UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE CIÊNCIAS E TECNOLOGIA

PROJETO COMPILADOR – ANALISADOR LÉXICO PARA LALG
RELATÓRIO – PARTE 1

COMPILADORES
PROF. DR. CELSO OLIVETE JÚNIOR

BRUNO SANTOS DE LIMA
LEANDRO UNGARI CAYRES

PRESIDENTE PRUDENTE
SETEMBRO - 2017

BRUNO SANTOS DE LIMA
LEANDRO UNGARI CAYRES

PROJETO COMPILADOR – ANALISADOR LÉXICO PARA LALG

RELATÓRIO

PARTE I

Relatório do projeto prático - parte 1, 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
SETEMBRO – 2017

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Este trabalho da disciplina de Compiladores tem como objetivo a construção de uma aplicação que realize a análise léxica para programas que utilizam a linguagem de programação LALG. Tal objetivo consiste na primeira parte do projeto da disciplina.

Este relatório está organizado em 4 Seções, incluindo essa introdução. A Seção 2 explica brevemente as funções de um analisador léxico para um compilador, na Seção 3 são descritas as expressões regulares utilizadas e por fim a Seção 4 apresenta a aplicação, como ela foi implementada e um guia de utilização, além de mostrar alguns exemplos de teste da aplicação.

2 FUNÇÕES DO ANALISADOR LÉXICO

O analisador léxico é o processo de analisar uma entrada de um código fonte, fazendo uma varredura caractere a caractere, tendo como principal objetivo separar os lexemas, palavras pertencentes a linguagem, classifica-los em Tokens.

Dentre as principais funções do analisador léxico estão:

- Extração dos lexemas do código fonte e classificação em Tokens.
- Eliminação de espaços em branco
- Eliminação de delimitadores e blocos de comentários
- Identificação de símbolos não pertencentes a linguagem

3 CONJUNTO DE EXPRESSÕES REGULARES

Para o desenvolvimento desse trabalho e consequentemente da aplicação foi necessário definir um conjunto de expressões regulares. As expressões regulares são importantes para ajudar na identificação dos Lexemas e Tokens processo fundamental da análise léxica, além de informar qual o alfabeto pertencente a linguagem.

A Tabela 1 apresenta os Tokens e a expressão regular utilizada para identifica-lo.

Token	Expressão Regular
IDENTIFICADOR	([_|a-z|A-Z][a-z|A-Z|0-9]*)
NUMERO_INTEIRO	([0-9]+{1,10})
PALAVRA_RESERVADA_PROGRAM	(program)
PALAVRA_RESERVADA_BEGIN	(begin)
PALAVRA_RESERVADA_END	(end)
PALAVRA_RESERVADA_VAR	(var)
PALAVRA_RESERVADA_PROCEDURE	(procedure)
PALAVRA_RESERVADA_READ	(read)
PALAVRA_RESERVADA_WRITE	(write)
PALAVRA_RESERVADA_INT	(int)
PALAVRA_RESERVADA_BOOLEAN	(boolean)
PALAVRA_RESERVADA_IF	(if)
PALAVRA_RESERVADA_THEN	(then)
PALAVRA_RESERVADA_ELSE	(else)
PALAVRA_RESERVADA_WHILE	(while)

PALAVRA_RESERVADA_DO	(do)
VALOR_LOGICO_TRUE	(true)
VALOR_LOGICO_FALSE	(false)
SIMBOLO_DOIS_PONTOS	(:)
SIMBOLO_PONTO_VIRGULA	(;)
SIMBOLO_VIRGULA	(,)
SIMBOLO_PONTO	(.)
SIMBOLO_IGUAL	(=)
SIMBOLO_DIFERENTE	(<>)
SIMBOLO_MENOR	(<)
SIMBOLO_MENOR_IGUAL	(<=)
SIMBOLO_MAIOR	(>)
SIMBOLO_MAIOR_IGUAL	(>=)
OPERADOR_ATRIBUICAO	(:=)
OPERADOR_ADICAO	(+)
OPERADOR_SUBTRACAO	(-)
OPERADOR_DIVISAO	(div)
OPERADOR_MULTIPLICACAO	(*)
OPERADOR_AND	(and)
OPERADOR_OR	(or)
OPERADOR_NOT	(not)
PARENTESES_ABRE	(( )
PARENTESES_FECHA	( ))
COMENTARIO_LINHA	("\/\/"{COMENTARIO_LINHA_NAO_FECHA}*{COMENTARIO_LINHA_FECHA})
COMENTARIO_LINHA_FECHA	(\n)
COMENTARIO_LINHA_NAO_FECHA	(^\n)
COMENTARIO_MULTI	{COMENTARIO_ABRE}{COMENTARIO_CORPO_MULTI}{COMENTARIO_FECHA}
COMENTARIO_ABRE	(\{)
COMENTARIO_FECHA	(\})
COMENTARIO_NAO_FECHA_MULTI	(^})
COMENTARIO_CORPO_MULTI	({COMENTARIO_NAO_FECHA_MULTI}*)

