

Curso

Algoritmos & Lógica de Programação

Uma abordagem com VisualG

Prof. José Cintra

<http://www.josecintra.com/blog>

GitHub

Apresentação

Esta apostila está dividida em 4 temporadas, abrangendo os principais tópicos exigidos por essa disciplina.

Para aprofundar os estudos, consulte a bibliografia no final da apostila e, não se esqueça, **pratique!**

- Resolva os exercícios propostos
- Faça download dos exercícios resolvidos no [GitHub](#)

Temporada 1



Conceitos Básicos

Algoritmo
=

Sequência lógica e não ambígua
de instruções que leva à
solução de um problema
em um tempo finito.

Algoritmos – Características básicas

- **Sequência lógica**
 - As instruções devem ser definidas em uma ordem correta.
- **Não ambígua**
 - A seqüência lógica e as instruções não devem dar margem à dupla interpretação.
- **Solução de um problema**
 - A seqüência lógica deve resolver exatamente o problema identificado.
- **Tempo finito**
 - A seqüência lógica não deve possuir iterações infinitas.

Algoritmos - Exemplo

Algoritmo: Sacar dinheiro

INÍCIO

1. Ir até o caixa eletrônico.
2. Colocar o cartão.
3. Digitar a senha.
4. Solicitar o saldo.
5. Se o saldo for maior ou igual à quantia desejada, sacar a quantia desejada; caso contrário sacar o valor do saldo.
6. Retirar dinheiro e cartão.

FIM.

Algoritmos Computacionais

Um algoritmo não representa, necessariamente, um programa de computador e sim os passos necessários para realizar uma tarefa.

Portanto, sua implementação pode ser feita por um computador, por um robô ou mesmo por um ser humano. Diferentes algoritmos podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros.

<https://pt.wikipedia.org/wiki/Algoritmo>

Pseudocódigo

Um algoritmo, para ser entendido e executado por um computador, precisa seguir um padrão rigoroso e um “vocabulário” bem definido.

Para isso utilizaremos uma linguagem que é chamada de pseudocódigo, também conhecida como português estruturado ou portugol.

Vamos escrever sequências de instruções que possam ser entendidas por qualquer programador, independente da linguagem de programação utilizada.

Algoritmos com VisualG

Existem diversas versões de linguagens que são usadas para construção de algoritmos computacionais voltadas para fins educacionais.

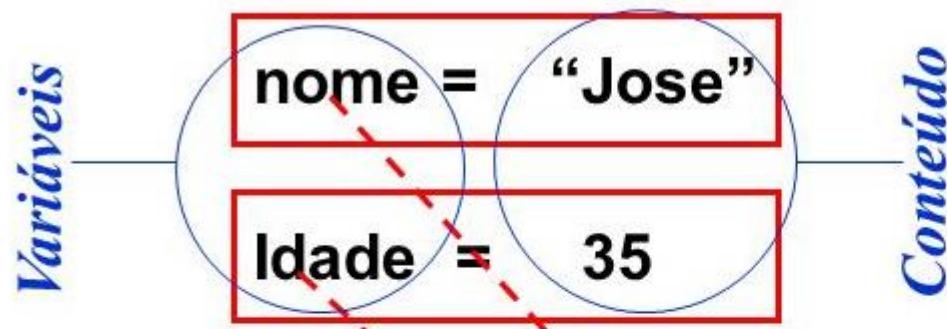
A estrutura da linguagem aqui utilizada está baseada no interpretador de algoritmos VisualG desenvolvido no Brasil.

A vantagem de aprender a programar usando uma **pseudolínguagem** é que a tradução do algoritmo para qualquer outra linguagem de programação “de verdade” é imediata, pois todas as linguagens imperativas seguem os mesmos conceitos aqui apresentados.

Variáveis e Tipos de Dados

- Algoritmos trabalham com **dados** (valores)
- Eses valores precisam ser armazenados pelo computador para serem processados;
- Uma **variável** representa uma posição na memória, onde pode ser armazenado um dado de um determinado tipo;
- Uma variável possui um **nome** e um valor de um determinado **tipo de dado** para que possa ser identificada.
- Além disso, um tipo de dado define uma série de **operações** que podem ser realizados com seus valores
- Durante a execução do algoritmo, a variável pode ter seu valor alterado (**seu valor pode variar**)
- Mudanças no valor das variáveis:
 - Por entrada de dados (digitação)
 - Por atribuição de valor direto no código

Variáveis



Memória Principal

0000
.....
0100	Jose
0101	35
.....
127Mb

- Uma variável é um espaço reservado na memória para armazenar um tipo de dado

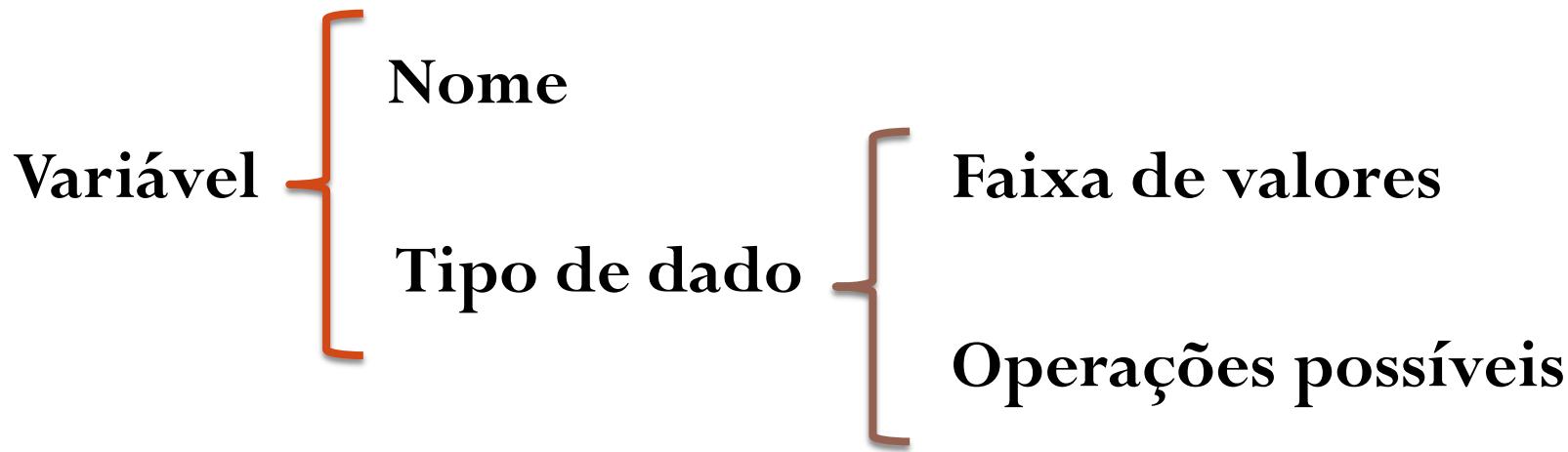
Tipos de Dados

TIPO	DESCRIÇÃO
Inteiro	Representa valores inteiros Ex.: 2, 5, -3, 100, -151...
Real	Representa valores reais Ex.: 10.0, 5.6, -3.45....
Caractere	Representa texto entre aspas duplas Ex.: "Corinthians", "B", "1234"
Logico	Representa os valores lógicos VERDADEIRO ou FALSO

