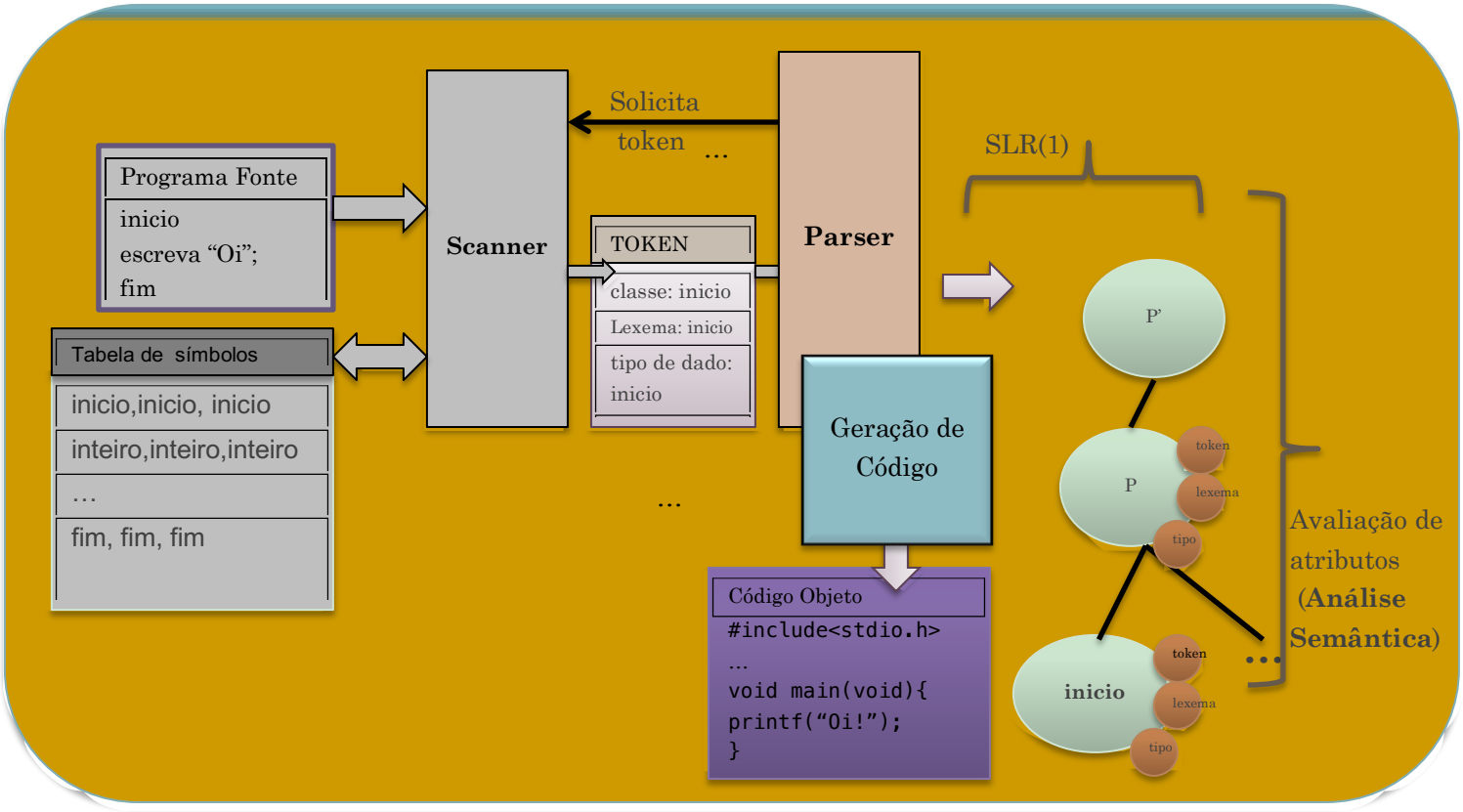
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
Profa. Dra. Deborah Silva Alves Fernandes

COMPILADORES – TRABALHO 3

Analisador Semântico e tradução dirigida pela sintaxe



1 . Descrição

A atividade prática Trabalho 3 (T3) – Analisador Semântico e Tradução Dirigida pela Sintaxe em Compiladores é um componente para a avaliação e desenvolvimento dos conhecimentos desenvolvidos nas disciplinas ofertadas para Ciência da Computação e Engenharia de Computação - Compiladores e Compiladores 1. O valor dessa atividade é 10,0 e compõe a média de aprovação na disciplina conforme definido no plano de curso.

