

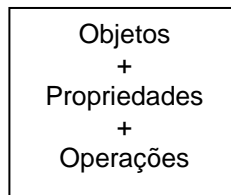
## Capítulo 4 – Algoritmos

### Introdução

### Conceitos importantes

- **Sistema:**  
Conjunto de objetos, dotados de propriedades características, capazes de se interagir, dentro de um determinado contexto.

#### SISTEMA



#### Exemplos:

Sistemas físicos	: êmbolo-pistão, geladeira
Sistemas químicos	: molécula, pilha
Sistemas biológicos	: célula, corpo humano
Sistemas matemáticos	: conjunto de equações
Sistemas ecológicos	: rio, mangue
Sistemas econômicos	: bolsa de valores, banco

- **Estado:**  
Conjunto de propriedades (atributos) relevantes dos objetos de um dado sistema, em um determinado instante.

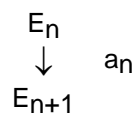
#### Representação:

$E_n$

#### Exemplos:

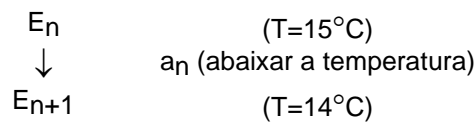
Sistema físico	(geladeira)	: temperatura, umidade
Sistema químico	( pilha )	: concentração, tensão
Sistema biológico	( célula )	: pressão osmótica, turgidez
Sistema matemático	( equação )	: valores, relações
Sistema ecológico	( mangue )	: população, poluição
Sistema econômico	( banco )	: ativo, passivo

- **Ação:**  
Evento que ocorre em um período de tempo finito estabelecendo um efeito intencionado e bem definido.

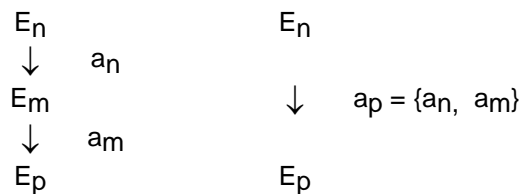


Toda vez que um determinado sistema estiver no estado  $E_n$ , e sofrer a aplicação da ação (operação)  $a_n$ , será levado ao estado  $E_{n+1}$ , após um certo intervalo de tempo.

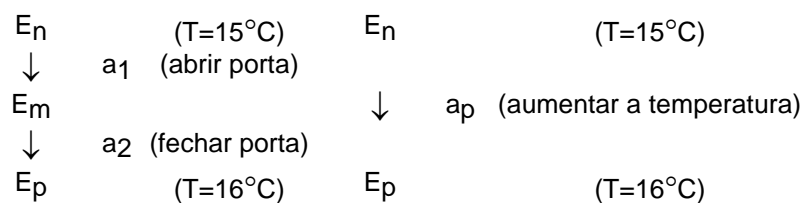
Exemplo: Sistema físico (geladeira): temperatura



- Processo:  
Sequência temporal de (sub)ações, cujo efeito acumulado é igual ao efeito de um único evento equivalente.



Exemplo: Sistema físico (geladeira): temperatura



- Programa:  
"Formulações concretas de algoritmos abstratos baseados em representações e estruturas específicas de dados." (Wirth)
- Algoritmo:

A palavra **algoritmo** deriva-se do nome de um matemático persa Abu ja'far Muhammad ibn-Musa Al-Khowarizmi (Algorimus, em latim)(780-850 d.C), professor do Instituto de Matemática de Bagdá e autor do livro Kitab al jabr w'al muqabala ("Regras de Restauração e Redução"). Este foi um dos primeiros textos escritos sobre Matemática, e responsável pela introdução da palavra álgebra ("redução", em árabe). Deve-se a este matemático o desenvolvimento dos primeiros procedimentos formalizados, passo-a-passo, para a realização de operações aritméticas, e em homenagem ao seu pioneirismo, qualquer procedimento formalizado recebe este nome.

Um algoritmo é uma descrição de um padrão de comportamento (o quê fazer) expressa em termos de um repertório finito e bem inteligível de ações primitivas (como fazer), as quais, supõe-se, *a priori*, sejam possíveis de se executar.

A noção de um algoritmo, de uma ordem executável para o estabelecimento de um efeito final, é muito comum na vida cotidiana receitas, manuais, partituras etc.

Ao escrever um algoritmo, começa-se considerando o acontecimento como um processo, dividindo-o em uma sequência de (sub)ações que deverão ser realizadas sucessivamente.

É necessário o uso de um conjunto de mecanismos que permita o desenvolvimento de algoritmos, e que seja suficientemente conciso para evitar ambiguidades, ao mesmo tempo em que procure libertar o programador do rigor e das limitações de uma linguagem de programação específica.

A seguir encontra-se uma proposta de definição dos elementos deste conjunto.

## Elementos de um algoritmo

### Alfabeto

Um algoritmo pode conter os seguintes símbolos:

- as letras do alfabeto padrão inglês:

A B C D E F G H I J K L M N O P Q R S T U V X W Y Z

a b c d e f g h i j k l m n o p q r s t u v x w y z

- os dez algarismos:

0 1 2 3 4 5 6 7 8 9

- outros símbolos:

+	- soma	>	- maior	&	- ampersete
-	- hífen	<	- menor		- barra em pé
*	- asterisco	=	- igual	~	- til
/	- barra	≥	- maior ou igual		
( )	- parênteses	≤	- menor ou igual		
{ }	- chave	≠	- diferente		
"	- aspas	!	- exclamação	#	- sustenido
.	- ponto	?	- interrogação	@	- arroba
,	- vírgula	%	- porcentagem	\	- contra-barra (barra invertida)
:	- dois pontos	—	- travessão		
;	- ponto-e-vírgula	^	- circunflexo		

### Comentários

São informações acrescentadas a um algoritmo com o objetivo de identificá-lo, explicar a sua função e esclarecer trechos.

Forma geral:

! texto !

Exemplo:

! ESTE É UM TRECHO DE COMENTÁRIO !

### Pseudo-comandos

São descrições genéricas de trechos de algoritmo que ainda deverão ser desenvolvidos, até que possam ser expressos em termos dos comandos e estruturas de controle básicos. Recomenda-se que, ao se desenvolver a idéia expressa na sentença, o pseudo-comando passe a fazer parte do algoritmo como um comentário.

Forma geral:

? texto ?

Exemplo:

em alguma versão do algoritmo:

? Calcular a soma de dois números ?

na versão final do algoritmo:

! Cálculo da soma de dois números !

### Forma geral de um algoritmo

```

início

! identificação !

! abstrações de dados !
! definições de dados !

! abstrações de comandos !
! definições de comandos !

fim.

```

#### Exemplo:

Fazer um algoritmo para ler dois valores do teclado e mostrar a sua soma na tela.

#### Primeiro passo:

Identificar o objetivo do algoritmo.

```

início

? ler dois valores inteiros e mostrar sua soma ?

fim.

```

#### Segundo passo:

Isolar processos.

```

início

! ler dois valores inteiros e mostrar sua soma !

? ler dois valores inteiros ?
? mostrar sua soma ?

fim.

```

#### Terceiro passo:

Isolar ações consideradas primitivas.

```

início

! ler dois valores inteiros e mostrar sua soma !

? definir um local para armazenar o primeiro valor ?
? definir outro local para armazenar o segundo valor ?
? ler um valor do teclado e armazená-lo ?
? ler outro valor do teclado e armazená-lo ?
? mostrar a soma dos valores armazenados ?

fim.

```

A partir deste simples procedimento de identificação, isolamento e definição, um algoritmo vai sendo construído, passo a passo, contendo cada etapa mais informações e detalhes que a anterior. Quando se tiver atingido o objetivo desejado através de uma descrição minuciosa de uma sequência bem definida de ações, então pode-se levá-la à implementação.

## Componentes de um algoritmo

### Valores

Valores são expressões de entidades que contenham informações (dados) pertinentes ao processo descrito pelo algoritmo.

Quanto à utilização podem ser classificados como:

- constantes  
quando não se modificam durante a execução de um algoritmo
- variáveis  
quando podem se modificar durante a execução de um algoritmo.

### Tipos

Um tipo é um conjunto de valores que apresentem as mesmas características e comportamento uniforme quando submetidos a operações associadas.

Um valor qualquer  $x$  pertence a um tipo  $T$ , simplesmente se  $x \in T$ .

Uma expressão  $E$  é de um tipo  $T$ , se o resultado da avaliação de  $E$  for um valor  $x$  que pertença a  $T$ .

Pode-se separar os seguintes tipos de valores:

- básicos (simples ou compostos)
- abstrações
- referências (apontadores)

Forma geral:

tipo <nome> = <descrição>

### Enumeração

Um tipo pode ser completamente definido a partir da enumeração de seus valores.

As enumerações são conjuntos ordenados (ou escalares) de constantes, cujo valor é o seu número de ordem, a partir de um (1), ou é um valor que lhe é conferido segundo uma ordem crescente.

Enumerações podem ser usadas para a definição de intervalos fechados, contínuos ou discretos.

Forma geral:

{ <itens> }

onde <itens> é uma especificação de intervalo:

- contínua:

<limite inferior>: <limite superior>

- discreta:

<item 1> , ... , <item N>  
 ou  
 <item 1> , <item 2>: <item 3>  
 ou  
 <item 1>:<item 2> , <item 3>:<item 4>  
 ou  
 <nome 1>  $\leftarrow$  <valor 1> , ... , <nome N>  $\leftarrow$  <valor N>

Exemplos:

- { "A","E","I","O","U" } - define uma enumeração de letras
- { "A": "Z" , "a": "z" } - define uma enumeração de letras
- { 1: 100 } - define uma enumeração de inteiros entre 1 e 100
- { 0 , 1 } - define uma enumeração de inteiros com valor 0 e 1

### Tipos básicos

Os tipos básicos estão associados às representações e manipulações de dados em uma determinada arquitetura de computadores.

- Bit

tipo BIT = {0,1}

Valores do tipo BIT só podem ser: 0 ou 1.

- Lógico

tipo LÓGICO = BIT

Segundo a associação: { FALSO = 0, VERDADEIRO = 1 }.

