UCS
UNIVERSIDADE
DE CAXIAS DO SUL

Introdução aos algoritmos

Bruno Tonet
Cristian Koliver

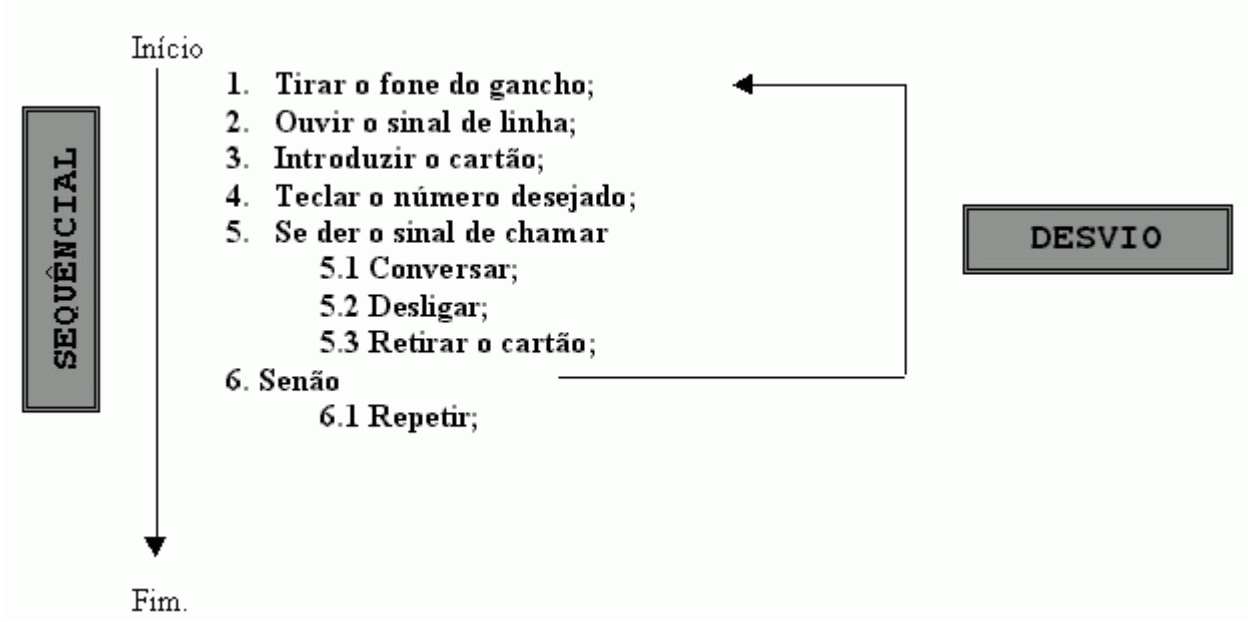
SUMÁRIO

1 - Algoritmos Não Computacionais

Um *algoritmo* é uma seqüência de instruções finita e ordenada de forma lógica para a resolução de uma determinada tarefa ou problema. São exemplos de algoritmos instruções de montagem, receitas, manuais de uso, etc. Um algoritmo não é a solução do problema, pois, se assim fosse, cada problema teria um único algoritmo; um algoritmo é um *caminho* para a solução de um problema. Em geral, existem muitos (senão infinitos) caminhos que levam a uma solução satisfatória.

Um *algoritmo* não computacional é um algoritmo cuja seqüência de passos, a princípio, não pode ser executada por um computador. Abaixo é apresentado um algoritmo não computacional cujo objetivo é usar um telefone público. Provavelmente você "executou" o algoritmo deste exemplo diversas vezes. O termo algoritmo está muito ligado à Ciência da Computação, mas, na realidade, ele pode ser aplicado a qualquer problema cuja solução possa ser decomposta em um grupo de instruções.

Exemplo 1.1



Um outro exemplo típico de algoritmo é uma receita culinária, como no exemplo abaixo.

Exemplo 1.2

Algoritmo para fritar um ovo

```
1.    Colocar um ovo na frigideira
2.    Esperar o ovo ficar frito
3.    Remover o ovo da frigideira
```

O algoritmo acima, no entanto, poderia ser mais detalhado e completo. Uma versão mais aceitável seria:

Exemplo 1.3

Algoritmo para fritar um ovo

```
1.    Retirar um ovo da geladeira
2.    Colocar a frigideira no fogo
3.    Colocar óleo
4.    Esperar até o óleo ficar quente
5.    Quebrar o ovo separando a casca
6.    Colocar o conteúdo do ovo na frigideira
7.    Esperar um minuto
8.    Retirar o ovo da frigideira
9.    Apagar o fogo
```

Essa segunda versão é mais completa e detalhada que a anterior. Nela, várias ações que estavam subentendidas foram explicitadas. No entanto, para que o algoritmo possa ser útil, é necessário ainda que quem faz uso dele conheça os termos utilizados nas instruções. O algoritmo do exemplo só será útil para alguém que seja fluente na língua portuguesa e conheça o significado dos verbos `Retirar`, `Colocar`, `Esperar` assim como dos substantivos utilizados no contexto de uma receita culinária. Em outras palavras, é preciso que a linguagem utilizada no algoritmo seja conhecida tanto por quem o *escreveu* quanto por quem vai *executá-lo*.

Para que o algoritmo possa ser executado por uma máquina é importante que as instruções sejam corretas e sem ambigüidades. Portanto, a forma especial de linguagem que utilizaremos é bem mais restrita que o Português e com significados bem definidos para todos os termos utilizados nas instruções. Essa linguagem é conhecida como **Português Estruturado** (às vezes também chamada de *Portugol*). O português estruturado é, na verdade, uma simplificação extrema do Português, limitada a umas poucas palavras e estruturas que têm um significado muito bem definido. Ao conjunto de palavras e regras que definem o formato das sentenças válidas chamamos **sintaxe da linguagem**. Durante este texto, a sintaxe do Português Estruturado será apresentada progressivamente e a utilizaremos em muitos exercícios de resolução de problemas.

Aprender as palavras e regras que fazem parte dessa sintaxe é fundamental; no entanto, não é o maior objetivo deste curso. O que realmente exigirá um grande esforço por parte do estudante é aprender a **resolver problemas** utilizando a linguagem. Para isso, há somente um caminho: resolver muitos problemas. O processo é semelhante ao de tornar-se competente em um jogo qualquer: aprender as regras do jogo (a sintaxe) é só o primeiro passo, tornar-se um bom jogador (programador) exige tempo, muito exercício e dedicação.

Embora o Português Estruturado seja uma linguagem bastante simplificada, ela possui todos os elementos básicos e uma estrutura semelhante à de uma linguagem típica para programação de computadores. Além disso, resolver problemas com português estruturado, pode ser uma tarefa tão complexa quanto a de escrever um programa em uma linguagem de programação qualquer. Portanto, neste curso, estaremos na verdade procurando desenvolver as habilidades básicas que serão necessárias para adquirir-se competência na programação de computadores.

Para praticar nossa sintaxe e testar nossos problemas, utilizaremos o software Visualg desenvolvida por Cláudio Morgado de Souza. E-mail: cmorgado@apoioinformatica.com.br.

2 - Algoritmos Computacionais

O computador, a princípio, não executa nada. Para que ele faça uma determinada tarefa - calcular uma folha de pagamento, por exemplo -, é necessário que ele execute um *programa*. Um *programa* é um conjunto de milhares de instruções que indicam ao computador, passo a passo, o que ele tem que fazer. Logo, um programa nada mais é do que um algoritmo computacional descrito em uma linguagem de programação. Uma linguagem de programação contém os comandos que fazem o computador escrever algo na tela, realizar cálculos

aritméticos, receber uma entrada de dados via teclado, e milhares de outras coisas, mas estes comandos precisam estar em uma ordem lógica.

O termo *processamento de dados* é muitas vezes utilizado em conjunto com computadores, pois, em geral, é isto o que eles fazem: processar dados. Daí podem extrair os dois componentes básicos de um algoritmo computacional (de agora em diante, esta palavra sempre utilizada no contexto de algoritmos computacionais): *dados* e *código*. Dados são os valores (números, nomes, etc.) de que precisamos para resolver o problema, e código são os comandos ou instruções que usaremos para manipular e "processar" os dados.

3 - Linearização de Expressões

Para a construção de algoritmos que realizam cálculo matemáticos, todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linhas, devendo também ser feito o mapeamento dos operadores da aritmética tradicional para os do Português Estruturado.

Exemplo 3.1	
$\left\{\left[\frac{2}{3}-(5-3)\right]+1\right\}.5$	`((2/3-(5-3))+1)*5`
Tradicional	**Computacional**

As tabelas seguintes mostram os operadores aritméticos disponíveis no Português Estruturado.

OPERADORES ARITMÉTICOS	PORTUGUÊS ESTRUTURADO
Adição	**+**
Subtração	**-**
Multiplicação	*****
Divisão	**/**
Divisão Inteira	****
Exponenciação	**^ ou Exp** (<*base*>,<*expoente*>)
Módulo (resto da divisão)	**%**

Os operadores relacionais realizam a comparação entre dois operandos ou duas expressões e resultam em valores lógicos (**VERDADEIRO** ou **FALSO**).

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

Exemplo 3.2

2+5>4 resulta **VERDADEIRO**

3<>3 resulta **FALSO**

Os operadores lógicos atuam sobre expressões e também resultam em valores lógicos **VERDADEIRO** ou **FALSO**.

OPERADORES LÓGICOS	PORTUGUÊS ESTRUTURADO	SIGNIFICADO
Multiplicação lógica	**E**	Resulta **VERDADEIRO** se ambas as partes forem verdadeiras.
Adição lógica	**Ou**	Resulta **VERDADEIRO** se uma das partes é verdadeira.
Negação	**Nao**	Nega uma afirmação, invertendo o seu valor lógico: se for **VERDADEIRO** torna-se **FALSO**, se for **FALSO** torna-se **VERDADEIRO**.

A tabela abaixo – chamada ***tabela-verdade*** – mostra os resultados das aplicações dos operadores lógicos conforme os valores dos operadores envolvidos.

A	B	A E B	A OU B	NÃO A	NÃO B
VERDADEIRO	**VERDADEIRO**	**VERDADEIRO**	**VERDADEIRO**	**FALSO**	**FALSO**
VERDADEIRO	**FALSO**	**FALSO**	**VERDADEIRO**	**FALSO**	**VERDADEIRO**
FALSO	**VERDADEIRO**	**FALSO**	**VERDADEIRO**	**VERDADEIRO**	**FALSO**
FALSO	**FALSO**	**FALSO**	**FALSO**	**VERDADEIRO**	**VERDADEIRO**

De acordo com a necessidade, as expressões podem ser unidas pelos operadores lógicos.

Exemplo 3.3

(2+5>4) e (3<>3) resulta **FALSO**, pois **VERDADEIRO** e **FALSO** resulta **FALSO**.

A modularização é a divisão de uma expressão em partes, proporcionando maior compreensão e definindo prioridades para a resolução da mesma. Como pôde ser observado no exemplo anterior, em expressões computacionais utilizamos somente parênteses "()" para modularização. Na sintaxe do Português Estruturado podemos ter parênteses dentro de parênteses, como seriam os colchetes e as chaves na matemática.

Os parênteses indicam quais sub-expressões, dentro de uma expressão, serão executados primeiro. A princípio, a execução é da esquerda para direita, mas além dos parênteses, existem prioridades entre os operadores envolvidos na expressão. Tais prioridades são mostradas nas tabelas seguintes.

OPERADOR ARITMÉTICO	PRIORIDADE
Exponenciação	3 (maior)
Multiplicação	2
Divisão	2
Adição	1
Subtração	1 (menor)

Exemplo 3.4

(2 + 2)/2 resulta 2 e 2 + 2/2 resulta 3

OPERADOR LÓGICO	PRIORIDADE
e	3
ou	2
nao	1

Exemplo 3.5

(2>3) ou (3<2) e (2<3) //resultado seria **Falso**
(2>3) e (3<2) ou (2<3) //resultado seria **Verdadeiro**

Entre as categorias de operadores também há prioridades, conforme mostrado na tabela abaixo.

OPERADOR	PRIORIDADE
Operadores aritméticos	3
Operadores relacionais	2
Operadores lógicos	1

Lembrete:

O software VisuAlg não possui relacionamento de categorias.
2*5>3 ou 5+1<2 e 2<7-2 // resulta em erro.
(2*5>3) ou (5+1<2) e (2<7-2) // certo seria assim.

4 - Forma Geral de um ALGORITMO

Nessa seção vamos conhecer os primeiros elementos que compõem o Português Estruturado e escrever alguns algoritmos.

