

TRABALHO 3 DE LABORATÓRIO

Estudo de Caso: Implementação analisador semântico –
tradução dirigida por sintaxe – para linguagem MGOL

Esta atividade é um componente para a avaliação e desenvolvimento dos conhecimentos envolvidos na disciplina Compiladores. O valor da 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, junho de 2018.

1. INTRODUÇÃO

O trabalho descrito neste 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 dia determinado. 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 determinados pelo professor.
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 linguagem de programação que será utilizada para desenvolver o trabalho será a mesma adotada nos trabalhos 1 e 2. É de responsabilidade dos alunos que no dia da apresentação todo o aparato para execução do trabalho esteja disponível.
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. Não serão aceitos programas que estejam utilizando geradores de analisadores léxico e sintático.
9. **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 já implementados durante o semestre. O programa deverá realizar análise semântica e tradução para código final através de um esquema S-atribuído. As regras semânticas e geração de código serão realizadas juntamente com a análise sintática.

O compilador funcionará da seguinte forma:

- O analisador sintático aciona o léxico solicitando tokens;
- Durante o processo de análise sintática, após a realização de uma redução, verifica-se a existência de regra semântica associada à sintática, se houver, esta será executada.

Para a realização da análise semântica juntamente com a sintática:

1. A análise semântica deverá acontecer em conjunto com as reduções das produções da gramática (analisador sintático);
2. **Todos os símbolos** terminais (token) e não terminais devem ter no **MÍNIMO** os seguintes atributos:
 - **Lexema:** é a representação textual identificada no texto fonte;
 - **Token:** se ele é palavra reservada, identificador, constante literal, constante numérica etc;
 - **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), =
3. O conteúdo dos atributos dos terminais poderá ser atribuído durante a análise léxica ou durante a análise semântica (aplicação de regras semânticas), dos não terminais na análise semântica.
4. A tabela 1 apresenta as regras semânticas associadas às regras sintáticas. A função **Imprimir** presente em algumas regras semânticas, gera texto para ser impresso no arquivo PROGRAMA.C, que é o arquivo objeto a ser gerado.
5. O símbolo "-" em ações semânticas indica que não há regra a ser aplicada durante a redução da análise sintática.
6. 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**".
7. Algumas regras utilizam a variável **Tx**. Esta é uma variável gerada automaticamente para a tradução das operações aritméticas e relacionais do programa fonte para o objeto (são também chamadas de variáveis temporárias quando geradas em um processo de compilação que possua código intermediário). Para utilizar a variável **Tx**:
 - É necessário desenvolver um contador que inicie de 0 até a quantidade de variáveis adequadas a tradução. Dessa forma, o código objeto possuirá as variáveis T0, T1, T2 ,..., necessárias a execução dos comandos.
 - A cada variável gerada, é necessário realizar sua declaração no programa obj. Para tal, deve ser desenvolvido um mecanismo que realize para a produção dessas variáveis com geração de números sequenciais e sua declaração no programa objeto.
8. No arquivo obj a ser gerado serão necessários ajustes para que a tradução seja completada. O desenvolvedor é responsável por adicionar cabeçalho com bibliotecas e ajustes finos para que o programa gerado (em linguagem C) funcione perfeitamente em um compilador C.
9. **SAÍDA 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.

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

	Regras Sintáticas	Regras semânticas
1	$P' \rightarrow P$	-
2	$P \rightarrow \text{inicio } V A$	-
3	$V \rightarrow \text{var inicio } LV$	-
4	$LV \rightarrow D LV$	-
5	$LV \rightarrow \text{var fim};$	Imprimir três linhas brancas no arquivo objeto;
6	$D \rightarrow \text{id TIPO};$	$\text{id.tipo} \leftarrow \text{TIPO.tipo}$ Imprimir (TIPO.tipo id.lexema ;)
7	$\text{TIPO} \rightarrow \text{inteiro}$	$\text{TIPO.tipo} \leftarrow \text{inteiro.tipo}$
8	$\text{TIPO} \rightarrow \text{real}$	$\text{TIPO.tipo} \leftarrow \text{real.tipo}$
9	$\text{TIPO} \rightarrow \text{literal}$	$\text{TIPO.tipo} \leftarrow \text{literal.tipo}$
10	$A \rightarrow \text{ES } A$	-
11	$\text{ES} \rightarrow \text{leia id};$	Verificar se o campo <i>tipo</i> do identificador está preenchido indicando a declaração do identificador (execução da regra semântica de número 6). Se sim, então: Se $\text{id.tipo} = \text{literal}$ Imprimir (scanf("%s", id.lexema);) Se $\text{id.tipo} = \text{inteiro}$ Imprimir (scanf("%d", &id.lexema);) Se $\text{id.tipo} = \text{real}$ Imprimir (scanf("%lf", &id.lexema);) Caso Contrário: Emitir na tela "Erro: Variável não declarada".
12	$\text{ES} \rightarrow \text{escreva ARG};$	Gerar código para o comando escreva no arquivo objeto. Imprimir (printf("ARG.lexema");)
13	$\text{ARG} \rightarrow \text{literal}$	$\text{ARG.atributos} \leftarrow \text{literal.atributos}$ (Copiar todos os atributos de literal para os atributos de ARG).
14	$\text{ARG} \rightarrow \text{num}$	$\text{ARG.atributos} \leftarrow \text{num.atributos}$ (Copiar todos os atributos de literal para os atributos de ARG).
15	$\text{ARG} \rightarrow \text{id}$	Verificar se o identificador foi declarado (execução da regra semântica de número 6). Se sim, então: $\text{ARG.atributos} \leftarrow \text{id.atributos}$ (copia todos os atributos de id para os de ARG). Caso Contrário: Emitir na tela "Erro: Variável não declarada".
16	$A \rightarrow \text{CMD } A$	-
17	$\text{CMD} \rightarrow \text{id rcb LD};$	Verificar se id foi declarado (execução da regra semântica de número 6). Se sim, então: Realizar verificação do <i>tipo</i> entre os operandos <i>id</i> e <i>LD</i> (ou seja, se ambos são do mesmo tipo). Se sim, então: Imprimir (id.lexema rcb.tipo LD.lexema) no arquivo objeto. Caso contrário emitir: "Erro: Tipos diferentes para atribuição". Caso contrário emitir "Erro: Variável não declarada".
18	$\text{LD} \rightarrow \text{OPRD opm OPRD}$	Verificar se tipo dos operandos são equivalentes e diferentes de <i>literal</i> . Se sim, então: Gerar uma variável numérica temporária Tx, em que x é um número gerado sequencialmente. $\text{LD.lexema} \leftarrow \text{Tx}$ Imprimir (Tx = OPRD.lexema opm.tipo OPRD.lexema) no arquivo objeto. Caso contrário emitir "Erro: Operandos com tipos incompatíveis".

	Regras Sintáticas	Regras semânticas
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. Se sim, então: OPRD.atributos ← id.atributos 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. Se sim, então: Gerar uma variável booleana temporária Tx, em que x é um número gerado sequencialmente. EXP_R.lexema ← Tx Imprimir (Tx = OPRD.lexema opr.tipo OPRD.lexema) no arquivo objeto. 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 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.

```

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

```

Figura 1 – Programa fonte a ser lido pelo sistema.

```

#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);  
}
```

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