

# Análise Léxica

Luiz Eduardo Galvão Martins

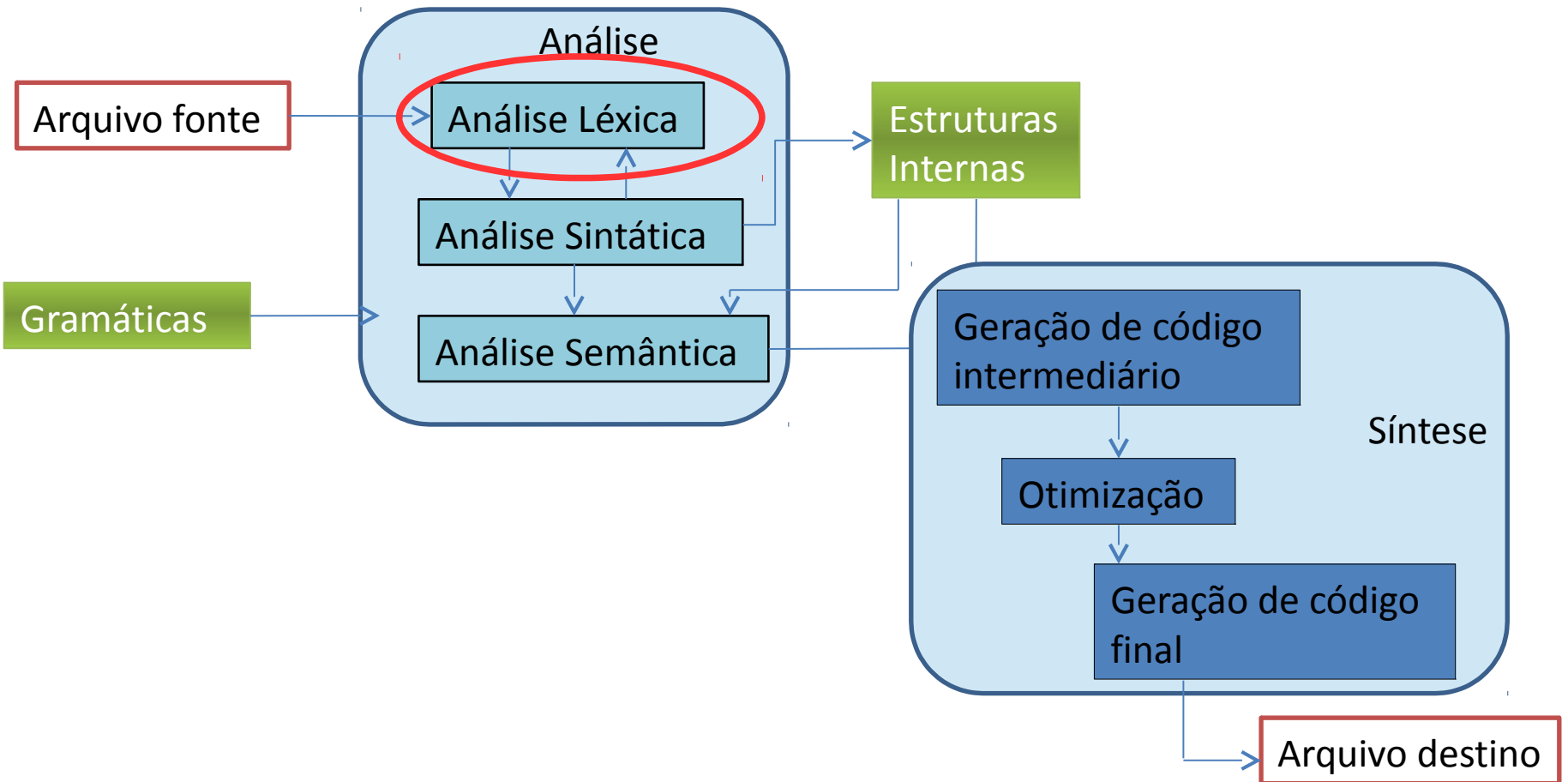
(adaptado por Ana Carolina Lorena)

UNIFESP - São José dos Campos

Agosto/2016

# Visão lógica

- Tarefas de um compilador



# Análise Léxica

- A **análise léxica** é a primeira tarefa que o compilador deve realizar sobre o arquivo fonte (entrada)
- Ela lê o programa fonte como um arquivo de caracteres e os separa em marcas ou *tokens*
  - **Marca:** sequência de caracteres que representa uma unidade lógica do programa-fonte
  - As marcas são então passadas para outras partes do compilador