A estrutura geral de um algoritmo é:

```
Algoritmo "<nome do algoritmo>"

var
< declaração de variáveis>
inicio
< lista de comandos>
fimalgoritmo
```

onde as palavras **algoritmo** e **fimalgoritmo** fazem parte da sintaxe da linguagem e sempre delimitam o **inicio** e **fim** de um algoritmo; a < **declaração de variáveis**> é a seção ou parte do algoritmo onde descrevemos os tipos de dados que serão usados na lista de comandos. Por exemplo, poderíamos definir que fruta é um tipo de dado que pode assumir apenas os valores maçã, pêra, banana, abacaxi e outras frutas, sobre os quais podemos

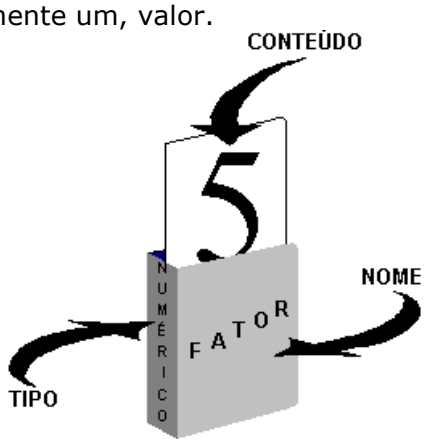
efetuar as operações comparar, comprar, comer e servir; **inicio** indica o fim das declarações e o início da seção de comandos; < **lista de comandos** > é apenas uma indicação de que entre as palavras **inicio** e **fimalgoritmo** podemos escrever uma lista com uma ou mais instruções ou comandos. É importante salientar que, quando um algoritmo é "executado", as instruções ou comandos de um algoritmo são sempre executados na ordem em que aparecem no mesmo.

As palavras que fazem parte da sintaxe da linguagem são *palavras* ***reservadas***, ou seja, não podem ser usadas para outro propósito em um algoritmo que não seja aquele previsto nas regras de sintaxe. A palavra **algoritmo**, por exemplo, é uma palavra reservada. Neste texto, as palavras reservadas sempre aparecerão em negrito.

5 - Variáveis

Uma ***variável*** pode ser vista como uma caixa com um ***rótulo*** ou ***nome*** colado a ela, que num dado instante guarda um determinado objeto. O ***conteúdo*** desta caixa não é algo fixo, permanente. Na verdade, essa caixa pode ter seu conteúdo alterado diversas vezes. Contudo, o conteúdo deve ser sempre do mesmo tipo.

Na figura abaixo, a caixa (variável) rotulada com **FATOR** contém o valor 5. Como seu tipo é numérico, em um determinado instante essa caixa poderá conter qualquer valor numérico (inteiro ou fracionário; positivo, negativo ou zero). Entretanto, em um determinado instante, ela conterá um, e somente um, valor.



Variáveis são palavras que tem um significado bem específico em um algoritmo. Para que o computador possa executar comandos que envolvem variáveis da maneira correta, ele deve conhecer os detalhes das variáveis que pretendemos usar. Esses detalhes são: o identificador desta variável e o tipo de valores que essa variável irá conter. Precisamos assim, de uma maneira de especificar esses detalhes e comunicá-los ao computador. Para isso devemos declarar nossas variáveis logo abaixo da expressão ***"VAR"*** que tem a seguinte forma:

```
VAR
<identificador 1>, <identificador 2>, ..., <identificador n>: <tipo das
variáveis>
```

onde **<identificador *i*>** é o nome (identificador) de uma variável e **<tipo das variáveis>** determina que tipo de valor as variáveis poderão receber.

Os identificadores das variáveis são usados para referenciá-las dentro do algoritmo. Tais identificadores devem ser claros e precisos, dando uma idéia do "papel" da variável no algoritmo.

18/02/2014

A identificação ou nomeação de variáveis segue algumas regras:

a. *nomes de variáveis não podem ser iguais a palavras reservadas;*

b. *nomes de variáveis devem possuir como primeiro caractere uma letra ou sublinhado '_' (os outros caracteres podem ser letras, números e sublinhado);*

c. *nomes de variáveis devem ter no máximo 127 caracteres;*

d. *Nomes de variáveis não podem conter espaços em branco;*

e. *na sintaxe do Português Estruturado, não há diferença entre letras maiúsculas de minúsculas (NOME é o mesmo que noMe).*

Exemplo 5.1

Identificadores válidos: NOME, TELEFONE, IDADE_FILHO, IdadeFilho, NOTA1, Est_Civil

Identificadores inválidos: 3Endereco, Estado Civil, PARA, algoritmo, numero/complemento

Você deve estar se perguntando por que a palavra "PARA e algoritmo" são identificadores inválidos. Eles são inválidos, pois são palavras reservadas da linguagem, veja outras palavras que você não deve utilizar como identificadores de variáveis.

PALAVRAS RESERVADAS			
aleatorio	e	grauprad	passo
abs	eco	inicio	pausa
algoritmo	enquanto	int	pi
arccos	entao	interrompa	pos
arcsen	escolha	leia	procedimento
arctan	escreva	literal	quad
arquivo	exp	log	radpgrau
asc	faca	logico	raizq
ate	falso	logn	rand
caracter	fimalgoritmo	maiusc	randi
caso	fimenquanto	mensagem	repita
compr	fimescolha	minusc	se
copia	fimfuncao	nao	sen
cos	fimpara	numerico	senao
cotan	fimprocedimento	numpcarac	timer
cronometro	fimrepita	ou	tan
debug	fimse	outrocaso	verdadeiro
declare	função	para	xou

Em Português Estruturado, só existem três tipos de dados, conforme a tabela abaixo.

TIPO	DESCRIÇÃO
INTEIRO	Representa valores inteiros. Exemplos: 10, 5, -5, -10
REAL ou NUMERICO	Representa valores reais (com ponto separador da parte decimal). Exemplos: 10, 15.5, -14.67
LITERAL ou CARACTERE	Representa texto (seqüência ou cadeia de caracteres) entre aspas duplas. Exemplo "Esta é uma cadeia de caracteres", "B", "1234"

LOGICO	Representa valores lógicos (VERDADEIRO ou FALSO).

6 - Operador de Atribuição

Para "colocar" um valor em uma variável dentro de um algoritmo, utilizamos o ***operador de atribuição*.** O operador de atribuição é representado por uma seta (**<-**) apontando para a esquerda.

Exemplo 6.1

Peso **<-** 78.7 // Este comando atribui à variável Peso o valor 78.7.
Nome **<-** "João da Silva" // Este comando atribui à variável Nome o valor "João da Silva".
Achei **<- FALSO** // Este comando atribui à variável Achei o valor **FALSO**.

É importante lembrar que só se pode atribuir às variáveis valores do mesmo tipo da variável. Assim, o seguinte comando seria inválido:

Exemplo 6.2

VAR
salario: **REAL**
INICIO
salario **<-** "Insuficiente"

Deve estar claro, também, que sempre à esquerda do comando de atribuição deve haver um (e somente um) identificador de variável. Assim, **são incorretos** os seguintes comandos:

Exemplo 6.2 "são incorretos"

2060 **<-** NumeroConta
NumeroAgencia+digitoControle **<-** 2345 + 0
NomeCliente+sobrenome **<-** "João" + "Silva"

7 - Linhas de Comentário

Os comentários são declarações não compiladas que podem conter qualquer informação textual que você queira adicionar ao código-fonte para referência e documentação de seu programa.

Uma Linha

São representados por duas barras normais (//). Todo o texto que você digitar após as duas barras será comentário.

Exemplo 7.1

```
// Este método calcula o fatorial de n...x <- y;
// Inicializa a variável x com o valor de y
```

8 - Comandos de E/S (Entrada/Saída)

Em geral, um programa que faz seu processamento e não tem como mostrar seus resultados é inútil (imagine, por exemplo, uma calculadora que realiza uma infinidade de operações matemáticas, mas não tem um *display* para mostrar os resultados!). Portanto, em algum ponto do algoritmo geralmente deve ocorrer à exibição de valores, e todas as linguagens de programação têm comandos para este fim. Em Português Estruturado

algoritmos usamos o comando **escreva** para isto. A sintaxe desse comando tem a seguinte forma:

```
Escreva (<expressão ou identificador ou constante>, <expressão ou
identificador  ou constante>, ..., <expressão ou identificador ou
constante>)
```

OBS.: No Visualg existem dois comandos escreva com finalidades diferentes quando usado consecutivamente.

Escreval (<expressão ou identificador ou constante>) *//Mostra o primeiro resultado na mesma linha depois em linhas diferentes.*
Escreva (<expressão ou identificador ou constante>) *//Mostra o resultado na mesma linha, mas em colunas diferentes.*

Exemplo 8.1

```
X <- 3.5
Y <- 4
Escreva ("O valor de X é", X)
Escreva (" E o valor de Y é ", Y)
Escreval (" A soma de X e Y é", X+Y)

Escreval ("O valor de X é", X)
Escreval ("E o valor de Y é ", Y)
Escreval ("A soma de X e Y é", X+Y)
```

Faria com que aparecesse na tela:

```
O valor de X é 3.5 E o valor de Y é  4 A soma de X e Y é 7.5
O valor de X é 3.5
E o valor de Y é  4
A soma de X e Y é 7.5
```

Nem todos os dados que um algoritmo manipula são gerados por ele. Um algoritmo (programa) de caixa automático, por exemplo, tem que obter do usuário o número da conta, a senha, a opção de serviço desejada, etc. Assim, deve haver um meio para que sejam digitados (ou fornecidos de outra maneira) dados para o algoritmo. Mais uma vez, todas as linguagens de programação permitem isto, e no nosso Português Estruturado usamos o comando **leia**. A sintaxe deste comando é:

```
Leia (<identificador>)
```

Exemplo 8.2

```
leia  (NumeroConta)
leia (NumeroAgencia)
leia (NomeCliente)
```

Você pode mandar uma mensagem antes para o usuário, assim ele sabe qual é o conteúdo que deve ser colocado, ou seja, digitado.

Exemplo 8.3

```
Escreva ("Digite seu nome: ")
Leia (nome)
Escreva ("Digite sua agencia: ")
Leia (NumeroAgencia)
Escreva ("Digite sua conta: ")
Leia (NumeroConta)
```

Deve estar claro que sempre *à* ***direita*** do comando **leia** haverá um *identificador de variável*. Assim, **são incorretos** os seguintes comandos:

Exemplo 8.4 "são incorretos"

```
leia (NumeroConta+60)
leia (12345)
leia (NomeCliente+Sobrenome)
```

Exercícios na página 39

9 - Construindo os Primeiros Algoritmos: Estruturas seqüenciais

De forma genérica, a construção de um algoritmo se resume às seguintes etapas:

a) entendimento do problema;

b) elaboração da solução algorítmica; e

c) codificação da solução no Português Estruturado;

Geralmente a etapa 2 é a mais complexa, pois depende da engenhosidade e experiência do "construtor".

Exemplo 9.1

Enunciado: Faça um programa que leia dois valores numéricos, e calcule e exiba a sua média aritmética.

Etapa 1

Simples, hein? Dos tempos de escola lembramos que a média aritmética de dois valores é calculada como (a+b)/2, e sendo assim a primeira etapa já está pronta.

Etapa 2

Os dados necessários serão os dois valores, que colocaremos em duas variáveis A e B, do tipo numérico, e uma terceira variável, que chamaremos Média, que armazenará a média aritmética calculada.

Etapa 3

A obtenção dos dados neste programa é simples e direta. Basta pedir ao usuário que digite os valores.

Etapa 4

O processamento aqui é o cálculo da média, usando o método citado acima, na **etapa 1**. O resultado do cálculo será armazenado na variável Média.

Etapa 5

Basta exibir o conteúdo da variável Média.

Solução:

```
1.    Algoritmo "Cálculo de Média Aritmética"
2.    VAR
3.    A,B,Media : REAL
4.    Inicio
5.    Escreva ("Programa que calcula a média aritmética de dois
valores.")
6.    Escreva ("Digite um valor : ")
7.    Leia (A)
8.    Escreva ("Digite outro valor : ")
9.    Leia (B)
10.   Media <- (A+B)/2
11.   Escreva ("A média dos dois valores é : ", Media)
12.   FimAlgoritmo
```

Comentários

Você deve ter notado que colocamos na tela instruções para o usuário usando o comando Escreva. Esta é uma boa técnica de programação, mesmo hoje em dia, com o ambiente do Windows, etc. Da mesma forma, ao imprimir o resultado, não mostramos simplesmente a média, mas explicamos ao usuário o que aquele valor significa.

Como pode ser analisado no tópico anterior todo programa possui uma estrutura seqüencial determinada por um **INÍCIO** e **FIM**. Em um algoritmo, estes limites são definidos com as palavras **Algoritmo** e **FimAlgoritmo**.

Exemplo 9.2

Enunciado: Algoritmo que lê o nome de um aluno, as notas de suas três provas e calcule e exibe a média harmônica das provas.

