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

3º Ficha Prática

Ano Lectivo de 05/06

1 Objectivos

Este ficha prática contém 1 exercício para ser resolvido nas aulas teórico-práticas com vista a consolidar definitivamente os conhecimentos práticos 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

Na medida em que esta ficha tem por objectivo reforçar os conhecimentos e aptidões que os alunos devem ter adquirido ao resolver a Ficha 2, começa-se por relembrar os conceitos básicos aí introduzidos.
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 sequÃªncia 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 sequÃªncia 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.
Tendo em vista o a consolidação da capacidade de desenvolver PLs com recurso a Geradores de Compiladores, propõe-se para esta aula o recurso à ferramenta Flex para gerar o **Analisador Léxico (AL)**), e 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 resolver o exercício proposto, proceda em etapas:

1. escreva a GIC; identifique os símbolos terminais, descreva-os à custa de Expressões Regulares e gere o AL; gere o *Parser* (Analisador Sintáctico) que lhe permitirá, apenas, verificar a correcção sintáctica das frases;

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

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

2.1 Gerador de Indices Remissivos

Considere uma situação em que se pretende gerar automaticamente, a partir de uma descrição como a que se ilustra abaixo, o Índice Remissivo a inserir no fim de um determinado documento. Recorda-se que um Índice Remissivo é uma lista, ordenada alfabeticamente, de todos os termos a destacar num documento, com a indicação das páginas do documento onde cada termo foi definido.
A descrição inicial, escrita de acordo com uma linguagem específica que deverá ser definida para o efeito, permite indicar para cada página (referida pelo seu número ou nome do apÃªndice) a lista de termos a destacar. Abaixo ilustra-se uma frase válida da dita Linguagem:

```
"exemplo.txt" 2 ≡

     INDICE
       1 = processador, linguagem, compilador
       2 = compilador, interpretador, gramatica
       3 = gramatica, GR, GIC
       4 = gramatica, linguagem, reconhecedor
       A = gramatica, YACC
       B = LRC, LISA
     FIM DO INDICE
     ◇
```

O Índice que deve ser gerado para este exemplo é apresentado a seguir:

```
INDICE REMISSIVO
        processador: 1
        linguagem: 1 4
        compilador: 1 2
        interpretador: 2
        gramatica: 2 3 4 A
        GR: 3
        GIC: 3
        reconhecedor: 4
        YACC: A
        LRC: B
        LISA: B
```

No contexto desta aula o que se pretende é: que escreva a gramática que define a linguagem pretendida e que desenvolva, com auxílio dos Geradores Lex e Yacc e seguindo os passos acima, o processador (reconhecedor+calculador e verificador) necessário, tomando por base essa gramática.

3 Resolução:

3.1 Versão 1

3.1.1 Lex

"ex2_1f03.l" 3 ≡

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

integer [0-9]
charI   [aA-zZ]
pal     [aA-zZ]+

%%

"FIM DO INDICE"     { return FIM;                       }
"INDICE"            { return INDICE;                     }
\=|\,               { return yytext[0];                  }
{integer}           { yylval.id = strdup(yytext); return IND_INT;         }
{charI}             { yylval.id = strdup(yytext); return IND_CHAR;        }
{pal}               { yylval.id = strdup(yytext); return PAL;             }
[ \n\t]

%%

int yywrap(void) { return 1;}

◇
```

3.1.2 Yacc

"ex2_1f03v1.y" 4 ≡

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

Term *newList;
char *indexT;

%}

%union {
   char *id;
}

%token INDICE FIM
%token <id> IND_INT IND_CHAR
%token <id> PAL

%start IndiceRemissivo

%%

IndiceRemissivo : INDICE ListaInd FIM  { print(newList); }
                ;
ListaInd        : Elem
                | ListaInd Elem
                ;
Elem            : Ident '=' Termos
                ;
Ident           : IND_INT     { indexT = strdup($1); }
                | IND_CHAR    { indexT = strdup($1); }
                ;
Termos          : Termo
                | Termos ',' Termo
                ;
Termo           : PAL               { newList = insert(newList,$1,indexT); }
                ;

%%

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

int main() {
  newList = initList();
  yyparse();
  return 0;
}
```

◇

3.2 Código *C*

"term.c" 5 ≡

```
#include <stdio.h>
#define MAX 10

typedef struct termList {
   char *identifier;
   char *index[MAX];
        struct termList *next;
} Term;

Term* initList() {
   return NULL;
}


Term* insert(Term *term, char *name, char* ind)
{
   int i = 0;
   if (term == NULL) {
      term = (Term*) malloc(sizeof(Term));
      term->identifier = name;
      term->index[0]   = ind;
   }
   else {
      if (strcmp(name,term->identifier)==0) {
      while(term->index[i]!=NULL)
         i++;
      term->index[i]=ind;
        }
      else
      term->next=insert(term->next,name,ind);
   }
   return term;
}

void print(Term *term) {
   int i;

   printf("\n\nINDICE REMISSIVO\n");
   while (term!=NULL) {
      printf("\t%s: ", term->identifier);
      for (i = 0; i < MAX; i++)
         if (term->index[i] != NULL)
            printf("%s ", term->index[i]);
         else
            break;
      printf("\n");
      term = term->next;
   }
   printf("\n");
}
```