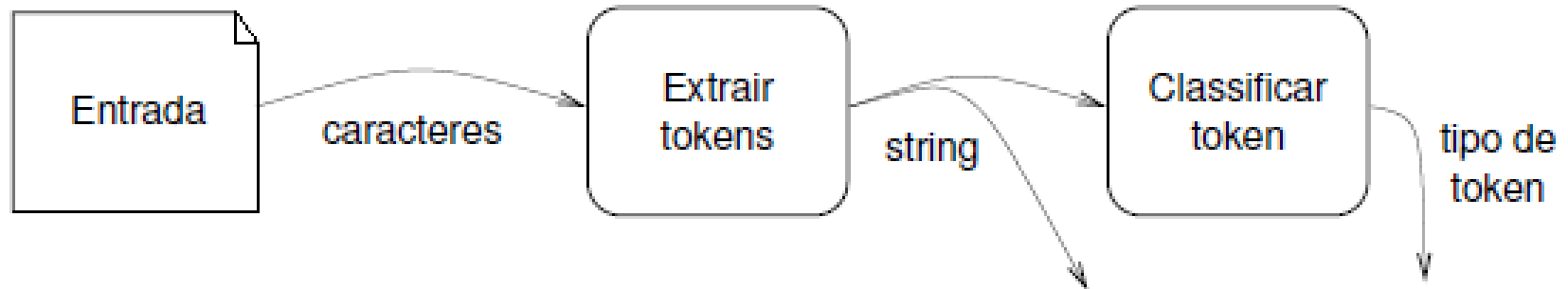
# Categorias de marcas

- Exemplos de tipos de *tokens*
  - Há categorias de marcas

Tipo	Exemplos
Identificador	contador cpf ra delta
Número	10 59 00 345
Literal	“Valor total = “
Palavra-chave (ou reservada)	If while
Símbolo especial	< <= == > >=

# Analizador Léxico

- Analisador Léxico ou Sistema de Varredura:  
programa de computador que realiza a análise léxica



# Análise Léxica

- É um caso de casamento de padrões
  - Método para especificar padrões: **expressões regulares**
    - Notação para representar padrões em cadeias de caracteres
  - Método para reconhecer padrões: **autômatos finitos**
    - Algoritmos para reconhecimento de padrões em cadeias de caracteres dadas por ERs

# Analizador Léxico

- Outras funções do Analizador Léxico:
  - Consumir comentários e outros caracteres não imprimíveis (espaço, \t, \n)
  - Manter contagem do número de linhas
    - Permite relacionar erros encontrados a sua linha
  - Diagnosticar erros léxicos
    - Ex.: identificador mal formado: j@
    - Normalmente passa para a frente o erro
    - Erros detectáveis são limitados (tem visão local, sem contexto)
      - Ex.: **fi** a>b **then** → não detectará erro e retornará fi como identificador

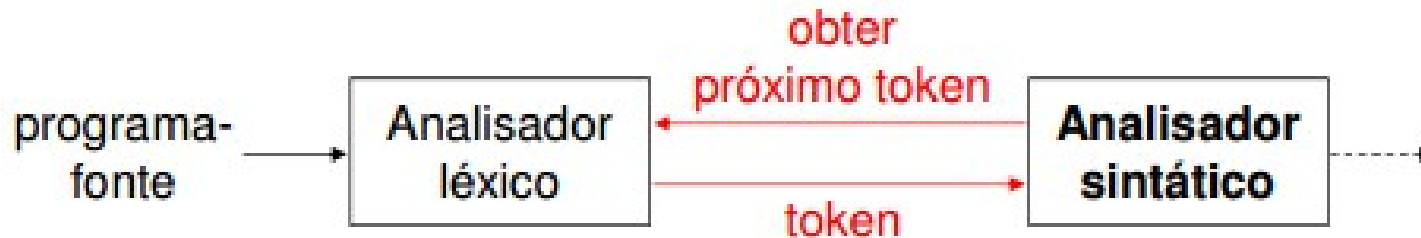
# Lexema

- A cadeia de caracteres associada a uma marca é denominada **lexema**
  - **Exemplo: marca if tem lexema “if”**
  - Palavras reservadas têm apenas um lexema cada
  - Mas uma marca pode ter potencial para representar infinitos lexemas
    - Ex.: identificador pode ter muitos lexemas
  - Mais comum: sistema de varredura retorna tipo da marca e seu lexema



