

Construção de um Analisador Léxico: Estrutura e Funcionamento

Alunos:

- Giovanni dos Santos 13695341
- Guilherme Augusto Fincatti da Silva 13676986
- Marcelo Eduardo Reginato 13676965
- Pedro Guilherme de Barros Zenatte 13676919
- Rhayna Christiani Vasconcelos Marques Casado 13676429

Professor: Thiago Alexandre Salgueiro Pardo

São Carlos, 2025

Conteúdo

1	Obj	etivo	1
2	Mat	teriais e Métodos	1
3	Res	ultados	4
	3.1	Descrição dos Autômatos	4
	3.2	Especificação dos Tokens	5
	3.3	Tabela de Transição de Estados	6
	3.4	Resultado da Implementação	8
	3.5	Execução do Analisador Léxico	11
4	Disc	cussão e Conclusão	11
5	Fon	tes	12

1 Objetivo

O principal objetivo deste trabalho é desenvolver um analisador léxico para a linguagem PL/0, capaz de ler um programa fonte escrito em um arquivo .txt e produzir como saída outro arquivo .txt contendo, linha a linha, o par token-classe correspondente a cada elemento reconhecido. O analisador também deve ser capaz de identificar e reportar eventuais erros léxicos encontrados no programa de entrada, por exemplo, o erro de identificador mal formado.

Para alcançar esse objetivo, foi necessário projetar e implementar um autômato de estados finitos (AFD) capaz de reconhecer os diferentes padrões da linguagem, bem como definir uma tabela de transição de estados que orientasse o processamento dos símbolos lidos, além de ser a nossa tabela de transições para a implementação do próprio código. Além disso, decisões importantes de projeto foram tomadas, tais como a estratégia de retrocesso no automato (lookahead), a forma de representar as classes dos tokens e o tratamento adequado para símbolos inválidos.

A correta implementação do analisador léxico é essencial para a construção de um compilador, pois representa a primeira fase da análise do códigofonte, preparando a estrutura de tokens que será utilizada nas etapas seguintes de análise sintática e semântica.

2 Materiais e Métodos

Para o desenvolvimento do analisador léxico, foi utilizada a linguagem de programação C, uma vez que possui elevada eficiência e controle de baixo nível sobre arquivos e estruturas de dados. O ambiente de desenvolvimento utilizado foi o Visual Studio Code junto ao GitHub para controle de versões e colaboração entre os membros da equipe. Esse repositório online permitiu organizar o código, acompanhar as alterações realizadas e integrar o trabalho de forma eficiente e segura.

Além disso, para auxiliar na construção e validação dos autômatos finitos determinísticos (AFDs) com saída, utilizamos a ferramenta JFLAP, que permitiu a criação gráfica dos autômatos e facilitou a verificação dos comportamentos esperados durante a leitura dos tokens. Sendo assim, a partir dos autômatos modelados foi possível construir a tabela de transições, a qual descreve as mudanças de estado para cada símbolo lido.

A implementação do analisador léxico foi desenvolvida com base nesses autômatos, realizando a leitura do arquivo de entrada caractere por caractere e direcionando-o ao estado apropriado conforme o símbolo lido. Para lidar com símbolos compostos e ambiguidades, foram adotadas estratégias como o retrocesso (lookahead), permitindo a análise correta de padrões que exigem a leitura do próximo caractere para a tomada de decisão. Um exemplo disso são os identificadores, no qual é necessário analisar o último caractere, mas sem incluí-lo no par token-classe.

Cada token identificado é associado a uma classe descritiva, e o par tokenclasse é escrito no arquivo de saída. Dessa forma, em casos de erro léxico, como a presença de símbolos inválidos na sequência de um determinado estado, o programa registra o erro na saída com a anotação apropriada, por exemplo, ERRO_NUMERO_MAL_FORMADO em caso de letras quando estamos formando dígitos.

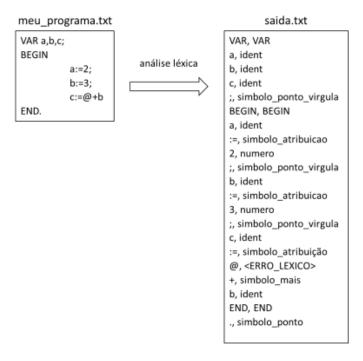


Figura 1: Exemplo de token-classe na saída do programa

Note que, para os lexemas correspondentes a palavras reservadas, isto é, as palavras brutas escritas em maiúsculo, como VAR, o par token-classe é formado pelo próprio lexema.

