UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELOY DE FREITAS ALMEIDA, FABRICIO MEDEIROS TOZO,
GEOVANE DE NOVAIS GOMES, JHONATAN MACHADO LEÃO,
MARCELO AUGUSTO SOARES GOMES

Relatório do desenvolvimento do anualizador léxico para a linguagem de programação *LittlePascal*

ALEGRE
2022

ELOY DE FREITAS ALMEIDA, FABRICIO MEDEIROS TOZO,
GEOVANE DE NOVAIS GOMES, JHONATAN MACHADO LEÃO,
MARCELO AUGUSTO SOARES GOMES

Relatório do desenvolvimento do anualizador léxico para a linguagem de programação *LittlePascal*

Este documento tem o objetivo de relatar as etapas de desenvolvimento do primeiro trabalho da disciplina de compiladores.
Professor: Rodrigo Freitas Silva

ALEGRE
2022

SUMÁRIO

1 INTRODUÇÃO

Objetivo do primeiro trabalho da disciplina de compiladores é a construção de Analisador Léxico para a linguagem de programação *LittlePascal*. Além disso, o desenvolvimento de uma IDE para a codificação e compilação da mesma.

1.1 OBJETIVOS

Os objetivos específicos são:

1. Criar um IDE para o desenvolvimento e compilação dessa linguagem.
2. Listagem dos lexemas que são reservadas pela linguagem.
3. Remoção de espaços em branco, tabulações e comentários.
4. Criar um autômato finito determinístico para o fluxo de entrada dessa linguagem.

1.1.1 Análise Léxica

A primeira fase de um compilador é o Analisador Léxico ou *Scanner* como também é chamado. É nessa fase que são reconhecidas as palavras reservadas, constantes, identificadores e outras palavras que pertencem a linguagem de programação.

A principal tarefa consiste ler um fluxo de caracteres que segue a direção esquerda para a direita e agrupando em TOKENS que são sequencias de caracteres com um significado coletivo. Tokens são palavras válidas, conhecidos também como lexemas. Exemplos são: WHILE, x1, 23. Também são detectados nesta fase erros léxicos e também elimina os comentários e os caracteres neutros como espaços em branco, tabulações ou avanço de linha, facilitando assim, a análise sintática.

1.1.2 Tabela de lexemas

A tabela a seguir demonstra de forma organizada os lexemas e os respectivos tokens, usados durante a execução do compilador para identificar os mesmos.

Lexema	**Token**	**Lexema**	**Token**	**Lexema**	**Token**
PROGRAM	PALAVRA_CHAVE	WHILE	PALAVRA_CHAVE	==	IGUALDADE
BEGIN	PALAVRA_CHAVE	DO	PALAVRA_CHAVE	:=	OPERADOR_ATRIBUIÇÃO
END	PALAVRA_CHAVE	REPEAT	PALAVRA_CHAVE	+	MAIS
CONST	PALAVRA_CHAVE	UNTIL	PALAVRA_CHAVE	-	MENOS
VAR	PALAVRA_CHAVE	BREAK	PALAVRA_CHAVE	*	MULTIPLICAÇÃO
INTEGER	PALAVRA_CHAVE	CONTINUE	PALAVRA_CHAVE	/	DIVISÃO
REAL	PALAVRA_CHAVE	OR	PALAVRA_CHAVE	|	PIPE
CHAR	PALAVRA_CHAVE	AND	PALAVRA_CHAVE	,	VIRGULA
STRING	PALAVRA_CHAVE	END.	PALAVRA_CHAVE	:	DOIS_PONTOS
LITERAL	PALAVRA_CHAVE	NUM	PALAVRA_CHAVE	;	PONTO_VÍRGULA
PROCEDURE	PALAVRA_CHAVE	<>	DIFERENTE	//	DUAS_BARRAS
FUNCTION	PALAVRA_CHAVE	<	MENOR	{	ABRE_CHAVE
IF	PALAVRA_CHAVE	<=	MENOR_IGUAL	}	FECHA_CHAVE
THEN	PALAVRA_CHAVE	>	MAIOR	(*	PARENTESE_ASTERISCO
ELSE	PALAVRA_CHAVE	>=	MAIOR_IGUAL	*)	ASTERISCO_PARENTESE

2 AUTOMATO DA LINGUAGEM

Para o reconhecimento dos lexemas no programa fonte fornecido pelo usuário, criamos o seguinte Automato para o identificação dos símbolos de cada lexema e verificar se estão de acordo com a linguagem.

