

# Lógica de Programação



Autor: Prof. Luiz Roberto Camilo  
Criação: 08 de Junho de 2011  
Última Atualização: 05 de Junho de 2014

---

## Sumário

1- Introdução.....	03
2- Algoritmo.....	04
3- Fluxograma.....	05
4- Pseudocódigo.....	14
5- A Linguagem de Programação VisuAlg.....	15
6- Tipos de Dados.....	17
7- Nomes de Variáveis e sua Declaração.....	18
8- Operadores.....	20
9- Comandos de entrada e saída de Dados.....	22
10- Estruturas de decisão.....	30
11- Estrutura CASO SELECIONE.....	47
12- Estruturas de repetição.....	53
13- Arrays.....	60

---

### Introdução

A Lógica de Programação é o principal fundamento utilizado pelos profissionais da área de informática na solução de problemas.

Podemos definir Lógica de Programação como uma técnica para ordenar pensamentos a fim de atingir um objetivo.

Em programação, estes pensamentos ordenados são denominados Algoritmos e representam uma sequência de comandos que devem obedecer às regras estabelecidas por uma Linguagem de Programação para que possam ser codificados.

Existem inúmeras linguagens de programa, como Java, Pascal, PHP, C, .Net, entre outras, cada qual com sua lista de comandos e regras específicas.

Cada comando representa uma ação que deve ser executada pelo computador segundo as definições da linguagem.

Um programa corretamente codificado, nada mais é do que a transcrição de um Algoritmo em um conjunto de comandos de uma determinada linguagem de programação que quando submetidos à execução em um computador atende/atinge um determinado objetivo.

A aplicação da Lógica de Programação no desenvolvimento de software tem como objetivo tornar estes Algoritmos mais eficientes robustos e de fácil codificação e manutenção.

Este assunto não é novo, vêm de longa data, desde o surgimento dos primeiros componentes eletrônicos/lógicos programáveis, e muita coisa foi feita desde então, mas vamos nos ater ao desenvolvimento da capacidade lógica, e exercitar habilidades na criação de algoritmos para solução de problemas específicos.

---

### Algoritmos

Os Algoritmos representam sequências de passos, que devem ser seguidos a fim de cumprir uma determinada tarefa.

Como na vida real, podemos resolver um mesmo problema de várias formas diferentes, no caso dos algoritmos é a mesma coisa. Podemos descrever diferentes algoritmos para atingir um mesmo objetivo. Como mencionado na introdução, aí que entra a Lógica de Programação, para chegarmos a melhor solução possível.

Podemos descrever praticamente tudo através de algoritmos, desde tarefas simples e rotineiras como os passos que executamos quando acordamos até o momento que saímos para a escola ou para o trabalho, até tarefas mais complexas como a configuração/instalação de uma determinada aplicação em um computador.

Os Algoritmos na área de programação representam uma sequência de comandos que devem ser executados para atingir um objetivo.

Podemos descrever um algoritmo de duas formas:

- Fluxograma
- Pseudocódigo

---

## Fluxograma

O Fluxograma é a representação gráfica de um algoritmo. Pode ser aplicado em qualquer situação onde se queira descrever uma sequência de passos de forma gráfica.

A representação gráfica é um recurso de fundamental importância no processo de desenvolvimento de software, pois facilita a visualização, entendimento e a divulgação do algoritmo pelo desenvolvedor, e por todos os demais envolvidos no processo, facilitando a interação e participação de todos.

Imagine o fluxograma como uma planta do código que será desenvolvido, agora imagine como seria a construção de uma casa sem a utilização de uma planta.

Em programação o fluxograma representa uma sequência lógica de comandos para se obter a solução de um problema.

Onde cada comando é representado por um símbolo e é acompanhado de um texto que complementa o seu significado.



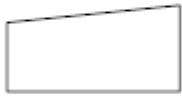

Símbolo	Função
 TERMINAL	Indica o INÍCIO ou FIM de um processamento Exemplo: Início do algoritmo
 PROCESSAMENTO	Processamento em geral Exemplo: Cálculo de dois números
 ENTRADA DE DADO MANUAL	Indica entrada de dados através do Teclado Exemplo: Digite a nota da prova 1
 EXIBIR	Mostra informações ou resultados Exemplo: Mostre o resultado do cálculo

Tabela 01

---

Vamos a um exemplo:

### Exemplo 01

Considere a seguinte situação:

Você precisa desenvolver uma aplicação que irá receber dois números digitados pelo usuário via teclado, deverá efetuar uma operação de adição entre os dois números e exibir o resultado em vídeo.

Este exemplo é bastante simples, mas contempla uma boa parte dos conceitos que deveremos dominar ao fim dos nossos estudos.

Antes de iniciarmos a criação do nosso Algoritmo para resolver o exemplo proposto, devemos entender o problema.

Precisamos identificar os passos necessários para chegar ao objetivo desejado.

Voltando ao exemplo proposto:

- 1- Precisamos desenvolver uma aplicação.
- 2- Essa aplicação vai ler dois números via teclado.
- 3- Somar os dois números.
- 4- Exibir o resultado em vídeo.

Como podemos ver acima, o exemplo proposto foi reescrito, mas em forma de tópicos. Esta técnica é importante para que possamos visualizar e entender de uma forma mais fácil, qual é o objetivo a ser atingido.

Além do que, observando os tópicos descritos, fica bem mais fácil de visualizar os passos necessários para desenvolver o algoritmo.

Entendido o problema, vamos passar agora para a criação do fluxograma.

---

### Fluxograma 01

Para isso vamos nos utilizar dos tópicos descritos acima.

- 1- Desenvolver uma aplicação. Por se tratar de uma solução de software o nosso trabalho sempre resultará em uma aplicação.

O primeiro símbolo que precisamos para descrever o nosso fluxograma é o terminal (ver tabela 01), que é utilizado para representar o início e o fim de um processamento.

Observe que além do símbolo, foi acrescentado em seu interior o texto Início, como já foi mencionado acima, a todo símbolo deve ser acrescentado em seu interior um complemento na forma de texto que explique o que este símbolo representa.



Figura 01

- 2- Ler dois números via teclado. Neste passo precisamos ler dois valores numéricos que serão digitados pelo usuário.

Para receber os dois números que serão digitados e possibilitar que os mesmos possam ser utilizados posteriormente pelo nosso algoritmo esses valores precisam ser armazenados em VARIÁVEIS.

Nos próximos capítulos entraremos em maiores detalhes sobre as variáveis, mas por hora, precisamos entender uma variável como um espaço na memória do computador que usamos para guardar valores.

Para criar uma variável precisamos definir um nome para a mesma.

É desejável que este nome nos lembre que tipo de variável estamos criando e para que ela foi criada.

---

Como precisamos ler dois valores, serão preciso duas variáveis:

- Numero01
- Numero02

Consultando novamente a tabela 01, podemos observar que existe um símbolo para entrada de dados via teclado, acrescentado a ele as variáveis que criamos teríamos então:

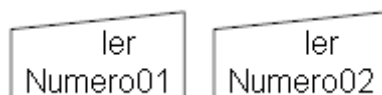


figura 02

- 3- Somar os números lidos. Em programação quando executamos uma ação que gera uma nova informação a partir de uma ou mais informações, estamos executando um processamento.

No nosso caso, estamos gerando uma nova informação a partir de duas outras. Quando efetuarmos a soma do valor armazenado em Numero01 com o valor de Numero02 estaremos criando uma nova informação, um novo valor, que também precisa ser armazenado em uma variável para que possamos utilizá-lo posteriormente. Daremos a essa variável o nome de Resultado.

Vejamos então como seria a representação desta operação matemática:

`Resultado <- Numero01 + Numero02`



Voltando a tabela 01 podemos observar que existe um símbolo para representar processamento, acrescentando a ele a nossa operação matemática teremos então:



```
Resultado = Numero01 + Numero02
```

figura 03

- 4- Exibir o resultado em vídeo. Agora que já lemos os dois números, somamos os seus valores e guardamos o resultado na variável Resultado, só falta exibir o seu valor.