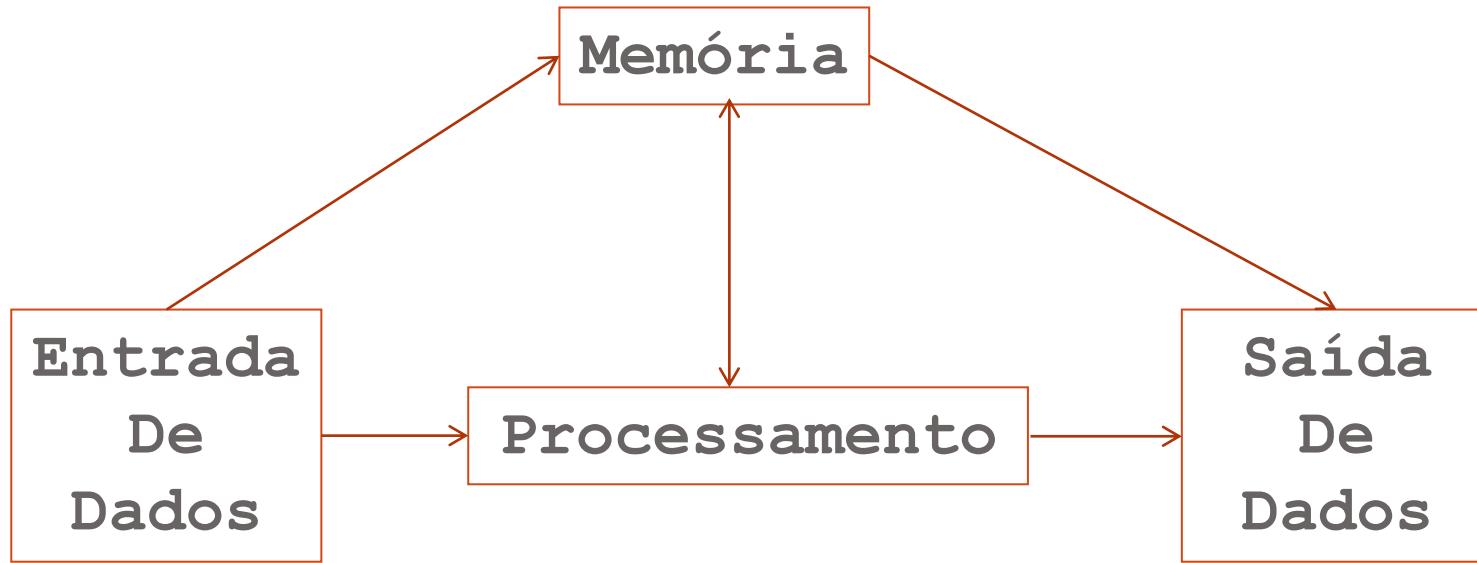
Tipos de Dados

Toda variável é associada a um tipo de dado.

O tipo de dado de uma variável identifica a **faixa de valores** que ela poderá receber, bem como as **operações** que poderão ser realizadas com essa variável.



Algoritmos – ciclo básico



Os dados da variáveis são armazenados na memória e depois recuperados para serem processados ou impressos

Forma geral de um algoritmo

Algoritmo "NomeDoAlgoritmo"

var

→ Aqui são declaradas os dados, as informações a serem utilizadas (variáveis e constantes)

inicio

→ Aqui inserimos as instruções a serem seguidas, ou seja o algoritmo propriamente dito

fimalgoritmo

Forma geral de um algoritmo

- As palavras ou comandos usados na linguagem são padronizadas e são chamadas de **palavras reservadas**.
- As palavras **algoritmo** e **fimalgoritmo** delimitam o início e o fim de um algoritmo;
- Após a palavra **var**, descreveremos os dados que serão utilizados pelo algoritmo;
- Entre as palavras reservadas **inicio** e **fimalgoritmo** definimos a sequência de comandos;

Declaração de variáveis

Antes de usarmos uma variável, precisamos declará-la, ou seja, definir o seu nome e tipo.

Declaramos variáveis logo após a palavra reservada var.

Exemplo:

```
var  
idade: inteiro  
nota1, nota2: real
```

Declaração de variáveis

Regras para identificação das variáveis em **Visualg**:

- Nomes de variáveis, também conhecidos como **identificadores**, não podem ser iguais a palavras reservadas;
- Nomes de variáveis devem possuir como primeiro caractere uma letra ou sublinhado. Os demais caracteres podem ser letras, números ou sublinhado;
- Nomes devem ter no máximo 127 caracteres;
- Nomes não podem ter espaços em branco;
- Não há diferença entre letras maiúsculas e minúsculas;

Declaração de variáveis

Exemplos:

Identificadores válidos:

`nome`, `telefone`, `endereco`, `idade_filho`

Identificadores inválidos:

`3endereco`, `algoritmo`, `Nome completo`

Não pode
começar
com número

Não pode
ser palavra
reservada

Não pode conter
espaços.
Nesse caso, use o
“underscore”

Declaração de variáveis

Exemplo:

algoritmo “dados”

var

nome: caractere

idade: inteiro

altura: real

nota_1, nota_2: real

inicio

fimalgoritmo

Utilizamos o separador “:” para distinguir o nome da variável e seu tipo.

Variáveis do mesmo tipo podem ser declaradas em conjunto

Operação de atribuição

A operação de **atribuição** consiste em armazenar um valor fixo em uma variável via código;

Para isso, utilizamos o **operador de atribuição** representado por uma seta apontando para a esquerda:

<-

Só podemos atribuir valores do mesmo tipo da variável.

Operador de atribuição

Exemplo:

algoritmo “exemplo”

var

nome: caractere

idade: inteiro

inicio

nome <- “José da Silva”

idade <- 27

fimalgoritmo

Operador de atribuição

Exemplo de erros de atribuição:

algoritmo "exemplo"

var

nome: caractere

idade: inteiro

inicio

nome <- José da Silva

idade <- 27.34

fimalgoritmo

Valores do tipo
caractere devem estar
entre aspas

Tipo inteiro não
pode receber
valores decimais

Entrada de Dados

Nem sempre queremos armazenar um valor fixo na variável através do comando de atribuição. Muitas vezes é necessário solicitarmos um valor ao usuário do algoritmo. Esse valor, na maior parte das vezes será informado através do dispositivo padrão de entrada de dados, que é o teclado.

Dessa forma o algoritmo fica flexível e útil, pois esse valor pode variar a cada execução do programa. Para isso existe o **comando de leitura** que veremos a seguir.



Entrada de Dados

A operação de leitura é usada para obter um valor digitado pelo usuário

Sintaxe → leia (<lista de variáveis>)

Exemplo:

leia (numero)

O valor digitado será armazenado na variável numero.

Saída de Dados

O comando de impressão é utilizado para “imprimir” valores no dispositivo de saída de dados, normalmente, o monitor.

Sintaxe: **escreva (<lista-de-variáveis>)**

Exemplos:

escreva (numero1)

→ Será mostrado na tela o conteúdo da variável numero1.

escreva ("O texto digitado foi ", k)

→ Será mostrado o texto entre “” e depois o conteúdo da variável k.

Operações matemáticas

Para realizarmos operações matemáticas, usamos operadores um pouco diferentes do que estamos acostumados. Vejamos:

Multiplicação: *

Divisão inteira: \

Divisão real: /

Adição: +

Subtração: -

Potência: ^

Além disso, as chaves e colchetes devem ser substituídos todos por parênteses

Expressões matemáticas

Para a construção de algoritmos que realizam cálculo matemáticos, todas as expressões aritméticas devem ser linearizadas;

Devendo também ser feito o mapeamento dos operadores da aritmética tradicional para os do Português Estruturado.

$\left\{ \left[\frac{2}{3} - (5 - 3) \right] + 1 \right\} \cdot 5$	$((2/3 - (5 - 3)) + 1) * 5$
Tradicional	Computacional

Expressões matemáticas

<-	Atribuição. x <- 2. A variável x recebeu o valor 2. Logo x = 2
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
a\b	Retorna o quociente da divisão inteira de a por b
a%b	Retorna o resto da divisão inteira de a por b
a^b	Retorna o valor de a elevado a b
a^1/b	Retorna a raiz b de a
aleatorio (a)	Retorna um número aleatório, em intervalo fechado, entre 0 e a

Exemplos:

Hierarquia	Operação
1	Parênteses
2	Função
3	-, + (unários)
4	^
5	*, /, \, %
6	+, -

$$3/4+5 = 5.75 \qquad \qquad 3/(4+5) = 0.33333333$$

$$3\backslash 2 * 9 = 9 \qquad \qquad 11 \% 3 ^ 2 = 2$$

$$11 \% (3 ^ 2) = 2 \qquad \qquad (11 \% 3) ^ 2 = 4$$

$$3\backslash 2 + (65-40)^{(1/2)} = 6$$

Comentários

O **Visualg** permite 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 "://".