Etapa 1: a média harmônica de três provas a, b e c é dada pela fórmula $\frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$

Etapa 2: os dados necessários serão o nome do aluno e os valores das provas. O algoritmo limita-se basicamente à própria fórmula.

```
1.    Algoritmo "MediaHarmonica"
2.    VAR
3.    a, b, c, MH: REAL
4.    NOME: CARACTER
5.    inicio
1.    escreva ("Entre com o nome do aluno: ")
2.    leia (nome)
3.    escreval  ("Entre com as notas das três provas")
4.    escreva ("Digite a primeira nota: ")
5.    leia  (a)
6.    escreva ("Digite a segunda nota: ")
7.    leia  (b)
8.    escreva ("Digite a terceira nota: ")
9.    leia  (c)
10.   MH <- 3/(1/a + 1/b +1/c)
11.   escreval ("A média harmônica do aluno: ", NOME, " é ", MH)
12.   FimAlgoritmo
```

Exemplo 9.3

Enunciado: Um algoritmo que lê o valor do raio e calcule a área do círculo correspondente.

Etapa 1: o cálculo da área do círculo é Pi*R^2.

Etapa 2: o dado necessário é o valor do raio, que será lido (colocado) na variável `Raio`.

```
1.    algoritmo "Calcula Área Circulo"
2.    var
3.    Area, Raio: REAL
4.    inicio
5.    Escreval ("Entre com o raio: ")
6.    Leia (Raio)
7.    Area <- Pi*Raio^2
8.    Escreva ("A área do circulo com o raio ", Raio, " é ", Area)
9.    fimalgoritmo
```

Você não precisa declarar o pi, pois já é uma função definida pelo programa Visualg.

Exercícios na página 41

10 - Estrutura Condicional

Na vida real tomamos decisões a todo o momento baseadas em uma situação existente. Em um algoritmo, chamamos esta situação de *condição.* Associada a uma condição, existirá uma alternativa possível de ações.

Exemplo 10.1

"se tiver R$ 10,00 sobrando então irei ao cinema hoje à noite."

A condição nesta frase é "tiver R$ 10,00 sobrando". Ela é uma expressão lógica, pois a pergunta "Tenho R$ 10,00 sobrando?" Pode (tem que) ser respondida com "Sim" ou "Não". ***Lembre-se, então:*** em um algoritmo, toda condição tem que ser uma expressão *lógica*, algo que possa-se pensar como " isto é **VERDADEIRO**" ou "isto é **FALSO**". Se a condição for verdadeira, a ação a ser executada é "irei ao cinema", se a resposta à pergunta "Tenho dinheiro suficiente?" for "Sim". Então, em um algoritmo, as ações são um ou mais comandos que serão realizados apenas se a avaliação da condição resulta **VERDADEIRO**.

Vamos colocar agora a frase do exemplo anterior em outra forma, mais parecida com nosso Português Estruturado:

Exemplo 10.2

se "tiver R$ 10,00 sobrando" **entao**
  "irei ao cinema"
fimse

Veja que grifamos três palavras: **se, entao e fimse**. Elas são muito importantes na estrutura dos comandos de decisão. Como próximo passo, vamos generalizar a estrutura que criamos acima:

```
se <condição> entao

   <ações (uma ou mais) a serem realizadas se a condição for verdadeira>
fimse
```

Para terminar a nossa comparação, devemos lembrar que os comandos de um algoritmo são sempre indispensável, e que o computador só lida com quantidades definidas (ou seja, ele não sabe o que é "ter R$ 10,00 sobrando"). Para aproximar mais nossa frase de um algoritmo, poderemos ter a seguinte forma:

Exemplo 10.3

se Dinheiro >= 10 **entao**
  Ir_ao_Cinema <- **VERDADEIRO**
Fimse

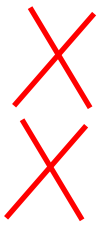
O exemplo acima poderia ser estendido para o caso do sujeito não ter dinheiro sobrando: "se tiver R$ 10,00 sobrando irei ao cinema hoje à noite, mas se não tiver ficarei vendo TV em casa". Neste caso, uma codificação possível para esse algoritmo seria:

Exemplo 10.4

```
se Dinheiro >= 10 entao
   Ir_ao_Cinema <- VERDADEIRO
   Ver_TV <- FALSO
Fimse
se Dinheiro < 10 entao
   Ir_ao_Cinema <- FALSO
   Ver_TV <- VERDADEIRO
Fimse
```

É importante frisar que **sempre** à direita do comando se deverá parecer uma **expressão lógica**, e uma expressão cujo resultado é **VERDADEIRO** ou **FALSO**. Assim, os seguintes comandos são incorretos:

Exemplo 10.5



```
se A <- B entao // É uma atribuição e não uma expressão
   ...
fimse
se A + B entao // É uma expressão aritmética e não uma expressão
   ...
fimse
```

Por outro lado, estão corretos os seguintes comandos:

Exemplo 10.6

```
se (A > B) e (A > C) e (B <> C) entao
   ...
fimse
se nao Achou entao // Correto se Achou foi declarada como logico
   ...
fimse
```

Seja o algoritmo abaixo:

Exemplo 10.7

Faça um Algoritmo para calcular a área de um circulo, fornecido o valor do raio, que deve ser positivo.

```
1.    Algoritmo "Calcula Area do Circulo"
2.    VAR
3.    Area, Raio: Real
4.    inicio
5.    Escreval ("Entre com raio do círculo")
6.    Leia (Raio)
7.    Se Raio > 0 entao
8.       Area <- PI*(Raio^2)
9.       Escreva ("A área do círculo de raio ", Raio, " é ", Area)
10.   fimse
11.   Se Raio <= 0 entao
12.      Escreva ("Raio não pode ser nulo ou negativo!")
13.   fimse
14.   fimalgoritmo
```

Observe que se a condição do primeiro é **verdadeira**, a segunda condição é **falsa** e vice-versa, e o conjunto de instruções a ser executado `se Raio <= 0` (apenas a instrução **escreva ("Raio não pode ser nulo ou negativo!")**) é uma alternativa para a condição `Raio`

> 0. Para expressar isso mais facilmente (e também por questões de eficiência), a maioria das linguagens de programação permite associar um conjunto de instruções a ser executado se a condição do comando resultar em **FALSO**. Em Português Estruturado, a sintaxe para tal é a seguinte:

```
se <condição> entao

   <ações (uma ou mais) a serem realizadas se a condição for verdadeira>

senao

   <ações (uma ou mais) a serem realizadas se a condição for falsa>
fimse
```

Utilizando o **senao**, o algoritmo para calcular a área de um circulo, ficaria assim:

Exemplo 10.8

```
1.    Algoritmo "Calcula Area do Circulo"
2.    VAR
3.    Area, Raio: Real
4.    inicio
5.    Escreval ("Entre com raio do círculo")
6.    Leia (Raio)
7.    Se Raio > 0 entao
8.       Area <- PI*(Raio^2)
9.       Escreva ("A área do círculo de raio ", Raio, " é ", Area)
10.   senao
11.      Escreva ("Raio não pode ser nulo ou negativo!")
12.   fimse
13.   fimalgoritmo
```

Exemplo 10.9

Algoritmo que peça ao usuário a quantia em dinheiro que tem sobrando e sugira, caso ele tenha 10 ou mais reais, que vá ao cinema, e se não tiver, fique em casa vendo TV.

```
1.    Algoritmo "AconselhaPrograma"
2.    Var
3.    Dinheiro: REAL
4.    inicio
5.    Escreval ("*** Serviço Informatizado de Sugestões ***")
6.    Escreva ("Quanto dinheiro você tem sobrando?")
7.    Leia (Dinheiro)
8.    Se Dinheiro >= 10 entao
9.       Escreval ("Vá ao cinema hoje à noite.")
10.   Senao
11.      Escreval ("Fique em casa vendo TV.")
12.   Fimse
13.      Escreva ("Obrigado e volte sempre.")
14.   Fimalgoritmo
```

Escolha...Caso

Em algumas situações é necessário termos várias soluções ligadas a respostas diferentes, neste caso o comando de alternativa simples ou composta não é uma solução prática, isto porque obrigará o programador a escrever muitas linhas de programa, além de ter que criar vários comandos de alternativas compostas e verificar a validade de suas condições para que o comando execute o caminho correto para uma determinada condição. Temos então o comando de alternativa de múltipla escolha.

O funcionamento deste comando obedece a seguinte regra:

```
escolha < expressão-de-seleção >
caso < exp 1 > , < exp 2 >, ... , < exp n >
   < lista-de-comandos-1 >
caso < exp 1 > , < exp 2 >, ... , < exp n >
   < lista-de-comandos-2 >
outrocaso
   < lista-de-comandos-3 >
fimescolha
```

Exemplo 10.10

Um determinado clube de futebol pretende classificar seus atletas em categorias e para isto ele contratou um programador para criar um programa que executasse esta tarefa. Para isso o clube criou uma tabela que continha a faixa etária do atleta e sua categoria. A tabela está demonstrada abaixo:

IDADE CATEGORIA

De 05 a 10 Infantil
De 11 a 15 Juvenil
De 16 a 20 Junior
De 21 a 25 Profissional

Construa um programa que solicite o nome e a idade de um atleta e imprima a sua categoria.

```
1.  Algoritmo "CLASSIFICAÇÃO DE ATLETAS
2.  var
3.  nome, categoria  : caractere
4.  idade : inteiro
5.  inicio
6.  Escreva("Nome do Atleta = ")
7.  Leia (nome)
8.  Escreva("Idade do Atleta = ")
9.  Leia (idade)
10. Escolha idade
11. caso 5,6,7,8,9,10
12.    categoria <- "Infantil"
13. caso 11,12,13,14,15
14.    categoria <- "Juvenil"
15. caso 16,17,18,19,20
16.    categoria <- "Junior"
17. caso 21,22,23,24,25
18.    categoria <- "Profissional"
19. outrocaso
20.    categoria <- "INVALIDO"
21. Fimescolha
22. Escreva ("Categoria = ",categoria)  escreva("Atleta: ",nome," é da categoria: ",categoria)
23. fimalgoritmo
```

11 - Testando o Algoritmo

Um algoritmo, depois de ser elaborado, pode (e deve) ser testado. Para tal, utilizamos um método conhecido como *teste de mesa*. O teste de mesa é como uma simulação de todos os passos, ou seja, entradas, comandos e instruções do algoritmo, a fim de saber se ele chega ao resultado a que se propõe e se a lógica está correta. Para tal, preenche-se uma tabela com valores para as variáveis e segue-se o fluxo de execução do algoritmo, simulando a execução de cada instrução, ou seja, refazendo o que o computador faria ao executar cada instrução. A cada comando simulado (executado), o valor das variáveis na tabela deve ser atualizado. Se, para uma instrução executada, uma ou mais variáveis não ficaram com os valores esperados, há um erro na lógica do algoritmo.

Exemplo 11.1

Para cada variável você deve fazer uma coluna e uma coluna para saída de dados.

Algoritmo	**Teste de Mesa**			
Algoritmo				
Var				
a,b,c: **REAL**	a	b	c	Saída
Inicio	?	?	?	
a <- 5	5	?	?	
b <- 15	5	15	?	
c <- a+b	5	15	20	
escreva (c)	5	15	20	20
a <- 10	10	15	20	
b <- 25	10	25	20	
c <- a+b	10	25	35	
escreva (c)	10	25	35	35
a <- a-b	(10-25)= -15	25	35	
escreva (a)	-15	25	35	-15
a <- 0	0	25	35	
b <- 0	0	0	35	
c <- 0	0	0	0	
Fimalgoritmo				

Exercícios na página 43

12 - Estrutura de Repetição

Nos exemplos e exercícios que vimos até agora sempre foi possível resolver os problemas com uma seqüência de instruções onde todas eram necessariamente executadas uma única vez. Os algoritmos que escrevemos seguiam, portanto, apenas uma seqüência linear de operações. Veja, por exemplo, um algoritmo para ler os nomes e as notas das provas de três alunos e calcular suas médias harmônicas. Uma possível solução seria repetir o trecho de código do algoritmo do **Exemplo 9.2** três vezes.