Recorrendo novamente à tabela da figura 01, podemos constatar que existe um símbolo específico para a exibição de valores em vídeo. Como a informação que queremos exibir está armazenada na variável Resultado, teremos então:



```
Resultado
```

figura04

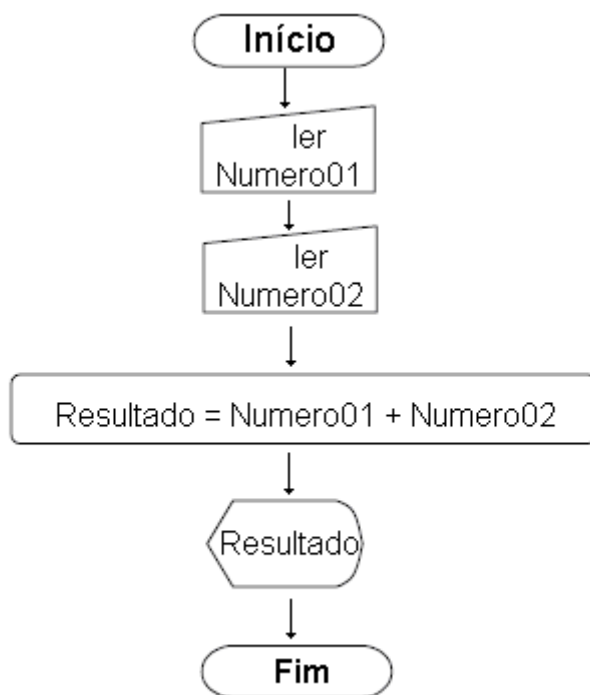
Para finalizar, acrescentamos um terminal com a descrição fim.



```
Fim
```

figura 05

Vejamos com ficou o fluxograma para o problema proposto.



Fluxograma 01

---

**Entendo o que foi feito.**

- 1- O primeiro passo foi entender o problema proposto, e dividir o problema em itens.
- 2- A partir dos itens, identificamos os passos a serem cumpridos.
- 3- E finalizamos, relacionando os passos ao símbolo correspondente.
- 4- Ao fim desses três passos temos o nosso fluxograma pronto.

Pronto, mas será que o meu algoritmo funciona?

Para responder a essa pergunta é muito simples, basta testarmos.

Para testar um algoritmo é muito simples, basta seguir os passos descritos pelo fluxograma, atribuir valores às variáveis de entrada, executar os processamentos e verificar se o resultado final é o esperado.

O nosso algoritmo lê duas variáveis, Numero01 e Numero02, vamos assumir que seus valores são 10 e 20 respectivamente.

Soma os seus valores  $10 + 20$ .

E apresenta o resultado da soma, no nosso caso 30.

Se seguirmos os passos descritos no fluxograma 01 e atribuirmos os valores acima, ao final será exibido o numero 30?

Sim. Isso significa que o nosso algoritmo funciona.

Este tipo de teste é chamado de Teste de Mesa. Onde são testadas várias possibilidades de combinações de valores de entrada e se verifica se os valores de saída correspondem ao resultado esperado.

---

Você pode estar pensando ...

Tudo isso para somar dois números?

O que estamos tentando transmitir aqui é um método para facilitar o desenvolvimento de algoritmos que possa ser aplicado em qualquer situação, tanto em problemas simples como o exemplo proposto como em problemas complexos.

Com o tempo, cada desenvolvedor molda sua própria forma de entender os problemas propostos e criar seus algoritmos, mas isso só vem com a prática.

E aproveitando, o “gancho”.

Não se aprende lógica lendo, copiando ou estudando algoritmos prontos, a única maneira de se aprender é praticando, através da repetição.

E para encerrar esse capítulo, é fundamental entender que não tem como desenvolver soluções de software profissionalmente, com qualidade, sem que seja feito um prévio estudo e elaborados os respectivos algoritmos.

Atualmente a ferramenta gráfica mais utilizada para descrever sistemas graficamente é a UML (Linguagem Universal de Modelagem).

Mas como o nosso foco é desenvolver a Lógica de Programação, podemos nos ater aos fluxogramas básicos e no Pseudocódigo.

Mas antes de passar para o próximo capítulo, vamos praticar?

---

### Exercícios – Lista 01

1. Elaborar um algoritmo que leia um número, some ao seu valor o número cinco e exiba o resultado da operação.
2. Descreva um algoritmo que leia um número, subtraia do seu valor o número três, multiplique por dois e exiba o resultado.
3. Crie um algoritmo que leia dois números, some o número dois aos valores lidos, e mostre os valores resultantes.
4. Elabore um algoritmo de leia um valor numérico, e some ao seu valor o número 10, leia um segundo valor, efetue a sua divisão por dois, e mostre os valores resultantes.
5. Crie um algoritmo que leia dois números, subtraia os seus valores (num1-num2) , some ao resultado o número 10 e mostre o valor da operação em vídeo.
6. Descreva um algoritmo que leia um número, subtraia do seu valor o número três, leia outro número e some ao seu valor o número dois, some os resultados das operações realizadas com os números lidos , subtraia do resultado o número 1 e mostre em vídeo o valor final da operação.
7. Criando uma tabuada. Crie um algoritmo que leia um número e mostre em como resultado a tabuada de multiplicação do número lido.
8. Elaborar um algoritmo que leia três números some 10 ao primeiro, subtraia cinco do segundo e multiplique o terceiro por dois, após efetuar estas operações mostre o resultado das operações.
9. Descrever um algoritmo que leia um número, subtraia do seu valor o número três, some o resultado ao número lido, e mostre o valor final da operação.
10. Crie um algoritmo que leia um número, divida o seu valor por dois e atribua o resultado da operação a duas variáveis diferentes, some a uma das variáveis o número três e a outra o número cinco, ao final, mostre o valor das duas variáveis.

---

Agora de acabamos de concluir a nossa primeira lista de exercícios, podemos estar pensando. Como que estes algoritmos com as operações matemáticas básicas vão me ajudar a aprender Lógica de Programação? Acredite, estamos apenas começando, se você tem em seu convívio alguma criança que esta aprendendo a dar seus primeiros passos, você sabe do que estou falando.

Tenha calma, passo a passo vamos nos desenvolver na arte da programação, e ao final desta apostila, após todos os exercícios resolvidos, você olhará para traz e terá suas respostas.

---

### Pseudocódigo

É a representação do algoritmos descritos com comandos textuais, que se aproxima das linguagens de programação, mas utilizam palavras da língua falada.

Têm como objetivo facilitar o entendimento do código escrito durante o processo de desenvolvimento e se caracterizam por ser um passo intermediário entre o fluxograma e o código propriamente dito.

A fim de exercitar o Pseudocódigo dos nossos algoritmos utilizaremos uma linguagem chamada VisuAlg.

---

## A Linguagem de Programação VisuAlg.

O VisuAlg é uma linguagem muito simples, que utiliza uma versão traduzida e adaptada dos pseudocódigos largamente utilizados nos livros de introdução à programação, conhecidos como "Portugol". Assim como toda linguagem, tem sua lista de comandos e regras de codificação. Como por exemplo, não utiliza acentos nem diferencia letras maiúsculas de minúsculas.

Neste momento vamos nos ater a uma breve apresentação dos recursos desta linguagem, apenas o mínimo necessário para transcrever os nossos fluxogramas, e no decorrer deste documento a cada novo recurso apresentado para os fluxogramas, apresentaremos o respectivo recurso no VisuAlg.

A estrutura básica de um pseudocódigo escrito em VisuAlg:

```
algoritmo "semnome"  
// Função :  
// Autor :  
// Data :  
// Seção de Declarações  
inicio  
// Seção de Comandos  
finalgoritmo
```

O código inicia pela palavra-reservada `algoritmo` seguido do seu nome delimitado por aspas duplas. Este nome será usado como título nas janelas de leitura de dados podendo ser utilizado para outros fins em futuras versões do VisuAlg.

A seção que se segue é a de declaração de variáveis, que termina com a linha que contém a palavra-reservada `inicio`. Deste ponto em diante está a seção de comandos, que continua até a linha em que se encontre a palavra-reservada `finalgoritmo`.

Esta última linha marca o final do pseudocódigo: todo texto existente a partir dela é ignorado pelo interpretador.

O VisuAlg permite e é desejável a inclusão de comentários. Qualquer texto precedido de `///  
"` é ignorado, até se atingir o final da sua linha. Por este motivo, os comentários não se estendem por mais de uma linha. Quando se deseja escrever comentários mais longos, que ocupem várias linhas, cada uma delas deverá começar por `///  
"`.

É muito bom, desde já criar o hábito de incluir comentários em seus códigos.



---

Facilmente o código de uma aplicação real passa das mil linhas. E com o passar do tempo até mesmo quem o escreveu, enfrentara dificuldades para lembrar o que e porque foi feito.

O ato de documentar o software desenvolvido vai muito além dos comentários em linha de código, mas já é um bom começo.

Antes de escrevermos o nosso primeiro pseudocódigo em VisuAlg precisamos ainda esclarecer alguns conceitos.

---

## Tipos de Dados

Quando definimos VARIÁVEL nos capítulos anteriores, apenas mencionamos que representam um espaço em memória para o armazenamento de uma informação.