◇

3.3 Versão 2

"ex2_1f03v2.l" 6 ≡

```
%{
#include "indice_remissivo.h"
#include "indice.tab.h"

int lineno = 1;
%}


%option noyywrap

%%

INDICE              { return INIINDICE; }
FIM[ ]+DO[ ]+INDICE { return FIMINDICE; }
"="                 { return IGUAL; }
","                 { return VIRGULA; }

[0-9]+              { yylval.numero = atoi(yytext); return INDICE; }
[A-Z][0-9]*         { yylval.string = strdup(yytext); return APENDICE; }
[a-zA-Z][a-zA-Z]+   { yylval.string = strdup(yytext); return TERMO; }

[ \t\r]
\n                  { lineno++; }


%%
◇
```

"ex2_1f03v2.y" 7 ≡

```
%{
#include "indice_remissivo.h"

extern int lineno;
extern char *yytext;

listaEntradas *parse_result = NULL;

void yyerror( const char *msg) {
   fprintf( stderr, "Erro linha %d: %s - %s\n", lineno, yytext, msg); }
%}

%union {
   int numero;
   char *string;
   seccao *s;
   listaPalavras *palavras;
   entrada *e;
   listaEntradas *l;
}

%token INIINDICE "INDICE"
%token FIMINDICE "FIM DO INDICE"
%token IGUAL     "="
%token VIRGULA   ","

%token <numero> INDICE
%token <string> APENDICE
%token <string> TERMO

%type <s> seccao
%type <palavras> lstpalavras
%type <e> entrada
%type <l> lstentradas

%%

indice: "INDICE" lstentradas "FIM DO INDICE" { parse_result = $2;                 }
      ;

lstentradas: entrada                         { $$ = listaEntradas_new( $1);        }
           | lstentradas entrada             { $$ = listaEntradas_insert( $1, $2); }
           ;

entrada: seccao "=" lstpalavras              { $$ = entrada_new( $1, $3);          }
       ;

seccao: INDICE                               { $$ = seccao_new_indice( $1);        }
      | APENDICE                             { $$ = seccao_new_apendice( $1);      }
      ;

lstpalavras: TERMO                           { $$ = listaPalavras_new( $1);        }
           | lstpalavras "," TERMO           { $$ = listaPalavras_insert( $1, $3); }
           ;

%%

int main(void) {
   if( yyparse() == 0) {
      printf( "O ficheiro foi processado com sucesso!\n");
   }

   prettyprint( parse_result);

   return 0;
}
◇
```

"indice_remissivo.h" 8 ≡

```
#ifndef _INDICE_REMISSIVO_H_
#define _INDICE_REMISSIVO_H_

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

enum tipo_seccao { SECCAO_INDICE, SECCAO_APENDICE };

typedef struct _seccao_ {
    enum tipo_seccao tipo;
    union {
       int indice;
       char *apendice;
    } u;
} seccao;

typedef struct _listaPalavras_ {
   char *palavra;
   struct _listaPalavras_ *next;
} listaPalavras;

typedef struct _entrada_ {
   seccao *s;
   listaPalavras *lista;
} entrada;

typedef struct _listaEntradas_ {
   entrada *e;
   struct _listaEntradas_ *next;
} listaEntradas;

void prettyprint( listaEntradas *lst);

listaEntradas *listaEntradas_new( entrada *e);
listaEntradas *listaEntradas_insert( listaEntradas *lista, entrada *e);


entrada *entrada_new( seccao *s, listaPalavras *lista);

listaPalavras *listaPalavras_new( char *p);
listaPalavras *listaPalavras_insert( listaPalavras *lista, char *p);

void listaPalavras_print( listaPalavras *lista);

seccao *seccao_new_indice( int i);
seccao *seccao_new_apendice( char *s);


#endif
```

◇

"indice_remissivo.c" 9 ≡

```
#include "indice_remissivo.h"

void print_entrada( entrada *e);
void print_seccao( seccao *s);

void prettyprint( listaEntradas *lst) {
    printf( "INDICE\n");
    for( ; lst != NULL; lst = lst ->next) {
        print_entrada(lst->e);
    }
    printf( "FIM DO INDICE\n");
}

void print_entrada( entrada *e) {
   listaPalavras *lst = NULL;

   print_seccao(e->s);
   for(lst = e->lista; lst != NULL; lst = lst->next) {
      printf("%s", lst->palavra);
      if( lst->next != NULL)
         printf(", ");
      else
         printf("\n");
   }
}

void print_seccao( seccao *s) {
   if(s->tipo == SECCAO_INDICE) {
      printf( "%d = ", s->u.indice);
   } else if( s->tipo == SECCAO_APENDICE) {
      printf( "%s = ", s->u.apendice);
   }
}

listaEntradas *listaEntradas_new( entrada *e) {
   listaEntradas *res = NULL;

   res = (listaEntradas *) calloc( 1, sizeof(listaEntradas));
   if( res == NULL) return NULL;

   res->e = e;
   return res;
}

listaEntradas *listaEntradas_insert( listaEntradas *lista, entrada *e) {
   listaEntradas *res = NULL;

   res = listaEntradas_new(e);
   if( res == NULL) return NULL;

   res->next = lista;
   return res;
}

entrada *entrada_new( seccao *s, listaPalavras *lista) {
   entrada *res = NULL;

   res = (entrada *) calloc( 1, sizeof(entrada));
   if( res == NULL) return NULL;

   res->s = s;
   res->lista = lista;

   return res;
}


listaPalavras *listaPalavras_new( char *p) {
```

4 Ficheiros

"ex2_1f03.l" Defined by 3.
"ex2_1f03v1.y" Defined by 4.
"ex2_1f03v2.l" Defined by 6.
"ex2_1f03v2.y" Defined by 7.
"exemplo.txt" Defined by 2.
"indice_remissivo.c" Defined by 9.
"indice_remissivo.h" Defined by 8.
"term.c" Defined by 5.