Durante o projeto, foram necessárias diversas decisões de implementação para lidar com questões desse tipo, como a definição do tratamento das pala-

vras reservadas, a implementação do autômato em si, o formato da tabela de transições, a criação de vetores auxiliares para seu correto funcionamento, a estratégia de armazenamento dos tokens lidos e a estrutura de controle dos estados do autômato, sempre visando garantir clareza, facilidade de manutenção e robustez no tratamento dos dados.

Em relação às palavras reservadas, optamos por não criar um autômato específico para cada uma delas, o que tornaria o autômato excessivamente grande, poluído e ineficiente. Em vez disso, implementamos uma tabela hash, aproveitando sua alta eficiência de busca. Assim sendo, sempre que o programa identifica um token como um identificador, ele consulta a tabela para verificar se o lexema corresponde a uma palavra reservada. Caso corresponda, o token identificado é então classificado corretamente como uma palavra reservada ao invés de um idenficador.

Quanto à implementação do autômato, decidimos utilizar uma matriz de transição para evitar uma solução "ad hoc" e tornar o código o mais elegante e estruturado possível.

Quando nos referimos à tal matriz de transição no código, estamos falando de uma matriz de inteiros, onde cada linha representa um estado e cada coluna representa um símbolo. No entanto, ao invés de criar uma matriz gigantesca, incluindo todos os caracteres do alfabeto (a-z, A-Z), todos os dígitos (0-9) e todos os símbolos especiais, o que, além de trazer mais trabalho para associar um caractere a um inteiro, deixaria o código muito poluído, optamos por uma solução que consideramos mais eficiente e que torna o código mais organizado, isto é, criamos um vetor de caracteres contendo apenas os símbolos que o analisador léxico reconhece.

Desse modo, com o apoio de uma função auxiliar, associamos cada símbolo desse vetor a uma coluna específica da matriz de transição, economizando espaço.

Por exemplo, o vetor de símbolos, um vetor de caracteres, guarda todos os caracteres reconhecidos. As posições de 0 até 51 no vetor correspondem às letras (a-z e A-Z), mas todas essas letras são tratadas como pertencentes à mesma coluna (coluna 0) na matriz de transição, uma matriz de inteiros. Dessa forma, a função auxiliar é responsável por mapear essas posições do vetor para o índice 0 da matriz. O mesmo raciocínio é aplicado aos dígitos e símbolos especiais.

Contudo, para cada símbolo especial, por sua vez, temos apenas um ajuste, isto é, sua posição no vetor é deslocada (shiftada) de forma que, por exemplo, o símbolo ':' que ocupa a posição 62 no vetor, é associado à coluna 2 na matriz de transição. Esse processo continua para todos os símbolos reconhecidos, garantindo uma representação compacta e eficiente.

Portanto, com a matriz de transição podemos garantir um código mais in-

dependente do autômato e mais bem organizado, tomando sempre o cuidado de colocar o vetor de símbolos em sincronia com a matriz de transição.

Uma última decisão importante foi o tratamento dos estados finais. Para isso, adotamos a convenção de representaá-los como constantes negativas. Dessa forma, sempre que a função de transição retorna um valor menor que zero, sabemos que chegamos a um estado final. Para distinguir entre os diferentes estados finais, associamos cada constante negativa a uma posição específica em um vetor de strings que descreve os estados finais. Por exemplo, o estado TK_ID é representado pelo valor -3, o qual corresponde à posição 2 no vetor de estados finais (calculada como -3 * (-1) - 1).

3 Resultados

O desenvolvimento do analisador léxico envolveu diversas etapas que se complementam para a construção de um sistema capaz de identificar corretamente os elementos da gramática da linguagem PL/0. Inicialmente, foram elaborados os autômatos responsáveis pelo reconhecimento dos diferentes padrões léxicos, considerando as particularidades específicas de cada token. A partir desses autômatos, consolidou-se uma especificação formal dos tokens, reunindo em uma tabela suas expressões regulares, representações e categorias associadas.

Sendo assim, com base na estrutura dos autômatos e nessas especificações dos tokens, foi possível construir a tabela de transição de estados, inicialmente elaborada como uma tabela separada, antes de ser implementada no código, a fim de melhor organização e visualização, que orienta a leitura e o processamento dos símbolos durante a análise. Por fim, com essas ferramentas foi viável implementar o código e, após a implementação do analisador, exemplos práticos de entrada e saída ilustram o funcionamento do sistema, evidenciando a identificação correta dos tokens e a detecção adequada de erros léxicos.

3.1 Descrição dos Autômatos

A base do funcionamento do analisador léxico é formada pelos autômatos finitos determinísticos com saída, neste caso utilizamos as Máquinas de Moore, elaborados para reconhecer os diferentes tokens da linguagem PL/0. Cada autômato foi projetado considerando os padrões léxicos a serem reconhecidos pela gramática e validado com o apoio da ferramenta JFLAP.