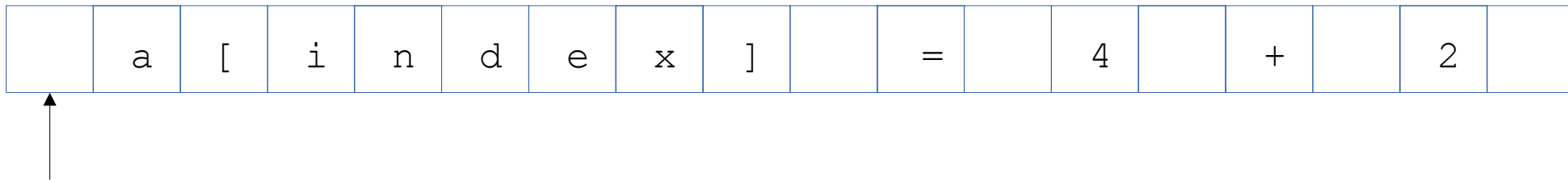
# Interação com o *Parser*

- Varredura ocorre sob controle do Analisador Sintático (*Parser*)
  - É uma sub-rotina do *Parser* que retorna a marca seguinte sempre que pedido
  - O *Parser* combina os tokens e verifica a boa formação (sintaxe) do programa-fonte



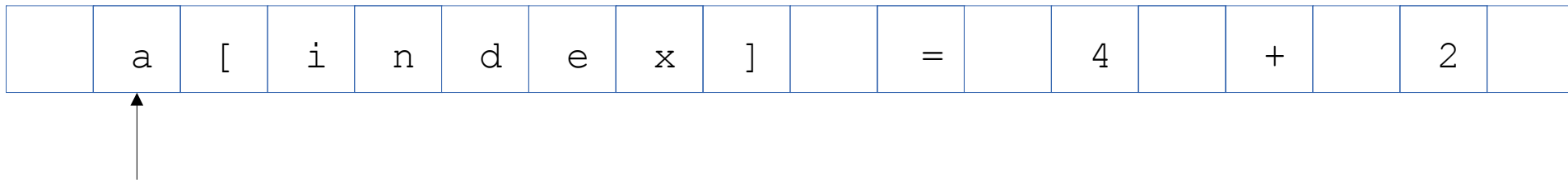
# Exemplo

`a[index] = 4 + 2`



# Exemplo

a[index] = 4 + 2

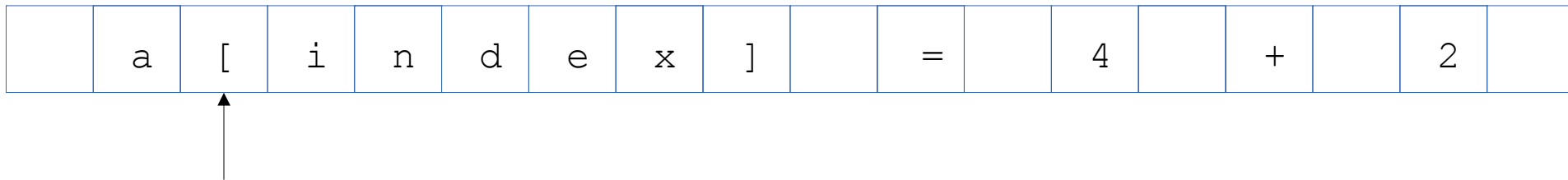


token = em verificação

lexema = a

# Exemplo

a[index] = 4 + 2



token = identificador  
lexema = a



Retorna ao Parser o par <ID,a>

# Exemplo

`a[index] = 4 + 2`

	a	[	i	n	d	e	x	]		=		4		+		2	
--	---	---	---	---	---	---	---	---	--	---	--	---	--	---	--	---	--



Se foi para próximo, tem que retroceder, pois [ é um caractere que precisa ser reconhecido

token = em verificação

lexema = [

# Exemplo

`a[index] = 4 + 2`

	a	[	i	n	d	e	x	]		=		4		+		2	
			↑														

token = símbolo especial

lexema = [




Retorna ao Parser o par <SIM,[>

# Exemplo

`a[index] = 4 + 2`

	a	[	i	n	d	e	x	]		=		4		+		2	
--	---	---	---	---	---	---	---	---	--	---	--	---	--	---	--	---	--



