**UNIVERSIDADE FEDERAL DE GOIÁS**
**ENGENHARIA DE COMPUTAÇÃO**
**COMPILADORES**

# TRABALHO 3 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, novembro de 2017.

# 1. INTRODUÇÃO

O trabalho apresentado nesse documento busca a realização de atividade prática da disciplina Compiladores e compõe a nota T3 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 T3

## *2.1. Regras DO TRABALHO*

1. O trabalho será desenvolvido pelos mesmos grupos anteriores ( máximo duas pessoas);
2. O trabalho (códigos fonte e executáveis) será entregue via moodle por um membro do grupo no até o dia: 04/12/2017 às 13:00hs (para cada dia de atraso serão descontados 0,3 por dia até o dia de apresentação).
3. As apresentações serão realizadas nos dias: 04/12/2017 (segunda-feira) e 07/12/2017 (quinta-feira), conforme horário marcado pelo professor**. A presença nessas datas-aulas está condicionada à sua apresentação.**
4. **Caso seja feito em duplas, o professor escolherá quem do grupo explicará o trabalho realizado sendo que a nota obtida será a mesma para os membros da equipe.**
5. A equipe poderá escolher a linguagem de programação que será utilizada para desenvolver o trabalho. Portanto, é de responsabilidade dos alunos que no dia da apresentação todo o aparato para execução do trabalho esteja diponível.
6. A evolução do trabalho será acompanhada pela professora em aula no laboratório.
7. O trabalho T3 é um componente do compilador que já está sendo desenvolvido através dos trabalhos T1 e T2, ou seja, os trabalhos T1, T2 e T3 se complementam. Dessa forma, **não serão avaliados os trabalhos que não estejam com os analisadores léxico e sintático funcionando ( conforme as descrições de T1 e T2) e o T3 complementado os demais para a conclusão do sistema.**
8. O programa a ser desenvolvido **deverá estar interligado aos demais** (T1 e T2) **e funcionando** (rodando/executando). **Não serão avaliados trabalhos que não estejam funcionando.**

## *2.2. Atividade a ser desenvolvida*

Desenvolver um programa computacional na linguagem escolhida que complemente os trabalhos T1 e T2 implementandos:

1. Que faça a análise semântica e geração de código para o programa objeto;
2. A análise semântica deve acontecer em conjunto com as reduções das produções da gramática (analisador sintático);
3. Todos os símbolos terminais (token) e não terminais devem ter no **MÍNIMO** os seguintes atributos:
    a. lexema: é a representação textual identificada no texto fonte;
    b. tipo: representa o tipo de dados ou operadores para algumas classes de tokens.
        i. Para operadores matemáticos (opm) o tipo pode conter o caracter '+', ou '-', ou '*' etc;
        ii. Para o token *int* o tipo pode conter "inteiro" ou "int" etc;
        iii. Para operador relacional (opr), >, <, ==, <=,>=
        iv. Para atribuição (rcb), =
    c. classe do token: se ele é palavra reservada, identificador, constante literal, constante númérica etc;

4. O conteúdo dos atributos podem ser atribuídos durante a análise léxica ou durante a análise semântica(aplicação de regras semânticas);
5. A função Imprimir gera texto para ser impresso no arquivo PROGRAMA.C, que é o arquivo objeto gerado.
6. O símbolo "-" em ações semânticas indica que não há regra a ser aplicada durante a redução da análise sintática.
7. Para cada erro semântico encontrado através da aplicação das regras semânticas, informar a linha do texto fonte onde se encontra e imprimir "Erro identificado na análise semântica".
8. A variável Tx é uma variável gerada automaticamente para a tradução das operações aritméticas e relacionais do programa fonte para o objeto. É necessário desenvolver um contador que inicie de 0 até a quantidade de variáveis necessárias a tradução. Dessa forma, o código objeto possuirá as variáveis T0, T1, T2 ,..., necessárias a execução dos comandos. A cada variável gerada, é necessário sua declaração no programa obj. Então, deve-se criar um mecanismo para realizar essa criação de variáveis com geração de números sequenciais e sua declaração no programa objeto.

Tabela 1 – Definições das regras semânticas para a linguagem ALG.

