Linguagem de Programação N.O.M.E - Nomeclatura Orientada a Macumba Exagerada

Felipe Pereira Lyssa Scherer Murilo Schaefer

2017

Introdução

A linguagem de programação NOME visa o estudo sobre o desenvolvimento de uma linguagem de programação e seu compilador.

Este projeto tem como finalidade apresentar especificação da linguagem de programação de alto nível que será utilizada como base para o projeto de compilador/interpretador para a disciplina de compiladores. O desenvolvimento do mesmo abrange melhor entendimento das dificuldades em atender os critérios da LP e desenvolvimento necessário de um projeto de compilador/interpretador.

1 Descrição da Linguagem

1.1 Operadores Lógicos

Está linguagem dá suporte a 4 operadores lógicos: OR, AND, XOR e NOT. Estes só podem ser usados para operações com variáveis booleanas e são operadores binários, com exceção ao NOT, que é unário. A representação simbólica de cada operação é equivalente ao seu nome e pode ser vista na Tabela 1.

Operação Lógica	**Símbolo**	**Exemplo**
OR	*or*	1 *or* 0
AND	*and*	1 *and* 1
XOR	*xor*	1 *xor* 0
NOT	*not*	*not* 1

Tabela 1 – Representação simbólica dos Operadores Lógicos

1.2 Operadores Aritméticos

Está linguagem dá suporte a 4 operadores aritméticos: adição, subtração, divisão e multiplicação. Estes operadores só podem operar variáveis numéricas e todas as operações são binárias. A representação simbólica de cada operação pode ser vista na Tabela 2.

Operadores Aritmética	Símbolo	Exemplo
Adição	$+$	$20 + 5$
Subtração	$-$	$10 - 3$
Divisão	$/$	$1/2$
Multiplicação	$*$	$3 * 6$

Tabela 2 – Representação simbólica dos Operadores Aritméticos

1.3 Operadores Relacionais

Há o suporte a 6 operadores relacionais: menor, maior, menor ou igual, maior ou igual, igual e diferente. Estes operadores são para operações com variáveis numéricas e são operadores binários. Os operadores igual e diferente são exceções e além de suportar operações com variáveis numéricas, também suportam variáveis booleanas mas, o operador não pode operar tipos incompatíveis de dados, como comparar uma variável inteira com uma booleana. A representação simbólica de cada operação pode ser vista na Tabela 3.

Operadores Relacionais	Símbolo	Exemplo
Maior	$>$	$3 > 2$
Menor	$<$	$1 < 5$
Maior ou igual	$>=$	$2 >= 2$
Menor ou igual	$<=$	$1 <= 2$
Igual	$==$	$4 == 4$
Diferente	$!=$	$3! = 4$

Tabela 3 – Representação simbólica dos Operadores Relacionais

1.4 Tipos de dados suportados

Está linguagem dá suporte a 3 tipos de dados: inteiro, caracter (char) e boolean. Algumas informações sobre cada tipo de dado pode ser visto na Tabela 4.

Tipo	Tamanho	Palavra reservada	Exemplo
Inteiro	4 bytes	*int*	4
Caracter	1 byte	*char*	'c'
Boolean	1 byte	*bool*	*true*

Tabela 4 – Representação dos Tipos de Dados

1.5 Identificadores

Existem algumas restrições para o nome das variáveis: o número máximo de caracteres que pode ser utilizado é 50. Toda variável precisa começar com o caracter '#'.Pode conter letras do alfabeto (a - Z), tanto maiúsculas quanto minúsculas e números.Após o segundo caracter é possivel inserir também o '_'.

1.6 Entrada e saída

A saída se dará através do printf. Pra escrever strings é necessário inserir o texto dentro de aspas duplas, para escrever o valor de uma variável basta escrever seu nome.

Exemplo Saída: printf("oi");

A entrada se dá através do comando read() onde o retorno é o valor lido de uma varável, por isso será necessária a sua atribuição a uma variável.

