Centro Federal de Educação Tecnológica

Compiladores

Implementação de um Compilador

Analisador Léxico e Tabela de Símbolos

Alunos:
Lucas Loscheider Reis Muniz
Pedro Brandão Belisário

28 de maio, 2022

1 Introdução

Este trabalho foi realizado com o intuito de desenvolver um compilador em linguagem Java para gramática proposta segundo a estrutura em 1, presente no Capítulo 2. Dessa forma, este relatório tem como objetivo descrever as decisões tomadas durante o projeto para a construção do analisador léxico e de sua tabela de símbolos.

2 Gramática

Uma linguagem só é passível de compilação quando a mesma possuí uma gramática. Uma vez que a gramática é um conjunto de regras que uma linguagem deve seguir para garantir sua coerência, coesão e corretude.

A seguir se tem a gramática proposta para o compilador a ser construído, ressaltando que todo comentário deve estar entre os símbolos de porcentagem (%), e que há diferenciação entre letras maiúsculas e minúsculas.

Algorithm 1 Gramática proposta

1: program ::= **routine** body
2: body ::= [decl-list] **begin** stmt-list **end**
3: decl-list ::= **declare** decl{“;”decl “;”}
4: decl ::= type ident-list
5: ident-list ::= identifier {“,” identifier}
6: type ::= **int** | **float** | **char**
7: stmt-list ::= stmt{“;”stmt}
8: stmt ::= assign-stmt | if-stmt | while-stmt | repeat-stmt | read-stmt | write-stmt
9: assign-stmt ::= identifier “:=” simple-expr
10: if-stmt ::= **if** condition **then** stmt-list **end** | **if** condition **then** stmt-list **else** stmt-list **end**
11: condition ::= expression
12: repeat-stmt ::= **repeat** stmt-list stmt-suffix
13: stmt-suffix ::= **until** condition
14: while-stmt ::= stmt-prefix stmt-list **end**
15: stmt-prefix ::= **while** condition **do**
16: read-stmt ::= **read** “(” identifier “)”
17: write-stmt ::= **write** “(” writable “)”
18: writable ::= simple-expr | literal
19: expression ::= simple-expr | simple-expr relop simple-expr
20: simple-expr ::= term | simple-expr addop term
21: term ::= factor-a | term mulop factor-a
22: factor-a ::= factor | **not** factor | “-” factor
23: factor ::= identifier | constant | “(” expression “)”
24: relop ::= “=” | “>” | “>=” | “<” | “<=” | “<>”
25: addop ::= “+” | “-” | **or**
26: mulop ::= “*” | “/” | **and**
27: constant ::= integer_const | float_const | char_const
28: integer_const ::= digit$^+$
29: float_const ::= digit$^+$ “.” digit$^+$
30: char_const ::= “ ‘ ” carac “ ’ ”
31: literal ::= “ “ ” caractere* “ ” ”
32: identifier ::= letter (letter | digit)*
33: letter ::= [A-Za-z]
34: digit ::= [0-9]
35: carac ::= um dos caracters ASCII
36: caractere ::= um dos caracters ASCII, exceto as aspas e quebra de linha

Linha	**Palavras Reservadas**	**Palavras Reservadas**
1	int	if
2	float	then
3	char	else
4	or	repeat
5	and	until
6	not	while
7	routine	do
8	begin	read
9	declare	write
10	end	

Table 1: Palavras Reservadas na gramática

Desta forma, a partir desta gramática (Algoritmo 1), pode-se retirar as palavras reservadas apresentadas na Tabela 1, visto que há diferenciação entre caixa alta e baixa.

3 Analisador Léxico

Para o algoritmo 1 proposto no capítulo 2, foi construído num primeiro momento um Autômato Finito Determinístico (AFD), o qual representa graficamente o processo de funcionamento do analisador léxico desta gramática. Para facilitar a compreensão ele foi dividido em 3 partes que estão representadas nas Figuras 1, 2, 3, em que elas respectivamente representam o processo de identificação de números, identificação dos identificadores e das palavras reservadas, e por último os outros tokens que não começam com letra ou número.

Para a gramática proposta é possível que exista números inteiros e números reais, como pode ser visto nas linhas 28 e 29 da gramática (Algoritmo 1). Como os tipos de números podem começar com um ou mais dígitos de 0 à 9, a separação deles no AFD (Figura 1) ocorre no Estado 2, em que se for encontrado algo diferente de número e ponto tem-se a formação de um número inteiro (Estado 3), caso seja encontrado um ponto seguido por algo diferente de um número lança-se um erro léxico (Estado 6), por fim caso seja

encontrado um número após o ponto, tem-se a formação de um número real quando se encontrar algo que não seja um número (Estado 7)

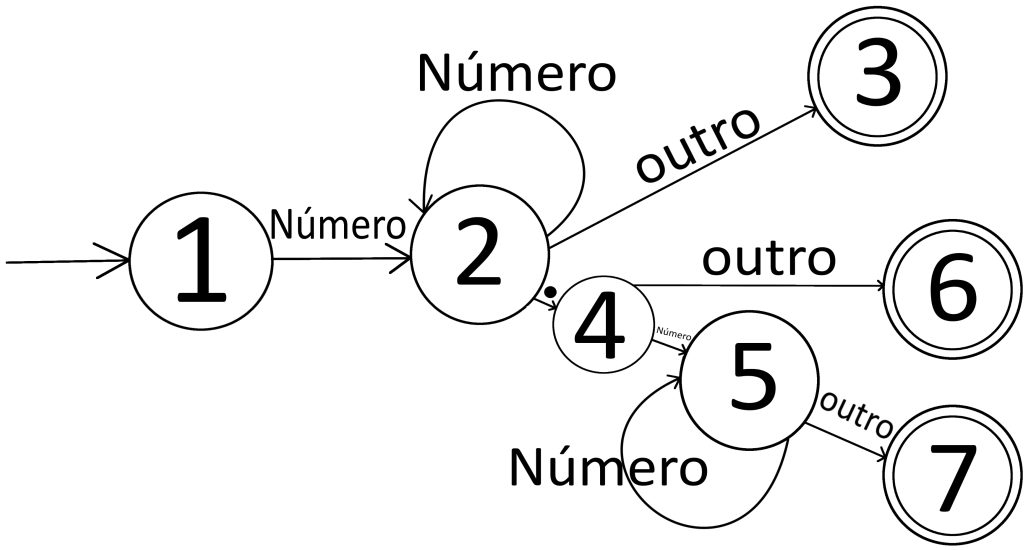

Figure 1: Na figura, tem-se o desenho de um AFD para identificação de números. O estado inicial é representado pelo número 1 e os estados finais estão representados pelo número 3, 6 e 7.