Mapa de estados finais:

1. Q1 - IDENTIFICADOR
2. Q3 - STRING
3. Q4 - NÚMERO INTEIRO
4. Q7 - NÚMERO REAL

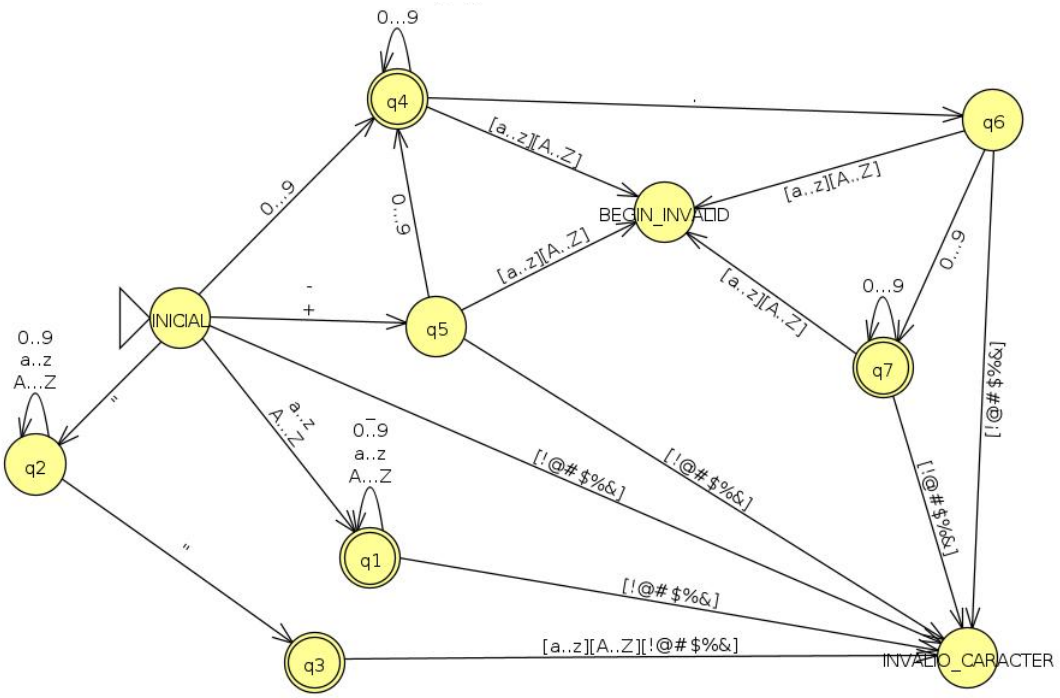

Figura 1 – Automato gramatica da linguagem

3 MODELAGEM

3.0.1 Modelo de domínio

Como escolhemos a linguagem de programação Java para o desenvolvimento do analisador léxico, decidimos identificar e modelar as entidades do sistema.

Seguimos o padrão de projeto Model View Controller (MVC) para organizar o código e ter mais facilidade no desenvolvimento e na manutenção.

Dessa forma, criamos o seguinte modelo de domínio para direcionar o desenvolvimento do sistema.