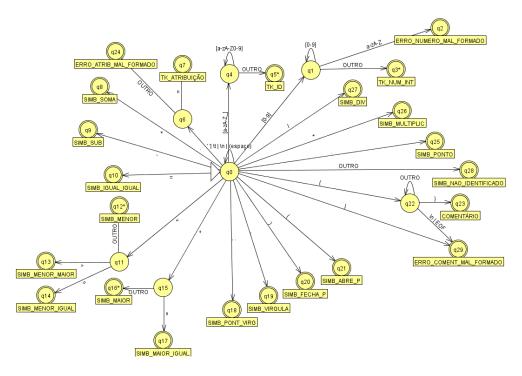


Figura 2: Autômato para Analisador Léxico

3.2 Especificação dos Tokens

Com os autômatos definidos, foi possível formalizar a especificação dos tokens aceitos.

A tabela a seguir apresenta as expressões regulares associadas a cada token, sua representação simbólica e a categoria correspondente, servindo como referência para a análise léxica.

Expressão Regular	Token	Categoria do Token
D = [0-9]		
$\mathbf{L} = [\mathbf{a}\text{-}\mathbf{z}\mathbf{A}\text{-}\mathbf{Z}_{_}]$		
$NUM_INT = D +$	TK_NUM_INT	Número inteiro (número)
$IDENT = L (L \mid D)^*$	TK_ID	Identificador
CONST = CONST	TK_CONSTANTE	Palavra reservada
VAR = VAR	TK_VAR	Palavra reservada
PROCEDURE = PROCEDURE	TK_PROCEDURE	Palavra reservada
$\mathrm{CALL} = \mathrm{CALL}$	TK_CALL	Palavra reservada
BEGIN = BEGIN	TK_BEGIN	Palavra reservada
$\mathrm{IF}=\mathrm{IF}$	TK_IF	Palavra reservada
$\mathrm{THEN} = \mathrm{THEN}$	TK_THEN	Palavra reservada
$\mathrm{WHILE} = \mathrm{WHILE}$	TK_WHILE	Palavra reservada
DO = DO	TK_DO	Palavra reservada
ODD = ODD	TK_ODD	Palavra reservada
:=	TK_ATRIBUIÇÃO	Operador de atribuição
+	SIMB_SOMA	Operador aritmético unitário
-	SIMB_SUB	Operador aritmético unitário
=	$SIMB_IGUAL_IGUAL$	Operador relacional
<>	SIMB_MENOR_MAIOR	Operador relacional
<	SIMB_MENOR	Operador relacional
<=	SIMB_MENOR_IGUAL	Operador relacional
>	SIMB_MAIOR	Operador relacional
>=	$SIMB_MAIOR_IGUAL$	Operador relacional
(SIMB_ABRE_P	Símbolo especial
)	SIMB_FECHA_P	Símbolo especial
,	SIMB_VIRGULA	Símbolo especial
;	SIMB_PONT_VIRG	Símbolo especial
	SIMB_PONTO	Símbolo especial
*	SIMB_MULTIPLIC	Símbolo especial
/	SIMB_DIV	Símbolo especial
$COMENTARIO = \{.*\}$		Comentário

Tabela 1: Especificação dos Tokens

3.3 Tabela de Transição de Estados

A partir da estrutura dos autômatos, foi construída uma tabela de transição de estados, que define o comportamento do analisador a cada novo símbolo lido.

Essa tabela orienta a movimentação entre estados e garante a correta identificação dos padrões esperados, além de auxiliar o programador a enxergar melhor o que está acontecendo na matriz de transição elaborada na implementação.

Para melhor visualização da tabela, voce pode acessar o link da tabela por aqui: Tabela de Transições

Como exemplo, podemos analisar a função de transição para a fita $\mathbf{A}\mathbf{A}\mathbf{B}+:$