Segundo a gramática proposta, tanto identificadores quanto palavras reservadas começam com uma letra, como pode ser visto na linha 32 do Algoritmo 1 e na Tabela 1, respectivamente. Dessa forma, para o analisador léxico esta diferenciação se dá na transição do Estado 9 para os Estados 10 e 11 como pode ser visto na Figura 2, em que o primeiro ocorre quando o lexema encontrado exista na Tabela de Símbolos, de modo que somente se retorna o Token para o lexema encontrado. Por sua vez, a transição do Estado 9 para o Estado 11 ocorre quando o lexema encontrado não existe na Tabela de Símbolos com isto se adiciona o lexema na tabela de símbolos e retorna o Token correspondente.

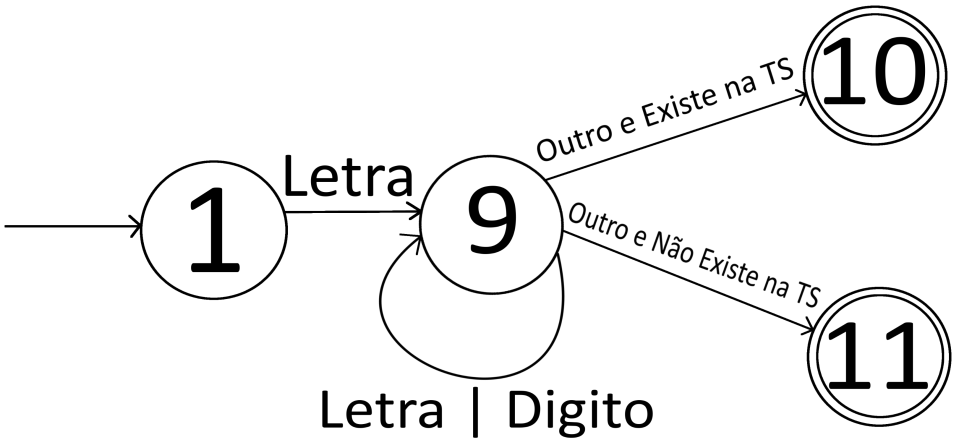

Figure 2: Na figura, tem-se o desenho de um AFD para o reconhecimento de identificadores e palavras reservadas. O estado inicial é representado pelo número 1 e os estados finais estão representados pelo números 10 e 11.

Para além de Tokens que representam números, identificadores e palavras reservadas, existe também os tokens para os caracteres especiais que apresentam algum significado na gramática. O processo de identificação deste conjunto de caracteres está representado na Figura 3 e o significado de cada um dos estados finais deste AFD está na Tabela 2.

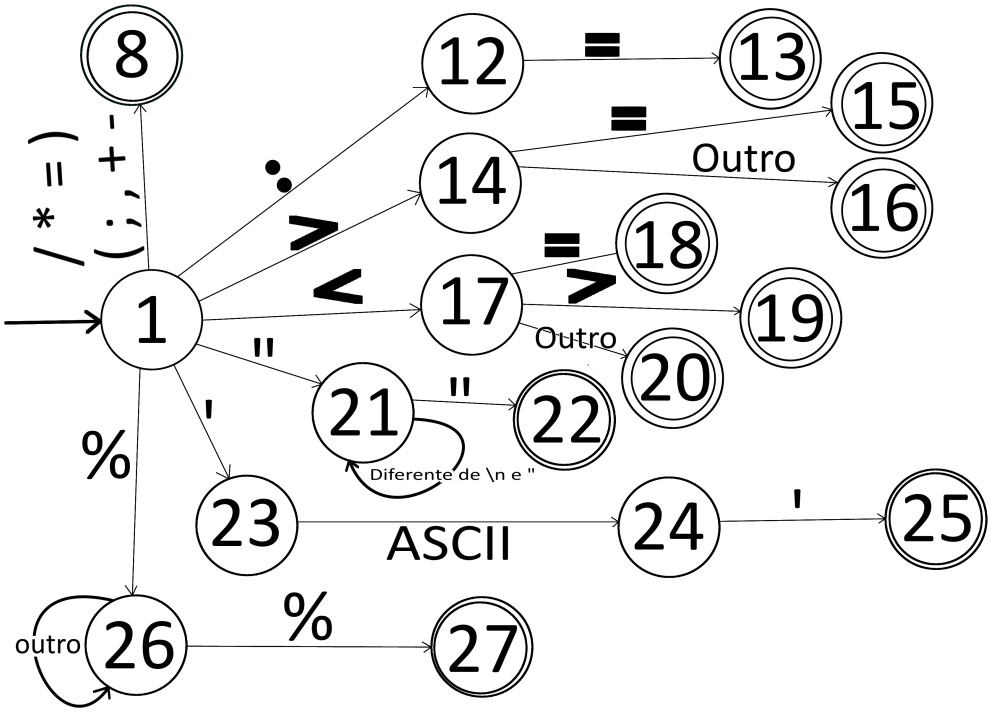

Figure 3: Na figura, tem-se o desenho de um AFD para identificar tokens que não começam com letra ou dígito. O estado inicial é representado pelo número 1 e os estados finais estão representados pelo números 8, 16, 15, 16, 18, 19, 20, 22, 25, 27.

Linha	**Estado Final**	**Significado**
1	8	Retorna o token correspondente a um dos símbolos da transição $1 \rightarrow 8$
2	13	retorna o token para associação
3	15	retorna o token para maior ou igual a
4	16	retorna o token para maior que
5	18	retorna o token para menor ou igual a
6	19	retorna token para diferente
7	20	retorna token para menor que
8	22	retorna o token para um literal
9	25	retorna o token para Char constante
10	27	Descarta-se tudo que estava entre os símbolos de %

Table 2: Significado dos estados finais do AFD da Figura 3

3.1 Código Analisador Léxico

O processo de codificação da gramática proposta, foi feito seguindo o padrão apresentado no Apêndice A do livro **Compiladores: princípios, técnicas e ferramentas - 2ª edição**, escrito por Alfred V. Aho, Jeffrey D. Ullmann, Monica S. Lam e Ravi Sethi, publicado pela editora Pearson. Dessa forma, o analisador léxico possui 7 classes, sendo a classe **Tag** um Enum para representação do tipo de cada token presente na gramática, do tipo de erro que ocorreu durante o processo de compilação. A classe responsável pela análise léxica, seguindo o exemplo do livro utilizado, é a classe Lexer.