Exemplo Entrada: var = read();

1.7 Atribuição

Está linguagem dá suporte a atribuição, seu símbolo é representado pelo símbolo igual (=). Com ele é possível atribuir valores às variáveis, desde que respeitando a sua tipagem. Por exemplo, a atribuição (1) não é válida, já que inteiros não aceitam caracteres. Já a atribuição (2) é válida.

$$int\ a = `c' \ (1)$$

$$int\ a = 5 \ (2)$$

Também é possível realizar a atribuição passando uma expressão aritmética, desde que o retorno da expressão seja do mesmo tipo de dados que a variável, como pode ser visto no exemplo (3).

$$int\ a = 5 + 3 \ (3)$$

1.8 Saltos condicionais

Está linguagem suporta apenas dois saltos condicionais: *if* e *switch case*. O *if* representa um bloco que só será acessado caso uma expressão condicional seja atendida. Já o *switch case* representa que o conteúdo de uma variável será comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado. A sintaxe destes dois saltos condicionais pode ser visto na Tabela 5

Função	**Sintaxe**	**Exemplo**
if	$if(< expr\_cond >)\{$ $\quad < inst >$ $\}else\{$ $\quad < inst >$ $\}$	$if(true)\{$ $\quad a = 1 + 2;$ $\}\ else\{$ $\quad a = 2 + 3;$ $\}$
switch case	$switch(< id >)\{$ $\quad case(< expr >) :< inst >$ $\}$	$switch(a)\{$ $\quad case(a ==' b') : a = `c';$ $\}$

Tabela 5 – Representação dos Saltos Condicionais

1.9 Estruturas de repetição

Está linguagem suporta as estruturas de repetição for e $while$. O for repete o bloco de comandos ($< inst >$) até que condição ($< expr\_cond >$) seja concluída. O for é

composto de três expressões separadas por ponto e vírgula(;). A primeira equivale a uma atribuição(<attr>), a segunda equivale a uma expressão condicional ($< expr\_cond >$) e a terceira equivale a uma expressão aritmética ($< expr\_arit >$). A estrutura *while* faz a validação de uma expressão lógica e enquanto esta for verdadeira, ele repete um bloco que contem um conjunto de instruções ($< inst >$). Os exemplos e descrições podem ser vistos na Tabela 6.

Função	**Sintaxe**	**Exemplo**
for	$for(< attr >; < expr\_cond >; < expr\_arit >)\{$ $\quad < inst >$ $\}$	$for(i = 0; i < 10; i = i + 1)\{$ $\quad int\ a = 4 + 1;$ $\}$
while	$while(< expr >)\{$ $\quad < inst >$ $\}$	$while(i < 10)\{$ $\quad i = i + 1;$ $\}$

Tabela 6 – Representação dos Laços de Repetição

1.10 Palavras Reservadas

As palavras reservadas são comandos de uso que não poderão ser utilizadas/alteradas pelo desenvolvedor pois possuem significado exclusivo. A seguir algumas palavras e seus respectivos significados:

- int, bool, char : tipos de variáveis
- and, xor, not, or: operadores lógicos
- for, while: utilizado em laços de repetições
- if, else, switch, case: usado para saltos condicionais.
- printf, read : entrada e saída
- true, false

2 EBNF

2.1 Sintaxe Geral:

O λ é utilizado quando não será mais inserido outras instruções.

$$< program > \rightarrow [< inst >] \quad (1)$$

$$< inst > \rightarrow (< if > | < while > | < for > | < switch >)\{< inst >\} \quad (2)$$

$$| \quad (\lambda \mid < attr >) \quad (3)$$

$$< attr > \rightarrow < tipo >< id >';'\{< inst >\} \quad (4)$$

$$| \quad < tipo >< id >'='(< expr\_arit > \mid < expr\_log >)';'\{< inst >\} \quad (5)$$

$$< tipo > \rightarrow (int \mid char \mid bool) \quad (6)$$

2.2 Operadores

(i) Operadores Lógicos:
Sintaxe léxica:

$$< expr\_log > \rightarrow < termo\_log > \{(and \mid or \mid xor) < termo\_log >\} \quad (7)$$
$$\mid not \ < termo\_log > \quad (8)$$
$$< termo\_log > \rightarrow '(' < expr\_log > ')' \quad (9)$$
$$\mid < id > \quad (10)$$

