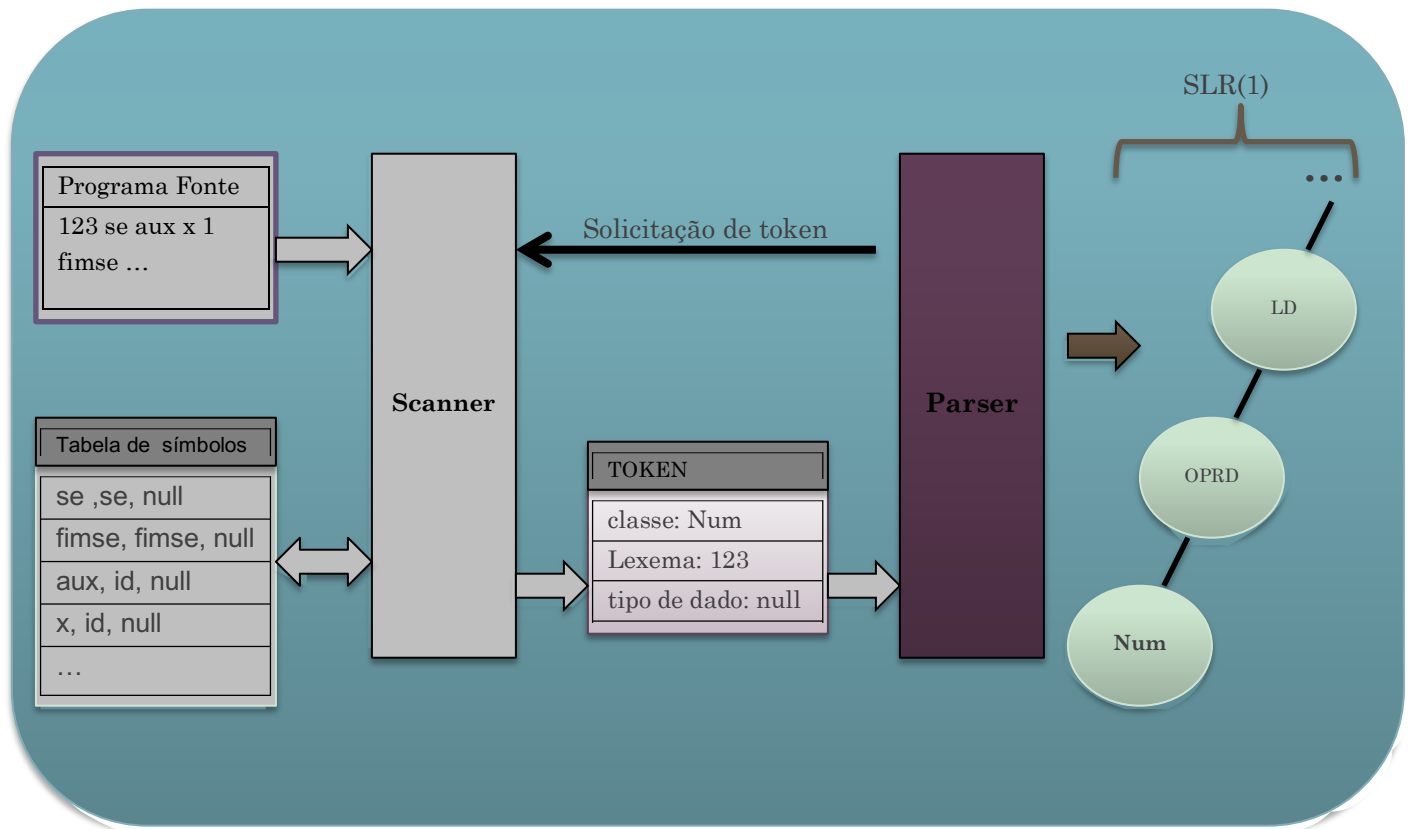


COMPILADORES – TRABALHO 2 – T2

Analizador Sintático



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 – Atividade complementar T2.1 – Entrega e realização INDIVIDUAL – Conjuntos *First* e *Follow* dos não terminais da gramática. Valor: 0,5 na nota final do trabalho T2.

2.2 – Atividade complementar T2.2 – Entrega e realização INDIVIDUAL - AUTÔMATO LR(0) com itens da gramática da TABELA 1. Valor: 1,0 na nota final do trabalho T2.