(a) operso	2	Ollono	2			۰							_	_	_	~			5	oppdea in	•			-	5	Caractere Estrangeiro (1,)	angeiro (.,	ī
	*		+		96		ø		ĺ				31	20	22	8	0	0	0	0	25	N	r	27	0			8
	*	~	-										9		9			9				**		n	n			**
	ERRO NUMERO MAL	FOR ANY LESSO, NAMES ON ANY CROSS ANALES FRED, NAMES F	DRIM ERRO, N	UNER ERRO	NUMBER ER	RO_NUMER D	TRRO NUMBR	SERRO NUMB	R ERRO NUME	R ERRO NUMB	R ERRO NUMB	R ERRO NUM	ER ERRO NUM	ER ERRO NUM	KIN CHRO NUN	WER STRO NO	INER ERRO NU	WERD JA ERS	AD NUMBER ER	TRO NUMBER &	SWIM CHRE	MUN CHRIS	MUM CHRIS	AN CHORD ONLY	LIMERO ERRIC	NUMERO MAL FORM	400	_
•	TICHUILDIT	TICHUMUNT	TICHING	UNIT TACK	UNLINT TR.	NUMBER T	KONDINONT	TICHUMONT	TICHUMUNT	TICHUMUNT	TICHUMON	TKOMINON !	T TICHUMON	T TRONUMON	II TICHUMON	OT TROBBE	INT TICKURANT TI		NUMBER TA	CHIMONT	TK NUM INT	TK NUM IN	TICKURUNT TICKURUNT TICKUM INT TICKUM INT TICKUM INT	IT IN NUM INT		TK NUM INT		
•	•	_	*	17	47	47	47		10	10	pir	49	10	**	42	47	47	17	47	107	47	41		40	'n			40
	O XI	10,00	100	TK, IB	Ä	9	95%	18,00	TK, IB	18,00	100	18,00	100	1K,00	18,00	18,00	TK, ID	×	ğ	5	IK ID	IK ID	IK ID	IK ID	IK IE			
	26		z	70	4	70	34		-			N.	24	24	34	70	34	34	34	35	7	ñ		ä	25			ă
	TK_ATRIBUÇÃO	TK_ATRIBUIÇÃO	TK_ATRIA	BUICK TR. AL	RBUCA TK.	ATRIBUIÇĂ TA	K ATRIBUIÇĂ	TR. ATRIBUIÇ	A TR. ATRIBUIÇ	A TK ATRIBUIG	A TICATRIBUG	A TK ATRIBUK	CA TR. ATRIBUA	CA TR. ATRIBUA	CA TRAFFIRM	IICA TK. ATRIB.	UCA TR. ATRIBL	uicko nk.	ATRIBUIÇĂ TA	CATRIBUCAL	N. ATRIBUIÇĂ	TK ATRIBUC	TRATEBUR	TKATRBOUGH	sucko TKAN	spucko		
•	SIND SCHALL	SIMB_SOMA	SIMB_SOMA	THE SIME	SIND SOMA SIN	SIND SOME SI	SIMB_SOMA	SIMB_SOMA	SIND SOMA	SIMB_SOMA	SIND SOMA	SIMB_SOMA	A SIMB SOMA	A SIMB_SOMA	A SIND SOMA	A SIMB SOMA	MA SIND SOMA		SIND SOMA SIL	SIMP_SOMA S	SIND SOMA	SIMB_SOMA	SIMP, SOMA	SIND SOMA	NA SIND	SOMA		
	SIVE SIVE	SIMB SIMB	SAME SUB	UB SIMB	SAME SUB. SAN	SAME SUB S	SIVE SUB	SIMB SIMB	SINE SUB	SIMB BUB		SIMB SUB	SINE SUB	SIMB SUB	SIMB SUE	3 SAME SU	SIVE SIVE SIVE SIVE SIVE SIVE SIVE SIVE		SIVE SUB SI	SAME SUB	SINE SUB	SIMB SUB	SIMB BUB	SINE SILE	SWE SWE	SUB		
2	SIMB IQUAL IQUAL	SIMB ICLIAL ICLIAL	SIMB IC.	SIVE ICLIAL SIVE ICLIAL SIVE ICLIAL	JOHN SIN	REPORT S	SIMB ICLIAL	SIMB IGUAL	SIMB ICLAL	SWB IGUAL	SINB ICLA	L SIMB IGUA	AL SINE ICLA	AL SIMB ICALA	AL SIMB ICAL	AL SINB IC.	SINB IGLIAL SINB IGLIAL.	THE YOUR SIN	IS IQUAL SA	WE KALAL S	SIMB ICLIAL	SIMB IGUAL	SWB IGLAL	SINB IGUAL ICI SINB IGUAL ICI SINB IGUAL	UAL ICL SINE	IGUAL IGUAL		
F	42		÷	ņ	2	Ç	0					9	2	ç	0	ç	ç	Ç	Ç	Ç	Ç	7		p	p			Ç