- Byte

tipo BYTE = BIT (8)

Valores compostos por (8) BITS: { 0000 0000, ... , 1111 1111 }.

- Caractere

tipo CARACTERE = BYTE

Valores correspondentes aos símbolos representados por bytes, segundo um código, por exemplo ASCII:

{ ^@	= 0000 0000, ... ,	
"0"	= 0011 0000, ... ,	"9" = 0011 1001, ... ,
"A"	= 0100 0001, ... ,	"Z" = 0101 1100, ... ,
"a"	= 0110 0001, ... ,	"z" = 0111 1100, ... ,
DEL	= 0111 1111 }	

- Inteiro

tipo INTEIRO = BYTE (n) para  $n > 0$

Exemplos:

BYTE (1) = 0000 1111 = 15 { -128, ..., +127 }  
 BYTE (2) = 0000 0000 0000 1111 = 15 { -32768, ..., 32767 }

- Real

tipo REAL = BYTE (n) para  $n > 0$

Segundo alguma descrição, de ponto fixo ou flutuante.

Exemplos:

Ponto fixo:

BYTE (1) = 1001.0100 = -1.25

Ponto flutuante:

BYTE (1) = 1 101 1010 = -1.25

Generalizando para notação científica (expoente):

sem expoente	com expoente	
100.0	1.0E+2	= $(1.0 \times 10^2)$
-3251.6	-3.2516E3	= $(-3.2516 \times 10^3)$
0.48	4.8E-1	= $(4.8 \times 10^{-1})$
-0.328	-3.280E-1	= $(-3.280 \times 10^{-1})$

Observações:

Utilizar o ponto ( . ) ao invés da vírgula ( , ) decimal.

Em geral,  $n = 4$  é o valor mais utilizado para os inteiros e os reais.

- Caracteres

tipo CARACTERES = BYTE (n) para  $n > 0$

Exemplos:

BYTE (1) = "" = $\epsilon$	- vazio
BYTE (1) = " " = $\beta$	- espaço em branco
BYTE (2) = "" = $\epsilon$	- vazio
BYTE (2) = " "	- espaços em branco
BYTE (2) = "A"	- uma letra maiúscula
BYTE (2) = "Aa"	- duas letras

Generalizando para valores literais:

"	= $\epsilon$	- constante literal vazia
" "	= $\beta$	- constante literal espaço
"0"		- símbolo do número zero
"JOÃO"		- palavra
"Esta também é uma constante"		- sentença

Observações:

- É necessário o uso de aspas em valores literais.
- Os caracteres podem ter um tamanho variável entre [0: 255].
- Em geral,  $n = 255$  é o valor mais utilizado para as cadeias de caracteres.

### Variáveis

São representações simbólicas de posições de memória do computador, de tamanho determinado, onde valores podem ser armazenados, e modificados, durante a execução de um algoritmo.

Uma variável é identificada por um nome associado a um endereço.

O valor inicial de uma variável, por definição, é indefinido, a menos que algum comando o especifique.

Exemplo:

ENDEREÇO	MEMÓRIA	NOME
1111		
	...	
1100	0000 1010	X
	...	
0000		

A variável de nome (X), está localizada no endereço binário 1100, e tem o valor binário 0000 1010, ou seja, o conteúdo da memória naquela posição é  $X = 10$ .

Determinação de endereço e conteúdo a partir da variável:

Forma geral:

&<nome>	(lê-se "endereço de")
@<endereço de nome>	(lê-se "conteúdo do endereço")

Exemplos:

&X	= 1100	
@(&X)	= 0000 1010	(ou simplesmente, $X = 10$ )



### - Identificadores:

Identificadores são nomes associados a conteúdos, descrições ou posições de memória, servem para representar constantes, variáveis, ou outras abstrações.

Regras para formação de identificadores:

- iniciar-se por uma letra;
- ter um tamanho definido e constante;
- pode conter outras letras ou algarismos;
- não deve conter espaços em branco;
- pode-se separar palavras utilizando-se o travessão ("\_").

Definição de tipos de variáveis

Seguindo o exemplo da Matemática, todas variáveis (como também expressões e constantes) têm um tipo associado. O tipo da variável caracteriza a classe de valores que poderá assumir, isto significa que cada variável pode ter uma representação com tamanho diferente das demais.

Exemplo:

$$x + 5.2 = 7.4 \text{ (em } \mathbf{R}) \Rightarrow x \in \mathbf{R}$$

A variável (x) deverá ser definida como real.

Forma geral:

<tipo> <lista de identificadores>

Exemplos:

inteiro	X
real	Y,Z
caractere	Letra
caracteres	Nome_da_Rua, Endereço
lógico	PRIMEIRO, ÚLTIMO, ACHOU

Observações:

- É necessário o uso de vírgulas como separador na lista de nomes de variáveis.
- Os nomes, em um determinado contexto, são únicos, não podendo ser usados repetidamente, mesmo que com tipos diferentes.
- É conveniente manter-se o uso de maiúsculas, ou minúsculas, de acordo com a definição, evitando o uso indistinto.

Contra-exemplos:

real	X/Y, _ASSIM_NÃO
inteiro	Y?, 1Z
caractere	#
caracteres	Nome da Rua, End.
lógico	X, x

## Exercícios

1. Definir uma enumeração para representar apenas as vogais.
2. Definir uma enumeração para representar os cinco primeiros pares.
3. Qual a faixa de valores inteiros representáveis com 4 bytes ?
4. Qual a faixa de valores reais representáveis com 4 bytes, usando 7 bits como exponte, em ponto flutuante ?
5. Relacionar endereços e conteúdos para o esquema abaixo:

ENDEREÇO	MEMÓRIA	NOME
1111		
	...	
0101		Z
0100		Y
0011	0000 1010	X
	...	
0000		

- a.)  $\text{@}(\&Y)$  =  $\text{@}(\&X)$
- b.)  $\text{@}(\&Z)$  =  $\text{@}(\&X) + \text{@}(\&Y)$
- c.)  $\text{@}(\&X+1)$  = 0000 0000
- d.)  $\text{@}(\&Y+1)$  = 0000 0001
- e.)  $\text{@}(\&Y)$  =  $\text{@}(\&Y-1) - \text{@}(\&Y+1)$

## Expressões

Expressões são descrições de transformações de valores de mesmo tipo, ou de tipos diferentes; servem para descrever operações capazes de alterar conteúdos de um objeto.

Pode-se ter expressões de qualquer um dos tipos básicos ou para conversões entre estes.

- aritméticas inteiras ou reais
- lógicas
- literais

### Expressões aritméticas

#### - Funções mais comuns

Nome	Argumento	Descrição
abs (x)	x: real	valor absoluto de x
exp (x)	x: real	$e^x$
ln (x)	x: real	$\log_e x$ ( $x > 0$ )
raiz (x)	x: real	raiz quadrada de x
cos (x)	x: real	cosseno de x (radianos)
sen (x)	x: real	seno de x (radianos)
arctg (x)	x: real	arco-tangente x (radianos)

#### - Operadores aritméticos

Símbolo	Operação
$\wedge$	potenciação
*	multiplicação
/	divisão
<b>div</b>	quociente inteiro
<b>mod</b>	resto inteiro
+	adição
-	subtração

Exemplos:

Matemática	Algoritmo
$2x + 5^3$	$2 * x + 5 ^ 3$
$\frac{x + 3}{x + 1}$	$(x + 3) / (x + 1)$

Observação:

Os operadores **div** e **mod**, são operadores especiais de divisão inteira: resto e quociente, respectivamente.

Exemplos:

$5 \text{ div } 2 = 2$	$4 \text{ div } 2 = 2$
$5 \text{ mod } 2 = 1$	$4 \text{ mod } 2 = 0$

## - Regras

- a.) A avaliação das operações é feita no sentido de leitura da expressão, se de mesma hierarquia.
- b.) Um operador aritmético nunca pode ficar implícito.
- c.) A sequência de operações é a seguinte:
  - avaliação de parênteses, para resolver problemas de prioridade, executando-se as operações dentro de parênteses mais internos primeiro;
  - cálculo de funções;
  - exponenciação (^);
  - multiplicação ou divisão (\* ou /, **div**, **mod**);
  - adição ou subtração (+ ou -).

### Observações:

- Para efeito de clareza, ou para mudar a precedência de operadores, pode-se separar as expressões por meio de parênteses.
- Os operadores associam-se, preferencialmente, da esquerda para a direita, com exceção do operador de exponenciação.
- Respeitam-se os tipos; exige-se conversões entre representações diferentes.

### Exemplos:

Nos exemplos abaixo, chama-se  $R_1, R_2, \dots, R_n$  aos resultados das operações, de acordo com a prioridade dos operadores.

- 1.)  $a * (c * b + (c - d) ^ 2)$   
 $a * (c * b + R_1 ^ 2)$   
 $a * (c * b + R_2)$   
 $a * (R_3 + R_2)$   
 $a * R_4$   
 $R_5$
- 2.)  $(c - d) * e ^ (k / (x + y) ^ 4)$   
 $R_1 * e ^ (k / (x + y) ^ 4)$   
 $R_1 * e ^ (k / R_2 ^ 4)$   
 $R_1 * e ^ (k / R_3)$   
 $R_1 * e ^ R_4$   
 $R_1 * R_5$   
 $R_6$
- 3.)  $(-c + (d ^ 2 - 5/a + c) ^ 0.5) / (6 * k)$   
 $(-c + (R_1 - 5/a + c) ^ 0.5) / (6 * k)$   
 $(-c + (R_1 - R_2 + c) ^ 0.5) / (6 * k)$   
 $(-c + (R_3 + c) ^ 0.5) / (6 * k)$   
 $(-c + R_4 ** 0.5) / (6 * k)$   
 $(-c + R_5) / (6 * k)$   
 $R_6 / (6 * k)$   
 $R_6 / R_7$   
 $R_8$

