Universidade Federal Fluminense
Niterói - 2022.2

Compiladores - Relatório Analisador Léxico

Lucas Fuzato, Tiago Lacerda e Viviane Romero

Projeto disponível em:

https://replit.com/@TiagoMLB/c-minus-compiler#.replit
https://github.com/TiagoLacerda/c-minus-compiler

1. Ferramentas Utilizadas
Toolbox - https://cyberzhg.github.io/toolbox/nfa2dfa
RegExr - https://regexr.com/
Python - https://www.python.org/
Visual Studio Code - https://code.visualstudio.com/
Replit - https://replit.com/~

2. Dificuldades e Problemas Enfrentados
Inicialmente, tivemos dificuldade em identificar tokens de comentários, pois estávamos quebrando a cadeia lida assim que um símbolo lido levava a um estado de não-aceitação. Contornamos este problema lendo a cadeia até o final e mantendo um ponteiro para o símbolo que gerava a cadeia aceita mais longa.

Tivemos também dificuldade em gerar uma representação visual do autômato completo gerado, uma vez que possui mais de 100 estados. Portanto, optamos por gerar representações visuais dos autômatos mais simples que o compõem.

3. Exemplo de Execução

```
main(void) {
    /* this is a very nice comment */
    int i = 0;
    int j = 8;
    while (i < 20)
    {
        if (i == j)
        {
            print(i);
        }
    }
    return 0;
}
```

```
main                              ['id']
(                                 ['open_parenthesis']
void                              ['void']
)                                 ['close_parenthesis']
{                                 ['open_brackets']
/* this is a very nice comment */ ['comment']
int                               ['int']
i                                 ['id']
=                                 ['assign']
0                                 ['num']
```

;	['semicolon']
int	['int']
j	['id']
=	['assign']
8	['num']
;	['semicolon']
while	['while']
(	['open_parenthesis']
i	['id']
<	['lt']
20	['num']
)	['close_parenthesis']
{	['open_brackets']
if	['if']
(	['open_parenthesis']
i	['id']
==	['eq']
j	['id']
)	['close_parenthesis']
{	['open_brackets']
print	['id']
(	['open_parenthesis']
i	['id']
)	['close_parenthesis']
;	['semicolon']
}	['close_brackets']
}	['close_brackets']
return	['return']
0	['num']
;	['semicolon']
}	['close_brackets']

4. Alfabeto

{
 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,
 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
 +, -, *, /, <, >, =, !,
 (,), [,], {, }, ;,
 ,
}

5. Expressões regulares

5.1. Keywords
else: else
if: if
int: int
return: return
void: void
while: while

Token	Expressão Regular	Valor atributo
else	else	-
if	if	-
int	int	-
return	return	-
void	void	-
while	while	-

5.2. Símbolos especiais
PLUS: +
MINUS: -
ASTERISK: *
SLASH: /
LT: <
LE: <=
GT: >
GE: >=
EQ: ==
NE: !=
ASSIGN: =
SEMICOLON: ;
COMMA: ,
OPENPARENTHESIS: (

CLOSEPARENTHESIS:)
OPENSQUAREBRACKETS: [
CLOSESQUAREBRACKETS:]
OPENBRACKETS: {
CLOSEBRACKETS: }
OPENCOMMENT: /*
CLOSECOMMENT: */

Token	Expressão Regular	Valor atributo
PLUS	+	-
MINUS	-	-
ASTERISK	*	-
SLASH	/	-
LT	<	-
LE	<=	-
GT	>	-
GE	>=	-
EQ	==	-
NE	!=	-
ASSIGN	=	-
SEMICOLON	;	-
COMMA	,	-
OPENPARENTHESIS	(	-
CLOSEPARENTHESIS	)	-
OPENSQUAREBRACKETS	[	-
CLOSESQUAREBRACKETS	]	-
OPENBRACKETS	{	-

CLOSEBRACKETS	}	-
OPENCOMMENT	/*	-
CLOSECOMMENT	*/	-

5.3. Comentários

/*(Σ)*/

5.4. Demais

LETTER: [a-zA-Z]
DIGIT: [0-9]
NUM: [0-9]+
ID: [a-zA-Z]+

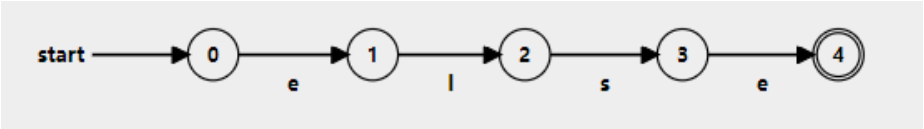
Token	Expressão Regular	Valor atributo
LETTER	[a-zA-Z]	Uma letra de a-z maiúscula ou minúscula
DIGIT	[0-9]	Um dígito numérico de 0-9
NUM	[0-9]+	Uma sequência de dígitos numéricos de 0-9
ID	[a-zA-Z]+	Uma sequência de letras de a-z maiúsculas ou minúsculas

