UNIVERSIDADE ESTADUAL PAULISTA

FACULDADE DE CIÊNCIAS E TECNOLOGIA

PROJETO COMPILADOR – ANALISADOR SINTÁTICO PARA LALG

RELATÓRIO – PARTE 2

COMPILADORES

PROF. DR. CELSO OLIVETE JÚNIOR

BRUNO SANTOS DE LIMA

LEANDRO UNGARI CAYRES

PRESIDENTE PRUDENTE

NOVEMBRO - 2017

BRUNO SANTOS DE LIMA

LEANDRO UNGARI CAYRES

PROJETO COMPILADOR – ANALISADOR SINTÁTICO PARA LALG

RELATÓRIO

PARTE 2

Relatório do projeto prático - parte II da disciplina de Compiladores, lecionada pelo docente Dr. Celso Olivete Júnior, no curso Bacharelado em Ciência da Computação – Departamento de Matemática e Computação da Faculdade de Ciências e Tecnologia (FCT Unesp – Presidente Prudente).

PRESIDENTE PRUDENTE

NOVEMBRO – 2017

**SUMÁRIO**

[1 INTRODUÇÃO 4](#_Toc499120413)

[2 FUNCIONAMENTO DO ANALISADOR SINTÁTICO CONSTRUIDO 5](#_Toc499120414)

[2.1 Descrição teórica do programa 5](#_Toc499120415)

[2.1.1 JavaCC 5](#_Toc499120416)

[2.2 Descrição da estrutura e organização do programa 6](#_Toc499120417)

[2.4 Processo de recuperação de erros 7](#_Toc499120418)

[2.5 Testes realizados na ferramenta 7](#_Toc499120419)

[2.5.1 Teste 1: Programa correto1 7](#_Toc499120420)

[2.5.2 Teste 2: Programa correto2 8](#_Toc499120421)

[2.5.3 Teste 3: Programa incorreto1 9](#_Toc499120422)

[2.5.4 Teste 4: Programa incorreto2 10](#_Toc499120423)

# **1 INTRODUÇÃO**

Este trabalho da disciplina de Compiladores tem como objetivo a construção de uma aplicação, que realize a análise sintática para programas que utilizam a linguagem de programação LALG.

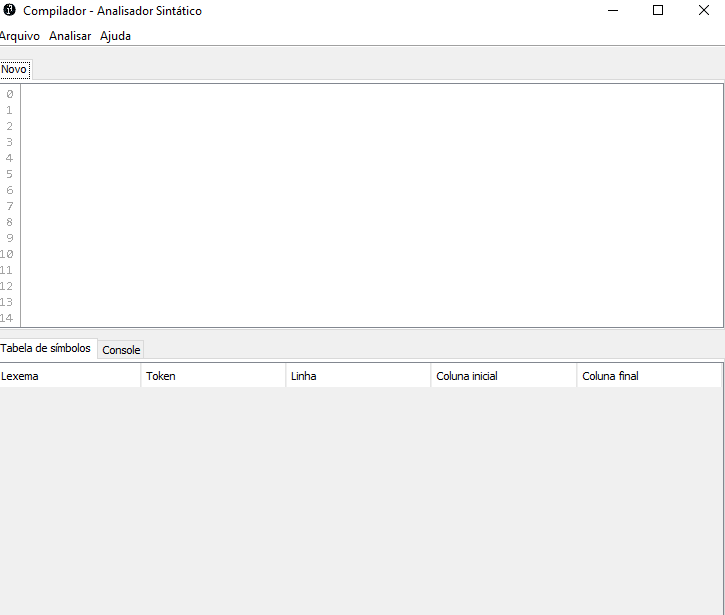
O analisador sintático é o processo que analisa uma dada sequência de entrada e determina sua estrutura gramatical seguindo uma determinada gramatica formal. Basicamente verifica se a sequência de entrada pertence a gramatica de uma determinada linguagem. Em caso negativo o analisador sintático informa um erro ou um conjunto de erros.

Na Seção 2 é relatado o funcionamento do analisador sintático construído contendo a descrição teórica do programa e sua descrição de estrutura e funcionamento. Além disso é apresentado testes de programas e os resultados da análise.

# **2 FUNCIONAMENTO DO ANALISADOR SINTÁTICO CONSTRUIDO**

## **2.1 Descrição teórica do programa**

A aplicação foi desenvolvida na linguagem de programação orientada a objetos Java utilizando a IDE NetBeans 8.2. Para a interface gráfica foi utilizado o pacote Swing do Java, além disso foi utilizado uma Biblioteca de Gerador denominada JavaCC. A aplicação encontrasse no repositório: <https://github.com/brunoslima/ProjetoCompilador>. Na Figura 1 é ilustrado a interface da ferramenta em seu estado atual.