2.3 – 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 equipe, apenas um componente deverá entregar 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 compactar com **.zip**.

2.4 – A entrega e a avaliação oral da implementação terão o valor total de 8,5 pontos.

2.5 – Nota total T2 = Nota T21 + Nota T22 + 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, sintáticos e 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:

PARTE DA ANÁLISE (vale 5,0 pontos)

- 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:
- Construir o autômato LR(0) para a gramática livre de contexto da TABELA 1 (item 2.2);
 - Obter os conjuntos FIRST/FOLLOW dos não terminais da gramática (item 2.1);
 - 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).
 - A tabela pode ser construída em um arquivo .csv. O upload pode ser realizado em uma matriz ou estrutura de dados à critério do programador.
 - 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.).

Algoritmo de análise	
(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] = <u>shift</u> t) {
(5)	empilha t na pilha;
(6)	seja a o próximo símbolo da entrada;
(7)	}else if (ACTION [s,a] = <u>reduce</u> 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)	}else if (ACTION [s,a] = <u>accept</u>) pare; /* a análise terminou*/
(13)	else chame uma rotina de recuperação do erro ;

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

- 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.

PARTE TRATAMENTO DE ERROS (vale 3,5 pontos)

- 3.3.5 Ao invocar uma rotina de **recuperação de ERRO (item 3.4 abaixo)**, esta deverá reestabelecer a análise sintática e também imprimir na saída (tela do computador) uma mensagem que informe o tipo do erro sintático encontrado (mensagem o mais específica possível) e a linha e coluna do código da entrada (programa fonte) onde ocorreu o erro.
- 3.4 Implementar **uma rotina de tratamento ou recuperação do Erro**.
- a. Realizar uma pesquisa sobre os métodos para recuperação do erro no analisador sintático (modo pânico, correção global, à nível de frase, outros), escolher e implementar pelo menos um modelo ou uma compilação de modelos de tratamento de erros para análise sintática;
 - i. **Essa parte da avaliação vale 2,0 pontos, quem implementar apenas um algoritmo, por exemplo, somente o modo pânico e souber explicar terá pontuação de até 1,0 ponto;**
 - ii. **Se a equipe implementar o pânico em conjunto com outro ou outros dois ou mais métodos em conjunto, poderá ter pontuação de até 2,0 pontos.**
 - b. Ao encontrar um erro, o PARSER emite mensagem conforme item 3.3.5, reestabelece a análise conforme o item 3.4.a. 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;
 - b. Realizará as análises consultando a tabela de análise conforme linhas (4) a (11) do algoritmo da Figura 1;
 - c. 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;
 - d. 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 na FIGURA1.

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

Identificação	Regra gramatical
1	$P' \rightarrow P$
2	$P \rightarrow \text{inicio } V A$
3	$V \rightarrow \text{varinicio } LV$
4	$LV \rightarrow D LV$
5	$LV \rightarrow \text{varfim } pt_v$
6	$D \rightarrow TIPO L pt_v$
7	$L \rightarrow \text{id vir } L$
8	$L \rightarrow \text{id}$
9	$TIPO \rightarrow \text{inteiro}$
10	$TIPO \rightarrow \text{real}$
11	$TIPO \rightarrow \text{literal}$
12	$A \rightarrow ES A$
13	$ES \rightarrow \text{leia id } pt_v$
14	$ES \rightarrow \text{escreva ARG } pt_v$
15	$ARG \rightarrow \text{lit}$
16	$ARG \rightarrow \text{num}$
17	$ARG \rightarrow \text{id}$
18	$A \rightarrow \text{CMD } A$
19	$\text{CMD} \rightarrow \text{id atr LD } pt_v$
20	$\text{LD} \rightarrow \text{OPRD opm OPRD}$
21	$\text{LD} \rightarrow \text{OPRD}$
22	$\text{OPRD} \rightarrow \text{id}$
23	$\text{OPRD} \rightarrow \text{num}$
24	$A \rightarrow \text{COND } A$
25	$\text{COND} \rightarrow \text{CAB CP}$
26	$\text{CAB} \rightarrow \text{se ab_p EXP_R fc_p então}$
27	$\text{EXP_R} \rightarrow \text{OPRD opr OPRD}$
28	$\text{CP} \rightarrow \text{ES CP}$
29	$\text{CP} \rightarrow \text{CMD CP}$
30	$\text{CP} \rightarrow \text{COND CP}$
31	$\text{CP} \rightarrow \text{fimse}$
32	$A \rightarrow \text{RA}$
33	$R \rightarrow \text{CABR CPR}$
34	$\text{CABR} \rightarrow \text{repita ab_p EXP_R fc_p}$
35	$\text{CPR} \rightarrow \text{ES CPR}$
36	$\text{CPR} \rightarrow \text{CMD CPR}$
37	$\text{CPR} \rightarrow \text{COND CPR}$
38	$\text{CPR} \rightarrow \text{fimrepita}$
39	$A \rightarrow \text{fim}$

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.