Exemplo 12.1

Algoritmo que lê os nomes dos alunos de uma turma de três alunos e as notas de suas três provas; o algoritmo calcula e exibe as médias harmônicas das provas de cada aluno.

```
1.    Algoritmo "MediaHarmonica"
2.    VAR
3.    a, b, c, MH: REAL
4.    NOME: caractere
5.    inicio
6.    escreva ("Entre com o nome do aluno: ")
7.    leia (nome)
8.    escreval  ("Entre com as notas das três provas")
9.    escreva ("Digite a primeira nota: ")
10.   leia  (a)
11.   escreva ("Digite a segunda nota: ")
12.   leia  (b)
13.   escreva ("Digite a terceira nota: ")
14.   leia  (c)
15.   MH <- 3/(1/a + 1/b +1/c)
16.   escreval ("A média harmônica do aluno: ", NOME, " é ", MH)
17.   escreva ("Entre com o nome do aluno: ")
18.   leia (nome)
19.   escreval  ("Entre com as notas das três provas")
20.   escreva ("Digite a primeira nota: ")
21.   leia  (a)
22.   escreva ("Digite a segunda nota: ")
23.   leia  (b)
24.   escreva ("Digite a terceira nota: ")
25.   leia  (c)
26.   MH <- 3/(1/a + 1/b +1/c)
27.   escreval ("A média harmônica do aluno: ", NOME, " é ", MH)
28.   escreva ("Entre com o nome do aluno: ")
29.   leia (nome)
30.   escreval  ("Entre com as notas das três provas")
31.   escreva ("Digite a primeira nota: ")
32.   leia  (a)
33.   escreva ("Digite a segunda nota: ")
34.   leia  (b)
35.   escreva ("Digite a terceira nota: ")
36.   leia  (c)
37.   MH <- 3/(1/a + 1/b +1/c)
38.   escreval ("A média harmônica do aluno: ", NOME, " é ", MH)
39.   FimAlgoritmo
```

A solução acima é viável apenas para uma turma de poucos alunos; para uma turma de 40 alunos, a codificação da solução seria por demais trabalhosa. Nesta seção, veremos um conjunto de estruturas sintáticas que permitem que um trecho de um algoritmo (lista de

comandos) seja repetido um determinado número de vezes, sem que o código correspondente tenha que ser escrito mais de uma vez. Em Português Estruturado possui três estruturas de repetição: **repita...ate**, **enquanto...faca** e **para...faca**.

13 - Comando repita...Ate

Nessa estrutura, todos os comandos da lista são executados e uma expressão lógica é avaliada. Isto se repete até que a avaliação da condição resulte em **FALSO**, quanto então o próximo comando a ser executado é o comando imediatamente após o **ate**. Cada repetição da lista de comandos também é chamada de iteração e essa estrutura também é chamada de laço de repetição. Sua forma geral é:

Verdadeiro

```
repita
   <lista de comandos>

ate <expressão lógica ou relacional>
```

Exemplo 13.1

Algoritmo que escreve os números de 1 a 10.

```
1.    algoritmo "DemonstraRepeticao"
2.    VAR
3.    i: INTEIRO
4.    inicio
5.    i<- 1
6.    repita
7.    escreva (i)
8.       i<- i + 1
9.    ate i > 10
10.   fimalgoritmo
```

```
Algoritmo "DemonstraRepeticao"
var
   N : inteiro
inicio
   N <- 1
   repita
      escreva(N)
      N <- N + 1
   ate N > 10
fimalgoritmo
```

No exemplo acima, a variável **i** controla o número de repetições do laço. Normalmente, a variável de controle do laço recebe um valor inicial, é incrementada (ou decrementada) de um valor constante no laço e tem seu valor testado no final do laço. Ao chegar a um determinado valor, o laço é interrompido. A inicialização da variável contadora deve acontecer fora do laço, antes do seu início.

Exemplo 13.2

Algoritmo que lê os nomes dos alunos de uma turma de três alunos e as notas de suas três provas; o algoritmo calcula e exibe as médias harmônicas das provas de cada aluno.

```
1.    Algoritmo "MediaHarmonica"
2.    var
3.    a, b, c, MH, i: real
4.    NOME: caractere
5.    inicio
6.    i <- 1
7.    Repita
8.      escreva ("Entre com o nome do aluno: ")
9.      leia (nome)
10.     escreval  ("Entre com as notas das três provas")
11.     escreva ("Digite a primeira nota: ")
12.     leia  (a)
13.     escreva ("Digite a segunda nota: ")
```

```
14.     leia  (b)
15.     escreva ("Digite a terceira nota: ")
16.     leia  (c)
17.     MH <- 3/(1/a + 1/b +1/c)
18.     escreval ("A média harmônica do aluno: ", NOME, " é ", MH)
19.     i <- i + 1
20.   ate i > 3
21.   FimAlgoritmo
```

Existem diversas maneiras de implementar o mesmo laço, mas todo laço com variável de controle deve conter:

a) inicialização da variável de controle;

b) incremento (aumento do valor da variável de controle) ou decremento (diminuição do valor da variável de controle) da variável de controle; e

c) teste de valor da variável de controle.

Exemplo 13.3

Algoritmo que escreve os números pares de 10 a 2.

```
1.    algoritmo "DecrementoNumerosPares"
2.    var
3.       i: inteiro
4.    inicio
5.    i <- 10
6.    Repita
7.      escreva (i)
8.      i <- i - 2
9.    ate i = 0
10.   Fimalgoritmo
```

Um cuidado fundamental que o construtor do algoritmo deve ter é o de certificar-se que a condição para que sejam mantidas as iterações torne-se, em algum momento, falsa, para que o algoritmo não entre em um laço infinito.

Exemplo 13.4

```
1.    algoritmo “laçoInfinito”
2.    VAR
3.    Contador: numerico
4.    inicio
5.    repita
6.       Contador <- 1
7.       Contador <- Contador + 1
8.    ate Contador = 10
9.    fimalgoritmo
```

No exemplo acima, a execução do algoritmo entra em um laço infinito porque a inicialização da variável **Contador** (instrução `Contador <- 1`) deveria ser feita antes do comando **repita**, ou seja, antes do laço. No exemplo, ela sempre voltará a ser 1 e nunca alcançará o valor 10.

Exemplo 13.5

```
1.  algoritmo "laçoInfinito"
2.  VAR
3.  Soma: numerico
4.  inicio
5.  Soma <- 1
6.  repita
7.     Soma <- Soma + 2
8.  ate Soma = 10
9.  escreva (soma)
10. fimalgoritmo
```

No exemplo acima, a execução do algoritmo entra em um laço infinito porque a variável `Soma` é incrementada de 2 em 2. Como ela começa com o valor 1, ela passará do valor 9 para o 11, ou seja, nunca será igual a 10. Para corrigir o problema, bastaria alterar a condição conforme o exemplo abaixo.

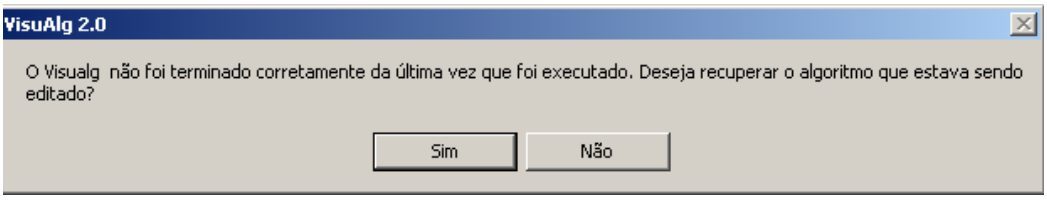
Exemplo 13.6

```
1.  algoritmo "laçoInfinito Consertado"
2.  VAR
3.  Soma: REAL
4.  inicio
5.  Soma <- 1
6.  repita
7.     Soma <- Soma + 2
8.  ate Soma > 10
9.  escreva (soma)
10. fimalgoritmo
```

Lembrete:

Quando ocorrer laço infinito no Visualg você ficará sem comunicação. Procura dar Ctrl+ALT+DEL e selecionar o programa Visualg e clicar no botão finalizar. Não se preocupe com seu algoritmo, pois quando ocorre isso o Visualg salva automaticamente.

Para recuperar seu algoritmo basta abrir Visualg novamente que mostrará uma janela como abaixo, perguntando se deseja recuperar ou não o algoritmo perdido no laço infinito.



Se você sabe que vai ocorrer um laço infinito e mesmo assim quer executar seu algoritmo, procura executar passo a passo pressionando a tecla **F8** para não trancar.

Exemplo 13.7

```
1.    algoritmo "Menu"
2.    VAR
3.    opcao:  inteiro
4.    inicio
5.    repita
6.       escreval  ("Cadastro de clientes")
7.       escreval  ("1-Incluir")
8.       escreval  ("2-Alterar")
9.       escreval  ("3-Consultar")
10.      escreval  ("4-Excluir")
11.      escreval  ("5-Sair")
12.      leia(opcao)
13.   ate opcao = 5
14.   fimalgoritmo
```

No exemplo acima, mostra como é feito um menu de cadastro de clientes, onde mostro as opções primeiro e depois compará-las conforme o usuário digitar;

14 - Comando Enquanto..faca

Na estrutura **enquanto..faca**, a expressão lógica é avaliada e, se ela for verdadeira, a lista de comandos é executada. Isso se repete até que a condição seja falsa. Veja a sua forma geral:

```
enquanto <expressão lógica ou relacional> faca

   <lista de comandos>

fimenquanto
```

A estrutura **enquanto...faca** também é uma estrutura de repetição, semelhante à **repita**. A diferença básica entre as duas estruturas é a posição onde é testada a expressão. Na estrutura **repita**, a condição é avaliada após a execução dos comandos, o que garante que os comandos serão executados pelo menos uma vez. Na estrutura **enquanto**, a expressão é avaliada no início e se o resultado for **FALSO** no primeiro teste, a lista de comandos não é executada nenhuma vez. Essa diferença faz com que em determinadas situações o uso de uma estrutura seja mais vantajoso que o uso da outra. O exemplo a seguir, onde são mostradas duas soluções para um mesmo problema, ilustra essa diferença:

Exemplo 14.1

Algoritmo que lê diversos números positivos e escreve, para cada um, sua raiz quadrada.

Exemplo 14.1

Algoritmo que lê diversos números positivos e escreve, para cada um, sua raiz quadrada.

```
1.  algoritmo "comEnquanto"
2.  var
3.  i: numerico
4.  inicio
5.  leia (i)
6.  enquanto i >=0 faca
7.    escreva (i^0.5)
8.    leia (i)
9.  fimenquanto
10. fimalgoritmo
```

```
1.  algoritmo "comRepita"
2.  var
3.  i: numerico
4.  inicio
5.  repita
6.    leia (i)
7.    se i >=0 entao
8.      escreva (i^0.5)
9.    fimse
10. ate i<0
11. fimalgoritmo
```

No primeiro algoritmo, se o valor lido de **i** for negativo, o algoritmo não deve escrever nada. Com o uso do comando **repita** (segundo algoritmo), para que isso ocorra, um teste do valor deve ser feito antes da escrita.

15 - Comando para..faca

O comando **para...faca** também permite a descrição, dentro de um algoritmo, de uma estrutura de repetição. Sua forma geral é:

```
para <variável de controle> de <valor inicial> ate <valor final> [passo
<incremento>] faca
   <lista de comandos>

fimpara
```

Na estrutura **para..faca**, a variável de controle é inicializada com **<valor inicial>** e no início de cada iteração, seu valor é comparado com **<valor final>**. Se o valor da variável for menor ou igual a **<valor final>**, a lista de comandos é executada e após ser executado o último comando da lista, a variável de controle é incrementada. Isto repete-se até que o valor da variável de controle seja maior que **<valor final>**, quando então é executado o comando imediatamente após a palavra **fimpara**. A instrução **passo** é necessária se o incremento for diferente de 1.

Exemplo 15.1

Um algoritmo que lê escreve os números ímpares de 1 a 1000.

```
1.  para i de 1 ate 1000 passo  2 faca // Incrementa i de 2 em 2
2.     escreva i, " é ímpar"
3.  fimpara
```

A estrutura **para..faca** é uma estrutura de repetição mais completa que as anteriores, pois ela incorpora a inicialização, incremento e teste de valor final da variável de controle. É preferencialmente utilizada em situações em que **sabe-se** previamente o número de repetições a serem feitas. Este número de repetições pode ser uma constante ou estar em uma variável.

A seguir, serão apresentados alguns problemas utilizando estruturas de repetição e desenvolvidas algumas soluções para os mesmos.

Exemplo 15.2

Algoritmo que lê 5 números e escreve todos os que forem positivos.

```
1.    Algoritmo "Positivos"
2.    var
3.    i, numero: inteiro
4.    inicio
5.    para i de 1 ate 5 passo 1 faca
6.       escreval ("Digete um numero")
7.       leia (numero)
8.       se numero>0 entao
9.          escreva (numero)
10.      fimse
11.   fimpara
12.   fimalgoritmo
```

Neste algoritmo são utilizadas duas variáveis, cada uma com uma função bem definida. A variável `i` é usada para controlar o número de repetições e a variável `numero` é utilizada para armazenar cada um dos valores lidos. Ao escrever um algoritmo, é importante ter bem clara a função de cada variável. Como serão lidos 5 números diferentes, a leitura de `numero` deve ser feita dentro do laço.