A classe Lexer ao ser instanciada cria uma tabela de símbolos e a preenche com as palavras reservadas da gramática (Tabela 1). Para fazer somente a análise léxica foi criado a função **análiseLexica** nesta classe. Esta função recebe um arquivo como argumento e chama a função **scan** até que se chegue no final do arquivo passado como argumento, a cada término da função **scan** se imprime no console o token recebido. Ao terminar a leitura do arquivo, se imprime dos erros encontrados no arquivo fonte e a linha que este erro ocorreu, e, após os erros, é impresso a tabela de símbolos gerada.

Diferentemente da abordagem adotada no livro texto utilizado como base para o código do compilador, foi adotado uma recuperação de erro léxico na função **scan** de forma que toda vez que o não se foi possível chegar em um estado final nos AFDs das Figuras 2 e 3, ou no Estado 6 da Figura 1, salva-se uma mensagem de erro e a função **scan** retorna **null**.

4 Testes

Para executar a análise léxica dos arquivos, ao executar o código deve-se escrever o nome do arquivo que deve ser analisado na janela que aparecerá. Ressalva-se que, caso o arquivo não estiver na pasta raiz do analisador léxico, o nome do arquivo deve conter também o seu endereço.

4.1 Teste 1

Algorithm 2 Teste 1

```
1: routine
2: declare
3: int a, b;
4: int resul;
5: float a, x;
6: begin
7: a := 12a;
8: x := 12;
9: read(a);
10: read(b);
11: read(c);
12: result := (a*b + 1) / (c+2);
13: write "Resultado: ";
14: write (result);
15: end
```

Para o Teste 1, não se gerou nenhum erro, e seu fluxo de tokens e tabela de símbolos estão apresentados nas tabelas 3 e 4, respectivamente.

Linha	**Token**	**Valor**	Linha	**Token**	**Valor**
1	ROUTINE	routine	2	DECLARE	declare
3	INT	int	4	IDENTIFICADOR	a
5	VIRGULA		6	IDENTIFICADOR	b
7	PONTO_VIRGULA		8	INT	int
9	IDENTIFICADOR	resul	10	PONTO_VIRGULA	
11	FLOAT	float	12	IDENTIFICADOR	a
13	VIRGULA		14	IDENTIFICADOR	x
15	PONTO_VIRGULA		16	BEGIN	begin
17	IDENTIFICADOR	a	18	ATRIBUICAO	
19	INT_CONST	12	20	IDENTIFICADOR	a
21	PONTO_VIRGULA		22	IDENTIFICADOR	x
23	ATRIBUICAO		24	INT_CONST	12
25	PONTO_VIRGULA		26	READ	read
27	ABRE_PARENTESES		28	IDENTIFICADOR	a
29	FECHA_PARENTESES		30	PONTO_VIRGULA	
31	READ	read	32	ABRE_PARENTESES	
33	IDENTIFICADOR	b	34	FECHA_PARENTESES	
35	PONTO_VIRGULA		36	READ	read
37	ABRE_PARENTESES		38	IDENTIFICADOR	c
39	FECHA_PARENTESES		40	PONTO_VIRGULA	
41	IDENTIFICADOR	result	42	ATRIBUICAO	
43	ABRE_PARENTESES		44	IDENTIFICADOR	a
45	MULTIPLICACAO		46	IDENTIFICADOR	b
47	SOMA		48	INT_CONST	1
49	FECHA_PARENTESES		50	DIVISAO	
51	ABRE_PARENTESES		52	IDENTIFICADOR	c
53	SOMA		54	INT_CONST	2
55	FECHA_PARENTESES		56	PONTO_VIRGULA	
57	WRITE	write	58	LITERAL	"Resultado: "
59	PONTO_VIRGULA		60	WRITE	write
61	ABRE_PARENTESES		62	IDENTIFICADOR	result
63	FECHA_PARENTESES		64	PONTO_VIRGULA	
65	END	end			

Table 3: Fluxo de Tokens do Teste 1

Linha	Id	**Token**	**Valor**
1	resul	IDENTIFICADOR	resul
2	declare	DECLARE	declare
3	do	DO	do
4	while	WHILE	while
5	float	FLOAT	float
6	result	IDENTIFICADOR	result
7	not	NOT	not
8	routine	ROUTINE	routine
9	else	ELSE	else
10	and	AND	and
11	repeat	REPEAT	repeat
12	end	END	end
13	if	IF	if
14	write	WRITE	write
15	a	IDENTIFICADOR	a
16	b	IDENTIFICADOR	b
17	read	READ	read
18	or	OR	or
19	c	IDENTIFICADOR	c
20	then	THEN	then
21	int	INT	int
22	”Resultado: ”	LITERAL	”Resultado: ”
23	char	CHAR	char
24	x	IDENTIFICADOR	x
25	until	UNTIL	until
26	begin	BEGIN	begin

Table 4: Tabela de Símbolos do Teste 1

4.2 Teste 2

Algorithm 3 Teste 2

```
1: routine
2: int a, b, c;
3: float d, _var
4: begin
5: read (a);
6: b := a * a;
7: c := b + a/2 * (35/b); %aplica formula
8: write c;
9: val := 34.2
10: c = val + 2. + a;
11: write (val)
12: end
```

A primeira execução Teste 2 (Algoritmo 3) apresentou os erros token mal formado na linha 3, e final inesperado de arquivo na linha 12. O primeiro erro se dá devido ao identificador começar com *underline* e o segundo porque o comentário não foi encerrado. Corrigindo estes erros tem-se o algoritmo 4.

Algorithm 4 Teste 2 - 1ª Correção

```
1: routine
2: int a, b, c;
3: float d, var
4: begin
5: read (a);
6: b := a * a;
7: c := b + a/2 * (35/b); %aplica formula%
8: write c;
9: val := 34.2
10: c = val + 2. + a;
11: write (val)
12: end
```

A execução do algoritmo 4, gerou o erro de número real mal formado

na linha 10 não reportado anteriormente, pois este trecho foi considerado como interno ao comentário que não havia sido fechado no teste anterior (Algoritmo 3).

Algorithm 5 Teste 2 - 2ª Correção

```
 1: routine
 2: int a, b, c;
 3: float d, var
 4: begin
 5: read (a);
 6: b := a * a;
 7: c := b + a/2 * (35/b); %aplica formula%
 8: write c;
 9: val := 34.2
10: c = val + 2.0 + a;
11: write (val)
12: end
```

Ao fazer a análise léxica do algoritmo 5, não foi gerado nenhum erro, e seu fluxo de tokens e tabela de símbolos estão nas tabelas 5 e 6, respectivamente.