(ii) Operadores Aritméticos:
Sintaxe léxica:

$$< expr\_arit > \rightarrow < termo\_arit > \ \{(+ \mid -) \ < termo\_arit >\} \quad (11)$$
$$< termo\_arit > \rightarrow < fator\_arit > \ \{(* \mid /) \ < fator\_arit >\} \quad (12)$$
$$< fator\_arit > \rightarrow '(' < exp\_arit > ')' \quad (13)$$
$$\mid < id > \quad (14)$$

(iii) Operadores Relacionais:
Sintaxe léxica:

$$< expr\_cond > \rightarrow < termo\_cond > \ \{(< \ \mid \ > \ \mid \ <= \ \mid \ >= \ \mid \ == \ \mid ! =) \quad (15)$$
$$< termo\_cond >\} \quad (16)$$
$$< termo\_cond > \rightarrow '(' < expr\_cond > ')' \quad (17)$$
$$\mid '(' < expr\_arit > ')' \quad (18)$$
$$\mid '(' < expr\_log > ')' \quad (19)$$

2.3 Saltos

(i) Salto Condicional If:
Sintaxe léxica:

$$< if > \rightarrow if'('< expr\_cond >')'\{' < inst >'\}'[else'\{' < inst >'\}'] \quad (20)$$

(ii) Salto Condicional Switch: Sintaxe léxica:

$$< switch > \rightarrow switch'('< id >')'\{'< case >'; break;' \{< case >'; break;' \}'\}' \quad (21)$$
$$< case > \rightarrow case'('(< id > \mid < num > \mid < char >') :'< inst >' \quad (22)$$

2.4 Laços de repetição

(i) for:
Sintaxe léxica:

$$< for > \rightarrow for'('< attr\_for >';'< expr\_cond >';'< expr\_arit >')\{' \quad (23)$$
$$< inst > \{< inst >\}'\}' \quad (24)$$
$$< attr\_for > \rightarrow < id >'='< expr\_arit > \quad (25)$$

(ii) while: Sintaxe léxica:

$$< while > \quad \rightarrow \quad 'while('< expr\_cond >')\{' \tag{26}$$
$$< inst > \{< inst >\}'\}' \tag{27}$$

2.5 Exemplo

$int\ a = 0$
$int\ b = a$
$for(a = 0; a < 100; a = a + 1) :$
    $b = b * a$
$while(a >= 0) :$
    $a = a - b$

3 Autômatos e Expressões Regulares (ER)

Nesta seção iremos apresentar os autômatos e expressões regulares da linguagem. Para maior compreensão e organização deste texto dividimos os autômatos em funções. O automato da Figura 1 ilustra o automato que conecta todas as funções.