Mas e se estas informações forem de tipos diferentes? Por exemplo, para armazenar um nome eu preciso reservar o mesmo espaço que para armazenar um número qualquer?

Por esse motivo, quando declaramos uma variável precisamos declarar também o seu tipo e em alguns casos e linguagens até o seu tamanho.

No VisuAlg temos quatro tipos de variáveis:

- Inteiro – define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais.
- Real – define variáveis numéricas do tipo real, ou seja, com casas decimais.
- Caractere – define variáveis do tipo string, ou seja, cadeias de caracteres.
- Lógico – define variáveis do tipo booleano, ou seja, com valor VERDADEIRO OU FALSO.

Além dos tipos primitivos descritos acima, é possível definir variáveis estruturadas do tipo VETOR, que serão apresentados oportunamente.

---

## Nomes de Variáveis e sua Declaração

Cada linguagem de programação tem suas regras para definição dos nomes das variáveis, no VisuAlg os nomes das variáveis devem começar por uma letra e depois conter letras, números ou *underline*, até um limite de 30 caracteres.

Vale lembrar que a definição dos nomes das variáveis devem lembrar o tipo de dado a ser armazenado, pois desta forma, você facilita o entendimento do código e a sua manutenção.

As variáveis podem ser simples ou estruturadas e não pode haver duas variáveis com o mesmo nome.

A seção de declaração de variáveis começa com a palavra-reservada *var*, e continua com as seguintes sintaxes:

*<lista-de-variáveis> : <tipo-de-dado>*

Na *<lista-de-variáveis>*, os nomes das variáveis estão separados por vírgulas.

Exemplos:

```
var a: inteiro  
Valor1, Valor2: real  
nome_do_aluno: caractere  
sinalizador: lógico
```

---

## Comando de Atribuição

Quando criamos uma variável esperamos que ela armazene algum valor. Para que essa variável assuma um valor específico podemos utilizar o comando de atribuição.

No VisuAlg quando precisamos, por exemplo que uma variável x assumo o valor 10 usamos a seguinte sintaxe:

```
X<-10
```

Que significa x recebe o valor 10.

Para atribuir uma string:

```
Nome <- "Jose Carlos"
```

E um valor booleano:

```
Teste <- FALSO
```

## Operadores

Para que possamos realizar operações matemáticas, lógicas ou de concatenação, o VisuAlg define os seguintes operadores :

### Aritméticos

$+, -$	Operadores unários, isto é, são aplicados a um único operando. São os operadores aritméticos de maior precedência. Exemplos: $-3$ , $+x$ . Enquanto o operador unário $-$ inverte o sinal do seu operando, o operador $+$ não altera o valor em nada o seu valor.
$/$	Operador de divisão inteira. Por exemplo, $5 / 2 = 2$ . Tem a mesma precedência do operador de divisão tradicional.
$+, -, *, /$	Operadores aritméticos tradicionais de adição, subtração, multiplicação e divisão. Por convenção, $*$ e $/$ têm precedência sobre $+$ e $-$ . Para modificar a ordem de avaliação das operações, é necessário usar parênteses como em qualquer expressão aritmética.
$\%$	Operador de módulo (isto é, resto da divisão inteira). Por exemplo, $8 \% 3 = 2$ . Tem a mesma precedência do operador de divisão tradicional.
$^$	Operador de potenciação. Por exemplo, $5 ^ 2 = 25$ . Tem a maior precedência entre os operadores aritméticos binários (aqueles que têm dois operandos).

### Caracteres

$+$	Operador de concatenação de <i>strings</i> (isto é, cadeias de caracteres), quando usado com dois valores (variáveis ou constantes) do tipo "caractere". Por exemplo: "Rio " + " de Janeiro" = "Rio de Janeiro".
-----	--

---

## Relacionais

<code>=, &lt;, &gt;, &lt;=, &gt;=, &lt;&gt;</code>	Respectivamente: igual, menor que, maior que, menor ou igual a, maior ou igual a, diferente de. São utilizados em expressões lógicas para se testar a relação entre dois valores do mesmo tipo. Exemplos: <code>3 = 3</code> (3 é igual a 3?) resulta em VERDADEIRO ; <code>"A" &gt; "B"</code> ("A" está depois de "B" na ordem alfabética?) resulta em FALSO.
--	---

## Lógicos

<code>nao</code>	Operador unário de negação. <code>nao VERDADEIRO = FALSO</code> , e <code>nao FALSO = VERDADEIRO</code> . Tem a maior precedência entre os operadores lógicos. Equivale ao NOT do Pascal.
<code>ou</code>	Operador que resulta VERDADEIRO quando um dos seus operandos lógicos for verdadeiro. Equivale ao OR do Pascal.
<code>e</code>	Operador que resulta VERDADEIRO somente se seus dois operandos lógicos forem verdadeiros. Equivale ao AND do Pascal.
<code>xou</code>	Operador que resulta VERDADEIRO se seus dois operandos lógicos forem diferentes, e FALSO se forem iguais. Equivale ao XOR do Pascal.

---

## Comandos de entrada e saída de Dados

Quanta informação, não é mesmo?

Mas para que possamos escrever nossos primeiros pseudocódigos em VisuAlg precisamos apresentar os comandos de entrada e saída de dados.

---

## Comando de entrada de dados

leia ( < lista-de-variáveis>)

Recebe valores digitados pelos usuário, atribuindo-os às variáveis cujos nomes estão em <lista-de-variáveis> (é respeitada a ordem especificada nesta lista). É análogo ao comando *read* do Pascal.



---

### Comando de saída de dados

escreva (<lista-de-expressões>)

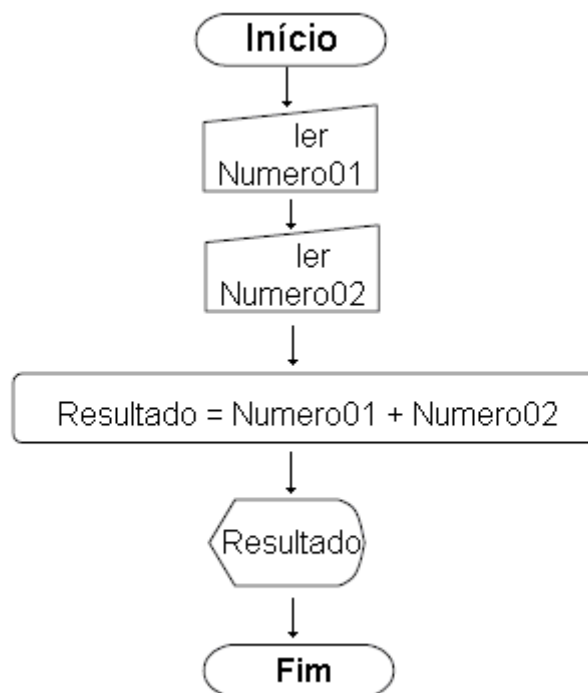
Escreve no dispositivo de saída padrão (isto é, na área à direita da metade inferior da tela do VisuAlg) o conteúdo de cada uma das expressões que compõem <lista-de-expressões>.

As expressões dentro desta lista devem estar separadas por vírgulas; depois de serem avaliadas, seus resultados são impressos na ordem indicada. É equivalente ao comando *write* do Pascal.

### Nosso Primeiro Pseudocódigo

Agora estamos prontos para escrever o nosso primeiro pseudocódigo.

Vamos relembrar o fluxograma criado para Exemplo 01:



Fluxograma 01

No Exemplo 01, foram lidos dois números, efetuada uma operação de adição entre eles e ao final foi exibido o resultado da operação.


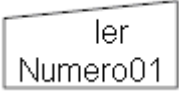
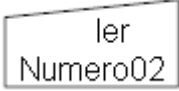
Vamos ver então como fica o pseudocódigo para esse fluxograma?

