Análise Léxica: Trabalho 1

Daniella Martins Vasconcellos[1], **Matheus Soppa Geremias**[1]

[1]Departamento de Ciência da Computação – Universidade Estadual de Santa Catarina (UDESC)

daniellavasconc@gmail.com, suppersoppa@gmail.com

Resumo. *Trabalho feito para a disciplina de Compiladores (COM0002), ministrada pelo Prof. Ricardo Ferreira Martins. O trabalho se divide em quatro sessões, sendo elas a definição da gramática, a especificação FLEX, o destaque do ambiente e o dicionário de símbolos.*

1. Definição da gramática

Inicialmente, foi realizado a definição da gramática, que pode ser utilizada para a especificação FLEX. Dessa maneira, caracterizou-se char, digit, espaço-opt, integer, double, symbol, punctuation, boolean, type, keyword, operator e string.

A escolha dessa classificação se deve ao fato de que essas categorias são a base de muitas linguagens de programação.

O resultado desta etapa pode ser visualizado na tabela da figura 1.

2. FLEX

No arquivo *"trab1.lex"*, fazemos a primeira análise léxica de detecção dos tokens do código em linguagem C, que se constituiu como a linguagem escolhida para o desenvolvimento do trabalho por conta da facilidade e a familiaridade de uso.

Antes de mais nada, declaramos o número de linhas e colunas da matriz se iniciando em 0, pelas variáveis **num_linhas** e **num_colunas**. A variável **num_tokens** também é iniciada com valor 0, pois ela contará o número total de tokens analisados pelo lex. Também foi declarado um *struct* do tipo **token**, que irá guardar os valores do id, linha, coluna, tipo e descrição de um determinado token a ser analisado. Essa estrutura é posteriormente chamada a cada análise de token, para que os valores anteriores possam ser guardados e depois recuperados para formarem a tabela de símbolos (próxima seção).

Definimos então o que são **dígitos** (qualquer número de 0 a 9) e **id**s (nome de variáveis, funções, classes, afins; devem começar com uma letra maiúscula ou minúscula, e depois podem seguir por outros caracteres alfanuméricos).

Então, a partir da gramática, começamos a definir como serão reconhecidos os tokens, e o que o programa deverá fazer caso ele encontre um token de determinado tipo; no caso, ele imprime qual o tipo do token, em qual linha (num_linhas) e coluna (num_colunas) ele se encontra, e adiciona um à variável do número total de tokens (num_tokens). As definições feitas neste trabalho foram:

- **Inteiro:** Um dígito (0-9) que pode ser seguido ou não por outros dígitos.
- **Real:** Um dígito (0-9) que pode ser seguido por outros dígitos, mas que contém um ponto (".") indicando as casas decimais. Depois do ponto há outros dígitos.

Figure 1. Gramática.

TIPO	TOKENS
<char>	::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
<digit>	::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<espaço-opt>	::= " " <espaço-opt> | ""
<integer>	::= <digit> | <digit><integer>
<double>	::= <integer>"."<integer>
<symbol>	::= "@" | "#" | "&" | "'" | '"' | <operator> | <punctuation>
<punctuation>	::= "?" | "." | "," | ";" | "`" | "!" | "^" | "~"
<boolean>	::= "TRUE" | "FALSE"
<type>	::= "double" | "int" | "long" | "char" | "signed" | "struct" | "enum" | "union" | "float" | "short" | "unsigned"
<keyword>	::= "auto" | "break" | "else" | "switch" | "case" | "register" | "typedef" | "extern" | "return" | "continue" | "for" | "void" | "do" | "if" | "static" | "while" | "default" | "goto" | "sizeof" | "volatile" | "const" |
<operator>	::= "+" | "-" | "*" | "/" | "%" | "++" | "--" | "=" | "+=" | "-=" | "=" | "/=" | "%=" | "==" | ">" | "<" | "!=" | ">=" | "<=" | "&&" | "||" | "!" | "&" | "^" | "~" | "<<" | ">>" | "|" | "?:" | "<<=" | ">>=" | "&=" | "^=" | "|=" | "->"
<string>	::= <char> | <char><string> | <symbol> | <symbol><string> | <integer> | <integer><string> | <double> | <double><string> | | <espaco-opt><string> | | <boolean><string> | <keyword> | <keyword><string> | ""