| | **Regras Sintáticas** | **Regras semânticas** |
|---|---|---|
| **1** | P'→ P | - |
| **2** | P→ **inicio** V A | - |
| **3** | V→**varinicio** LV | - |
| **4** | LV→D LV | - |
| **5** | LV→**varfim;** | Imprimir três linhas brancas no arquivo objeto; |
| **6** | D→**id** TIPO**;** | id.tipo ← TIPO.tipo<br>Imprimir ( **TIPO.tipo id.lexema ;** ) |
| **7** | TIPO→**inteiro** | TIPO.tipo ← inteiro.tipo |
| **8** | TIPO→**real** | TIPO.tipo ← real.tipo |
| **9** | TIPO→**literal** | TIPO.tipo ← literal.tipo |
| **10** | A→ES A | - |
| **11** | ES→**leia id;** | Verificar se o campo *tipo* do identificador está preenchido indicando a declaração do identificador (execução da regra semântica de número 6).<br>Se sim, então:<br>Se id.tipo = literal Imprimir ( **scanf("%s", id.lexema);** )<br>Se id.tipo = inteiro Imprimir ( **scanf("%d", &id.lexema);** )<br>Se id.tipo = real Imprimir ( **scanf("%lf", &id.lexema);** )<br>Caso Contrário:<br>Emitir na tela "Erro: Variável não declarada". |
| **12** | ES→**escreva** ARG**;** | Gerar código para o comando escreva no arquivo objeto.<br>Imprimir ( **printf("ARG.lexema");** ) |
| **13** | ARG→**literal** | ARG.atributos ← literal.atributos (Copiar todos os atributos de literal para os atributos de ARG). |
| **14** | ARG→**num** | ARG.atributos ← num.atributos (Copiar todos os atributos de literal para os atributos de ARG). |
| **15** | ARG→**id** | Verificar se o identificador foi declarado (execução da regra semântica de número 6).<br>Se sim, então:<br>ARG.atributos ← id.atributos (copia todos os atributos de id para os de ARG). |

| | | |
|---|---|---|
| | | Caso Contrário:<br>Emitir na tela "Erro: Variável não declarada". |
| **16** | A→CMD A | - |
| **17** | CMD →**id rcb** LD**;** | Verificar se **id** foi declarado (execução da regra semântica de número 6).<br>Se sim, então:<br>\| Realizar verificação do *tipo* entre os operandos *id* e *LD* (ou seja, se<br>\| ambos são do mesmo tipo*)*.<br>\| Se sim, então:<br>\| \| Imprimir (**id.lexema rcb.tipo LD.lexema**) no arquivo objeto.<br>\| Caso contrário emitir:"Erro: Tipos diferentes para atribuição".<br>Caso contrário emitir "Erro: Variável não declarada". |
| **18** | LD→OPRD **opm** OPRD | Verificar se tipo dos operandos são equivalentes e diferentes de *literal.*<br>Se sim, então:<br>Gerar uma variável numérica temporária Tx, em que x é um número gerado sequencialmente.<br>LD.lexema ← Tx<br>Imprimir (**Tx = OPRD.lexema opm.tipo OPRD.lexema**) no arquivo objeto.<br>Caso contrário emitir "Erro: Operandos com tipos incompatíveis". |
| **19** | LD→OPRD | LD.atributos ← OPRD.atributos (Copiar todos os atributos de OPRD para os atributos de LD). |
| **20** | OPRD→**id** | Verificar se o identificador está declarado.<br>Se sim, então:<br>OPRD.atributos ← id.atributos<br>Caso contrário emitir "Erro: Variável não declarada". |
| **21** | OPRD→**num** | OPRD.atributos ← num.atributos (Copiar todos os atributos de num para os atributos de OPRD). |
| **22** | A→COND A | - |
| **23** | COND→CABEÇALHO CORPO | Imprimir ( **}** ) no arquivo objeto. |
| **24** | CABEÇALHO→**se** (EXP_R) **então** | Imprimir ( **if (EXP_R.lexema) {** ) no arquivo objeto. |
| **25** | EXP_R→OPRD **opr** OPRD | Verificar se os tipos de dados de OPRD são iguais ou equivalentes para a realização de comparação relacional.<br>Se sim, então:<br>Gerar uma variável booleana temporária Tx, em que x é um número gerado sequencialmente.<br>EXP_R.lexema ← Tx<br>Imprimir (**Tx = OPRD.lexema opr.tipo OPRD.lexema**) no arquivo objeto.<br>Caso contrário emitir "Erro: Operandos com tipos incompatíveis". |
| **26** | CORPO→ES CORPO | - |
| **27** | CORPO→CMD CORPO | - |
| **28** | CORPO→COND CORPO | - |
| **29** | CORPO→**fimse** | - |
| **30** | A→**fim** | - |

# 3. Programa fonte – Entrada do Sistema

O sistema deverá ler o programa fonte a ser disponibilizado em FONTE.ALG e imprimir na tela as reduções realizadas bem como a cópia das ações semânticas realizadas. O FONTE.ALG deverá ter o conteúdo apresentado na Figura 1.

Figura 1 – Programa fonte a ser lido pelo sistema.

```
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
```

# 4. Programa objeto – arquivo de saída do sistema

O sistema Compiler receberá o programa da figura 1 - FONTE.ALG - como fonte e deverá gerar, a partir dos processamentos de análise léxica, sintática e tradução dirigida por sintaxe o PROGRAMA.C da figura 2.

Figura 2 – Programa objeto a ser gerado pelo compilador (PROGRAMA.C).

```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);
}
```