

# CONSTRUÇÃO DE ALGORITMOS

Texto Introdutório

Profs. Carlos Nehab e Manuel Martins

# ÍNDICE

O CONCEITO DE ALGORITMO	1
INTRODUÇÃO	1
ESTRUTURAS DE CONTROLE	2
ESTRATÉGIA DE PONTUAÇÃO	3
REPRESENTAÇÃO DE ALGORITMOS	5
INTRODUÇÃO	5
Representação em Linguagem Estruturada	5
Representação em Diagrama Estruturado	6
Representação em Fluxograma	6
REPRESENTAÇÃO DAS ESTRUTURAS	7
Estrutura de Seqüência	7
Estrutura de Seleção Simples	7
Estrutura de Seleção Completa	7
Estrutura da Repetição (com teste a priori)	8
CONTEXTO COMPUTACIONAL	12
MEMÓRIA - MODELO ESQUEMÁTICO	12
Introdução	12
Tabela de Símbolos	13
INSTRUÇÕES PRIMITIVAS	14
Instrução de Atribuição	14
Instrução de Leitura	15
Instrução de Impressão	15
TIPOS DE DADOS SIMPLES	17
INTRODUÇÃO	17
PRINCIPAIS TIPOS	17
COMANDO PARA DEFINIÇÃO DE TIPO	18
EXPRESSÕES ARITMÉTICAS	18
Funções Simples	19
Operadores e funções envolvendo variáveis do Tipo Data	20
EXPRESSÕES LITERAIS	21
Funções Literais Usuais	21
EXPRESSÕES RELACIONAIS	22
EXPRESSÕES LÓGICAS	23
Tabelas Verdade dos Operadores Lógicos	23

## O Conceito de Algoritmo

### Introdução

Na descrição de um *processo* - seja uma *receita culinária*, uma *rotina administrativa* ou um *programa de computador* espera-se que um conjunto básico de ações - chamadas de instruções primitivas - sejam conhecidas e realizáveis pelo *responsável* pela execução deste processo - seja o *cozinheiro*, um *auxiliar administrativo* ou o *computador*.

Um *algoritmo* - entendido como uma *rotina*, um *processo* - pode ser conceituado de diversas formas:

- uma Sequência ordenada, finita e não ambígua de etapas que conduzem à solução de um problema.
- descrição de um conjunto padronizado de ações primitivas, bem definidas e executáveis, que encadeiam a realização de uma tarefa.
- processo de cálculo ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, as regras formais para a obtenção do resultado ou da solução do problema (Aurélio).

Assim, o conceito de *algoritmo*, embora fortemente associado a área de computação, pode ser entendido no seu sentido mais amplo, ou seja, como sinônimo de *processo*, *rotina* ou *procedimento* e, neste sentido, pode estar imerso em qualquer contexto.

Considere os exemplos preliminares que se seguem:

---

#### Exemplo 1.

##### **Contexto Culinário**

Misture os ingredientes.  
Unte o tabuleiro com manteiga.  
Despeje a mistura no tabuleiro.  
Se (há queijo parmesão) então  
    Espalhe sobre a mistura.  
Leve o tabuleiro no forno.  
Enquanto (não corar).  
    Deixe o tabuleiro no forno.  
Deixe esfriar.  
Experimente antes de servir.

##### **Contexto Computacional**

Armazene os graus do aluno.  
Calcule a média destes graus.  
Se (média é superior a 7) então  
    Exibe mensagem "Aprovado"  
Senão  
    Exibe mensagem "Reprovado"

##### **Contexto Administrativo**

Verifique preenchimento do formulário.  
Se (preenchimento correto) então  
    Arquive o documento.  
    Forneça protocolo.  
Senão  
    Lamente.  
    Torne a lamentar.  
    Mande o cliente comprar outro formulário.  
Despeça-se educadamente do cliente.

Assim, ações como

*Unte o tabuleiro com manteiga      e      Leve o tabuleiro ao forno*

são consideradas atividades primitivas do domínio do executor deste processo: “o cozinheiro” !

Desta forma, em cada contexto considerado, o conjunto das instruções primitivas são o ponto de partida para a criação de processos de complexidade crescente.

Entretanto, a *forma de agregação das instruções* são aspectos pertinentes a qualquer processo, independentemente, portanto, do contexto a que se referem.

Estes *mecanismos de agregação* dizem respeito às estruturas lógicas que regulam a forma de como a execução de um processo se articula com as instruções primitivas que o compõem.

## Estruturas de Controle

Nos exemplos introdutórios analisaremos, intuitivamente, as três formas básicas de agregar instruções primitivas: *seqüenciação*, *seleção* (escolha) e *repetição*.

Assim, por exemplo, nos trechos

*Misture os ingredientes.*

*Unte o tabuleiro com manteiga.*

*Despeje a mistura no tabuleiro.*

*Arquive o documento.*

*Forneça protocolo.*

*Armazene os graus do aluno*

*Calcule as médias destes graus.*

é senso comum que a idéia que desejamos transmitir é a de que estas instruções devem ser executadas *uma em seqüência à outra*.

Este é o primeiro mecanismo de agregação utilizado para gerar processos complexos a partir de *processos mais simples*, ou seja: a *Seqüenciação*.

Outra forma de agregação pode ser observada nos trechos:

*se (há queijo parmezon) então*

*Espalhe sobre a mistura*

*se (média é superior a 7) então*

*Exiba mensagem "Aprovado".*

*Senão*

*Exiba mensagem "Reprovado"*

É natural que a intenção nestes trechos é que as instruções primitivas sejam ou não executadas, em função da condição estabelecida ser ou não atendida.

No primeiro caso, somente quando a *condição*

*há queijo parmezon*

for verdadeira, a instrução

*Espalhe sobre a mistura*

será, de fato, executada pelo cozinheiro...

No segundo caso, dependendo da condição

*Média é superior a 7*

ser satisfeita uma **única** das duas instruções

*Exibe mensagem "Aprovado"*      ou      *Exibe mensagem "Reprovado"*

será realizada...

Assim, há um mecanismo de *Seleção* das instruções a serem executadas, que depende da situação de *veracidade* ou *falsidade* da(s) condição(ões) imposta(s) quando do momento da execução do trecho considerado.

Finalmente, uma terceira forma de agregação de instruções pode ser visualizada no trecho:

**enquanto** (*não corar*)

*deixe o tabuleiro no forno.*

onde percebemos que um mecanismo de *Repetição* subordinado à condução **não corou?** regula a execução da instrução primitiva

*Deixe o tabuleiro no forno*

Esta discussão introduz as características estruturais que podem ser observadas nos processos ou algoritmos e aborda, intuitivamente, as principais formas de agregação que permitem criar algoritmos de complexidade crescente a partir de processos mais simples.

## Estratégia de Pontuação

E importante assinalar uma questão fundamental para a correta redação de algoritmos: o uso adequado de *Pontuação* ou *Delimitadores*.