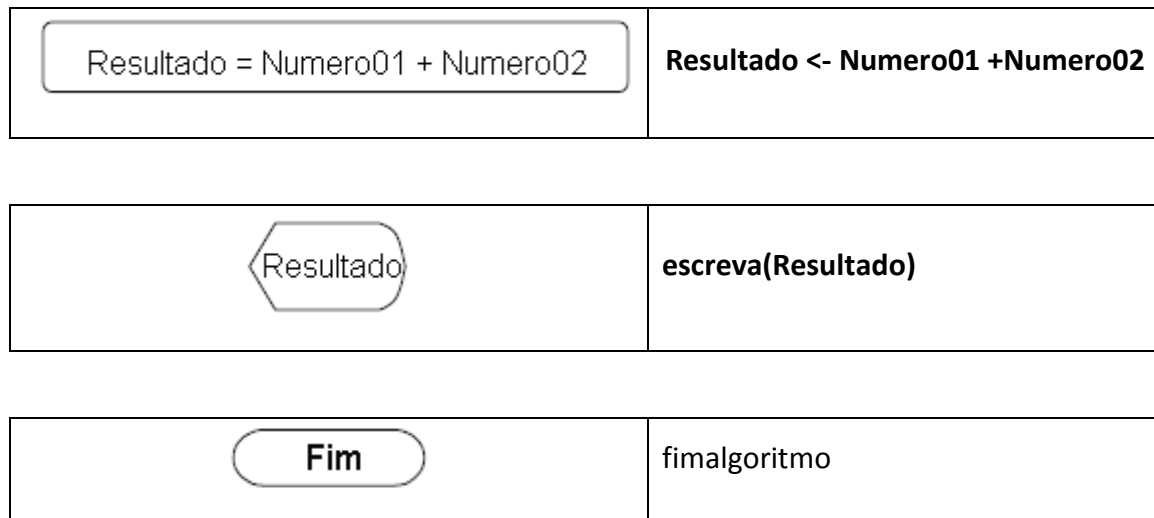
---

Relembrando a estrutura básica de um pseudocódigo no VisuAlg:

```
algoritmo "semnome"  
// Função :  
// Autor :  
// Data :  
// Seção de Declarações  
inicio  
    // Seção de Comandos  
finalgoritmo
```

Vamos às correspondências:

	<pre>algoritmo "exemplos01" // Função : lê dois números e mostra              o valor // Autor : Luiz Roberto Camilo // Data : 10/08/2011 // Seção de Declarações  var     Numero01: inteiro     Numero02: inteiro     Resultado: inteiro  inicio</pre>
	<pre>// Seção de Comandos  leia(Numero01)</pre>
	<pre>leia(Numero02)</pre>



Veja como ficou o pseudocódigo:

```
algoritmo "exemplo01"  
// Função : lê dois numero e mostra o valor  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
    Numero01: inteiro  
    Numero02: inteiro  
    Resultado: inteiro  
inicio  
    // Seção de Comandos  
    leia(Numero01)  
    leia(Numero02)  
    Resultado <- Numero01 +Numero02  
    escreva(Resultado)  
fimalgoritmo
```

---

### Entendendo o que foi feito

Da mesma forma como relacionamos cada tópico criado a partir da descrição do problema proposto a um símbolo do fluxograma, agora relacionamos cada símbolo a um comando do VisuAlg.

Note que o que foi acrescentado à estrutura básica do pseudocódigo está em destaque.

Que tal praticarmos o que foi aprendido?

---

**Exercícios – Lista 02**

- 1- Agora que aprendemos como escrever um pseudocódigo utilizando o VisuAlg, vamos aproveitar para praticar criando os respectivos pseudocódigos para os fluxogramas da lista de exercícios 01.

---

## Estruturas de Decisão

Invariavelmente quando formos propor um algoritmo para um problema, vamos nos deparar com momentos onde o próximo passo a ser tomado estará vinculado a uma tomada de decisão.

Mas o que vem a ser uma tomada de decisão?

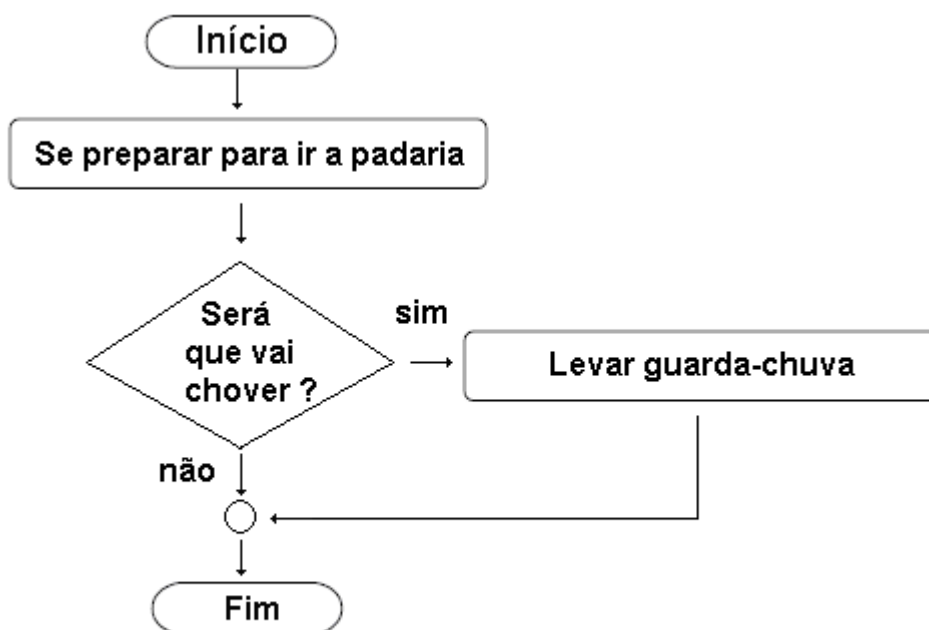
Uma tomada de decisão nada mais é do que uma condição. Onde o próximo passo depende do resultado da condição.

Vamos a um exemplo, considere a seguinte situação:

**Exemplo 02**

Você está se preparando para sair de casa para ir até a padaria que fica a duas quadras da sua casa. Mas o tempo não está muito bom. Então você se pergunta: Será que vai chover? Se for preciso levar o guarda-chuva.

Esta é uma situação hipotética, mas que exemplifica muito bem uma tomada de decisão. Veja o fluxo:



Fluxograma 02

O fluxograma 02 ilustra uma situação onde uma ação foi executada em função da resposta a uma pergunta.

Se a resposta para a pergunta for **SIM**, a ação **Levar o guarda-chuva** será executada, se a resposta for **NÃO**, a ação **Levar o guarda-chuva**, não será executada.

Houve uma mudança na ordem de execução das ações do fluxograma em função de uma tomada de decisão, por esse motivo as estruturas de decisão podem ser também chamadas de estruturas de desvio.



---

Em VisuAlg o comando correspondente se chama **SE ... ENTÃO**, e possui a seguinte estrutura:

```
se <expressão-lógica> entao
    <sequência-de-comandos>
fimse
```

Que significa:

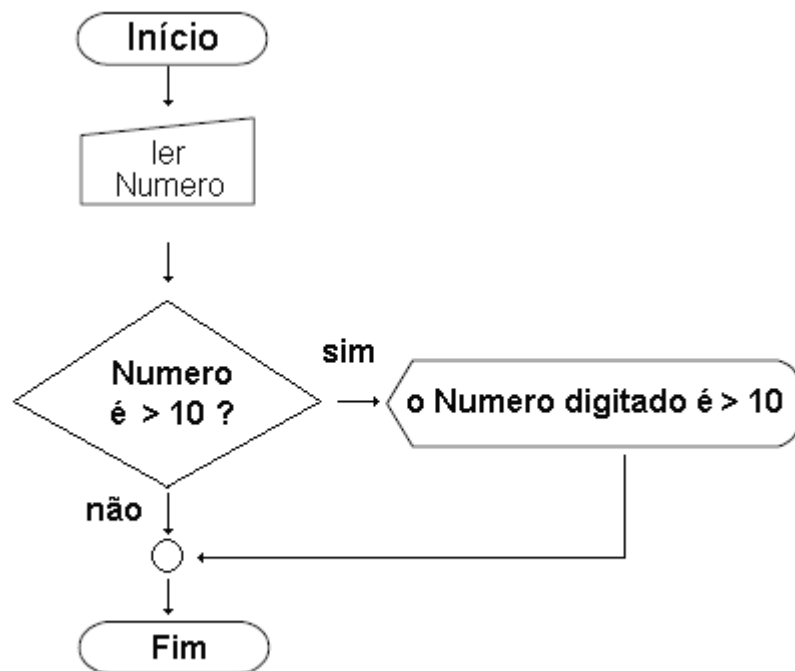
```
se < a expressão lógica for verdadeira> então
    execute a <sequência de comandos >
até encontrar um
fimse
```

Simples não é mesmo?

Vejamos outro exemplo:

**Exemplo 03**

Considere a seguinte situação, você precisa desenvolver um algoritmo que leia um número e exiba uma mensagem se ele for maior que 10. Veja o fluxograma para esse algoritmo.