2 – Entregável e Notas

2.1 - Entregar na data determinada pelo professor, EXCLUSIVAMENTE via plataforma Turing, O CÓDIGO desenvolvido para o analisador semântico com tradução dirigida por sintaxe a ser descrito nas seções abaixo. Caso seja realizado em equipe, apenas um componente deverá efetuar a entraga na plataforma. O NOME do código deverá seguir o padrão: ASem-NomeAluno1-NomeAluno2.extensão. **Exemplo**: ASem-DeborahFernandes-FulanoPrado.c .

- Se for entregar um projeto com vários arquivos, junte-os em uma pasta com o nome ASem-NomeAluno1-NomeAluno2 e inclua dentro da pasta um arquivo .txt explicando como abrir e rodar os códigos do programa. Utilize compressão .zip

2.2 – Nota Total: = 10,0

3 – O que fazer?

O programa a ser desenvolvido deverá estar de acordo com as definições de projeto descritas abaixo e será avaliado pelo professor com relação a cada critério estabelecido. Portanto, leia com atenção.
Desenvolver um programa computacional na linguagem escolhida para o projeto que, acoplado ao T1 e ao T2 (analisador léxico, tabela de símbolos e sintático com tratamento de erros) implemente o compilador que atenda às solicitações descritas abaixo:

Observe o conjunto de regras semânticas que contemplam avaliação de atributos, análise semântica, geração de código e tratamento de erros semânticos da TABELA 1.

3.1 O símbolo **" – "** em ações semânticas indica que não há regra semântica associada.

3.2 ***Imprimir(...)*** indica que deverá ser realizada uma impressão no arquivo *.obj* que neste trabalho será PROGRAMA.C (arquivo objeto a ser gerado pelo compilador desenvolvido).

3.3 ***Emitir mensagem de ERRO semântico***, indicará a impressão na saída padrão da mensagem do Erro semântico encontrado, seguido da **linha e coluna** do código fonte onde este ocorreu.

3.4 A impressão de linhas brancas (REGRA 5) no .obj indica o local onde as variáveis deverão ser declaradas. Pode ser ajustado pelo programador de acordo com sua necessidade.

3.5 Regras semânticas da forma: terminal ou não-terminal.atributo ← terminal ou não-terminal.atributo indicam a ocorrência de amarração de atributos. Aconselha-se o uso de uma pilha semântica para tal.

3.6 Nas marcações (A), (B), (C), (D) e (E) o aluno deverá analisar as necessidades de regras semânticas associadas às sintáticas e construí-las. Essas podem envolver acesso e/ou atualização da tabela de símbolos, escrita de código no arquivo .obj, emissão de mensagem de erro semântico, avaliação de atributos, etc. Dicas importantes: (1) Observe as demais regras na tabela para que possa ter uma ideia de como compor as novas e se há necessidade de criá-las. (2) Nos slides de Análise Semântica disponíveis na TURING você encontrará anotações sobre a avaliação de esquema L-atribuído em uma análise ascendente (*bottom-up*).

3.7 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. Tais variáveis são chamadas de "variáveis temporárias" e serão utilizadas e 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 a produção dessas variáveis com geração de números sequenciais e sua declaração no programa objeto.

4 – Passos para o desenvolvimento do projeto

4.1 Ler as regras semânticas associadas às sintáticas;

4.2 Ajustar o item 3.6;

4.3 Implementar as regras semânticas **nas reduções** das produções sintáticas do trabalho T2;

4.4 Criar o arquivo .obj que será PROGRAMA.C;

4.5 No arquivo obj PROGRAMA.C a ser gerado, serão necessários ajustes para que a tradução seja completa. *O desenvolvedor é responsável por criar rotinas para adicionar cabeçalho com bibliotecas e ajustes finos para que o programa gerado (em linguagem C) funcione perfeitamente em um compilador C.*

4.6 Saída do Sistema (T3): Impressão das regras sintáticas no *prompt* e caso **não haja erros** na fase de análise, gerar o arquivo PROGRAMA.C. Este será a tradução do FONTE.alg (MGOL) para C confeccionada a partir do método tradução dirigida pela sintaxe.