Por exemplo, na *Aritmética* são utilizados os *parênteses*, os *colchetes* e as *chaves* para definir a efetiva ordem em que se deseja a realização de operações em uma expressão (além das manjadas regras como "*a multiplicação vem antes da soma*", etc).

Assim as expressões

$5 + 8 \times 3 - 2$       e       $(5 + 8) \times (3 - 2)$

nos conduzem, naturalmente, a resultados diferentes. Os parênteses (etc) são considerados *sinais de pontuação*, no sentido de possibilitarem a correta identificação de *como* a agregação dos símbolos utilizados deverá ser interpretada.

Da mesma forma, no Português, os sinais de pontuação - *virgula*, *ponto-e-vírgula*, *dois-pontos* e *travessão* são considerados elementos de pontuação<sup>1</sup>.

---

1. Uma conhecida brincadeira sugere que se estabeleça uma pontuação adequada para que a frase "Levar uma pedra do Brasil a Portugal uma andorinha só não faz verão" seja compreensível...

# Construção de Algoritmos - Texto Introdutório

## Professores: Nehab e Manuel

---

Considere as duas formas que se seguem, para a redação do exemplo do *Contexto Administrativo*, onde apenas alteramos o alinhamento vertical – a *indentação* – da última instrução:

*se (Preenchimento correto) então*

*Arquive o documento;  
Forneça protocolo.*

*Senão*

*Lamente;  
Torne a lamentar;  
Mande o cliente comprar outro formulário.*

*Despeça-se educadamente do cliente.*

*Se (preenchimento correto) então*

*Arquive o documento;  
Forneça o protocolo.*

*Senão*

*Lamente;  
Torne a lamentar;  
Mande o cliente comprar outro formulário.*

*Despeça-se educadamente do cliente.*

Para cada uma das duas situações, a questão que se coloca é a óbvia: o *executor* do algoritmo se despede educadamente do cliente *apenas* preenchimento do formulário estiver incorreto, ou procederá desta forma em *qualquer circunstância*? Se apenas o alinhamento fosse responsável pela interpretação, estaria garantida a confusão!

Suponha que o *Algoritmo* em discussão fosse redigido como se segue, portanto, sem que alinhamentos possuam qualquer importância.

*Se (preenchimento correto) então Arquive o documento; Forneça protocolo. Senão Lamente; Torne a lamentar. Mande o cliente comprar outro formulário. Despeça-se educadamente do cliente.*

E agora, sem o alinhamento, a forma de interpretá-lo é única? Felizmente, é... Perceba que a função pretendida para o *ponto-e-vírgula* e para o *ponto* são distintas. O *ponto* define a finalização de uma estrutura - seja uma *Seqüência*, uma *Seleção* ou uma *Repetição*.

Suponha, entretanto, que inventássemos uma outra forma de delimitar as estruturas de *Seleção* e de *Repetição* através de marcadores - as expressões *Fim-do-se* e *Fim-do-enquanto* - que serão utilizadas para definir onde as estruturas de *Seleção* e de *Repetição* terminam...

Nesta estratégia a redação dos algoritmos, bem como sua adequada interpretação, tornam-se mais simples de estabelecer.

A idéia é que todas as instruções dos blocos da condicional estejam entre as expressões *então/senão* e entre *senão/fim-do-se*; da mesma forma, todas as instruções dentro do processo de repetição são escritas entre as expressões *Enquanto* e *Fim-do-enquanto*. O uso de diferentes alinhamentos passa a ser apenas uma forma de facilitar a leitura, não sendo essencial para a adequada interpretação do algoritmo. As redações que se segue ilustram esta discussão.

*Se (preenchimento correto) então*

*Arquive o documento  
Forneça protocolo*

*Senão*

*Lamente  
Torne a lamentar  
Mande o cliente comprar outro formulário*

*Fim-do-se*

*Despeça-se educadamente do cliente*

*Se (há queijo parmezon)*

*Espalhe sobre a mistura*

*Fim-do-se*

*Leve o tabuleiro ao forno*

*Enquanto (não corar)*

*Deixe o tabuleiro no forno*

*Fim-do-Enquanto*

*Deixe esfriar*

*Experimente antes de servir*

Em cada linguagem de programação que estudar, você perceberá que a forma de pontuação utiliza uma combinação de *sinais de pontuação* propriamente ditos e *delimitadores*<sup>2</sup>...

---

<sup>2</sup> Solicite exemplos de seu professor...

## Representação de Algoritmos

Nesta unidade estudaremos as principais formas de representação das três estruturas básicas de controle usadas na descrição de processos estruturados: *Seqüência*, *Seleção* (ou decisão) e *Repetição*.

Assim, nesta etapa, é absolutamente irrelevante o contexto do algoritmo em discussão, mas apenas como se articulam as ações para a concepção de estruturas mais complexas.

Abordaremos as seguintes representações: *Linguagem Estruturada*, também chamada de *Pseudocódigo* ou *Pseudo-linguagem*, o algumas vezes útil, embora inadequado, *Fluxograma Convencional* e o interessante *Diagrama Estruturado*.

## Introdução

Considere o processo a seguir, descrito de forma livre.

*"Olhe para a rua. Caso venha um carro, espere; caso contrário, atravesse".*

Observe que este processo pode ser entendido como constituído de dois sub-processos a serem executados um depois do outro - ou seja, em seqüência:

- o primeiro, constituído exclusivamente da instrução primitiva *Olhe para a rua*;
- o segundo, uma *estrutura de seleção*, cuja condição para teste é *Vem Carro?*

É importante assinalar que uma estrutura de seleção, embora de nível de complexidade maior do que a de uma simples instrução primitiva *pode e deve ser entendida como um todo!* Assim, neste exemplo, sob o ponto de vista da execução deste processo, há duas etapas.

## Representação em Linguagem Estruturada

Na representação de uma estrutura condicional *Pseudo-linguagem*, note que:

- a *condição* está descrita *entre* as expressões "*Se*" e "*então*";
- as *instruções* a serem executadas quando a condição for *verdadeira*, serão descritas *entre* as expressões "*Então*" e "*Senão*";
- as *instruções* a serem executadas quando a *condição* é *falsa* estão descritas entre as expressões "*Senão*" e "*Fim-do-se*".

Linguagem Estruturada	
<b>Etapa 1</b>	Olhe para a rua
<b>Etapa 2</b>	Se (vem carro) então Espere Senão Atravesse Fim-do-se

Obs: A representação de estruturas de repetição será abordada formalmente adiante.