Fluxograma 03

---

Vamos ver agora com fica o pseudocódigo:

```
algoritmo "exemplo03"  
// Função : lê um número e mostra uma mensagem se o valor for > 10  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
    Numero: inteiro  
inicio  
    // Seção de Comandos  
    leia(Numero)  
  
    se (Numero > 10) então  
        escreva("o numero digitado é maior que 10")  
    fimse  
  
finalgoritmo
```

Agora que estamos nos tornando praticamente especialistas em fluxograma e pseudocódigos, gostaria de chamar a atenção para mais uma dica de boas práticas.

A primeira para quem não se lembra foi sobre os comentários, agora gostaria de chamar a atenção para o que chamamos de **IDENTAÇÃO**.

Esta boa prática vem sendo aplicada em todos os códigos que escrevemos desde a apresentação do primeiro pseudocódigo.

---

O que você acharia se o código do Exemplo 03 fosse apresentado da seguinte forma:

```
algoritmo "exemplo03"  
// Função : lê um numero e mostra uma mensagem se o valor for > 10  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
Numero: inteiro  
inicio  
// Seção de Comandos  
leia(Numero)  
se (Numero > 10) então  
    escreva("o numero digitado é maior que 10")  
fimse  
finalgoritmo
```

Percebeu a diferença?

Neste caso específico, que estamos tratando de um código de poucas linhas, para alguns pode até parecer que não tem diferença alguma. Mas para quem sabe o que significa indentar o código e está acostumado a essa prática, faz muita diferença.

Indentar o código, significa utilizar os recursos de tabulação e espaços para destacar pontos específicos do código. Diferente do que acontece no código acima em que todas as linhas são iniciadas na mesma coluna.

Compare as duas apresentações do código do exemplo 03, qual é mais fácil de identificar onde começa e termina o bloco de comando, onde começa e termina o comando SE...ENTÃO ?

Vamos praticar?

---

**Exercícios – Lista 03**

1. Elabore um algoritmo (fluxograma e pseudocódigo) que leia um número e mostre o seu valor, se ele for menor que cinco.
2. Crie um algoritmo que leia um número, some ao seu valor cinco e mostre o resultado se ele for maior que dez.
3. Desenvolva um algoritmo que leia um número, subtraia de seu valor o número 10 e mostre o resultado se ele for maior que três.
4. Crie um algoritmo que leia três números some 5 aos seus valores e mostre os valores resultantes maiores que 10.
5. Elabore um algoritmo que leia dois números e mostre os seus valores multiplicados por 10 se a soma dos valores originais for menor que 20.
6. Desenvolva um algoritmo que leia um número e mostre o seu resultado multiplicado por 3, se o resultado for maior que 15.
7. Crie um algoritmo que leia três números, some seus valores e mostre o resultado se ele for maior que 20.
8. Elabore um algoritmo que leia dois números e mostre os seus valores se eles forem iguais.
9. Crie um algoritmo que leia um numero e mostre o seu valor dividido por 2, se o resultado for maior que 20.
10. Elabore um algoritmo que leia um número e mostre a tabuada do número lido se ele for menor que 10.

---

Os exemplo 02 e 03 apresentam situações onde uma ação é executada apenas quando a condição é satisfeita. Mas existem situações onde preciso que outra ação seja executada caso a condição não seja atendida.

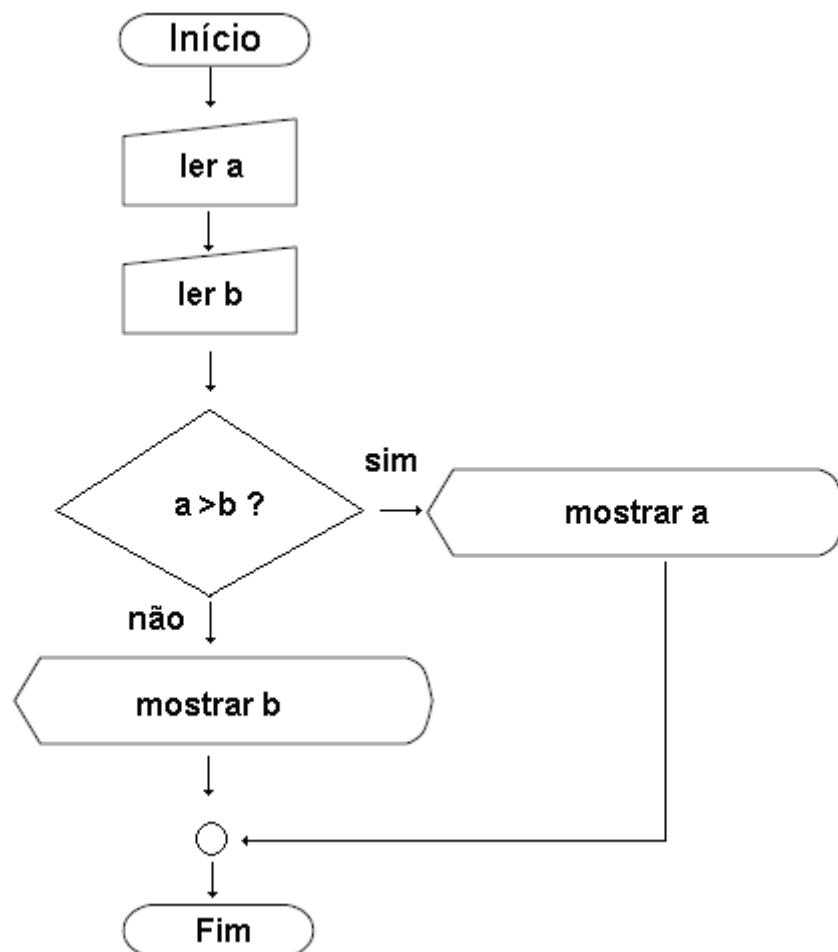
A esta estrutura de decisão damos o nome de **SE...ENTÃO...SENÃO**.

Nesta estrutura uma ação é tomada caso a condição seja atendida e outra caso não seja atendida.

Vamos a um a exemplo e tudo ficará mais claro.

**Exemplo 04**

Considere a seguinte situação, onde sua aplicação precisa ler dois números, A e B e deve mostrar na tela apenas o maior valor.



Fluxograma 04

Perceba que diferente da estrutura anterior, **SE...ENTÃO**, neste caso, após o teste da condição, ações diferentes são executadas tanto para o resultado verdadeiro quanto para o resultado falso .

---

Vamos agora ao pseudocódigo.

```
algoritmo "exemplo04"  
// Função : lê dois números e mostra o maior  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
    a : inteiro  
    b : inteiro  
  
inicio  
    // Seção de Comandos  
    leia(a)  
    leia(b)  
  
    se (a > b) então  
        escreva("a é maior")  
        escreva(a)  
    senao  
        escreva("b é maior")  
        escreva(b)  
    fimse  
  
fimalgoritmo
```

