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

2Âª 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 Gramáticas Independentes de Contexto (GIC) para definir a sintaxe de Linguagens;
- uso de Gramáticas Tradutoras (GT) para definir a semântica estática e dinâmica de Linguagens;
- uso de GT para desenvolver programas eficientes, baseados em algoritmos standard guiados por Tabelas de Decisão (construídas a partir de Autómatos Finitos Deterministas com stack), para reconhecer e processar Linguagens, desencadeando Acções Semânticas específicas ao reconhecer as produções da gramática;
- geração automática de programas a partir de especificações formais;
- uso da ferramenta Yacc, disponível em ambiente Linux, para geração automática de processadores de linguagens, nomeadamente para criação de *Tradutores Dirigidos pela Sintaxe* (*Analisadores Sintácticos* e *Analisadores Semânticos*).

2 Tradutores Dirigidos pela Sintaxe

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 e converter todas as palavras correctas em símbolos terminais dessa linguagem. O segundo nível é a **análise sintáctica** que pega nos símbolos recebidos do AL e verifica se a sequência em causa respeita as regras de derivação, ou produções, da gramática. O terceiro nível é a **análise semântica** que calcula o valor, ou significado, exacto da frase e, então, valida se a sequência de símbolos sintacticamente correcta cumpre todas as condições de contexto que a tornam semanticamente válida. O quarto nível é a **tradução** que pega no significado exacto da frase válida e constrói, ou calcula, o resultado final.
Com esse fim em vista e assumindo que o 1Âº nível já está implementado (graças ao uso do Flex para gerar o **Analisador Léxico (AL)**), propõe-se para esta aula o recurso à ferramenta Yacc para gerar os **Analisadores Sintáctico e Semântico** e o **Tradutor** a partir da Gramática Tradutora da Linguagem a processar.
Para cada um dos exercícios, proceda em três etapas:

1. escreva a GIC e gere o *Parser* (Analisador Sintáctico) que lhe permitirá, apenas, verificar a correcção sintáctica das frases;

2. escreva, depois, uma primeira versão da GT, acrescentando á GIC anterior as Acções Semânticas necessárias para, apenas, validar a correcção semântica das frases;

3. escreva, a seguir, uma versão final da GT, acrescentando á GT anterior as Acções Semânticas necessárias para calcular e escrever o resultado desejado.

2.1 Máquina de Venda de Chocolates

Retome o problema 3.1 da Ficha 1, em que se pretende simular o funcionamento de uma máquina de vender chocolates dado o stock no início do dia, a quantia inicial de trocos e os registos das vendas diárias.
O objectivo, para esta aula, é calcular *o stock final e o dinheiro acumulado*, considerando todos os pedidos válidos ao longo do dia. O sistema deve ainda assinalar todos os pedidos que não puderam ser satisfeitos, por o chocolate pedido não existir em stock, ou a quantia não ser suficiente.
Concretamente, o que se pretende é: que desenvolva, com auxílio do Gerador Yacc, o processador (reconhecedor+calculador e verificador) pedido, tomando por base a gramática da linguagem para descrever o estado inicial da máquina e os registos de vendas efectuadas durante o dia—criada na aula anterior (Ficha Prática 1); utilize o AL desenvolvido também nessa aula.

Comecemos por considerar o seguinte exemplo (já referido na ficha anterior):

Exemplo

```
"exemplo2_1f02.txt" 2 ≡
     (
       STOCK  { twix - 0.70 - 5, mars - 0.50 - 1, dove - 0.65- 10, kitkat - 0.60 - 9 } ;

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

       VENDAS { dove - 1.0, mars - 0.7, kitkat - 0.8, kitkat - 0.9,
         mars - 0.5, dove - 1., twix - 7. }
     )
     ◇
```

Solução

- Lex

"ex2_1f02.l" 3 ≡

```
%{
#include "y.tab.h"
%}

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}                   { yylval.qt=atoi(yytext);     return(INT);           }
{real}                  { yylval.pr=atof(yytext);     return(REAL);          }
{string}                { yylval.name=strdup(yytext); return(ID);            }
{separadores}           { return(yytext[0]);     }
{comentarios}|[ \n\t]   { ; }
.                       { printf("Unrecognized symbol"); }

%%

int yywrap(void) { return 1;}

◇
```