Linha	**Token**	**Valor**	Linha	**Token**	**Valor**
1	ROUTINE	routine	2	INT	int
3	IDENTIFICADOR	a	4	VIRGULA	
5	IDENTIFICADOR	b	6	VIRGULA	
7	IDENTIFICADOR	c	8	PONTO_VIRGULA	
9	FLOAT	float	10	IDENTIFICADOR	d
11	VIRGULA		12	IDENTIFICADOR	var
13	BEGIN	begin	14	READ	read
15	ABRE_PARENTESES		16	IDENTIFICADOR	a
17	FECHA_PARENTESES		18	PONTO_VIRGULA	
19	IDENTIFICADOR	b	20	ATRIBUICAO	
21	IDENTIFICADOR	a	22	MULTIPLICACAO	
23	IDENTIFICADOR	a	24	PONTO_VIRGULA	
25	IDENTIFICADOR	c	26	ATRIBUICAO	
27	IDENTIFICADOR	b	28	SOMA	
29	IDENTIFICADOR	a	30	DIVISAO	
31	INT_CONST	2	32	MULTIPLICACAO	
33	ABRE_PARENTESES		34	INT_CONST	35
35	DIVISAO		36	IDENTIFICADOR	b
37	FECHA_PARENTESES		38	PONTO_VIRGULA	
39	WRITE	write	40	IDENTIFICADOR	c
41	PONTO_VIRGULA		42	IDENTIFICADOR	val
43	ATRIBUICAO		44	FLOAT_CONST	34.2
45	IDENTIFICADOR	c	46	IGUAL	
47	IDENTIFICADOR	val	48	SOMA	
49	FLOAT_CONST	2.0	50	SOMA	
51	IDENTIFICADOR	a	52	PONTO_VIRGULA	
53	WRITE	write	54	ABRE_PARENTESES	
55	IDENTIFICADOR	val	56	FECHA_PARENTESES	
57	END	end			

Table 5: Fluxo de Tokens do Teste 2 após a correção de erros

Linha	Id	**Token**	**Valor**
1	declare	DECLARE	declare
2	do	DO	do
3	while	WHILE	while
4	float	FLOAT	float
5	not	NOT	not
6	routine	ROUTINE	routine
7	else	ELSE	else
8	and	AND	and
9	repeat	REPEAT	repeat
10	end	END	end
11	if	IF	if
12	write	WRITE	write
13	val	IDENTIFICADOR	val
14	a	IDENTIFICADOR	a
15	b	IDENTIFICADOR	b
16	read	READ	read
17	or	OR	or
18	c	IDENTIFICADOR	c
19	d	IDENTIFICADOR	d
20	var	IDENTIFICADOR	var
21	then	THEN	then
22	int	INT	int
23	char	CHAR	char
24	until	UNTIL	until
25	begin	BEGIN	begin

Table 6: Tabela de Símbolos do Teste 2 após a correção de erros

4.3 Teste 3

Algorithm 6 Teste 3

```
1: routine
2: declare
3: int a, aux;
4: float b;
5: begin
6: B := 0;
7: read (a);
8: read(b);
9: if (a<>) b then
10: begin
11:
12: if (a>b) then
13: aux := b;
14: b := a;
15: a := aux;
16: end;
17: write(a;
18: write(b)
19: end
20: else
21: write("Numeros iguais.");
22: end
```

Para o Teste 3 (Algoritmo 6), não se gerou nenhum erro, e seu fluxo de tokens e tabela de símbolos estão apresentados nas tabelas 7 e 8, respectivamente.

Linha	**Token**	**Valor**	Linha	**Token**	**Valor**
1	ROUTINE	routine	2	DECLARE	declare
3	INT	int	4	IDENTIFICADOR	a
5	VIRGULA		6	IDENTIFICADOR	aux
7	PONTO_VIRGULA		8	FLOAT	float
9	IDENTIFICADOR	b	10	PONTO_VIRGULA	
11	BEGIN	begin	12	IDENTIFICADOR	B
13	ATRIBUICAO		14	INT_CONST	0
15	PONTO_VIRGULA		16	READ	read
17	ABRE_PARENTESES		18	IDENTIFICADOR	a
19	FECHA_PARENTESES		20	PONTO_VIRGULA	
21	READ	read	22	ABRE_PARENTESES	
23	IDENTIFICADOR	b	24	FECHA_PARENTESES	
25	PONTO_VIRGULA		26	IF	if
27	ABRE_PARENTESES		28	IDENTIFICADOR	a
29	DIFERENTE	<>	30	FECHA_PARENTESES	
31	IDENTIFICADOR	b	32	THEN	then
33	BEGIN	begin	34	IF	if
35	ABRE_PARENTESES		36	IDENTIFICADOR	a
37	MAIOR_QUE	>	38	IDENTIFICADOR	b
39	FECHA_PARENTESES		40	THEN	then
41	IDENTIFICADOR	aux	42	ATRIBUICAO	
43	IDENTIFICADOR	b	44	PONTO_VIRGULA	
45	IDENTIFICADOR	b	46	ATRIBUICAO	
47	IDENTIFICADOR	a	48	PONTO_VIRGULA	
49	IDENTIFICADOR	a	50	ATRIBUICAO	
51	IDENTIFICADOR	aux	52	PONTO_VIRGULA	
53	END	end	54	PONTO_VIRGULA	
55	WRITE	write	56	ABRE_PARENTESES	
57	IDENTIFICADOR	a	58	PONTO_VIRGULA	
59	WRITE	write	60	ABRE_PARENTESES	
61	IDENTIFICADOR	b	62	FECHA_PARENTESES	
63	END	end	64	ELSE	else
65	WRITE	write	66	ABRE_PARENTESES	
67	LITERAL	"Numeros iguais."	68	FECHA_PARENTESES	
69	PONTO_VIRGULA		70	END	end
71	FIM_DE_ARQUIVO				

Table 7: Fluxo de Tokens do Teste 3

Linha	Id	**Token**	**Valor**
1	a	IDENTIFICADOR	a
2	b	IDENTIFICADOR	b
3	B	IDENTIFICADOR	B
4	read	READ	read
5	or	OR	or
6	declare	DECLARE	declare
7	aux	IDENTIFICADOR	aux
8	then	THEN	then
9	do	DO	do
10	while	WHILE	while
11	float	FLOAT	float
12	int	INT	int
13	not	NOT	not
14	routine	ROUTINE	routine
15	”Numeros iguais.”	LITERAL	”Numeros iguais.”
16	else	ELSE	else
17	and	AND	and
18	repeat	REPEAT	repeat
19	char	CHAR	char
20	end	END	end
21	until	UNTIL	until
22	begin	BEGIN	begin
23	if	IF	if
24	write	WRITE	write