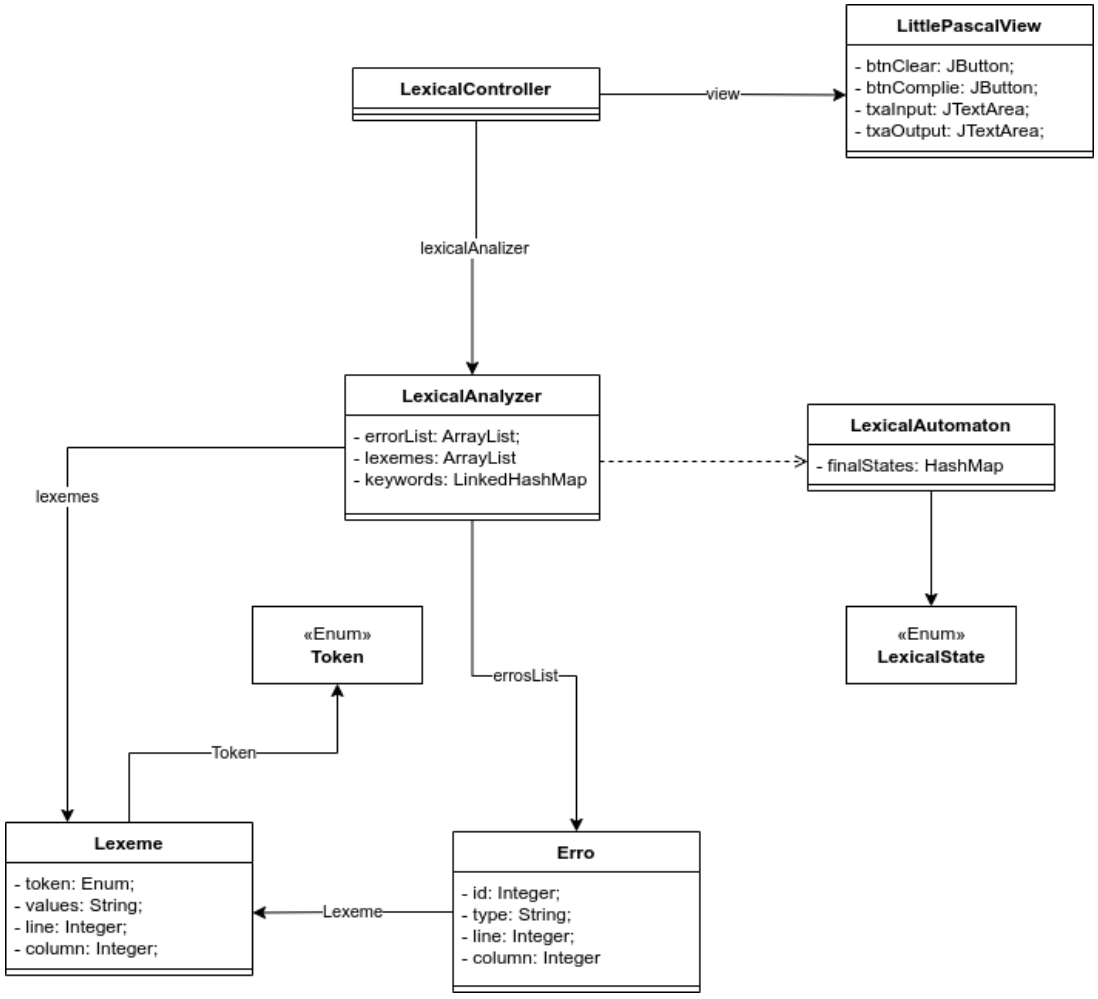

Figura 2 – Modelo de domínio

O *LexicalController* envia o código fonte da *view* (*LittlePascalView*) para o *LexicalAnalyzer* que é responsável pela lógica do programa. O *LexicalAnalizer* é o coração do compilador, é responsável pelo processamento dos lexema e mantém a tabela de símbolos da linguagem e a lista de erros encontrados. Que no final da execução são exibidos na *view*.

3.0.2 Protótipo da interface

1. *Entrada de Dados* e *Saída de Dados*

   Entrada de Dados é a área de texto onde escrevemos o código para passar pela Análise Léxica. Os resultados da análise léxica, como erros de declaração e caracteres inválidos serão apontados na área de *Saída de Dados*.