- Yacc

"ex2_1f02.y" 4 ≡

```
%{
#include <stdio.h>
#include "mvc.c"


MVC *mvc;
float money=0.0;

%}

%union {
   char *name;
   int qt;
   float pr;
}

%type <name> Nome
%type <qt> Quantidade
%type <pr> QuantiaI IdMoeda Preco

%token STOCK TROCOS VENDAS
%token <name> ID
%token <pr> REAL
%token <qt> INT



%%

MVC        : '(' STOCK  '{' Stock '}' ';'
                TROCOS '{' Trocos '}' ';'
                VENDAS '{' Vendas '}'
             ')'
           ;
Stock      : Descricao
           | Stock ',' Descricao
           ;
Descricao  : Nome '-' Preco '-' Quantidade   { mvc = insert(mvc,$1,$3,$5); }
           ;
Trocos     : Moeda
           | Trocos ',' Moeda
           ;
Moeda      : IdMoeda '-' Quantidade             { money += $1*$3;          }
           ;
Vendas     : Venda
           | Vendas ',' Venda
           ;
Venda      : Nome '-' QuantiaI                  { if (lookupQt(mvc,$1) != 0) {
                                                      money += lookupPrice(mvc,$1);
                                                      update(mvc,$1);
                                                  }
                                                  else
                                printf("\nATENCAO!!! Pedido de %s nao satisfeito!\n",$1);
                                     }
           ;
Nome       : ID                                 { $$ = $1;        }
           ;
Preco      : REAL          { $$ = $1;        }
           ;
Quantidade : INT           { $$ = $1;        }
           ;
IdMoeda    : REAL          { $$ = $1;        }
           ;
QuantiaI   : REAL          { $$ = $1;        }
           ;
```

- Código C

"mvc.c" 5 ≡

```
typedef struct mvc {
   char *nome;
   int quantity;
   float price;
   struct mvc *next;
} MVC;

MVC* init(MVC *mvc) {
   mvc = NULL;
   return mvc;
}


MVC* insert(MVC *mvc, char *nome1, float pr, int qt)
{
   if (mvc == NULL) {
      mvc = (MVC*) malloc(sizeof(MVC));
      mvc->nome       = nome1;
      mvc->price      = pr;
      mvc->quantity   = qt;
   }
   else
      mvc->next=insert(mvc->next,nome1,pr,qt);

   return mvc;
}

float lookupPrice(MVC *mvc,char *nome1) {

   while (strcmp(nome1,mvc->nome)!=0)
      mvc = mvc->next;

   return mvc->price;
}

int lookupQt(MVC *mvc,char *nome1) {

   while (strcmp(nome1,mvc->nome)!=0)
      mvc = mvc->next;

   return mvc->quantity;
}
```

◇

File defined by 5, 6.

"mvc.c" 6 ≡

```
void update(MVC *mvc,char *nome1) {

   while (strcmp(nome1,mvc->nome)!=0)
      mvc = mvc->next;

   mvc->quantity = mvc->quantity-1;
}

void print(MVC *stock) {
   printf("\n\n\t---------- Estado final da maquina -----------\n");
   printf("\t|\t\t\t\t\t     |\n");
        printf("\t|\tChocolate    |\t   Quantidade\t     |\n");
        printf("\t-----------------------------------------------\n");

   while(stock!=NULL) {
      printf("\t|\t%s\t     |\t\t%d\t     |\n",stock->nome,stock->quantity);
      stock=stock->next;
   }

        printf("\t-----------------------------------------------\n");
   printf("\n\n");
}
```

◇

File defined by 5, 6.

2.2 Anuário dos Medicamentos brancos

Considere agora de novo o problema da Ficha 1, desta vez o 3.2, em que se pretende criar um sistema de consulta a esses medicamentos acessível a qualquer farmácia via um browser HTML. Como então foi explicado, 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).
No contexto desta aula o que se pretende é: que desenvolva, com auxílio do Gerador Yacc, o processador (reconhecedor+calculador e verificador) pedido, tomando por base a gramática da linguagem para definir o ano a que o Symposium Terapêutico diz respeito, a lista das classes e de fabricantes válidos, e descrever a informação envolvida no lote de medicamentos a considerar—criada na aula anterior (Ficha Prática 1); utilize o AL desenvolvido também nessa aula. Não se esqueça que o seu sistema deve detectar e sinalizar todas as situações em que a classe do medicamento ou os fabricantes indicados não sejam válidos (não façam parte da lista inicial).