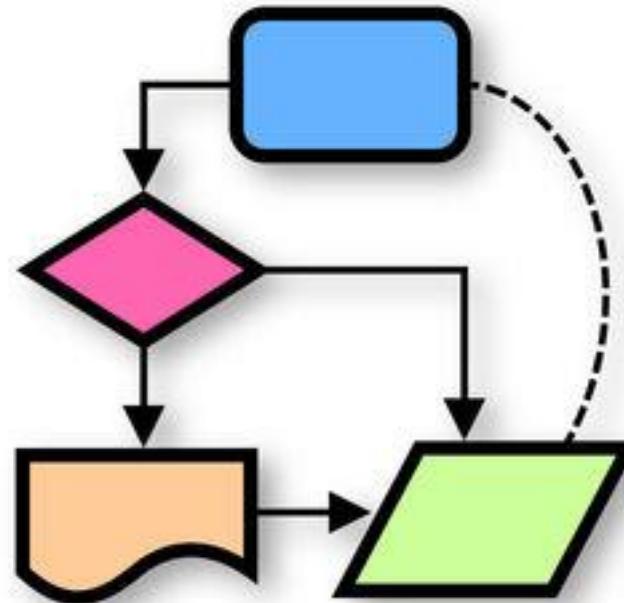
Exemplo de Algoritmo

```
algoritmo "calcula_media"  
    // Objetivo: Calcular a média aritmética de 2 números  
    // informados pelo usuário  
var  
    numero1,numero2 : real  
    media : real  
inicio  
    // Entrada de dados  
    escreva("Digite o primeiro número: ")  
    leia(numero1)  
    escreva("Digite o segundo número: ")  
    leia(numero2)  
    //Cálculo e impressão da média  
    media <- (numero1 + numero2) / 2  
    escreva("Média = ", media)  
fimalgoritmo
```

No comando LEIA, a execução do programa para e espera a digitação do valor que será armazenada na variável entre parênteses

O resultado do cálculo é armazenado na variável MEDIA através do comando de atribuição

Temporada 2



Estruturas de Controle

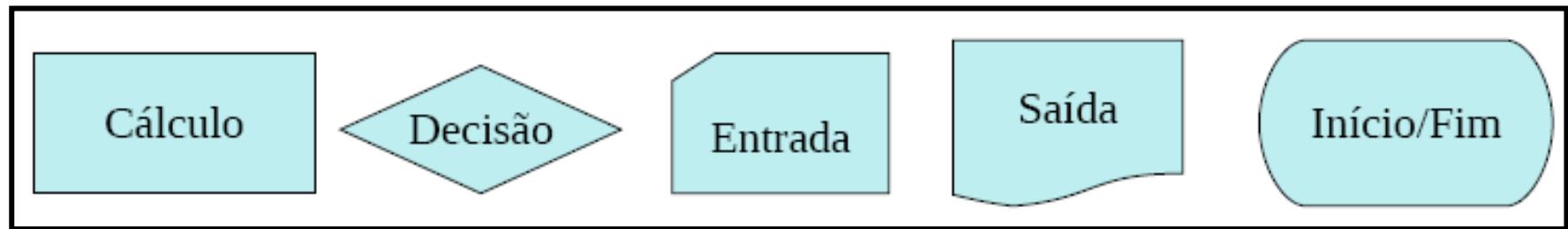
Fluxogramas

- Fluxograma: Representação gráfica da solução algorítmica de um problema por meio de símbolos geométricos



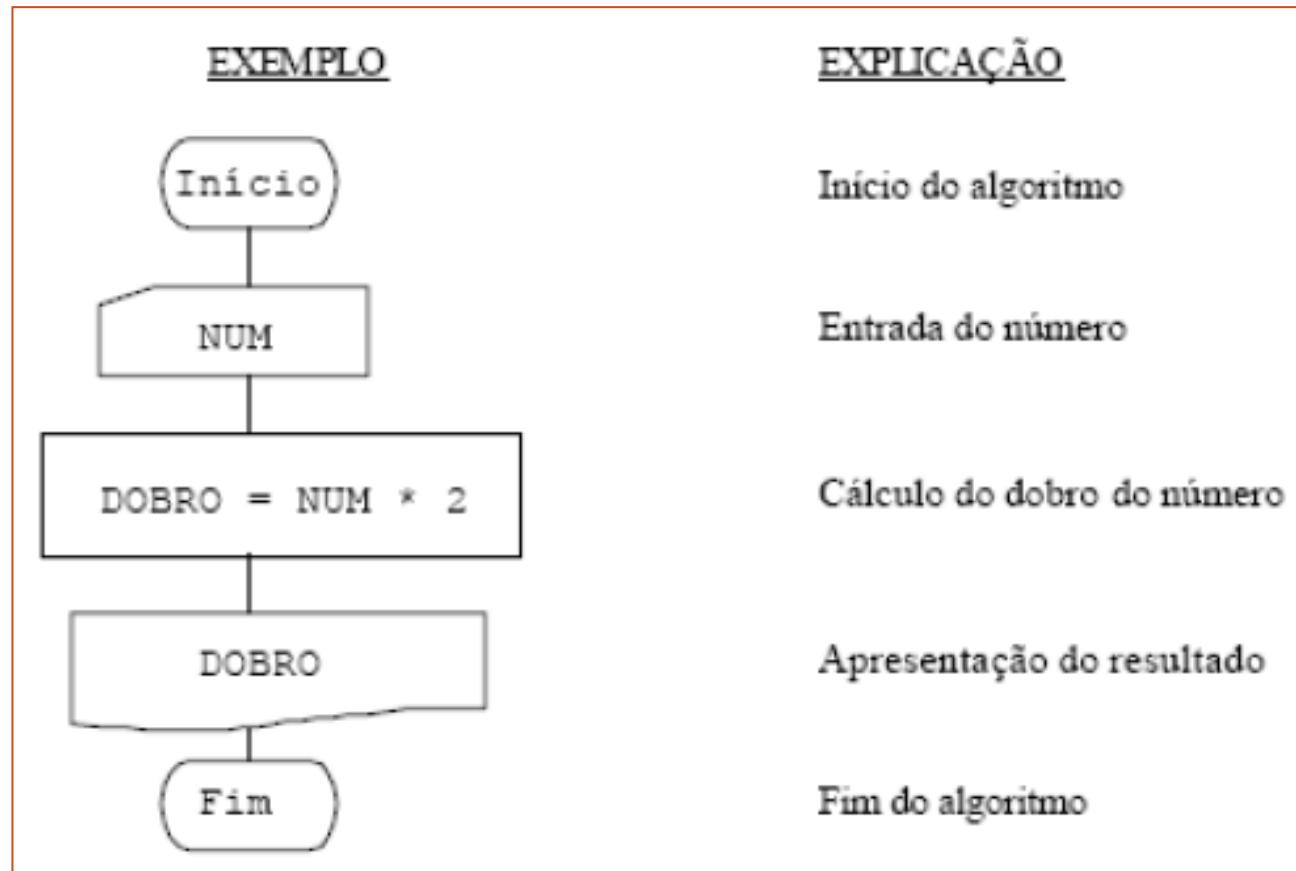
Fluxogramas

Fluxograma: Símbolos alternativos



Fluxogramas

Fluxograma do algoritmo para calcular o dobro de um número:



Estruturas de controle de fluxo

Os algoritmos desenvolvidos até o momento constituem uma sequência de ações que sempre são executadas em sua totalidade de forma sequencial independentemente do valor da entrada de dados.

Para a resolução de determinados problemas, pode ser necessário executar fluxos diferentes, baseado em uma decisão.

Exemplo: um algoritmo capaz de efetuar o cálculo do imposto de renda de um determinado contribuinte. Neste caso dependendo da quantidade de dependentes, do valor de sua renda e outros fatores o cálculo será feito de formas distintas, com diferentes resultados.

Expressões Lógicas

Na vida real tomamos decisões a todo momento baseadas em escolhas;

Em algoritmos, chamamos tais decisões de condições ou expressões lógicas;

Associada a uma expressão lógica, sempre existirá uma única alternativa: **Verdadeiro ou Falso**;

Ex.: Se o Corinthians não vencer o Santos na próxima partida, então o técnico será demitido.

