## Compiladores 1ª Série de Exercícios

**CES-41** 

Professor: Fábio Carneiro Mokarzel Aluno: Carlos Matheus Barros da Silva

Julho de 2019

Todos os códigos desta lista podem ser acessados no repositório do GitHub: https://github.com/CarlosMatheus/CES-41-Exercise-Lists

## Exercício 1

O Exercicio foi resolvido com sucesso. Seu funcionamento pode ser observado pela saída representada pelo Código 2 que é referente a entrada representada no Código 1.

Pilha	Entrada	Ação	Produção
	id * ( ( id + id ) * ( id * ( id + id ) ) ) \$ id * ( ( id + id ) * ( id * ( id + id ) ) ) \$	Expandir Expandir	E -> TE'
' T' F	id * ( (id + id ) * (id * (id + id ) ) ) \$  id * ( (id + id ) * (id * (id + id ) ) ) \$	Expandir	F -> id
'T'id	id * ( (id + id ) * (id * (id + id ) ) ) \$	Dempilhar, avançar	1 - 1
'т'	* ( ( id + id ) * ( id * ( id + id ) ) ) \$	Expandir	T' -> *FT'
' T' F *	* ( (id + id ) * (id * (id + id ) ) ) \$	Dempilhar, avançar	
'T'F 'T')E(	( ( id + id ) * ( id * ( id + id ) ) ) \$ ( ( id + id ) * ( id * ( id + id ) ) ) \$	Expandir Dempilhar, avançar	F -> (E)
' T' ) E	( id + id ) * ( id * ( id + id ) ) ) \$	Expandir	E -> TE'
'T')E'T	(id + id) * (id * (id + id)))\$	Expandir	T -> FT'
'T') E'T'F	( id + id ) * ( id * ( id + id ) ) ) \$	Expandir	F -> (E)
'T')E'T')E(	( id + id ) * ( id * ( id + id ) ) ) \$	Dempilhar, avançar	l
' T' ) E' T' ) E ' T' ) E' T' ) E' T	id + id ) * ( id * ( id + id ) ) ) \$	Expandir Expandir	E -> TE'
' T' ) E' T' ) E' T ' T' ) E' T' ) E' T' F	id + id ) * ( id * ( id + id ) ) ) \$ id + id ) * ( id * ( id + id ) ) ) \$	Expandir	F -> id
' T' ) E' T' ) E' T' id	id + id ) * ( id * ( id + id ) ) ) \$	Dempilhar, avançar	1 - 14
'T')E'T')E'T'	+ id ) * ( id * ( id + id ) ) ) \$	Expandir	T' -> ε
'T')E'T')E'	+ id ) * ( id * ( id + id ) ) ) \$	Expandir	E' -> +TE
' T' ) E' T' ) E' T +	+ id ) * ( id * ( id + id ) ) ) \$	Dempilhar, avançar	
' T' ) E' T' ) E' T ' T' ) E' T' ) E' T' F	id ) * ( id * ( id + id ) ) ) \$	Expandir	T -> FT'
'T')E'T')E'T'F 'T')E'T')E'T'id	<pre>id ) * ( id * ( id + id ) ) ) \$ id ) * ( id * ( id + id ) ) ) \$</pre>	Expandir Dempilhar, avançar	F -> id
'T')E'T'	) * ( id * ( id + id ) ) ) \$	Expandir	Τ' -> ε
'T')E'T')E'	) * ( id * ( id + id ) ) ) \$	Expandir	E' -> ε
'T')E'T')	) * ( id * ( id + id ) ) ) \$	Dempilhar, avançar	i -
'T')E'T'	* ( id * ( id + id ) ) ) \$	Expandir	T' -> *FT
'T')E'T'F*	* ( id * ( id + id ) ) ) \$	Dempilhar, avançar	
'T')E'T'F 'T')E'T')E(	( id * ( id + id ) ) ) \$	Expandir	F -> (E)
' T' ) E' T' ) E ( ' T' ) E' T' ) E	( id * ( id + id ) ) ) \$ id * ( id + id ) ) ) \$	Dempilhar, avançar Expandir	E -> TE'
' T' ) E' T' ) E' T	id * (id + id ) ) ) \$	Expandir	T -> FT'
'T')E'T')E'T'F	id * ( id + id ) ) ) \$	Expandir	F -> id
'T') E'T') E'T'id	id * ( id + id ) ) ) \$	Dempilhar, avançar	
' T' ) E' T' ) E' T'	* ( id + id ) ) ) \$	Expandir	T' -> *FT
' T' ) E' T' ) E' T' F * ' T' ) E' T' ) E' T' F	* ( id + id ) ) ) \$	Dempilhar, avançar Expandir	F -> (E)
'T')E'T')E'T')E(	( id + id ) ) ) \$ ( id + id ) ) ) \$	Dempilhar, avançar	F -> (E)
'T')E'T')E	id + id ) ) ) \$	Expandir	E -> TE'
'T')E'T')E'T')E'T	id + id ) ) ) \$	Expandir	T -> FT'
'T')E'T')E'T')E'T'F	id + id ) ) ) \$	Expandir	F -> id
' T' ) E' T' ) E' T' ) E' T' id	id + id ) ) ) \$	Dempilhar, avançar	
' T' ) E' T' ) E' T' ) E' T' ' T' ) E' T' ) E' T' ) E'	+ id ) ) ) \$  + id ) ) ) \$	Expandir Expandir	T' -> ε  E' -> +TE
'T')E'T')E'T+	+ id ) ) ) \$	Dempilhar, avançar	L -> +1E
'T')E'T')E'T')E'T	id ) ) \$	Expandir	T -> FT'
'T')E'T')E'T')E'T'F	id ) ) ) \$	Expandir	F -> id
' T' ) E' T' ) E' T' ) E' T' id	id ) ) ) \$	Dempilhar, avançar	<u>_</u> .
' T' ) E' T' ) E' T' ) E' T'	() ) ) \$	Expandir	T' -> ε
'T')E'T')E'T')E'	() ) ) \$	Expandir	E' -> ε
' T' ) E' T' ) E' T' ) ' T' ) E' T' ) E' T'	) ) ) \$  ) ) \$	Dempilhar, avançar Expandir	  T' -> ε
T ) E T ) E T 'T' ) E'T' ) E'	) ) \$	Expandir	T -> ε  E' -> ε
'T')E'T')	) ) \$	Dempilhar, avançar	
' T' ) E' T'	) \$	Expandir	Τ' -> ε
' T' ) E'	) <del>\$</del>	Expandir	E' -> ε
' T' )	) \$	Dempilhar, avançar	
'т'	\$	Expandir	T' -> ε
1	\$	Expandir	E' -> ε
	\$	Dempilhar, avançar	
		Encerrar com sucesso	1