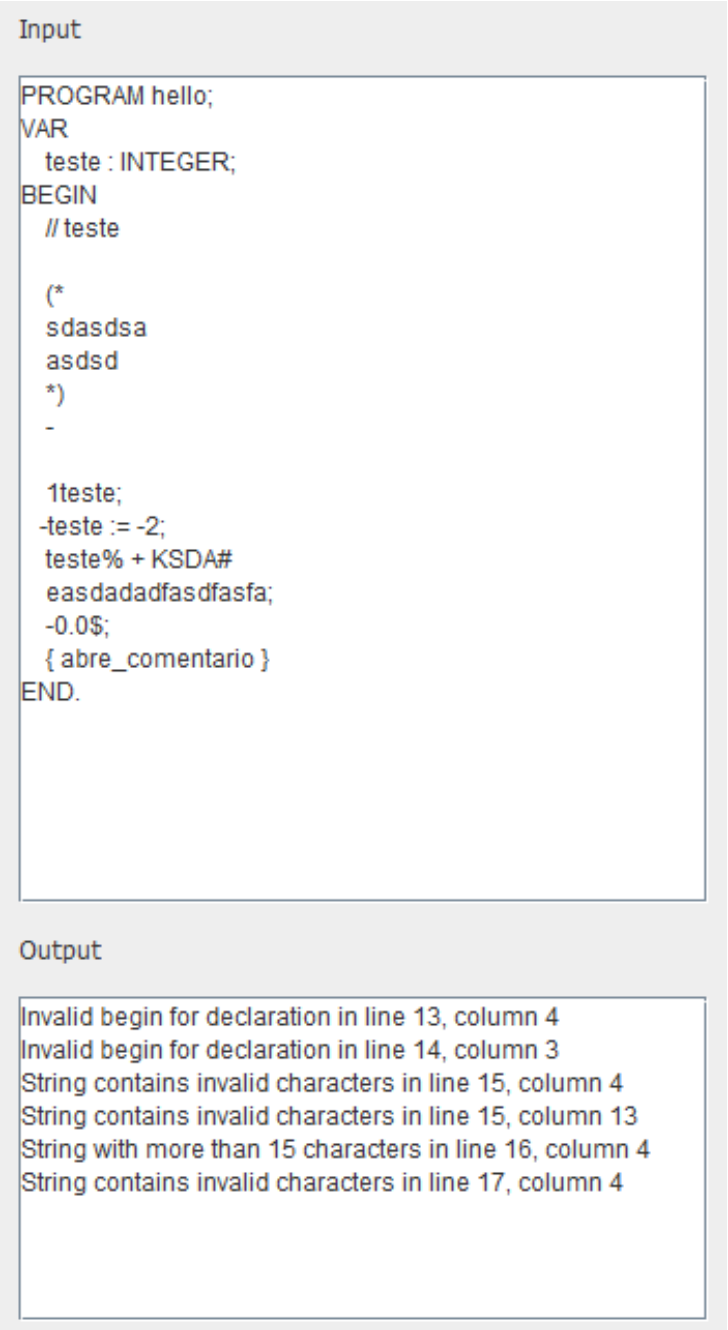

Figura 3 – *Entrada* e *Saída de Dados*

2. Tabela de Símbolos

   Encontramos na janela ao lado a Tabela de mapeamento dos Lexemas, Tokens, Linhas e Colunas. Quando um erro é identificado ele será apresentado em vermelho, apontando sua Linha e Coluna, e qual é o Token do seu erro.

Lexame	Token	Line	Column
PROGRAM	PALAVRA_CHAVE	1	0
hello	IDENTIFICADOR	1	8
;	PONTO_VIRGULA	1	14
VAR	PALAVRA_CHAVE	2	0
teste	IDENTIFICADOR	3	0
:	DOIS_PONTOS	3	6
INTEGER	PALAVRA_CHAVE	3	8
;	PONTO_VIRGULA	3	16
BEGIN	PALAVRA_CHAVE	4	0
-	MENOS	11	0
1teste	BEGIN_INVALID	13	0
;	PONTO_VIRGULA	13	7
-teste	BEGIN_INVALID	14	0
:=	OPERADOR_ATRIBUICAO	14	7
-2	INTEIRO	14	10
;	PONTO_VIRGULA	14	13
teste%	INVALID_CARACTERE	15	0
+	MAIS	15	7
KSDA#	INVALID_CARACTERE	15	9
easdadadfas...	IDENTIFICADOR	16	0
;	PONTO_VIRGULA	16	18
-0.0$	INVALID_CARACTERE	17	0
;	PONTO_VIRGULA	17	6
END.	PALAVRA_CHAVE	19	0

Figura 4 – Tabela de Símbolos

3. *Limpar* e *Compilar*

   Botão *Clear*: Limpa todo código executado e a Tabela de Símbolos.

   Botão *Compile*: Executa o código escrito na Janela de *Entrada de Dados*.