**Figura 1** - Instantâneo da interface atual da ferramenta

## **2.1.1 JavaCC**

O JavaCC é um conjunto de Bibliotecas que podem ser adicionadas a um projeto com o objetivo de facilitar a análise sintática. O JavaCC é um gerador de analisador sintático aberto da linguagem Java. O JavaCC permite a elaboração de um analisador sintático através de uma gramática fornecida em E-BNF. O analisador sintático gerado por ele é descendente, o qual utiliza classes gramaticais LL(1), o que exclui recursividade a esquerda, sendo necessário transformar a gramática orignal da LALG em LL(1). Para maiores informações consulte: <https://javacc.org/>

## **2.2 Descrição da estrutura e organização do programa**

Neste trabalho, o JavaCC foi utilizado da seguinte forma: primeiramente fram importadas as bibliotecas do JavaCC para o projeto, em seguida foi construído um arquivo chamado Grammar.jj que contém métodos para realizar a análise. Esse arquivo quando compilado gera automaticamente um conjunto de classes, com seus respectivos métodos implementados, que guiados pelas descrições do Grammar.jj realizam a análise sintática sobre um conjunto de entrada.

A aplicação contém um conjunto de classes organizadas em pacotes. A classe Arquivo.java é responsável por abrir e ler um arquivo texto que contém um código fonte que sofrerá uma análise sintática.

Com o código-fonte lido através de um arquivo ou simplesmente digitado na área de texto pode-se executar a análise sintática e como resultado é informado se a análise foi concluída com sucesso ou sem contém erros, caso contenha erros são apresentados uma listagem dos mesmos.

As demais classes e pacotes são responsáveis pela construção da interface gráfica da aplicação.

**2.3 Descrição do funcionamento do programa**

O funcionamento do programa ocorre do seguinte modo: dada uma sequência de entrada a mesma é analisada seguindo as diretrizes do arquivo Grammar.jj. Este arquivo contém a gramatica da linguagem LALG, bem como seus terminais e não terminais. Logo no início do arquivo contém alguns métodos responsáveis por realizar a comparação de tokens, recuperar e salvar os possíveis erros gerados.

O arquivo é formado por um conjunto de procedimentos recursivos, cada um desses procedimentos está relacionado com um não terminal, assim a cada token lido da sequência de entrada é verificado se ele se enquadra na gramatica descrita, caso se enquadre este realiza chamadas nos procedimentos da regra na qual ele se enquadra, caso não se enquadre em nenhuma regra ou mesmo quebre alguma regra um método é chamado para armazenar o erro encontrado e assim através do modo pânico a análise sintática continua a acontecer.

## **2.4 Processo de recuperação de erros**

Nem sempre a análise sintática vai se deparar com um programa que não tenha nenhum erro gramatical, sendo assim foi implementado um mecanismo de recuperação de erros para guardar todos os erros encontrados pela análise.

Existe uma classe chamada RecuperacaoErros que é responsável por manter a lista de todos os erros encontrados durante a análise sintática.

Como dito anteriormente a recuperação de erros ocorre utilizando o modo pânico, assim quando um token é lido e identificado que quebra a regra gramatical os demais tokens vindos na sequência são descartados até que uma nova regra possa ser formada, assim continuar a análise sintática.

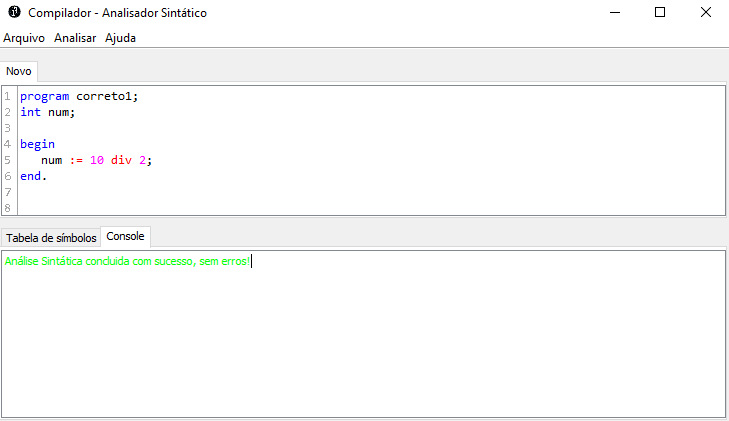
Ao ler um token é observado a regra atual em que seguindo a gramatica e também é espiado os tokens seguintes, utilizando a função *LOOKAHEAD* no arquivo Grammar.jj, caso exista um erro é visto com base na gramatica o token que era esperado e é chamada uma função que armazena os erros na lista de erros da classe RecuperacaoErros e os tokens seguintes são descartados utilizando a ideia do modo pânico.

## **2.5 Testes realizados na ferramenta**

A seguir são descritos alguns testes aplicados sobre a ferramenta. Os testes têm como objetivo mostrar o bom funcionamento da ferramenta e da sua capacidade em realizar a análise sintática com tratamento de erros.