Figura 1: A tabela de execução para entrada id\*((id+id)\*(id\*(id+id)))\$

Figura 2: Tabela dos Primeiros

## Exercício 2

O Exercício foi resolvido com sucesso. Para a gramática do exemplo da Seção 5.4.4 do Capítulo V dos Slides Teóricos de CES-41 e a tabela de produções de um analisador preditor não-recursivo, no mesmo exemplo, foi feita a tabela de execução simulando a análise da sentença id \* ((id + id) \* (id \* (id + id)))\$. A tabela de execução pode ser vista na Figura 1.

## Exercício 3

O Exercicio foi resolvido com sucesso. A tabela de produções do analisador sintático preditor não-recursivo foi feita com sucesso. O Primeiro de cada símbolo pode ser visto na Figura 2. O Seguinte pode ser visto na Figura 3. A tabela dos Primeiros e Seguinstes pode ser vista na Figura 4. A tabela de produções do analisador sintático preditor não-recursivo pode ser vista na Figura 5 inteira. Devido à sua largura, ela foi separada em dividida em seguimentos em apresentada da Figura 6 à Figura 9 de modo a facilitar a visualização.

```
/* Programa para contar as ocorrencias das
palavras de um texto */

program AnaliseDeTexto {

/* Variaveis globais */

global:
    char nomes[50,10], palavra[10];
    int ntab, nocorr[50];
        char c; logic fim;

functions:

/* Funcao para procurar uma palavra na tabela de palavras */

int Procura () {
```