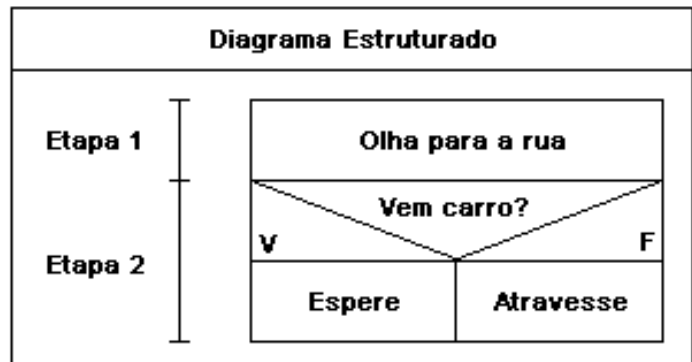
## Representação em Diagrama Estruturado

Observe a representação em diagrama estruturado.

Um algoritmo é representado por um *retângulo* cujas *fatias* horizontais são suas etapas em seqüência.

Além disso, a representação de estruturas de seleção e de repetição são *retângulos especiais*.

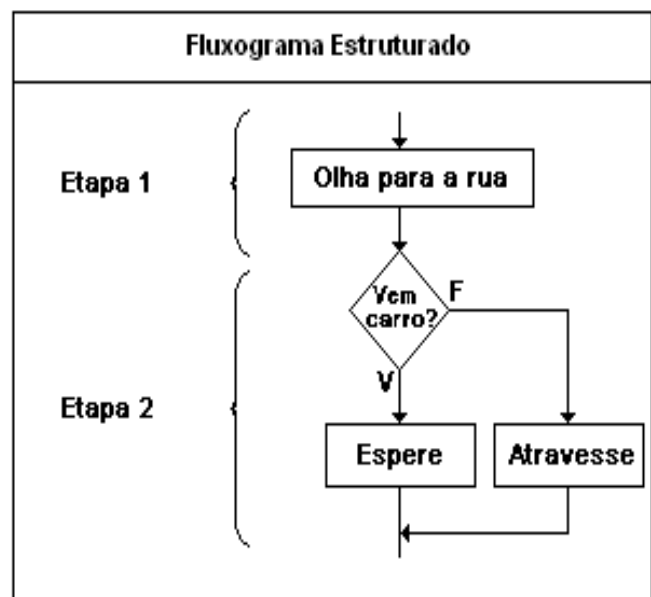
Observe, intuitivamente, o retângulo utilizado para a estrutura de seleção deste exemplo.



## Representação em Fluxograma

A representação em Fluxograma possui duas características principais:

- as *instruções primitivas* são expressas em *retângulos* (se houver várias instruções primitivas em seqüência para simplificar, podem ser descritas em um único retângulo).
- os *testes* (tanto nas *estruturas de seleção* quanto nas de repetição) são explicitados em *losangos*, onde a mágica é o “*siga a seta*” para acompanhar que instruções serão executadas quando a condição do teste for *verdadeira* ou for *falsa*.

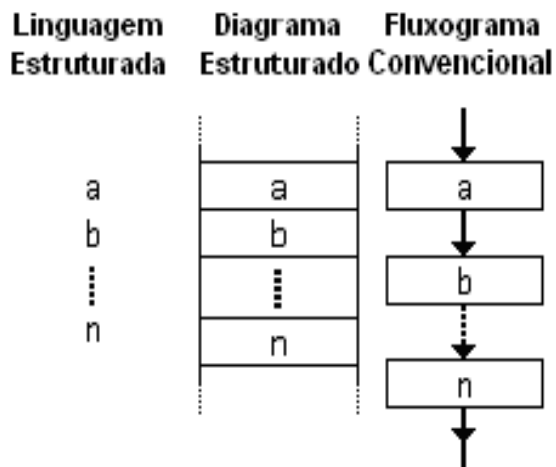


Obs.: Nesta forma de representação, pode ocorrer que a lógica do processo descrito seja um verdadeiro labirinto (adiante esta discussão será enfatizada), muito embora, neste exemplo, sejam indicadas as duas etapas que constituem este processo.

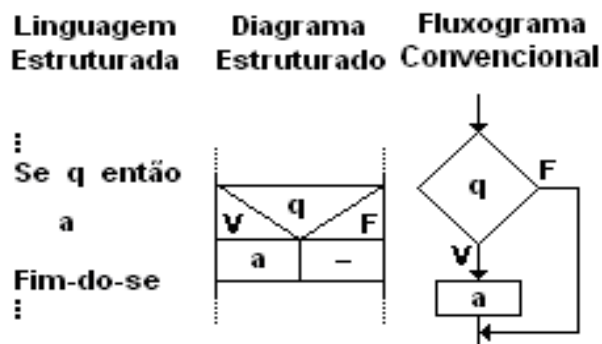


## Representação das Estruturas

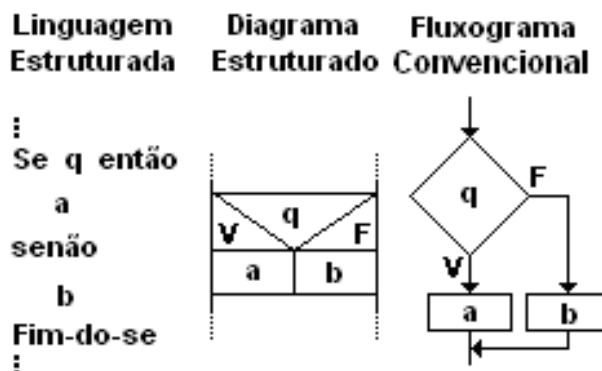
### *Estrutura de Seqüência*



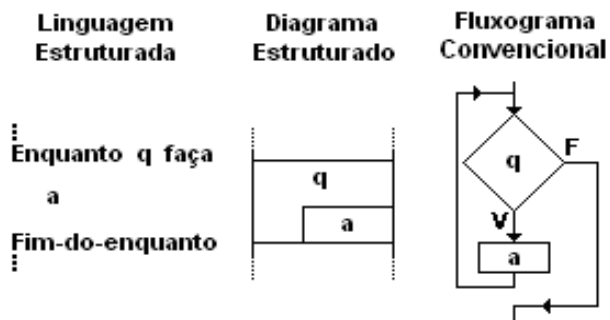
### *Estrutura de Seleção Simples*



### *Estrutura de Seleção Completa*



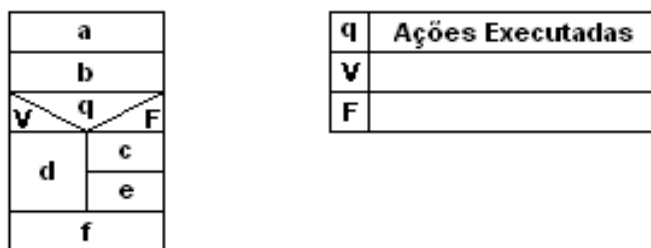
## Estrutura de Repetição (com teste a priori)



Nos três exemplos que se seguem, assuma que **a**, **b**, etc, são *instruções primitivas* e **q**, **q1**, **q2**, etc, são condições a serem testadas. Perceba que o *algoritmo* está representado em apenas uma das formas estudadas. Para cada um dos exemplos:

- Represente o algoritmo nas duas outras formas;
- Complete o quadro de *Ações Executadas*, onde devem ser explicitadas. Em cada linha, as instruções a serem executadas em cada situação de veracidade ou falsidade das condições existentes.