Tabela 1 - Expressões Regulares utilizadas

Com base nas expressões regulares definidas pode observar que o alfabeto A é:

A = (“a-z”,”A-Z”,”0-9”,”+”,”-“,”*”,”(“,”)”,”.”, ”,”,”,”;”,”:”,”>”,”<”,”=”,”{”,”}”,”/”)

Além disso um número inteiro pode ser representado com até 10 caracteres, mais que isso ele passa a ser outro número inteiro.

4 APLICAÇÃO DESENVOLVIDA

A aplicação foi desenvolvida na linguagem de programação JAVA utilizando a IDE NetBeans 8.2. Sendo assim é uma aplicação totalmente orientada á objetos. Para a interface grafia foi utilizado o pacote Swing do Java e para facilitar a identificação dos Lexemas e Tokens foi utilizado uma Biblioteca de Gerador denominada JFlex. Na Figura 1 é ilustrado a interface inicial da ferramenta.

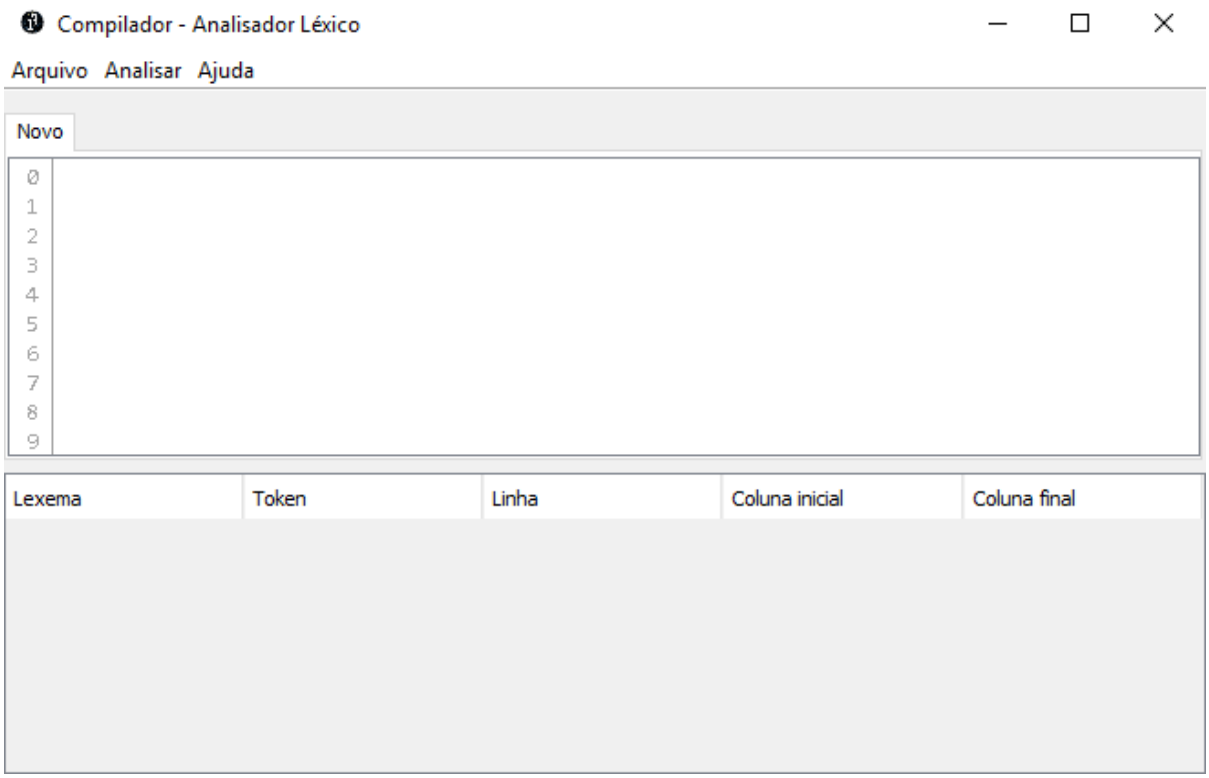


Figura 1 - Instantâneo da interface inicial da ferramenta

4.1 Gerador JFlex

O JFlex é um conjunto de Bibliotecas que podem ser adicionadas a um projeto com o objetivo de facilitar a análise léxica. O JFlex é um gerador de analisador léxico e utiliza de expressões regulares previamente estabelecidas para realizar a analise léxica em uma entrada separando os Lexemas dando sua classificação em Tokens de acordo com as

expressões regulares. O JFlex é baseado em soluções com autômatos finitos determinísticos. Para maiores informações consulte: http://jflex.de/

Neste trabalho o JFlex foi utilizado da seguinte forma: Primeiramente foi importado as bibliotecas do JFlex para o projeto, em seguida foi construído um arquivo chamado Lexer.lex que contém o conjunto de expressões regulares e um método para realizar a análise, esse arquivo Lexer.lex gera uma classe Java automaticamente Lexer.java que contém toda a implementação do gerador com base nas expressões regulares definidas.

4.2 Organização e interação entre as classes

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 léxica.