Figura 5 – *Limpar* e *Compilar*

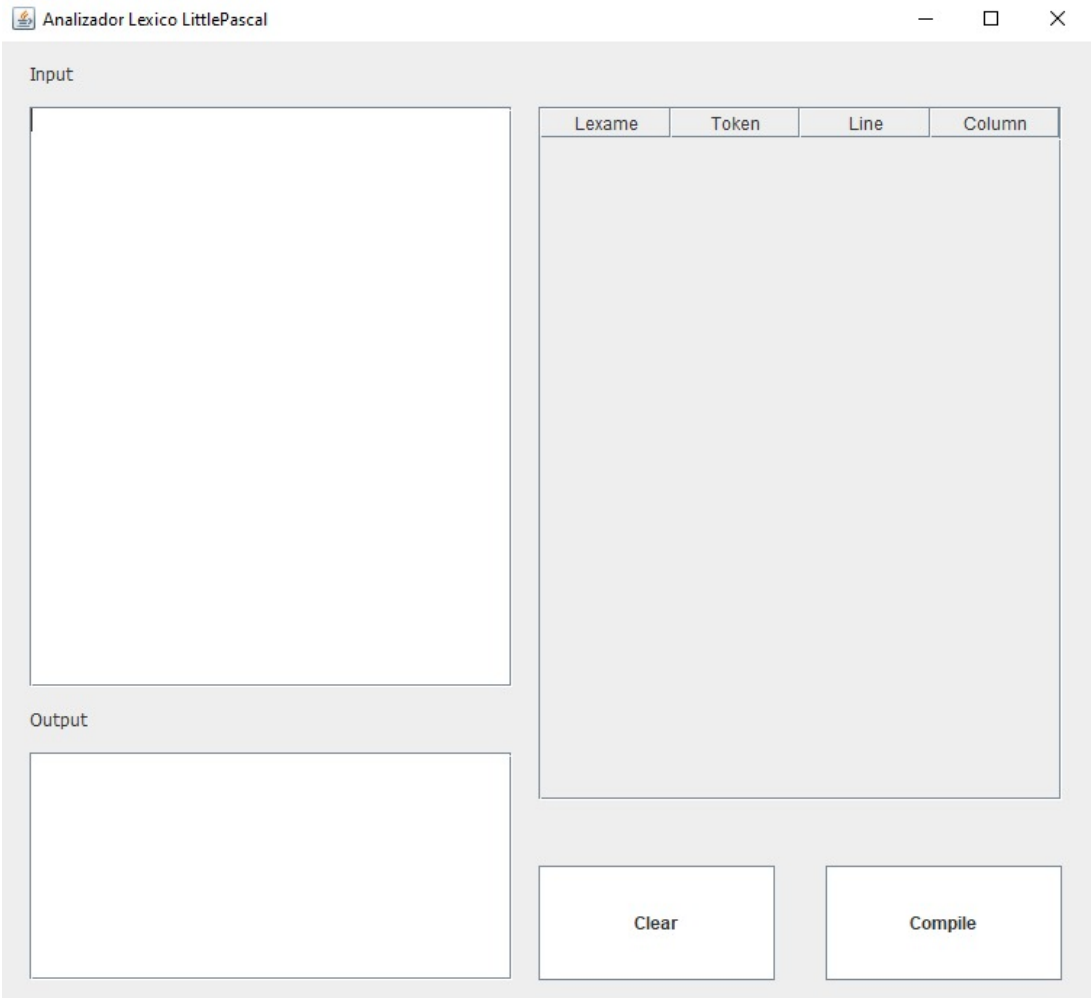

Figura 6 – Layout Limpo

4 EXECUÇÃO DO PROGRAMA

O nosso programa inicia lendo o tipo de código-fonte escrito pelo usuário e fazendo uma análise conforme representada no autômato da linguagem. Em seguida ele retira os comentários e vai para o *LexicalAnalyzer*. Nesse ponto o programa analisa palavra por palavra quebrando o código em linhas e consequentemente criando uma tabela de lexemas das palavras analisadas, e caso nesse processo ele detecte um erro ele joga na lista de erros. Feito isso, ele nos da uma tabela detalhada com os lexemas encontrados, seguidos do seu token, da linha e da coluna na qual ela foi escrita e depois são exibidas na interface principal.

