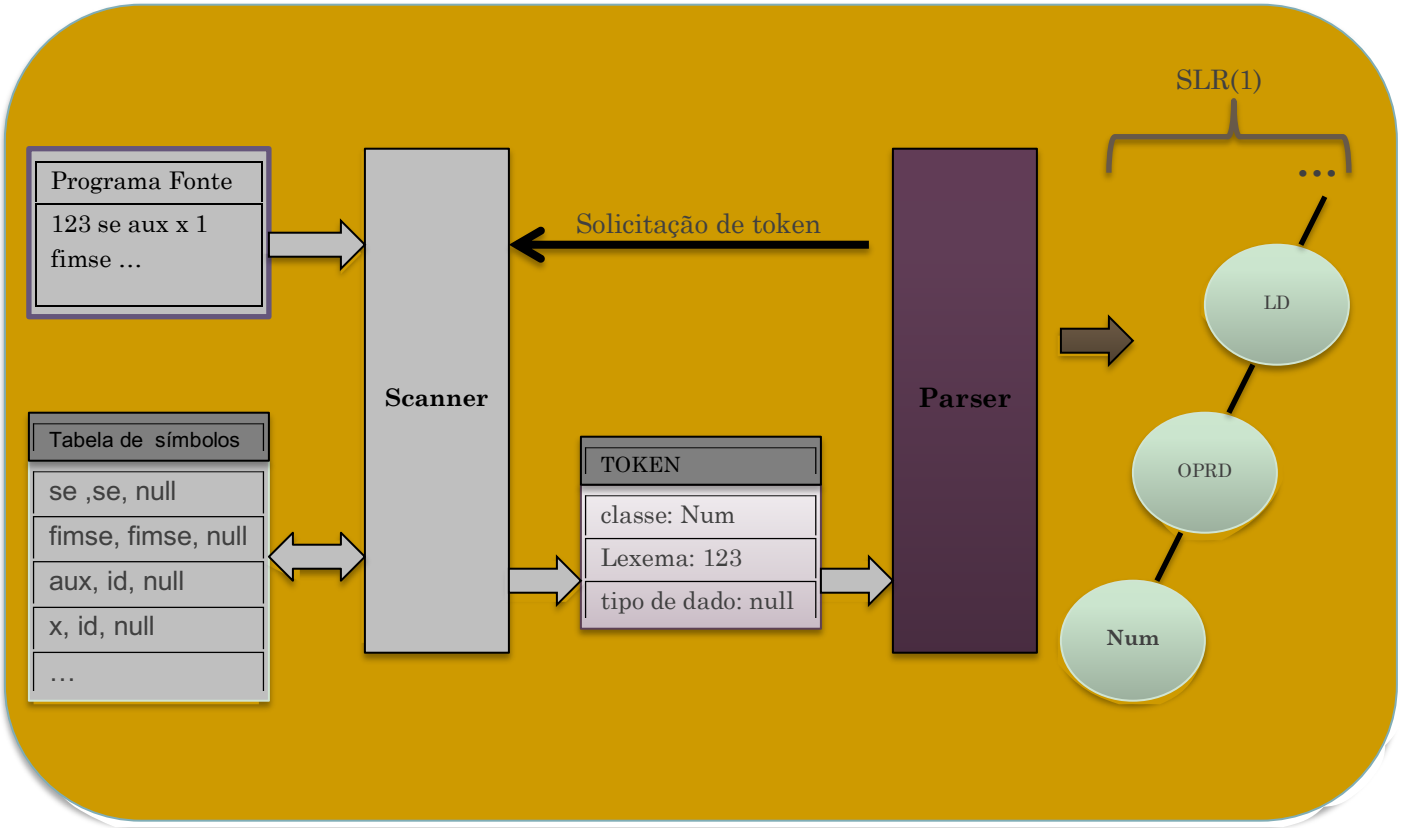
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

COMPILADORES – TRABALHO 2 – T2

Analisador Sintático



Profa. Dra. Deborah Fernandes
2021-2

1 . Descrição

A atividade prática Trabalho 2 (T2) – Analisador Sintático 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 - Entregáveis

2.1 – Entregar, a atividade complementar T2.1 – Conjuntos *First* e *Follow* dos não terminais da gramática da TABELA 1. Essa atividade é INDIVIDUAL e vale 0,5 na nota final do trabalho T2.

2.2 – Entregar, a atividade complementar T2.2 – AUTÔMATO LR(0) com itens da gramática da TABELA 1. Essa atividade é INDIVIDUAL e vale 0,5 na nota final do trabalho T2.

2.3 – Entregar, a atividade complementar T2.3 – pesquisa e escrita sobre tratamento de erros no analisador sintático. Essa atividade é INDIVIDUAL e vale 0,5 na nota final do trabalho T2.

2.4 – Entregar na data determinada pelo professor, EXCLUSIVAMENTE via plataforma Turing, O CÓDIGO desenvolvido para o analisador sintático a ser descrito nas seções abaixo. Caso seja realizado em duplas, apenas um componente deverá entregá-lo na plataforma.