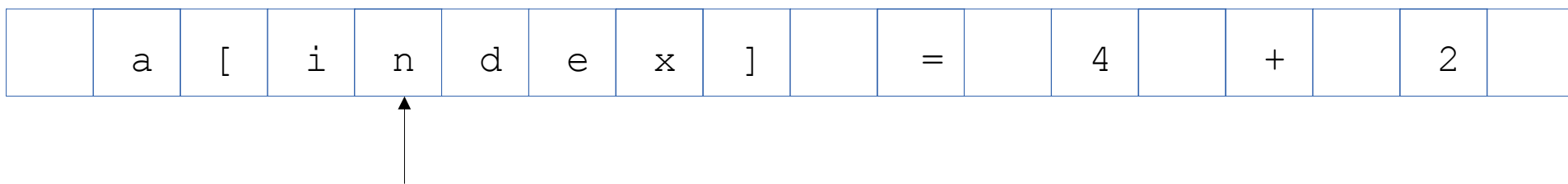
Se foi para próximo, tem que retroceder, pois i é um caractere que precisa ser reconhecido

token = em verificação

lexema = i

# Exemplo

`a[index] = 4 + 2`



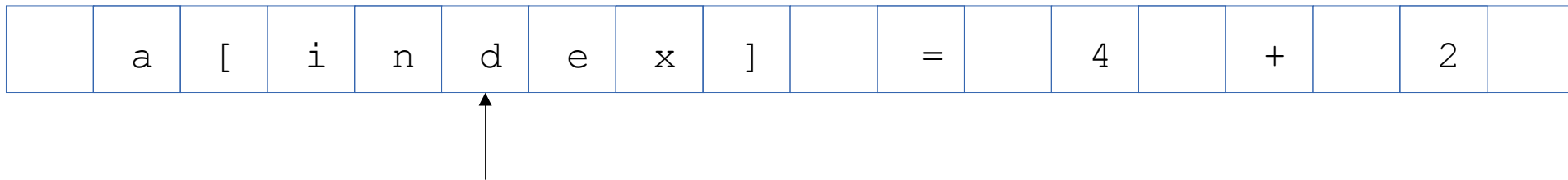
token = em verificação

lexema = in



# Exemplo

`a[index] = 4 + 2`

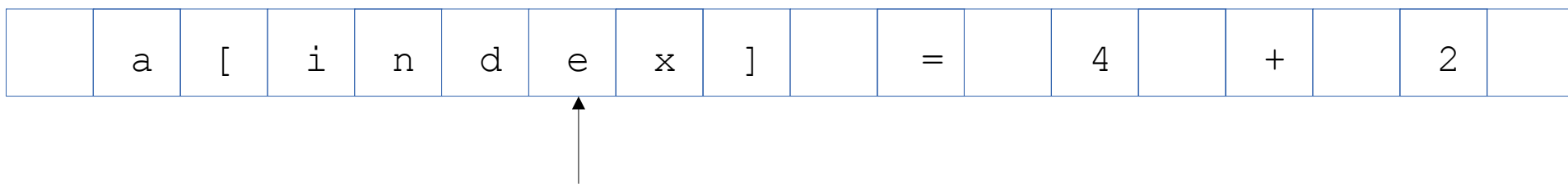


token = em verificação

lexema = ind

# Exemplo

`a[index] = 4 + 2`

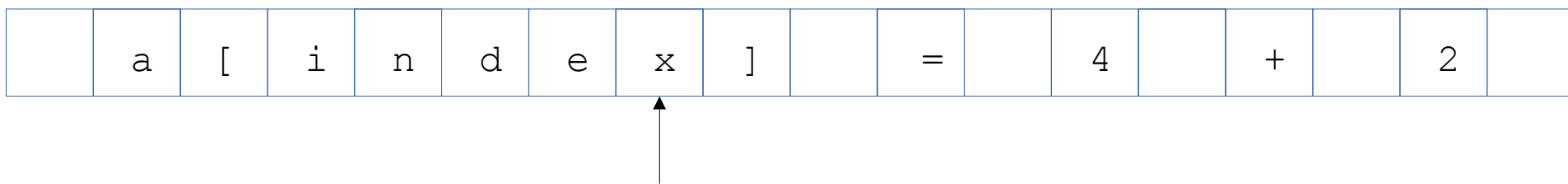


token = em verificação

lexema = inde

# Exemplo

`a[index] = 4 + 2`

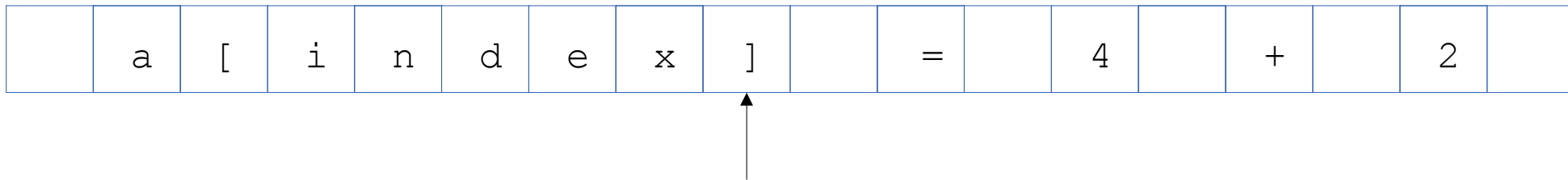


token = em verificação

lexema = index