6. Autômatos (NFA, DFA e DFA mínimo)

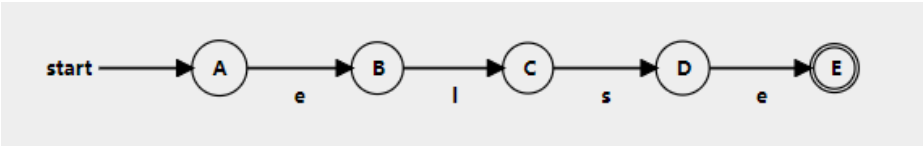
6.1. Keywords

- **else**

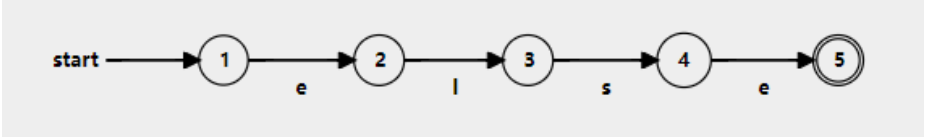
NFA:



DFA:

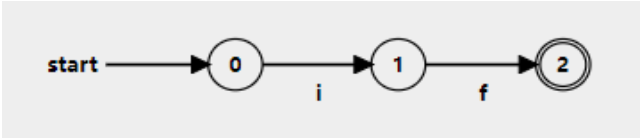


DFA minimo:

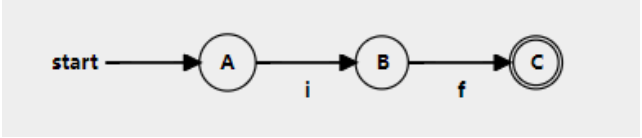


- **if**

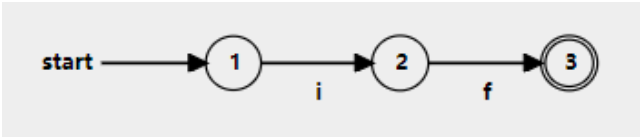
NFA:



DFA:



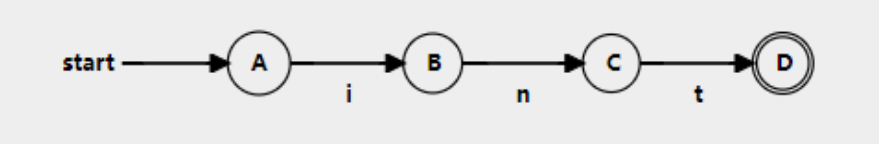
DFA minimo:



- **int**

NFA:

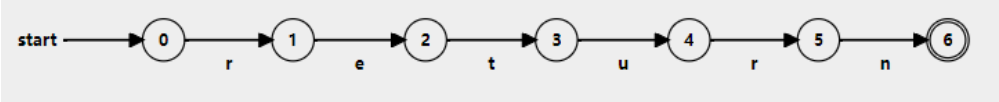
DFA:



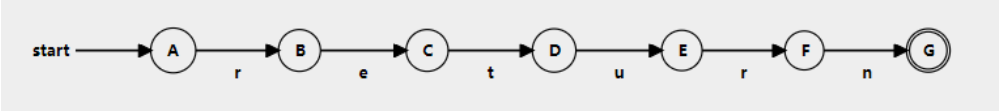
DFA minimo:

- **return**

NFA:



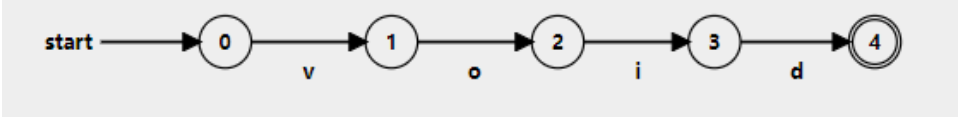
DFA:



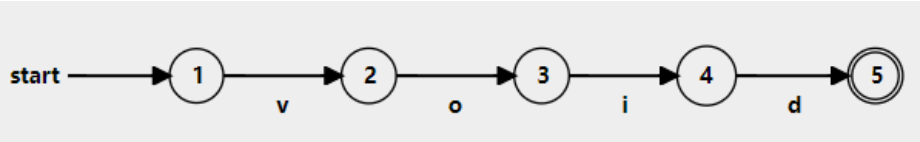
DFA minimo:

- **void**

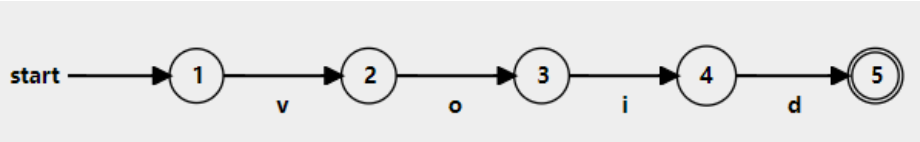
NFA:



DFA:

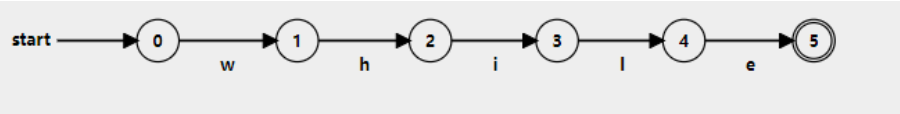


DFA minimo:

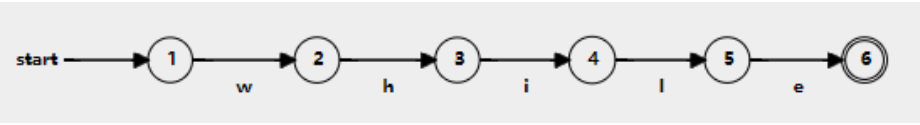


- **while**

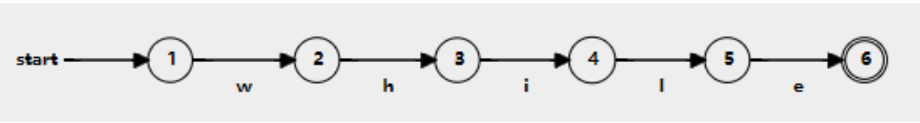
NFA:



DFA:

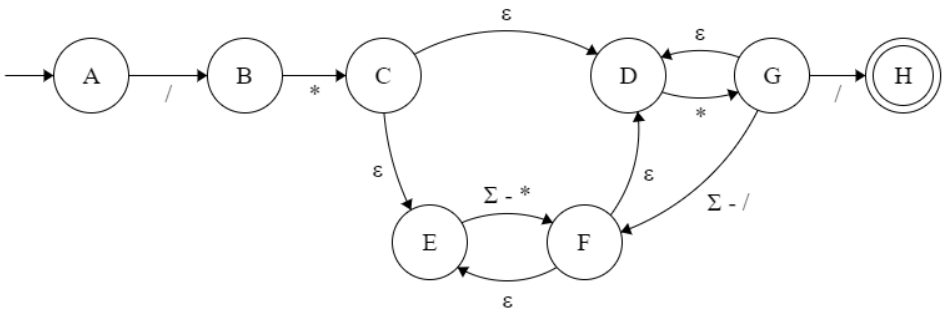


DFA mínimo:

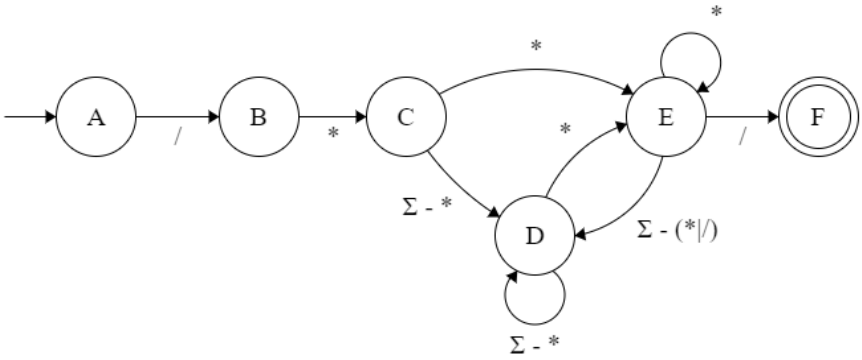


- **COMMENT**

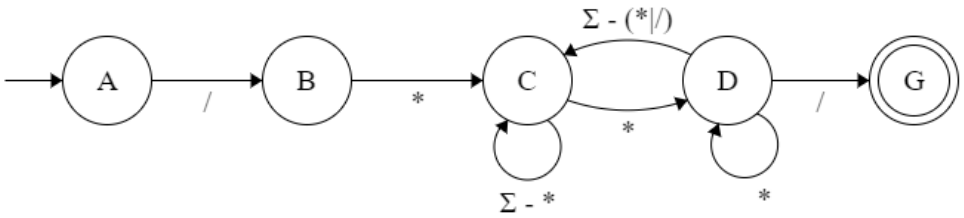
NFA:



DFA:



DFA minimo:



- **LE:**

NFA:

DFA:

DFA mínimo:

- **GE:**

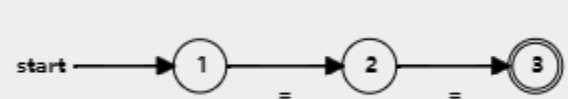
NFA:

DFA:

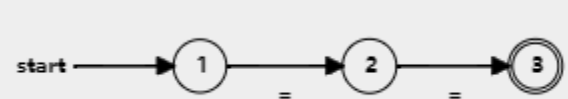
DFA mínimo:

- **EQ:**

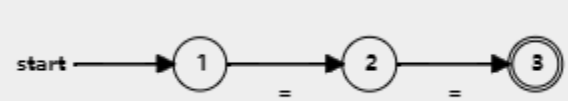
  NFA:



  DFA:

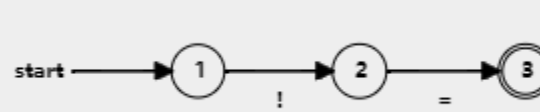


  DFA mímimo:



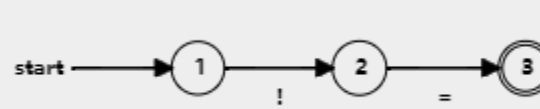
- **NE:**

  NFA:



  DFA:

  DFA mínimo:



- **LETTER**

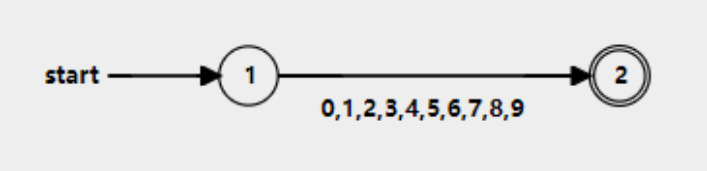
  NFA e DFA com muitos estados para demonstrar no desenho

DFA mínimo:

- **DIGIT**

NFA e DFA com muitos estados para demonstrar no desenho

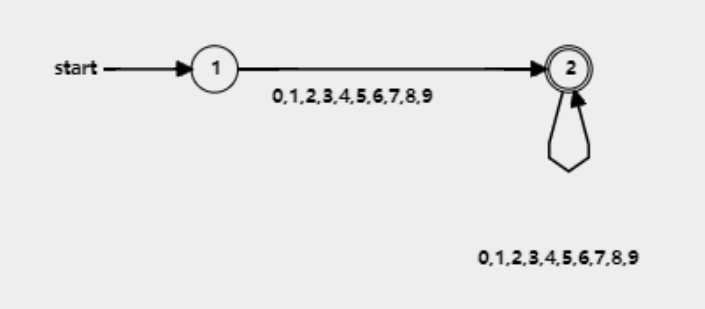
DFA mínimo:



- **NUM**

NFA e DFA com muitos estados para demonstrar no desenho

DFA mínimo:



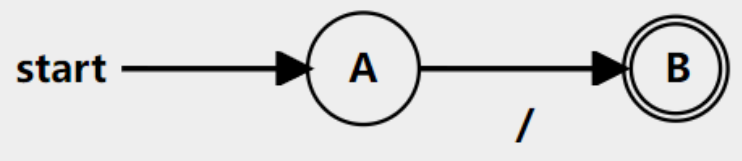
- **ID**

NFA e DFA com muitos estados para demonstrar no desenho

DFA mínimo:

- **PLUS, MINUS, ASTERISK, SLASH, LT, GT, ASSIGN, SEMICOLON, COMMA, OPENPARENTHESIS, CLOSEPARENTHESIS, OPENSQUAREBRACKETS, CLOSESQUAREBRACKETS, OPENBRACKETS, CLOSEBRACKETS**

  NFA, DFA e DFA mínimo:



Análogo aos demais tokens de apenas um símbolo...