ş	SIMB MENOR	SIMB MENOR	SWB W.	ENO! SIMB	MENOF SIN	NB MENOF S.	WENCH	SIMB MENG	H SIMB MENC	H SIMB MENG	+ SIMB MENG	THE MEN	OF SIMB MEN	OF SWB MEN	OF SWB NEX	VOF SIMB ME	SINE MENCH	NOR SIN	IS NENO+ St.	WE MENOF 5	SWB MENOF	SWB MENO	SWB MENO	THE SAME MENOR		SIMB MENOR		_
p	SIMB MENOR MAIO	INB MENCH MAIOL SIMB MENCH MAIOR	SWB W	TENO! SIVE	MENOF SIN	VIB. MENOF S	THE MENO	SIMB MENG	H SIMB MENC	H SWB MENG	* SIMB MEN	THE MEN	OF SWB NEW	OF SWB WEN	OF SWB NEY	NOT SIMB ME	ONCH SIMB ME	MOR MUSEN	IB MENOF SI	WE MENOF 5	SIMB MENO!	SWB NENO	1 SWB MENO	SINE MENCH	SNOR N SIMB	MENOR MADR		_
2	SIMB MENDICIDAR	AB MENOR IGLIAI SIMB MENOR IGLIAL	SIMB M	TENO! SIMB	MENOF SIN	VIB MENOF S	WENCH	SIMB MENG	H SIMB MENC	H SIMB MENG	F SIMB MENG	NEW BINE NEW	OF SIMB MEN	OF SWB NEW	OF SIMB MES	NOT SIMB ME	INDEBNE NE	MOR IQ SIN	IB MENOF SA	WE MENDY S	SIMB MENO?	SWB NEND	1 SIMB MENO	SINE MENCH K SINE MENCH K SINE MENCH K SINE MENCH K SINE MENCH K SINE MENCH K SINE MENCH	SNOR K SIMB	MENDIC IQUAL		
ņ	91		10	2	4	2	*			,		9				2	*	*	*	8	8	44		10	46			*
2	SIMB MADR	SIMB MAIOR	SIMB M	NACH SIME	MAJOR SIN	VIB MAJOR S.	WALL MAJOR	DAME MAD	R SIMB MAID.	R SIMB MAIO	CENTE MAIO	R SIMB MAK	OR SIMB MAIL	OR SIMB MAC	OR SIMB NA	OR SIMB MA	NOR SIMB MA	ADR SIN	IS MAJOR SI	WE MADE S	SIMB MAIOR	SIMB MAIOR	SIMB MADE	SINE MADE	NOR SIME	MAIDR		
	SIMB MAIOR IGUAL	INB MADE IGUAL SIMB MADE IGUAL	SIMB MA	NACK SIME	MAJOR SIN	UB MAJOR S.	NUMB MAJOR	SIMB MAIO	R SIMB MAIO.	R SIMB MAIO	CEMB MAID	R SIMB MAK	OR SIMB MAIL	OR SIMB MAK	OR SIMB MAN	OR SIMB MA	NOR SIMB MA	WOR YOU SHE	IS MAJOR SI	WE MADR 5	SIMB MAIOR	SIMB MAIOR	SIMB MADE	SINE MACK SINE M	NOR IG SINE	MADR IGUAL		
2	SIMB FONT VING	SIMB FONT VING	SMB	SMB FONT SMB FONT	PONT SA	SIMB PONT S	TWO LOW	NO. SWE	SWB FONT	SMB PONT	SIMB FOND	NOT BMB	T SWB PON	T SWB PON	T SWB PON	VI SMB PG	ON SMB IN	WE SHY IN	B PONT S	WE PONT 'S	SMB PONT	SWB PONT	SMB PONT	SAME FONT:	WI VIR SIME	PONT_VING		_
;	SIMB VINGULA	SIMB VINGULA	SIMB VI	THOU SWE	WHICH SA	WE VINCUES	WHE VINCUA	T SWE VINC.	A SWE VINC.	A SIMB VINCE	A SIMB, VING.	UL SIMB VING	THE SIME VING	THE SAME VING	THE SAME VIEW	DUI SIMB VIII	SIME VHICLE SIME V	TOUR SIL	IB VINGUES.	WE VINCUE	SWE VHOU	SIMB VINCE	SME VHCU SME VHCU SME VHCU SME VHCU SME VHCU.A.	ILA SIMB VINGULA		SIMB VINGULA		
2	SIMB FECHA P	SIMB FECHA P	SWB FL	BOHA SIMB	PECHA SIA	WE FECHA S	WAS FECHA	A SIMB, FECH	A SIMB FECH.	A SMB FECH	A SIMB FECH	W SWB FECK	HA SIMB FECK	HA SIMB FECK	HA SIMB FEC.	HA SMB FB	CHA SIMB FEL	CHA.P. SIN	IB FECHA SE	WE FECHAS	SMB FECHA	SIMB FECH	1 SWB FECH	SING FICHA SING FECHA	CHA P SMB	FECHA P		
