Universidade do Estado de Santa Catarina – UDESC

Centro de Ciências Tecnológicas – CCT

Apostila sobre VisuAlg

Christian J. Pereira

Joinville, 2011

Sumário

1 - Introdução Aos Algoritmos

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. De maneira geral, existem muitos caminhos que levam a uma solução. Para praticar os algoritmos e os problemas, utilizaremos o software Visualg desenvolvida por Cláudio Morgado de Souza. E-mail: cmorgado@apoioinformatica.com.br. Para programar em Visualg, utiza-se o Portugol (ou Português Estruturado), ele é, na verdade uma simplificação do português, limitado à um conjunto de palavras e regras que definem o que chamamos de sintaxe da linguagem, esta sintaxe será apresentada aos poucos.

2 - 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 Portugol, a tabela a seguir mostra um exemplo.

{ [2/3 - (5 - 3)]+1} . 5	((2/3 - (5-3)) +1)*5
Tradicional	Computacional

As tabelas a seguir mostram os operadores aritméticos em Portugol

Operadores Aritméticos	Portugol
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Divisão Inteira	\
Exponenciação	^ ou Exp(<base>,<expoente>)
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	Portugol
Maior	>
Menor	<
Maior ou Igual	>=
Menor ou Igual	<=
Igual	=
Diferente	<>

Por exemplo, 2+5>4 resulta em VERDADEIRO, 3 <> 3 resulta em FALSO

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

Operadores Lógicos	Portugol	Significado
Multiplicação Lógica	E	Resulta VERDADEIRO quando ambas as partes forem verdaderas.
Adição Lógica	Ou	Resulta VERDADEIRO quando uma das partes for verdadeira
Negação	Nao	Nega uma afirmação, invertendo seu valor: caso seja FALSO, torna-se VERDADEIRO.

Por exemplo, (2+5>4) E (3<>3) resulta FALSO,pois a afirmação 3<>3 é FALSO.

(2+5>4) E (Nao(3<>3)) Resulta VERDADEIRO.

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 Portugol podemos ter parênteses dentro de parênteses, como seriam os colchetes e as chaves na matemática.

A prioridade dos operadores está descrita nas tabelas a seguir:

Operador Aritmético	Prioridade
Exponenciação	3 (Maior)
Multiplicação/Divisão	2
Adição/Subtração	1 (Menor)

(2+2)/2 resulta em 2, e 2 + 2 / 2 resulta em 3.

Operador Lógico	Prioridade
E	3
Ou	2
Nao	1

(2>3) ou (3<2) e (2<3) resulta em FALSO, (2>3) e (3<2) ou (2<3) resulta em VERDADEIRO.

Também existe prioridade entre as categorias dos operadores.

Operador	Prioridade
Aritméticos	3
Relacionais	2
Lógicos	1

Vale lembrar que o 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), assim seria o correto.

3 - Forma Geral de um ALGORITMO

Nessa seção vamos conhecer os primeiros elementos que compõem o Portugol 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 destaque.

4 - 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.

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.

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 underline '_' (os outros caracteres podem ser letras, números e underline);
c. Nomes de variáveis devem ter no máximo 127 caracteres;
d. Nomes de variáveis não podem conter espaços em branco;
e. Na sintaxe do Portugol , não há diferença entre letras maiúsculas de minúsculas (NOME é o mesmo que noMe);
f. Duas ou mais variáveis não podem possuir o mesmo nome.

Exemplo

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 Portugol, só existem quatro tipos de dados, conforme a tabela abaixo.

Tipo	Descrição
Inteiro	Representa valores inteiros. Exemplos: 10,5,-5,-10
Real ou Numérico	Representa valores reais (com ponto separador da parte decimal) Exemplos: 10,15.5, -14,67
Literal ou Caractere	Representa texto (sequencia ou cadeia de caracteres) entre aspas duplas. Exemplos: “Esta é uma cadeia de caracteres”, “B”, “12”
Lógico	Representa valores lógicos (VERDADEIRO ou FALSO)

5 - 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
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:

```
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:

2060 <- NumeroConta
NumeroAgencia+digitoControle <- 2345 + 0
NomeCliente+sobrenome <- “João” + “Silva”

6 - 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
// Este método calcula o fatorial de n...x <- y;

7 - 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 Portugol utilizamos 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
```
X <- 3.5
Y <- 4
Escreva ("O valor de X é", X)
Escreva (" E o valor de Y é ", Y)
Escreva (" 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
```
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

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:

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

8 - 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

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.

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**

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.

9 - 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

"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 o Portugol:

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:

```
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:

```
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:

```
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:

```
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:

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 permitem associar um conjunto de instruções a ser executado se a condição do comando resultar em FALSO. em Portugol, 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:

```
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
```

10 - 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

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)
23. fimalgoritmo
```

11 - 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 três vezes.

Exemplo

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 Portugol possui três estruturas de repetição: repita...ate, enquanto...faca e para...faca.

