

# TRABALHO 2 DE LABORATÓRIO

## ATIVIDADE PRÁTICA EM COMPILADORES

Esta atividade é um componente para a avaliação e desenvolvimento dos conhecimentos envolvidos na disciplina Compiladores. O valor dessa atividade é 10,0 e compõe a média de aprovação na disciplina conforme plano de curso.

Prof. Dra. Deborah Silva Alves Fernandes – UFG/INF

Goiânia, OUTUBRO, 2017.

## 1. INTRODUÇÃO

O trabalho introduzido nesse documento busca a realização de atividade prática Compiladores e compõe a nota T2 das atividades avaliativas expostas no plano de curso.

A disciplina de compiladores preocupa-se em estudar técnicas e teorias para a construção de um compilador. Para tal, durante o semestre investigar-se-á seus componentes sobre aspectos teóricos e práticos.

## 2. ATIVIDADE PRÁTICA T2

### 2.1. Regras DO TRABALHO

1. Trabalho individual ou em dupla (AS MESMAS DEFINIDAS NO T1);
2. O trabalho (códigos fonte e executáveis) será entregue via moodle na data definida pelo professor: (para cada dia de atraso serão descontados 0,3 por dia até o dia de apresentação).
3. As apresentações serão realizadas nos dias e horários definido pelo professor (dentro dos horários de aula regulares da disciplina).
4. O professor arguirá o(s) aluno(s) quanto a questões sobre o desenvolvimento do trabalho.
5. Em caso de duplas, o professor escolherá a qualquer momento da apresentação, quem responderá a pergunta a ser realizada. A nota será a mesma para ambos os alunos.
6. O aluno poderá escolher a linguagem de programação que será utilizada para desenvolver o trabalho. Portanto, é de responsabilidade do aluno que no dia da apresentação todo o aparato para execução do trabalho esteja disponível.
7. A evolução do trabalho será acompanhada pela professora durante as aulas até o dia da entrega.
8. Cópias de trabalhos de colegas ou de semestres anteriores terão nota 0,0.
9. O trabalho T2 deverá complementar o trabalho T1, já realizado anteriormente. **Não serão aceitos trabalhos os quais não contenham o trabalho T1 (analisador léxico) embutido.**
10. Durante a apresentação o professor poderá questionar quaisquer itens relacionados ao trabalho, ou seja detalhes do analisador sintático, conexão com o analisador léxico e estruturas e recursos utilizados na implementação.
11. Não é permitido o uso de geradores de analisadores léxico e sintático.

### 2.2. Atividade a ser desenvolvida

Desenvolver um programa computacional na linguagem escolhida que implemente:

1. Um analisador sintático SLR(0) que reconheça as sentenças que podem ser formadas a partir da gramática livre de contexto disponível na Figura 1.
2. Para tal, deve se seguir os passos abaixo:
  - a. Criar a gramática livre de contexto aumentada, caso necessário;
  - b. Enumerar a gramática;
  - c. Criar o autômato LR de itens pontilhados (itens canônicos) para formação da tabela sintática;
  - d. Gerar os conjuntos Primeiro e Seguinte (*first/follow*) dos não-terminais da gramática;
  - e. Construir a tabela sintática;
  - f. Implementar o algoritmo para análise sintática LR disponível na Figura 2.
3. O analisador sintático a ser desenvolvido deve basear-se no item 2.f e todas as vezes que fizer movimentos com o apontador de entrada  $a$ , mencionado no algoritmo da Figura 2, deverá chamar a função “Analisador Léxico” desenvolvida no trabalho T1.

4. A implementação do analisador sintático se dará através da implementação de um autômato de pilha. Para tomar as decisões sobre ação/redução, ele usará a tabela sintática (com 58 estados e 28 símbolos) e uma estrutura de dados do tipo pilha. A estrutura do tipo pilha poderá ser implementada ou pode ser utilizada uma já disponível em biblioteca da linguagem escolhida.
5. As lacunas da tabela sintática (espaços sem ações de redução/empilhamento/aceita) devem ser preenchidas com códigos de erros que deverão indicar na tela o tipo de erro sintático encontrado (se falta operador aritmético, relacional, atribuição, ...), juntamente com a linha e coluna (informação disponível do analisador léxico) da ocorrência do erro. Para maiores informações – consultar slides 16 a 22 do tópico Recuperação de erro no analisador sintático da disciplina no moodle.

### 3. Gramática livre de contexto a ser utilizada

A gramática disponível na Figura 1 deverá ser utilizada para o desenvolvimento do trabalho. Ela contém todas as produções necessárias para a realização de análise sintática da linguagem hipotética Mdpgol. A figura 2 apresenta o algoritmo de análise sintática a ser implementado.

Figura 1 – Gramática livre de contexto para análise sintática a ser desenvolvida.