## Exercícios

1. Utilizando  $R_1, R_2, \dots, R_n$  como resultados de operações, representar, nas expressões abaixo, a prioridade das operações:

a)  $(B - C \wedge E) + (3 * (C \text{ div } D - K) \wedge I) - J \text{ mod } L$

b)  $(A + B \wedge C - 8 \text{ div } C * D) \wedge K * J - 3 * A/B$

c)  $((A * B * C + D) * 3 - J/4 * M) * M - J \text{ div } (K+L)$

2. Transformar as expressões aritméticas abaixo, da linguagem matemática para a representação algorítmica:

a)  $X^2 - Y^3$

b)  $\sqrt{P(P-A)(P-B)(P-C)}$

c)  $\pi (A+B)(A^2 + (B-C)^2)^{1/2}$

d)  $\frac{-B + (B^2 - 4AC)^{0.5}}{2A}$

e)  $\frac{1}{(B^2 - A^2)^3}$

3. Dado que as variáveis reais A, B, C, D valem, respectivamente, 2, 1, 3, 0.5 e que as variáveis inteiras J, K, L valem, 3, 4, 10, determinar os resultados das seguintes expressões:

a)  $K / J * L$

b)  $L \text{ div } K$

c)  $A * L + B - C / D - K * J$

d)  $D / A + B * C$

e)  $L \wedge 3 + C \text{ mod } A * D$

## Expressões literais

## - Funções predefinidas

Nome	Argumento	Descrição
símbolo (x)	x: caractere(s)	retirar o primeiro símbolo
concatenar (x,y)	x: caractere(s) y: caractere(s)	concatenar (junta) x com y
maiúscula (x)	x: caractere	passar letra para maiúscula
minúscula (x)	x: caractere	passar letra para minúscula

## Exemplos:

Prática	Algoritmo
(vazio)	$\varepsilon$ ou ""
"a"	"a"
"a" $\varepsilon$	símbolo ("a")
"a" + $\varepsilon$	concatenar ("a", $\varepsilon$ )
"abcdefgh"	
"a" "bcdefgh"	símbolo ("abcdefgh")
"a" + "bcdefgh"	concatenar ("a","bcdefgh")

## Observações:

- Para efeito de avaliação de ordem considera-se que os símbolos são representados internamente por um código numérico padronizado (por exemplo, o código ASCII).

## Exemplo:

"A" é menor que "a"

"A" é maior que "0"

$\beta$  é o menor de todos

"~" é o maior de todos

A comparação de duas cadeias de caracteres é feita símbolo a símbolo. A menor delas será a de menor número de símbolos necessários para decisão, se comparados os códigos correspondentes.

## Exemplo:

"BANANA"<"BANANa"<"BANana"<"banana"<"BANANAS"

## Exercícios

1. Mostrar o resultado das operações abaixo:

X = "ae"  
 Y = "I"  
 W = "ou"  
 Z =  $\beta$

- a) concatenar (X,Y)
- b) concatenar (concatenar(X,Y),W)
- c) concatenar (maiúscula(X),concatenar(Y,W))
- d) concatenar (maiúscula(X),maiúscula(concatenar(Y,W)))
- e) concatenar (concatenar(X,Z),concatenar(concatenar(Y,Z),W))

2. Mostrar os resultados relacionados das operações abaixo:

X = "123"  
 Y = "4"  
 W = "56"  
 Z = "-"

- a) concatenar (Z, símbolo(X))
- b) concatenar (X, símbolo(W))
- c) símbolo (concatenar (Y, Z))
- d) símbolo (símbolo (X))
- e) símbolo (concatenar (símbolo (Y), Z))

3. Mostrar o resultado das operações abaixo:

X = "abc"  
 Y = "Ab"  
 W = "ABC"  
 Z = "aB"

- a)  $X > Y$  ?
- b)  $X < W$  e  $Z > X$  ?
- c)  $W < Z$  ou  $(Y > W \text{ e } X < Z)$  ?

## Expressões lógicas

## - Operadores relacionais

Símbolo	relação
>	maior que
<	menor que
=	igual
≥	igual ou maior que
≤	igual ou menor que
≠	diferente

## - Conectivos lógicos

Símbolo	relação
&	e
	ou
~	não

## Observação:

Esses conectivos também seguem regras de prioridade.

## Exemplos:

Abaixo,  $R_1, \dots, R_n$  serão resultados aritméticos, e  $p_1, p_2, \dots, p_n$  serão resultados lógicos, de acordo com a prioridade:

1.)  $a * (c * b + (c - d) ^ 2) > c + 4 * d$   
 $a * (c * b + R_1 ^ 2) > c + 4 * d$   
 $a * (c * b + R_2) > c + 4 * d$   
 $a * (R_3 + R_2) > c + 4 * d$   
 $a * R_4 > c + 4 * d$   
 $R_5 > c + 4 * d$   
 $R_5 > c + R_6$   
 $R_5 > R_7$   
 $p_1$

2.)  $(c - d) * e ^ (k / (x + y) ^ 4) > 0 \& c * k - (x + y) < 4$   
 $R_1 * e ^ (k / (x + y) ^ 4) > 0 \& c * k - (x + y) < 4$   
 $R_1 * e ^ (k / R_2 ^ 4) > 0 \& c * k - (x + y) < 4$   
 $R_1 * e ^ (k / R_3) > 0 \& c * k - (x + y) < 4$   
 $R_1 * e ^ R_4 > 0 \& c * k - (x + y) < 4$   
 $R_1 * R_5 > 0 \& c * k - (x + y) < 4$   
 $R_6 > 0 \& c * k - (x + y) < 4$   
 $p_1 \& c * k - (x + y) < 4$   
 $p_1 \& R_7 - (x + y) < 4$   
 $p_1 \& R_7 - R_8 < 4$   
 $p_1 \& R_9 < 4$   
 $p_1 \& p_2$   
 $p_3$

3.)  $(-c + d) > 6 \& (a + c) < -5 \mid a = 0$   
 $R_1 > 6 \& (a + c) < -5 \mid a = 0$   
 $p_1 \& (a + c) < -5 \mid a = 0$   
 $p_1 \& R_2 < -5 \mid a = 0$   
 $p_1 \& p_2 \mid a = 0$   
 $p_1 \& p_2 \mid p_3$   
 $p_4 \mid p_3$   
 $p_5$



## Exercícios

1. Utilizando  $R_1, R_2, \dots, R_n$  como resultados de operações aritméticas, e  $p_1, p_2, \dots, p_n$  como resultado de operações lógicas, representar, nas expressões abaixo, a prioridade das operações:

a)  $(B - C \wedge E > 0) \mid (3 * (C \text{ div } D - K) \wedge M \leq J \bmod L)$

b)  $(A + B \wedge C < X) \& (X - 8 \text{ div } C * D) \wedge K > A/B \mid H = 0$

c)  $\sim ((A * B + C) > D \& J/4 * M < N * K) \mid J \text{ div } (K+L) = 0$

2. Transformar as expressões lógicas abaixo, da linguagem matemática para notação algorítmica:

a)  $X^2 - Y^3 > X^{(A+4)} \quad e \quad A < 5$

b)  $0 < \sqrt{P(P-A)(P-B)(P-C)} < P^2 + 2P + 1$

c)  $\pi (A+B)(A^2 + (B-C)^2)^{1/2} = 0 \quad e \quad (A=B \text{ ou } A+C > 0)$

d)  $\frac{-B + (B^2 - 4AC)^{0.5}}{2A} \quad e \quad \text{não } (A=0)$

e)  $\frac{1}{(B^2 - A^2)^3} \quad e \quad \text{não } (A=0 \quad e \quad B=0)$

3. Sabendo-se que as variáveis reais A, B, C, D valem, respectivamente, 2, 1, 3, 0.5 e que as variáveis inteiras J, K, L valem, 3, 4, 10, determinar os resultados finais das seguintes expressões:

a)  $K / J * L > 0 \& K * J < 5$

b)  $L \text{ div } K < 10 \& K \neq 0.5$

c)  $A * K + B \geq 8 \& -C / D < 20 \mid -K * J > -10$

d)  $\sim (D / A + B > 0) \& \sim (B * C \leq 4)$

e) **verdadeiro**  $\& (I \wedge 3 + D * C > K \text{ div } A)$

## Transferências de valor

- Forma geral:

$$\langle \text{dado} \rangle \leftarrow \langle \text{valor} \rangle$$

O valor é transferido para o lugar de memória se for de tipo compatível, senão ocorrerá erro.

Exemplos:

inteiro	X, Y
real	P, Q
caractere	A
caracteres	S
lógico	M, N

$X \leftarrow 0$  ( X recebe o valor zero inteiro )

$Y \leftarrow X + 1$  ( Y recebe o valor de X mais 1 )

$P \leftarrow 0.0$  ( P recebe o valor zero real )

$Q \leftarrow P$  ( Q recebe o valor de P )

$P \leftarrow X$  ( P recebe o valor inteiro de X )

$P \leftarrow \exp(X)$  ( Y recebe e = 2,718... elevado a X )

$Q \leftarrow \text{raiz}(P)$  ( Q recebe o valor da raiz de P )

$M \leftarrow \text{verdadeiro}$  ( M recebe o valor lógico V )

$N \leftarrow \text{falso}$  ( N recebe o valor lógico F )

$M \leftarrow N$

$A \leftarrow \varepsilon$  ( A recebe o símbolo vazio )

$S \leftarrow "A"$  ( S recebe a letra A )

$A \leftarrow \beta$  ( A recebe o espaço em branco )

$S \leftarrow \text{"sentença"}$

$S \leftarrow \text{concatenar} ("A ", \text{símbolo}(S))$

Contra-exemplos:

$0 \leftarrow X$  ( do lado esquerdo, sempre é uma variável )

$0 \rightarrow X$  ( não é considerada a inversão de sinal )

Se no lugar do valor houver uma expressão, seu valor será calculado, e o resultado transferido para a memória.

Exemplos:

$$Y \leftarrow 2 * X + 5 ^ 3$$

Suponha que (x) seja, neste momento, igual a 1, então

$$Y = 2 * X + 5 ^ 3 = 2 * 1 + 5 ^ 3 = 2 + 125 = 127$$

este resultado, será transferido (atribuído) para a memória, na posição com nome "Y".

$$M \leftarrow \text{verdadeiro} \& (M \mid N)$$

$$P \leftarrow P + \exp(\text{raiz}(P) + \sin(Q + 0.707))$$

Contra-exemplos:

$X \leftarrow \varepsilon$       ( tipo inteiro recebendo caractere vazio )  
 $A \leftarrow X$       ( tipo caractere recebendo inteiro )  
 $A \leftarrow A$       ( variável recebendo a si mesma, sem sentido )  
  
 $M \leftarrow V$       ( "V" não é definido como verdadeiro )  
 $A \leftarrow a$       ( valor do tipo caractere não expresso entre aspas )

Conversões entre tipos

Para permitir a conversão de valores entre os diversos tipos básicos, pode-se usar o próprio nome do tipo como função de conversão.

