

Compiladores

Análise Léxica - Cap. 3 - Parte II

Expressões e Definições Regulares - Revisão

- Expressões regulares descrevem todas as linguagens que podem ser construídas daqueles operadores aplicados aos símbolos de algum alfabeto
- Definições regulares constituem-se numa seqüência de definições para as expressões regulares de forma que podemos usar essas definições em expressões subseqüentes de maneira recursiva.

Associatividade de operadores e precedência

- Diz-se que um operador + associa à esquerda porque um operando que está entre dois sinais +, pertencerá ao operador de sua esquerda.
 - Ex: $10+5+6 \rightarrow (10+5)+6$
 - Obs: os operadores +, -, / e * (multiplicação) são associativos à esquerda
- Alguns operadores como exponenciação e atribuição (=) são associativos à direita
 - Ex: $a=b=c \rightarrow a=(b=c)$

Associatividade de operadores e precedência

- Observe a construção de expressões regulares com operadores associativos à direita
 - Ex: $\text{direita} \rightarrow \text{id} = \text{direita} \mid \text{id}$
 $\text{id} \rightarrow a \mid b \mid c \mid \dots \mid z$
- O mesmo “raciocínio” vale para os operadores associativos à esquerda
 - Ex: $\text{soma} \rightarrow \text{soma} + \text{fator} \mid \text{fator}$
 $\text{fator} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid \dots \mid 9$

Associatividade de operadores e precedência

- Sabemos que o operador $*$ (multiplicação) tem precedência sobre o operador $+$. Ou seja, $*$ utiliza os operandos antes de $+$.
- Além disso, sabemos que $*$ (multiplicação) e $/$ (divisão) têm precedência sobre os operadores $+$ e $-$. Assim, nos exemplos a seguir, o operando 5 é usado pelo operador $*$ em ambos os casos.
 - Ex1: $9+5*2$
 - Ex2: $9*5+2$
- Separando os operadores pelo nível de precedência temos que:
 - Recursão à esquerda: $+$ $-$
 - Recursão à esquerda: $*$ $/$

Associatividade de operadores e precedência

- Uma gramática para expressões aritméticas com operadores +, -, * e / e que considere a precedência de operadores é:

expr \rightarrow expr + term | expr – term | term

term \rightarrow term * factor | term / factor | factor

factor \rightarrow 0|1|2|3|...|9

- Observe que as produções **expr** e **term** são recursivas e suas recursões acontecem sempre pelo operando que está à esquerda do sinal.

Exercícios - Associatividade de operadores e precedência

- Construa uma GLC para gerar todas as expressões contendo os átomos representados pelos símbolos terminais citados na tabela abaixo. A gramática deverá seguir as seguintes regras de precedência e associatividade

Associatividade

Esquerda

Esquerda

Esquerda

-

Precedência

<< , >>

&

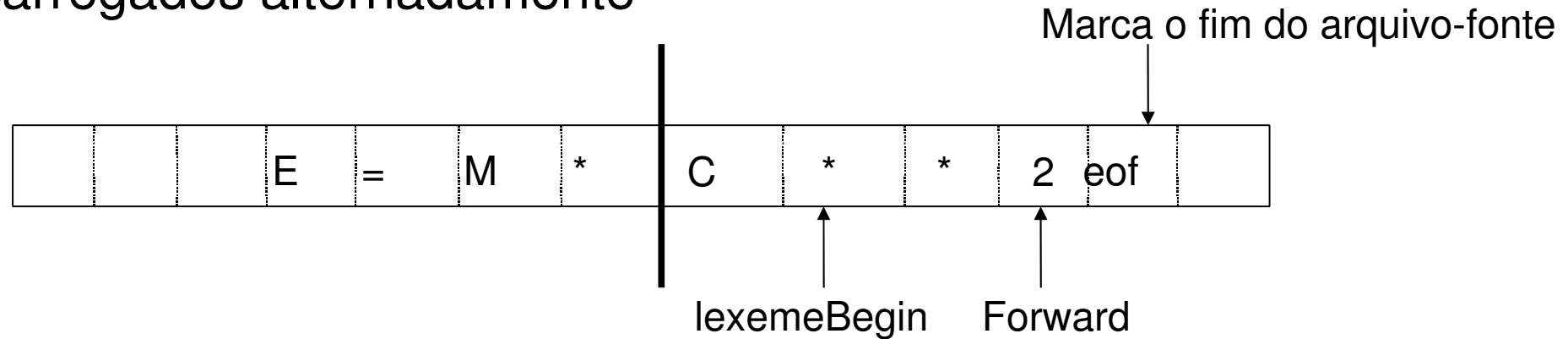
|

!