### Exemplo 2.

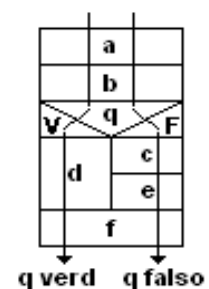


### Solução

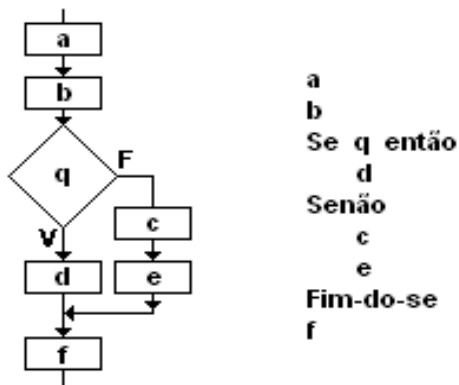
Observe que este algoritmo, a rigor, possui 4 etapas em seqüência: as instruções primitivas **a**, **b**, uma estrutura de seleção (condicional) e a instrução primitiva **f**.

Analisando a figura podemos observar que, em qualquer situação, as instruções primitivas **a**, **b** e **f** serão sempre executadas (veja em *itálico* na tabela de ações). Entretanto, em função da estrutura condicional, observamos que: se **q** é verdadeira, é executada a ação **d**; caso contrário, são executadas as ações **c** e **e** (escritas sublinhadas na tabela, para melhor compreensão).

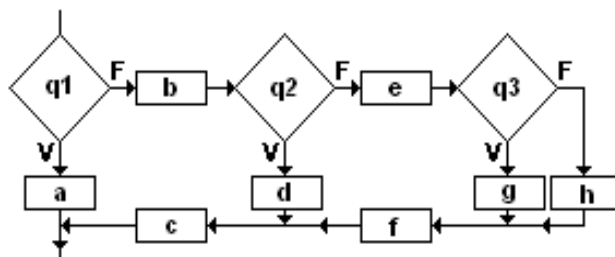
q	Ações Executadas
V	<i>a, b, d, f</i>
F	<i>a, b, <u>c</u>, <u>e</u>, f</i>



A representação deste exemplo em *Linguagem Estruturada Convencional* é imediata.



Exemplo 3.



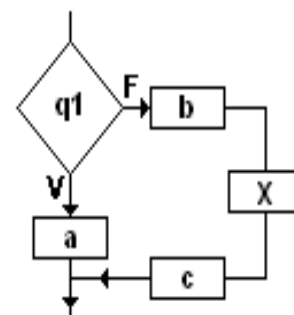
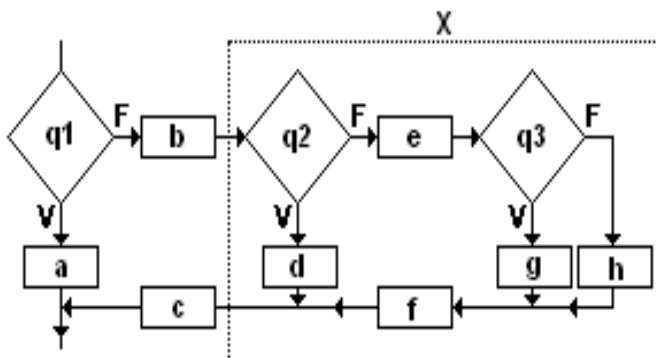
q1	q2	q3	Ações Executadas
V	V	V	
V	V	F	
V	F	V	
V	F	F	
F	V	V	
F	V	F	
F	F	V	
F	F	F	

Solução

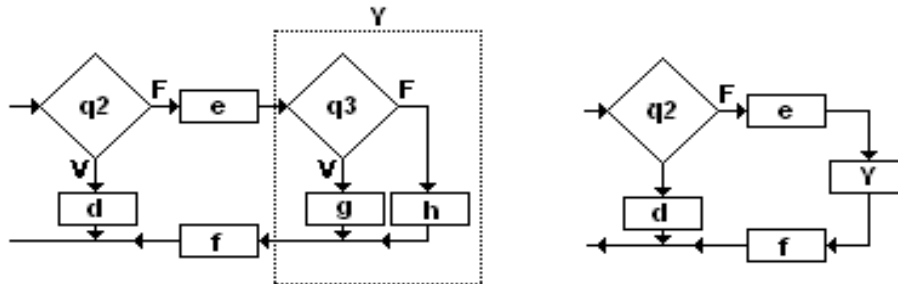
Observe que este algoritmo é constituído, para quem olha de fora, digamos assim, de uma única estrutura de Seleção (condicional)!

Quando a condição **q1** é verdadeira, é executada a instrução **a**; caso contrário são executados os processos **b**, **X**, e **c**, conforme indicado.

A figura da direita ilustra o desenho do fluxograma com a etapa **X** substituindo a região tracejada.



Por outro lado, o sub-processo **X** é constituído, também, de uma única seleção com condição **q2**; quando **q2** é verdadeira, é executada a instrução **d**; caso contrário, são executadas as etapas **e**, **Y** e **f**, conforme indicado a seguir. Note também que **Y** é uma simples condicional.

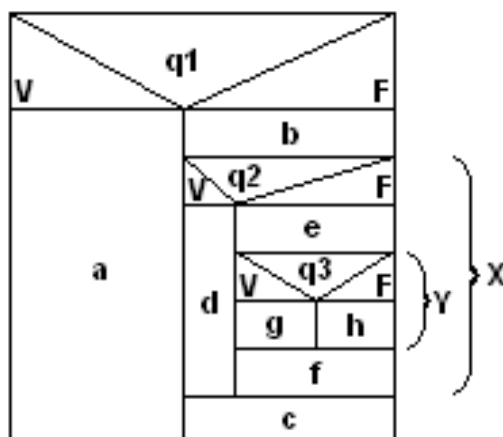


Refazendo esta análise, podemos preencher a tabela de ações e descrever o algoritmo proposto na forma de *Diagrama Estruturado* e *Linguagem Estruturada*:

Note que a Tabela de Ações pode ser preenchida como se segue:

- as 4 primeiras linhas (onde **q1** verdadeiro) são preenchidas com **a** e mais nada...;
- as 4 últimas linhas (onde **q1** é falsa) necessariamente começam com **b** e terminam com **c**;
- quando **q1** é falsa e **q2** é verdadeira, além de **b** e **c**, já incluídas na tabela, é executada **d** e pronto;
- quando (ainda) **q1** é falsa e **q2** também, além de **b** e **c**, já incluídas na tabela, são executadas **e** e **f**;
- finalmente, ainda quando **q1** e **q2** são falsas, dependendo de **q3** ser executada **g** ou **h** (duas últimas linhas)...