Exemplos:

inteiro	X, Y
real	P, Q
caractere	A
caracteres	S
lógico	M, N

$X \leftarrow \text{inteiro}("0")$   
 $X \leftarrow \text{inteiro}(M)$   
 $Y \leftarrow \text{inteiro}(P)$   
 $Y \leftarrow \text{inteiro}(P+2*Q)$

$P \leftarrow \text{real}(Y)$   
 $Q \leftarrow \text{real}(\text{lógico}(X) \text{ ou } M)$

$A \leftarrow \text{caractere}(\text{inteiro}(P)+X)$   
 $S \leftarrow \text{caractere}(\text{inteiro}(1.0))$

$M \leftarrow \text{lógico}(0)$   
 $N \leftarrow \text{lógico}(\text{inteiro}(P))$   
 $M \leftarrow \text{lógico}(\text{inteiro}(\text{caractere}(0)))$

Observações:

- A conversão para caractere é feita segundo algum código (por exemplo, o ASCII), e é feita apenas se existir um símbolo de código correspondente, ocorrendo erro, caso contrário.
- A conversão entre tipos não é automática, entretanto para não complicar a notação de expressões, as situações abaixo deverão ser consideradas como existentes, apesar de não precisarem ser explicitadas:
  - a.) conversão de inteiro para real em expressão real;
  - b.) conversão de caracteres para caractere, em caso unitário.

### Definição de valores iniciais

O comando de transferência serve também para estabelecer valores iniciais para constantes, ou variáveis, em a sua definição. Uma expressão aritmética ou lógica, avaliada neste nível, deverá ter valores definidos.

Exemplo:

```

real      X ← 0.0
inteiro   I ← 1,
          J ← 1
caracteres NOME ← ε
lógico    PRIMEIRO ← falso

real      Y ← 2 * X
inteiro   K ← I + J

```

### Transferência múltipla

É possível fazer múltiplas transferências, desde que estejam agrupadas.

Exemplo:

$$(X, Y, Z) \leftarrow (0.0, 1.0, 2.0)$$

equivalente a uma função sobrejetora

```

X ← 0.0
Y ← 1.0
Z ← 2.0

```

$$(NOME, ENDEREÇO) \leftarrow (" ", " ")$$

equivalente a uma função sobrejetora

```

NOME      ← ε
ENDEREÇO  ← β
(X, Y, Z) ← 0.0

```

equivalente a uma função injetora

```

X ← 0.0
Y ← 0.0
Z ← 0.0

```

ou

$$X \leftarrow 0.0; Y \leftarrow 0.0; Z \leftarrow 0.0$$

ou

$$(X, Y, Z) \leftarrow (0.0, 0.0, 0.0)$$

## Transferência de valor entre dispositivos e memória

A transferência de valores entre os dispositivos de entrada e saída e as variáveis poderá ser indicada da mesma forma que uma transferência simples, ou múltipla, utilizando-se dispositivos predefinidos, como variáveis genéricas, sem definição de tipo, cuja atribuição implica em uma conversão automática para o tipo da variável que receber o valor:

Entrada:

teclado, cartão, fita, disco etc.

Saída:

tela, impressora, cartão, fita, disco etc.

- Entrada de datos

Forma geral:

```

<variável> ← <dispositivo>
ou
(<variável 1>, ... ) ← <dispositivo 1>
ou
(<variável 1> , ... ) ← (<dispositivo 1> , ... )

```

Exemplos:

X	← teclado	( X recebe um valor do teclado )
(X,Y)	← teclado	( X e Y recebem valores do teclado )
(X,Y)	← fita	( X e Y recebem valores de uma fita )
(X,Y)	← (disco, teclado)	( X recebe um valor de disco e Y recebe um valor do teclado )

Observações:

- Supõe-se, que nos dispositivos, os dados estarão separados, pelos menos, por um espaço em branco.
- Para atribuir um mesmo valor lido a várias variáveis proceder como indicado abaixo:

```
X ← teclado
(Y, Z) ← X
```

Contra-exemplos:

leia X ( comando não definido )

$Y \leftarrow X \leftarrow \text{teclado}$  ( múltiplo uso de transferência )

## - Saída de resultados

Forma geral:

$$\begin{aligned} &<\text{dispositivo}> \leftarrow <\text{variável}> \\ &\text{ou} \\ &<\text{dispositivo}> \leftarrow (<\text{variável } 1> , \dots ) \\ &\text{ou} \\ &<\text{dispositivo}> \leftarrow \text{"mensagem"} \\ &\text{ou} \\ &<\text{dispositivo}> \leftarrow (<\text{mensagem } 1> , <\text{variável } 1> , \dots ) \end{aligned}$$

Observação:

Supõe-se que os vários dados não estejam separados, sendo impressos em uma mesma linha até o seu limite, continuando na linha seguinte, se preciso.

Exemplos:

tela	$\leftarrow$ X	(envia o valor em X para a tela )
tela	$\leftarrow$ "Bom dia!"	(envia a mensagem para a tela )
disco	$\leftarrow$ (X,Y)	(envia valores em X e Y para o disco)
impressora	$\leftarrow$ X	(envia o valor em X para a impressora)
impressora	$\leftarrow$ ("X = ", X)	(envia tudo para a impressora)

Contra-exemplos:

imprima X		(comando não definido)
(X,Y,Z)	$\leftarrow$ (fita,disco)	(falta dispositivo)
(X,Y)	$\leftarrow$ (fita,disco,teclado)	(mais de uma transferência)
tela	$\leftarrow$ Bom dia!	(mensagem sem aspas)
disco	$\leftarrow$ "X = ", X	(faltam os parênteses)

- Recursos especiais de edição:

- Para a exibição de valores:

- Notação posicionada à direita de um campo:

<valor inteiro> : <tamanho do campo>  
 <valor real > : <tamanho do campo>:<parte fracionária>  
 <valor literal> : <tamanho do campo>  
 <valor lógico > : <tamanho do campo>

Exemplo:

tela ← 5:10      imprimirá

									5
--	--	--	--	--	--	--	--	--	---

tela ← 1.5:10:4    imprimirá

				1	.	5	0	0	0
--	--	--	--	---	---	---	---	---	---

tela ← "@":10      imprimirá

									@
--	--	--	--	--	--	--	--	--	---

tela ← falso:10    imprimirá

					F	A	L	S	O
--	--	--	--	--	---	---	---	---	---

Observações:

- Se o tamanho especificado estiver em conflito com o valor a ser exibido, esse tamanho não será respeitado.
- A notação científica será a forma normal de exibição para variáveis reais, a menos que definido o contrário.

- Para a manipulação de tela:

- Posicionamento em uma linha e coluna:

tela (linha, coluna) ← "mensagem"

Exemplo:

tela (01,01) ← "Começa aqui"

- Limpeza de tela (\lt):

tela ← \lt

- Limpeza de linha (\ll):

tela ← \ll

Exemplo:

tela (01,01) ← "Começa aqui"

tela (07,01) ← \ll

- Mudança de linha (\ml):

Exemplo:

tela ← ("mensagem 1", \ml, "mensagem 2")

imprimirá:

mensagem1  
mensagem2

- Para a manipulação de impressora:

- Tabulação:

impressora (coluna) ← "mensagem"

- Mudança de linha (\ml):

Exemplo:

impressora ← \ml

- Mudança de página (\mp):

Exemplo:

impressora ← \mp



## Estruturas de controle

Há três estruturas básicas de controle: a sequência simples, a estrutura condicional e a estrutura repetitiva. Cada estrutura, exceto a sequência simples, tem uma notação correspondente, sintaticamente fechada, ou seja, é obrigatório o uso de delimitadores específicos para seu início e fim, independente do número de comandos.

### Sequência simples

Forma geral:

$$\begin{array}{c} \langle \text{comando 1} \rangle \\ \dots \\ \langle \text{comando N} \rangle \\ \\ \text{ou} \\ \langle \text{comando 1} \rangle ; \dots ; \langle \text{comando N} \rangle \end{array}$$

Execução:

$$\begin{array}{ll} E_0 & (p_0) \\ \downarrow & a_1 \\ E_1 & (p_1) \\ \downarrow & a_0 = a_1 \dots a_n \\ \dots & \\ \downarrow & a_n \\ E_n & (p_n) \end{array}$$

Se existir uma única ação ( $a_0$ ), capaz de levar o sistema do estado inicial ( $E_0$ ), para o qual é definida uma certa propriedade, ou proposição, ( $p_0$ ), até o estado final ( $E_n$ ), também para o qual é definida uma outra propriedade ( $p_n$ ), pode-se dizer que:

$$(p_0) \ a_0 \ (p_n)$$

ou seja, se  $p_0$  for verdadeira, e se  $a_0$  for executada,  $p_n$  será verdadeira.

Se a ação unitária for composta por um conjunto de outras ações, tal que

$$(p_0) \ a_1 \ (p_1) \ \dots \ a_n \ (p_n)$$

então pode-se dizer que

$$(p_0) \ \langle \text{comando 1} \rangle ; \dots ; \langle \text{comando N} \rangle \ (p_n).$$

Os comandos (definidos pelas ações  $a_i$ 's) serão executados na ordem em que aparecerem no texto, um em cada linha; ou separados por ponto-e-vírgula (;), se na mesma linha. A um conjunto de comandos, associados, ou não, a uma estrutura de controle, chamaremos de **bloco de comandos**, ou simplesmente, **bloco**.

Exemplo:

```

início
! algoritmo para calcular valores !
! definição de dados !
!#1!   real   X, Y
!#2!   inteiro N ← 0
! cálculos !
!#3!   X ← 0.0 ; Y ← X + 1.0
!#4!   N ← N + 1
fim.