4.7 Mais algumas observações:

- As regras semânticas podem ser executadas assim que todas as atividades de uma ação de redução do sintático tenham sido realizadas. No momento após a impressão da regra reduzida, realiza-se uma chamada ao semântico que executará a(s) regra(s) semântica(s) associadas à(s) sintática(s) (FIGURA 1, linha 12).
- O sistema continuará identificando erros léxicos e sintáticos. Caso seja encontrado um ou mais erros léxicos, sintáticos e/ou semânticos o programa **continua a análise, mas não poderá gerar código .obj.**

TABELA 1 – Regras sintáticas (T2) e regras semânticas (T3).

	Regra gramatical	Regras Semânticas
1	P' → P	-
2	P→ **inicio** V A	-
3	V→ **varinicio** LV	-
4	LV→ D LV	-
5	LV→ **varfim** pt_v	Imprimir três linhas brancas no arquivo objeto (espaço para futura inserção de declaração de variáveis), pode ser ajustado pelo desenvolvedor.
6	D→ TIPO L pt_v	**(A)** Amarração de atributos, organizar a passagem de valores do atributo TIPO.tipo, para L.TIPO;
7	L→ id vir L	**(B)** Amarração de atributos, organizar a passagem de valores do atributo.
8	L→ id	**(C)** Ajustar o preenchimento de id.tipo na tabela de símbolos: **Impressão do id no .obj**
9	TIPO→ **inteiro**	TIPO.tipo ← inteiro.tipo Imprimir (**TIPO.tipo**);
10	TIPO→ **real**	TIPO.tipo ← real.tipo Imprimir (**TIPO.tipo**);

11	TIPO→ **literal**	TIPO.tipo ← literal.tipo Imprimir (**TIPO.tipo**);
12	A→ ES A	-
13	ES→ **leia** id pt_v	Verificar se o campo *tipo* do identificador está preenchido indicando a declaração do identificador (execução da regra semântica de número 6). Se sim, então: Se id.tipo = literal Imprimir (**scanf("%s", id.lexema);**) Se id.tipo = inteiro Imprimir (**scanf("%d", &id.lexema);**) Se id.tipo = real Imprimir (**scanf("%lf", &id.lexema);**) Caso Contrário: Emitir na tela "Erro: Variável não declarada", linha e coluna onde ocorreu o erro no fonte.
14	ES→ **escreva** ARG pt_v	Gerar código para o comando escreva no arquivo objeto. Imprimir (**printf("ARG.lexema");**)
15	ARG→ lit	ARG.atributos ← literal.atributos (Copiar todos os atributos de literal para os atributos de ARG).
16	ARG→ num	ARG.atributos ← num.atributos (Copiar todos os atributos de literal para os atributos de ARG).
17	ARG→ id	Verificar se o identificador foi declarado (execução da regra semântica de número 6). **Se sim, então:** ARG.atributos ← id.atributos (copia todos os atributos de id para os de ARG). **Caso Contrário:** Emitir na tela "Erro: Variável não declarada" , linha e coluna onde ocorreu o erro no fonte.
18	A→ CMD A	-
19	CMD→ id rcb LD pt_v	Verificar se **id** foi declarado (execução da regra semântica de número 6). Se sim, então: | Realizar verificação do *tipo* entre os operandos *id* e *LD* (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", | linha e coluna onde ocorreu o erro no fonte. Caso contrário emitir "Erro: Variável não declarada" ", linha e coluna onde ocorreu o erro no fonte.
20	LD→ OPRD opm OPRD	Verificar se tipo dos operandos de de LD são equivalentes e diferentes de *literal.* **Se sim, então:** Gerar uma variável numérica temporária Tx, em que x é um número gerado sequencialmente. LD.lexema ← Tx Imprimir (**Tx = OPRD.lexema opm.tipo OPRD.lexema**) no arquivo objeto. **Caso contrário** emitir "Erro: Operandos com tipos incompatíveis" ", linha e coluna onde ocorreu o erro no fonte.