A lista de erros citada, ocorre ao mesmo tempo no qual a palavra está passando pelo *LexicalAnalyzer*, identificando assim a palavra, sendo essa um "erro", ela então é adicionada a lista de erros, tendo também o seu tipo de token descrito, e a linha e coluna na qual foi detectado.

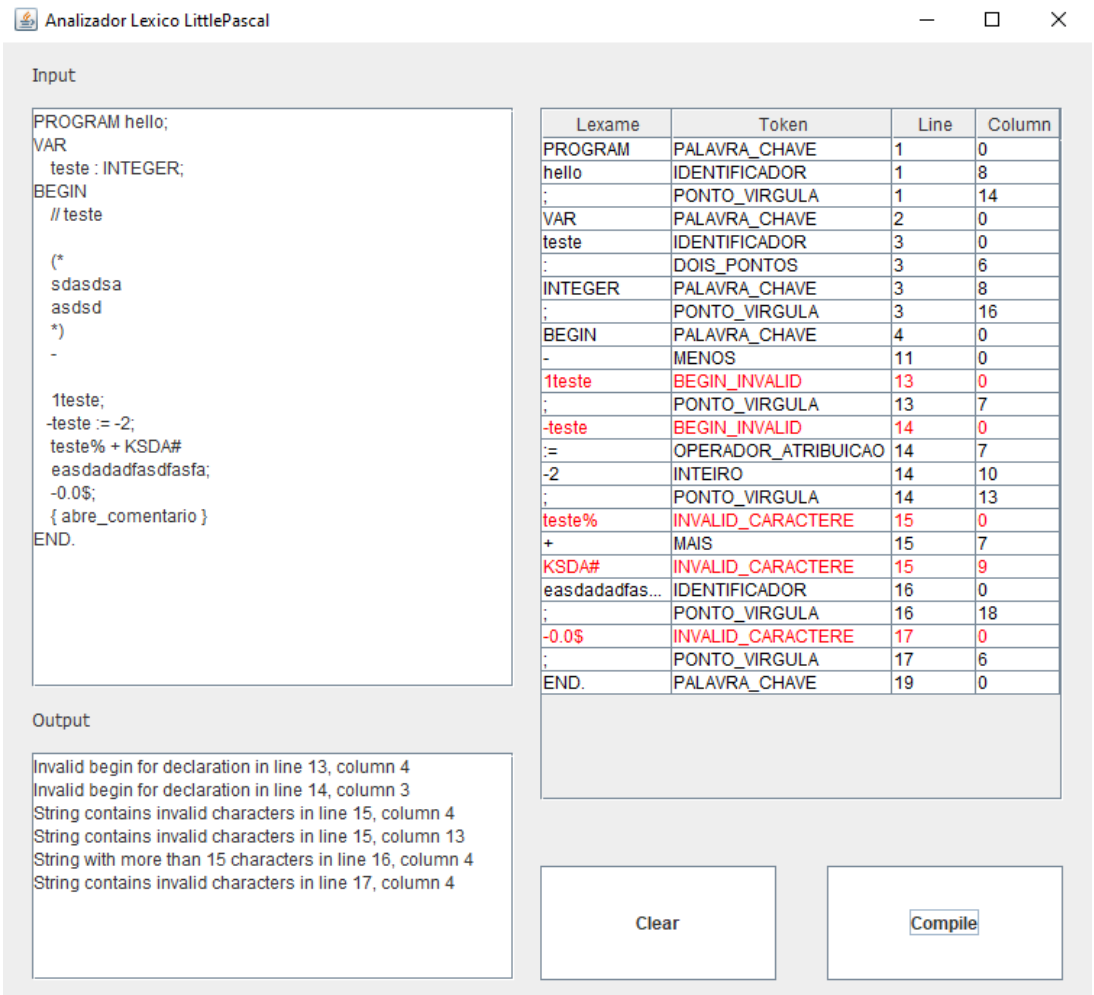

Figura 7 – Layout Compilado

5 METODOLOGIA

Para o desenvolvimento deste trabalho foram utilizados as seguintes ferramentas.

1. Java - openjdk 11.0.12. Para o desenvolvimento da aplicação.
2. Draw.io. Para a criação do modelo de dominio e do prototipo da interface.
3. Jflap7.1. Para a modelagem do automato.
4. Overleaf. Para criação do relatório.