ä	SIMB ABRE P	SIMB ABRE P	SIMB AL	BYE SWB	ABRE I SIN	AB ABRE 18	WINE ABRE	SIMB ABRE	SIMB ABRE	SIMB ABRE	SIMB ABRE	U SIMB ABRO	E I SIMB ABRO	E I SIMB ABRO	E I SIMB ABN	TE I SIMB AB	SING ABITE (SING A	HE P SIN	IB ABRE 150	MB ABRE 1	SIMB ABRE	SIMB ABRE	SIMB ABRE	P SIMB ABIDE P	SPE P SIMB	ABRE P		_
8	22		22	22	22	22	33		22 2	22 22	22	33	22	33	22	55	33	22	8	33	22	200		2	20			22
a	COMENTARIO	COMENTARIO	COMEN	TARK COM.	ENTARK CO.	MENTARK C	COMENTARIC	COMENTAR	C COMENTAR	IC COMENTAR.	CCOMENTAR	W COMENTAL	UC COMENTAS	NC COMENTAL	NIC COMENTA	VINC COMENIA	COMENTARIC		MENTARIC CL	DMENTARIC (COMENTARIC	COMENTARY	COMENTARIC COMENTARIC COMENTARIC COMENTARIC COMENTARIO	IO COMENTARIO		COMENTARIO		_
2	EFFED ATHUS MAL P	FERRI ATHER MAL FOR	A CHIEF MAY	ATHUB EPONE.	ATHUS DR	HD ATHUB E	SPITE CHIE	BRID ATR	B EFFED ATHS.	B DRND ATHER	3 EPHID ATHS	HE EIGHD ATH	HE GREE ATR.	HIS CHOOL BY	US EFFED AT	HUB ERRED AN	THUS EVEND AS	THUS MA DR	HD ATHUB EN	RHD ATHUS &	SPITA CHIE	ERRO ATHUS	TEMB ATHER	B MUDDED AN	THUS MA EPORT.	ATHE END ATHE		_
12	SIME PONTO	SIMB_PONTO	SIME P.	ONTO SIME	PONTO SIA	VIB_PONTO S	WINE PONTS	SAME FOND	D SIMB, PONT	C SIME PONT.	D SIMB, PONT	NO SIMB FOR	TO SIMB FON	TO SAME INDIV	TO SIME POR	VIC SIMB_PC	SINS FONTS SINS FONTS BINS FONTS SINS FONTS		IB PONTO S.	WE PONTO 5	SIME PONTO	SIME PONT.	SIMB FONTO SIMB FONTO SIMB FONTO SIMB FONTO SIMB FONTO	O SIMB PONTO		SIMB PONTO		
2	SIMB MULTIPLIC	SIMB MULTIPLIC	SIMB M.	TULIN SIME	WILLIAM SAN	WE MULTIP S	WILLIAM WILLIAM	SIMB MULT	T SWE MULT	T SIMB MULT	THE MILE	THE WILL	TE SINE MILE	III SIMB MUL.	IN SIME MIL	UTF SIME M.	ALTE SIMB MIL	ATTITUDE SIN	IS MULTIP SA	WE MULTIPE	SWE MULTIP	SWE MULTI	SWE MILET	SHE NITE SHE	JULIANCE SIME	MULTIPLE		
ta ta	SAME DIV	SIMB DIV	SIMB DR	į,	SIMB DIV SIMB DIV		SIMB DIV	SIMB DIV	SIMB DIV	SWE DIV SWE DIV SWE DIV	SIMB DIV	SIMB DIV	SIMB DIV	SIMB DIV. SIMB DIV. SIMB DIV. SIMB DIV. SIMB DIV.	SIMB DIV	SWB DIV	SINB DIV. SINB DIV. SINB DIV. SINB DIV. SINB DIV. SINB DIV.	7 886	B DIV S	WE DIV	WE DIV	SIMB DIV	SIMB DIV	SWB DIV	VICENS V	DIV.		
a	SIMB NÃO IDENTIFE	CONCINENTE SING NOT CHAIR BANG II CHAIR BANG	W SWE OU	WO IL SIMB	NAO IL SIN	WE NAME IT S.	THE NAME IL	CEMB NAO	IL SIMB NAO	IL SIMB NAO	C SIMB NAO	IL SIMB NAO	IL SIMB NAO	IL SIMB NAO	O IL SIMB NAC	O IL SIMB NA	O IL SIMB NA	O IDENTISM	TE NÃO IL SE	WIS NAC IL S	TI OWN BAR	SIMB NAO L	TEMB NÃO L	IDEN SIMB NA	NO IDEN SIMB	NAO IDENTIFICADO		
	COUNTY OFFICE ASSESSMENT	the Advantage are 1000 decreased and 1000 and	. ^	-	A comment	200	The same					The second of	200000000000000000000000000000000000000	2000	200000		Section of the last	Annual Control	200	1	Section Section		Control of the last	Section Street, Section 5	Control of Control			_

