



UNIVERSIDADE FEDERAL DE ITAJUBÁ

ECOM05 - LINGUAGENS FORMAIS

PROF. THATYANA DE FARIA PIOLA SERAPHIM

---

## Linguagem VAL

---

**Grupo:**

André Prado Arantes - arantesandre97@gmail.com - 2016011733

Lívia Cunha Granato - liviagranato@hotmail.com - 24601

Victor Pereira Moreira - tanabebr@gmail.com - 2016012632

4 de Junho, 2019

# Sumário

<b>1</b>	<b>Sobre a linguagem VAL</b>	<b>3</b>
<b>2</b>	<b>Expressões Regulares</b>	<b>3</b>
2.1	Operações Aritméticas . . . . .	4
2.2	Operações Relacionais . . . . .	4
2.3	Blocos de Comando . . . . .	4
2.4	Símbolos Especiais . . . . .	5
<b>3</b>	<b>Tokens</b>	<b>5</b>
<b>4</b>	<b>Regras de Utilização</b>	<b>6</b>
4.1	Programa . . . . .	6
4.1.1	Autômato Função Principal . . . . .	6
4.2	Identificadores . . . . .	6
4.2.1	Autômato Identificador . . . . .	6
4.3	Funções . . . . .	7
4.3.1	Autômato Função . . . . .	7
4.3.2	Autômato Parâmetro . . . . .	7
4.4	Strings . . . . .	8
4.4.1	Autômato String . . . . .	8
4.5	Reais . . . . .	8
4.5.1	Autômato Real . . . . .	8
4.6	Definição e Atribuição de Valores . . . . .	9
4.6.1	Autômato Atribuição . . . . .	9
4.7	Condição . . . . .	9
4.7.1	Autômato Condição . . . . .	9
4.8	Entrada e Saída de Informação . . . . .	10
4.8.1	Autômato Entrada . . . . .	10
4.8.2	Autômato Saída . . . . .	10
4.9	Operações Aritméticas . . . . .	11
4.9.1	Autômato Aritmético . . . . .	11
4.10	Operações Relacionais . . . . .	12
4.10.1	Autômato Relacional . . . . .	12
4.11	Operações lógicas . . . . .	13
4.11.1	Autômato Lógico . . . . .	13
4.12	Condicionais . . . . .	14
4.12.1	Autômato Condicional . . . . .	14
4.13	Repetição . . . . .	15
4.13.1	Autômato Repetição . . . . .	15
4.13.2	Autômato Iterador . . . . .	15
4.14	Enquanto . . . . .	16
4.14.1	Autômato Enquanto . . . . .	16
4.15	Quebra de linha . . . . .	16
4.15.1	Autômato Quebra de linha . . . . .	16

<b>5</b>	<b>Exemplos de Programas</b>	<b>17</b>
5.1	Exemplo 1 - Hello World com variável . . . . .	17
5.2	Exemplo 2 - Hello World sem variável . . . . .	17
5.3	Exemplo 3 - Hello World com concatenação de string e variável . . . . .	17
5.4	Exemplo 4 - Função Soma e Interação com usuário . . . . .	17
5.5	Exemplo 5 - Function, Condicionais, Repetição RT . . . . .	17
5.6	Exemplo 6 - Function, ENQ . . . . .	18

---

## 1 Sobre a linguagem VAL

A linguagem VAL foi criada com o intuito de ser simples e facilitar a programação, utilizando-se de uma estrutura baseada em *Tags*, assim como vemos na linguagem HTML e com palavras reservadas que são de fácil entendimento e memorização.

O código é de uso geral e se assemelha ao funcionamento de outras linguagens de programação, como C (no sentido de ser uma linguagem estruturada, procedural e de propósito geral) e Python (pela falta de necessidade de “;” ao final de cada linha de comando e pela “ausência” da declaração de variáveis).

## 2 Expressões Regulares

início	<MAIN></MAIN>
função	<FUNCTION></FUNCTION>
quebra de linha	\n
real	(“-”)? {dígito}+ (“.”){dígito}*
string	“({letra}*{dígito})*”
letra	{a-zA-Z}
dígito	{0-9}
booleano	{falso}   {vdd}
entrada	>>
saída	<<
atribuição	()
condicional	<SE></SE>
desvio condicional	<SENAO></SENAO>
desvio condicional encadeado	<ESE></ESE>
repetição	“<RT></RT>”   “<ENQ></ENQ>”
lógico	(“E”   “OU”   “!”)
variável / identificador	{letra}*({letra}   {dígito})*

Figura 1: Expressões Regulares

## 2.1 Operações Aritméticas

Soma	"+"
Subtração	"-"
Multiplicação	"X"
Divisão	"/"
Exponencial	"**"
Resto da divisão (mod)	"%"
Concatenação (string - string ou string-variável)	"+"

Figura 2: Operações Aritméticas

## 2.2 Operações Relacionais

Igual	=
Menor	<
Menor-Igual	<=
Maior	>
Maior-Igual	>=
Diferente	?