- Dica: Crie um não terminal *fator* para as unidades básicas das expressões. O *fator* irá produzir a negação (**!**) de uma expressão (*expr*), bem como as unidades básicas **id** , **num** e uma expressão parentizada (*expr*). O não terminal *expr* é o símbolo inicial da gramática, e representa uma seqüência de átomos que formam uma expressão válida.

Buffers de Entrada

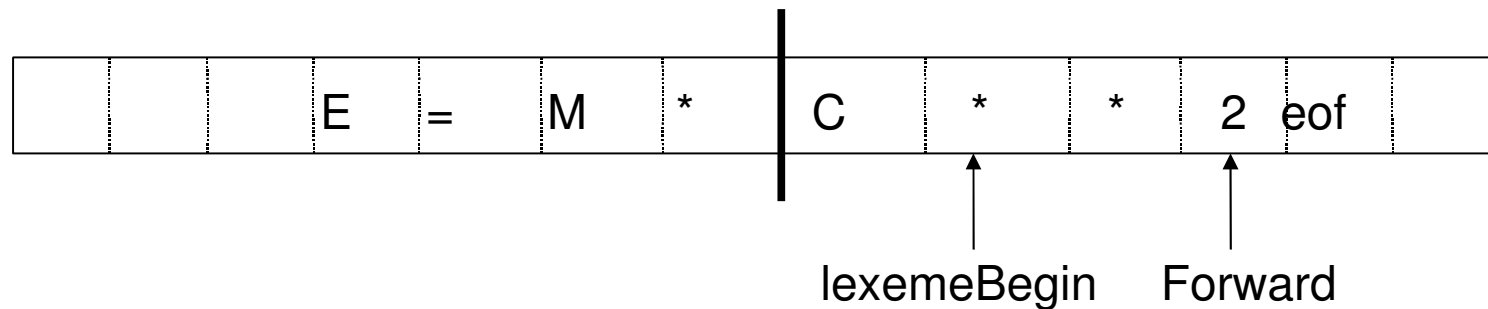
- Técnicas de buffering são usadas para reduzir o custo no processamento de um único caractere de entrada
- Um esquema utilizada é a adoção de dois buffers que são recarregados alternadamente



- O apontador lexemeBegin marca o início do lexema corrente, cuja extensão estamos tentando determinar
- O apontador forward lê adiante, até que ocorra um casamento de padrão

Buffers de Entrada

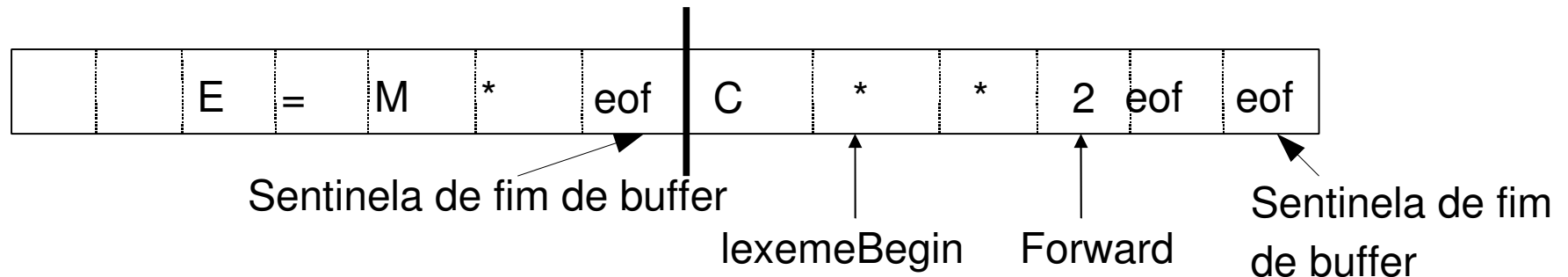
- Cada buffer possui o mesmo tamanho N
- Uma vez que o lexema é encontrado, forward é configurado para apontar para o caractere mais à direita (último) desse lexema



- Depois disso, lexemeBegin deve apontar para o caractere após o lexema recém-encontrado
- Na Figura acima, vemos que Forward passou do fim do próximo lexema e precisa ser recuado uma posição
- Avançar o apontador Forward exige duas comparações:
 - testar se chegamos ao fim de um dos buffers
 - testar se o caractere apontado por Forward faz parte do lexema

Buffers de Entrada

- A fim de evitar uma comparação se usarmos caracteres sentinelas ao final de cada buffer



```
Switch (*forward++){
case eof:
    if (forward == fim primeiro buffer){
        recarrega segundo buffer;
        forward=inicio segundo buffer
    }else if (forward == fim segundo buffer){
        recarrega primeiro buffer;
        forward=inicio primeiro buffer
    }else /* eof dentro de um buffer significa que é fim de arquivo*/
        termina lexico
    break;

...
}
```