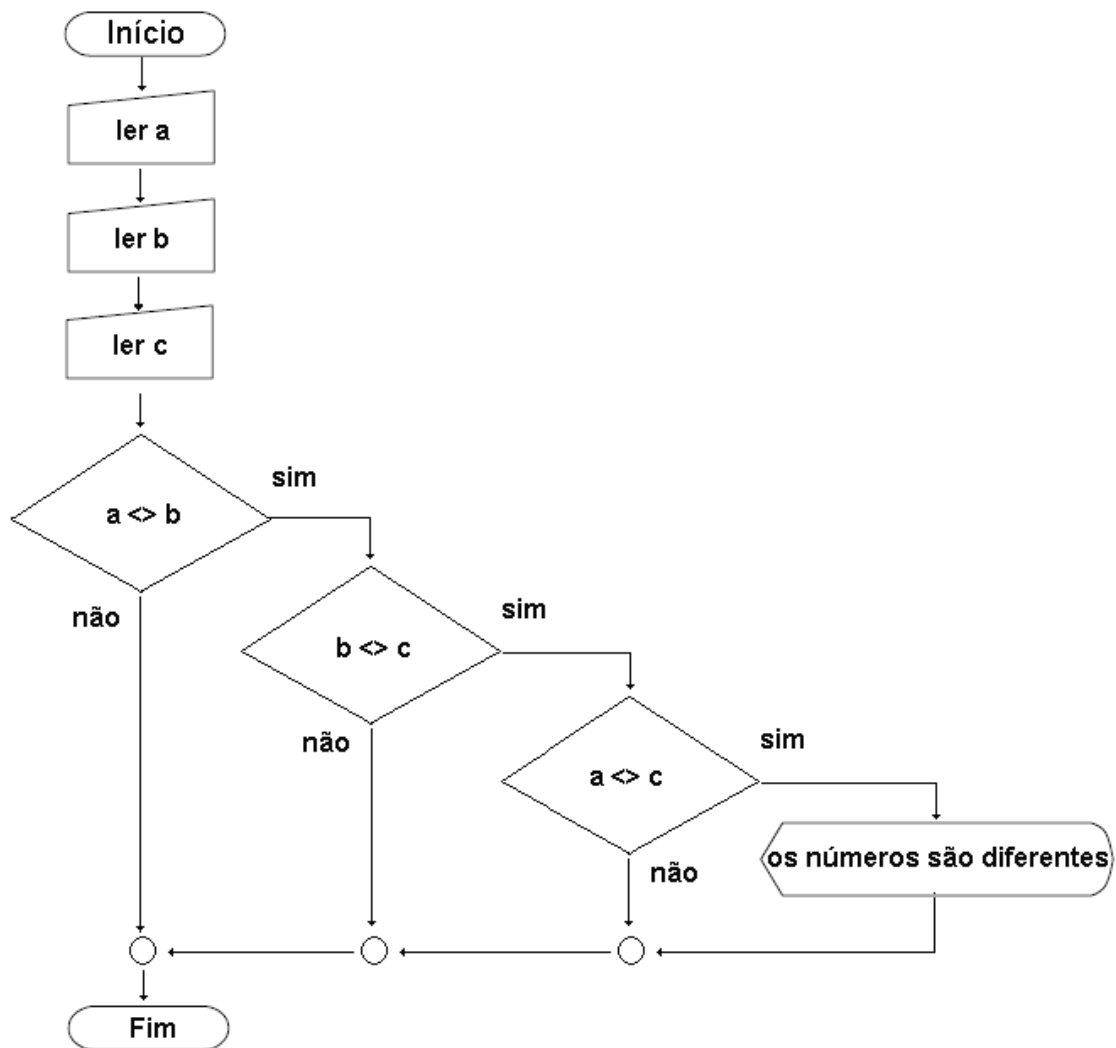
Existem ainda situações onde uma única condição pode não ser suficiente para realizar o teste exigido, neste caso podemos utilizar quantas condições forem necessárias.

Considere a seguinte situação, onde precisamos comparar três números para verificar se os seus valores são diferentes.



**Exemplo 05**

Leia três números e mostre a mensagem, números diferentes, caso a afirmação seja verdadeira.



Fluxograma 05

Vamos entender o que foi feito.

---

No exemplo 05 foram utilizadas três comparações para a solução do problema, primeiro foram lidos os três números, depois foi feita a primeira comparação, entre **a** e **b**, e se o resultado for negativo, **a=b** o algoritmo é finalizado, pois só nos interessa números diferentes. Em caso afirmativo o próximo teste foi entre **b** e **c**, da mesma forma o algoritmo foi finalizado em caso negativo. Neste ponto pode parecer que o problema já está resolvido e basta apenas informar a mensagem. Mas nós sabemos apenas que **a** é diferente de **b** e que **b** é diferente de **c**, isso não nos garante que **a** e **c** também sejam diferentes, por esse motivo a terceira comparação **a** com **c**, e só então exibimos ou não a mensagem.

---

Vejamos o código do exemplo 05

algoritmo "exemplo05"

// Função : lê três números e mostra a mensagem Números diferentes caso a afirmação seja verdadeira.

// Autor : Luiz Roberto Camilo

// Data : 10/06/2011

// Seção de Declarações

**var**

**a : inteiro**

**b : inteiro**

**c : inteiro**

**inicio**

// Seção de Comandos

**leia( a )**

**leia( b )**

**leia( c )**

**se (a <> b) então**

**se (b <> c) então**

**se (a <> c) então**

**escreva("Os números são diferentes")**

**fimse**

**fimse**

**fimse**

**fimalgoritmo**

Este exemplo é bastante simples, mas nos possibilita demonstrar o encadeamento de estruturas de decisão e principalmente a grande razão de se estudar Lógica de Programação.

Se observarmos com atenção o fluxograma 05, facilmente podemos perceber que ele não é a única forma de se chegar à solução do problema proposto.

Como já foi dito anteriormente, não existe apenas uma solução para o mesmo problema. E neste caso, com a utilização do operador lógico **e**, podemos simplificar significativamente o nosso fluxograma.

Quando tivemos o primeiro contato com os operadores lógicos disponíveis no VisuAlg, nos foi apresentado o operador **e**, que segundo sua definição retorna **VERDADEIRO** quando os dois operandos forem verdadeiros.

A	B	Resultado
F	F	F
F	V	F
V	F	F
V	V	V

Voltando para o exemplo 05, nossas comparações foram:

a <> b

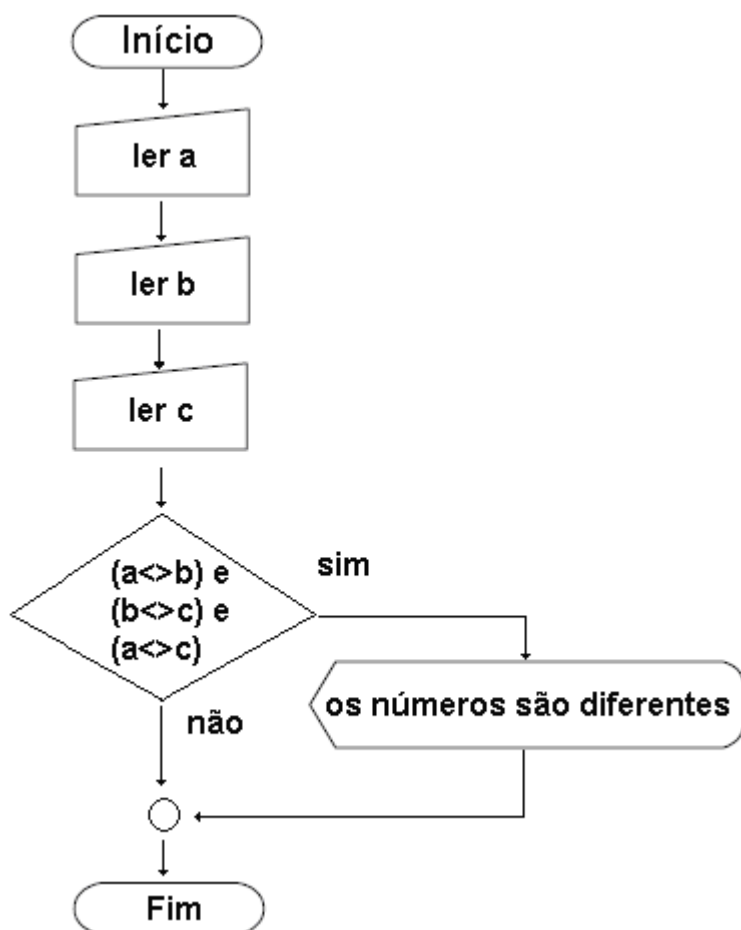
b <> c

c <> a

Utilizando o operador lógico **e** ficaria: (a <> b) **e** (b <> c) **e** (a <> c)

a <> b	b <> c	a <> c	Resultado
F	F	F	F
F	V	V	F
V	F	V	F
V	V	F	F
V	V	V	V

A partir da tabela verdade acima, o nosso fluxograma ficaria assim:



Fluxograma 06

---

Agora vamos ver como fica o código:

```
algoritmo "exemplo06"  
// Função: (exemplo05 modificado) lê três números e mostra a mensagem Números  
//           diferentes caso a afirmação seja verdadeira.  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
    a : inteiro  
    b : inteiro  
    c : inteiro  
inicio  
    // Seção de Comandos  
    leia( a )  
    leia( b )  
    leia( c )  
  
    se ( ( a <> b ) e ( b <> c ) e ( a <> c ) ) então  
        escreva("Os números são diferentes")  
    fimse  
  
finalgoritmo
```

Nada melhor que praticar para entender o conteúdo apresentado.  
Então vamos lá.

---

**Exercícios – Lista 04**

1. Desenvolva um algoritmo (fluxograma e pseudocódigo) que leia dois números e mostre o menor.
2. Elabore um algoritmo que leia dois números, some cinco ao de menor valor, compare os dois valores e mostre o maior.
3. Desenvolva um algoritmo que leia um número e mostre a seguinte mensagem, número maior que dez, se ele for maior que dez e menor igual a dez se ele for menor ou igual a dez.
4. Elabore um algoritmo que leia três números e mostre o maior
5. Elabore um algoritmo que leia três números some cinco ao menor valor e mostre o resultado.
6. Desenvolva um algoritmo que leia dois números e mostre os números em ordem crescente.
7. Crie um algoritmo que leia três números e mostre seus valores em ordem crescente.
8. Elabore um algoritmo que leia um número, se ele for maior que dez, some cinco ao seu valor, se for menor ou igual, some 20 e mostre se o resultado for maior que vinte e cinco.
9. Desenvolva um algoritmo que leia quatro números, some os dois primeiros e subtraia os dois últimos, some os resultados e mostre a seguinte mensagem, resultado maior que dez, caso a afirmação esteja correta ou resultado menor ou igual a dez.
10. Crie um algoritmo que leia dois números, multiplique o menor por 10, e divida o maior por 2, some os seus valores e verifique se o resultado é par, em caso afirmativo exiba a mensagem, o resultado é par, caso contrario, exiba a mensagem, o resultado é impar.