# Exemplo

a[index] = 4 + 2



token = identificador  
lexema = index

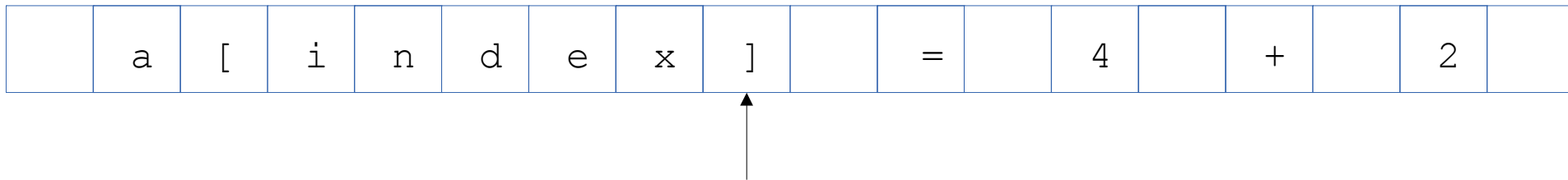


Retorna ao Parser o par <ID,index>

Note que todos os passos para identificar index corresponderam ao resultado de **uma chamada do analisador léxico pelo Parser**

# Exemplo

`a[index] = 4 + 2`



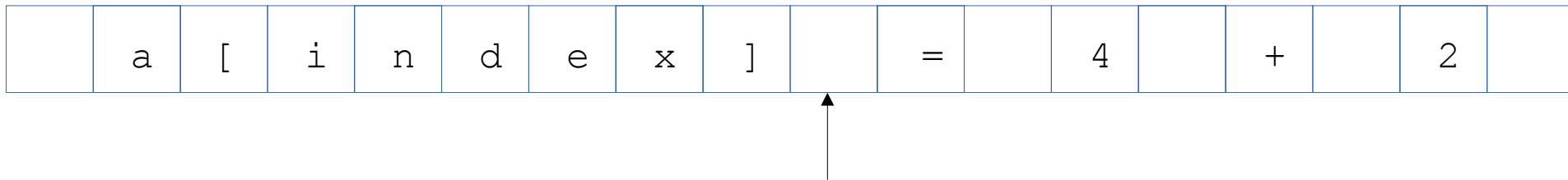
Se foi para próximo, tem que retroceder, pois `]` é um caractere que precisa ser reconhecido

token = em verificação

lexema = `]`

# Exemplo

`a[index] = 4 + 2`



Neste caso não precisa de retrocesso, pois espaço em branco é normalmente delimitador de marcas