Expressões Lógicas

Nos algoritmos computacionais, as expressões lógicas são feitas através dos operadores relacionais através de comparações:

Por exemplo:

idade > 18

Nesse caso, se a variável **idade** for igual a 19 ou **qualquer** valor maior que 19, a expressão será **VERDADEIRA**. Caso contrário será **FALSO**

Expressões Lógicas

OPERADORES RELACIONAIS	PORTUGUÊS ESTRUTURADO
Maior	>
Menor	<
Maior ou igual	>=
Menor ou igual	<=
Igual	=
Diferente	<>

Exemplo: Sejam duas variáveis, A = 5 e B = 3:

Expressão	Resultado
A = B	Falso
A <> B	Verdadeiro
A > B	Verdadeiro
A < B	Falso
A >= B	Verdadeiro
A <= B	Falso

Estruturas de decisão

Sintaxe:

```
se (<expressão-lógica>) então  
    <sequência de comandos>  
senão  
    <sequência de comandos>  
fimse
```

Exemplo: Considere um problema que exija uma decisão.

Tomemos como exemplo uma divisão onde haja a necessidade de que o algoritmo verifique se o divisor é igual ou diferente de zero. Se for igual não é possível dividir. Se for diferente é possível dividir →

Estruturas de decisão

Algoritmo em Portugol

```
algoritmo "Divisao"
```

```
var
```

```
    numero1, numero2: real
```

```
    resultado: real
```

```
inicio
```

```
    escreva ("Digite o dividendo ==> ")
```

```
    leia (numero1)
```

```
    escreva ("Digite o divisor ==> ")
```

```
    leia (numero2)
```

```
    se (numero2 = 0) entao
```

```
        escreva ("impossivel dividir por 0")
```

```
    senao
```

```
        resultado <- numero1/numero2
```

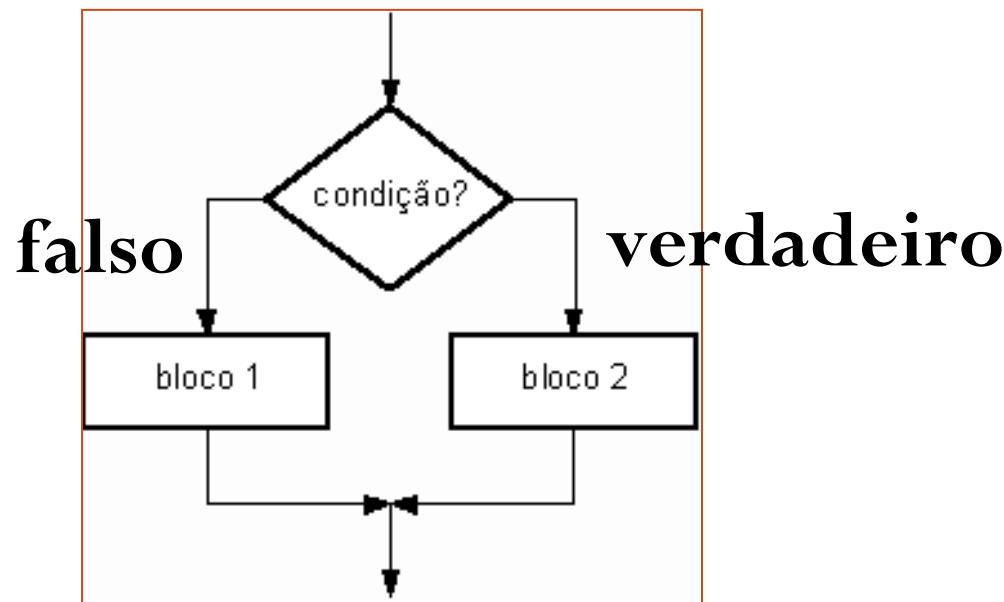
```
        escreva ("O resultado eh ==>", resultado)
```

```
    fimse
```

```
fimalgoritmo
```

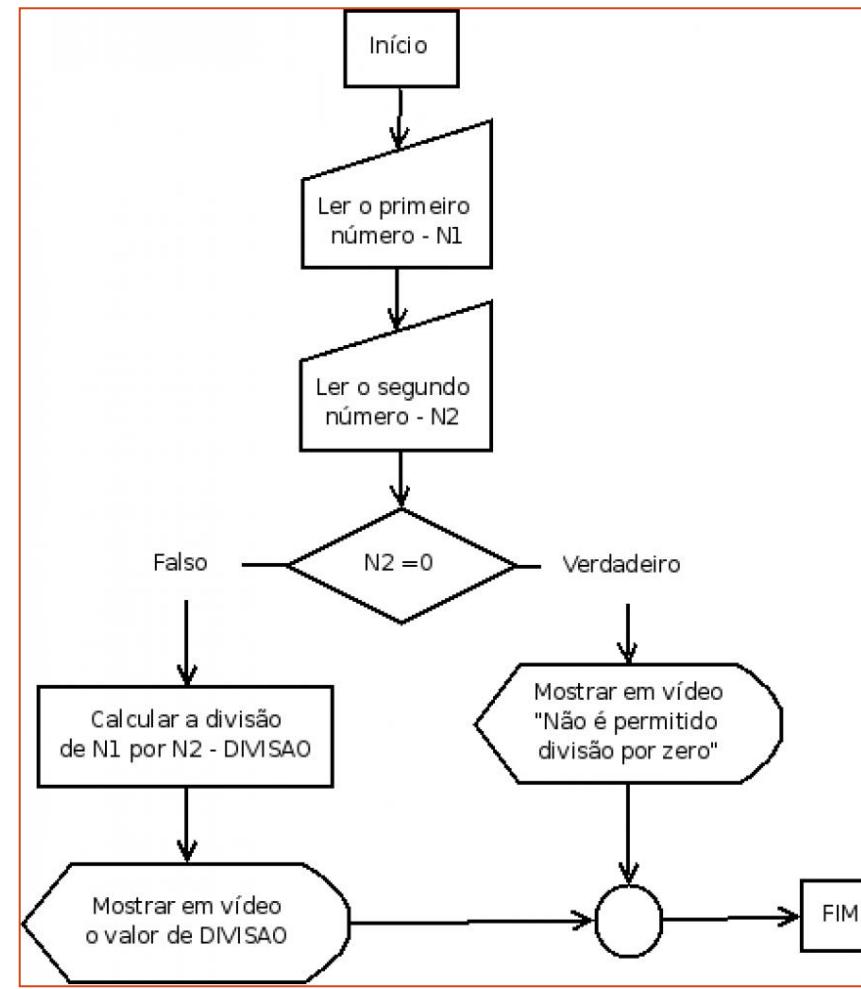
Estruturas de decisão

Estrutura de decisão em fluxograma



Estruturas de decisão

Fluxograma para o problema da divisão apresentado anteriormente:



Operadores lógicos

Operadores lógicos combinam resultados lógicos, gerando novos valores. São eles: **E**, **OU** e **NÃO**

Exemplo: Se “hoje for sábado” **E** “eu tiver dinheiro”, então irei para a balada.

Nesse caso, o conector “E” ligou as duas condições. O resultado só será verdadeiro, ou seja, eu irei para a balada somente se as duas forem verdadeiras.

Operadores lógicos

A “tabela-verdade” abaixo mostra todos os valores possíveis para duas variáveis A e B:

A	B	A e B	A ou B	não A
F	F	F	F	V
F	V	F	V	V
V	F	F	V	F
V	V	V	V	F

Expressões lógicas Compostas

Em algoritmos computacionais, combinando operadores relacionais e operadores lógicos criamos expressões lógicas compostas.

Por exemplo, se A = 5, B = 8 e C = 1:

(A = B) E (B > C)	é falso (f e v)
(A <> B) OU (B < C)	é verdadeiro (v ou f)
NÃO (A > B)	é verdadeiro (não f)
(A < B) E (B > C)	é verdadeiro (v e v)
(A >= B) OU (B = C)	é falso (f ou f)
NÃO (A <= B)	é falso (não v)

Exemplo Expressão Lógica Composta

algoritmo "voto_obrigatorio"

// Leia a idade de uma pessoa e verifique se seu voto é obrigatório
// O voto é obrigatório quando a pessoa possui entre 18 e 64 anos

var

idade : inteiro

inicio

escreva("Digite sua idade: ")

leia(idade)

se (idade > 17) E (idade < 65) entao

escreval("Voto obrigatório")

fimse

fimalgoritmo

Estruturas de Decisão Aninhadas

Em algumas situações, necessitamos decidir entre três ou mais alternativas para tomarmos um decisão. Como as condições lógicas só permitem duas alternativas, temos que usar condições aninhadas nesses casos.