Com o código fonte lido através de um arquivo ou simplesmente digitado na área de texto pode-se executar a análise léxica, a classe AnalisadorLexico.java é a responsável por acionar o gerador lexer provido pelo JFlex e realizar a análise léxica e por último construir a tabela de símbolos léxicos. Para identificar cada um dos Tokens presentes na linguagem é utilizado uma classe enum Simbolo.java.

Uma classe Item.java representa cada lexema analisado, sendo que esse lexema é inserido como um Item na tabela de símbolos léxicos. Cada item da tabela é formado por um lexema, seu token correspondente, número da linha em que o lexema se encontra, sua coluna inicial e coluna final.

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

4.3 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 léxica e construir a tabela de símbolos léxicos.

4.3.1 Teste 1: Programa correto

São dois testes realizados sobre dois programas corretamente escritos com todos os tokens pertencentes a linguagem e caracteres presentes no alfabeto. As Figuras 2 e 3 mostram os testes de um código simples podendo mostrar assim a tabela completa.

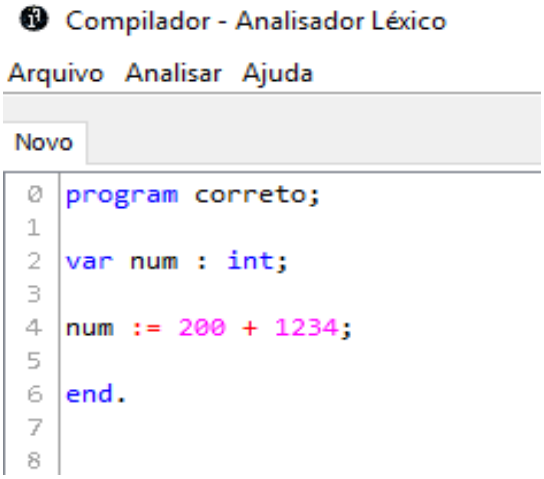


Figura 2 - Exemplo 1: Programa correto1

Lexema	Token	Linha	Coluna inicial	Coluna final
program	PALAVRA_RESERVADA_PROGRAM	0	0	6
correto	IDENTIFICADOR	0	8	14
;	SIMBOLO_PONTO_VIRGULA	0	15	15
var	PALAVRA_RESERVADA_VAR	2	0	2
num	IDENTIFICADOR	2	4	6
:	SIMBOLO_DOIS_PONTOS	2	8	8
int	PALAVRA_RESERVADA_INT	2	10	12
;	SIMBOLO_PONTO_VIRGULA	2	13	13
num	IDENTIFICADOR	4	0	2
:=	OPERADOR_ATRIBUICAO	4	4	5
200	NUMERO_INTEIRO	4	7	9
+	OPERADOR_ADICAO	4	11	11
1234	NUMERO_INTEIRO	4	13	16
;	SIMBOLO_PONTO_VIRGULA	4	17	17
end	PALAVRA_RESERVADA_END	6	0	2
.	SIMBOLO_PONTO	6	3	3

Figura 3 - Exemplo 2: Tabela de símbolos léxicos do programa correto1

As Figuras 4 e 5 mostram um código robusto mostrando apenas parte da tabela.