21	LD→ OPRD	LD.atributos ← OPRD.atributos (Copiar todos os atributos de OPRD para os atributos de LD).
22	OPRD→ id	Verificar se o identificador está declarado. **Se sim, então:** OPRD.atributos ← id.atributos **Caso contrário** emitir "Erro: Variável não declarada" ", linha e coluna onde ocorreu o erro no fonte.
23	OPRD→ num	OPRD.atributos ← num.atributos (Copiar todos os atributos de num para os atributos de OPRD).
24	A→ COND A	-
25	COND→ CAB CP	Imprimir (**}**) no arquivo objeto.
26	CAB→ **se** ab_p EXP_R fc_p **então**	Imprimir (**if (EXP_R.lexema) {**) no arquivo objeto.
27	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" ", linha e coluna onde ocorreu o erro no fonte.
28	CP→ ES CP	-
29	CP→ CMD CP	-
30	CP→ COND CP	-
31	CP→ **fimse**	-
32	A→ R A	**(D)** Verificar as necessidades e gerar as regras semânticas e de tradução.
33	R → CABR CPR	**(E)** Verificar as necessidades e gerar as regras semânticas e de tradução.
34	CABR → **repita** ab_p EXP_R fc_p	**(F)** Verificar as necessidades e gerar as regras semânticas e de tradução.
35	CP_R→ ES CP_R	**(G)** Verificar as necessidades e gerar as regras semânticas e de tradução.
36	CP_R→ CMD CP_R	**(H)**Verificar as necessidades e gerar as regras semânticas e de tradução.
37	CP_R→ COND CP_R	**(I)**Verificar as necessidades e gerar as regras semânticas e de tradução.
38	CP_R→ **fimrepita**	**(J)**Verificar a necessidade de atualizar o que será utilizado em EXP_R para teste no repita; verificar a necessidade de gerar outras regras semânticas e de tradução além desta.
39	A→ **fim**	-

Algoritmo para análise sintática

```
(1)   Seja a o primeiro símbolo de w$;
(2)   while {  /*Repita indefinidamente*/
(3)   seja s o estado no topo da pilha;
(4)       if (ACTION [s,a] = shift t ) {
(5)                empilha t na pilha;
(6)                seja a o próximo símbolo da entrada;
(7)       }else if (ACTION [s,a] = reduce A-> β )  {
(8)                  desempilha símbolos | β | da pilha;
(9)                  faça o estado t agora ser o topo da pilha;
(10)                 empilhe GOTO[t,A] na pilha;
(11)                 imprima a produção A-> β ;
(12)                 INVOCAR SEMÂNTICO(A-> β);
(13)      }else if (ACTION [s,a] = accept )  pare; /* a análise terminou*/
(14)      else  invocar uma rotina de recuperação do erro;
```

Figura 1 – Algoritmo de análise sintática ascendente *shift-reduce*.

4 – Resultado final do Projeto

O Parser (FIGURA 2) realizará o processo de análise sintática e invocará o semântico que fará verificações semânticas e geração de código:

- invocando o SCANNER (T1), sempre que necessitar de um novo TOKEN, consultando as tabelas ACTION e GOTO para decidir sobre as produções a serem aplicadas até a raiz da árvore sintática seja alcançada e não haja mais tokens a serem reconhecidos pelo SCANNER;
- Quando houver uma redução, conduzirá uma chamada à rotina que executará as regras semânticas associadas à regra sintática que foi reduzida;
- Se não houver nenhum erro léxico, sintático ou semântico, um PROGRAMA.C será gerado, caso ocorra pelo menos um erro, a fase de análise continua, porém o arquivo .obj não será criado.