- **Palavra-chave:** Tokens com funções reservadas dentro do código. Definimos: auto, register, typedef, extern, static, sizeof.
- **Palavra-chave de fluxo:** Tokens com funções reservadas dentro do código voltadas exclusivamente a como o código está sendo lido (laços de repetição, iteração, quebra de repetição, etc). Aqui registramos: break, else, switch, case, return, continue, for, do, if, while, default, goto.
- **Palavra-chave de tipo de dado:** Tokens com funções reservadas dentro do código voltadas exclusivamente a definir tipos de identificadores. Definimos: double, int, char, struct, enum, union, float, void.
- **Palavra-chave de modificador de dado:** Tokens com funções reservadas dentro do código voltadas exclusivamente a modificar valores de variáveis. Definimios: long, signed, short, unsigned, volatile, const.
- **Valor booleano:** Tokens que podem assumir dois valores exclusivamente: True ou False.
- **Identificadores:** Segmento de texto que pode assumir funções variáveis dentro do código.
- **Pontuações:** Definidas aqui como ? . , ; ‘ ! ^
- **Símbolos:** Definidos aqui como @ # & : _ ( ) [ ] { } ’ \
- **Operadores:** Tokens referentes a operadores matemáticos. Definimos: + - * /
- **Outros operadores:** Tokens referentes a operadores lógico-matemáticos.

As mudanças entre os tokens e a gramática definida anteriormente foram realizadas para aumentar o reconhecimento na tabela de símbolos.

Na função main, lemos o arquivo para conferir seus tokens, chamamos a função yylex(); e printamos o total de linhas do código, bem como o total de tokens existentes.

A seguinte sequência de comandos deve ser realizada no terminal para que o arquivo seja executado (considerando o arquivo em C *"codigo-exemplo.c"* fornecido pelo professor) e então impressos todos os tokens existentes no código, bem como sua localização na tabela de símbolos.

flex trab1.lex
gcc lex.yy.c -lfl -o trab1
./trab1 codigo-exemplo.c

3. Destaque do ambiente

Esse projeto foi compilado utilizando um processador *Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz*, com o sistema operacional *Linux Ubuntu 20.04 focal*, utilizando a versão do compilador *lex 1.5.003*.

4. Tabela de símbolos

Toda a tabela de símbolos printada pelo terminal ao utilizar os comandos acima está mostrada nas imagens de números 2 a 8 a seguir.

5. Considerações finais

Agradecimentos ao Professor Ricardo Ferreira Martins pela disponibilização dos materiais utilizados como base para o estudo feito nesse trabalho.

Figure 2. IDs: 1 ao 40.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
0	#	Outros símbolos	0	0
1	include	Identificador	0	1
2	<	Outro operador	0	9
3	stdio	Identificador	0	10
4	.	Pontuação	0	15
5	h	Identificador	0	16
6	>	Outro operador	0	17
7	#	Outros símbolos	1	0
8	include	Identificador	1	1
9	<	Outro operador	1	9
10	stdlib	Identificador	1	10
11	.	Pontuação	1	16
12	h	Identificador	1	17
13	>	Outro operador	1	18
14	#	Outros símbolos	2	0
15	include	Identificador	2	1
16	<	Outro operador	2	9
17	locale	Identificador	2	10
18	.	Pontuação	2	16
19	h	Identificador	2	17
20	>	Outro operador	2	18
21	#	Outros símbolos	3	0
22	define	Identificador	3	1
23	TAM	Identificador	3	8
24	10	Inteiro	3	12
25	int	Tipo de dado	5	0
26	main	Identificador	5	4
27	(	Outros símbolos	5	8
28	)	Outros símbolos	5	9
29	{	Outros símbolos	6	0
30	setlocale	Identificador	7	0
31	(	Outros símbolos	7	9
32	LC	Identificador	7	10
33	_	Outros símbolos	7	12
34	ALL	Identificador	7	13
35	,	Pontuação	7	16
36	"	Outros símbolos	7	18
37	"	Outros símbolos	7	19
38	)	Outros símbolos	7	20
39	;	Pontuação	7	21
40	int	Tipo de dado	8	0