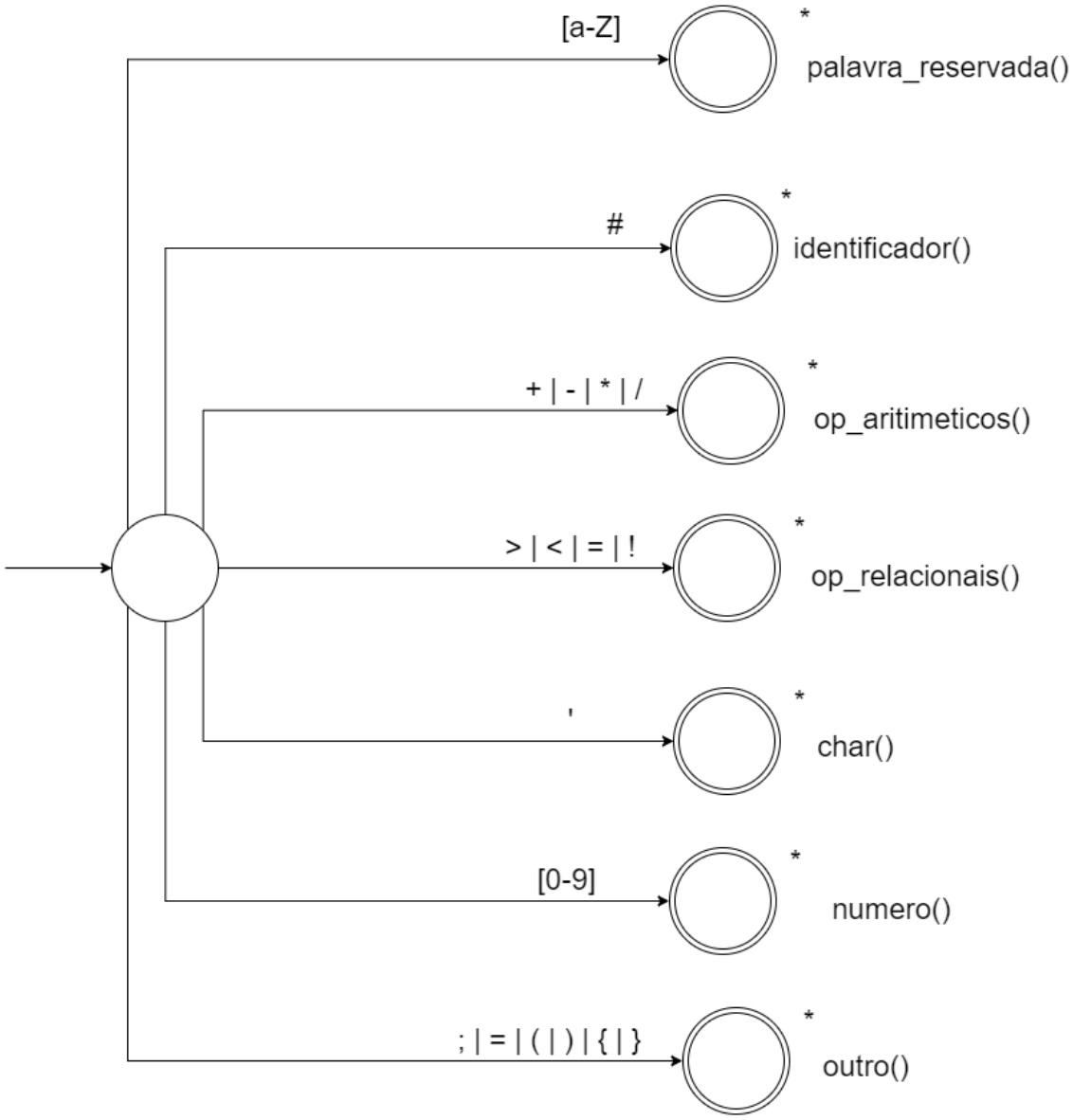

Figura 1 – Automato Geral

3.1 Palavras Reservadas

ER das Palavras Reservadas: $[a - Z]^+$

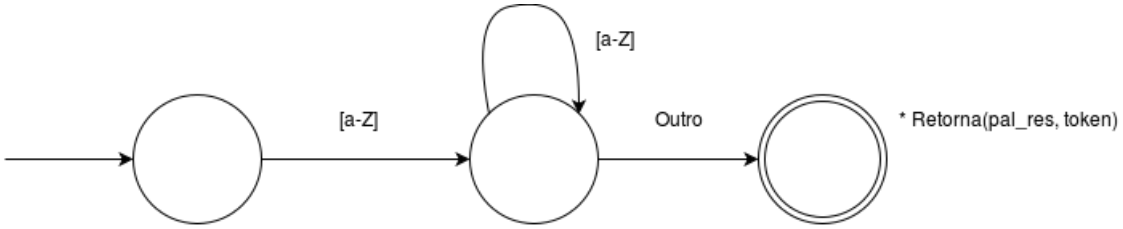

Figura 2 – Automato para palavras reservadas

3.2 Identificadores

ER dos Identificadores: $(\#)\ ([a-Z]|[0-9])\ ([a-Z]|[0-9]|'\_')^*$

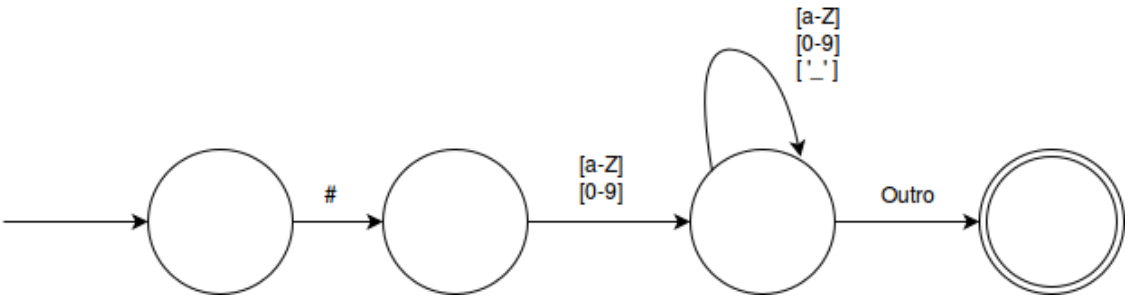

Figura 3 – Automato para identificadores

3.3 Operadores Aritméticos