A	Seguinte(A)
Prog	\$
Decls	{
CmdComp	
Declaracao	int real {
LDaux	{
Tipo	ID
ListId	;
LIaux	;
ListCmd	}
Comando	{ ID }
LCaux	}
CmdAtrib	
Expressao	;)
Termo	+ ; )
Eaux	; )

Figura 3: Tabela dos Seguintes

A -> α	Prim(α)	Prim(A)	Seg(A)
Prog -> DeclsCmdComp Decls -> DeclaracaoLDaux LDaux -> E LDaux -> Decls	int real int real E int real	int real int real ɛ int real ɛ int real	\$ { {
Declaracao -> TipoListId; Tipo -> int Tipo -> real ListId -> IDLIaux	int real int real ID	int real int real int real int real ID	int real {     ID     ID     ;
LIaux -> \( \) LIaux -> ,ListId CmdComp -> {ListCmd} ListCmd -> ComandoLCaux	ε , { { ID	ε, ε, { ID	; ; }
LCaux -> \varepsilon LCaux -> ListCmd Comando -> CmdComp Comando -> CmdAtrib CmdAtrib -> ID=Expressao;	ε { ID ID ID	ε { ID ε { ID { ID ID	} { ID } { ID }
Expressao -> TermoEaux	( ID CTE	( ID CTE	; ); ); ); ); ); ); ); ); ); ); ); ); );

Figura 4: Tabela dos Primeiros e Seguintes

	Atom de extrado													
His terminal	1 ,	i int	real	I Ib		1 (	)			1 (	)	(18		1 8
Pers    Becla   Candomp   Becla   Candomp   Blass   Blass   ListEd   ListEd	line o c	Prop - Omeranding Declar-endeldous Beclaraces -= TipoListldy Ideas -> becla Tipo -= int	Pres -> Decisionalismo Decis Decisionalismo Beciscoco -> Elgodistify Ideos -> Decis Tipo -> resi	Listid -> DOLING Listid -> Considence Listid -> Considerio Listid -> Deliverio Listid -> Listid Listid -> Deliverio Listid -> Turodica Open -> Deliverio	linux -> ,Lintz	CadComp => (ListCad) LOREX -> r  ListCad - CamandacCosex Communds - CadComp Lideax -> ListCad	Monan -> c		Eng of Management	Expresse -> Terminaca Verm -> (Expresse)	From sit 1	Exposesso -> Sumoface Verso -> ON		
ListOnd Comands Ideas OmdArrib Expression Surms				Commando -> Conditorià Licura -> Listino Conditorià -> II-Compressas;	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		Edwar -> c		Eaux -> "Expressas			East of a	Terms or CTE	Turns of CTE

Figura 5: Tabela de produções

Não terminal	;	int	real
Prog Decls CmdComp Declaracao LDaux Tipo ListId LIaux ListCmd Comando LCaux CmdAtrib Expressao Termo Eaux	LIaux -> ε Εαυx -> ε	Prog -> DeclsCmdComp Decls -> DeclaracaoLDaux  Declaracao -> TipoListId; LDaux -> Decls Tipo -> int	Prog -> DeclsCmdComp Decls -> DeclaracaoLDaux  Declaracao -> TipoListId; LDaux -> Decls Tipo -> real

Figura 6: Tabela de produções, parte  $1\,$ 

Átomo de ent							
ID	,	<b> </b>	}	l			
ListId -> IDLIaux  ListCmd -> ComandoLCaux Comando -> CmdAtrib LCaux -> ListCmd CmdAtrib -> ID=Expressao; Expressao -> TermoEaux Termo -> ID	LIaux -> ,ListId	CmdComp -> {ListCmd}  LDaux -> ε  ListCmd -> ComandoLCaux  Comando -> CmdComp  LCaux -> ListCmd	LCaux -> ε				