12 - 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:

repita
<lista de comandos>

ate <expressão lógica ou relacional>

Exemplo
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
```

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.

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.

13 - 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

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.

14 - 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.

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.

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.

15 - 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.

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.

15.1 - 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

```
IDADE:   VETOR [1..5] DE INTEIRO
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

```
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

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
```

15.2 - 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 >

Exemplo
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
PESSOAS[1,3]<- “Joao”

Exemplo

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 i 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
```

16 - 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.

16.1 - 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 cód ASCII
Compr(c: caracter): inteiro	Retorna a dimensão do caracter

Copia(c:caracter, posini,posfim :inteiro):caracter	Copia um determinado trecho de caracter
Cos (valor:real):real	Cosseno
Cotan(valor:real):real	Cotangente
Exp(<base>,<expoente>)	Potenciação
Grauprad(valor: real): real	Converte grau para radianos
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 (subs, c: caracter) : inteiro	Retorna a posição do caractere
Quad(valor: real): real	Elevado ao quadrado
Radpgrau(valor: real) : real	Converte radiano para grau
Raizq(valor:real):real	Raiz quadrada
Rand: real	Gerador de numeros aleatórios entre 0 e 1
Randi (limite: inteiro) :inteiro	Gerador de numeros aleatórios com um limite determinado
Sen (valor: real):real	Seno
Tan (valor: real):real	Tangente

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

```
1.  Algoritmo "RETORNA O SOBRENOME"
2.  var
3.  nome, sobrenome : Caractere
4.  quant_caracteres, local_espcao : INTEIRO
5.  inicio
6.  nome <- "Joao Silva"
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
```

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.

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

```
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. ("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
```

16.2 - 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

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

Exercícios

Capítulo 8

Exercício 1
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 em km/h, e exibir a seguinte frase:
"A velocidade média do <piloto> foi <velocidade média> km/h."

Exercício 2
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 pedido (queijo, presunto, banana, etc). Uma turma vai à pizaria e pde uma determinada quantidade de chopps e uma pizza grande com uma determinada quantidade de coberturas. Faça um algoritmo que calcule e conte e, sabendo quantas pessoas estão à mesa, quanto que cada um deve pagar (não esqueca dos 10% do garçom).

Exercício 3
Escreva um algoritmo que calcule o número de notas que deve ser dado de troco para um pagamento efetuado. O algoritmo deve ler o valor a ser pago, e o valor realmente pago. Supor que o troco seja dado em notas de 50, 20, 10, 5, 2 e 1 real, dando sempre a menor quantidade de notas.

Capítulo 9 e 10

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: "Número Negativo!".

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 à 5 vezes o salário da pessoa, o algoritmo deverá escrever "Financiamento Concedido", senão, ele deverá escrever "Financiament Negado". Independente de conceder ou não o financiamento, o algoritmo escreverá depois a frase: "Obrigado por nos consultar."

Exercício 3
Faça um programa que lê 4 valores, I, A, B, C, onde I é um número inteiro positivo e A, B e C são quaisquer valores reais. O programa deve escrever os valores lidos e:

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

Capítulo 11 à 14

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 pessoa do grupo deve ler o nome da pessoa, a altura, o peso e o sexo. Calcular 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 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 à 12, calcular e escrever a soma inteiros de 1 até o número lido.

Exerício 4
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: 28 = 1+2+4+7+14.

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 ele 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

Exercício 6
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 MDC e o MMC de a, b, escrevendo-os juntamente com os valores lidos.

Capítulo 15

Exercício 1
Escrever um algoritmo que lê um vetor de 13 posições que é o gabarito de um teste de loteria esportiva, contendo os valores 1, 2 ou 3. Ler, a seguir, para cada apostador, o número de seu cartão e um vetor de respostas. Verificar para cada apostador o número de acertos e escrevê-lo na tela, caso tenha 13 acertos, aparecer também a mensagem: “GANHADOR, PARABÉNS!”.

Exercício 2
Escrever um algoritmo que lê um vetor de 15 posições e o escreve. Ordene a seguir os elementos do vetor em ordem crescente e escreva novamente o vetor na tela.

Exercício 3
Escrever um algotirmo que lê, para um vetor de 30 posições, vinte valores que ocuparão

as primeiras 20 posições do vetor. Ordene, a seguir os elementos em ordem crescente. leia, a seguir 10 valores, um por vez, e insira-os nas posições adequadas do vetor ordenado, de forma que o mesmo continue ordenado. Escreva o vetor ao final.

Exercício 4
Na teoria de sistemas define-se elemento minimax de uma matriz, o menor elemento da linha em que se encontra o maior elemento da matriz. Escrever um algoritmo que lê uma matriz 10x10 e determina o elemento minimax desta matriz, escrevento a matriz e a posição do elemento minimax ao final.

Exercício 5
Escreva um algoritmo que ordene os elementos de cada linha de uma matriz 10x10.

Capítulo 16

Exercício 1
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.

Exercício 2
faça uma subrotina que receba uma matriz 10x10, o número de uma linha, o número de uma coluna e retorne uma matriz 9x9, resultante da remoção da coluna e da linha informados.