Exemplo 15.3

Algoritmo que lê um número `N` e escreve todos os números de 1 a `N`.

```
1.    algoritmo "determina o tamanho do laço"
2.    var
3.    i, N: INTEIRO
4.    inicio
5.    leia (N)
6.    para i de 1 ate N faca
7.       escreva (i)
8.    fimpara
9.    fimalgoritmo
```

Vale observar que, como nesse algoritmo é lido apenas um número, sua leitura deve ser feita fora da estrutura de repetição. Note que não possui a sintaxe passo, pois o **passo +1** é definido como padrão.

Lembrete:
O valor I e N do exemplo acima tem que ser inteiro, pois se for declarado como um valor real ou numérico o algoritmo retornara com um erro de sitaxe.

Exercícios na página 44

16 - Variáveis Compostas Homogêneas

A declaração de variáveis, uma a uma, é suficiente para a codificação algorítmica da solução de uma ampla gama de problemas, mas é insuficiente para resolver um grande número de problemas computacionais. Imagine, por exemplo, como faríamos para construir um algoritmo, que lesse os nome de 500 pessoas e imprimisse um relatório destes mesmos nomes, mas ordenados alfabeticamente. Não seria uma tarefa simples, pois teríamos que definir 500 variáveis do tipo literal, como é mostrado abaixo:

Exemplo 16.1

```
1.  algoritmo "Inviável"
2.  var
3.  nome1, nome2, nome3, nome4, nome5,..., nome499, nome500: literal
4.  inicio
5.  leia (nome1,nome2,...,nome500)
6.  ...
7.  Fimalgoritmo
```

Considere o tamanho do algoritmo, e o trabalho braçal necessário para construí-lo. Para resolver problemas como este, e outros, existem as ***variáveis indexadas***. A declaração de uma variável indexada corresponde, na verdade, à declaração de várias variáveis cujo identificador difere apenas por um *índice*. O índice corresponde a um valor numérico começando por 1. Cada variável indexada pode receber valores no decorrer do algoritmo como se fosse uma variável comum.

16.a Variáveis Indexadas Unidimensionais (Vetores)

Variáveis indexadas com uma única dimensão, também conhecidas como ***vetores***, são referenciadas por um único índice. A sintaxe para declaração é:

```
<identificador> : vetor [<tamanho>] de < tipo >

Tamanho [VI..VF]=> Vi= Valor inicial do índice e VF valor Final do índice.
```

Exemplo 16.2

```
8.  IDADE: VETOR [1..5] DE INTEIRO
9.  NOMES: VETOR [1..5] DE CARACTERE
```

A declaração acima corresponde à declaração de 10 variáveis: nomes[1], nomes[2], nomes[3], nomes[4], nomes[5], idades[1], idades[2], idades[3], idades[4] e idades[5].

Para se atribuir um valor a um elemento do vetor devemos utilizar o seguinte padrão:

```
< identificador>[<posição>] <- <valor>
```

Exemplo 16.3

```
1.  nomes[1] <- “João da Silva”
2.  idades[1] <- 35
3.  nomes[3] <- “Maria Aparecida”
4.  idades[3] <- idades[1]
5.  i <- 5
6.  idades[i] <- 45
```

Exemplo 16.4

Algoritmo que lê um vetor NUMERO de 6 posições e o escreve. A seguir, ele conta quantos valores de NUMERO são negativos e escreva esta informação.

```
1.  Algoritmo "vetores"
2.  VAR
3.  NUMERO: VETOR [1..6] DE REAL
4.  I, conta_neg: INTEIRO
5.  inicio
6.  conta_neg <- 0
7.  para i de 1 ate 6 faca
8.    leia (NUMERO[i])
9.    se NUMERO[i] < 0 entao
10.     conta_neg <- conta_neg + 1
11.   fimse
12. fimpara
13. para i de 1 ate 6 faca
14.   escreval (NUMERO[i])
15. fimpara
16. escreva ("Total de números negativos: ", conta_neg)
17. fimalgoritmo
```

16.b Variáveis Indexadas Bidimensionais (Matrizes)

Variáveis indexadas com duas dimensões, também conhecida como ***matrizes***, são referenciadas por dois índices, cada qual começando por 1. A sintaxe para declaração é:

```
<identificador> : vetor [<tamanho1>,<tamanho2>] de < tipo >

Tamanho [VI..VF]=> Vi= Valor inicial do índice e VF valor Final do índice.
```

Exemplo 16.5

```
PESSOAS: VETOR [1..2,1..3] DE CARACTERE
```

A declaração acima corresponde à declaração de 6 variáveis: PESSOAS[1,1], PESSOAS [1,2], PESSOAS[1,3], PESSOAS[2,1], PESSOAS[2,2], e PESSOAS [2,3].

Para se atribuir um valor a um elemento do vetor devemos utilizar o seguinte padrão:

```
< identificador>[<posição 1>,<posição 2>] <- <valor>
```

Exemplo 16.6

```
PESSOAS[1,3]<- "Tonet"
```

Exemplo 16.7

Algoritmo que lê uma matriz vê Valores(3,3) e calcula as somas:
a) da linha 3 de Valores;
b) da coluna 2 de Valores;
c) da diagonal principal;
d) da diagonal secundária; e
e) de todos os elementos da matriz.

```
1.    Algoritmo "Matriz"
2.    VAR
3.    VALORES : VETOR [1..3,1..3] DE REAL
4.    somaLinha3, somaColuna2, somaDiagPrinc, somaDiagsecu, somaTudo: REAL
5.    i, j: INTEIRO //os índice sempre inteiro
6.    inicio
7.    somaLinha3 <- 0
8.    somaColuna2 <- 0
9.    somaDiagPrinc <- 0
10.   somaDiagsecu <- 0
11.   somaTudo <- 0
12.   Para i de 1 ate 3 faca
13.      Para j de 1 ate 3 faca
14.         Escreva("Digite um valor para a matriz")
15.         Leia (VALORES[i,j])
16.         somaTudo <- VALORES[i,j] + somaTudo
17.         se i=3 entao
18.            somaLinha3 <- VALORES[i,j]+ somaLinha3
19.         fimse
20.         se j=2 entao
21.            somaColuna2 <- VALORES[i,j]+ somaColuna2
22.         fimse
23.         se i=j entao
24.            somaDiagPrinc <- VALORES[i,j]+ somaDiagPrinc
25.         fimse
26.         se j=4-i entao
27.            somaDiagsecu <- VALORES[i,j]+ somaDiagsecu
28.         fimse
29.      fimpara
30.   fimpara
31.   Para i de 1 ate 3 faca
32.      para j de 1 ate 3 faca
33.         escreval (VALORES[i,j])
34.      fimpara
35.   fimpara
36.   escreval ("Soma de todos os elementos é ", somaTudo)
37.   escreval ("Soma dos elementos da linha 3 é ", somaLinha3)
38.   escreval ("Soma dos elementos da coluna 2 é ", somaColuna2)
39.   escreval ("Soma dos elementos da diagonal principal é ",
somaDiagPrinc)
40.   escreval ("Soma dos elementos da diagonal secundária é ",
somaDiagsecu)
41.   fimalgoritmo
```

Vetores Exercícios na página 45
Matriz Exercícios na página 46

17 - Subalgoritmos

São trechos de algoritmos que efetuam um ou mais cálculos determinados. Ao invés de escrever-se um algoritmo grande, escrevem-se vários algoritmos menores, os quais, não isoladamente, mas em conjunto, resolvem o problema proposto. É conveniente utilizá-los quando uma determinada tarefa é efetuada em diversos lugares no mesmo algoritmo. Ao invés de escrever-se um trecho diversas vezes, escreve-se um sub-algoritmo e chama-se-o diversas vezes.

- Eles reduzem o tamanho do algoritmo.
- Facilitam a compreensão e visualização do algoritmo.
- São declarados no início do algoritmo e podem ser chamados em qualquer ponto após sua declaração.
- Eles podem ser Funções que retorna algum valor ou Procedimento (Subrotina) que não retorna nada.

17.A Funções

Uma função é um instrumento (Estático) que tem como objetivo retornar um valor ou uma informação. A chamada de uma função é feita através da citação do seu nome seguido opcionalmente de seus argumentos iniciais entre parênteses. As funções podem ser predefinidas pela linguagem ou criadas pelo programador de acordo com o seu interesse.

Funções Predefinidas do Visualg

O visulag vem com bibliotecas de funções predefinidas que você pode utilizar em seus programas.

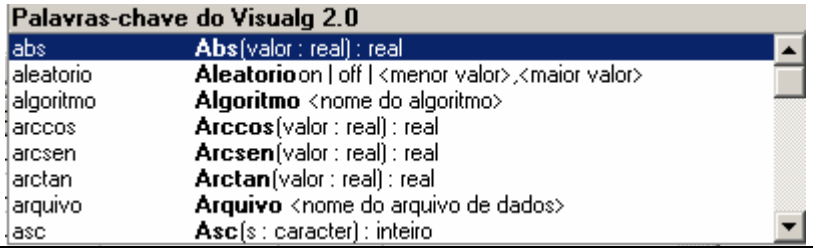
Veja a tabela abaixo:

FUNÇÃO	DESCRIÇÃO
Abs (valor : real) : real	Valor absoluto
Arccos (valor : real) : real	Arco cosseno
Arcsen (valor : real) : real	Arco seno
Arctan (valor : real) : real	Arco tangente
Asc (s : caracter) : inteiro	Retorna o código ASCII
Compr (c : caracter) : inteiro	Retorna a dimensão do caractere
Copia (c : caracter , posini, posfin : inteiro) : caracter	Copia um determinado trecho do caractere
Cos (valor : real) : real	Cosseno
Cotan (valor : real) : real	Co-tangente
Exp (<base>,<expoente>)	Potenciação
Grauprad (valor : real) : real	Converte grau para radiano
Int (valor : real) : inteiro	Converte o valor em inteiro
Log (valor : real) : real	Logaritmo de base 10
Logn (valor : real) : real	Logaritmo natural (ln)

Maiusc (c : caracter) : caracter	Converte em Maiúscula
Minusc (c : caracter) : caracter	Converte em Minúscula
Numpcarac (n : inteiro ou real) : caracter	Converte um numero inteiro ou real para caractere
Pi : real	Valor Pi
Pos (subc, c : caracter) : inteiro	Retorna a posição do caractere.
Quad (valor : real) : real	Elevado quadrado
Radpgrau (valor : real) : real	Converte Radiano para grau.
Raizq (valor : real) : real	Raiz quadrada
Rand : real	Gerador de números aleatórios entre 0 e 1
Randi (limite : inteiro) : inteiro	Gerador de números inteiros aleatórios com um limite determinado
Sen (valor : real) : real	Seno
Tan (valor : real) : real	Tangente

DICA:

Pressionando **(CTRL+J)** o visualg mostra uma Lista de funções predefinidas, a sua utilização é muito simples basta selecionar a função desejada e dar um Enter, depois é só passar os parâmetros desejados.



Exemplo 17.1

```
1.    Algoritmo "RETORNA O SOBRENOME"
2.    var
3.    nome, sobrenome : Caractere
4.    quant_caracteres, local_espcao : INTEIRO
5.    inicio
6.    nome <- "Bruno Tonet"
7.    quant_caracteres <- Compr (nome)
8.    local_espcao <- POS (" ",nome)
9.    sobrenome <- Copia (nome, local_espcao + 1 , quant_caracteres)
10.   Escreva("Seu sobrenome é  ", sobrenome)
11.   fimalgoritmo
```

Exemplo 17.2

```
1.    Algoritmo "RETORNA UM VALOR INTEIRO"
2.    var
3.    valorReal : REAL
4.    valorInteiro : INTEITO
5.    inicio
6.    valorReal <- 5.87978098980980989089898
7.    valorInteiro <- INT(valorReal)
8.    Escreva("Valor inteiro ", valorInteiro)
9.    fimalgoritmo
```

Criando Funções

A criação de uma Função deve ser declarada, com os demais objetos, no início do programa. Este tipo de subalgoritmo sempre retornam um e apenas um valor ao algoritmo que lhe chamou. Cada função tem associada ao seu valor de retorno um tipo explícito. Da mesma maneira com que os parâmetros são fixos para todas as chamada o retorno também é fixo.

```
Algoritmo "<nome do algoritmo>"
var
   <declaração de variáveis globais>
<definição da função>
inicio
   < lista de comandos>
fimalgoritmo
```

Sintaxe da Função

```
funcao <identificador> ([var]<parâmetros>) <tipo de retorno>

var

<declaração de variáveis locais>

inicio

<lista de comandos>

retorne <variável de retorno>
fimfuncao
```

Identificador: Nome da função.