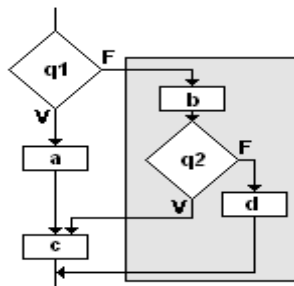
q1	q2	q3	Ações Executadas
V	V	V	a
V	V	F	a
V	F	V	a
V	F	F	a
F	V	V	b, d, c
F	V	F	b, d, c
F	F	V	b, e, g, f, c
F	F	F	b, e, h, f, c



```

Se q1 então
  a
Senão
  b
  x → Se q2 então
    d
    Senão
      e
      Se q3 então ← Y
        g
      senão
        h
      Fim-do-se ← Y
    f
  x → Fim-do-se
  c
Fim-do-se
  
```

Exemplo 4.



q1	q2	Ações Executadas
V	V	
V	F	
F	V	
F	F	

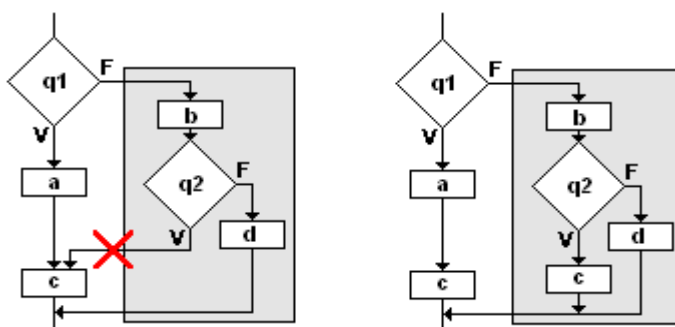
Solução

Observe que este algoritmo, descrito sob a forma de fluxograma, explicita uma situação curiosa: não é possível *separar* os lados *verdadeiro* e *falso* da condicional **q1**, uma vez que a instrução **c** é utilizada por ambos os *lados*! Dizemos, intuitivamente, que este algoritmo não é estruturado...

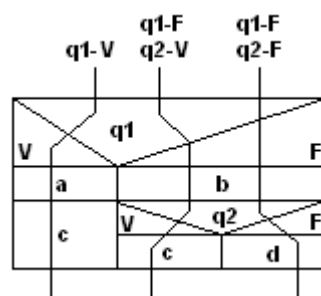
Uma situação como esta só é possível na *Representação em Fluxograma*; é impossível um algoritmo descrito em *Diagrama Estruturado* ou na *Linguagem Estruturada* ter uma anomalia semelhante à exibida neste exemplo (portanto a expressão estruturada nestas representações, não é gratuita...).

Assim, para expressar este algoritmo em uma dessas duas formas é necessário eliminar a ligação *mais escura* e *duplicar a instrução c* na saída de **q2**, conforme sugerido na figura. Estritamente falando, criamos um algoritmo equivalente tornando-o representável de forma estruturada.

Nesta condição, a solução é a que se segue:



q1	q2	Ações Executadas
V	V	a, c
V	F	a, c
F	V	b, c (o novo "c")
F	F	b, d



Se q1 então  
  a  
  c  
Senão  
  b  
  Se q2 então  
    c  
  Senão  
    d  
  Fim-do-se  
Fim-do-se

## Contexto Computacional

### Memória Modelo Esquemático

#### Introdução

Um modelo introdutório para Memória Principal de um computador é o de uma seqüência de células numeradas capazes de armazenar, cada uma, a cada instante, uma dada informação.

A referência a cada célula da memória é realizada através de seu endereço numérico.

Assim, uma ação primitiva para criar um determinado valor em uma célula pode ser expressa da seguinte forma:

*Armazene o valor **25** na posição de memória de endereço **2***

e o resultado obtido pode ser representado, esquematicamente, como na figura ao lado.

Memória			
0	1	2	3
		25	
4	5	6	7

Basicamente, para uma informação ser *tratada* por um computador, precisa estar armazenada em sua *Memória Principal*. Assim, em um programa para calcular a média de dois dados, é necessário que os dados estejam previamente armazenados na memória para que a média seja *calculada* e fique disponível em uma *célula* da mesma.

Um programa que *receba* dois números, calcule sua média e *exiba* este valor no vídeo possui, necessariamente, três *instruções* bem distintas.

- uma instrução primitiva de Leitura, que transferirá os dados de algum equipamento de Entrada (teclado, por exemplo) para a *memória*.
- uma instrução primitiva de Atribuição, que calculará o valor da média a partir dos valores das parcelas já armazenadas na memória e a gravará em determinada posição na memória.
- e, finalmente, uma instrução primitiva de impressão que transferirá uma cópia do valor da média, já armazenada na memória, para algum equipamento de saída (vídeo, por exemplo).

---

#### Exemplo 5.

Se desejamos *computar* a média obtida por um aluno em um curso, podemos, numa formulação preliminar, escrever a seguinte seqüência de instruções intuitivas:

- *Leia o valor do grau obtido na 1ª prova e o armazene na posição de memória **1***;
- *Leia o valor do grau obtido na 2ª prova e o armazene na posição de memória **3***;
- *Atribua a posição de memória **7** a média dos valores armazenados nas posições **1** e **3***;
- *Imprima o valor armazenado na posição de memória de número **7***.

Memória			
0	1	2	3
	6		7
4	5	6	7
			6,5

Supondo que os dois graus fornecidos foram **6** e **7**, após a execução da terceira ação (primitiva), a memória poderá ser representada, esquematicamente, como sugere a figura.

## Tabela de Símbolos

A descrição do processo anterior, que explicita na sua própria formulação os *endereços* utilizados para o armazenamento das informações é, sem dúvida, um fator excessivamente limitante para a descrição eficaz de algoritmos computacionais.

O conceito de *nome simbólico*, que possibilita a associação de apelidos às células da memória, é a estratégia que viabiliza uma maior liberdade na descrição de programas. Assim, a gerência dos endereços das células utilizadas em um algoritmo é deslocada do *programador* para a *máquina*.

Esta associação é feita através da chamada *Tabela de Símbolos* que, em essência, gerencia a associação dos *Nomes Simbólicos* escolhidos aos endereços das células de memória disponíveis para utilização.

**Tabela de Símbolos**

Nome Simbólico	Endereço

Neste texto utilizaremos o termo *variável* para referência ao nome simbólico (ou apelido) associado a uma posição de memória, durante a execução de um particular processo.

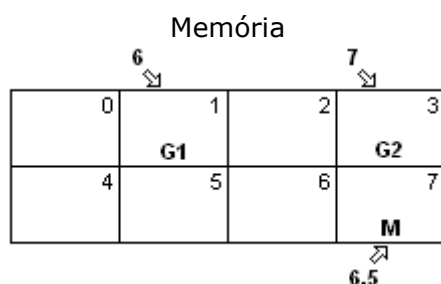
---

### Exemplo 6.

O exemplo anterior, que calcula a média obtida por um aluno em um curso, pode ser descrito de forma mais concisa como se segue (use a intuição, por enquanto...).