Table 8: Tabela de Símbolos do Teste 3

4.4 Teste 4

Algorithm 7 Teste 4

```
1: routine
2: declare
3: int pontuacao, pontuacaoMaxina, disponibilidade;
4: char pontuacaoMinima;
5: begin
6: pontuacaoMinima = 50;
7: pontuacaoMaxima = 100;
8: write("Pontuacao do candidato: );
9: read(pontuacao);
10: write("Disponibilidade do candidato: ");
11: read(disponibilidade);
12: %
13: Processamento
14: %
15: while (pontuacao>0 and (pontuacao<=pontuacaoMaxima) do
16: if ((pontuação > pontuacaoMinima) and (disponibilidade=1)) then
17: write("Candidato aprovado.")
18: else
19: write("Candidato reprovado.")
20: end
21: write("Pontuacao do candidato: ");
22: read(pontuacao);
23: write("Disponibilidade do candidato: ");
24: read(disponibilidade);
25: end;
26: end
```

A execução do algoritmo 7, gerou o erro de literal mal formado na linha 8. Visto que não há uma aspas terminando a construção deste literal. Após sua correção tem-se o algoritmo 8

Algorithm 8 Teste 4 - Corrigido

```
1: routine
2: declare
3: int pontuacao, pontuacaoMaxina, disponibilidade;
4: char pontuacaoMinima;
5: begin
6: pontuacaoMinima = 50;
7: pontuacaoMaxima = 100;
8: write("Pontuacao do candidato: ");
9: read(pontuacao);
10: write("Disponibilidade do candidato: ");
11: read(disponibilidade);
12: %
13: Processamento
14: %
15: while (pontuacao>0 and (pontuacao<=pontuacaoMaxima) do
16: if ((pontuação > pontuacaoMinima) and (disponibilidade=1)) then
17: write("Candidato aprovado.")
18: else
19: write("Candidato reprovado.")
20: end
21: write("Pontuacao do candidato: ");
22: read(pontuacao);
23: write("Disponibilidade do candidato: ");
24: read(disponibilidade);
25: end;
26: end
```

Ao fazer a análise léxica do algoritmo 8, não houve a ocorrencia de nenhum erro. Desta forma seu fluxo de tokens e tabela de símbolos estão respectivamente representados nas tabelas 9 e 10

Linha	**Token**	**Valor**	Linha	**Token**	**Valor**
1	ROUTINE	routine	2	DECLARE	declare
3	INT	int	4	IDENTIFICADOR	pontuacao
5	VIRGULA		6	IDENTIFICADOR	pontuacaoMaxina
7	VIRGULA		8	IDENTIFICADOR	disponibilidade
9	PONTO_VIRGULA		10	CHAR	char
11	IDENTIFICADOR	pontuacaoMinima	12	PONTO_VIRGULA	
13	BEGIN	begin	14	IDENTIFICADOR	pontuacaoMinima
15	IGUAL		16	INT_CONST	50
17	PONTO_VIRGULA		18	IDENTIFICADOR	pontuacaoMaxima
19	IGUAL		20	INT_CONST	100
21	PONTO_VIRGULA		22	WRITE	write
23	ABRE_PARENTESES		24	LITERAL	"Pontuacao do candidato "
25	FECHA_PARENTESES		26	PONTO_VIRGULA	
27	READ	read	28	ABRE_PARENTESES	
29	IDENTIFICADOR	pontuacao	30	FECHA_PARENTESES	
31	PONTO_VIRGULA		32	WRITE	write
33	ABRE_PARENTESES		34	LITERAL	"Disponibilidade do candidato "
35	FECHA_PARENTESES		36	PONTO_VIRGULA	
37	READ	read	38	ABRE_PARENTESES	
39	IDENTIFICADOR	disponibilidade	40	FECHA_PARENTESES	
41	PONTO_VIRGULA		42	WHILE	while
43	ABRE_PARENTESES		44	IDENTIFICADOR	pontuacao
45	MAIOR_QUE	>	46	INT_CONST	0
47	AND	and	48	ABRE_PARENTESES	
49	IDENTIFICADOR	pontuacao	50	MENOR_IGUAL	<=
51	IDENTIFICADOR	pontuacaoMaxima	52	FECHA_PARENTESES	
53	DO	do	54	IF	if
55	ABRE_PARENTESES		56	ABRE_PARENTESES	
57	IDENTIFICADOR	pontuação	58	MAIOR_QUE	>
59	IDENTIFICADOR	pontuacaoMinima	60	FECHA_PARENTESES	
61	AND	and	62	ABRE_PARENTESES	
63	IDENTIFICADOR	disponibilidade	64	IGUAL	
65	INT_CONST	1	66	FECHA_PARENTESES	
67	FECHA_PARENTESES		68	THEN	then
69	WRITE	write	70	ABRE_PARENTESES	
71	LITERAL	"Candidato aprovado."	72	FECHA_PARENTESES	
73	ELSE	else	74	WRITE	write
75	ABRE_PARENTESES		76	LITERAL	"Candidato reprovado."
77	FECHA_PARENTESES		78	END	end
79	WRITE	write	80	ABRE_PARENTESES	
81	LITERAL	"Pontuacao do candidato "	82	FECHA_PARENTESES	
83	PONTO_VIRGULA		84	READ	read
85	ABRE_PARENTESES		86	IDENTIFICADOR	pontuacao
87	FECHA_PARENTESES		88	PONTO_VIRGULA	
89	WRITE	write	90	ABRE_PARENTESES	
91	LITERAL	"Disponibilidade do candidato "	92	FECHA_PARENTESES	
93	PONTO_VIRGULA		94	READ	read
95	ABRE_PARENTESES		96	IDENTIFICADOR	disponibilidade
97	FECHA_PARENTESES		98	PONTO_VIRGULA	
99	END	end	100	PONTO_VIRGULA	
101	END	end			

Table 9: Fluxo de Tokens do Teste 4 após sua correção

Linha	Id	**Token**	**Valor**
1	declare	DECLARE	declare
2	pontuacaoMaxima	IDENTIFICADOR	pontuacaoMaxima
3	”Disponibilidade do candidato: ”	LITERAL	”Disponibilidade do candidato: ”
4	do	DO	do
5	while	WHILE	while
6	float	FLOAT	float
7	not	NOT	not
8	routine	ROUTINE	routine
9	else	ELSE	else
10	and	AND	and
11	repeat	REPEAT	repeat
12	”Candidato aprovado.”	LITERAL	”Candidato aprovado.”
13	end	END	end
14	”Pontuacao do candidato: ”	LITERAL	”Pontuacao do candidato: ”
15	if	IF	if
16	write	WRITE	write
17	pontuacaoMinima	IDENTIFICADOR	pontuacaoMinima
18	read	READ	read
19	or	OR	or
20	disponibilidade	IDENTIFICADOR	disponibilidade
21	pontuacaoMaxina	IDENTIFICADOR	pontuacaoMaxina
22	then	THEN	then
23	int	INT	int
24	pontuacao	IDENTIFICADOR	pontuacao
25	pontuação	IDENTIFICADOR	pontuação
26	”Candidato reprovado.”	LITERAL	”Candidato reprovado.”
27	char	CHAR	char
28	until	UNTIL	until
29	begin	BEGIN	begin