```

Execução:

Variável:	X	Y	N
Linha:			
#1	--	--	
#2			0
#3	0.0	1.0	
#4			1

Contra-exemplos:

```

início
! algoritmo para calcular valores !
! definição de dados !
real   X,Y ← 0.0           (X não é inicializado)
inteiro M ← 0
! cálculos                 (falta uma exclamação)
Y ← X + 1.0                (X não tem valor definido)
N ← N + 1                  (variável não definida)
fim.

```

```

início
! algoritmo para calcular valores !
! definição de dados !
real   (X,Y ← 0.0          (falta parênteses)
inteiro N = 0              (atribuição indefinida)
cálculos !                (falta uma exclamação)
Y ← X + 1.0 N ← N + 1      (falta ; entre comandos)
fim.

```

```

                                     (falta o início)
! algoritmo para calcular valores !
! definição de dados !
real   (X,Y ← 0.0          (falta parênteses)
inteiro N → 0              (atribuição indefinida)
cálculos                  (faltam exclamações)
Y ← X + 1.0: N ← N + 1     (falta ; entre comandos)
fim.

```

Outro exemplo:

```

início
! algoritmo para calcular valores !
! definição de dados !
!#1!   real X ← 0.0,
!#2!   Y ← 0.0
! leitura !
!#3!   X ← teclado
! cálculo !
!#4!   X ← X + 1
!#5!   Y ← X * X
! impressão !
!#6!   tela ← " 2 "
!#7!   tela ← "(X+1) = ",Y)
fim.

```

Execução:

Dado:

X = 0.0

Variável:	X	Y	teclado	tela
Linha:				
#1	0.0			
#2		0.0		
#3	0.0		0.0	
#4	1.0			
#5		1.0		
#6				2
#7				(X+1) = 1.0

Dado:

X = 1.0

Variável:	X	Y	teclado	tela
Linha:				
#1	0.0			
#2		0.0		
#3	1.0		1.0	
#4	2.0			
#5		4.0		
#6				2
#7				(X+1) = 4.0

## Exercícios (mostrar a execução)

1. Fazer um algoritmo para:
  - mostrar seu nome na tela.
2. Fazer um algoritmo para:
  - ler um nome qualquer do teclado;
  - mostrar esse nome na tela.
3. Fazer um algoritmo para:
  - ler um valor inteiro (X) do teclado;
  - calcular e mostrar na tela o dobro de X.
4. Fazer um algoritmo para:
  - ler dois valores reais (X e Y) do teclado;
  - calcular e mostrar na tela:
    - a soma destes valores
    - o produto deles
    - o quociente entre eles
    - o valor absoluto da diferença entre eles (use  $\text{ABS}(X-Y)$ ).
5. Fazer um algoritmo para:
  - ler o diâmetro de uma esfera do teclado;
  - calcular e mostrar:
    - a área da superfície esférica;
    - o volume da esfera.
6. Fazer um algoritmo para:
  - ler dois valores inteiros (X e Y) do teclado;
  - mostrar na tela o resultado da operação lógica que verifica se a operação diferença, entre eles, é igual a zero.
7. Mostrar a execução do algoritmo abaixo:

```

      início
      ! algoritmo para calcular valores !
      ! definição de dados !
!#1!   real  E,
!#2!       m,
!#3!       c ← 3.0E08 ! m/s !
      ! leitura !
!#4!   tela ← "Valor da massa (m) = "
!#5!   m ← teclado
      ! cálculo !
!#6!   E ← m * c ^ 2
      ! impressão !
!#7!   tela ← ("Energia = ", E)
      fim.

```

Dados:

```

m = 0.1 kg
m = 1.0E-8 kg
m = 80 kg

```

## Alternativas

### Alternativa dupla

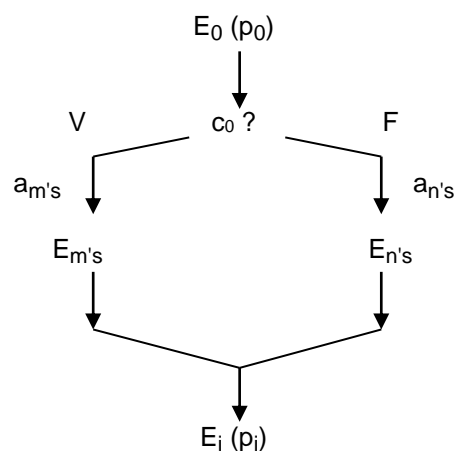
Forma geral :

```
se <condição> então
| <bloco de comandos 1>
senão
| <bloco de comandos 2>
fim se
```

onde :

<condição> é uma expressão lógica qualquer  
<bloco de comandos> é uma sequência de comandos.

Execução :



Se existir uma certa propriedade, ou proposição,  $(p_0)$ , e uma certa condição  $(c_0)$ , além de uma outra propriedade  $(p_n)$ , pode-se dizer que:

$$((p_0 \bullet c_0) a_m's (p_n)) \bullet ((p_0 \bullet \neg c_0) a_n's (p_n))$$

ou então

$(p_0)$  se  $c_0$  então  $a_m's$  senão  $a_n's$  fim se  $(p_n)$ .

Primeiro, avalia-se a condição. Se for verdadeira, apenas o primeiro bloco de comandos (definido pelas ações  $a_m's$ ) será executado; se for falsa, somente o segundo bloco de comandos (definido pelas ações  $a_n's$ ) será executado. O próximo comando a ser executado será aquele após o "fim se".

Exemplo :

```

início
! algoritmo para ler dois valores e determinar o maior !
! definição de dados !
!#1!   real A, B
! leitura !
!#2!   ( A, B ) ← teclado
! verificação, supondo A diferente de B !
!#3!   se ( A > B ) então
!#4!   | tela ← "A é o maior"
        senão ! A > B !
!#5!   | tela ← "B é o maior"
        fim se ! A > B !
fim.
```

Execução :

Dados : A = 0 e B = -5

Variável :	A	B	teclado	tela
Linha :				
#1	--	--		
#2	0.0	-5.0	0.0 -5.0	
#3	A > B ? V			
#4	A é o maior			

Dados : A = 0 e B = 5

Variável :	A	B	teclado	tela
Linha :				
#1	--	--		
#2	0.0	5.0	0.0 5.0	
#3	A > B ? F			
#4	B é o maior			

Contra-exemplos :

```

início
! algoritmo para ler dois valores e determinar o maior !
! definição de dados !
  real A                                (falta a definição de B)
! leitura !
  A, B ← teclado                        (faltam os parênteses)
! verificação, supondo A diferente de B !
  se A > B                              (falta o "então")
  | tela ← "A é o maior"
  senão A > B                            (o teste não é repetido)
  | tela ← "B é o maior"
  fim se ! A > B !
fim.

```

```

início
! algoritmo para ler dois valores e determinar o maior !
! definição de dados !
  real A, B
! leitura !
  (A, B ← teclado                        (falta um parênteses)
! verificação, supondo A diferente de B !
  se A > B então
  | tela ← "A é o maior"
  se não ! A > B !                        (diferente de "senão")
  | tela ← "B é o maior"
  fim.                                  (falta o "fim se")

```

## Exercícios (mostrar a execução)

1. Fazer um algoritmo para :
  - ler um nome do teclado e verificar se é igual ao seu nome;
  - mostrar conforme o caso :  
"NOME CORRETO" ou "NOME INCORRETO".
2. Refazer o anterior para usar um único comando de impressão.
3. Fazer um algoritmo para :
  - ler dois números inteiros do teclado;
  - calcular e mostrar o quociente do primeiro pelo segundo, se este for diferente de zero, senão mostrar a mensagem :  
"ERRO - DIVISÃO POR ZERO".
4. Fazer um algoritmo para :
  - ler dois números reais do teclado (A e B, diferentes);
  - verificar e mostrar qual o maior deles.
5. Refazer o anterior prevendo a possibilidade de serem iguais e mostrar a mensagem :  
" A = B ", neste caso.
6. Mostrar a execução do algoritmo abaixo :

```

início
! algoritmo para verificar lados de um triângulo !
! definição de dados !
!#1!   real A, B, C
! leitura dos valores !
!#2!   (A, B, C) ← teclado
! teste de existência do triângulo !
!#3!   se ((A < B+C) & (B < A+C) & (C < A+B)) então
|   ! teste de tipo de triângulo !
!#4!   |   se ((A=B) & (B=C)) então
!#5!   |   | tela ← "TRIÂNGULO EQUILÁTERO"
|   |   senão ! A≠B | B≠C !
!#6!   |   | se ((A=B) | (A=C) | (B=C)) então
!#7!   |   | tela ← "TRIÂNGULO ISÓSCELES"
|   |   senão ! A≠B & B≠C !
!#8!   |   | tela ← "TRIÂNGULO ESCALENO"
|   |   fim se ! algum lado igual !
|   fim se ! dois lados iguais !
|   senão ! não existe triângulo !
!#9!   |   tela ← "NÃO É TRIÂNGULO"
fim se ! for triângulo !
fim.

```

Dados :

```

A = 2.0, B = 2.0 e C = 1.0
A = 2.0, B = 3.0 e C = 1.0
A = 1.0, B = 1.0 e C = 1.0
A = 2.0, B = 2.5 e C = 1.0

```

### Alternativa simples

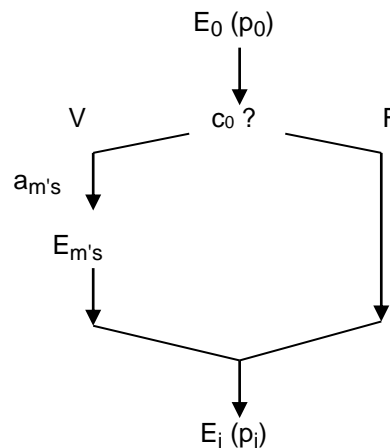
Forma geral :

```
se <condição> então
| <bloco de comandos>
fim se
```

onde :

<condição>                    é uma expressão lógica qualquer  
<bloco de comandos>        é uma sequência de comandos.

Execução :



Se existir uma certa propriedade, ou proposição, ( $p_0$ ), e uma certa condição ( $c_0$ ), além de uma outra propriedade ( $p_n$ ), pode-se dizer que :

$$((p_0 \bullet c_0) a_m's (p_n)) \bullet ((p_0 \bullet \neg c_0) \Rightarrow (\text{implica em}) (p_n))$$

ou então,

( $p_0$ ) se  $c_0$  então  $a_m's$  fim se ( $p_n$ ).