Figura 7: Tabela de produções, parte 2

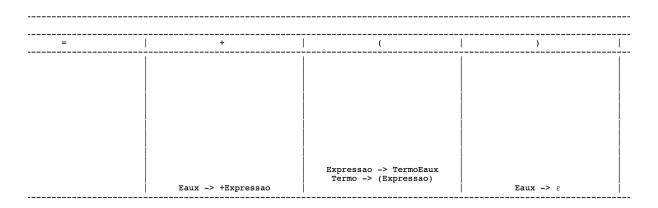


Figura 8: Tabela de produções, parte 3

CTE	*	\$	
	 		ĺ
Expressao -> TermoEaux Termo -> CTE			

Figura 9: Tabela de produções, parte 4

```
local:
  \mathbf{int} \ i \ , \ \inf \ , \ \sup \ , \ \operatorname{med} \ , \ \operatorname{posic} \ , \ \operatorname{compara} \ ;
      logic achou, fimteste;
statements:
  achou <- false; inf <- 1; sup <- ntab;
  while (!achou && sup >= inf) {
     \mod <- \ (\inf \ + \ \sup \ ) \ / \ 2;
     if (palavra[i] < nomes[med, i])
  compara <- ~1;</pre>
                  else if (palavra[i] > nomes[med,i])
                    compara <- 1;
                  if (palavra[i] = '\0' || nomes[med,i] = '\0')
                    fimteste <- true;
     if (compara = 0)
       achou <- true;
     else if (compara < 0)
       \sup <- \ \mathrm{med} \ - \ 1;
     else inf <- med + 1;
  \mathbf{if} (achou) posic <- med;
  else posic <- ~inf;</pre>
  return posic;
} /* Fim da funcao Procura */
/* Funcao para inserir uma palavra na tabela de palavras */
void Inserir (int posic) {
local:
  int i, j; logic fim;
statements:
  ntab <\!\!- ntab + 1;
  for (i \leftarrow ntab; i > posic+1; i \leftarrow i-1) {
          \begin{array}{lll} \text{fim} \; < - \; \; \text{false} \; ; \\ \text{for} \; \; (j \; < \! - \; 0 \; ; \; ! \; \text{fim} \; ; \; \; j \; < \! - \; j + 1) \; \; \{ \end{array}
               nomes[i,j] \leftarrow nomes[i-1,j];
               if (nomes[i,j] = '\0') fim \leftarrow true;
      nocorr[i] \leftarrow nocorr[i-1];
  }
       nomes[posic,j] <- palavra[j];
if (palavra[j] = '\0') fim <- true;
  nocorr[posic] <- 1;
} /* Fim da funcao Inserir */
/* Funcao para escrever a tabela de palavras */
void ExibirTabela () {
local:
 int i; logic fim;
statements:
                          ", "Palavra
  write ("
                    " Num. de ocorr.");
  for (i \leftarrow 1; i \leftarrow 50; i \leftarrow i+1) write ("-");
  for (i <- 1; i <= ntab; i <- i+1) {
  write ("\n"); fim <- false;</pre>
     write ("\n
     for (j <- 0; !fim; j <- j+1) {
    if (nomes[i,j] = '\0') fim <- true;
               else write (nomes[i,j]);
       }
```

```
write (" | ", nocorr[i]);
} /* Fim da funcao ExibirTabela */
/* Modulo\ principal\ */
main {
local:
  \mathbf{int} \ i \ , \ posic \ ;
       char c; logic fim;
statements:
  ntab <\!\! -0;
  write ("Nova palavra? (s/n): ");
  read(c);
  while (c = 's' || c = 'S') {
            write ("\nDigite a palavra: ");
     fim <- false;

for (i <- 0; !fim; i <- i+1) {
                read (palavra[i]);
if (palavra[i] = '\n') {
                      fim <- true;
                      palavra[i] <- '\0';
            }
     posic <- Procura ();
     if (posic > 0)
       nocorr[posic] <- nocorr[posic] + 1;</pre>
     else
       call Inserir (~posic, i);
            write ("\n\nNova palavra? (s/n): ");
            read (c);
  }
  call ExibirTabela ();
} /* Fim da funcao main */
} /* Fim do programa AnaliseDeTexto */
                             Código 1: Código de entrada para o analisador
program AnaliseDeTexto {
     global:
         char nomes [50, 10], palavra [10];
          int ntab, nocorr[50];
          char c;
         logic fim;
     functions:
          int Procura ( ) {
              local:
                   \mathbf{int} \ \mathsf{i} \ , \ \mathsf{inf} \ , \ \mathsf{sup} \ , \ \mathsf{med} \ , \ \mathsf{posic} \ , \ \mathsf{compara} \ ;