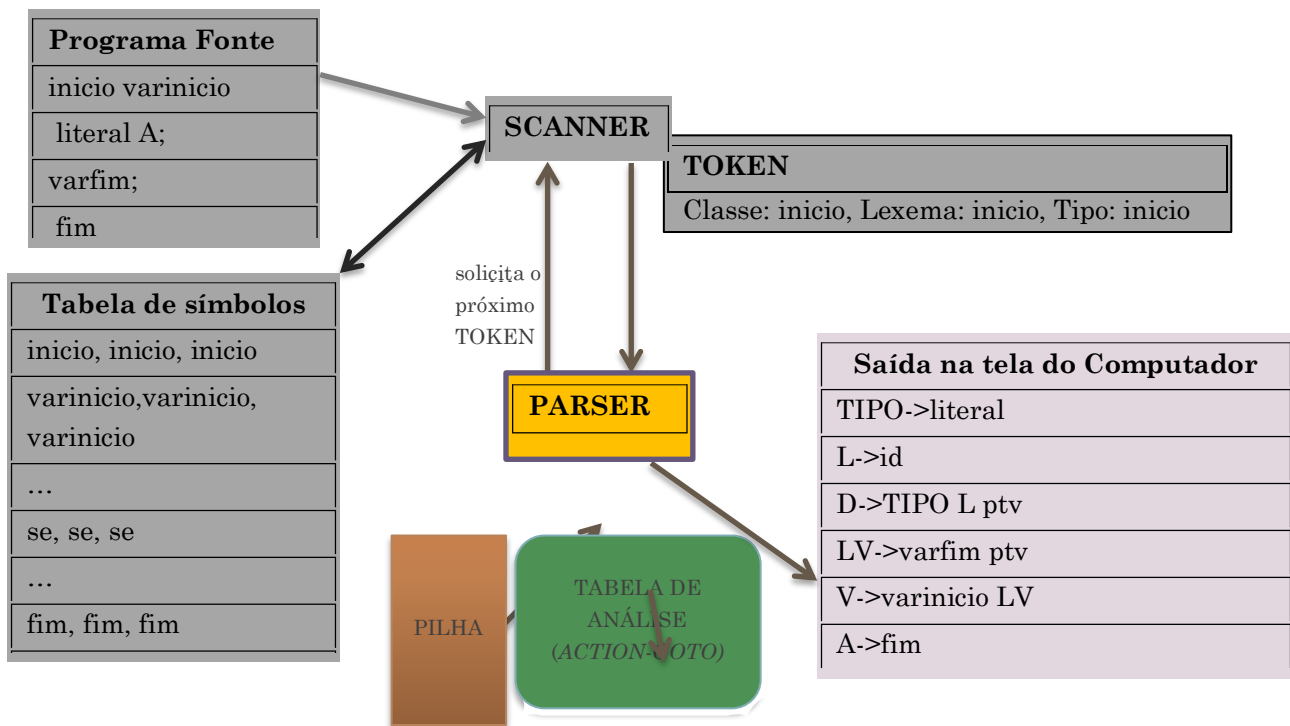


FIGURA 2 – Resultado do PARSER.

4 – Arquivo Sugestão para teste

Programa fonte em linguagem Mgol: FONTE.ALG.

```
inicio
  varinicio
    literal A;
    inteiro B, D, E;
    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
    fimse
  B<-B+1;
  B<-B+2;
  B<-B+3;
  D<-B;
  C<-5.0;
  repita (B<5)
    C<-B+2;
    escreva C;
    B<-B+1;
  fimrepita
  escreva "\nB=\n";
  escreva D;
  escreva "\n";
  escreva C;
  escreva "\n";
  escreva A;
fim
```

FIGURA 3 – Código fonte em linguagem MGOL (Fonte.alg).