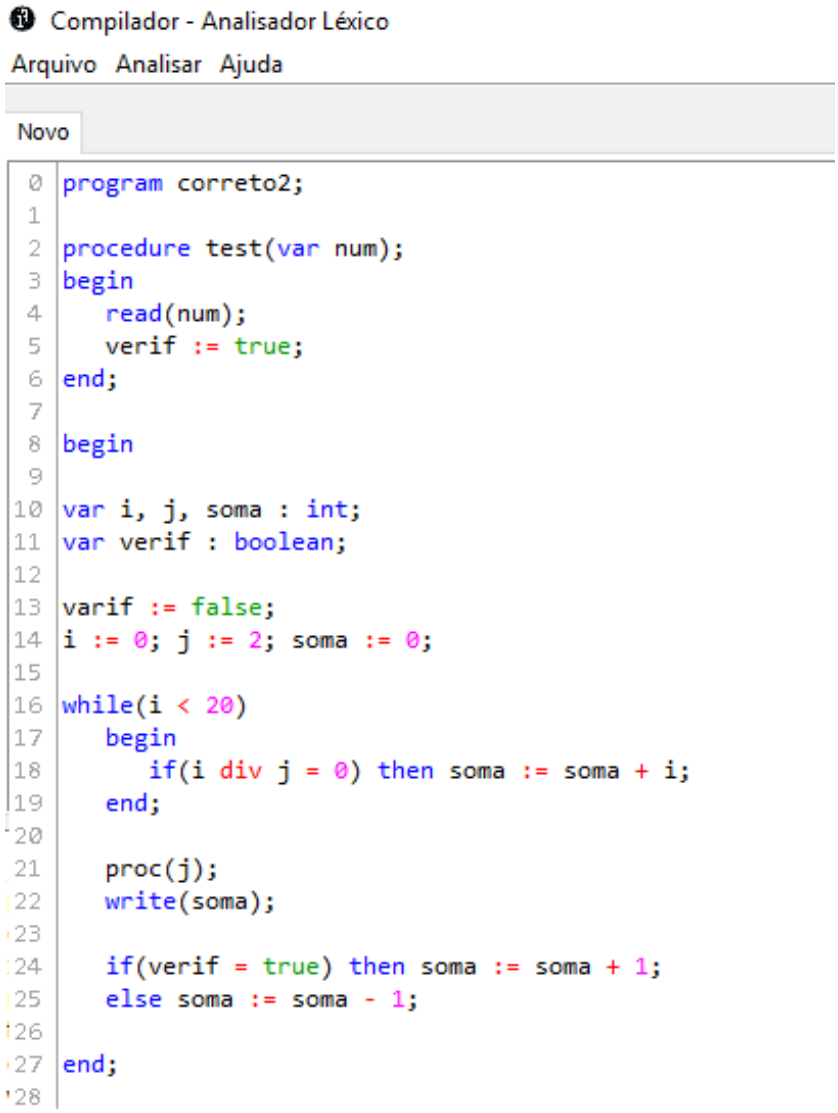
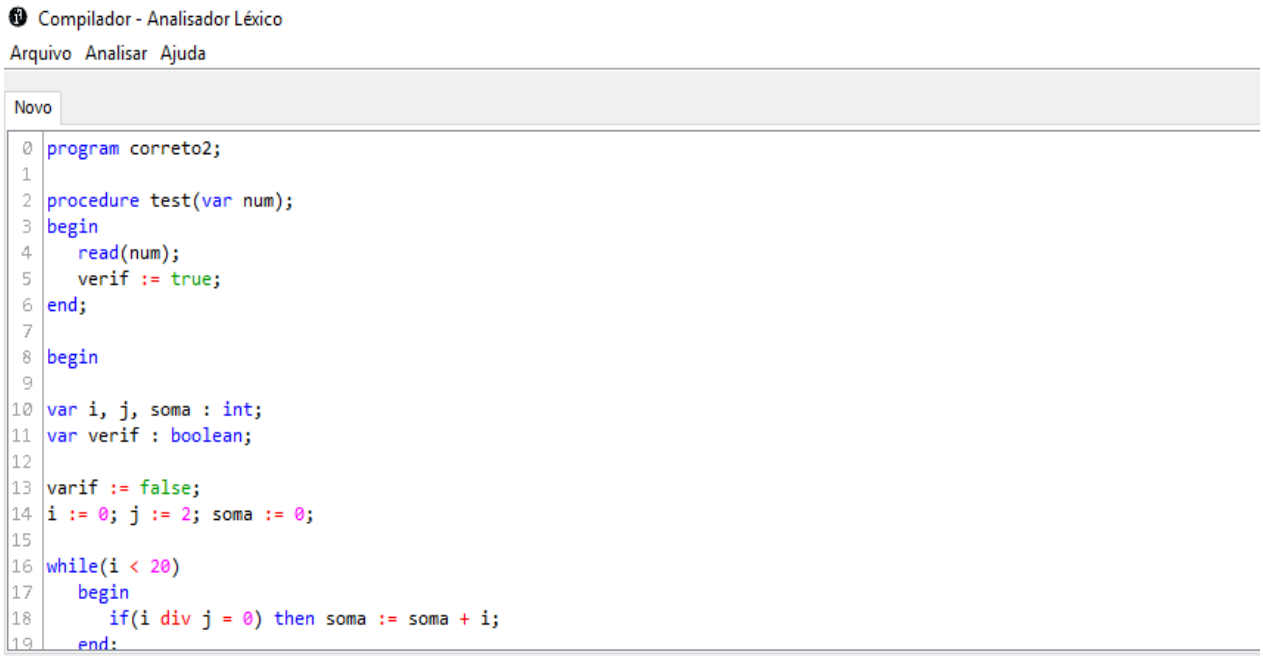