Exemplo

"exemplo2_2f02.txt" 7 ≡

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

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

◇

Solução

- Lex

"ex2_2f02.l" 8 ≡

```
%{
#include "y.tab.h"
%}

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

%%

int yywrap(void) { return 1;}

◇
```

- Yacc

"ex2_2f02.y" 9 ≡

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

%token SYMP ANO ID REAL INT

%%

SymposiumT      : SYMP ':' Ano
                  '[' Classes ']'
                  '[' Medicamentos ']'    { printf("Texto reconhecido!"); }
                ;
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            : ID
                ;
NomeC           : ID
                ;
Comp            : ID
                ;
NFab            : ID
                ;
Pr              : REAL
                ;
Cod             : INT
                ;


%%


int yyerror(char* error) {
   fprintf(stdout,"Error: %s\n", error);
   return 0;
}

int main() {
  return yyparse();
}
```

◇

2.3 Documento anotado em XML

Neste caso pretende-se processar Documentos XML—textos vulgares semeados de anotações, ou marcas, tal como descrito no exercício 3.3 da Ficha 1.
Tomando por base a descrição de Documento XML aí apresentada, pretende-se desenvolver um programa que valide se toda a marca que abre fecha, pela ordem correcta (uma marca aberta dentro de outra, terá de fechar antes da primeira), e que liste todas as marcas encontradas indicando a frequência de cada uma (número de ocorrências sobre o total de marcas). O processador também deve verificar que a mesma marca abre sempre associada ao mesmo conjunto de atributos. Se o documento-fonte for válido, deve então ser gerada uma versão LaTeX em que cada fragmento de texto marcado é assinalado entre chavetas precedidas por um comando LaTeX cujo nome é igual ao nome do elemento.
No contexto desta aula o que se pretende é: que desenvolva, com auxílio do Gerador Yacc, o processador (reconhecedor+calculador e verificador) pretendido, tomando por base a gramática de um Documento XML criada na aula anterior (Ficha Prática 1); utilize o AL desenvolvido também nessa aula.

Solução - 1a Parte

- Lex

"ex2_3f02.l" 11 ≡

```
%{
#include <stdio.h>
#include "y.tab.h"
%}

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

%x FimMarca

%%

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

%%

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

◇

- Yacc

"ex2_3f02.y" 12 ≡

```
%{
#include <stdio.h>
#include "list.c"

int open = 0, close = 0;
char * abre;

%}

%union
{
        int  number;
        char *name;
}

%token <name> ID VAL PCDATA
%token <char> IG BEA BEF BD

%type <name> IdEle

%start DocXML

◇
```

File defined by 12, 13.

"ex2_3f02.y" 13 ≡

```
%%

DocXML     : MarcAbre Conteud MarcFech          { printf("Marcas Abertura: %d.\n", open);
                                                  printf("Marcas Fecho: %d.\n\n", close);
                                                }
           ;
MarcAbre   : BEA Elem BD                        { open++; }
      ;
MarcFech   : BEA BEF IdEle BD                   { close++;
                                                  abre = pop();
                                                  if (strcmp(abre,$3)!=0) {
                                                       printf("Documento mal formado!\n");
                                                       exit(0); }
                                                 }
           ;
Elem       : IdEle Atribs                       { push($1); }
           ;
Atribs     : Atribs Atrib
           |
           ;
Atrib      : IdAtr IG VAL
           ;
Conteud    : Compon
           | Conteud Compon
           ;
Compon     : MarcAbre
           | MarcFech
           | PCDATA
           ;
IdEle      : ID
           ;
IdAtr      : ID
           ;

%%


int yyerror(char* error) {
   fprintf(stdout,"Error: %s\n", error);
   return 0;
}


int main() {
  initList();
  return yyparse();
}
◇
```

File defined by 12, 13.

- Código C

"list.c" 14a ≡

```
char * list[50];

void initList() {
        int i = 0;

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

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

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

   list[i] = pal;
}

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

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

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

      return aux;
}
```

◇

- Exemplo de teste

"exemplo2_3f02.txt" 14b ≡

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

◇

3 Makefile

"makefile" 15 ≡

```
# ex2_1
ex21:
   lex ex2_1f02.l
   yacc -d ex2_1f02.y
   gcc lex.yy.c y.tab.c -o ex21

# ex2_2
ex22:
   lex ex2_2f02.l
   yacc -d ex2_2f02.y
   gcc lex.yy.c y.tab.c -o ex22

# ex2_3
ex2_3:
   lex ex2_3.l
   yacc -d ex2_3.y
   gcc lex.yy y.tab.c -o ex23

# clean
clean:
   rm -f y.tab.c y.tab.c lex.yy.c
   rm ex21 ex22 ex23
```

◇

4 Ficheiros

"ex2_1f02.l" Defined by 3.
"ex2_1f02.y" Defined by 4.
"ex2_2f02.l" Defined by 8.
"ex2_2f02.y" Defined by 9.
"ex2_3f02.l" Defined by 11.
"ex2_3f02.y" Defined by 12, 13.
"exemplo2_1f02.txt" Defined by 2.
"exemplo2_2f02.txt" Defined by 7.
"exemplo2_3f02.txt" Defined by 14b.
"list.c" Defined by 14a.
"makefile" Defined by 15.
"mvc.c" Defined by 5, 6.