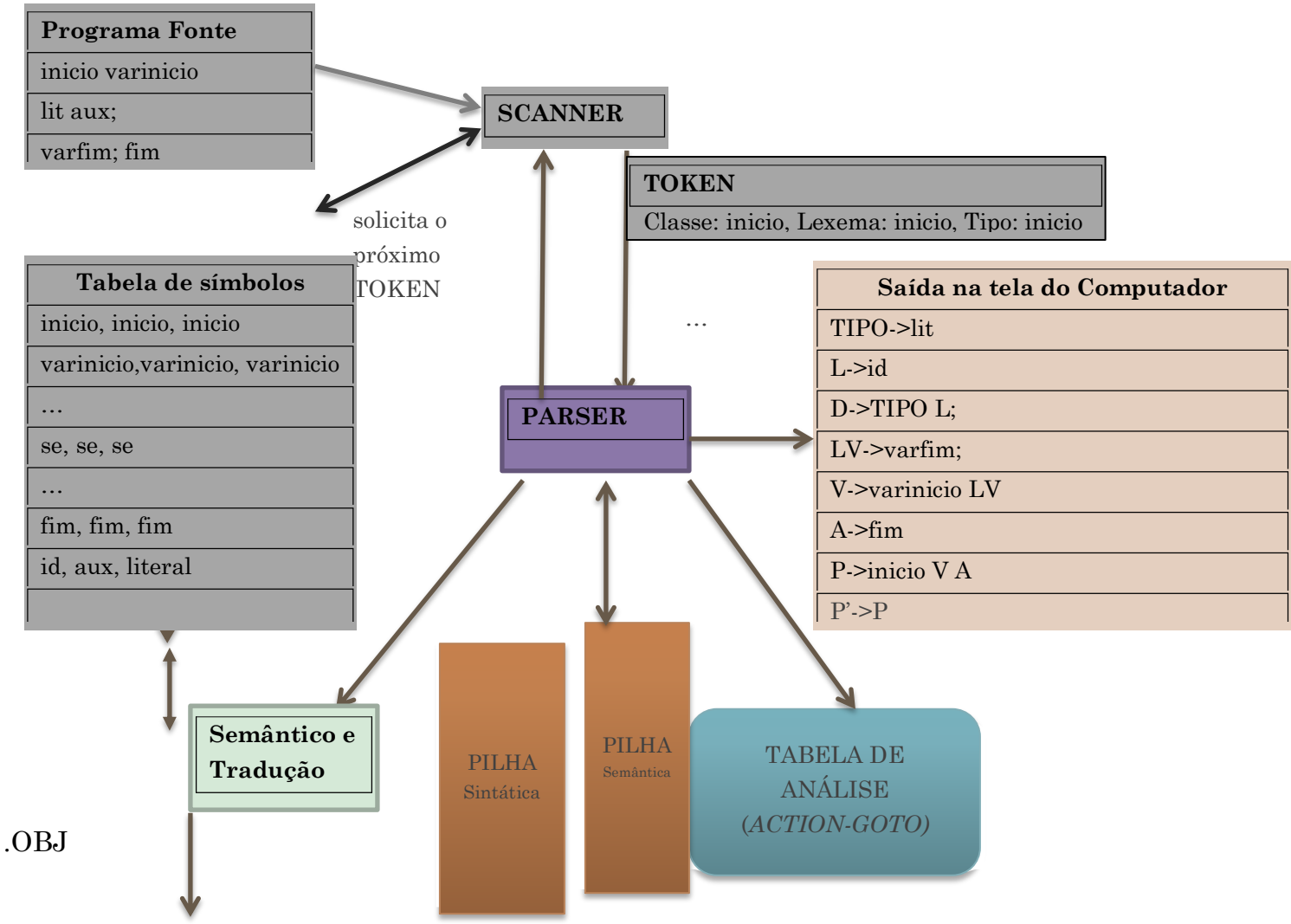


FIGURA 2 – Resultado do PROJETO (T3).

5 – Resultado T1+T2+T3

Ao final de todos os três trabalhos práticos da disciplina, teremos como resultado do estudo de caso um pequeno compilador que converterá o programa fonte em linguagem Mgol, FONTE.ALG - FIGURA 3 (a) em PROGRAMA.C, FIGURA 3(b).

Fonte.ALG

```
inicio
   varinicio
      literal A;
      inteiro B,D;
      real C ;
   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
```

(a)

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);
}
```

(b)

FIGURA 3 – Código fonte (a), código objeto (b).