A alternativa simples pode ser considerada como um caso particular da alternativa dupla. Se a condição for verdadeira, o bloco de comandos (definido pelas ações  $a_m's$ ) será executado; se for falsa, nenhum comando do bloco será executado. O próximo comando a ser executado será aquele após o "fim se".

Exemplo :

```

início
! algoritmo para ler dois valores e
verificar se o primeiro é maior que o segundo !
! definição de dados !
!#1!   real A, B
! leitura !
!#2!   . ( A, B ) ← teclado
! verificação de A em relação a B !
!#3!   se ( A > B ) então
!#4!   | tela ← "O primeiro é o maior"
      fim se ! A > B !
fim.
```





```
início
! algoritmo para ler dois valores e
  verificar se o primeiro é maior que o segundo !
! definição de dados !
  real A, B
! leitura !
  (A, B) ← teclado
! verificação de A em relação a B !
  se  $A \geq B$  então
    | se  $A > B$  então
    | | tela ← "O primeiro é o maior"
    | fim se !  $A > B$  !
  senão !  $A \geq B$  !
    | tela ← "O segundo é o maior"
  senão
    | tela ← "Não há maior"
  fim se !  $A > B$  !
fim.
```

(sobra um "senão")

## Exercícios (mostrar a execução)

1. Fazer um algoritmo para :
  - ler um nome do teclado;
  - verificar se é igual ao seu nome;
  - se for mostrá-lo.
2. Fazer um algoritmo para :
  - ler dois números inteiros do teclado;
  - calcular e mostrar o quociente do primeiro pelo segundo se este for diferente de zero.
3. Fazer um algoritmo para :
  - ler três números reais do teclado (A, B e C);
  - verificar se A é maior que a soma do valor absoluto dos outros.
4. Fazer um algoritmo para :
  - ler dois números reais do teclado (A e B);
  - verificar se satisfazem à seguinte condição :  

$$ABS(A + 0.5) > LN(B^{3.0/4.}) / 2$$
5. Fazer um algoritmo para :
  - ler dois números reais do teclado (A e B, diferentes);
  - verificar se ambos são maiores que zero;
  - se forem, mostrar uma mensagem :  

$$"S\tilde{A}O\ MAIORES\ QUE\ ZERO"$$
6. Mostrar a execução do algoritmo abaixo :

```

início
! algoritmo para ordenar dois valores !
! definição de dados !
!#1!   real   A, B, C
!#2!   lógico ORDENAR
! leitura dos valores !
!#3!   (A, B) ← teclado
! decide se é necessário ordenar !
!#4!   ORDENAR ← (A > B)
!#5!   se ( ORDENAR ) então
!#6!   | C ← A
!#7!   | A ← B
!#8!   | B ← C
      fim se ! ORDENAR !
! impressão !
!#9!   tela ← ("Na ordem crescente : ", A, B)
fim.

```

Dados :

A = 2.0, B = 1.0  
 A = 1.0, B = 2.0  
 A = 2.0, B = 2.0

### Alternativa múltipla

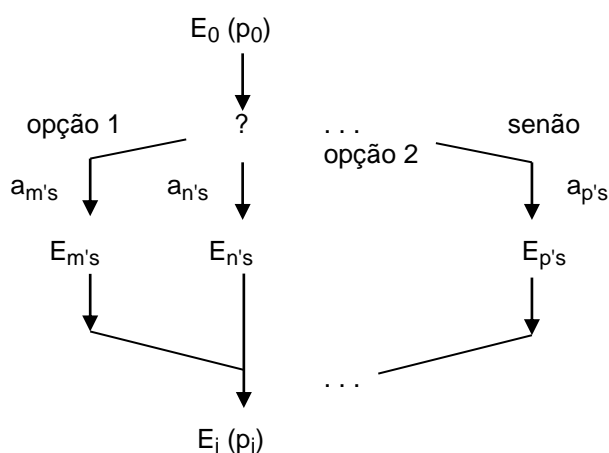
Forma geral :

```

escolher <valor>
| <opção 1> : <bloco de comandos 1>
| ...
| <opção N> : <bloco de comandos N>
senão
| <bloco de comandos M>
fim escolher
  
```

Execução :

Execução :



Cada opção será avaliada na ordem em que for colocada, verificando-se a igualdade em relação ao valor fornecido. Se houver alguma opção satisfeita, o bloco de comandos associado será executado, e o controle passará ao próximo comando após o "fim escolher". Caso não existir opção satisfeita, o último bloco será executado. Este último bloco é opcional.

Exemplos :

```

início
! algoritmo para informar sobre um valor lido !
! definição de dados !
  inteiro X
! leitura !
  X ← teclado
! escolha segundo o valor lido !
  escolher ( X )
  | -1 : tela ← "Valor negativo"
  | 0 : tela ← "Valor nulo"
  | 1 : tela ← "Valor positivo"
senão
  | tela ← "Outro valor"
fim escolher
fim.
  
```

O algoritmo acima é equivalente a

```

início
! algoritmo para informar sobre um valor lido !
! definição de dados !
  inteiro X
! leitura !
  X ← teclado
! escolha segundo o valor lido !
  se ( X = -1 ) então tela ← "Valor negativo"
  senão ! X = -1 !
    | se ( X = 0 ) então tela ← "Valor nulo"
    | senão ! X = 0 !
      | se X = 1 então tela ← "Valor positivo"
      | senão tela ← "Outro valor"
      | fim se ! X = 1 !
    | fim se ! X = 0 !
  fim se ! X = -1 !
fim.

```

Outro exemplo :

```

início
! algoritmo para informar sobre valores lidos !
! definição de dados !
  inteiro N
! leitura !
  N ← teclado
! escolha segundo o valor lido !
  escolher N
  | 1 , 2      : N ← N + 1          ! 1 ou 2      !
  | 3 : 6      : N ← N - 1          ! 3, 4, 5, 6 !
  |           tela ← "Passou de dois"
  senão
    | se ( N ≤ 0 ) então
    | | tela ← "Menor ou igual a zero"
    | senão ! N ≤ 0 !
    | | tela ← "Maior ou igual a sete"
    | fim se ! N ≤ 0 !
  fim escolher
fim.

```

O algoritmo anterior é equivalente a

```

início
! algoritmo para informar sobre valores lidos !
! definição de dados !
  inteiro N
! leitura !
  N ← teclado
! escolha segundo o valor lido !
  se ((N = 1) | (N = 2)) então
    | N ← N + 1
  senão ! N = 1 | N = 2 !
    | se ((N ≥ 3) & (N ≤ 6)) então
      | | N ← N - 1
      | | tela ← "Passou de dois"
    | senão ! N ≥ 3 & N ≤ 6 !
      | | se (N ≤ 0) então
        | | | tela ← "Menor ou igual a zero"
        | | | senão ! N ≤ 0 !
        | | | tela ← "Maior ou igual a sete"
        | | fim se ! N ≤ 0 !
      | fim se ! N ≥ 3 | N ≤ 6 !
    fim se ! N = 1 | N = 2 !
fim.

```

Os comparadores e conectivos lógicos usados adequadamente, substituem com vantagens, testes equivalentes :

```

início
! algoritmo para informar sobre valores lidos !
! definição de dados !
  inteiro N
! leitura !
  N ← teclado
! escolha segundo o valor lido !
  se (N = 1) então
    | N ← N + 1
  fim se ! N = 1 !
  se (N = 2) então
    | N ← N + 1
  fim se ! N = 2 !
  se (N ≥ 3) então
    | se (N ≤ 6) então
      | | N ← N - 1
      | | tela ← "Passou de dois"
    | fim se ! N ≤ 6 !
  fim se ! N ≥ 3 !
  se (N ≤ 0) então
    | tela ← "Menor ou igual a zero"
  fim se ! N ≤ 0 !
  se (N ≥ 7) então
    | tela ← "Maior ou igual a sete"
  fim se ! N ≥ 7 !
fim.

```

Observação :

O conectivo lógico OU ( | ) é equivalente a dois testes seguidos :

```
se ( N = 1 ) então
|  N ← N + 1
fim se
se ( N = 2 ) então
|  N ← N + 1
fim se
```

O conectivo lógico E ( & ) é equivalente a dois testes encaixados :

```
se ( N ≥ 3 ) então
|  se ( N ≤ 6 ) então
|  |  N ← N - 1
|  |  tela ← "Passou de dois"
|  fim se
fim se
```

Mais um exemplo :

```
início
! algoritmo para informar sobre frutas !
! definição de dados !
  caracteres FRUTA
! leitura !
  FRUTA ← teclado
! escolha segundo o valor lido !
  escolher ( FRUTA )
  |  "BANANA":  tela ← "Comprar 01 dúzia"
  |  "MAÇÃ"   :  tela ← "Comprar argentinas"
  |             tela ← "Comprar verdes"
  |             tela ← "Comprar nacionais"
  |  "LARANJA": tela ← "Comprar 03 KG"
  fim escolher
fim.
```

O algoritmo anterior é equivalente a

```

início
! algoritmo para informar sobre frutas !
! definição de dados !
  caracteres FRUTA
! leitura !
  FRUTA ← teclado
! escolha segundo o valor lido !
  se ( FRUTA = "BANANA" ) então
    | tela ← "Comprar 01 dúzia"
  senão ! FRUTA = "BANANA" !
    | se ( FRUTA = "MAÇÃ" ) então
      | | tela ← "Comprar argentinas"
      | | tela ← "Comprar verdes"
      | | tela ← "Comprar nacionais"
    senão ! FRUTA = "MAÇÃ" !
      | | se ( FRUTA = "LARANJA" ) então
        | | | tela ← "Comprar 03 KG"
        | | fim se ! FRUTA = "LARANJA" !
        | fim se ! FRUTA = "MAÇÃ" !
      fim se ! FRUTA = "BANANA" !
fim.