Figure 3. IDs: 41 ao 80.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
41	numeros	Identificador	8	4
42	[	Outros símbolos	8	11
43	TAM	Identificador	8	12
44	]	Outros símbolos	8	15
45	;	Pontuação	8	16
46	int	Tipo de dado	9	0
47	i	Identificador	9	4
48	,	Pontuação	9	5
49	aux	Identificador	9	7
50	,	Pontuação	9	10
51	contador	Identificador	9	12
52	;	Pontuação	9	20
53	printf	Identificador	11	0
54	(	Outros símbolos	11	6
55	"	Outros símbolos	11	7
56	Entre	Identificador	11	8
57	com	Identificador	11	14
58	dez	Identificador	11	18
59	n	Identificador	11	22
60	meros	Identificador	11	23
61	para	Identificador	11	29
62	preencher	Identificador	11	34
63	o	Identificador	11	44
64	array	Identificador	11	46
65	,	Pontuação	11	51
66	e	Identificador	11	53
67	pressione	Identificador	11	55
68	enter	Identificador	11	65
69	ap	Identificador	11	71
70	s	Identificador	11	73
71	digitar	Identificador	11	75
72	cada	Identificador	11	83
73	um	Identificador	11	88
74	:	Outros símbolos	11	90
75	n	Identificador	11	91
76	"	Outros símbolos	11	92
77	)	Outros símbolos	11	93
78	;	Pontuação	11	94
79	for	Fluxo	13	0
80	(	Outros símbolos	13	4

Figure 4. IDs: 81 ao 120.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
81	i	Identificador	13	5
82	=	Outro operador	13	7
83	0	Inteiro	13	9
84	;	Pontuação	13	10
85	i	Identificador	13	12
86	<	Outro operador	13	14
87	TAM	Identificador	13	16
88	;	Pontuação	13	19
89	i	Identificador	13	21
90	++	Outro operador	13	22
91	)	Outros símbolos	13	24
92	{	Outros símbolos	13	26
93	scanf	Identificador	14	0
94	(	Outros símbolos	14	5
95	"	Outros símbolos	14	6
96	%	Outro operador	14	7
97	d	Identificador	14	8
98	"	Outros símbolos	14	9
99	,	Pontuação	14	10
100	&	Outros símbolos	14	12
101	numeros	Identificador	14	13
102	[	Outros símbolos	14	20
103	i	Identificador	14	21
104	]	Outros símbolos	14	22
105	)	Outros símbolos	14	23
106	;	Pontuação	14	24
107	}	Outros símbolos	15	0
108	printf	Identificador	17	0
109	(	Outros símbolos	17	6
110	"	Outros símbolos	17	7
111	Ordem	Identificador	17	8
112	atual	Identificador	17	14
113	dos	Identificador	17	20
114	itens	Identificador	17	24
115	no	Identificador	17	30
116	array	Identificador	17	33
117	:	Outros símbolos	17	38
118	n	Identificador	17	39
119	"	Outros símbolos	17	40
120	)	Outros símbolos	17	41

Figure 5. IDs: 121 ao 160.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
121	;	Pontuação	17	42
122	for	Fluxo	19	0
123	(	Outros símbolos	19	4
124	i	Identificador	19	5
125	=	Outro operador	19	7
126	0	Inteiro	19	9
127	;	Pontuação	19	10
128	i	Identificador	19	12
129	<	Outro operador	19	14
130	TAM	Identificador	19	16
131	;	Pontuação	19	19
132	i	Identificador	19	21
133	++	Outro operador	19	22
134	)	Outros símbolos	19	24
135	{	Outros símbolos	19	26
136	printf	Identificador	20	0
137	(	Outros símbolos	20	6
138	"	Outros símbolos	20	7
139	%	Outro operador	20	8
140	4	Inteiro	20	9
141	d	Identificador	20	10
142	"	Outros símbolos	20	11
143	,	Pontuação	20	12
144	numeros	Identificador	20	14
145	[	Outros símbolos	20	21
146	i	Identificador	20	22
147	]	Outros símbolos	20	23
148	)	Outros símbolos	20	24
149	;	Pontuação	20	25
150	}	Outros símbolos	21	0
151	for	Fluxo	23	0
152	(	Outros símbolos	23	4
153	contador	Identificador	23	5
154	=	Outro operador	23	14
155	1	Inteiro	23	16
156	;	Pontuação	23	17
157	contador	Identificador	23	19
158	<	Outro operador	23	28
159	TAM	Identificador	23	30
160	;	Pontuação	23	33