Vejamos um exemplo onde temos que informar ao usuário se um número digitado é **Positivo**, **Negativo** ou **Neutro**. Temos três possibilidades. Um número N é:

- Positivo se $N > 0$
- Negativo, se $N < 0$
- Neutro, se $N = 0$

Vejamos a solução a seguir



Decisões Aninhadas Exemplo

```
algoritmo "positivo_negativo_neutro"
// Leia um número inteiro e informe se ele é negativo, positivo ou neutro
var
    numero : inteiro
inicio
    escreva("Informe o número: ")
    leia(numero)
    se (numero > 0) entao
        escreval("Número Positivo")
    senao
        se (numero < 0) entao
            escreval("Número Negativo")
        senao
            escreval("Número Neutro")
        fimse
    fimse
fimalgoritmo
```

Veja que, dentro de um bloco de decisão, podemos colocar qualquer instrução, inclusive um outro bloco de decisão.

Dessa forma, cada um tem os seu **FIMSE**.

Nesse exemplo, temos um outro bloco **SE** dentro do bloco **SENAO**.

Estrutura de Múltipla Escolha

Uma alternativa às condições aninhadas é a seleção de múltipla escolha que compara um valor a algumas expressões, desviando o fluxo de código para o bloco indicado pela primeira expressão verdadeira.

Sua estrutura básica é:

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

Vejamos um exemplo a seguir



Estruturas de Múltipla Escolha - Exemplo

```
algoritmo "positivo_negativo_neutro"
// Leia a sigla de um estado do SUDESTE brasileiro e informe seu nome
var
    sigla : caractere
inicio
    escreva("Informe a sigla do estado: ")
    leia(sigla)
    escolha sigla
        caso "SP"
            escreva("São Paulo")
        caso "MG"
            escreva("Minas Gerais")
        caso "RJ"
            escreva("Rio de Janeiro")
        caso "ES"
            escreva("Espírito Santo")
        outrocaso
            escreva("Não pertence ao SUDESTE")
    fimescolha
fimalgoritmo
```

Estruturas de repetição

Em alguns algoritmos, é necessário executar uma mesma tarefa por um número determinado ou indeterminado de vezes.

Exemplos:

- Escrever na tela os números de 1 a 10
- Escrever na tela a tabuada do número 2
- Calcular a média de todos os alunos de uma classe

Essa necessidade gerou a criação das **estruturas de repetição**.

Estrutura de repetição ENQUANTO

Neste caso, uma tarefa será repetida enquanto uma determinada condição for verdadeira.

Sintaxe:

```
enquanto (<expressão lógica>) faça  
    <sequência de comandos>  
fim enquanto
```

Estrutura de repetição ENQUANTO

Observações:

- A expressão lógica é avaliada antes de cada repetição do laço. Enquanto seu resultado for VERDADEIRO, a sequência de comando será executada. Por isso é chamada de **condição de parada**.
- Normalmente é usada uma ou mais variáveis para compor a condição de parada. Essas variáveis são chamadas de **variáveis de controle**
- Para que o laço tenha fim, a condição de parada, em algum momento deve ser atendida, caso contrário, teremos um laço de repetição infinito

Estrutura de repetição ENQUANTO

```
algoritmo "Conta10_com_Enquanto"
//Este algoritmo exibirá os número de 1 até 10
var
    contador : inteiro //Variável de controle

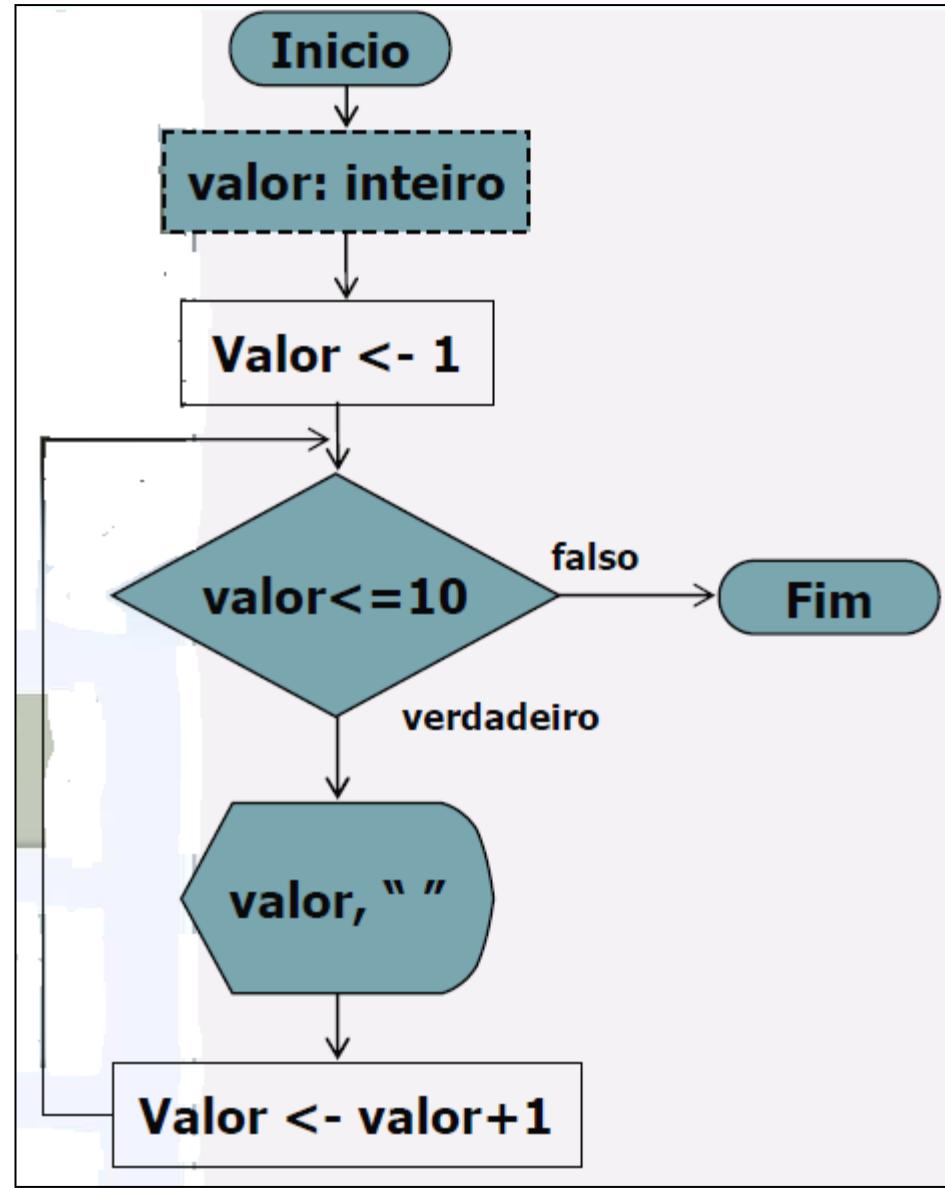
    inicio
        contador <- 1 //Valor de início da contagem

        //Esse bloco será repetido 10 vezes
        enquanto (contador <= 10) faca
            escreval(contador)
            contador <- contador + 1
        fimenquanto
    finalgoritmo
```

Condição de parada

Atribuição que força o fim do laço

Representação em fluxograma



Estrutura de repetição REPITA

Semelhante à instrução ENQUANTO só que, neste caso, o teste de parada é feito no final e, portanto, o bloco de comando será executado pelo menos uma vez

Sintaxe:

repita

<seqüência de comandos>

ate (<expressão lógica ou relacional>)

Estrutura de repetição REPITA

```
algoritmo "Conta10_com_Repita"
var
  contador:inteiro //Variável de controle

  inicio
    contador <- 1 //Início da contagem

    repita      //Início do laço de repetição
      escreval(contador)
      contador <- contador + 1
    ate (contador > 10)

  finalgoritmo
```

Repare que a condição é inversa a da estrutura ENQUANTO

Estrutura de repetição PARA

A estrutura **PARA** engloba todos os “requisitos” para realizar a repetição em uma única linha de comando. É usada principalmente em contagens, onde conhecemos o valor inicial e final da repetição.

```
variável de controle      Valor inicial      Valor final      Valor do incremento
para <variável> de <início> ate <fim> passo <incremento> faça
                                                <lista de comandos>
fimpara
```

The diagram illustrates the components of a FOR loop template with red annotations:

- Variável de controle** points to the placeholder <variável>.
- Valor inicial** points to the placeholder <início>.
- Valor final** points to the placeholder <fim>.
- Valor do incremento** points to the placeholder <incremento>.