- O NOME do código deverá seguir o padrão: ASin-NomeAluno1-NomeAluno2.extensão. **Exemplo**: ASin-DeborahFernandes-FulanoPrado.c .
- Se for entregar um projeto com vários arquivos, junte-os em uma pasta com .zip.

2.5 – A entrega e arguição oral da implementação terão o valor total de 8,5 pontos.

2.6 – Nota total = Nota T21 + Nota T22 + Nota T23 + Nota T2.

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 em relação a cada critério estabelecido. NÃO SERÁ PERMITIDO o uso de geradores de analisadores léxicos E NEM Regex para solucionar o problema proposto. Leia com atenção.

Desenvolver um programa computacional na linguagem escolhida para o projeto que, acoplado ao T1 (analisador léxico), implemente:

3.1 Um analisador sintático SLR(1) que reconheça as sentenças que podem ser formadas a partir da gramática livre de contexto disponível na TABELA 1.

3.2 Passos de projeto:

a. Construir o autômato LR(0) para a gramática livre de contexto da TABELA 1 (item 2.2);

b. Obter os conjuntos FIRST/FOLLOW dos não terminais da gramática (item 2.1);

c. Construir a tabela de análise sintática **SLR** com as colunas AÇÃO (*shift, reduce, accept e error*) e DESVIOS (*goto*), baseadas nos itens 2.1 e .2.2. À critério do programador, pode ser criada uma ou duas tabelas (uma para ações –ACTION- e outra para os desvios - GOTO).

i. A tabela pode ser construída em um arquivo .csv. O upload pode ser realizado em uma matriz, estrutura de dados, map, à critério do programador, ou poderá ser construída diretamente em uma estrutura no programa.

ii. As lacunas da tabela sintática – coluna AÇÕES (espaços sem ações de redução/empilhamento/aceita) devem ser preenchidas com códigos de erros que deverão indicar o tipo de erro sintático encontrado (se falta operador aritmético, relacional, atribuição, aguarda um id, um se, um "(" , etc.).

3.3 Implementar o algoritmo de análise sintática *shift-reduce* da FIGURA 1 - PARSER.

3.3.1 Uma estrutura de dados do tipo pilha deverá ser criada para apoiar o reconhecimento da sentença (implementação do autômato de pilha). Ela é inicializada com o estado 0 (estado inicial do autômato LR) ao topo. As operações de empilhamento e desempilhamento apontadas no algoritmo serão realizadas sobre esta pilha.

3.3.2 No algoritmo de análise, todas as vezes em que houver um movimento com o apontador de entrada ***a*** o programa deverá chamar a função **"SCANNER"** do trabalho T1 que retornará um TOKEN e seus atributos em ***a***. O campo de ***a*** que será utilizado na análise é a "classe".

3.3.3 Todas as vezes que for acionada uma consulta ACTION ou GOTO, a(s) tabela(s) desenvolvida(s) no item 3.2(c) deverá ser consultada.

3.3.4 Imprimir a produção significa apresentar na saída padrão (tela do computador) a regra que foi reduzida.

3.3.5 Ao invocar uma rotina de **recuperação de ERRO (item 3.4 abaixo),** além desta reestabelecer a análise sintática, deverá ser impressa uma mensagem na saída (tela do computador) informando o tipo do erro sintático encontrado a linha e a coluna onde ocorreu no código de entrada (programa fonte).

3.4 Implementar **uma rotina de tratamento ou recuperação do Erro.**

a. Conforme a pesquisa realizada em T23 (item 2.3), escolher e implementar um modelo ou uma compilação de modelos de tratamento de erros para análise sintática (modo pânico ou outro);

b. Ao encontrar um erro, o PARSER emite mensagem conforme item 3.3.5, reestabelece a análise conforme 3.4.1 e continua o processo para todo o restante do código fonte.

3.5 O **PARSER** invocará:

a. O SCANNER nas linhas (1) e (6) do algoritmo de análise na FIGURA1 (o token retornado pelo analisador léxico deverá ser válido, ou seja, diferente de token erro e comentário);
b. Uma rotina que emitirá o tipo do erro sintático encontrado (mensagem na tela informando que houve erro sintático e qual terminal era aguardado para leitura, linha e coluna onde ocorreu o erro), linha (13) do algoritmo de análise na FIGURA1;
c. Uma rotina que fará uma recuperação do erro (modo pânico ou outro) para continuar a análise sintática até que o final do programa fonte seja alcançado, linha (13) do algoritmo de análise (parser) na FIGURA1.

Algoritmo de análise (PARSER)

```
(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 | β | símbolos da pilha (a quantidade de símbolos de beta);
(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)      } else if (ACTION [s,a] = accept )  pare; /* a análise terminou*/
(13)      else  invocar uma rotina de recuperação do erro;
```

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

TABELA 1 – Produções da gramática livre de contexto para o Trabalho 2.