Figura 3: Tabela de Transição

```
\delta^*(q_0, AAB+) = \delta^*(\delta(q_0, A), AB+)
= \delta^*(q_4, AB+)
= \delta^*(\delta(q_4, A), B+)
= \delta^*(q_4, B+)
= \delta^*(\delta(q_4, B), +)
= \delta^*(q_4, +)
= \delta^*(\delta(q_4, +), \varepsilon)
= \delta^*(q_5, \varepsilon).
```

O estado q_5 é o responsável por retornar o token TK_ID.

3.4 Resultado da Implementação

Com a implementação concluída, foram realizados testes utilizando programas de exemplo.

O analisador léxico gerou corretamente os pares token-classe no arquivo de saída, além de identificar e reportar eventuais erros léxicos, conforme ilustrado nos exemplos a seguir.

Exemplo 1: Comparação entre Arquivo de Entrada e Arquivo de Saída

9

Arquivo de Entrada:

VAR n, fat; PROCEDURE fatorial; BEGIN fat:1; WHILE n>=1 DO BEGIN fat:=fat!n; n:=n-1@; END END; BEGIN n:=4; CALL fatorial END.

Arquivo de Saída:

```
VAR, VAR
n, TK ID
"SIMB VIRGULA
fat, TK ID
;, SIMB PONT VIRG
PROCEDURE, PROCEDURE
fatorial, TK ID
;, SIMB_PONT_VIRG
BEGIN, BEGIN
fat, TK ID
ERRO ATRIB MAL FORMADO
1, TK NUM INT
;, SIMB PONT VIRG
WHILE, WHILE
n, TK ID
>=, SIMB MAIOR IGUAL
1, TK NUM INT
DO, DO
BEGIN, BEGIN
fat, TK_ID
:=, TK ATRIBUICAO
fat, TK ID
SIMB_NAO_IDENTIFICADO
n, TK ID
;, SIMB PONT VIRG
n, TK ID
:=, TK ATRIBUICAO
n, TK ID
-, SIMB SUB
Eu 1, TK NUM INT
SIMB NAO IDENTIFICADO
;, SIMB PONT VIRG
END, END
END, END
;, SIMB_PONT_VIRG
BEGIN, BEGIN
n, TK ID
:=, TK ATRIBUICAO
4, TK NUM INT
;, SIMB PONT VIRG
CALL, CALL
fatorial, TK ID
END, END
., SIMB PONTO
```

Exemplo 2: Comparação entre Arquivo de Entrada e Arquivo de Saída

10

Arquivo de Entrada:

```
VAR n, fat;
BEGIN
n:45!;
fat:=1;
WHILE n<>1 DO
BEGIN
fat:=fat*n;
n:=n-1;
n := n*1;
END {manfiosos}
END.
```

Arquivo de Saída:

```
VAR, VAR
n, TK ID
"SIMB VIRGULA
fat, TK ID
;, SIMB_PONT_VIRG
BEGIN, BEGIN
n, TK ID
ERRO ATRIB MAL FORMADO
45, TK_NUM_INT
SIMB NAO IDENTIFICADO
;, SIMB PONT VIRG
fat, TK ID
:=, TK ATRIBUICAO
1, TK NUM INT
;, SIMB_PONT_VIRG
WHILE, WHILE
n, TK ID
<>, SIMB_MENOR_MAIOR
1, TK NUM INT
DO, DO
BEGIN, BEGIN
fat, TK ID
:=, TK ATRIBUICAO
fat, TK ID
*, SIMB MULTIPLIC
n, TK ID
;, SIMB PONT VIRG
n, TK ID
:=, TK_ATRIBUICAO
n, TK ID
-, SIMB SUB
1, TK NUM INT
;, SIMB PONT VIRG
n, TK ID
:=, TK ATRIBUICAO
n, TK ID
*, SIMB MULTIPLIC
1, TK NUM INT
;, SIMB PONT VIRG
END, END
{manfiosos}, COMENTARIO
END, END
., SIMB PONTO
```

3.5 Execução do Analisador Léxico

Após a implementação do analisador léxico, definimos um procedimento simples para a sua execução utilizando Makefile.

Atenção

O programa foi desenvolvido em linguagem C e deve ser compilado com um compilador compatível, como o GCC.