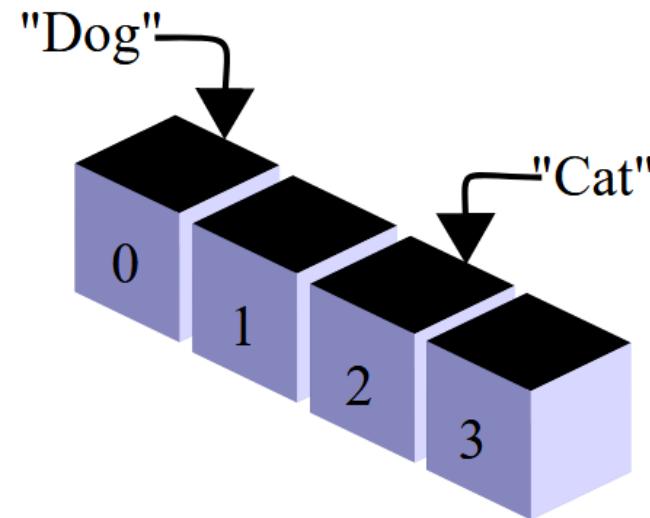
Exemplo Estrutura PARA

```
algoritmo "Conta10_com_PARA"  
//Este algoritmo exibirá os número de 1 até 10  
var  
contador : inteiro //Variável de controle
```

Início

Variável de controle	Valor inicial	Valor final	Valor do incremento
para contador de 1 ate 10 passo 1 faca			
escreval(contador)			
fimpara			
fimalgoritmo			

Temporada 3



Estruturas de Dados

Cadeias de Caracteres (Strings)

Cadeias de Caracteres - Conceitos

- As variáveis do tipo **Caractere**, mais conhecidas como **Strings**, armazenam uma sequência de caracteres, ou seja , uma palavra, uma frase, etc.
- Exemplos:
 - “José da Silva”
 - “Rua José da Silva, 123”
 - “VISUALG”
 - “123”

Observe que as constantes literais devem ser delimitadas por aspas duplas. Os espaços importam!

Cadeias de Caracteres - Conceitos

Cadeias de caracteres em VisualG são indexadas a partir do número 1 até N onde N é o tamanho da String (Número de caracteres)

Exemplo:

Na palavra “algoritmo”, a letra “a” está na posição 1 e a letra “g”, na posição 3

Criação de Variáveis Caractere

Para criarmos uma variável do tipo caractere, primeiramente declaramos os seu tipo na seção var e depois atribuímos um valor para a variável.

Exemplo:

```
var nome : caractere
```

No corpo do algoritmo:

```
nome <- "José da Silva"
```

OU

```
leia (nome)
```

Operações com Caracteres

- Concatenação

A concatenação ou junção de 2 ou mais variáveis caractere pode ser feita com o operador “+”

Exemplo:

```
a <- "Palmeiras"  
b <- "campeão"  
c <- a + " é " + b  
escreva(c);
```

Vai imprimir → Palmeiras é campeão

Operações com Caracteres

- Comparação

A comparação de duas variáveis caractere pode ser feita com os operadores relacionais $>$, \geq , $<$, \leq e \neq

Exemplo:

```
a <- "ANTONIO"  
b <- "ZÉ"  
escreva(a > b);
```

Vai imprimir → FALSO

Obs: “ZÉ” é maior que “ANTONIO” pois a letra “A” vem antes de “Z” na tabela ASCII

Funções de Caracteres

- Comprimento

Para sabermos o tamanho de uma variável String, ou seja, quantos caracteres ela possui, usamos a função **compr()**

Exemplo:

```
c <- "Java"  
escreva(compr(c)) ;
```

Vai imprimir → 4

Obs: Os espaços importam!

Funções de Caracteres

- Substrings

Podemos extrair uma parte da String, informando a sua posição e números de caracteres desejados usando a função copia()

Exemplo:

```
a <- "VISUALG";  
b <- copia(a,5,3)  
escreva(b);
```

Vai imprimir → ALG

Funções de caracteres

- Posição dos caracteres na String

Podemos saber qual posição um caractere ou substring ocupa na String com a função pos()

Exemplo:

```
s <- "ALGORITMO";
```

```
System.out.print(pos("G", s);
```

Vai imprimir → 3

Obs: se o caractere não existir, a função retorna 0

Exercício Resolvido

Algoritmo "imprime_letras"

// Leia uma variável do tipo caractere e depois imprima cada
// uma de suas letras

var

palavra : caractere

tam,contador : inteiro

inicio

escreva ("Digite uma palavra: ")

leia(palavra)

tam <- compr(palavra)

para contador de 1 ate tam faca

escreval(copia(palavra,contador,1))

fimpara

fimalgoritmo

Vetores (Arrays)

Vetores

- Um vetor (= array) é uma estrutura de dados que armazena uma **sequência de valores** (variáveis), todos do **mesmo tipo**, em **posições consecutivas** da memória.
- Cada elemento do vetor é identificado por sua posição dentro do vetor (**índice do vetor**).
- São conhecidos como **Estruturas de Dados Unidimensionais Homogêneas**. Unidimensionais, pois forma uma sequência linear de valores consecutivos. Homogêneas, pois todos os valores são do mesmo tipo

Vetores

- A quantidade de elementos deve ser definida no momento da declaração;
- Os elementos possuem **ordinalidade**, cada um pode ser identificado pela sua posição (**índice do vetor**);
- Cada elemento do vetor, por meio do seu índice, pode ser acessado como uma variável individual.
- No vetor abaixo de nome “Notas”, o valor do elemento de índice 4 é 6,5.

8,5	7,0	9,0	6,5	8,0	6,0	...
1	2	3	4	5	6	...
Vetor Notas						

← Valores
← Índices

Declaração de Vetores

Sintaxe

```
var nomevetor: vetor[faixa] de <tipo>
```

Exemplos:

```
var notas: vetor[1..10] de real
```

→ Cria um vetor de nome “notas” com 10 números reais

```
var x: vetor[1..50] de inteiro
```

→ Cria um vetor de nome “x” com 50 números inteiros

Vetores – Atribuição de Valores

Para atribuir um valor a um elemento do vetor, precisamos indicar seu índice

Sintaxe:

nomevetor [índice] <- valor

Exemplos:

notas [1] <- 10.0

Atribuímos o valor 10.0 para o elemento de índice 1

x <- 2

notas [3] <- x

Atribuímos o valor da variável x (2) para o elemento de índice 3

Notas [x] <- 5.0

Atribuímos o valor 5.0 para o elemento de índice x que é 2

Leitura e Escrita de Vetores

Podem ser trabalhados como variáveis comuns, bastando indicar o índice

Exemplos:

Leia (vet[1])

Escreva (vet[1])

Exemplo de Algoritmo com Vetor

algoritmo "vetor"

// Crie um vetor inteiro com 10 elementos e atribua, para
// cada elemento, o valor do dobro de seu próprio índice

var

 vet : vetor [1..10] de inteiro

 indice : inteiro

inicio

 para indice de 1 ate 10 faca

 vet[indice] <- indice * 2

 escreva(vet[indice])

 fimpara

Fimalgoritmo

Resultado:

elementos

2	4	6	8	10	12	14	16	18	20
1	2	3	4	5	6	7	8	9	10

índices

Matrizes

Matrizes

Enquanto que um vetor é uma estrutura de dados homogênea unidimensional (cresce em apenas uma direção), uma matriz é uma **estrutura de dados homogênea multidimensional** (ela pode crescer em mais de uma direção).

Dessa forma, uma matriz se assemelha à uma planilha, onde os elementos são identificados por dois índices: **Linha e Coluna**

Matriz M	Coluna 1	Coluna 2	Coluna 3
Linha 1	M[0][0]	M[0][1]	M[0][2]
Linha 2	M[1][0]	M[1][1]	M[1][2]
Linha 3	M[2][0]	M[2][1]	M[2][2]

Nome da matriz

Índice da linha

Índice da coluna

Fonte: SENAC

Operações com Matrizes

De mesma forma como fizemos com os vetores, as operações de atribuição e leitura com matrizes seguem o mesmo padrão, com a exceção que sempre deveremos informar dois índices, a linha e a coluna do elemento.

$$M = \begin{bmatrix} 3 & 8 & 5 \\ 9 & 7 & 3 \\ 4 & 2 & 9 \end{bmatrix}$$

Por exemplo, na matriz M 3x3 acima, o elemento M[1,3] é 5.