Leia G1  
Leia G2  
 $M \leftarrow (G1+G2)/2$   
Imprima M

Representando no esquema que se segue o mecanismo de atribuição automática de endereços de memória aos nomes simbólicos utilizados na formulação das instruções, temos:



**Tabela de Símbolos**

Nome Simbólico	Endereço
<b>G1</b>	<b>1</b>
<b>G2</b>	<b>3</b>
<b>M</b>	<b>7</b>

## Instruções Primitivas

### Instrução de Atribuição

#### Formato

*variável*  $\leftarrow$  *expressão*

#### Semântica

O resultado da *expressão* será armazenado sob o nome simbólico que está à esquerda do sinal de atribuição:  $\leftarrow$

Tal *expressão* poderá ser um simples valor numérico ou poderá conter operações envolvendo informações já armazenadas na memória e associados a nomes simbólicos previamente definidos.

---

### Exemplo 7.

No trecho do algoritmo descrito a seguir retratamos em um *esquema* que chamaremos de *Chinês*, uma seqüência de *fotografias* da memória, que refletem, a cada passo, o conteúdo de cada posição de memória.

#### Algoritmo

$A \leftarrow 3$   
 $B \leftarrow 2$   
 $C \leftarrow A - B$   
 $B \leftarrow 5$   
 $C \leftarrow A + B$

#### Chinês

A	B	C
3		
3	2	
3	5	1
3	5	1
3	5	8

#### Memória

	0	1	2	3
		A		B
	4	5	6	7
			C	

Diagram illustrating memory state transitions:  
- Initial state: A at index 1, B at index 3.  
- Transition 1: A points to index 3 (value 1), B points to index 5 (value 5).  
- Transition 2: A points to index 5 (value 5), B points to index 8 (value 8).

Note que este esquema (o Chinês...) possui uma vantagem sobre a representação da memória sob a forma de tabela de células: nele, você tem conhecimento do conteúdo da memória ao longo do tempo...

---

### Exemplo 8.

Considere o trecho de algoritmo descrito.

Observe que uma instrução como  $A \leftarrow A + A$  deve ser interpretada com cuidado: o Lado esquerdo (**A**) deve ser substituído pelo resultado da expressão do lado direito ( $A + A$ ); como o valor de **A** neste instante é **3**, a operação de soma do lado direito é realizada com o valor **3**; logo, o resultado da expressão é **6** e substituirá, portanto, o valor atual de **A**. Portanto há uma questão temporal a ser considerada nestas situações.

	A	B
$A \leftarrow 3$	3	
$A \leftarrow A + A$	6	
$A \leftarrow A + A$	12	
$A \leftarrow A + A$	24	
$B \leftarrow A + 1$	24	25
$B \leftarrow B + 1$	24	26



## Instrução de Leitura

### Formato

**Leia** lista-de-variáveis

### Semântica

A execução da instrução de leitura pressupõe que os dados serão fornecidos do meio externo (teclado, arquivo gravado em disco, etc) e serão armazenados na memória sob os nomes simbólicos, na ordem em que figuram no comando.

---

### Exemplo 9.

Assuma que a instrução de Leitura interprete o Teclado como o dispositivo padrão de Entrada de Dados e que sejam digitados os valores **3** e **5** quando da solicitação de dados pela execução da instrução **Leia A, B**.

O esquema indicada sugere o *Chinês* e os valores disponíveis na *Memória* durante a execução do algoritmo dado.

Algoritmo	Chinês	Memória																													
Leia A, B C ← A + B A ← 7	<table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>3</td><td>5</td><td></td></tr><tr><td>7</td><td></td><td>8</td></tr></table>	A	B	C	3	5		7		8	<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td></td><td></td><td>A</td><td></td><td>B</td></tr><tr><td></td><td>4</td><td></td><td>5</td><td>6</td></tr><tr><td></td><td></td><td></td><td>C</td><td>7</td></tr></table>		0	1	2	3			A		B		4		5	6				C	7
A	B	C																													
3	5																														
7		8																													
	0	1	2	3																											
		A		B																											
	4		5	6																											
			C	7																											

## Instrução de Impressão

### Formato

**Imprima** variáveis ou "texto"

### Semântica

A execução da instrução de impressão pressupõe que informações estão armazenadas na memória e serão colocadas no meio externo (tela de vídeo, arquivo em disco, impressora, etc) através da referência nos nomes simbólicos constantes da lista de variáveis. A opção "texto" prevista no formato da instrução permite também que sejam explicitados textos para a documentação das saídas.

### Exemplo 10.

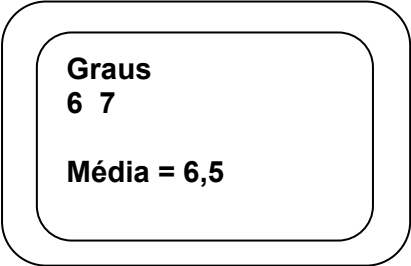
Assuma que a instrução de impressão utilize o vídeo como dispositivo padrão de Saída de Dados. No exemplo que se segue, observe que a instrução de impressão não é adequadamente representada no esquema do *Chinês* nem, é claro, no de *Memória*. Estas instruções poderiam ser representadas, pictoricamente, na tela exibida.

Algoritmo	Saída (vídeo...)
$A \leftarrow 3$ $B \leftarrow 5$ $C \leftarrow 2$ <b>Imprima</b> A,B,C <b>Imprima</b> A,B,C <b>Imprima</b> A,B,C	

---

### Exemplo 11

Analise o trecho de algoritmo descrito a seguir e a representação da saída gerada.

Algoritmo	Saída (vídeo...)
<i>Leia</i> G1, G2 <i>Imprima</i> "Graus" <i>Imprima</i> G1, G2 $M \leftarrow (G1+G2)/2$ <i>Imprima</i> <i>Imprima</i> "Média =", M	

Observe que assumimos que podem ser separados por *ponto-e-vírgula* os elementos que compõem as informações para uma mesma instrução de saída. Estas informações são "impressas" na mesma linha do dispositivo de saída, conforme a segunda e terceira instruções de impressão.

## Tipos de Dados Simples

### Introdução

Na introdução do conceito de Memória sugerimos que cada célula (posição) de memória pode armazenar a cada instante, uma determinada informação.

Na realidade este é um modelo simplificado do mecanismo de armazenamento de informações na memória.

Uma informação numérica, textual, etc exige, de fato, mais do que uma posição de memória para armazená-la. O número de células necessárias depende do tipo de dado utilizado.