Figura 4 - Exemplo 2: Programa correto2



Lexema	Token	Linha	Coluna inicial	Coluna final
program	PALAVRA_RESERVADA_PROGRAM	0	0	6
correto2	IDENTIFICADOR	0	8	15
;	SIMBOLO_PONTO_VIRGULA	0	16	16
procedure	PALAVRA_RESERVADA_PROCEDURE	2	0	8
test	IDENTIFICADOR	2	10	13
(	PARENTESES_ABRE	2	14	14
var	PALAVRA_RESERVADA_VAR	2	15	17
num	IDENTIFICADOR	2	19	21
)	PARENTESES_FECHA	2	22	22
;	SIMBOLO_PONTO_VIRGULA	2	23	23
begin	PALAVRA_RESERVADA_BEGIN	3	0	4
read	PALAVRA_RESERVADA_READ	4	3	6
(	PARENTESES_ABRE	4	7	7
num	IDENTIFICADOR	4	8	10
)	PARENTESES_FECHA	4	11	11
;	SIMBOLO_PONTO_VIRGULA	4	12	12
verif	IDENTIFICADOR	5	3	7
:=	OPERADOR_ATRIBUICAO	5	9	10

Figura 5 - Exemplo 2: Tabela de símbolos léxicos do programa correto1

4.3.2 Teste 2: Programa com comentários

A seguir tem-se o segundo teste realizado. O segundo teste mostra os comentários tanto de uma única linha quando de múltiplas linhas. Observando a Figura 6 pode-se perceber que o analisador léxico cumpre sua função de ignorar comentários corretos. Contudo na Figura 7 é ilustrado um caso onde o comentário de múltiplas linhas é aberto, porem ele não é fechado.