1	$P' \rightarrow P$
2	$P \rightarrow \text{inicio } V \ A$
3	$V \rightarrow \text{var inicio } LV$
4	$LV \rightarrow D \ LV$
5	$LV \rightarrow \text{var fim};$
6	$D \rightarrow \text{id TIPO};$
7	$TIPO \rightarrow \text{int}$
8	$TIPO \rightarrow \text{real}$
9	$TIPO \rightarrow \text{lit}$
10	$A \rightarrow ES \ A$
11	$ES \rightarrow \text{leia id};$
12	$ES \rightarrow \text{escreva ARG};$
13	$ARG \rightarrow \text{literal}$
14	$ARG \rightarrow \text{num}$
15	$ARG \rightarrow \text{id}$
16	$A \rightarrow CMD \ A$
17	$CMD \rightarrow \text{id rcb LD};$
18	$LD \rightarrow OPRD \ opm \ OPRD$
19	$LD \rightarrow OPRD$
20	$OPRD \rightarrow \text{id}$
21	$OPRD \rightarrow \text{num}$
22	$A \rightarrow COND \ A$
23	$COND \rightarrow \text{CABEÇALHO CORPO}$
24	$\text{CABEÇALHO} \rightarrow \text{se (EXP\_R) então}$
25	$\text{EXP\_R} \rightarrow OPRD \ opr \ OPRD$
26	$\text{CORPO} \rightarrow ES \ \text{CORPO}$
27	$\text{CORPO} \rightarrow CMD \ \text{CORPO}$
28	$\text{CORPO} \rightarrow COND \ \text{CORPO}$
29	$\text{CORPO} \rightarrow \text{fimse}$
30	$A \rightarrow \text{fim}$

Figura 2 – Algoritmo para análise sintática LR. Fonte: fonte: Compiladores, Aho e Ullman-pg. 160.

```
[PSEUDO]seja  $a$  o primeiro símbolo de  $w$ $;  
while (1){ /* repita indefinidamente*/  
  seja  $s$  o estado no topo da pilha;  
  if ( ACTION[ $s,a$ ] = shift  $t$  ) {  
    empilha  $t$  na pilha;  
    seja  $a$  o próximo símbolo da entrada;  
  } else if ( ACTION[ $s,a$ ] = reduce  $A \rightarrow \beta$  ) {  
    desempilha símbolos  $|\beta|$  da pilha;  
    faça o estado  $t$  agora ser o topo da pilha;  
    empilhe GOTO[ $t,A$ ] na pilha;  
    imprima a produção  $A \rightarrow \beta$ ;  
  } else if ( ACTION[ $s,a$ ] = accept ) pare; /* a análise terminou */  
  else chame uma rotina de recuperação de erro;  
}
```

**FIGURA 4.36** Algoritmo de análise LR.

#### 4. Programa fonte a ser lido

O analisador sintático deverá ler o programa fonte a ser disponibilizado em FONTE.ALG e imprimir na tela todas as produções reduzidas. O FONTE.ALG deverá ter o conteúdo apresentado na Figura 3.

Figura 3 – Programa fonte a ser lido pelo analisador .

```
inicio  
  varinicio  
    A literal;  
    B inteiro;  
    D inteiro;  
    C real;  
  varfim;  
  
  escreva "Digite B";  
  leia B;  
  escreva "Digite A:";  
  leia A;  
  se(B>2)  
  entao  
    se(B<=4)  
    entao  
      escreva "B esta entre 2 e 4";  
    fimse  
  fimse  
  
  B<-B+1;  
  B<-B+2;
```

```

B<-B+3;
D<-B;
C<-5.0;

escreva "\nB=\n";
escreva D;
escreva "\n";
escreva C;
escreva "\n";
escreva A;
fim

```

## 5. Finalização ( sobre conclusão dos trabalhos T1, T2 e T3)

Ao final de todos os três trabalhos práticos da disciplina, aplicaremos as técnicas aprendidas em sala e desenvolveremos um pequeno compilador que utilizando dos tokens reconhecidos (Trabalho 1), das sentenças aceitas pela linguagem (Trabalho 2) e da tradução dirigida por sintaxe (Trabalho 3) a serem implementadas compilará o programa em linguagem ALG: FONTE.ALG (Figura 3) em PROGRAMA.C da Figura 4.

Figura 4 – Programa objeto a ser gerado pelo compilador ao final de todos os trabalhos da disciplina (PROGRAMA.C).

```

#include<stdio.h>

typedef char literal[256];
void main(void)
{
    /*----Variaveis temporarias----*/
    int T0;
    int T1;
    int T2;
    int T3;
    int T4;
    /*-----*/
    literal A;
    int B;
    int D;
    double C;

    printf("Digite B");
    scanf("%d",&B);
    printf("Digite A:");
    scanf("%s",A);
    T0=B>2;
    if(T0)
    {
        T1=B<=4;
        if(T1)
        {
            printf("B esta entre 2 e 4");
        }
    }
    T2=B+1;
    B=T2;
    T3=B+2;
    B=T3;
    T4=B+3;
    B=T4;
    D=B;
    C=5.0;
    printf("\nB=\n");
    printf("%d",D);
    printf("\n");
    printf("%lf",C);
    printf("\n");
    printf("%s",A);
}

```