```

Contra-exemplos :

```
início  
! algoritmo para informar sobre um valor lido !  
! definição de dados !  
inteiro X  
! leitura !  
X ← teclado  
! escolha segundo o valor lido !  
escolher (falta a variável)  
| -1 : tela ← "Valor negativo"  
| 0 : tela ← "Valor nulo"  
| 1 : tela ← "Valor positivo"  
senão  
| (falta comando)  
fim escolher  
fim.
```



```

início
! algoritmo para informar sobre valores lidos !
! definição de dados !
  inteiro N, P
! leitura !
  (N, P) ← teclado
! escolha segundo o valor lido !
  escolher N e P                (duas variáveis)
  | 1 e 2      : N ← N + 1      ("e" não é definido)
  | 3 até 5    : N ← N - 1      ("até" não é definido)
  |            tela ← "Passou de dois"
senão
  | se  $N \leq 0$  então
  | | tela ← "Menor ou igual a zero"
  | senão !  $N \leq 0$  !
  | | tela ← "Maior ou igual a cinco"
  | fim se !  $N \leq 0$  !
  fim escolher
fim.

```

```

início
! algoritmo para informar sobre frutas !
! definição de dados !
  inteiro FRUTA
! leitura !
  FRUTA ← teclado
! escolha segundo o valor lido !
  escolher FRUTA                (opções de tipo diferente)
  | "BANANA": tela ← "Comprar 01 dúzia"
  | "MAÇÃ"  : tela ← "Comprar argentinas"
  |          tela ← "Comprar verdes"
  |          tela ← "Comprar nacionais"
  | "LARANJA": tela ← "Comprar 03 KG"
  fim escolher
fim.

```

## Exercícios (mostrar a execução)

## 1. Fazer um algoritmo para :

- ler um nome do teclado;
- mostrar o telefone correspondente, de acordo com a lista :

Artur	221
Bernardo	211
Eustáquio	311
Luiz	312
Mário	332

## 2. Refazer o algoritmo anterior, incluindo para mostrar a mensagem :

NÃO CONSTA DA LISTA", se o nome lido não estiver na lista.

## 3. Fazer um algoritmo para :

- ler um símbolo do teclado;
- mostrar as seguintes mensagens, segundo o caso :
  - "SINAL DE MENOR"
  - "SINAL DE MAIOR"
  - "SINAL DE IGUAL"
  - "OUTRO SINAL"

## 4. Fazer um algoritmo para :

- ler um símbolo do teclado;
- mostrar as seguintes mensagens, segundo o caso :
  - "LETRA MAIÚSCULA"
  - "LETRA MINÚSCULA"
  - "ALGARISMOS"
  - "OUTRO SÍMBOLO"

## 5. Fazer um algoritmo para :

- ler um valor inteiro (LADOS de um polígono);
- ler um valor real (LADO) , tamanho de cada lado;
- dependendo do valor de lados (3, 4, 5 ou 6) :
  - se LADOS = 3, calcular e mostrar o perímetro;
  - se LADOS = 4, calcular e mostrar a área;
  - se LADOS = 5, calcular e mostrar o perímetro e a área;
  - se LADOS = 6, calcular e mostrar a área de cada triângulo interno.

## 6. Fazer um algoritmo para :

- calcular e mostrar o custo de transporte de um frete;
- ler o valor de custo fixo e o destino do frete;
- para cada destino existe um custo variável :

DESTINO	CUSTO
Sabará	0.5 * FIXO
Monlevade	0.3 * FIXO
Contagem	0.1 * FIXO
Mariana	0.2 * FIXO
Ipatinga	0.4 * FIXO
Ouro Branco	0.4 * FIXO

## Repetição simples

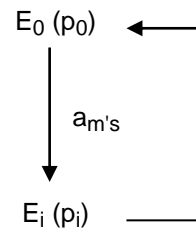
Forma geral :

```

repetir
| <bloco de comandos>
fim repetir

```

Execução :



O bloco de comandos definidos pelas ações  $a_m$ 's são repetidos infinitas vezes, um de cada vez.

Pode haver várias formas condicionais desta estrutura, como se descreve a seguir.

- Forma com interrupção :

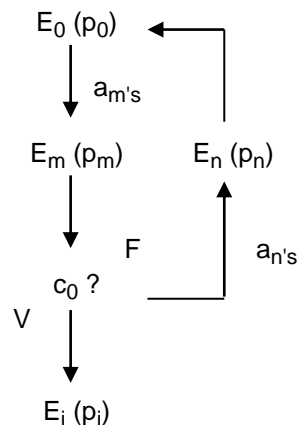
Forma geral :

```

repetir
| <bloco de comandos 1>
| se <condição> então parar fim se
| <bloco de comandos 2>
fim repetir

```

Execução :



Na forma com interrupção, executa-se o primeiro bloco de comandos (ações  $a_m$ 's). Existe(m) um (ou mais) teste(s) de condição que serão avaliados a cada repetição; se a condição for verdadeira, abandona-se imediatamente a execução do bloco de comandos restantes (ações  $a_n$ 's). O próximo comando a ser executado será aquele seguinte ao "fim repetir".

Esta forma trata as condições como medidas de exceção, e deve ser usada com bastante critério, pois a execução de não cobertos pela avaliação de uma condição pode levar à execução incorreta. Sugere-se o uso de formas padronizadas e seguras, deixando esta expressão livre para aplicações particulares.

- Forma com teste no início :

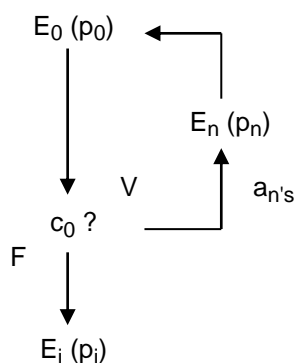
Forma geral :

```

repetir
| se <condição> então
| | <bloco de comandos>
| senão
| | parar
| fim se
fim repetir

```

Execução :



Se existir uma certa propriedade, ou proposição, ( $p_0$ ), e um certa condição ( $c_0$ ) pode-se dizer que :

$$(p_0 \bullet c_0) \text{ a}_n\text{'s } (p_0)$$

ou seja,

enquanto ( $p_0 \bullet c_0$ ) repetir  $a_n\text{'s}$  fim repetir .

Na forma com o teste no início, a condição será avaliada antes de começar a execução do bloco de comandos. Se a condição for verdadeira, o bloco de comandos será executado uma vez; após esta execução, a condição será novamente avaliada; se for falsa, o bloco de comandos não será mais executado. Quando se encerrar a execução da estrutura, o próximo comando a ser executado será aquele na linha seguinte ao "fim repetir".

Recomenda-se usar esta forma por ser aquela mais segura, pois evita a execução do bloco de comandos, se a condição não for satisfeita, inclusive para a primeira vez. Para efeito de simplificação e rápida identificação com estruturas implementadas em linguagens de programação, pode-se usar uma notação simplificada, com teste automático :

```

repetir enquanto <condição>
| <bloco de comandos>
fim repetir

```

Exemplo :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X ← 1
  repetir
  | se (  $X \leq 10$  ) então
  | | tela ← X
  | | X ← X + 1
  | senão
  | | parar
  | fim se
  fim repetir
fim.

```

com teste automático :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
!#1!  inteiro X
! repetição !
!#2!  X ← 1
!#3!  repetir enquanto (  $X \leq 10$  )
!#4!  | tela ← X
!#5!  | X ← X + 1
      fim repetir ! enquanto  $X \leq 10$  !
fim.

```

Execução :

Variável :	X	tela
Linha :		
#1	--	
#2	1	
#3	$X \leq 10 ? V$	
#4		1
#5	2	
#3	$X \leq 10 ? V$	
#4		2
#5	3	
#3	$X \leq 10 ? V$	
#4		3
#5	4	
...		
#3	$X \leq 10 ? V$	
#4		9
#5	10	
#3	$X \leq 10 ? V$	
#4		10
#5	11	
#3	$X \leq 10 ? F$	

Contra-exemplos :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  repetir enquanto  $X \leq 10$            (falta o valor inicial)
  |  tela  $\leftarrow$  X
  |  X  $\leftarrow$  X + 1
  fim repetir ! enquanto  $X \leq 10$  !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X  $\leftarrow$  1
  repetir enquanto  $X \leq 10$ 
  |  tela  $\leftarrow$  X           (falta o incremento)
  fim repetir ! enquanto  $X \leq 10$  !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X  $\leftarrow$  0
  repetir enquanto  $X \leq 10$            (repete 11 vezes)
  |  tela  $\leftarrow$  X
  |  X  $\leftarrow$  X + 1
  fim repetir ! enquanto  $X \leq 10$  !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X  $\leftarrow$  1
  repetir enquanto  $X < 10$            (repete 09 vezes)
  |  tela  $\leftarrow$  X
  |  X  $\leftarrow$  X + 1
  fim repetir ! enquanto  $X < 10$  !
fim.

```

## Exercícios (mostrar a execução)

1. Fazer um algoritmo para :
  - mostrar os dez primeiros números naturais maiores que 100.
2. Fazer um algoritmo para :
  - mostrar os dez números entre de 0.1 até 1.0.
3. Fazer um algoritmo para :
  - mostrar os números de 100 até 200 variando de 10 em 10.
4. Fazer um algoritmo para :
  - ler de uma fita um conjunto de informações;
  - cada informação contém um NOME e um SALÁRIO;
  - a última informação não entrará nos cálculos e tem o valor de SALÁRIO igual a zero;
  - ler e mostrar cada informação.
5. Fazer um algoritmo para :
  - ler de uma fita um conjunto de informações;
  - cada informação contém um NOME e um SALÁRIO;
  - a última informação não entrará nos cálculos e tem o valor de SALÁRIO igual a zero;
  - para cada informação fazer :
    - calcular e mostrar um novo SALÁRIO com aumento de 20%;
    - determinar o nome com o maior valor de SALÁRIO.
    - calcular e mostrar quantos NOMES foram lidos.
6. Mostrar a execução do algoritmo abaixo :

```

      início
      ! algoritmo para calcular alguns divisores !
      ! definição de dados !
!#1!   inteiro D, N
      ! leitura !
!#2!   N ← teclado
      ! repetição !
!#3!   D ← 1
!#4!   repetir enquanto ( D ≤ N div 2 )
!#5!   | se ( N mod D = 0 ) então
!#6!   | | tela ← D
      | fim se
!#7!   | D ← D + 1
      fim repetir
fim.
```

Dados :