                    logic achou, fimteste;
              statements:
                   achou <- false ;
                   inf <\!\!-1 \ ;
                   \sup <- ntab ;
                   \mathbf{while} \ ( \ ! achou \&\& \ sup >= \ inf \ )
                        \bmod <\!- \ ( \ \inf \ + \ \sup \ ) \ / \ 2 \ ;
                        compara \leftarrow 0;
                        fimteste <- false ;
                        for ( i \leftarrow 0 ; !fimteste && compara = 0 ; i \leftarrow i + 1 )
```

```
_{
m else}
                      if ( palavra [i ]> nomes [med , i ])
                  compara <- 1;
if ( palavra [i ]= '\0' || nomes [med , i ]= '\0' )
                      fimteste <- true ;
             if (compara = 0)
                 achou <- true ;
             _{
m else}
                  if (compara < 0)
                      \sup < - \mod - 1;
                  else
                      inf < - med + 1;
         if ( achou )
             posic <- med ;
             posic <- ~ inf ;
         return posic ;
}
void Inserir ( int ) {
    local:
         int i, j;
         logic fim;
    statements:
        \begin{array}{l} ntab < - \ ntab \ + \ 1 \ ; \\ \textbf{for} \ ( \ i < - \ ntab \ ; \ i > = \ posic \ + \ 1 \ ; \ i < - \ i \ - \ 1 \ ) \end{array}
             {
                  fim <- true ;
             nocorr [i]<- nocorr [i - 1];
         fim <- false ;
         for (j < 0; !fim; j < j + 1)
             fim <- true ;
         nocorr [posic]<-1;
}
void ExibirTabela ( ) {
    local:
         int i;
         logic fim;
    statements:
         write ( "
                             ", "Palavra
                                                        ", " Num. de ocorr.") ;
         for ( i < -1 ; i <= 50 ; i < -i + 1 ) write ( "-") ;
         for (i \leftarrow 1; i \leftarrow ntab; i \leftarrow i + 1)
             write ( "\n
                                    ") ;
             fim <- false ;
             for (j \leftarrow 0; !fim; j \leftarrow j + 1)
             {
                  \mathbf{i}\,\mathbf{f}\ (\ nomes\ [\,i\ ,\ j\ ]=\ `\setminus 0\ '\ )
                      fim \leftarrow true;
                  else
                      write ( nomes [i , j ]);
             write ( " | ", nocorr [i ]);
        }
}
```

```
main {
        local:
                int i, posic;
                \mathbf{char}\ c \ ;
                logic fim;
        statements:
                ntab <- 0 ;

write ( "Nova palavra? (s/n): ") ;

read ( c ) ;

while ( c = 's' || c = 'S' )
                {
                       write ( "\nDigite a palavra: ") ; fim <- false ; for ( i <-0 ; !fim ; i <-i+1 )
                                \begin{array}{lll} \text{read} & ( & \text{palavra} & [ & i & ] \ ) & ; \\ \textbf{if} & ( & \text{palavra} & [ & i & ] = & ` \backslash n \ ' \ ) \end{array}
                                {
                                       \label{eq:fim_def} \text{fim} <\!\!- \text{ true };
                                        palavra [i]<- '\0';
                                }
                        }
                        posic <- Procura( ) ;
                        if (posic > 0)
                               nocorr [posic] <- nocorr [posic] + 1;
                        call Inserir( \tilde{} posic , i ) ; write ( \tilde{} \n\nNova palavra? (s/n): ") ;
                        read ( c ) ;
                call ExibirTabela( );
}
```

}

Código 2: Código de saída do analisador