Identificação	Regra gramatical
1	P' → P
2	P→ **inicio** V A
3	V→ **varincio** LV
4	LV→ D LV
5	LV→ **varfim** pt_v
6	D→ TIPO L pt_v
7	L→ id
8	TIPO→ **inteiro**
9	TIPO→ **real**
10	TIPO→ **literal**
11	A→ ES A
12	ES→ **leia** id pt_v
13	ES→ **escreva** ARG pt_v
14	ARG→ lit
15	ARG→ num
16	ARG→ id
17	A→ CMD A
18	CMD→ id rcb LD pt_v
19	LD→ OPRD opm OPRD
20	LD→ OPRD
21	OPRD→ id
22	OPRD→ num
23	A→ COND A
24	COND→ CAB CP
25	CAB→ **se** ab_p EXP_R fc_p **então**
26	EXP_R→ OPRD opr OPRD
27	CP→ ES CP
28	CP→ CMD CP
29	CP→ COND CP
30	CP→ **fimse**
31	A→ R A
32	R → CABR CPR
33	CABR → **repita** ab_p EXP_R fc_p
34	CPR→ ES CPR
35	CPR→ CMD CPR
36	CPR→ COND CPR
37	CPR→ **fimrepita**
38	A→ **fim**

3 – Resultado final do Parser

O PARSER (FIGURA 2) realizará o processo de análise sintática:

- invocando o SCANNER (T1), sempre que necessitar de um novo TOKEN;
- inserindo e removendo o topo da pilha;
- 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;
- Mostrando na tela os erros cometidos, bem como sua localização do fonte (linha, coluna);
- Reestabelecendo a análise para que o restante do código fonte seja analisado.

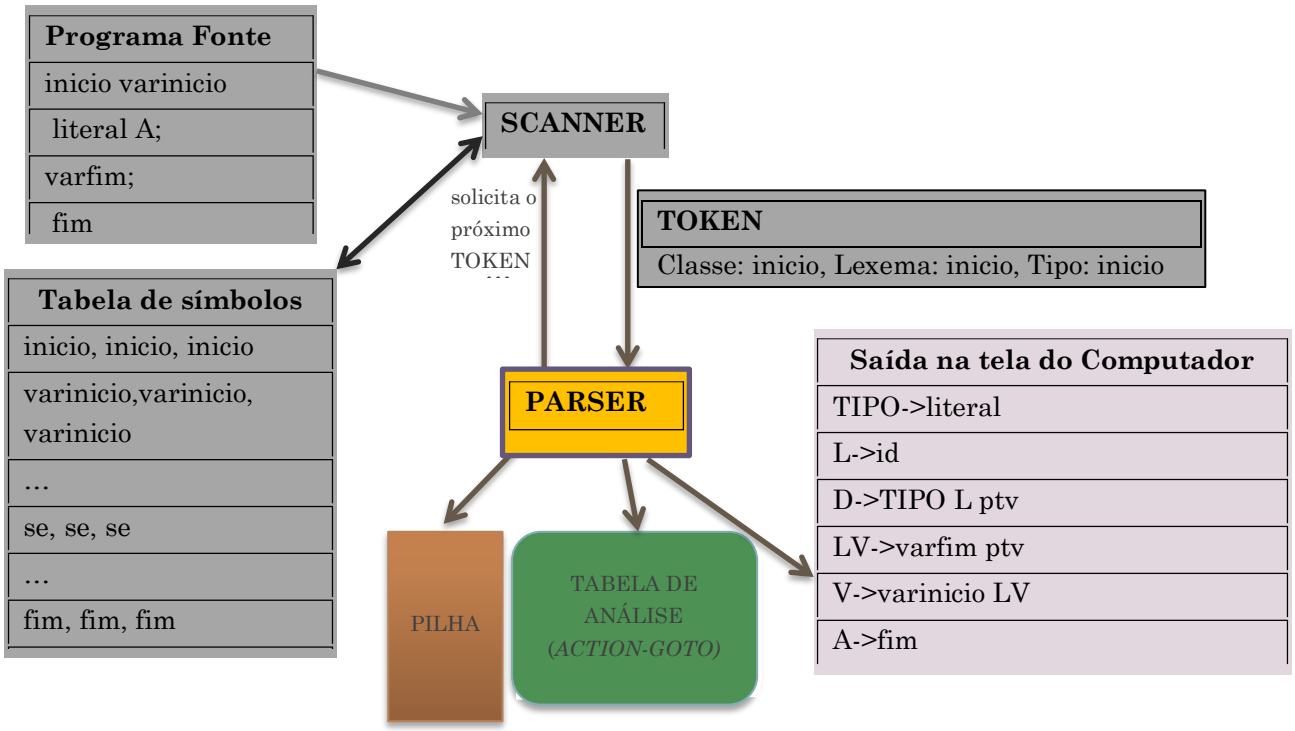


FIGURA 2 – Resultado do PARSER.