```

N = 0
N = 12
N = 11
```

- Forma com teste no início e variação :

Forma geral :

```

<controle> ← <início>
repetir
| se não <fim> então
| | <bloco de comandos>
| | <controle> ← <controle> + <variação>
| senão
| | parar
| fim se
fim repetir

```

com teste automático :

```

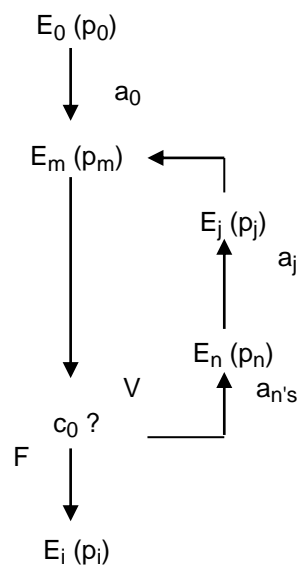
repetir para <controle> ← <início> : <fim> : <variação>
| <bloco de comandos>
fim repetir

```

onde :

<início>	é um comando de atribuição
<controle>	é uma variável
<fim>	é uma expressão
<variação>	também é uma expressão

Execução :





Esta estrutura com variação é equivalente a:

```

<início>
repetir enquanto <variável> ≤ <fim>
|  <bloco de comandos>
|  <variável> ← <variável> + <variação positiva>
fim repetir

```

ou

```

<início>
repetir enquanto <variável> ≥ <fim>
|  <bloco de comandos>
|  <variável> ← <variável> + <variação negativa>
fim repetir

```

Na forma com teste no início e variação, primeiro será executado o comando de atribuição (definido pela ação  $a_0$ ), estabelecendo o valor inicial da variável, chamada **variável de controle**. Logo após, será feita uma comparação com o valor apresentado na expressão <fim>, para verificar se objetivo já foi alcançado : se o valor atual da variável é maior que o valor da expressão <fim>, caso o valor da variação seja **positiva**; ou se o valor atual é menor que o valor da expressão <fim>, caso a variação seja **negativa**. Não tendo sido alcançado o objetivo, o bloco de comandos (definidos pelas ações  $a_{n's}$ ) será executado uma vez, a variável de controle sofrerá a ação definida por  $a_j$ , sendo modificada pelo valor da <variação> (também chamada de **passo**), e a comparação dos valores será repetida, mais uma vez. Quando o objetivo for alcançado, a execução do bloco de comandos da estrutura terminará, e o próximo comando a ser executado será aquele na linha seguinte ao "fim repetir".

O uso desta estrutura está limitado pelas condições abaixo :

- a variável de controle não pode ser modificada no corpo da estrutura;
- os valores de <variação> e de <fim> devem ser fixos, e não devem ser modificados no corpo da estrutura.

Observações :

Se estas condições não forem satisfeitas, pode ocorrer uma incoerência na execução da estrutura.

Se o valor da <variação> for igual a (1) poderá ser omitido.

Exemplo :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
inteiro X
! repetição !
X ← 1
repetir
| se ( X ≤ 10 ) então
| | tela ← X
| | X ← X + 1
| senão
| | parar
| fim se
fim repetir ! para X !
fim.

```

Na forma com teste e variação automáticos :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
!#1! inteiro X
! repetição !
!#2! repetir para ( X ← 1 : 10 : 1 )
!#3! | tela ← X
      fim repetir ! para X !
fim.

```

Execução :

Variável :	X	tela
Linha :		
#1	--	
#2	1	
#2	$X \leq 10 ? V$	
#3		1
#2	2	
#2	$X \leq 10 ? V$	
#3		2
#2	3	
#2	$X \leq 10 ? V$	
#3		3
#2	4	
...		
#2	$X \leq 10 ? V$	
#3		9
#2	10	
#2	$X \leq 10 ? V$	
#3		10
#2	11	
02	$X \leq 10 ? F$	

Contra-exemplos :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
inteiro X
! repetição !
  repetir para X ← X : 10 : 1      (valor inicial indefinido)
  | tela ← X
  fim repetir ! para X !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
inteiro X
! repetição !
  repetir para X ← 1 : X : 1      (repetição indefinida)
  | tela ← X
  fim repetir ! para X !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  repetir para X ← X : X : 1          (repetição indefinida)
  |  tela ← X
  fim repetir ! para X !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  repetir para 1 : 10 : 1          (falta variável)
  |  tela ← X
  fim repetir ! para X !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  repetir para X ← 1 até 10 passo 1  (estrutura incorreta)
  |  tela ← X
  fim repetir ! para X !
fim.

```

## Exercícios (mostrar a execução)

1. Fazer um algoritmo para :  
- mostrar os dez primeiros números naturais menores que 100.
2. Fazer um algoritmo para :  
- mostrar os dez primeiros números naturais de 100 até 110.
3. Fazer um algoritmo para :  
- mostrar de 0.2 até 2.0, variando de 0.2 em 0.2.
4. Fazer um algoritmo para :  
- mostrar os números de 100 até 200 variando de 20 em 20.
5. Fazer um algoritmo para :  
- mostrar os 100 primeiros números pares.
6. Mostrar a execução do algoritmo abaixo :

```

início
! algoritmo para calcular alguns divisores !
! definição de dados !
!#1!   inteiro D, N
! leitura !
!#2!   N ← teclado
! repetição !
!#3!   repetir para ( D ← 1 : (N div 2) )
!#4!   | se ( N mod D = 0 ) então
!#5!   | | tela ← D
        | fim se
        fim repetir
fim.

```

Dados :

```

N = 0
N = 12
N = 11

```

- Forma com teste no fim

Forma geral :

```

repetir
| <bloco de comandos>
| se <condição> então
| | parar
| fim se
fim repetir

```

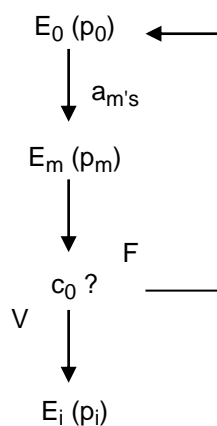
com teste automático :

```

repetir até <condição>
| <bloco de comandos>
fim repetir

```

Execução :



Se existir uma certa propriedade, ou proposição,  $(p_0)$ , e um certa condição  $(c_0)$  pode-se dizer que :

$$(p_0) \ a_m's \ (p_0 \bullet c_0)$$

ou seja,

$(p_0)$  repetir  $a_m's$  até  $c_0$  fim repetir .

Na forma com o teste no fim, o bloco de comandos (definido pelas ações  $a_m's$ ) será executado **pelo menos uma vez**, e a condição será avaliada depois. Se a condição for falsa, o bloco de comandos será executado mais uma vez; se for verdadeira, o objetivo foi alcançado e o bloco de comandos não será mais executado. Quando encerrar a execução da estrutura, o próximo comando a ser executado será aquele na linha seguinte ao "fim repetir".

Exemplo :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X ← 0
  repetir
    | X ← X + 1
    | tela ← X
    | se ( X = 10 ) então
    |   | parar
    |   fim se
  fim repetir ! até X = 10 !
fim.

```

com teste automático :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
!#1!  inteiro X
! repetição !
!#2!  X ← 0
      repetir até ( X = 10 )
!#3!  | X ← X + 1
!#4!  | tela ← X
!#5!  fim repetir ! até X = 10 !
fim.

```

Execução :

Variável :	X	tela
Linha :		
#1	--	
#2	0	
#3	1	
#4		1
#5	X = 10 ? F	
#3	2	
#4		2
#5	X = 10 ? F	
#3	3	
#4		3
...		
#5	X = 10 ? F	
#3	9	
#4		9
#5	X = 10 ? F	
#3	10	
#4		10
#5	X = 10 ? V	

Contra-exemplos :

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  repetir até X = 10
  |  X ← X + 1                      (variável indefinida)
  |  tela ← X
  fim repetir ! até X = 10 !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X ← 0
  repetir até X > 10
  |  X ← X + 1                      (repete 11 vezes)
  |  tela ← X
  fim repetir ! até X > 10 !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X ← 1
  repetir até X = 10
  |  X ← X + 1                      (repete 09 vezes)
  |  tela ← X
  fim repetir ! até X = 10 !
fim.

```

```

início
! algoritmo para contar de 1 até 10 !
! definição de dados !
  inteiro X
! repetição !
  X ← 1
  repetir até 10                    (falta variável)
  |  X ← X + 1
  |  tela ← X
  fim repetir ! até X = 10 !
fim.

```

## Exercícios (mostrar a execução)

1. Fazer um algoritmo para :
  - mostrar os dez primeiros números naturais maiores que 201.
2. Fazer um algoritmo para :
  - mostrar os dez primeiros números naturais entre 9 e 19.
3. Fazer um algoritmo para :
  - mostrar dez valores igualmente espaçados entre 0.3 e 3.0.
4. Fazer um algoritmo para :
  - ler de uma fita um conjunto de informações;
  - cada informação contém NOME e SALÁRIO;
  - a última informação não entrará nos cálculos e tem o valor de SALÁRIO igual a zero;
  - ler e mostrar cada informação.
5. Fazer um algoritmo para :
  - ler de uma fita um conjunto de informações;
  - cada informação contém NOME e SALÁRIO;
  - a última informação não entrará nos cálculos e tem o valor de SALÁRIO igual a zero;
  - para cada informação fazer :
    - calcular e mostrar um novo SALÁRIO com aumento de 20%;
    - determinar o nome com o maior valor de SALÁRIO.
    - calcular e mostrar quantos NOMES foram lidos.
6. Mostrar a execução do algoritmo abaixo :

```

início
! algoritmo para calcular alguns divisores !
! definição de dados !
!#1!   inteiro D, N
! leitura !
!#2!   N ← teclado
! repetição !
!#3!   D ← 1
!#4!   repetir até ( D > N div 2 )
!#5!   | se ( N mod D = 0 ) então
!#6!   | | tela ← D
!#6!   | fim se
!#7!   | D ← D + 1
!#7!   fim repetir
fim.

```

Dados :

```

N = 0
N = 12
N = 11

```