ER dos Operadores Aritméticos: $('+'|'-'|'*'|'/')$

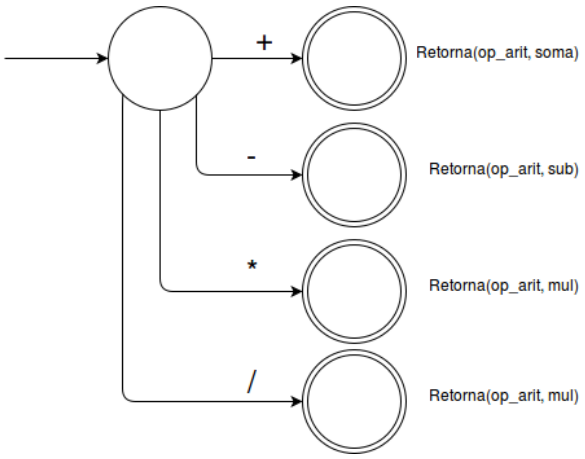

Figura 4 – Automato para Operadores Aritméticos

3.4 Operadores Relacionais

ER dos Operadores Relacionais: $('<'|'>'|'!'|'=')('=')?$

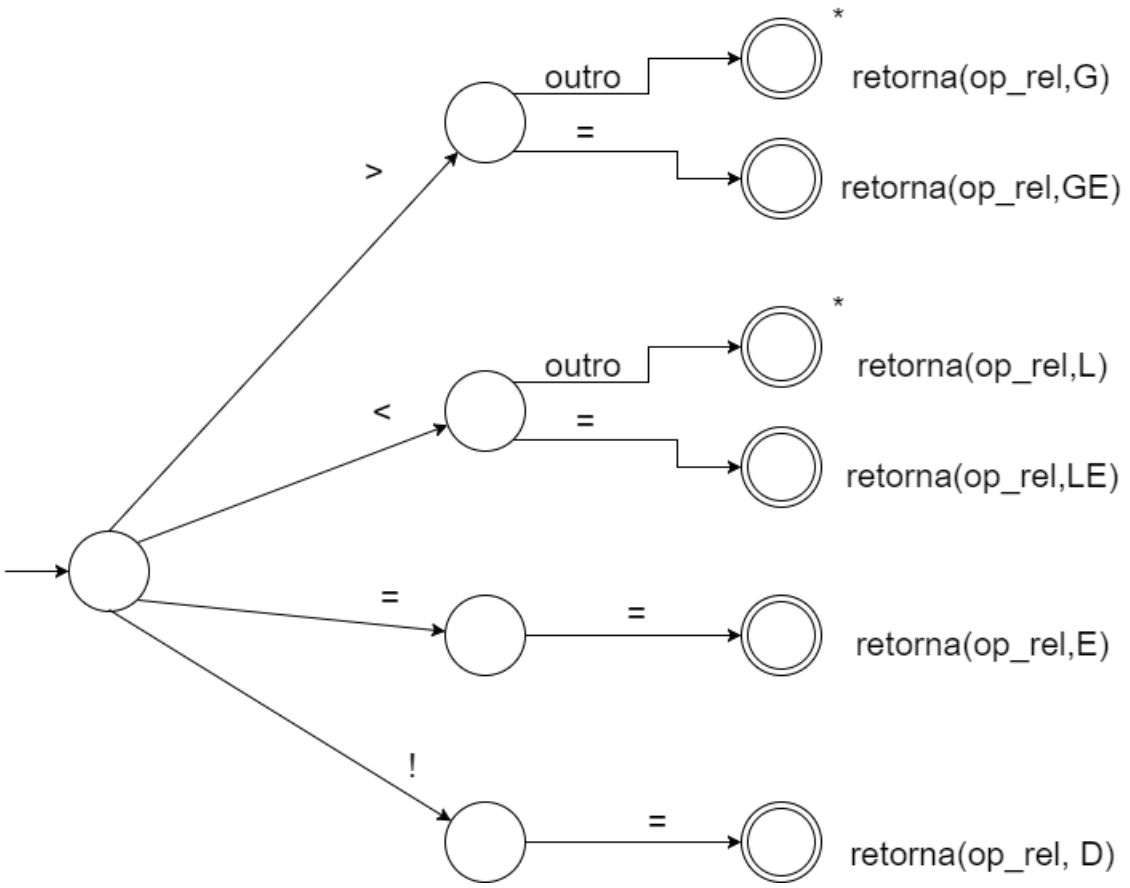

Figura 5 – Automato para Operadores Relacionais

3.5 Números

ER dos Números: $([0-9])^{+}$

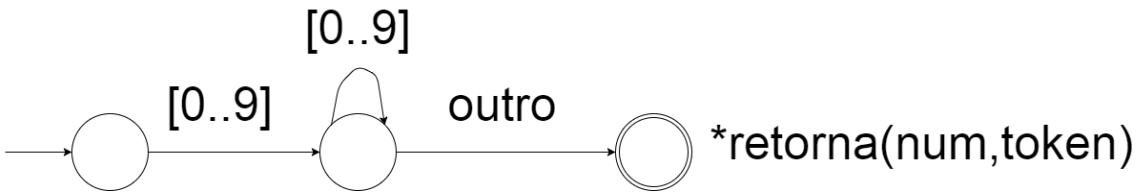

Figura 6 – Automato para Números

3.6 Caracteres Soltos

ER dos Caracteres soltos: ($';' \mid '=' \mid '(' \mid ')' \mid '\{' \mid '\}' \mid ' : '$)

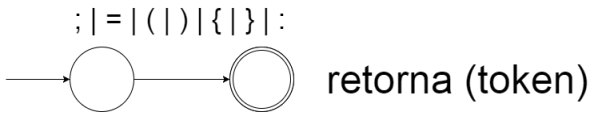

Figura 7 – Automato para Caracteres Soltos

3.7 Caracteres para a variável do tipo Char

ER dos Caracteres para a variável do tipo Char: $(')(\ .\ |\ \backslash n\ )(')$

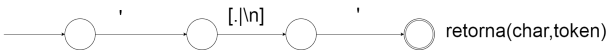

Figura 8 – Automato para Caracteres para a variável do tipo Char