Passagem de parâmetros por referência: utiliza-se a construção **VAR** antes dos identificadores para indicar a passagem por referência. Os identificadores são separados por vírgula.

Parâmetros: Entre um mesmo tipo de dados são separados por vírgula. Entre tipos de dados a separação é feita com ponto-e-vírgulas '**;**'.

Tipo de retorno da função: Real, Inteiro, Lógico ou Caractere.

Declaração de variáveis locais: idêntica a declaração de variáveis globais. As variáveis declaradas localmente tem validade dentro do escopo da função.

Retorne: local onde é colocado a variável de retorno.

Cuidados

Sempre declare as variáveis globais antes da função.
A função sempre fica dentro do escopo Algoritmo e Fim Algoritmo.
Procure não Declarar variáveis globais com o mesmo nome das variáveis da função.

Exemplo 17.3

```
1.    ALGORITMO "Funções Personalizadas"
2.    var
3.    Valor_1,Valor_2, soma: real
4.
5.    FUNCAO FSoma(Recebe_valor1, Recebe_valor2: Real):Real
6.    var
7.    total : real
8.    Inicio
9.    total<-Recebe_valor1+Recebe_valor2
10.   retorne total
11.   fimfuncao
12.
13.   INICIO
14.   Escreva ("Valor_1 : ")
15.   LEIA (Valor_1)
16.
17.   Escreva ("Valor_2 : ")
18.   LEIA (Valor_2)
19.   soma<-FSoma(Valor_1,Valor_2)
20.
21.   ESCREVA ("Soma das vaiáveis é ", soma)
22.   FIMALGORITMO
```

17.B Procedimento (Sub_rotinas)

Sintaxe Procedimento:

```
procedimento <identificador> ([var]<parâmetros>)

var

<declaração de variáveis locais>

inicio

<lista de comandos>
fimprocedimento
```

Identificador: Nome do procedimento.

Passagem de parâmetros por referência: utiliza-se a construção VAR antes dos identificadores para indicar a passagem por referência. Os identificadores são separados por vírgula.

Parâmetros: Entre um mesmo tipo de dados são separados por vírgula. Entre tipos de dados a separação é feita com ponto-e-vírgulas '**;**'.

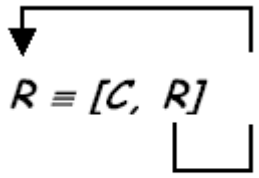
Exemplo 17.4

```
1.    ALGORITMO "Procedimento"
2.
3.    var
4.    A,B,C,D,CONT,AUX:Inteiro
5.
6.    Procedimento TROCA(var x, y: inteiro)
7.    var
8.    Aux : inteiro
9.    INICIO
10.   Aux <- x
11.   x <- y
12.   y <- Aux
13.   FIMProcedimento
14.
15.   INICIO
16.   LEIA (A,B,C,D)
17.   Enquanto NAO((A<=B) e (B<=C) e (C<=D)) faca
18.      se (D<C) entao
19.       TROCA(D,C)
20.      FIMSE
21.      SE C<B ENTAO
22.         TROCA(C,B)
23.      FIMSE
24.      SE B<A ENTAO
25.         TROCA(A,B)
26.      FIMSE
27.   FIMENQUANTO
28.   ESCREVA (A," ",B," ",C," ",D)
29.   FIMALGORITMO
```

Exercícios na página 47

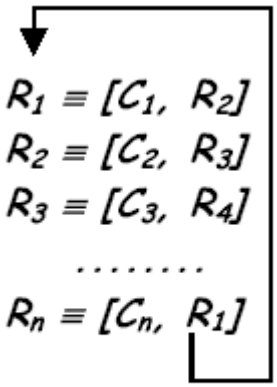
18 - Funções definidas recursivamente

Um algoritmo que para resolver um problema divide-o em subprogramas mais simples, cujas soluções requerem a aplicação dele mesmo, é chamado recursivo, seja de forma direta ou indireta. Em geral, uma rotina recursiva R pode ser expressa como uma composição formada por um conjunto de comandos **C** (que não contém chamadas a **R**) e uma chamada (recursiva) à rotina **R**:



Recursão direta

Entretanto, pode-se ter também uma forma indireta de recursão, na qual as rotinas são conectadas através de uma cadeia de chamadas recursivas que acaba retornando a primeira que foi chamada:



Recursão indireta

Para que esteja bem definida, uma função recursiva deve possuir as seguintes propriedades:

(1) Deve haver certos argumentos, chamando valores básicos, para os quais a função não se refere a ela mesma.
(2) Sempre que a função se refere a ela mesma o argumento deve estar relacionado a um valor básico e/ou a um valor anterior.

Vejamos um exemplo clássico para esclarecermos o conceito: calculo do fatorial de um número. A definição de fatorial é:

F(n) = 1 se n = 0 ou n = 1;

F(n) = n.F(n-1), se n>1.

onde n é um numero inteiro positivo. Uma propriedade (facilmente verificável) dos fatoriais é que:

n! = n . (n-1)!

Esta propriedade é chamada de propriedade recursiva: o fatorial de um numero pode ser calculado através do fatorial de seu antecessor. Ora, podemos utilizar esta propriedade para escrevermos uma rotina recursiva para o calculo de fatorial. Veja:

F(4) = 4.F(4-1)

F(3) = 3.F(3-1)

F(2) = 2.F(2-1)

F(1) = 1.F(1-1)

F(0) = 1 imagina a função subindo, pois encontrou F(0) = 1

F(1) = 1.1

F(2) = 2.1

F(3) = 3.2

F(4) = 4.6

resposta é 24.

Qual é a lógica neste problema?

4x3x2x1=24

Se fosse nº 6 qual seria a resposta?

6x5x4x3x2x1=720

Vamos passar isso para o Computador

Os algoritmos recursivos têm em geral a forma seguinte:

- caso de base (base de recursão), onde o problema é resolvido diretamente (sem chamada recursiva)
- caso geral, onde o problema é resolvido com uma chamada recursiva
- caso geral onde o tamanho do problema é menor a cada chamada

Esquematicamente, os algoritmos recursivos têm a seguinte forma:

se "condicao para o caso de base" ***entao***
    resolucao direta para o caso de base
senao
    uma ou mais chamadas recursivas
fimse

Um algoritmo recursivo pode ter um ou mais casos de base e um ou mais casos gerais. E para que o algoritmo termine, as chamadas recursivas devem convergir em direção ao caso de base, senão o algoritmo não terminará jamais. Convergir significa ter uma parte menor do problema para ser resolvido.

```
F(4) = 4.F(4-1)
        F(3) = 3.F(3-1)
                F(2) = 2.F(2-1)
                        F(1) = 1.F(1-1)
                                F(0) = 1 ---------- Caso Base
                        F(1) = 1.1
                F(2) = 2.1
        F(3) = 3.2
F(4) = 4.6
```

Exemplo 18.1

```
30.   Algoritmo "Função Recursiva"
31.   var
32.   A, Fatorial: Inteiro
33.
34.   Funcao Fat (x:Inteiro):Inteiro
35.   inicio
36.   se x=0 entao
37.      retorne  1
38.   senao
39.      retorne x * Fat (x-1)
40.   Fimse
41.   FimFuncao
42.
43.   inicio
44.   Leia (A)
45.   Fatorial <- Fat (A)
46.   Escreva ("Fatorial ", A, " é ", Fatorial)
47.   FimAlgoritmo
```

Vantagens da Recursão

Simplifica a solução de alguns problemas;
Geralmente, um código com recursão é mais conciso;
Caso não seja usada, em alguns problemas, é necessário manter o controle das variáveis manualmente.

Desvantagens da Recursão

Funções recursivas são mais lentas que funções iterativas, pois muitas chamadas consecutivas a funções são feitas;
Erros de implementação podem levar a estouro de pilha. Isto é, caso não seja indicada uma condição de parada, ou se esta condição nunca for satisfeita, entre outros.

Exercícios na página 49

19 - Referências

http://www.apoioinformatica.inf.br/
http://www.consiste.dimap.ufrn.br/~david/
http://www.inf.pucrs.br/%7Eegidio/algo1/
http://dein.ucs.br/Disciplinas/sis218-algoritmos/2003-2/sis218d/cronog_algo.html
http://www.inf.ufpr.br/info/
http://www.angelfire.com/bc/fontini/algoritm.html
apostila de lógica de programação "criação de algoritmos e programas" professor renato da costa

Gostaria de agradecer pessoalmente o criador do Visualg Professor Cláudio Morgado de Souza por der dado apoio a este Manual e tirado algumas dúvidas do programa.

Mande um e-mail: btonet@ucs.br para receber todas as soluções dos exercícios
Visite a página : http://dein.ucs.br/napro

Capitulo 1 – 8 Exercícios

1) Escreva as expressões abaixo na forma na sintaxe do Português Estruturado.

A. $a + \frac{b}{c} =$

B. $\frac{2x^2 - 3x^{(x+1)}}{2} + \frac{\sqrt{x+1}}{x} =$

C. $2h - \left(\frac{45}{3x} - 4h(3-h)\right)^{22k} =$

D. $\frac{\sqrt{-6^x + 2y}}{3^9} =$

2) Escreva as Expressões da forma convencional.

A. a + b + ((34+e*9)/u-89 ^ (1/2)) =

B. 12+1/((4*a)/45) ^ (1/2) =

C. ((a+x) ^ (2+w)-3a)/2 =

D. (12*x)/(36-9 ^ y) =

4) Resolva as expressões lógicas, determinando se a expressão é verdadeira ou falsa:

A. 2>3=

B. (6<8)ou(3>7)=

C. não (2<3)=

D. (5>=6 ou 6<7 ou não(a+5-6=8) {onde a = 5}

E. (34>9 e 5+u = 34) ou (5=15/3 e 8>12) = ((u = 29) e 8>12) {onde u = 29}

5) Classifique os conteúdo das variáveis abaixo de acordo com seu tipo, assinalando com N os dados numéricos, com L os lógicos, com C os literais.

( ) 0	(C) "abc"	(C) "João"
( ) 5.7	( ) 1012	(L) **FALSO**
( ) -49	( ) +342	( ) 569
(C) "Lucas"	(C) "VERDADEIRO"	( ) 0.00001
(L) **VERDADEIRO**	( ) -545	(C) " 444 "

6) Assinale com um X os nomes de variáveis válidos.

(X) abc	( ) 3abc	(X) a
( ) 123a	( ) –a	(X) acd1
( ) -_ad	( ) A&a	( ) guarda-chuva
(X) A123	(X) Aa	(X) guarda_chuva
( ) ABC DE	( ) etc.	(X) b316
( ) leia	( ) enquanto	( ) escreva

7) Assinalar os comandos de atribuição considerados inválidos:

```
var
NOME, COR, TESTE, DIA: caracter
SOMA, NUM: inteiro
Salario: real
X: lógico
```

a. ( ) NOME <- “5”
b. (X) SOMA <- NUM + 2 * X
c. (X) TESTE <- SOMA
d. ( ) NUM <- SOMA
e. ( ) COR <- "PRETO"
f. (X) X <- X + 1
g. (X) NUM <- "*ABC*"
h. ( ) DIA <- "seGUNDA"
i. (X) SOMA + 2 <- NUM
j. ( ) X <- (NOME = COR)
k. ( ) salário <- 5.000 salario <- 5.000
l. ( ) salário <- 150 salario <- 150
m. (X) salário <- “insuficiente” salario <- "insuficiente"

8) Quais os valores armazenados em SOMA, NOME e TUDO, supondo-se que NUM, X, COR, DIA, TESTE e TESTE2 valem, respectivamente, 5, 2, "AZUL", "TERÇA", FALSO e VERDADEIRO?

A. NOME <- DIA
b) SOMA <- (NUM^2/X) + (X + 1)
c) TUDO <- NÃO ((TESTE OU TESTE2) E (X <> NUM))

NOME receberá "TERÇA"
SOMA receberá 15,5
TUDO receberá FALSO

9) Analise o seguinte algoritmo e descreva o que ele faz.

```
1.  Algoritmo "PrimeiroAlgoritmo"
2.  var
3.  NOTA1, NOTA2, NOTA3, NOTA4, MEDIA: real
4.  NOME: caracter
5.  inicio
6.      leia (NOME)
7.      leia (NOTA1)
8.      leia (NOTA2)
9.      leia (NOTA3)
10.     leia (NOTA4)
11.     MEDIA <- (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4;
12.     escreva (NOME, " obteve ", MEDIA)
13. fimalgoritmo
```

10) No seguinte algoritmo existem erros? Em caso afirmativo, onde?

```
1.  algoritmo "Teste"
2.  var
3.  Maria: caracter
4.  idade: numerico
5.  _letra: literal
6.  Maria: real
7.  1alt:   caracter
8.  peso :  tonelada
9.  Fernando literal
10. inicio
11.     leia (nome)
12.     leia idade
13.     escreva (idade)
14.     dade = 678
15.     leia "letra"
16.     leia ABC
17.     escreva (letra)
18.     letra <- A
19. fimalgoritmo
```

Capitulo 9 Exercícios

Exercício 1

☺ Escrever um algoritmo que lê 3 valores - a, b e c - e calcula:

a) A área do trapézio que tem a como a base maior, b como base menor e c como altura

$$área\,do\,trapézio = \frac{(base\,maior + base\,menor)}{2} * altura$$

b) A área do quadrado que tem o valor da variável b como lado

$$área\,do\,quadrado = lado^2$$

c) A área da superfície de um cubo que tem c por aresta

$$área\,do\,cubo = 6 * aresta^2$$

Exercício 2

☺ Faça um algoritmo que determine o volume de uma caixa dágua cilíndrica, sendo que o raio e a altura devem ser fornecidos (lidos pelo teclado).