Figura 3: Operações Relacionais

## 2.3 Blocos de Comando


início	<main> (trocar "main" por qualquer outra função)
fim	</main> (trocar "main" por qualquer outra função) 

Figura 4: Blocos de Comando

## 2.4 Símbolos Especiais

separador	“.” (casas decimais)   “;” (estrutura repetição)   “,” (vetores e parâmetros)
abre e fecha parênteses	“( ” ”)” (indica prioridade de execução nas operações aritméticas) e (passagem de parâmetros)
comentário mesma linha	“###”
comentário várias linhas	“# {texto} #”
abertura função (tag)	<
fechamento função (tag)	>
finalização função (tag)	<\
abre e fecha aspas	“ ” ” (delimita string)
abre e fecha colchetes	[ ] (delimita condição e estrutura de repetição)

Figura 5: Símbolos Especiais

## 3 Tokens

EXPRESSÃO	PALAVRA RESERVADA (TOKEN)
inicioDoPrograma	MAIN
inicioDaFunção	FUNCTION
verdadeiro	vdd
falso	falso
se	SE
e se	ESE
senão	SENAO
repita	RT
enquanto	ENQ
e	E
ou	OU
retorna (função)	RETORNA
nulo	NULO
tamanho	TAM

Figura 6: Tokens

## 4 Regras de Utilização

### 4.1 Programa

Essa é a função principal do programa, onde estarão todas as operações e comandos a serem realizadas na linguagem. Assim como explicado anteriormente, os blocos de comandos devem vir entre *Tags*, e estas, por sua vez contêm as palavras reservadas que indicam sua "função".

#### 4.1.1 Autômato Função Principal

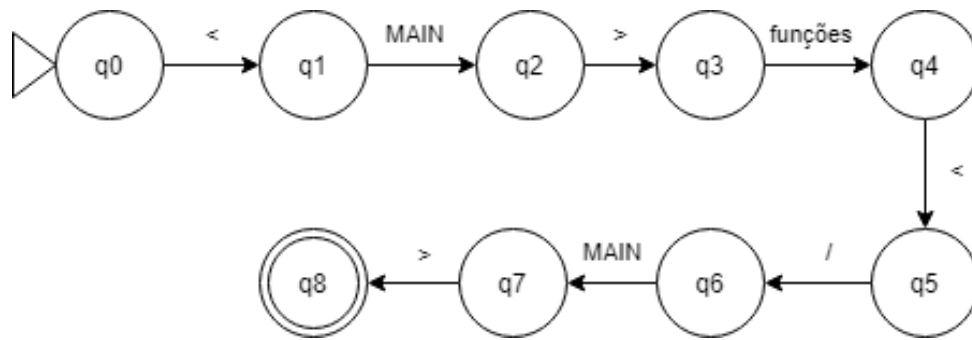


Figura 7: Função Main

### 4.2 Identificadores

São utilizados para atribuição de nomes às funções e variáveis definidas pelo usuário.

1. Todo identificador deverá iniciar por uma letra (a..z ou A..Z)
2. Não poderá conter símbolos especiais, onde após o primeiro caractere pode ser utilizado letras e dígitos.
3. Utiliza-se identificadores de, no máximo, 32 caracteres por estes serem significativos.
4. Não poderá ser uma palavra reservada, sequer nome de funções e variáveis já existentes no programa.

#### 4.2.1 Autômato Identificador

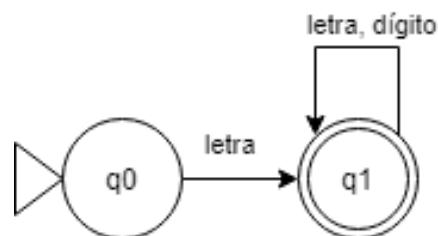


Figura 8: Identificador

### 4.3 Funções

A chamada de funções será idêntica a linguagens de programação que já conhecemos. Denotada pelo nome e a passagem de parâmetros opcionalmente.

Ex:

`funcao(parametro1, parametro2)`

#### 4.3.1 Autômato Função

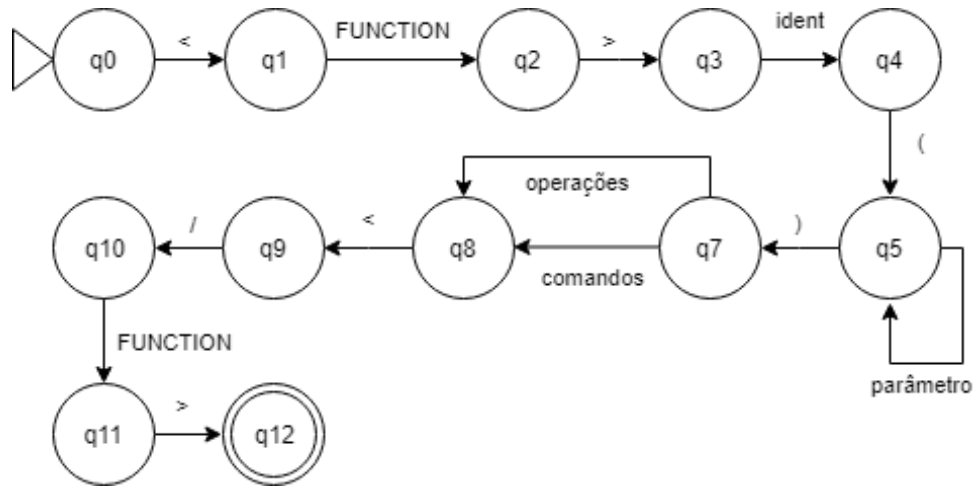


Figura 9: Função

#### 4.3.2 Autômato Parâmetro

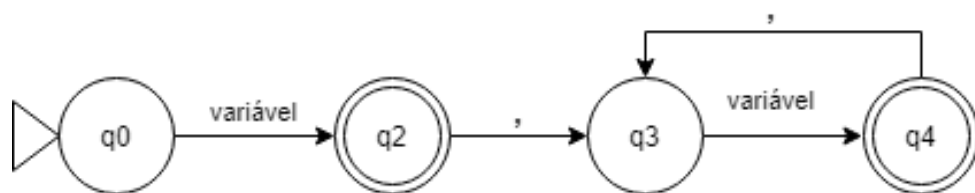


Figura 10: Parâmetro



## 4.4 Strings

Para determinar que um conjunto de letras/dígitos é de fato uma string é necessário o uso de aspas para que assim a linguagem possa interpretá-las. Os textos definidos como *Strings* são iniciados, obrigatoriamente, por uma letra e podem conter, na sequência, quantas letras e dígitos desejar.

Ex :

```
<< "Hello World"
```

### 4.4.1 Autômato String

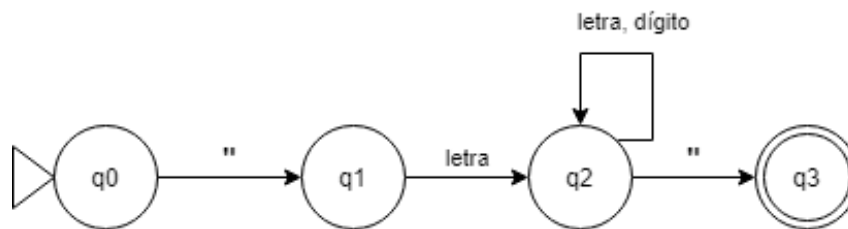


Figura 11: Strings

## 4.5 Reais

Os números Reais são compostos por números inteiros e racionais.

- Inteiro: deverão ser atribuídos com um menos (-) quando negativos e apenas o número, quando positivos.
- Racionais: deverão ser atribuídos com um menos (-) quando negativos e apenas o número, quando positivos; e quando necessário, adicionar um ponto (.) para separar as casas decimais.

### 4.5.1 Autômato Real

A seguir os números inteiros são representados pelos estados finais **q1** ou **q5** e os números racionais pelos estados **q3** ou **q7**.

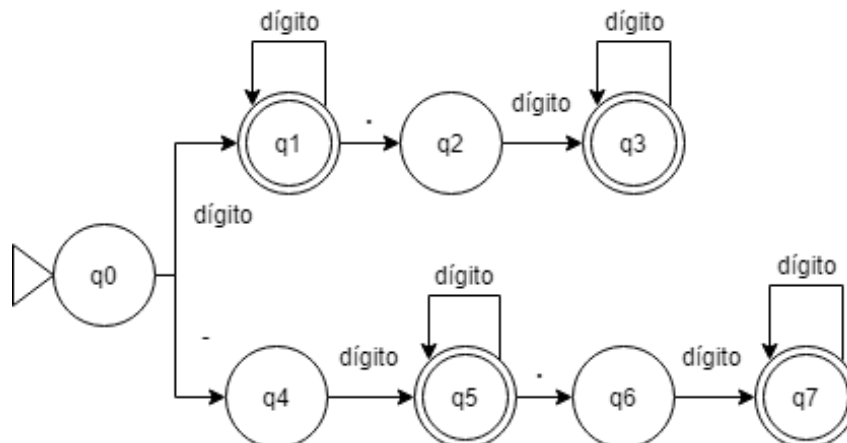


Figura 12: Real

## 4.6 Definição e Atribuição de Valores

Uma variável deverá ser definida por um identificador e seguida por uma atribuição, que é por meio dessa que a linguagem identificará automaticamente o tipo da variável. Assim, ao declarar uma variável obrigatoriamente deve haver uma atribuição, caso contrário, será interpretado como erro de sintaxe.

---

Ex 1:  
valor1(9.2)

---

Ex 2:  
palavra1("reconhece")

---

Ex 3:  
teste1 -> nao sera reconhecido!