Table 10: Tabela de Símbolos do Teste 4 após correção

4.5 Teste 5

Algorithm 9 Teste 5

```
1: declare
2: integer a, b, c, maior;
3: char outro;
4: begin
5: repeat
6: write("A: ");
7: read(a);
8: write("B: ");
9: read(b);
10: write("C: ");
11: read(c);
12: if ( (a>b) and (a>c) ) end
13: maior := a
14:
15: else
16: if (b>c) then
17: maior := b;
18:
19: else
20: maior := c
21: end
22: end
23: write("Maior valor:"");
24: write (maior);
25: write ("Outro? (S/N)");
26: read(outro);
27: until (outro = 'N' —— outro = 'n)
28: end
```

A execução do algoritmo 9, gerou o 3 erros, foram eles: um literal mal formado na linha 23, dois tokens mal formados na linha 27 e um fim inesperado de arquivo também na linha 27. O primeiro erro ocorreu porque tem uma aspas sem par, os dois tokens mal formados na linha 27 ocorreram por causa das duas barras verticais, e por fim o final inesperado de arquivo se

deu porque a aspas simples não possui um par a fechando. Após a correção destes erros tem-se o algoritmo 10

Algorithm 10 Teste 5 - Corrigido

```
1: declare
2: integer a, b, c, maior;
3: char outro;
4: begin
5: repeat
6: write("A: ");
7: read(a);
8: write("B: ");
9: read(b);
10: write("C: ");
11: read(c);
12: if ( (a>b) and (a>c) ) end
13: maior := a
14:
15: else
16: if (b>c) then
17: maior := b;
18:
19: else
20: maior := c
21: end
22: end
23: write("Maior valor:");
24: write (maior);
25: write ("Outro? (S/N)");
26: read(outro);
27: until (outro = 'N' or outro = 'n')
28: end
```

Ao fazer a análise léxica do algoritmo 10, não houve a ocorrencia de nenhum erro. Desta forma seu fluxo de tokens e tabela de símbolos estão respectivamente representados nas tabelas 11 e 12

Linha	**Token**	**Valor**	Linha	**Token**	**Valor**
1	DECLARE	declare	2	IDENTIFICADOR	integer
3	IDENTIFICADOR	a	4	VIRGULA	
5	IDENTIFICADOR	b	6	VIRGULA	
7	IDENTIFICADOR	c	8	VIRGULA	
9	IDENTIFICADOR	maior	10	PONTO_VIRGULA	
11	CHAR	char	12	IDENTIFICADOR	outro
13	PONTO_VIRGULA		14	BEGIN	begin
15	REPEAT	repeat	16	WRITE	write
17	ABRE_PARENTESES		18	LITERAL	”A: ”
19	FECHA_PARENTESES		20	PONTO_VIRGULA	
21	READ	read	22	ABRE_PARENTESES	
23	IDENTIFICADOR	a	24	FECHA_PARENTESES	
25	PONTO_VIRGULA		26	WRITE	write
27	ABRE_PARENTESES		28	LITERAL	”B: ”
29	FECHA_PARENTESES		30	PONTO_VIRGULA	
31	READ	read	32	ABRE_PARENTESES	
33	IDENTIFICADOR	b	34	FECHA_PARENTESES	
35	PONTO_VIRGULA		36	WRITE	write
37	ABRE_PARENTESES		38	LITERAL	”C: ”
39	FECHA_PARENTESES		40	PONTO_VIRGULA	
41	READ	read	42	ABRE_PARENTESES	
43	IDENTIFICADOR	c	44	FECHA_PARENTESES	
45	PONTO_VIRGULA		46	IF	if
47	ABRE_PARENTESES		48	ABRE_PARENTESES	
49	IDENTIFICADOR	a	50	MAIOR_QUE	>
51	IDENTIFICADOR	b	52	FECHA_PARENTESES	
53	AND	and	54	ABRE_PARENTESES	
55	IDENTIFICADOR	a	56	MAIOR_QUE	>
57	IDENTIFICADOR	c	58	FECHA_PARENTESES	
59	FECHA_PARENTESES		60	END	end
61	IDENTIFICADOR	maior	62	ATRIBUICAO	
63	IDENTIFICADOR	a	64	ELSE	else
65	IF	if	66	ABRE_PARENTESES	
67	IDENTIFICADOR	b	68	MAIOR_QUE	>
69	IDENTIFICADOR	c	70	FECHA_PARENTESES	
71	THEN	then	72	IDENTIFICADOR	maior
73	ATRIBUICAO		74	IDENTIFICADOR	b
75	PONTO_VIRGULA		76	ELSE	else
77	IDENTIFICADOR	maior	78	ATRIBUICAO	
79	IDENTIFICADOR	c	80	END	end
81	END	end	82	WRITE	write
83	ABRE_PARENTESES		84	LITERAL	”Maior valor:”
85	FECHA_PARENTESES		86	PONTO_VIRGULA	
87	WRITE	write	88	ABRE_PARENTESES	
89	IDENTIFICADOR	maior	90	FECHA_PARENTESES	
91	PONTO_VIRGULA		92	WRITE	write
93	ABRE_PARENTESES		94	LITERAL	”Outro? (S/N)”
95	FECHA_PARENTESES		96	PONTO_VIRGULA	
97	READ	read	98	ABRE_PARENTESES	
99	IDENTIFICADOR	outro	100	FECHA_PARENTESES	
101	PONTO_VIRGULA		102	UNTIL	until
103	ABRE_PARENTESES		104	IDENTIFICADOR	outro
105	IGUAL		106	CHAR_CONST	’N’
107	OR	or	108	IDENTIFICADOR	outro
109	IGUAL		110	CHAR_CONST	’n’
111	FECHA_PARENTESES		112	END	end