Compilador - Analisador Léxico

Arquivo Analisar Ajuda

Novo

```
0  //Este é um programa com comentario
1
2  program comentario;
3
4  begin
5
6  {Este e um bloco de comentario
7  com mais de uma linha}
8
9  var verif : boolean;
10 verif := true;
11
12 end.
13
14
15
```

Lexema	Token	Linha	Coluna inicial	Coluna final
program	PALAVRA_RESERVADA_PROGRAM	2	0	6
comentario	IDENTIFICADOR	2	8	17
;	SIMBOLO_PONTO_VIRGULA	2	18	18
begin	PALAVRA_RESERVADA_BEGIN	4	0	4
var	PALAVRA_RESERVADA_VAR	9	0	2
verif	IDENTIFICADOR	9	4	8
:	SIMBOLO_DOIS_PONTOS	9	10	10
boolean	PALAVRA_RESERVADA_BOOLEAN	9	12	18
;	SIMBOLO_PONTO_VIRGULA	9	19	19
verif	IDENTIFICADOR	10	0	4
:=	OPERADOR_ATRIBUICAO	10	6	7
true	VALOR_LOGICO_TRUE	10	9	12
;	SIMBOLO_PONTO_VIRGULA	10	13	13
end	PALAVRA_RESERVADA_END	12	0	2
.	SIMBOLO_PONTO	12	3	3

Figura 6 - Exemplo 1: Programa com comentários corretos

Compilador - Analisador Léxico

Arquivo Analisar Ajuda

Novo

```
0  //Este é um programa com comentario
1
2  program comentario;
3
4  begin
5
6  {Este e um bloco de comentario
7  com mais de uma linha
8
9  var verif : boolean;
10 verif := true;
11
12 end.
```

Lexema	Token	Linha	Coluna inicial	Coluna final
begin	PALAVRA_RESERVADA_BEGIN	4	0	4
{	ERROR	6	0	0
Este	IDENTIFICADOR	6	1	4
e	IDENTIFICADOR	6	6	6
um	IDENTIFICADOR	6	8	9
bloco	IDENTIFICADOR	6	11	15
de	IDENTIFICADOR	6	17	18
comentario	IDENTIFICADOR	6	20	29
com	IDENTIFICADOR	7	0	2
mais	IDENTIFICADOR	7	4	7
de	IDENTIFICADOR	7	9	10
uma	IDENTIFICADOR	7	12	14

Figura 7 - Exemplo 2: Programa com comentário de múltiplas linhas incorreto

4.3.3 Teste 3: Programa incorreto

A seguir são apresentados programas incorretos contendo caracteres que não fazem parte da linguagem, observe a Figura 8 um exemplo deste programa e a Figura 9 com a tabela de símbolos léxicos relacionada a este mesmo programa.

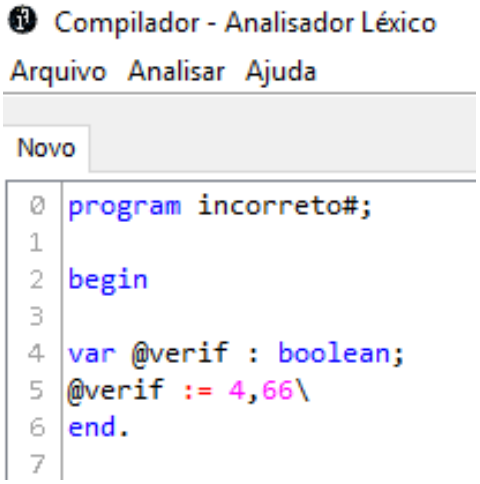


Figura 8 - Exemplo 1 - Programa incorreto com caracteres inválidos

Lexema	Token	Linha	Coluna inicial	Coluna final
program	PALAVRA_RESERVADA_PROGRAM	0	0	6
incorreto	IDENTIFICADOR	0	8	16
#	ERROR	0	17	17
;	SIMBOLO_PONTO_VIRGULA	0	18	18
begin	PALAVRA_RESERVADA_BEGIN	2	0	4
var	PALAVRA_RESERVADA_VAR	4	0	2
@	ERROR	4	4	4
1	NUMERO_INTEIRO	4	5	5
erif	IDENTIFICADOR	4	6	9
:	SIMBOLO_DOIS_PONTOS	4	11	11
boolean	PALAVRA_RESERVADA_BOOLEAN	4	13	19
;	SIMBOLO_PONTO_VIRGULA	4	20	20
@	ERROR	5	0	0
verif	IDENTIFICADOR	5	1	5
:=	OPERADOR_ATRIBUICAO	5	7	8
4	NUMERO_INTEIRO	5	10	10
,	SIMBOLO_VIRGULA	5	11	11
66	NUMERO_INTEIRO	5	12	13

Figura 9 - Exemplo 1 - Parte da tabela de símbolos do programa incorreto com caracteres inválidos