---

### 4.6.1 Autômato Atribuição

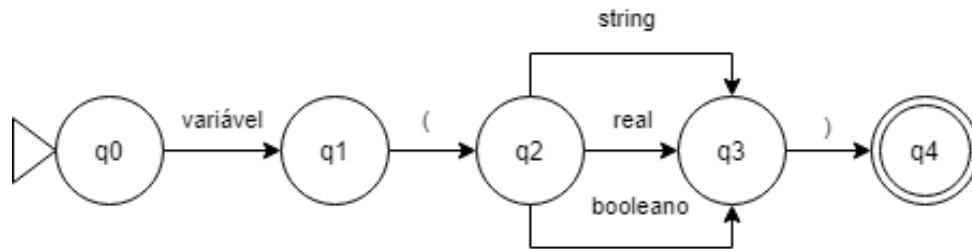


Figura 13: Atribuição

## 4.7 Condição

O autômato de condição pode estar presente tanto em estruturas condicionais quanto em estruturas de repetição como o ENQUANTO e em operações lógicas.

### 4.7.1 Autômato Condição

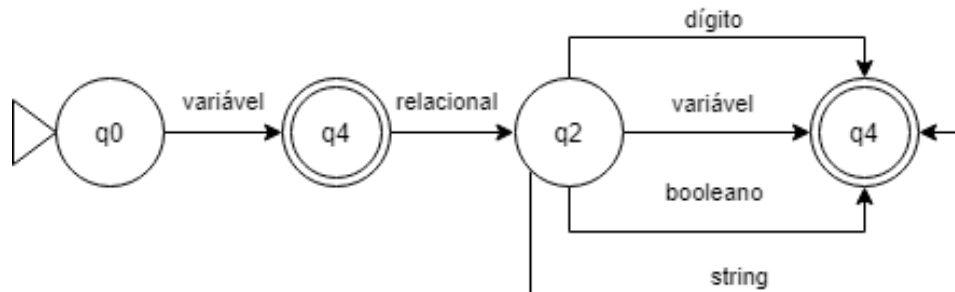


Figura 14: Condição

## 4.8 Entrada e Saída de Informação

O termo Entrada refere-se à informação que o usuário digita e submete ao console para ser identificada pelo programa e, portanto, é sempre vinculada a uma variável. É representada pelo símbolo >>. Já a Saída, representada pelo símbolo <<, é referente à informação que o programa retorna ao usuário, podendo ser tanto uma variável como um texto.

---

Ex:

```
<< variavelNome("Digite seu nome") -> saida  
>> Jose da Silva                    -> entrada
```

---

### 4.8.1 Autômato Entrada

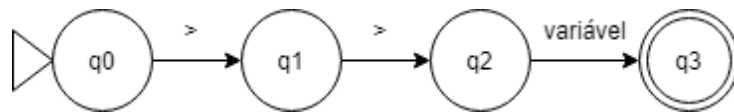


Figura 15: Entrada

### 4.8.2 Autômato Saída

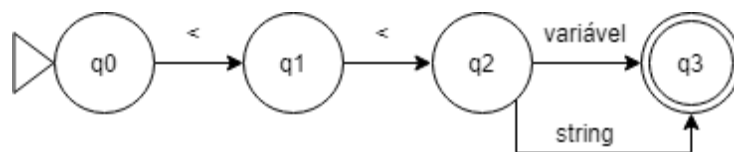


Figura 16: Saída

## 4.9 Operações Aritméticas

As operações aritméticas básicas utilizam os seguintes símbolos identificadores: Soma (+), Subtração (-), Divisão (/), Multiplicação (X), Exponenciação (^), Módulo (%). Para realizá-las, deve-se colocar uma variável antes e uma depois do símbolo desejado. Para estabelecer uma prioridade na realização dessas operações é necessário o uso de parênteses "()", sendo assim, a linguagem sempre realizará as operações mais internas previamente às outras.

Ex 1:

variavelC X (variavelA + variavelB)

Outra operação prevista é a de concatenação de Strings ou String com Variável, definida também pelo símbolo (+). A variável deve vir sozinha ou delimitado por parênteses caso deseje realizar uma operação, sendo que ao final toda a linha de comando é transformada em uma String. É importante denotar que a operação realizada entre parênteses resulta em um valor que será interpretado como variável.

Ex 2:

<<"Seu nome e: " + variavelNome

### 4.9.1 Autômato Aritmético

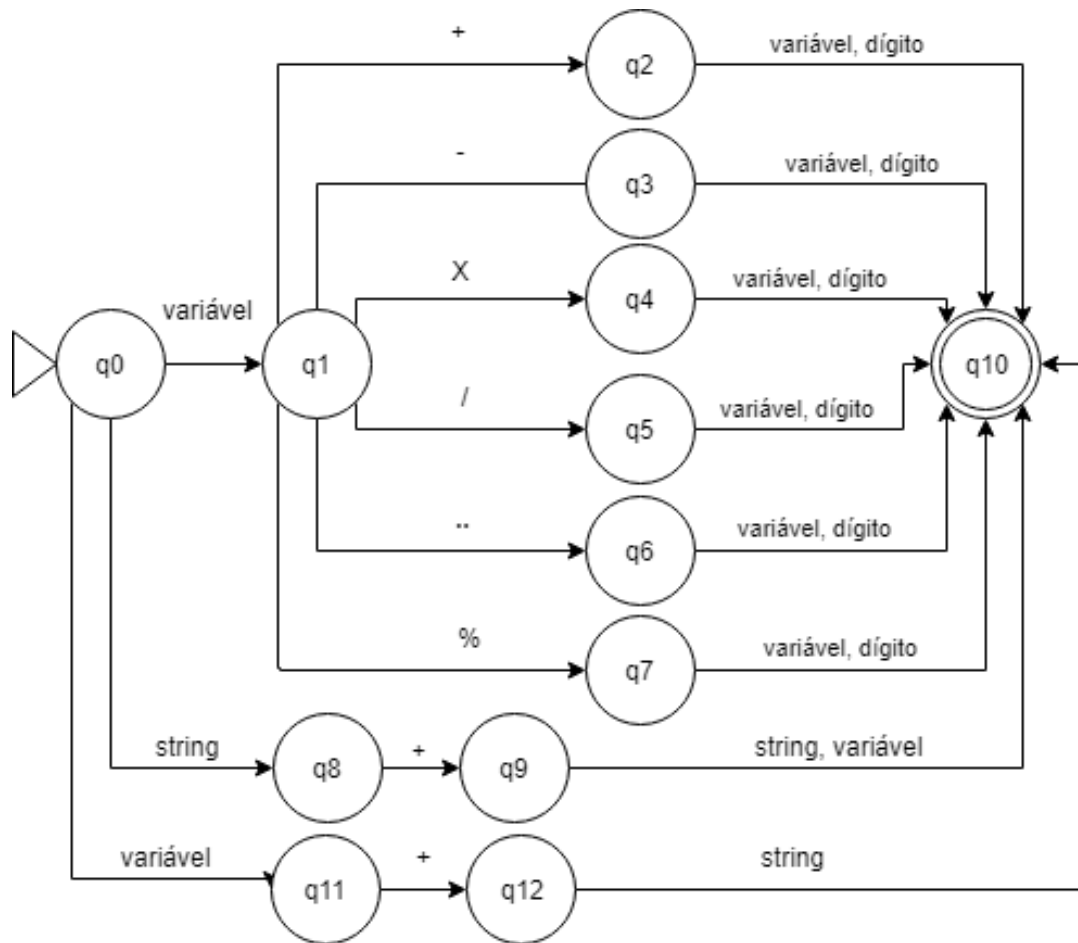


Figura 17: Aritméticos

## 4.10 Operações Relacionais

As operações relacionais são comparações utilizadas como Condições. Para utilizá-las, é preciso comparar sempre uma variável com outro valor, seja este uma variável, uma string, um dígito ou até um valor booleano.

---

Ex 1:

`variavelA > variavelB`

---

Ex 2:

`teste > variavelB`

---

### 4.10.1 Autômato Relacional

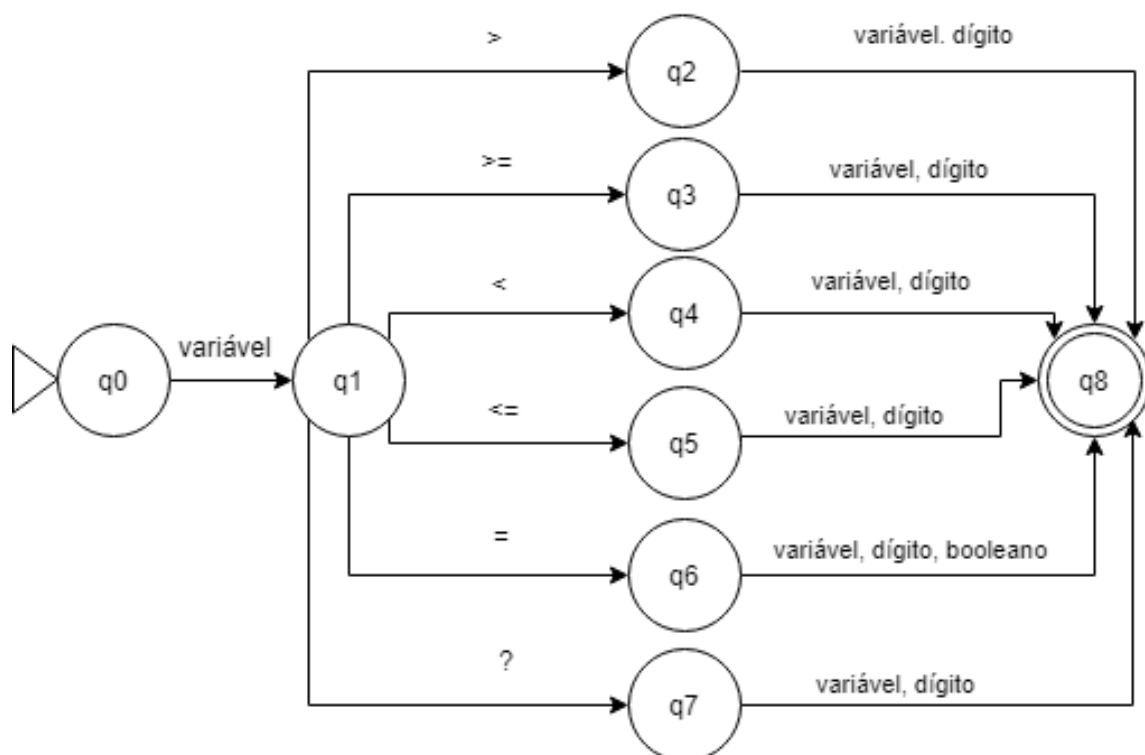


Figura 18: Relacionais

### 4.11 Operações lógicas

As operações lógicas são operações que podem ser representadas na Tabela Verdade, em formato binário, combinando condições simples. Seguem alguns exemplos na tabela abaixo:

Função	Exemplo de Representação
E (conjunção)	$[(a > b) \text{ E } (b \leq 5)]$
OU (disjunção)	$[(a > b) \text{ OU } (b \leq 5)]$
NÃO (negação)	$!b$

Figura 19: Tabela de Comandos Lógicos

Como pode-se notar, a operação de Negação (!) é vinculada a uma "variável" e, portanto, não é uma operação de comparação, diferentemente das operações de Conjunção (E) e Disjunção (OU), que são estruturas de condições lógicas comparativas.

Para utilizá-la, portanto, deve-se ter uma condição (que pode, ou não ser negada pelo operador "!"), a operação lógica e outra condição. Em um segundo caso, podemos ter a negação de uma condição de variável única.

#### 4.11.1 Autômato Lógico

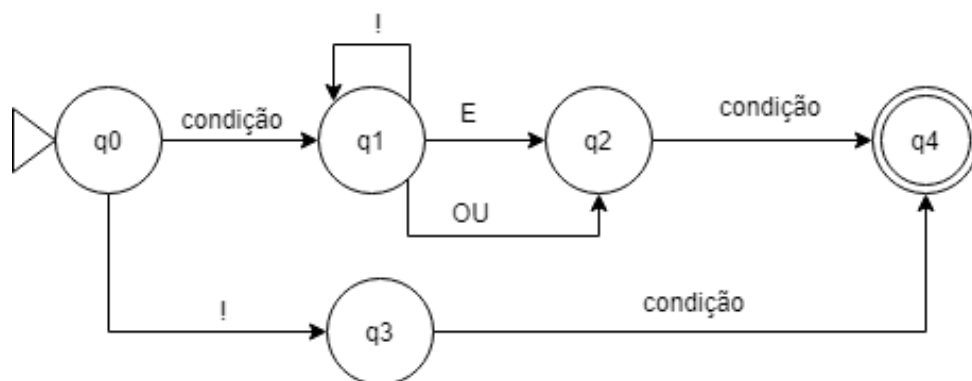


Figura 20: Comandos Lógicos

## 4.12 Condicionais

As condicionais são estruturas que determinam se um bloco de comandos será ou não executado, a partir de uma condição lógica ou relacional.

Para compor a estrutura condicional, é preciso utilizar o comando em formato de *Tag* (<E>, <ESE>, <SENAO>). Caso a opção seja o <SENAO>, não é necessário a atribuição de uma condição, basta colocar os comandos e operações e prosseguir com a finalização da *Tag* </SENAO>. Já para os dois primeiros casos <E> e <ESE>, deve-se adicionar a condição delimitada por colchetes "[ ]", seguida dos comandos e operações. Lembrando sempre ao final de fechar com a *Tag* correspondente.