---

## Estrutura CASO SELECIONE

Em algumas situações precisamos executar ações diferentes para um determinado valor que pode ser assumido por uma mesma variável. Considere por exemplo uma aplicação que exiba o mês em formato texto a partir de um mês digitado em formato numérico.

### Exemplo 07





Fluxograma 07

---

O VisuAlg implementa (com certas variações) o comando *case* do Pascal. A sintaxe é a seguinte:

```
escolha <expressão-de-seleção>
    caso <exp11>, <exp12>, ..., <exp1n>
        <seqüência-de-comandos-1>
    caso <exp21>, <exp22>, ..., <exp2n>
        <seqüência-de-comandos-2>
    ...
Outrocaso
    <seqüência-de-comandos-extra>
fimescolha
```

---

E agora o código:

algoritmo "exemplo07"

// Função: **Mostra o mês em formato texto a partir de mês digitado em formato Numérico.**

// Autor : **Luiz Roberto Camilo**

// Data : **10/06/2011**

// Seção de Declarações

**var**

**m : inteiro**

inicio

// Seção de Comandos

**leia( m )**

**escolha m**

**caso 1**

**escreval ("Janeiro.")**

**caso 2**

**escreval ("Fevereiro.")**

**caso 3**

**escreval ("Março.")**

**caso 4**

**escreval ("Abril.")**

**caso 5**

**escreval ("Maio.")**

**caso 6**

**escreval ("Junho.")**

**caso 7**

**escreval ("Julho.")**

**caso 8**

**escreval ("Agosto.")**

**caso 9**

**escreval ("Setembro.")**

**caso 10**

**escreval ("Outubro.")**

**caso 11**

**escreval ("Novembro.")**

**caso 12**

---

```
        escreval ("Dezembro.")
    outrocaso
        escreval ("Mês inválido.")
    fimescolha
finalgoritmo
```

Agora é a sua vez.

Vamos praticar?

---

**Exercícios – Lista 05**

1. Elabore um algoritmo que receba um número e escreva na tela o dia da semana corresponde, considerando que o domingo é o primeiro dia da semana e corresponde ao numero um.

---

## Estruturas de Repetição

Eventualmente observamos que determinados trechos do algoritmo é repetido um determinado número de vezes ou até que uma condição seja atendida.

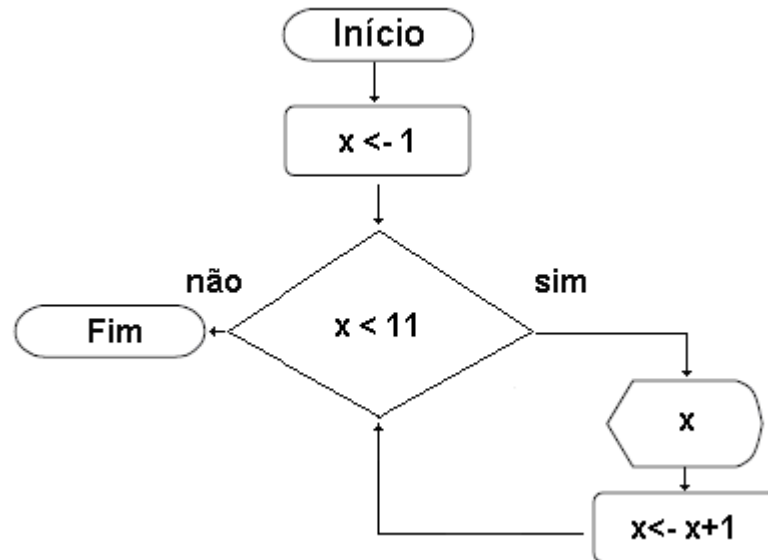
Para que este trecho de código não tenha que ser escrito N vezes e facilitar o seu entendimento no algoritmo, foram criadas as estruturas de repetição.

Existem três tipos básicos de laços:

- para...até...faca ( laço for )
- enquanto...faca ( laço while )
- repita...até ( laço repeat until)

Cada qual com suas características que os tornam mais adequados para determinada situação.

No exemplo a seguir, os números de 1 a 10 são exibidos em ordem crescente.

**Exemplo 08**

Fluxograma 08

Agora veja como fica o código em VisuAlg para cada um dos tipos de estruturas de decisão.

---

**para...ate...faca**

Esta estrutura repete uma seqüência de comandos um determinado número de vezes.

```
para <variável> de <valor-inicial> ate <valor-limite> [passo <incremento>] faca  
    <seqüência-de-comandos>  
fimpara
```

```
algoritmo "exemplo08"  
// Função: "Números de 1 a 10"  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
  
    j: inteiro  
  
inicio  
  
    // Seção de Comandos  
    para j de 1 ate 10 faca  
        escreva (j:3)  
    fimpara  
  
finalgoritmo
```

Se, logo no início da primeira repetição, <valor-inicial > for maior que <valor-limite > (ou menor, quando <incremento> for negativo), o laço não será executado nenhuma vez. O exemplo a seguir não imprime nada.



---

```
algoritmo "exemplo08"  
// Função: "Números de 1 a 10(não funciona)"  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
    j: inteiro  
inicio  
    // Seção de Comandos  
    para j de 10 ate 1 faca  
        escreva (j:3)  
    fimpara  
finalgoritmo
```

Este outro exemplo, no entanto, funcionará por causa do **passo -1**:

```
algoritmo "exemplo08"  
// Função: "Números de 1 a 10"  
// Autor : Luiz Roberto Camilo  
// Data : 10/06/2011  
// Seção de Declarações  
var  
    j: inteiro  
inicio  
    // Seção de Comandos  
    para j de 10 ate 1 passo -1 faca  
        escreva (j:3)  
    fimpara  
finalgoritmo
```

---

**enquanto...faca**

Esta estrutura repete uma seqüência de comandos enquanto uma determinada condição (especificada através de uma expressão lógica) for satisfeita.

```
enquanto <expressão-lógica> faca
    <seqüência-de-comandos>
fimenquanto
```

O mesmo exemplo anterior pode ser resolvido com esta estrutura de repetição:

```
algoritmo "exemplo08"
// Função: "Números de 1 a 10(com enquanto...faca)"
// Autor : Luiz Roberto Camilo
// Data : 10/06/2011
// Seção de Declarações
var
    j: inteiro
inicio
    // Seção de Comandos
    j <- 1
    enquanto j <= 10 faca
        escreva (j:3)
        j <- j + 1
    fimenquanto
finalgoritmo
```

Como o laço enquanto...faca testa sua condição de parada **antes** de executar sua seqüência de comandos, esta seqüência poderá ser executada **zero ou mais vezes**.

---

### repita ...até

Esta estrutura repete uma sequência de comandos até que uma determinada condição (especificada através de uma expressão lógica) seja satisfeita.

```
repita
    <seqüência-de-comandos>
ate <expressão-lógica>
```

Considerando ainda o mesmo exemplo:

```
algoritmo "exemplo08"
// Função: "Números de 1 a 10"
// Autor : Luiz Roberto Camilo
// Data : 10/06/2011
// Seção de Declarações
var
    j: inteiro
inicio
    // Seção de Comandos
    j <- 1
    repita
        escreva (j:3)
        j <- j + 1
    ate j > 10
fimalgoritmo
```

Como o laço repita...ate testa sua condição de parada **depois** de executar sua sequência de comandos, esta sequência poderá ser executada **uma ou mais vezes**.

É hora de praticar.

---

**Exercícios – Lista 06**

1. Elabore um algoritmo que exiba na tela os números de 1 a 20.
2. Desenvolva um algoritmo que leia um número e mostre na tela os números entre o número lido e 20.
3. Crie um algoritmo que leia dois números e mostre os números entre eles.
4. Elabore um algoritmo que leia um número e mostre a tabuada do número lido, utilizando uma estrutura de repetição.
5. Desenvolva um algoritmo que leia três números e mostre na tela os números entre os dois maiores.
6. Crie um algoritmo que leia um número menor que 10 e mostre o seu valor.
7. Elabore um algoritmo que leia dois números, um maior que dez e outro menor que 5, mostre os números lidos.
8. Desenvolva um algoritmo que leia um número menor que cinco e mostre os números pares entre o número lido e o número 20.
9. Crie um algoritmo que leia um número maior que 20 e mostre na tela os números entre o número digitado e o número 1 em ordem decrescente.
10. Elabore um algoritmo que leia um número entre cinco e dez e mostre o seu valor na tela.