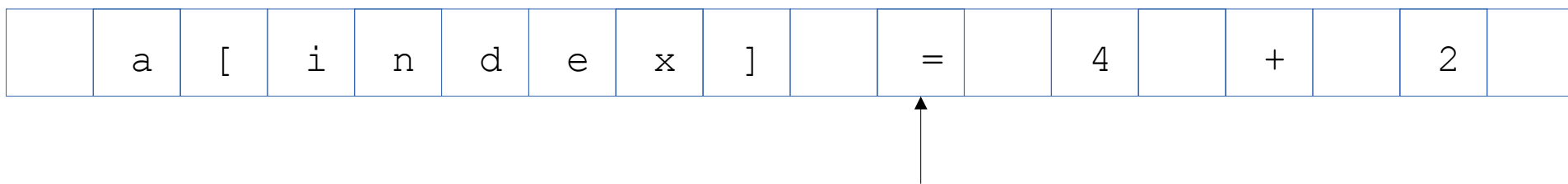
token = símbolo especial  
lexema = ]



Retorna ao Parser o par <SIM,]>

# Exemplo

`a[index] = 4 + 2`

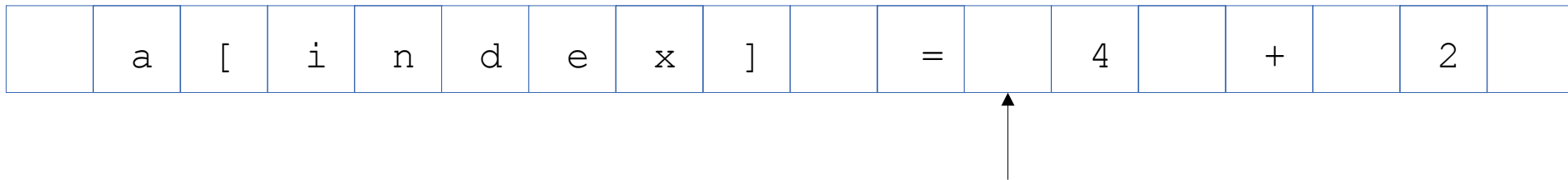


token = em verificação

lexema = =

# Exemplo

`a[index] = 4 + 2`



token = símbolo especial  
lexema = =

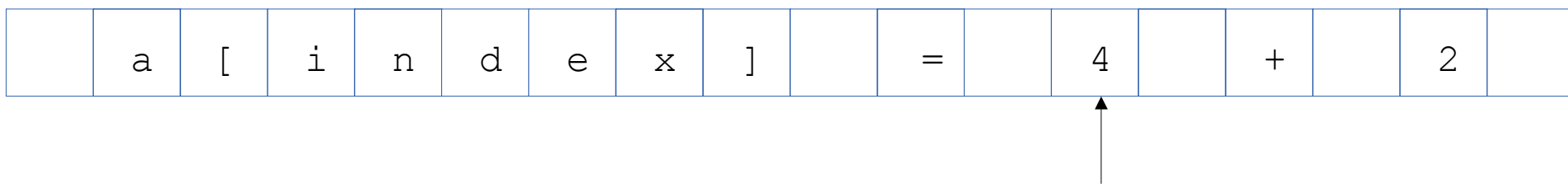


Retorna ao Parser o par <SIM,=>



# Exemplo

`a[index] = 4 + 2`

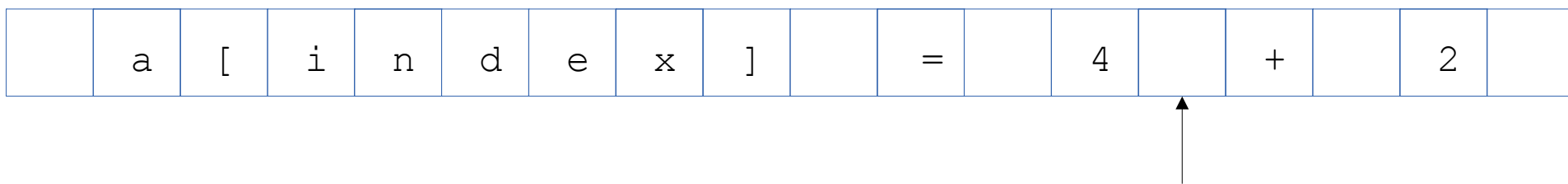


token = em verificação

lexema = 4

# Exemplo

`a[index] = 4 + 2`



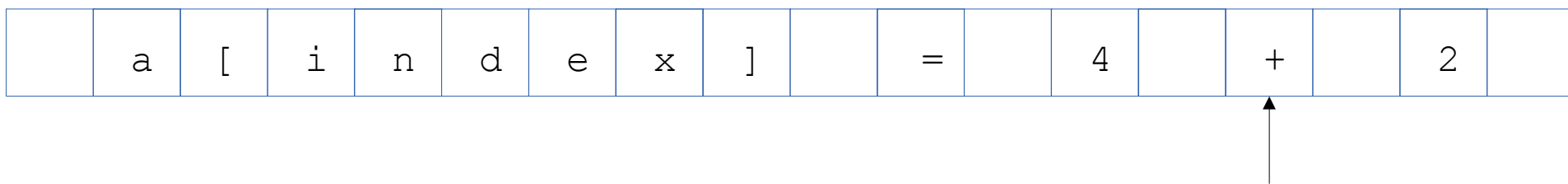
token = número  
lexema = 4