Para compilar e executar o programa, basta seguir os passos descritos abaixo no terminal Linux:

- Acesse o diretório do projeto: cd Compiladores/Trabalho1/Código_Compiladores
- Caso deseje utilizar um arquivo de entrada diferente, coloque o arquivo na pasta /Código_Compiladores e altere o caminho do arquivo .txt diretamente na função main antes da compilação.
- Compile o programa utilizando o comando: make
- Execute o programa com: make run
- Observação: caso existam arquivos de compilação antigos, recomenda-se executar o comando: make clean antes de compilar novamente.

Observação

O resultado da execução será gerado no arquivo output.txt.

4 Discussão e Conclusão

O desenvolvimento do analisador léxico exigiu a superação de diversos desafios que surgiram ao longo do projeto. Desde o início, ficou claro que a construção dos autômatos para o reconhecimento dos tokens seria uma etapa central e fundamental para todo o desenvolvimento do projeto. Sendo assim, foi necessário compreender em detalhes as especificações da linguagem PL/0 e traduzir seus padrões léxicos em formalismos operacionais bem definidos.

Nesse processo, foi de suma importância a utilização da ferramenta JFLAP, pois possibilitou a modelagem gráfica dos autômatos e a validação de seus comportamentos antes da implementação no código, apesar que alguns dos símbolos essa ferramenta não reconhece, como a-zA-Z.

A partir dos autômatos construídos, elaboramos a tabela de transição de estados em uma planilha, o que permitiu organizar de maneira sistemática a lógica de movimentação entre estados a cada caractere lido, como podemos observar no exemplo realizado anteriormente com a fita $\mathbf{A}\mathbf{A}\mathbf{B}+$. Essa etapa foi essencial para garantir que, durante a implementação da tabela, o programador pudesse visualizar com clareza o comportamento de cada etapa do processo.

Durante a implementação, novos desafios se apresentaram, como a implementação de uma tabela hash para o tratamento de palavras reservadas, a necessidade de ter um olhar a frente a cada caractere e a própria função de transição.

De maneira geral, o projeto proporcionou uma experiência prática rica sobre a importância da análise léxica no processo de construção de compiladores e sobre como essa análise se desenvolve. Além disso, também houve a necessidade constante de planejamento e de atenção aos detalhes, especialmente relacionados à gramática da linguagem PL/0, o que evidenciou a relevância de decisões técnicas sólidas para a robustez do sistema. Nesse contexto, torna-se plausível depreender que a qualidade de um analisador léxico está diretamente ligada à firmeza de seus fundamentos. Portanto, como resultado desse trabalho, foi possível alcançar plenamente os objetivos propostos, obtendo-se um analisador funcional, eficiente e capaz de lidar de maneira apropriada com os diferentes elementos da linguagem.

5 Fontes

As principais referências utilizadas para a realização deste trabalho foram os materiais disponibilizados em sala de aula, isto é, os slides fornecidos pelo professor Thiago Alexandre Salgueiro Pardo, que abordam os conceitos de análise léxica e autômatos finitos.

Linguagem PL/0

Gramática da linguagem PL/0

```
Fonte: Wirth, Niklaus. (1976). Algorithms + Data Structures = Pro-
grams. Prentice Hall, Inc. (mais detalhes em Wikipedia).
<programa> ::= <bloco> .
<declaracao> ::= <constante> <variavel>  procedimento>
<constante> ::= CONST ident = numero <mais const> ;
<mais const> ::= , ident = numero <mais const>
\langle \text{variavel} \rangle ::= \text{VAR ident } \langle \text{mais var} \rangle ; \quad | \quad \lambda
<mais var> ::= , ident <mais var> 
cedimento> ::= PROCEDURE ident ; <bloco> ;
<procedimento> \lambda
<comando> ::= ident := <expressao>
  | CALL ident
   BEGIN < comando > < mais cmd > END
   IF <condicao> THEN <comando>
   WHILE < condicao > DO < comando >
  |\lambda|
<mais cmd> ::= ; <comando> <mais cmd>
<expressao> ::= <operador unario> <termo> <mais termos>
<operador unario> ::= - | + | \lambda
<termo> ::= <fator> <mais fatores>
<mais termos> ::= - <termo> <mais termos> | + <termo>
<mais termos> | \lambda
\langle \text{fator} \rangle ::= \text{ident} \quad | \quad \text{numero}
                               | (\langle expressao \rangle)|
<mais fatores> ::= * <fator> <mais fatores> | / <fator>
<mais fatores> \lambda
<condicao> ::= ODD <expressao>
  | <expressao> <relacional> <expressao>
<relacional> ::= = | <> | < | <= | >
```

Notas:

- Comentários são de única linha, entre chaves { }.
- Identificadores são formados por letras e dígitos, começando por uma letra.
- Só há números inteiros, formados por um ou mais dígitos (de 0 a 9).