Um modelo mais adequado de uso da memória pressupõe que na *Tabela de Símbolos* sejam informados o endereço da célula inicial utilizada para armazenar a informação e o número de células necessárias (que dependerá apenas do tipo de dado a ser usado).

Assim esta discussão torna clara a importância, na *Construção de Algoritmos*, a especificação dos tipos de dados a serem armazenados nas variáveis utilizadas em um particular algoritmo.

Na discussão de algoritmos, no capítulo anterior, estas questões não foram abordadas, considerando a importância, naquele momento, de enfatizarmos as estruturas lógicas de programação.

Este capítulo aborda a forma como podem ser declarados os tipos de dados a serem utilizados nos algoritmos. Assim, cada *nome simbólico* em um algoritmo terá associado um tipo de dado. Desta forma o sistema poderá gerenciar, com a *Tabela de Símbolos*, o correto armazenamento de seus valores na memória.

### Principais Tipos

Os *tipos de dados básicos* que utilizaremos neste texto são *Inteiro*, *Real*, *Literal* (texto), *Data* e *Lógico*. Este último, sem dúvida pouco natural para quem se inicia na arte de programar, é fundamental sob o ponto de vista conceitual, como veremos oportunamente.

A utilização de *nomes simbólicos* pressupõe, a partir de agora, a especificação do *tipo de dado* que será associado ao mesmo:

<i>Inteiro</i>	valores numéricos inteiros
<i>Real</i>	valores numéricos não inteiros
<i>Literal</i>	valores alfanuméricos (seqüência de caracteres)
<i>Data</i>	valores representativos de datas
<i>Lógico</i>	valores que assumem apenas dois estados: verdadeiro (.V.) ou falso (.F.)

### Exemplo 12.

Convencionaremos, conforme expresso nos exemplos que se seguem, que um valor literal deverá ser expresso entre aspas e uma data deverá ser expressa entre colchetes.

Valor	Tipo	Valor	Tipo
123	inteiro	{01/03/1992}	data
"123"	literal	"abcd"	literal de tamanho 4
325,27	real (não inteiro)	"A B C"	literal de tamanho 5
0,12345	real (não inteiro)	.V.	valor lógico verdadeiro
{23/07/94}	data	.F.	valor lógico falso

## Comando para Definição de Tipo

Para declarar que um *nome simbólico* está associado a um determinado *tipo de dado*, utilizaremos um *comando declarativo* que, após a palavra **var**, explicita os nomes das variáveis desejadas e os tipos que serão associados assim:

```
var nomes: tipo
    nomes: tipo
    nomes: tipo
    :
```

onde *tipo* pode assumir as opções *inteiro*, *real*, *literal*, *data* ou *lógico*.

---

### Exemplo 13.

```
var   DtNasc   : data      var   Pedaco   : literal      var   G1, G2, M : real
      Nome     : literal   N1, N2      : inteiro      Dt1, Dt2       : data
      N        : inteiro   Pagon      : lógico
      Salário   : real
```

Na notação a ser utilizada podem ser declaradas diversas variáveis em uma única declaração de tipo.

## Expressões Aritméticas

Expressões aritméticas são expressões onde intervêm *variáveis* do tipo numérico e os operadores associados a este tipo de dado.

Os *operadores numéricos* que serão assumidos em nossa pseudo-linguagem são:

Operador	Operação	Operador	Operação
+	adição	—	subtração
*	multiplicação	/	divisão
^	exponenciação		

Quando ocorrem vários operadores, a ordem dos operadores é a usual da matemática: primeiro a *potenciação*, em seguida a *multiplicação* e *divisão* e, a seguir, a *adição* e *subtração*.

Quando uma operação envolve operadores de mesma precedência, as operações devem ser realizadas da esquerda para a direita. Além disso, o uso dos parênteses deve ser incentivado para facilitar a compreensão das expressões.

### Funções Simples

É comum estarem disponíveis nas linguagens de programação as chamadas *funções internas*, utilizadas para cálculos matemáticos. Por exemplo, o logaritmo de um número **X** – escreve-se **log(X)**, o seno de um ângulo **X** – escreve-se **sen(X)**, etc. Assumiremos que estas funções, sempre que necessário, poderão ser livremente usadas em nossa *Pseudo-Linguagem*.

Além das funções trigonométricas e logarítmicas é útil dispor, em nossa *pseudo-linguagem*, de mais duas funções especiais, que calculam o resto e o quociente da *divisão inteira* de dois números inteiros.

**Mod(A, B)** : fornece o resto da divisão inteira entre A e B

**Div(A, B)** : fornece o quociente da divisão inteira de A por B

---

#### Exemplo 14.

Analise o resultado da variável **X** em cada uma das atribuições indicadas. Assuma que **A=2, B=4, C=5** e **D=14**.

Atribuição	Resultado
$X \leftarrow A - B * C$	$2 - 4 * 5 = -18$
$X \leftarrow A ^ B - C$	$2 ^ 4 - 5 = 11$
$X \leftarrow A * B + C$	$2 * 4 + 5 = 13$
$X \leftarrow A * (B + C)$	$2 * (4 + 5) = 18$
$X \leftarrow A * (B - C)$	$2 * (4 - 5) = -2$

Atribuição	Resultado
$X \leftarrow \text{Mod}(D, B)$	resto... = 2
$X \leftarrow \text{Div}(D, B)$	quoc... = 3
$X \leftarrow \text{Mod}(0, A)$	resto... = 0
$X \leftarrow \text{Div}(0, A)$	quoc... = 0
$X \leftarrow \text{Div}(A, 0)$	não definido...

---

#### Exemplo 15.

Suponha que uma universidade utiliza como código de matrícula, um número inteiro no formato AASDDDD onde:

- os dois primeiros algarismos são os dois últimos dígitos do ano da matrícula;
- o terceiro dígito vale 1 ou 2, conforme o aluno tenha se matriculado no 1º ou 2º semestre;
- os 4 últimos dígitos são o seqüencial propriamente dito da matrícula do aluno.

Crie um algoritmo que leia o número de matrícula de um aluno e imprima o ano e o semestre em que foi matriculado.

Solução

##### Algoritmo Matrícula

```
Var NumMatric, Ano, Semestre, Aux : Inteiro
Leia NumMatric
Aux ← Div(NumMatr, 10000)
Semestre ← Mod(Aux, 10)
Ano ← Div(Aux, 10)
Imprima "Ano é", Ano
Imprima "Semestre", Semestre
```

Fim

*Exemplo 16.*

Crie um algoritmo que leia uma data no formato DDMMAA e a imprima no formato AAMMDD. Assuma que a data é lida pelo algoritmo em uma variável inteira e não em uma variável do tipo data.