Table 11: Fluxo de Tokens do Teste 5 após sua correção

Linha	Id	**Token**	**Valor**
1	”A: ”	LITERAL	”A: ”
2	’N’	CHAR_CONST	’N’
3	declare	DECLARE	declare
4	”C: ”	LITERAL	”C: ”
5	do	DO	do
6	integer	IDENTIFICADOR	integer
7	while	WHILE	while
8	float	FLOAT	float
9	not	NOT	not
10	routine	ROUTINE	routine
11	maior	IDENTIFICADOR	maior
12	else	ELSE	else
13	and	AND	and
14	repeat	REPEAT	repeat
15	outro	IDENTIFICADOR	outro
16	end	END	end
17	if	IF	if
18	write	WRITE	write
19	’n’	CHAR_CONST	’n’
20	a	IDENTIFICADOR	a
21	”B: ”	LITERAL	”B: ”
22	b	IDENTIFICADOR	b
23	read	READ	read
24	or	OR	or
25	c	IDENTIFICADOR	c
26	then	THEN	then
27	int	INT	int
28	char	CHAR	char
29	until	UNTIL	until
30	”Outro? (S/N)”	LITERAL	”Outro? (S/N)”
31	begin	BEGIN	begin
32	”Maior valor:”	LITERAL	”Maior valor:”

Table 12: Tabela de Símbolos do Teste 5 após correção

4.6 Teste 6

O sexto teste aplicado no analisador léxico foi de um algoritmo para o cálculo do volume de um esfera para um raio dado pelo usuário, como pode ser visto no Algoritmo 11

Algorithm 11 Teste 6

```
1: routine
2: declare
3: float raio, volume;
4: begin
5: raio := 0;
6: volume := 0;
7:
8: write("ESTE PROGRAMA MOSTRA O VOLUME DE UMA ESFERA
   DE ACORDO COM O RAIO.");
9: write("Digite o valor do RAIO: ");
10: read(raio);
11:
12: volume := 4 * 3.14 * raio * (raio/3);
13:
14: write("O volume da esfera e: ");
15: write(volume);
16:
17: end
```

A execução deste algoritmo não gerou nenhum erro léxico, e apresentou o fluxo de tokens e tabela de símbolos presentes na Tabelas 13 e 14, respectivamente.

Linha	Token	Valor	Linha	Token	Valor
1	ROUTINE	routine	2	DECLARE	declare
3	FLOAT	float	4	IDENTIFICADOR	raio
5	VIRGULA		6	IDENTIFICADOR	volume
7	PONTO_VIRGULA		8	BEGIN	begin
9	IDENTIFICADOR	raio	10	ATRIBUICAO	
11	INT_CONST	0	12	PONTO_VIRGULA	
13	IDENTIFICADOR	volume	14	ATRIBUICAO	
15	INT_CONST	0	16	PONTO_VIRGULA	
17	WRITE	write	18	ABRE_PARENTESES	
19	LITERAL	"ESTE PROGRAMA MOSTRA O VOLUME DE UMA ESFERA DE ACORDO COM O RAIO."	20	FECHA_PARENTESES	
21	PONTO_VIRGULA		22	WRITE	write
23	ABRE_PARENTESES		24	LITERAL	"Digite o valor do RAIO: "
25	FECHA_PARENTESES		26	PONTO_VIRGULA	
27	READ	read	28	ABRE_PARENTESES	
29	IDENTIFICADOR	raio	30	FECHA_PARENTESES	
31	PONTO_VIRGULA		32	IDENTIFICADOR	volume
33	ATRIBUICAO		34	INT_CONST	4
35	MULTIPLICACAO		36	FLOAT_CONST	3.1399999
37	MULTIPLICACAO		38	IDENTIFICADOR	raio
39	MULTIPLICACAO		40	ABRE_PARENTESES	
41	IDENTIFICADOR	raio	42	DIVISAO	
43	INT_CONST	3	44	FECHA_PARENTESES	
45	PONTO_VIRGULA		46	WRITE	write
47	ABRE_PARENTESES		48	LITERAL	"O volume da esfera e: "
49	FECHA_PARENTESES		50	PONTO_VIRGULA	
51	WRITE	write	52	ABRE_PARENTESES	
53	IDENTIFICADOR	volume	54	FECHA_PARENTESES	
55	PONTO_VIRGULA		56	END	end
57	FIM_DE_ARQUIVO				

Table 13: Fluxo de Tokens do Teste 6

Linha	Id	Token	Valor
1	raio	IDENTIFICADOR	raio
2	read	READ	read
3	or	OR	or
4	declare	DECLARE	declare
5	"ESTE PROGRAMA MOSTRA O VOLUME DE UMA ESFERA DE ACORDO COM O RAIO."	LITERAL	"ESTE PROGRAMA MOSTRA O VOLUME DE UMA ESFERA DE ACORDO COM O RAIO."
6	then	THEN	then
7	do	DO	do
8	while	WHILE	while
9	float	FLOAT	float
10	int	INT	int
11	volume	IDENTIFICADOR	volume
12	not	NOT	not
13	routine	ROUTINE	routine
14	else	ELSE	else
15	and	AND	and
16	repeat	REPEAT	repeat
17	char	CHAR	char
18	"Digite o valor do RAIO: "	LITERAL	"Digite o valor do RAIO: "
19	end	END	end
20	until	UNTIL	until
21	"O volume da esfera e: "	LITERAL	"O volume da esfera e: "
22	begin	BEGIN	begin
23	if	IF	if
24	write	WRITE	write

Table 14: Tabela de Símbolos do Teste 6

4.7 Teste 7

O sétimo e último teste aplicado no analisador léxico foi de um algoritmo para o cálculo da idade de uma pessoa à partir de sua data de nascimento Algoritmo 12. Por fim, a execução deste algoritmo não gerou nenhum erro léxico, e apresentou o fluxo de tokens e tabela de símbolos presentes na Tabelas 15 e 16, respectivamente.

Algorithm 12 Teste 7

```
1: routine
2: declare
3: int dias, dia, meses, mes, anos, ano, calculo, diaatual, mesatual, anoatual;
4: begin
5:
6: diaatual := 3;
7: mesatual := 9;
8: anoatual := 2022;
9:
10: write("Insira os dados pessoais. ");
11: write("Dia de nascimento da pessoa: ");
12: read(dia);
13: write("Mês de nascimento da pessoa: ");
14: read(mes);
15: write("Ano de nascimento da pessoa: ");
16: read(ano);
17:
18: calculo := 365*anoatual + 30*mesatual + diaatual - 365*ano - 30*mes -
    dia;
19:
20: anos := calculo/365;
21:
22: meses := calculo/30;
23:
24: dias := calculo - (calculo*365*30);
25:
26: write("Você tem: ");
27: write(anos);
28: write(", ");
29: write(meses);
30: write(" e ");
31: write(dias);
32: write(" de idade.");
33:
34: end
```