Para atribuirmos o valor 0 para esse elemento, faremos:

M[1,3] <- 0

E para ler um valor para esse elemento, faremos:

Leia(M[1,3])

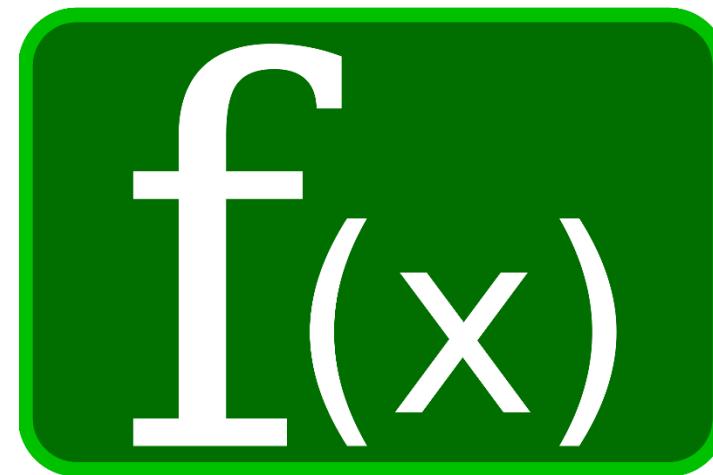
Matrizes - Exemplo

```
algoritmo "matriz"
var
i, j : inteiro
matriz:vetor[1..3,1..3] de inteiro
inicio
    para i de 1 ate 3 faca
        para j de 1 ate 3 faca
            escreva("Informe o elemento [",i,j,"]")
            leia(matriz[i,j])
        fimpara
    fimpara
    i <- 1
    j <- 1
    para i de 1 ate 3 faca
        escreval("")
        para j de 1 ate 3 faca
            escreva(matriz[i,j])
        fimpara
    fimpara
fimalgoritmo
```

Neste exemplo, estamos criando uma matriz de dimensões 3x3 e depois solicitando ao usuário que informe os valores para cada elemento.

Repare que, como essa matriz possui duas dimensões, necessitamos de duas variáveis para os índices e dois laços de repetição para processá-los

Temporada 4

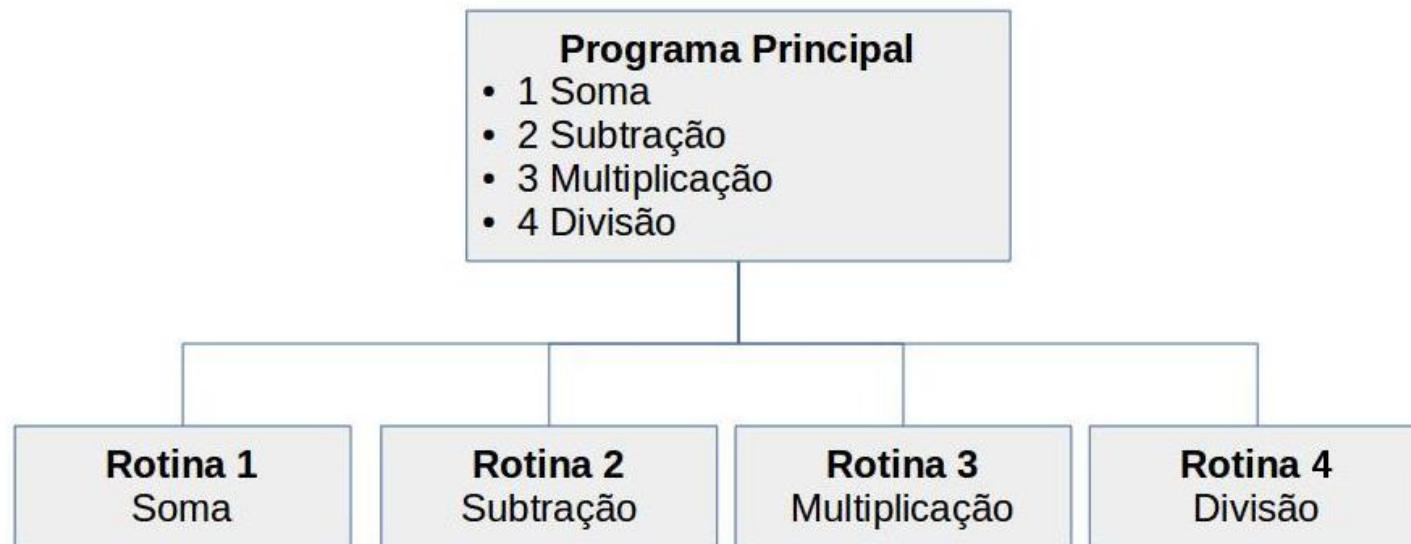


Modularização

Modularização

Um algoritmos complexo é melhor abordado se for dividido em várias partes menores (**módulos** ou **sub-rotinas**). Modularizar, portanto, significa **subdividir** um algoritmo em partes menores **independentes** que se comunicam entre si para atingir um objetivo específico.

Um módulo é independente de tal forma que pode ser reutilizado até mesmo em outros algoritmos.



Fonte: [Forumeiros](#)

Modularização

O conceito de modularização se baseia nos seguintes princípios:

- **Abstração:** é a habilidade de se concentrar nos aspectos essenciais de um problema, ignorando características “menos importantes”.
- **Encapsulamento:** vem do verbo **encapsular**, ou seja, agrupar as funcionalidades em cápsulas (módulos) independentes com objetivos bem definidos. É daqui que surgem conceitos como **package** ou **namespaces**, que não abordaremos nesse curso.
- **Ocultamento de informação:** Significa “esconder” os dados que são importantes apenas para o módulo em questão. Cada módulo é como se fosse uma **caixa-preta** que disponibilizaria somente as informações necessárias para o mundo externo. É daqui que surge o conceito de **visibilidade** que veremos a seguir.
- **Reusabilidade:** Planejar o módulo de forma que sua **utilidade** não se limite às necessidades do algoritmo principal para o qual foi inicialmente projetado,

Interface

Cada módulo se comunica um com o outro através de um contrato, uma interface (assinatura) do módulo. Nessa interface, nós descrevemos tanto as informações que o módulo precisa receber para executar sua função, bem como as informações que o módulo retorna para o módulo que o chamou.

Dessa forma, no módulo existem dois componentes: o **corpo** e a **interface**.

- O **corpo** é o algoritmo interno que executa a função para a qual o módulo foi construído.
- A **interface** que pode ser vista como a descrição dos dados de entrada e de saída. O conhecimento dessa interface é tudo o que é necessário para a sua utilização.

A interface de um módulo é definida em termos de parâmetros. Um **parâmetro** é um tipo de variável utilizada para a troca de mensagens. Existem três tipos:

- **Parâmetros de entrada**, que são os valores recebidos pelo módulo.
- **Parâmetros de saída**, que são os valores retornados pelo módulo.
- **Parâmetros de entrada e saída**, que permitem tanto a passagem como o retorno dos valores.

Sub-rotinas

Em **VisuALG**, existem dois tipos de módulos ou sub-rotinas:

- **Função**: uma função é um módulo que realiza uma tarefa e retorna um único valor de saída, assim como em uma função Matemática.
- **Procedimento**: um procedimento é um módulo que realiza uma tarefa, mas não retorna nenhum valor de saída.

Ambos os tipos de sub-rotinas podem receber, opcionalmente parâmetros de entrada que podem ser utilizados internamente para executar suas funções.