Solução

```
Algoritmo DataInvertida
  Var Data, Mês, Ano, Dia, DataInv: Inteiro
  Leia Data
  Ano ← Mod(Data, 100)
  Aux ← Div(Data, 100)
  Mês ← Mod(Aux, 100)
  Dia ← Div(Aux, 100)
  DataInv ← Ano + Mês * 100 + Dia * 10000
  Imprima "Data lida", Data
  Imprima "Data Invertida", DataInv
Fim
```

## Operadores e funções envolvendo variáveis do Tipo Data

Algumas funções envolvendo datas são úteis e disponíveis em várias linguagens de programação. Na nossa pseudo-linguagem serão assumidas as funções que se seguem, onde Data, Data1 e Data2 são variáveis do tipo data:

Hoje() : função que retorna a data do sistema  
Ano(Data) : função que retorna o valor numérico do ano da data  
Mês(Data) : função que retorna o valor numérico do mês da data  
Dia(Data) : função que retorna o valor numérico do dia da data

Obs: Em alguns ambientes também é usual ter disponível uma função que cria uma data a partir do dia, mês e ano numéricos. Em geral seu formato é Data(ano, mês, dia).

Também é comum usar o operador *diferença*, assim: Data1 – Data2, que fornece o número de dias decorridos entre as duas datas.

---

*Exemplo 17.*

Analise o resultado das expressões indicadas, onde Data1 = {12/12/96}. Data2 = {5/01/1997}.

Atribuição

Atribuição	Resultado
Ano(Data1)	1996 (inteiro)
Mês(Data1)	3 (inteiro)
Dia(Data1)	12 (inteiro)
Data2 - {30/12/96}	6 (Inteiro)
Data( ano(Data2) ; mês(Data2) + 1; dia(Data2))	{5/02/97} (data)

## Expressões Literais

Expressões literais são expressões cujo resultado são valores literais

Em geral, nestas expressões, ocorre o operador de concatenação (justaposição) de literais e/ou funções que extraem partes de uma literal (adiante).

O operador de concatenação, que designaremos por **&** (o **e** comercial), opera sobre duas literais VarTexto1 e VarTexto2, resultando na justaposição de ambas.

---

### Exemplo 18.

Se L1 = "José", L2 = "#Maria", L3 = "Carlos##" e L4 "#" (onde o sinal # representa um espaço em branco), temos:

Expressão	Resultado	Expressão	Resultado
L1 & L2	"José#Maria"	L1 & L4	"José#"
L2 & L1	"#MariaJosé"	L1 & L4 & L2	"José##Maria"
L3 & L1	"Carlos##José"	L1 & L3	"JoséCarlos##"

## Funções Literais Usuais

Na tabela que se segue estão explicitadas as principais funções que tratam variáveis literais e que serão supostas disponíveis em nossa pseudo-linguagem.

Função	Resultado
Trim(VarTexto)	Elimina de VarTexto as caracteres eventualmente em branco ao seu final;
Comp(VarTexto)	Comprimento de VarTexto (incluindo os eventuais brancos ao final, caso presentes);
Esquerda(VarTexto, N)	Gera uma literal com os N primeiros caracteres de VarTexto;
Direita(VarTexto, N)	Gera uma literal com os N últimos caracteres de VarTexto;
Sub(VarTexto, N, M)	Extrai M caracteres sucessivos de VarTexto começando em seu caractere de ordem N, se M e N são inconsistentes retorna 0 (zero);
Pos(VarTexto, Peçaço)	Retorna a posição da primeira ocorrência da literal Peçaço como subliteral de VarTexto, caso peçaço não ocorra em VarTexto, retorna o valor 0 (zero);
ConvDataTexto(Data1)	Função que retoma Data1 como texto, concatenando o ano completo, o mês e o dia, nesta ordem;
ConvTextoData(Texto1)	Função que retoma uma variável tipo data, sendo fornecido um texto da forma "AAAAMDD".

---

**Exemplo 19.**

Se Lit1 = "Construção#de#Algoritmos", Lit2 = "Lógica#de#Programação" e Lit3 = "de#"

Expressão	Resultado	Expressão	Resultado
Trim(Lit3)	"de"	Esquerda(Lit2, 6)	"Lógica"
Comp(Lit1)	24	Direita(Lit2, 4)	"ação"
Comp(Lit3)	3	Pos(Lit2, Lit3)	8
Comp(Trim(Lit3))	2	Pos(Lit1, "xxx")	0
Sub(Lit1, 8, 4)	"ção#"	Pos(Lit2, "a")	6

## Expressões Relacionais

*Expressões Relacionais* são expressões onde intervêm os operadores chamados relacionais, ou seja, operadores que comparam duas grandezas. Na tabela que se segue estão explicitados os operadores que serão assumidos em nossa linguagem.

Operador	Relação
=	é igual a
<	é precedecessor de
<=	é precedecessor ou igual a
>	é sucessor de
>=	é sucessor ou igual a
<>	é diferente

O resultado de uma expressão relacional é um valor lógico: **.V.** (Verdadeiro) ou **.F.** (Falso).

Obs.: Nas linguagens em geral, não estão disponíveis os sinais de  $\leq$  (menor ou igual),  $\geq$  (maior ou igual) e  $\neq$  (diferente), sendo utilizados os sinais indicados na tabela.

---

**Exemplo 20.**

Considere que:

Lit1 ← "Arara"	Lit2 ← "ar"
Data1 ← {12/04/1994}	Data2 ← {12/05/1993}
N1 ← 7	N2 ← 4,5

Analise os valores assumidos pelas expressões indicadas.

Expressão Relacional	Resultado	Expressão Relacional	Resultado
N1 < N2	.F.	Hoje() > Data2	.V.
N2 >= 4	.V.	Esquerda(Lit1, 3) = Lit2	.F.
Data1 < Data 2	.F.	Ano(Data1) = 1994	.V.



## Expressões Lógicas

Expressões Lógicas são expressões onde intervêm os operadores chamados lógicos, ou seja, operadores que conectam expressões relacionais ou outras expressões lógicas de menor complexidade.

Seguem os operadores que serão assumidos em nossa linguagem:

Operador	Operação
não	Negação
.e.	Conjunção
.ou.	Disjunção

## Tabelas Verdade dos Operadores Lógicos

A tabelas a seguir explicitam como são atribuídos valores **.V.** ou **.F.** às expressões lógicas que utilizam estes operadores.

Exp	Não Exp	Exp1	Exp2	Exp1 .e. Exp2	Exp1 .ou. Exp2
V	F	V	V	V	V
F	V	F	V	F	V
		V	F	F	V
		F	F	F	F

---

### Exemplo 21.

Assuma que:

A ← V

B ← .F.

M ← 3,5

N ← 7

S ← "João"

T ← "José"

Analise o valor atribuído a X, variável lógica, em cada uma das expressões.

X ← Expressão	Valor de X	X ← Expressão	Valor de X
X ← (M >= N)	.F.	X ← A .ou. B	.V.
X ← (M = N/2)	.V.	X ← A .e. B	.F.
X ← (S > T)	.F.	X ← não A	.F.
X ← não (M <> N)	.F.	X ← A .ou. (não B)	.V.