V = PI * Raio^2 * Altura

Exercício 3

😐 Escrever um algoritmo que lê o nome de um funcionário, o número de horas trabalhadas, o valor que recebe por hora e o número de filhos. Com estas informações, calcular o salário deste funcionário, sabendo que para cada filho, o funcionário recebe 3% a mais, calculado sobre o salário bruto.

Exercício 4

😐 Escrever um algoritmo que lê o nome de um vendedor, o seu salário fixo, o total de vendas por ele efetuadas e o percentual que ganha sobre o total de vendas. Calcular o salário total do vendedor. Escrever o número do vendedor e seu salário total.

Exercício 5

☹ Faça um algoritmo que leia o nome de um piloto, uma distância percorrida em km e o tempo que o piloto levou para percorrê-la (em horas). O programa deve calcular a velocidade média - **Velocidade = Distância / Tempo** - em km/h, e exibir a seguinte frase:

`A velocidade média do <nome do piloto> foi <velocidade media calculada> km/h.`

DESAFIO 1

☠ Em uma pizzaria, cada tulipa de chopp custa R$ 0,80 e uma pizza mista grande custa R$10,00 mais R$1,50 por tipo de cobertura pedida (queijo, presunto, banana, etc.). Uma turma vai à pizzaria e pede uma determinada quantidade de "chopps" e uma pizza grande com uma determinada quantidade de coberturas. faça um algoritmo que calcule e conta e, sabendo quantas pessoas estão à mesa, quanto que cada um deve pagar (não esqueça os 10% do garçom).

DESAFIO 2

☠ Escreva um algoritmo que calcule o número de notas e de moedas que deve ser dado de troco para um pagamento efetuado. O algoritmo deve ler o valor a ser pago e o valor efetivamente pago. Supor que o troco seja dado em notas de 50, 20, 10, 5, 2 e 1 real.

Capitulo 10 e 11 Exercícios

Exercício 1

☺ Escreva um programa que leia um número inteiro. Se o número lido for positivo, escreva uma mensagem indicando se ele é par ou ímpar. Se o número for negativo, escreva a seguinte mensagem "Este número não é positivo".

Exercício 2

☺ Faça um algoritmo que receba o valor do salário de uma pessoa e o valor de um financiamento pretendido. Caso o financiamento seja menor ou igual a 5 vezes o salário da pessoa, o algoritmo deverá escrever "Financiamento Concedido"; senão, ele deverá escrever "Financiamento Negado". Independente de conceder ou não o financiamento, o algoritmo escreverá depois a frase "Obrigado por nos consultar."

Exercício 3

😐 Fazer um algoritmo que escreva o conceito de um aluno, dada a sua nota. Supor notas inteiras somente. O critério para conceitos é o seguinte:

Nota	**Conceito**
nota inferiores a 3	conceito E
nota de 3 a 5	conceito D
notas 6 e 7	conceito C
notas 8 e 9	conceito B
nota 10	conceito A

Exercício 4

☹ A empresa XYZ decidiu conceder um aumento de salários a seus funcionários de acordo com a tabela abaixo:

SALÁRIO ATUAL	ÍNDICE DE AUMENTO
0 – 400	15%
401 – 700	12%
701 – 1000	10%
1001 – 1800	7%
1801 – 2500	4%
ACIMA DE 2500	SEM AUMENTO

Escrever um algoritmo que lê, para cada funcionário, o seu nome e o seu salário atual. Após receber estes dados, o algoritmo calcula o novo salário e escreve na tela as seguintes informações:
<nome do funcionário> <% de aumento> <salário atual> <novo salário>

Desafio

☠ Faça um programa que lê 4 valores *I, A, B* e *C* onde *I* é um número inteiro e positivo e *A*, *B*, e *C* são quaisquer valores reais. O programa deve escrever os valores lidos e:

- se *I* = 1, escrever os três valores *A, B* e *C* em ordem crescente;
- se *I* = 2, escrever os três valores *A*, *B* e *C* em ordem decrescente;
- se *I* = 3, escrever os três valores *A*, *B*, e *C* de forma que o maior valor fique entre os outros dois;
- se I não for um dos três valores acima, dar uma mensagem indicando isto.

Capitulo 12 e 15 Exercícios

Exercício 1

☺ Escrever um algoritmo que lê um número desconhecido de valores, um de cada vez, e conta quantos deles estão em cada um dos intervalos [0,25], (25,50], (50,75], (75,100].

Exercício 2

☺ Escrever um algoritmo que leia informações sobre um grupo de 250 pessoas e calcule alguns dados estatísticos. Para cada pessoas do grupo deve ler o nome da pessoa, a altura, o peso e o sexo ("F" para feminino e "M" para o masculino). Calular e escrever:
A quantidade total de homens e mulheres e o percentual de cada.
A média de peso das pessoas (somatório dos pesos de todas as pessoas pela quantidade de pessoas)
O nome da pessoa mais alta.

Exercício 3

😐 Escrever um algoritmo que gera e escreve os 4 primeiros números perfeitos. Um número perfeito é aquele que é igual à soma dos seus divisores. Ex: 6 = 1+2+3, 28 = 1+2+4+7+14.

Exercício 4

😐 Escrever um algoritmo que lê um número não determinado de valores para m, todos inteiros e positivos, um de cada vez. Se m for par, verificar quantos divisores possui e escrever esta informação. Se m for ímpar e menor do que 12 calcular e escrever o fatorial de m. Se m for ímpar e maior ou igual a 12 calcular e escrever a soma dos inteiros de 1 até numero lido.

Exercício 5

☹ Faça um algoritmo que gere uma tabela com os números de 1 a 10 e mostre o seu quadrado, cubo, fatorial, número de divisores e uma mensagem dizendo se o número é primo ou não. A cada 20 linhas deve ser escrito o cabeçalho novamente:

"Número	Quadrado	Cubo	Fatorial	Divisores	Primo"
1	1	1	1	1	Sim
2	4	8	2	2	Sim

Desafio

☠ Escrever um algoritmo que lê um conjunto não determinado de pares de valores a, b, todos inteiros e positivos, e para cada par lido, obtém o M.D.C. e o M.M.C. de a,b, escrevendo-os juntamente com os valores lidos.

Capitulo 16 Vetor Exercícios

Exercício 1
☺ Escreva um algoritmo que lê um vetor A(10) e escreva a posição de cada elemento igual a 10 deste vetor.

Exercício 2
☺ Escrever um algoritmo que lê um vetor X(100) e o escreve. Substitua, a seguir, todos os valores nulos de X por 1 e escreva novamente o vetor X.

Exercício 3
☺ Faça um algoritmo que leia 100 valores e os escreva na ordem contrária à que foram digitados.

Exercício 4
😐 Escrever um algoritmo que lê um vetor N(80) ~~e o escreve~~. Encontre, a seguir, o menor elemento e a sua posição no vetor N e escreva: "O menor elemento de N é = ... e a sua posição é ... ".

Exercício 5
😐 Escrever um algoritmo que lê um vetor N(20) ~~e o escreve~~. Troque, a seguir, o 1* elemento com o último, o 2* com o penúltimo, etc até o 10* com o 11* e escreva o vetor N assim modificado.

Exercício 6
😐 Escreva um algoritmo que gera os 10 primeiros números primos acima de 100 e os armazena em um vetor **X(10)** escrevendo, no final, o vetor **X**.

Exercício 7
☹ Escrever um algoritmo que lê um vetor G(13) que é o gabarito de um teste de loteria esportiva, contendo os valores 1(coluna 1), 2(coluna 2) e 3(coluna do meio). Ler, a seguir, para cada apostador, o número de seu cartão e um vetor Resposta R(13). Verificar para cada apostador o número de acertos e escrever o número do apostador e seu número de acertos. Se tiver 13 acertos, acrescentar a mensagem: "GANHADOR, PARABENS".

Exercício 8
☹ Escrever um algoritmo que lê um vetor A(15) e o escreve. Ordene a seguir os elementos de A em ordem crescente e escreva novamente A.

☠ **DESAFIO**
Escrever um algoritmo que lê, para um vetor V(30), vinte valores que ocuparão as 20 primeiras posições do vetor V. Ordene, a seguir, os elementos de V em ordem crescente. Leia, a seguir 10 valores A, um por vez, e insira-os nas posições adequadas do vetor V, de forma que o mesmo continue ordenado em ordem crescente. Escreva o vetor V assim formado.

Capitulo 16 Matriz Exercícios

Exercício 1

☺ Escrever um algoritmo para armazenar valores inteiros em uma matriz (5,6). A seguir, calcular a média dos valores pares contidos na matriz e escrever seu conteúdo.

Exercício 2

☺ Escrever um algoritmo para ler uma matriz (7,4) contendo valores inteiros (supor que os valores são distintos). Após, encontrar o menor valor contido na matriz e sua posição.

Exercício 3

☺ Escreva um algoritmo que lê uma matriz M(5,5) e calcula as somas:

a) da linha 4 de M.
b) da coluna 2 de M.
c) da diagonal principal.
d) da diagonal secundária.
e) de todos os elementos da matriz.
f) Escreva estas somas e a matriz.

Exercício 4

😐 Escrever um algoritmo que lê uma matriz M(5,5) e cria 2 vetores SL(5), SC(5) que contenham respectivamente as somas das linhas e das colunas de M. Escrever a matriz e os vetores criados.

Exercício 5

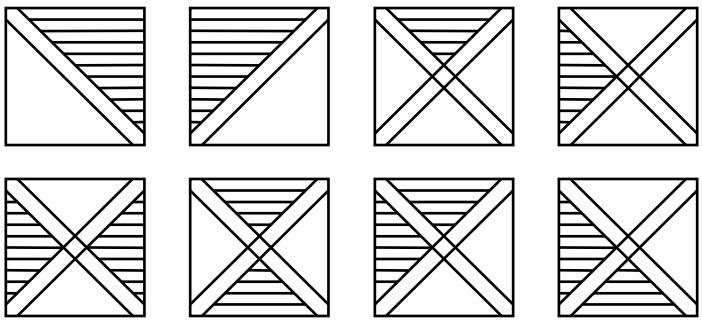
😐 Escrever um algoritmo que lê duas matrizes **N1(4,6)** e **N2(4,6)** e cria:

a) Uma matriz **M1** que seja a soma de **N1** e **N2**
b) Uma matriz **M2** que seja a diferença de **N1** com **N2**

Escrever as matrizes lidas e calculadas.

Exercício 6

😐 Escrever um algoritmo que lê uma matriz M(6,6) e calcula as somas das partes hachuradas. Escrever a matriz M e as somas calculadas.



Exercício 7

☹ Na teoria de Sistemas define-se elemento mínimax de uma matriz, o menor elemento da linha em que se encontra o maior elemento da matriz. Escrever um algoritmo que lê uma matriz A(10,10) e determina o elemento mínimax desta matriz, escrevendo a matriz A e a posição do elemento mínimax.

Exercício 8

☹ Escreva um algoritmo que ordene os elementos de cada linha de uma matriz M[10,10].

☠ Desafio

Escrever um algoritmo que gere e escreva o quadrado mágico de ordem 7.

Um quadrado mágico de ordem n (sendo n um número ímpar) é um arranjo de números de 1 até n^2 em uma matriz quadrada de tal modo que a soma de cada linha, coluna ou diagonal é a mesma.

5			4	7
6	4			3
2	0	3		
	1	9	2	0
		5	8	1

A figura mostra um quadrado mágico de ordem 5. A regra de formação é relativamente fácil de ser verificada: Comece com o 1 no meio da primeira linha. À partir daí siga para cima e para à esquerda diagonalmente (quando sair do quadrado suponha que os lados superior e inferior estão unidos e os lados esquerdo e direito da mesma forma). Em cada quadrado que passar coloque o valor do quadrado anterior acrescido de uma unidade. Quando atingir um quadrado já preenchido, desça um quadrado e o preencha e continue seguindo a diagonal até ter colocado o valor n^2.