Funções

Para definirmos uma função em VisualG, utilizamos a seguinte sintaxe:

```
funcao <nome> [(<parâmetros>)] :<tipo-dado>
// Seção de Declarações Internas
inicio
// Seção de Comandos
[retorne <expressão>]
fimfuncao
```

Nome da função Declaração dos Parâmetros de Entrada Tipo de dado do Parâmetro de retorno

Retorno do valor calculado para o algoritmo/módulo chamador

Funções - Exemplo

Neste exemplo vamos calcular a soma de 2 números através de uma função

```
algoritmo "funcao_soma"
```

```
var numero1, numero2 : real
```

```
funcao soma(n1,n2 : real) : real
```

```
var resultado : real
```

```
inicio
```

```
    resultado <- n1 + n2
```

```
    retorno resultado
```

```
fimfuncao
```

```
inicio
```

```
escreva("Informe o valor de numero1: ")
```

```
leia(numero1)
```

```
escreva("Informe o valor de numero2: ")
```

```
leia(numero2)
```

```
escreva ("Soma = ", soma(numero1,numero2))
```

```
Fimalgoritmo
```

Assinatura da função: Aqui declaramos uma função de nome **SOMA** que possui 2 **parâmetros de entrada** (n1 e n2) do tipo **REAL** e retorna um valor do tipo **REAL** que será a soma dos valores de entrada. Os valores de n1 e n2 serão passados pelo algoritmo principal.

Declaração da variável interna da função. Essa variável só é visível aqui

Corpo da função: Aqui calculamos o valor da soma dos parâmetros para a variável **RESULTADO** e retornamos esse valor para o algoritmo principal através do comando **RETORNE**

Algoritmo Principal: Aqui lemos as variáveis numero1 e numero2 e passamos seus valores para que a função calcule a soma.

Chamada da função: Veja a correspondência entre os valores realizada durante a chamada. Esses valores passados na chamada são designados como **ARGUMENTOS**.

Funções – Outro Exemplo

```
algoritmo "funcao_par"
var numero : inteiro
funcao par(n : inteiro) : logico
var resultado : logico
inicio
    resultado <- FALSO
    resultado <- (numero % 2 = 0)
    retorno resultado
fimfuncao
inicio
escreva("Informe um número: ")
leia(numero)
se (par(numero)) entao
    escreva ("O número ", numero, " é par")
senao
    escreva ("O número ", numero, " é ímpar")
fimse
fimalgoritmo
```

Função que recebe um número INTEIRO como parâmetro e devolve um resultado BOOLEANO indicando se esse número é PAR ou não

Visibilidade

A visibilidade de variáveis, que se baseia no princípio do **OCULTAMENTO de INFORMAÇÕES**, determina em que ponto do algoritmo essa variável estará acessível para ser manipulada.

Em **VisuALG**, variáveis podem ser **locais** ou **globais**. Uma variável é dita **local** a um módulo se ela é declarada naquele módulo e, portanto, só estará acessível nesse módulo. Caso contrário, ela será **global** e, portanto, acessível em qualquer ponto do programa.

No exemplo da página anterior, as variáveis **numero1** e **numero2** são globais, pois foram declaradas no algoritmo principal. Já a variável **resultado** foi declarada dentro do módulo e, desse modo, só está acessível para o próprio módulo. Qualquer tentativa de acessar a variável **resultado** fora da função **soma**, resultará em erro.

Visibilidade – Contra-exemplo

```
algoritmo "funcao_soma"
```

```
var numero1, numero2 : real
```

```
funcao soma(n1,n2 : real) : real
```

```
var resultado : real
```

```
inicio
```

```
    resultado <- numero1 + numero2
```

```
    retorno resultado
```

```
fimfuncao
```

```
inicio
```

```
escreva("Informe o valor de numero1: ")
```

```
leia(numero1)
```

```
escreva("Informe o valor de numero2: ")
```

```
leia(numero2)
```

```
escreva ("Soma = ", resultado)
```

```
Fimalgoritmo
```

Variáveis globais são declaradas no algoritmo principal e são visíveis em todas as partes do algoritmo

Variáveis locais são declaradas dentro do módulo e são visíveis somente aqui

Nesse ponto, alteramos a função para usar os valores das variáveis globais ao invés dos parâmetros. Como as variáveis globais são visíveis em todo o algoritmo, não ocorrerá erro ao serem acessadas dentro da função

Já, nesse ponto, a variável local **resultado** não é visível e ocorrerá um erro.

Esse exemplo foi escrito propositalmente com erros para demonstrar o conceito de VISIBILIDADE

Procedimentos

Em VisuALG, procedimento é um subprograma que não retorna nenhum valor. Sua sintaxe é a seguinte:

```
procedimento <nome> [(<seqüência-parâmetros>)]  
// Seção de Declarações Internas  
inicio  
// Seção de Comandos  
fimprocedimento
```

Procedimento - Exemplo

```
algoritmo "proc_soma"  
    var numero1, numero2, resultado : real  
  
procedimento soma(n1,n2 : real)  
inicio  
    // resultado é a variável global  
    resultado <- n1 + n2  
fimprocedimento  
  
inicio  
    escreva("Informe o valor de numero1: ")  
    leia(numero1)  
    escreva("Informe o valor de numero2: ")  
    leia(numero2)  
    Soma(numero1,numero2) ←  
    escreva ("Soma = ", resultado)  
Fimalgoritmo
```

Veja que um procedimento não retorna valores, então não pode ser usado como operandos em uma expressão nem como parâmetros da função ESCREVA

Passagem de Parâmetros

Nas chamadas de sub-rotinas, existem duas formas para passarmos os parâmetros:

- A **passagem por valor** que é a que utilizamos nos exemplos anteriores. Nesse tipo de comunicação, os parâmetros agem como variáveis locais, com a diferença que são inicializados previamente. Nesse caso, os valores das variáveis globais utilizadas como parâmetros não são alterados pela sub-rotina.
- Na **passagem por referência**, a sub-rotina não recebe apenas um valor, mas a própria variável (seu endereço). Dessa forma, o valor original da variável pode ser alterado dentro da sub-rotina

Passagem de Parâmetros por Referência

```
algoritmo "proc_soma"  
    var numero1, numero2, resultado : real
```

```
procedimento soma(n1,n2 : real; var resultado: real)
```

```
inicio
```

```
    resultado <- n1 + n2
```

```
fimprocedimento
```

```
inicio
```

```
    escreva("Informe o valor de numero1: ")
```

```
    leia(numero1)
```

```
    escreva("Informe o valor de numero2: ")
```

```
    leia(numero2)
```

```
    Soma(numero1,numero2,resultado)
```

```
    escreva ("Soma = ", resultado)
```

```
Fimalgoritmo
```

Para passar um parâmetro por referência,
Use a palavra VAR

Veja que o valor da variável RESULTADO foi
Alterado dentro do procedimento quando usamos
Passagem por referência

Considerações Finais

São boas práticas e devem ser incentivadas:

- Modularização
- Variáveis locais ao invés de globais
- Passagem de parâmetros por valor

Motivos:

- Redução da complexidade
- Otimização da memória
- Segurança
- Redução de erros
- Reusabilidade
- Diminuição de custos e prazos

Exercícios Propostos

1) Um sistema de equações lineares do tipo: $ax + by = c$ pode ser resolvido segundo: $x = \frac{ce - bf}{ae - bd}$ $y = \frac{af - cd}{ae - bd}$ $dx + ey = f$

Escreva um algoritmo que lê os coeficientes **a,b,c,d,e** e **f** e calcule de **x** e **y**.

2) Escreva um algoritmo que leia 3 números inteiros e mostre o maior deles.

3) Leia um número **N** e escreva os **N** números maiores que **N**.

Exemplo: **N = 4** → escrever: 5,6,7,8

4) Leia um vetor com 5 números inteiros e um número inteiro **N**. Calcule o produto escalar entre o número **N** e o vetor.

5) Escreva uma função para verificar se uma palavra informada pelo usuário é palíndroma ou não.

FIM

Obrigado!

Para obter materiais complementares e mais exercícios resolvidos, consulte:

- <http://www.josecintra.com/blog>
- [GitHub](#)

Saiba mais...

Links

- [Manual do VisualG 3](#)
- [www.inf.ufsc.br/~vania/teaching/ine5201/index.html](#)
- [www.univasf.edu.br/~mario.godoy/Disciplina-Algoritmos.htm](#)
- [https://www.dca.ufrn.br/~affonso/DCA800/pdf/algoritmos_parte1.pdf](#)

Livros

- “Lógica de Programação” por André Luis Forbellone e outros
- “Lógica de Programação com VISUALG” por Fabíola Ventavoli
- “Algoritmos: Teoria e Prática” por Thomas Cormen e outros