Linha	Token	Valor	Linha	Token	Valor
1	ROUTINE	routine	2	DECLARE	declare
3	INT	int	4	IDENTIFICADOR	dias
5	VIRGULA		6	IDENTIFICADOR	dia
7	VIRGULA		8	IDENTIFICADOR	meses
9	VIRGULA		10	IDENTIFICADOR	mes
11	VIRGULA		12	IDENTIFICADOR	anos
13	VIRGULA		14	IDENTIFICADOR	ano
15	VIRGULA		16	IDENTIFICADOR	calculo
17	VIRGULA		18	IDENTIFICADOR	diaatual
19	VIRGULA		20	IDENTIFICADOR	mesatual
21	VIRGULA		22	IDENTIFICADOR	anoatual
23	PONTO_VIRGULA		24	BEGIN	begin
25	IDENTIFICADOR	diaatual	26	ATRIBUICAO	
27	INT_CONST	3	28	PONTO_VIRGULA	
29	IDENTIFICADOR	mesatual	30	ATRIBUICAO	
31	INT_CONST	9	32	PONTO_VIRGULA	
33	IDENTIFICADOR	anoatual	34	ATRIBUICAO	
35	INT_CONST	2022	36	PONTO_VIRGULA	
37	WRITE	write	38	ABRE_PARENTESES	
39	LITERAL	"Insira os dados pessoais. "	40	FECHA_PARENTESES	
41	PONTO_VIRGULA		42	WRITE	write
43	ABRE_PARENTESES		44	LITERAL	"Dia de nascimento da pessoa: "
45	FECHA_PARENTESES		46	PONTO_VIRGULA	
47	READ	read	48	ABRE_PARENTESES	
49	IDENTIFICADOR	dia	50	FECHA_PARENTESES	
51	PONTO_VIRGULA		52	WRITE	write
53	ABRE_PARENTESES		54	LITERAL	"Mês de nascimento da pessoa: "
55	FECHA_PARENTESES		56	PONTO_VIRGULA	
57	READ	read	58	ABRE_PARENTESES	
59	IDENTIFICADOR	mes	60	FECHA_PARENTESES	
61	PONTO_VIRGULA		62	WRITE	write
63	ABRE_PARENTESES		64	LITERAL	"Ano de nascimento da pessoa: "
65	FECHA_PARENTESES		66	PONTO_VIRGULA	
67	READ	read	68	ABRE_PARENTESES	
69	IDENTIFICADOR	ano	70	FECHA_PARENTESES	
71	PONTO_VIRGULA		72	IDENTIFICADOR	calculo
73	ATRIBUICAO		74	INT_CONST	365
75	MULTIPLICACAO		76	IDENTIFICADOR	anoatual
77	SOMA		78	INT_CONST	30
79	MULTIPLICACAO		80	IDENTIFICADOR	mesatual
81	SOMA		82	IDENTIFICADOR	diaatual
83	SUB_HIFEN		84	INT_CONST	365
85	MULTIPLICACAO		86	IDENTIFICADOR	ano
87	SUB_HIFEN		88	INT_CONST	30
89	MULTIPLICACAO		90	IDENTIFICADOR	mes
91	SUB_HIFEN		92	IDENTIFICADOR	dia
93	PONTO_VIRGULA		94	IDENTIFICADOR	anos
95	ATRIBUICAO		96	IDENTIFICADOR	calculo
97	DIVISAO		98	INT_CONST	365
99	PONTO_VIRGULA		100	IDENTIFICADOR	meses
101	ATRIBUICAO		102	IDENTIFICADOR	calculo
103	DIVISAO		104	INT_CONST	30
105	PONTO_VIRGULA		106	IDENTIFICADOR	dias
107	ATRIBUICAO		108	IDENTIFICADOR	calculo
109	SUB_HIFEN		110	ABRE_PARENTESES	
111	IDENTIFICADOR	calculo	112	MULTIPLICACAO	
113	INT_CONST	365	114	MULTIPLICACAO	
115	INT_CONST	30	116	FECHA_PARENTESES	
117	PONTO_VIRGULA		118	WRITE	write
119	ABRE_PARENTESES		120	LITERAL	"Você tem: "
121	FECHA_PARENTESES		122	PONTO_VIRGULA	
123	WRITE	write	124	ABRE_PARENTESES	
125	IDENTIFICADOR	anos	126	FECHA_PARENTESES	
127	PONTO_VIRGULA		128	WRITE	write
129	ABRE_PARENTESES		130	LITERAL	", "
131	FECHA_PARENTESES		132	PONTO_VIRGULA	
133	WRITE	write	134	ABRE_PARENTESES	
135	IDENTIFICADOR	meses	136	FECHA_PARENTESES	
137	PONTO_VIRGULA		138	WRITE	write
139	ABRE_PARENTESES		140	LITERAL	" e "
141	FECHA_PARENTESES		142	PONTO_VIRGULA	
143	WRITE	write	144	ABRE_PARENTESES	
145	IDENTIFICADOR	dias	146	FECHA_PARENTESES	
147	PONTO_VIRGULA		148	WRITE	write
149	ABRE_PARENTESES		150	LITERAL	" de idade."
151	FECHA_PARENTESES		152	PONTO_VIRGULA	
153	END	end			

Table 15: Fluxo de Tokens do Teste 7

Linha	Id	**Token**	**Valor**
1	declare	DECLARE	declare
2	” e ”	LITERAL	” e ”
3	do	DO	do
4	while	WHILE	while
5	float	FLOAT	float
6	”Ano de nascimento da pessoa: ”	LITERAL	”Ano de nascimento da pessoa: ”
7	not	NOT	not
8	mesatual	IDENTIFICADOR	mesatual
9	routine	ROUTINE	routine
10	else	ELSE	else
11	and	AND	and
12	repeat	REPEAT	repeat
13	”Você tem: ”	LITERAL	”Você tem: ”
14	dias	IDENTIFICADOR	dias
15	end	END	end
16	mes	IDENTIFICADOR	mes
17	”, ”	LITERAL	”, ”
18	if	IF	if
19	write	WRITE	write
20	anoatual	IDENTIFICADOR	anoatual
21	”Dia de nascimento da pessoa: ”	LITERAL	”Dia de nascimento da pessoa: ”
22	read	READ	read
23	or	OR	or
24	ano	IDENTIFICADOR	ano
25	meses	IDENTIFICADOR	meses
26	diaatual	IDENTIFICADOR	diaatual
27	” de idade.”	LITERAL	” de idade.”
28	”Mês de nascimento da pessoa: ”	LITERAL	”Mês de nascimento da pessoa: ”
29	then	THEN	then
30	calculo	IDENTIFICADOR	calculo
31	int	INT	int
32	char	CHAR	char
33	until	UNTIL	until
34	anos	IDENTIFICADOR	anos
35	”Insira os dados pessoais. ”	LITERAL	”Insira os dados pessoais. ”
36	begin	BEGIN	begin
37	dia	IDENTIFICADOR	dia

Table 16: Tabela de Símbolos do Teste 7