Capitulo 17 Subalgoritmo Exercícios

Exercício 1

☺ Escrever um algoritmo para determinar se um determinado número inteiro é par ou ímpar. Utilizar um subalgoritmo de função que retorna um valor lógico para indicar se o valor recebido é par ou não.

Exercício 2

☺ Escreva um algoritmo que lê um número não determinado de valores m, todos inteiros e positivos, um valor de cada vez, e, se m<10 utiliza um subalgoritmo do tipo função que calcula o fatorial de m, e caso contrário, utiliza um subalgoritmo do tipo função para obter o número de divisores de m (quantos divisores m possui). Escrever cada m lido e seu fatorial ou seu número de divisores com uma mensagem adequada". Neste caso, temos um programa principal e dois subalgoritmos.

Exercício 3

😐 Escreva um algoritmo que apresente um menu com três opções:

1 – Inserir
2 – Remover
3 – Escrever na Tela o Vetor
4 – Sair

Quando for escolhida a opção número 1, uma subrotina chamada **insere** deve inserir um elemento (número) em um vetor. A subrotina deve receber por parâmetro o número a ser inserido, a posição (índice) a ser inserido, o tamanho do vetor e o nome do vetor.

Quando for escolhida a opção número 2, uma subrotina chamada **remove** deve eliminar um elemento de um vetor. A subrotina deve receber por parâmetro a posição (índice) do elemento a ser eliminado, o tamanho do vetor e o nome do vetor.
Quando for escolhida a opção número 3, uma subrotina chamada **escreve** deve escrever na tela os elementos do vetor.

Exercício 4
😐 Faça uma subrotina que receba uma matriz **M**(10,10), o número de uma linha **L**, o número de uma coluna ***C*** e retorne a matriz **N**(9,9) resultante da remoção da linha **L** e da coluna ***C***

Exercício 5
☹ Faça uma subrotina que receba dois vetores V1(100) e V2(100) em que cada posição contem um dígito e retorne V3(101) com a soma dos números nos vetores.

Ex: V1 = 0 0 0 3 2 1
V2 = 0 0 4 7 3 2

V3 = 0 0 5 0 5 3

Exercício 6
☹ A tabela abaixo expressa os valores de apartamentos de diferentes metragens em diferentes bairros da cidade de Porto Alegre. As colunas desta Matriz 5x5 mostram a metragem e as linhas o nome do Bairro. Faça um **algoritmo principal** que chama os seguintes subalgoritmos:

a) **Subrotina** de Leitura da matriz 5x5;
b) **Subrotina** de Escrita da matriz 5x5;
c) **Função/Subrotina** que calcula o apartamento mais caro de Porto Alegre;
d) **Função/Subrotina** que calcula o apartamento mais barato do bairro que tem o apartamento mais caro de Porto Alegre;
e) **Subrotina** que confere um aumento de 5% a todos os apartamentos que custam menos de R$ 250.000,00.

	100 m²	***150 m²***	***200 m²***	***250m²***	***300 m²***
Centro	70	80	90	100	200
Bela Vista	120	180	240	300	360
Petrópolis	100	150	250	300	450
Moinhos	180	250	360	410	540
Bom Fim	90	130	170	210	350

Valores expressos em mil Reais

O **algoritmo principal** deve, nesta ordem: ler a matriz; escrevê-la; escrever o valor do apartamento mais caro de Porto Alegre; escrever o valor do apartamento mais barato do bairro que tem o apartamento mais caro de Porto Alegre; conferir o aumento a todos os apartamentos que custam menos de R$ 250.000,00 e escrever novamente a matriz com os valores modificados.

☠ Desafio

Escreva uma função que receba um vetor literal de 1000 posições e retorne o número de palavras do vetor. As palavras são separadas por espaços em branco.

Capitulo 18 Recursão Exercícios

Exercício 1
☺ Escrever um algoritmo, utilizando um subalgoritmo recursivo, para calcular a soma dos 'n' primeiros inteiros positivos, sendo 'n' um valor fornecido pelo usuário.

Exercício 2
😐 Escrever um algoritmo, utilizando um subalgoritmo recursivo, que eleve um número inteiro qualquer a uma potência. Devem ser fornecidos o número e a potência.

Exercício 3
😐 Faça uma função recursiva que receba um vetor de 100 posições e retorne o somatório dos elementos pares (ou ímpares) do vetor.

Exercício 4
☹ Escrever um algoritmo, utilizando um subalgoritmo recursivo, para calcular o N-esimo termo da série de Fibonacci.

☠ Desafio

Escrever um algoritmo, utilizando um subalgoritmo recursivo, que leia um valor inteiro qualquer e realize uma pesquisa em um vetor de 100 posições. No algoritmo principal deve ser informado se o valor lido está ou não contido no vetor. Caso ele esteja, também deve ser informada a sua posição.

Resumo

20 - FORMA GERAL DE UM ALGORITMO

```
Algoritmo "<nome do algoritmo>"

var
< declaração de variáveis>
inicio
< lista de comandos>
fimalgoritmo
```

21 - DECLARAÇÃO DAS VARIÁVEIS

```
 VAR
<identificador 1>, <identificador 2>, ..., <identificador n>: <tipo das
variáveis
```

TIPO	DESCRIÇÃO
INTEIRO	Representa valores inteiros. Exemplos: 10, 5, -5, -10
REAL ou NUMERICO	Representa valores reais (com ponto separador da parte decimal). Exemplos: 10, 15.5, -14.67
LITERAL ou CARACTER	Representa texto (seqüência ou cadeia de caracteres) entre aspas duplas. Exemplo "Esta é uma cadeia de caracteres", "B", "1234"
LOGICO	Representa valores lógicos (VERDADEIRO ou FALSO).

PALAVRAS RESERVADAS			
aleatorio	e	grauprad	passo
abs	eco	inicio	pausa
algoritmo	enquanto	int	pi
arccos	entao	interrompa	pos
arcsen	escolha	leia	procedimento
arctan	escreva	literal	quad
arquivo	exp	log	radpgrau
asc	faca	logico	raizq
ate	falso	logn	rand
caracter	fimalgoritmo	maiusc	randi
caso	fimenquanto	mensagem	repita
compr	fimescolha	minusc	se
copia	fimfuncao	nao	sen
cos	fimpara	numerico	senao
cotan	fimprocedimento	numpcarac	timer
cronometro	fimrepita	ou	tan
debug	fimse	outrocaso	verdadeiro
declare	função	para	xou

22 - OPERADOR DE ATRIBUIÇÃO

```
Variável <- "Texto" ou Valor
```

OPERADORES ARITMÉTICOS	PORTUGUÊS ESTRUTURADO
Adição	**+**
Subtração	**-**
Multiplicação	*****
Divisão	**/**
Divisão Inteira	****
Exponenciação	**^ ou Exp** (<*base*>,<*expoente*>)
Módulo (resto da divisão)	**%**

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

OPERADORES LÓGICOS	PORTUGUÊS ESTRUTURADO	SIGNIFICADO
Multiplicação lógica	**E**	Resulta **VERDADEIRO** se ambas as partes forem verdadeiras.
Adição lógica	**Ou**	Resulta **VERDADEIRO** se uma das partes é verdadeira.
Negação	**Nao**	Nega uma afirmação, invertendo o seu valor lógico: se for **VERDADEIRO** torna-se **FALSO**, se for **FALSO** torna-se **VERDADEIRO**.

23 - Linhas de Comentário

```
// Este método calcula o fatorial de n...x <- y;
```

24 - Comandos de E/S (Entrada/Saída)

```
Escreva (<expressão ou identificador ou constante>, <expressão ou
identificador  ou constante>, ..., <expressão ou identificador ou
constante>)
```

```
Leia (<identificador>)
```

25 - Estrutura Condicional

```
se <condição> entao
   <ações (uma ou mais) a serem realizadas se a condição for verdadeira>
fimse
```

```
se <condição> entao
   <ações (uma ou mais) a serem realizadas se a condição for verdadeira>
senao
   <ações (uma ou mais) a serem realizadas se a condição for falsa>
fimse
```

```
escolha < expressão-de-seleção >
caso < exp 1 > , < exp 2 >, ... , < exp n >
   < lista-de-comandos-1 >
caso < exp 1 > , < exp 2 >, ... , < exp n >
   < lista-de-comandos-2 >
outrocaso
   < lista-de-comandos-3 >
fimescolha
```

26 - Estrutura de Repetição

```
repita
  <lista de comandos>
ate <expressão
```

```
enquanto <expressão lógica ou relacional> faca
   <lista de comandos>
fimenquanto
```

```
para <variável de controle> de <valor inicial> ate <valor final> [passo
<incremento>] faca
   <lista de comandos>
fimpara
```

27 - Variáveis Indexadas Unidimensionais (Vetores)

```
<identificador> : vetor [<tamanho>] de < tipo >
Tamanho [VI..VF]=> Vi= Valor inicial do índice e VF valor Final do índice.
< identificador>[<posição>] <- <valor>
```

28 - Variáveis Indexadas Bidimensionais (Matrizes)

```
<identificador> : vetor [<posição 1>,< posição 2>] de < tipo >
Tamanho [VI..VF]=> Vi= Valor inicial do índice e VF valor Final do índice.
< identificador>[<posição 1>,<posição 2>] <- <valor>
```

29 - Subalgoritmos

Funções Predefinidas do Visualg

FUNÇÂO	DESCRIÇÃO
Abs (valor : real) : real	Valor absoluto
Arccos (valor : real) : real	Arco cosseno
Arcsen (valor : real) : real	Arco seno
Arctan (valor : real) : real	Arco tangente
Asc (s : caracter) : inteiro	Retorna o código ASCII
Compr (c : caracter) : inteiro	Retorna a dimensão do caractere
Copia (c : caracter , posini, posfin : inteiro) : caracter	Copia um determinado trecho do caractere
Cos (valor : real) : real	Cosseno
Cotan (valor : real) : real	Co-tangente
Exp (<base>,<expoente>)	Potenciação
Grauprad (valor : real) : real	Converte grau para radiano

Int (valor : real) : inteiro	Converte o valor em inteiro
Log (valor : real) : real	Logaritmo de base 10
Logn (valor : real) : real	Logaritmo natural (ln)
Maiusc (c : caracter) : caracter	Converte em Maiúscula
Minusc (c : caracter) : caracter	Converte em Minúscula
Numpcarac (n : inteiro ou real) : caracter	Converte um numero inteiro ou real para caractere
Pi : real	Valor Pi
Pos (subc, c : caracter) : inteiro	Retorna a posição do caractere.
Quad (valor : real) : real	Elevado quadrado
Radpgrau (valor : real) : real	Converte Radiano para grau.
Raizq (valor : real) : real	Raiz quadrada
Rand : real	Gerador de números aleatórios entre 0 e 1
Randi (limite : inteiro) : inteiro	Gerador de números inteiros aleatórios com um limite determinado
Sen (valor : real) : real	Seno
Tan (valor : real) : real	Tangente

Criando Funções

```
Algoritmo "<nome do algoritmo>"
var
   <declaração de variáveis globais>
<definição da função>
inicio
   < lista de comandos>
fimalgoritmo
```

Sintaxe da Função

```
funcao <identificador> ([var]<parâmetros>) <tipo de retorno>
var
<declaração de variáveis locais>
inicio
<lista de comandos>
```

```
retorne <variável de retorno>
fimfuncao
```

Identificador: Nome da função.

Passagem de parâmetros por referência: utiliza-se a construção **VAR** antes dos identificadores para indicar a passagem por referência. Os identificadores são separados por vírgula.

Parâmetros: Entre um mesmo tipo de dados são separados por vírgula. Entre tipos de dados a separação é feita com ponto-e-vírgulas '**;**'.

Tipo de retorno da função: Real, Inteiro, Lógico ou Caractere.

Declaração de variáveis locais: idêntica a declaração de variáveis globais. As variáveis declaradas localmente tem validade dentro do escopo da função.

Retorne: local onde é colocado a variável de retorno.

Cuidados

Sempre declare as variáveis globais antes da função.
A função sempre fica dentro do escopo Algoritmo e Fim Algoritmo.
Procure não Declarar variáveis globais com o mesmo nome das variáveis da função.

Procedimento (Sub_rotinas)

Sintaxe Procedimento:

```
procedimento <identificador> ([var]<parâmetros>)
var
<declaração de variáveis locais>
inicio
<lista de comandos>
fimprocedimento
```

Identificador: Nome do procedimento.

Passagem de parâmetros por referência: utiliza-se a construção VAR antes dos identificadores para indicar a passagem por referência. Os identificadores são separados por vírgula.

Parâmetros: Entre um mesmo tipo de dados são separados por vírgula. Entre tipos de dados a separação é feita com ponto-e-vírgulas '**;**'.