %d\n", contaA);
      printf("N. Marcas Fechadas: %d\n", contaF);  }
            return 0;
          }

◇
```

Solução c)

ATENCAO: nesta resolucao simplificada é assumido que as marca de abertura so têm identificador de elemento (nao há atributos).

"ex3_3c.l" 19 ≡

```
%{
%{
char * abre;
char * list[50];

void initList() {
        int i = 0;

   for (i = 0; i < 50; i++)
           list[i] = NULL;
}

void addElem(char *pal) {
   int i=0;

   while(list[i]!=NULL && i < 50) {
      i++; }

   list[i] = pal;
}

char* deleteElem() {
   int i=0;
   char *aux;

      while(list[i]!=NULL && i < 50) {
   i++; }

      aux = list[i-1];
      list[i-1]=NULL;

      return aux;
}


%}

id           [A-Z][a-z]+

%%

"<"{id}">"       { ECHO;
                yytext[yyleng-1]='\0'; addElem(&yytext[1]);
      }
"<""/"{id}">"    { ECHO;
              yytext[yyleng-1]='\0';
              abre = deleteElem();
            if (strcmp(abre,&yytext[2])!=0) {
                   printf("Documento mal formado! \n");
           exit(0); }
      }

%%


int yywrap(void) { return 1;}

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

◇

Por exemplo, consideremos o seguinte documento XML mal-formado:

"ex3_3c.txt" 20 ≡

```
<Medicamentos>
    <Medicamento>
        <Codigo>some text</Codigo>
        <Classe></Classe>
        <Composicao></Composicao>
        <Preco></Preco>
        <Fabricantes>
            <Fabricante>some data</Fabricante>
        </Fabricantes>
        <MedicaEquiv>
            <Med></Med>
        </MedicaEquiv>
    </Medicamento>
</Medicamentos>
◇
```

O resultado obtido será:

```
<Medicamentos>
    <Medicamento>
        <Nome>Moment</Nome>
        <Codigo>1</Codigo>
        <Classe>Analgesico</Classe>
        <Composicao>Paracetamol</Composicao>
        <Preco>4.5</Preco>
        <Fabricantes>
            <Fabricante>pcdata</Fabricante>
        </Fabricantes>
        <MedicaEquiv>
            <Med>GenericoA</Med>
            <Med>GenericoB</Med>
        </MedicaEquiv>
</Medicamentos>ERRO: marca de fecho nao esperada!
```

4 Makefile

```
all1 : to_tex to_dvi to_pdf

all2: to_tex to_dvi to_pdf to_compile

to_tex: pl105f01resDCC.w
    nuweb pl105f01resDCC.w

to_dvi: pl105f01resDCC.tex
    latex pl105f01resDCC.tex

to_pdf: pl105f01resDCC.dvi
    dvipdf pl105f01resDCC.dvi
```

```
to_compile: ex21a1 ex21a2 ex21c ex22 ex23 ex31 ex32 ex33 ex33a ex33b


ex21a1: lex ex2_1a1.l
    gcc lex.yy.c -o ex2_1a1

ex21a2: lex ex2_1a2.l
    gcc lex.yy.c -o ex2_1a2

ex21c:  lex ex2_1c.l
    gcc lex.yy.c -o ex2_1c

ex22:   lex ex2_2.l
    gcc lex.yy.c -o ex2_2

ex23:   lex ex2_3.l
    gcc lex.yy.c -o ex2_3

ex31:   lex ex3_1.l
    gcc lex.yy.c -o ex3_1

ex32:  lex ex3_2.l
    gcc lex.yy.c -o ex3_2

ex33:   lex ex3_3.l
    gcc lex.yy.c -o ex3_3

ex33a:  lex ex3_3a.l
    gcc lex.yy.c -o ex3_3a

ex33b:  lex ex3_3b.l
    gcc lex.yy.c -o ex3_3b

ex33c:  lex ex3_3c.l
    gcc lex.yy.c -o ex3_3c

clean:
    rm -f ex2_1a1 ex2_1a2 ex2_1c ex2_2 ex2_3 ex3_1 ex3_2 ex3_3 ex3_3a ex3_3b ex3_3c
    rm -f *.log *.dvi *.toc *.aux lex.yy.c
```

5 Ficheiros

"ex2_1.txt" Defined by 3a.
"ex2_1a1.l" Defined by 2a.
"ex2_1a2.l" Defined by 2b.
"ex2_1b.l" Defined by 3b.
"ex2_1c.l" Defined by 4.
"ex2_1c.txt" Defined by 5.
"ex2_2.l" Defined by 6a.
"ex2_2.txt" Defined by 6b.
"ex2_3.l" Defined by 7, 8a.

"ex2_3.txt" Defined by 8b.
"ex3_1.l" Defined by 11.
"ex3_1.txt" Defined by 10.
"ex3_2.l" Defined by 14.
"ex3_2.txt" Defined by 13.
"ex3_3.l" Defined by 16, 17a.
"ex3_3.txt" Defined by 15.
"ex3_3a.l" Defined by 17b.
"ex3_3b.l" Defined by 18.
"ex3_3c.l" Defined by 19.
"ex3_3c.txt" Defined by 20.