---

## Array

Um array pode ter dimensão, assim denominado **VETOR**, ou mais de uma dimensão, **MATRIZ**.

Quando definimos variável, mencionamos que uma variável corresponde a um espaço em memória para o armazenamento de uma informação. Podemos definir um array como uma variável em memória, onde o espaço reservado foi dividido em vários pedaços, onde cada pedaço é identificado através de um número, referente à posição de uma determinada informação no array em questão. O número de cada posição do array é chamado de **índice**.

## Vetor

O vetor é um array de uma dimensão.

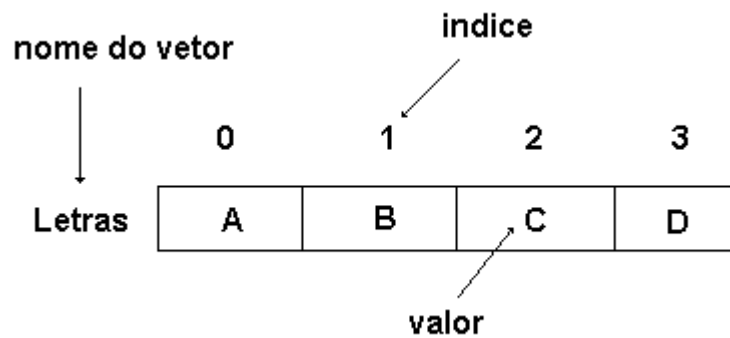


Figura 02

A figura 02 apresenta um vetor de quatro posições chamado Letras.

O acesso às informações armazenadas no vetor é feita através da combinação do nome mais o índice da posição desejada, por exemplo:

Letras[1] corresponde ao valor B e

Letras[3] corresponde ao valor D

Sua definição no VisuAlg seria:

Letras: vetor [0..3] de caractere

É importante lembrar que em algumas linguagens os arrays começam a partir do índice 0 e em outras com o índice 1. Para maiores detalhes consulte o manual da respectiva linguagem.

## Matriz

A matriz é um array com mais de uma dimensão.

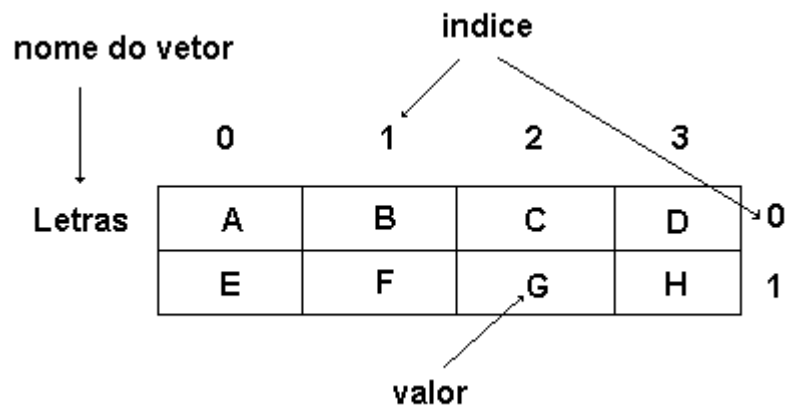


Figura 03

Semelhante aos vetores, as matrizes se diferenciam por apresentar duas ou mais dimensões e para acessar aos dados armazenados é necessário a utilização de mais de um índice, de acordo com o número de dimensões da matriz.

O acesso a um dado na matriz da figura 02 seria assim:

Letras[1,0] que corresponde ao dado B e  
Letras[2,1] que corresponde ao dado G

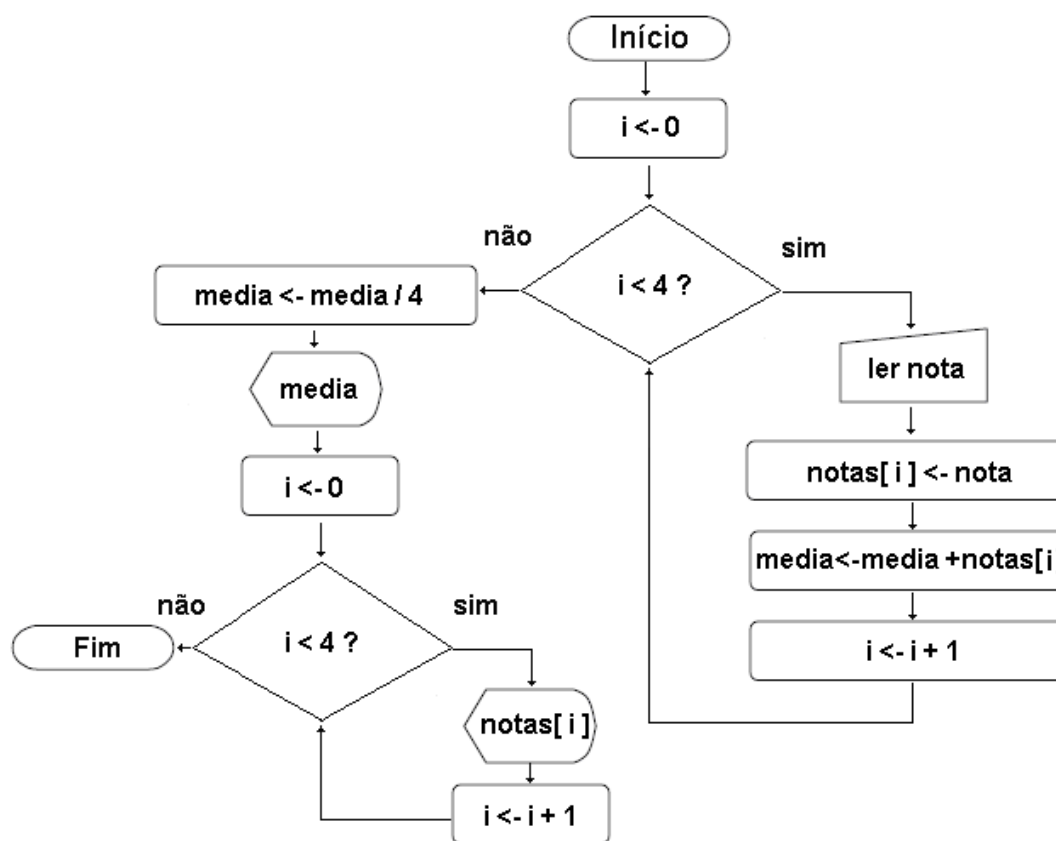
Em VisuAlg definiríamos como:

Letras: vetor [0..3,0..1] de caractere

**Exemplo 09**

Considere que você precise desenvolver um algoritmo que leia quatro notas de um aluno, efetue o cálculo da média e mostre o resultado e as notas lidas.

Veja o fluxograma:



Fluxograma 09

Este fluxograma, como todos os outros exemplos apresentados podem ser construídos de N formas diferentes. No exemplo 09 optamos por fazer desta forma, apesar de não ser a mais simplificada para demonstrar que os dados lidos no início do fluxograma continuam disponíveis no final, quando foram exibidos, pois foram armazenados no vetor notas.

Vamos ver como ficou o código?



---

```
algoritmo "exemplo09"
// Função: "ler quatro notas de um aluno, calcular e mostrar a média e as notas lidas"
// Autor : Luiz Roberto Camilo
// Data : 10/06/2011
// Seção de Declarações
var
    i: inteiro
    nota: real
    media: real
    notas: vetor[0..3]
inicio
    // Seção de Comandos
    //este loop recebe os dados
    para i de 0 ate 4 faca
        leia(nota)
        notas[i]<-nota
        media<- media + nota[i]
    fimpara

    media<-media/4
    escreve(media)

    // este loop mostra os dados
    para i de 0 ate 4 faca
        escreve(notas[i])
    fimpara
finalgoritmo
```

Vamos praticar?

---

**Exercícios – Lista 07**

1. Elaborar um algoritmo para o exemplo 09 que utilize apenas uma estrutura de repetição.
2. Desenvolva um algoritmo que leia cinco números e armazene um vetor, após receber os números ordene os dados em ordem crescente e mostre os valores.
3. Criar um algoritmo que leia quatro notas de cinco alunos e calcule e mostre as médias e as notas dos alunos.