### **2.5.1 Teste 1: Programa correto1**

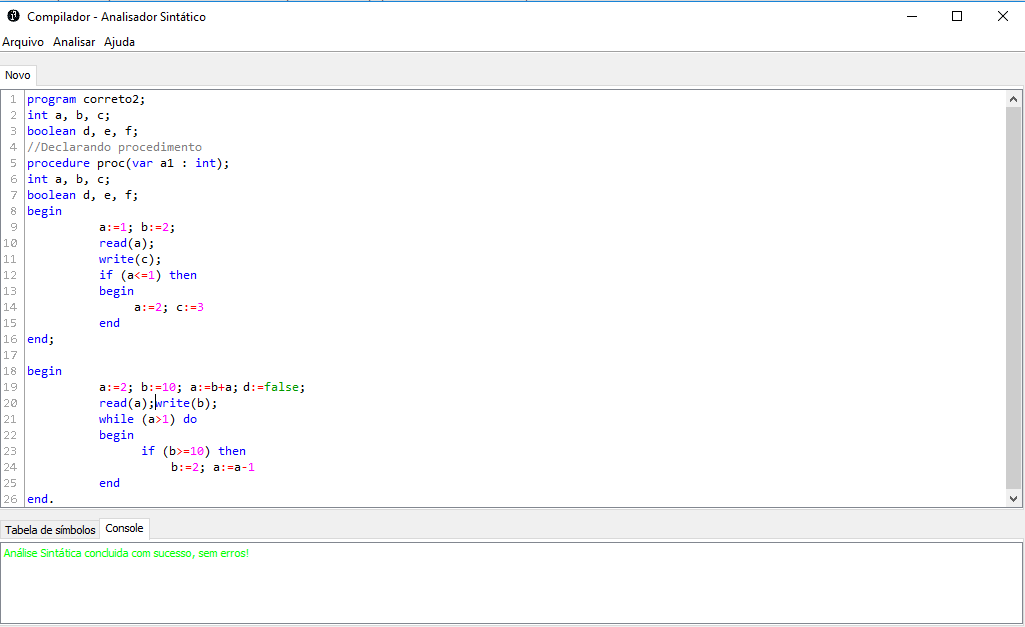
O primeiro teste foi aplicado testando um programa previamente conhecido e correto no qual atribui a divisão do número 10 por 2. Note o resultado da saída como análise sintática concluída e sem erros.



**Figura 2** - Teste 1: Programa correto1

### **2.5.2 Teste 2: Programa correto2**

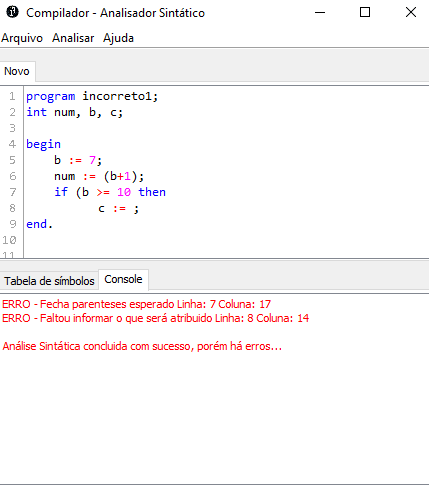
O segundo teste foi aplicado testando um programa previamente conhecido e correto, entretanto mais complexo que o anterior. Note o resultado da saída como análise sintática concluída e sem erros.



**Figura 3** – Teste 2: Programa correto2

### **2.5.3 Teste 3: Programa incorreto1**

O terceiro teste realizado foi com o programa incorreto1 para constatar que a ferramenta consegue identificar erros. No caso do programa incorreto1 ele contém dois erros: a falta do fecha parênteses na linha 7 (comando if) e a ausência de atribuição de um valor na variável c na linha 8.



**Figura 4** – Teste 3: Programa incorreto1

### **2.5.4 Teste 4: Programa incorreto2**

O último teste que esse relatório apresenta consiste em um programa incorreto mais complexo contendo uma maior quantidade de erros. A seguir é descrito o programa incorreto2.

program correto;

int a, b, c;

boolean d, e, f;

procedure proc(var a1 : int);

int a, b, c;

boolean d, e, f;

begin

a:=1;

if (a<1)

a:=12

end;

begin

a:=2;

b:=10;

c:=11;

a:=b+c;

d:=true;

e:=false;

f:=true;

read(a);

write(b);

if (d)

begin

a:=20;

b:=10\*;

c:=a div b

end;

while (a>1)

begin

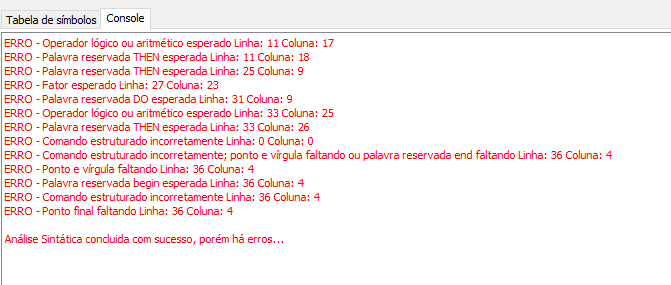
if (b>10)

b:=2;

a:=a-1

end

end.



**Figura 3** – Teste 5 - Erros encontrados no programa incorreto2