Ex 1:

```
<SE> [condicao]
    #inserir comandos e operacoes aqui
</SE>
```

Ex 2:

```
<SENAO>
    #inserir comandos e operacoes aqui
</SENAO>
```

### 4.12.1 Autômato Condicional

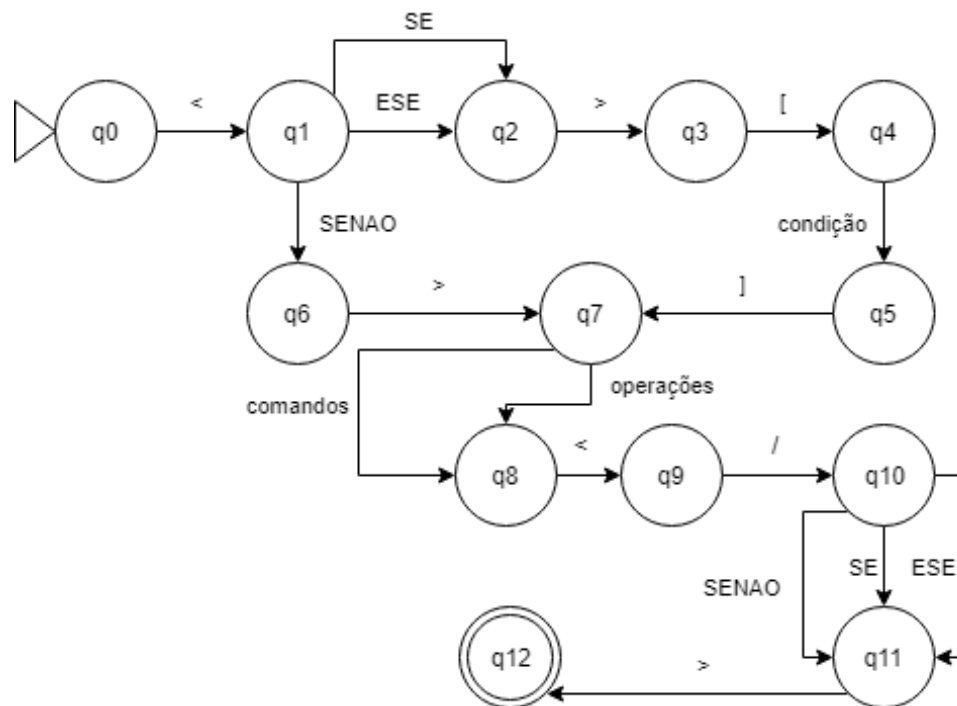


Figura 21: Comandos Condicionais

### 4.13 Repetição

O comando de repetição estabelece um laço de repetição através de um iterador, este por sua vez, tem a função de manter o laço por um número determinado de vezes, até o mesmo seja finalizado. Dentro, estarão contidos todos os comandos e operações a serem executados.

#### 4.13.1 Autômato Repetição

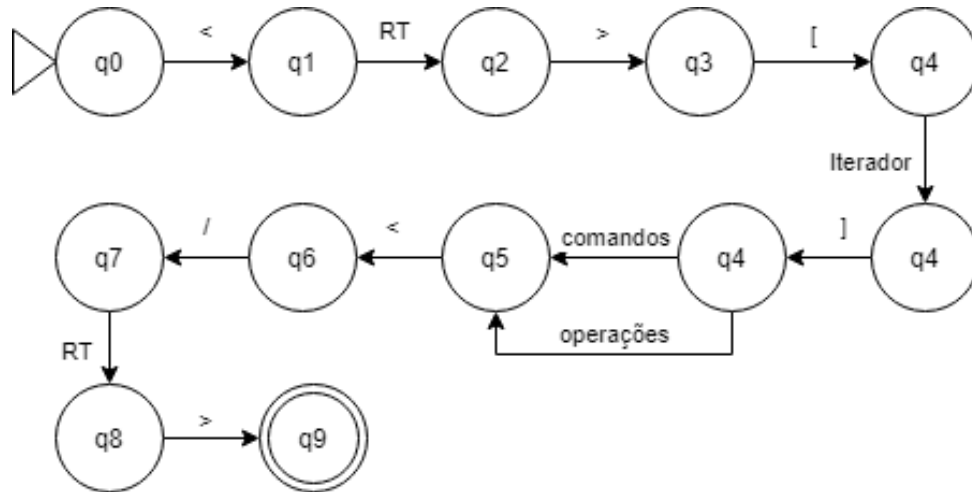


Figura 22: Repetição

#### 4.13.2 Autômato Iterador

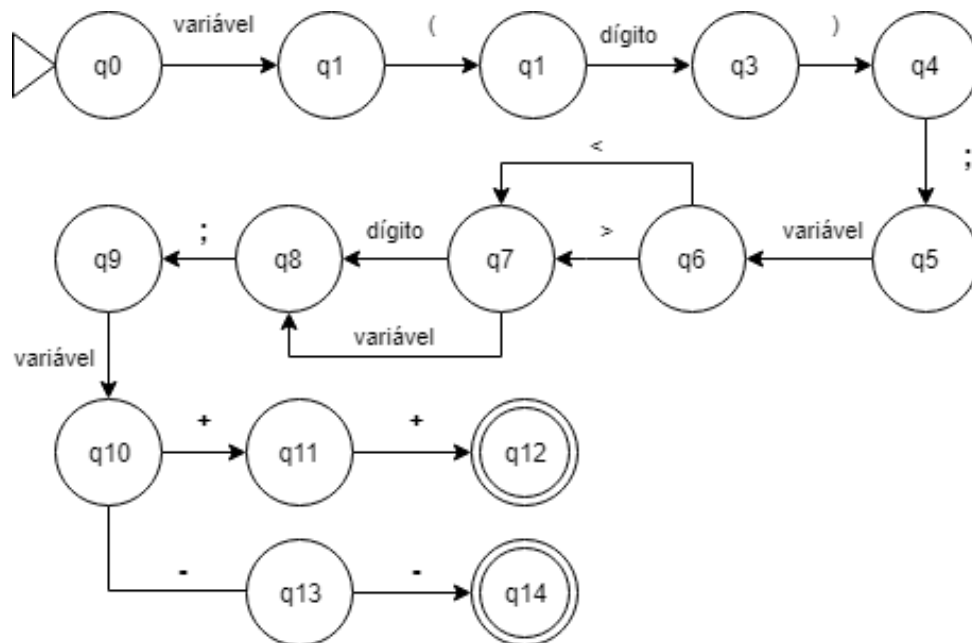


Figura 23: Iterador



## 4.14 Enquanto

O comando enquanto serve para fazer um laço de repetição utilizando-se de uma condição para que o mesmo continue em *Looping*, sendo que quando essa condição é quebrada, o laço é então finalizado. É semelhante ao Repetição, ele também traz comandos e operações no seu "interior".