Figure 6. IDs: 161 ao 200.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
161	contador	Identificador	23	35
162	++	Outro operador	23	43
163	)	Outros símbolos	23	45
164	{	Outros símbolos	23	47
165	for	Fluxo	24	0
166	(	Outros símbolos	24	4
167	i	Identificador	24	5
168	=	Outro operador	24	7
169	0	Inteiro	24	9
170	;	Pontuação	24	10
171	i	Identificador	24	12
172	<	Outro operador	24	14
173	TAM	Identificador	24	16
174	-	Operador	24	20
175	1	Inteiro	24	22
176	;	Pontuação	24	23
177	i	Identificador	24	25
178	++	Outro operador	24	26
179	)	Outros símbolos	24	28
180	{	Outros símbolos	24	30
181	if	Fluxo	25	0
182	(	Outros símbolos	25	3
183	numeros	Identificador	25	4
184	[	Outros símbolos	25	11
185	i	Identificador	25	12
186	]	Outros símbolos	25	13
187	>	Outro operador	25	15
188	numeros	Identificador	25	17
189	[	Outros símbolos	25	24
190	i	Identificador	25	25
191	+	Operador	25	27
192	1	Inteiro	25	29
193	]	Outros símbolos	25	30
194	)	Outros símbolos	25	31
195	{	Outros símbolos	25	33
196	aux	Identificador	26	0
197	=	Outro operador	26	4
198	numeros	Identificador	26	6
199	[	Outros símbolos	26	13
200	i	Identificador	26	14

Figure 7. IDs: 201 ao 240.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
201	]	Outros símbolos	26	15
202	;	Pontuação	26	16
203	numeros	Identificador	27	0
204	[	Outros símbolos	27	7
205	i	Identificador	27	8
206	]	Outros símbolos	27	9
207	=	Outro operador	27	11
208	numeros	Identificador	27	13
209	[	Outros símbolos	27	20
210	i	Identificador	27	21
211	+	Operador	27	23
212	1	Inteiro	27	25
213	]	Outros símbolos	27	26
214	;	Pontuação	27	27
215	numeros	Identificador	28	0
216	[	Outros símbolos	28	7
217	i	Identificador	28	8
218	+	Operador	28	10
219	1	Inteiro	28	12
220	]	Outros símbolos	28	13
221	=	Outro operador	28	15
222	aux	Identificador	28	17
223	;	Pontuação	28	20
224	}	Outros símbolos	29	0
225	}	Outros símbolos	30	0
226	}	Outros símbolos	31	0
227	printf	Identificador	33	0
228	(	Outros símbolos	33	6
229	"	Outros símbolos	33	7
230	nElementos	Identificador	33	8
231	do	Fluxo	33	19
232	array	Identificador	33	22
233	em	Identificador	33	28
234	ordem	Identificador	33	31
235	crescente	Identificador	33	37
236	:	Outros símbolos	33	46
237	n	Identificador	33	47
238	"	Outros símbolos	33	48
239	)	Outros símbolos	33	49
240	;	Pontuação	33	50

Figure 8. IDs: 241 ao 280.

ID	DESCRIÇÃO	TIPO	LINHA	COLUNA
241	for	Fluxo	34	0
242	(	Outros símbolos	34	4
243	i	Identificador	34	5
244	=	Outro operador	34	7
245	0	Inteiro	34	9
246	;	Pontuação	34	10
247	i	Identificador	34	12
248	<	Outro operador	34	14
249	TAM	Identificador	34	16
250	;	Pontuação	34	19
251	i	Identificador	34	21
252	++	Outro operador	34	22
253	)	Outros símbolos	34	24
254	{	Outros símbolos	34	26
255	printf	Identificador	35	0
256	(	Outros símbolos	35	6
257	"	Outros símbolos	35	7
258	%	Outro operador	35	8
259	4	Inteiro	35	9
260	d	Identificador	35	10
261	"	Outros símbolos	35	11
262	,	Pontuação	35	12
263	numeros	Identificador	35	14
264	[	Outros símbolos	35	21
265	i	Identificador	35	22
266	]	Outros símbolos	35	23
267	)	Outros símbolos	35	24
268	;	Pontuação	35	25
269	}	Outros símbolos	36	0
270	printf	Identificador	37	0
271	(	Outros símbolos	37	6
272	"	Outros símbolos	37	7
273	n	Identificador	37	8
274	"	Outros símbolos	37	9
275	)	Outros símbolos	37	10
276	;	Pontuação	37	11
277	return	Fluxo	38	0
278	0	Inteiro	38	7
279	;	Pontuação	38	8
280	}	Outros símbolos	39	0