Retorna ao Parser o par <NUM,4>

# Exemplo

`a[index] = 4 + 2`

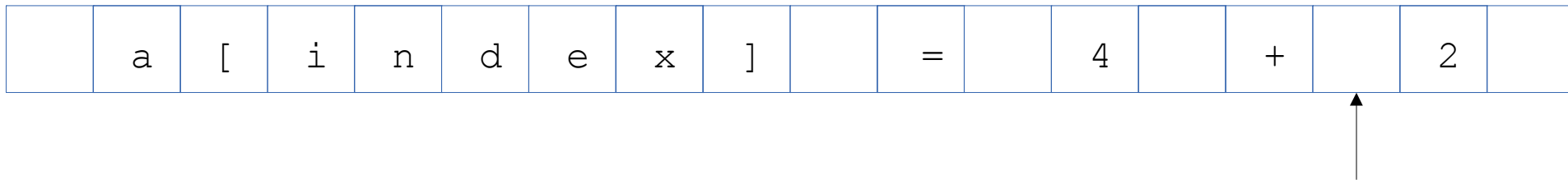


token = em verificação

lexema = +

# Exemplo

`a[index] = 4 + 2`



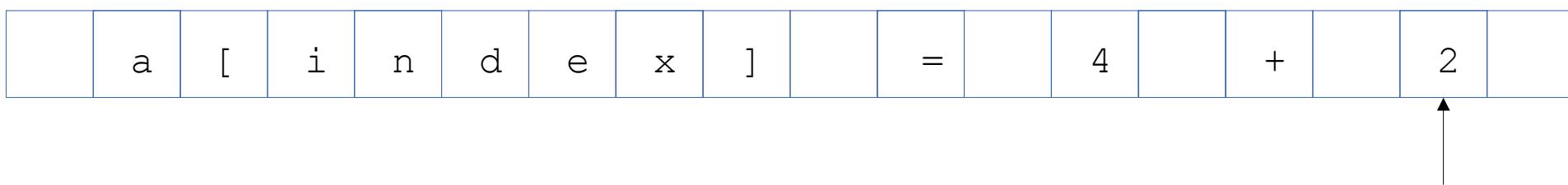
token = símbolo especial  
lexema = +



Retorna ao Parser o par <SIM,+>

# Exemplo

`a[index] = 4 + 2`

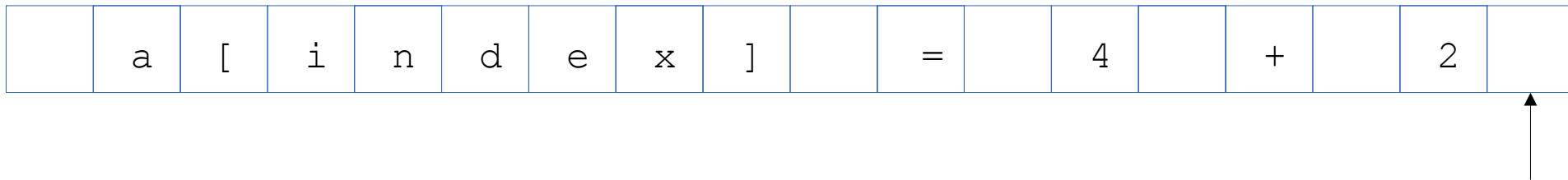


token = em verificação

lexema = 2

# Exemplo

`a[index] = 4 + 2`



token = número  
lexema = 2



Retorna ao Parser o par <NUM,2>

# Bibliografia Sugerida

Capítulo 2 Compiladores Kenneth Louden

Alguns slides do Prof Dr Thiago Pardo, ICMC-USP