### 4.14.1 Autômato Enquanto

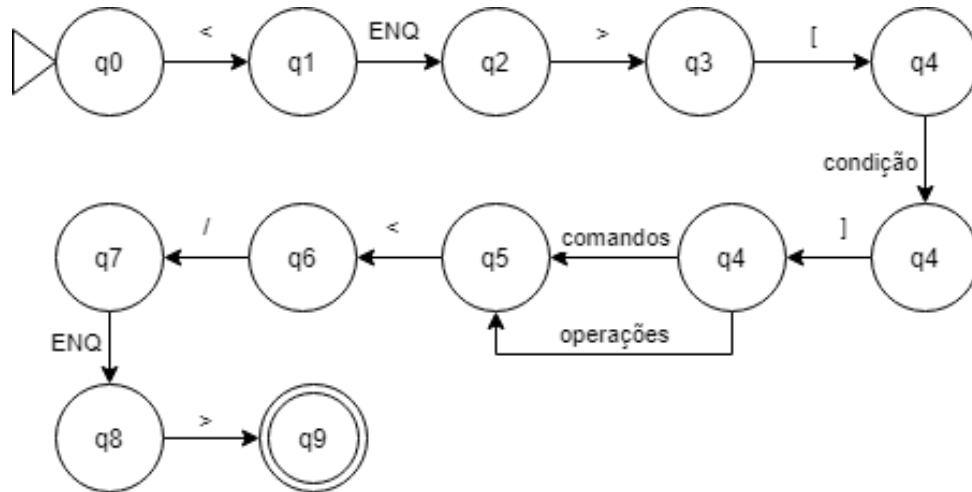


Figura 24: Enquanto

## 4.15 Quebra de linha

O comando de quebra de linha é vinculado a uma String (no comando de saída do terminal) e é utilizado quando deseja-se pular a linha. De forma simples, basta utilizar do comando `"\n"`.

### 4.15.1 Autômato Quebra de linha



Figura 25: Pular linha

## 5 Exemplos de Programas

### 5.1 Exemplo 1 - Hello World com variável

---

```
<MAIN>
  texto ("Hello World")
<< texto
</MAIN>
```

---

### 5.2 Exemplo 2 - Hello World sem variável

---

```
<MAIN>
  << "Hello World"
</MAIN>
```

---

### 5.3 Exemplo 3 - Hello World com concatenação de string e variável

---

```
<MAIN>
  texto ("Hello ")
  << texto + "World"
</MAIN>
```

---

### 5.4 Exemplo 4 - Função Soma e Interação com usuário

---

```
<MAIN>
  << v1("Por favor, insira o primeiro valor:")
  >> 1
  << v2("Por favor, insira o segundo valor:")
  >> 2
  << "A soma e:" + (v1+v2)
</MAIN>
```

---

### 5.5 Exemplo 5 - Function, Condicionais, Repetição RT

---

```
<FUNCTION> vasculhaVetor(vetor)
  par(0)
  impar(0)
  <RT> [i(0); i++; i < vetor.TAM]
    <SE> [ i % 2 = 0 ]
      par++
    </SE>
    <SENAO>
      impar++
    </SENAO>
  </RT>

  <SE> [par > impar]
    RETORNA "0 seu vetor tem mais numeros pares"
  </SE>
  <ESE> [par = impar]
    RETORNA "0 seu vetor o mesmo numero de numeros pares e impares"
```

---

```

    </ESE>
    <SENAO>
        RETORNA "0 seu vetor tem mais numeros impares"
    </SENAO>

</FUNCTION>

<MAIN>
    vetor (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    << vasculhaVetor(vetor) ##printa o retorno
</MAIN>

```

---

## 5.6 Exemplo 6 - Function, ENQ

---

```

<FUNCTION> vasculhaVetor(vetor)
    par(0)
    impar(0)
    i(0)
    <ENQ> [i < vetor.TAM]
        <SE> [ i % 2 = 0 ]
            par++
        </SE>
        <SENAO>
            impar++
        </SENAO>
        i++
    </ENQ>

    <SE> [par > impar]
        RETORNA "0 seu vetor tem mais numeros pares"
    </SE>
    <ESE> [par = impar]
        RETORNA "0 seu vetor o mesmo numero de numeros pares e impares"
    </ESE>
    <SENAO>
        RETORNA "0 seu vetor tem mais numeros impares"
    </SENAO>

</FUNCTION>

<MAIN>
    vetor (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    << vasculhaVetor(vetor) ##printa o retorno
</MAIN>

```

---