UCS
UNIVERSIDADE
DE CAXIAS DO SUL

Introdução aos algoritmos

Cristian Koliver
Bruno Tonet

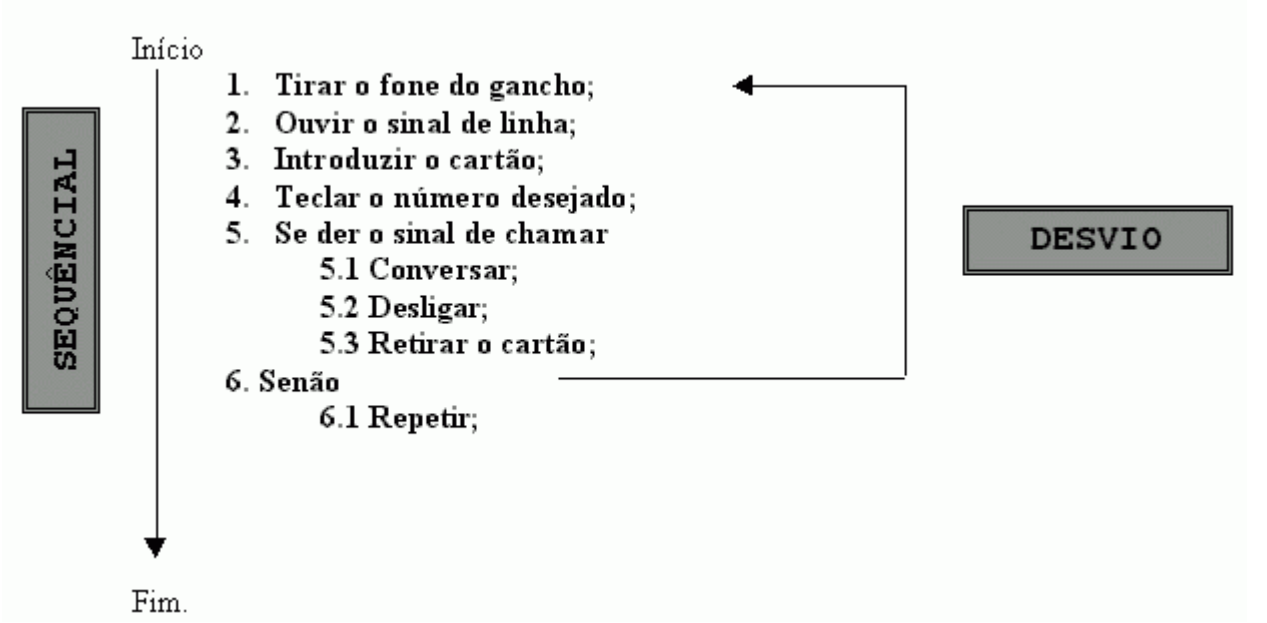
SUMÁRIO

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.

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 e contribuir, cada um, para a realização da tarefa em questão.

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	**^**
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**.
Ou exclusivo	**xou**	Resulta em **VERDADEIRO** se as partes forem diferentes, e em **FALSO** se forem iguais.

A tabela abaixo – chamada ***tabela-verdade*** – mostra os resultados da aplicação dos operadores lógicos conforme os valores dos operandos envolvidos.

A	B	A E B	A OU B	NÃO A	NÃO B	A XOU B
VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	FALSO	FALSO	FALSO
VERDADEIRO	FALSO	FALSO	VERDADEIRO	FALSO	VERDADEIRO	VERDADEIRO
FALSO	VERDADEIRO	FALSO	VERDADEIRO	VERDADEIRO	FALSO	VERDADEIRO
FALSO	FALSO	FALSO	FALSO	VERDADEIRO	VERDADEIRO	FALSO

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

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

OPERADOR	PRIORIDADE
Operadores aritméticos	3
Operadores relacionais	2
Operadores lógicos	1

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 "<tipo do algoritmo>"
< declaração de variáveis>
inicio
< lista de comandos>
fimalgoritmo
```

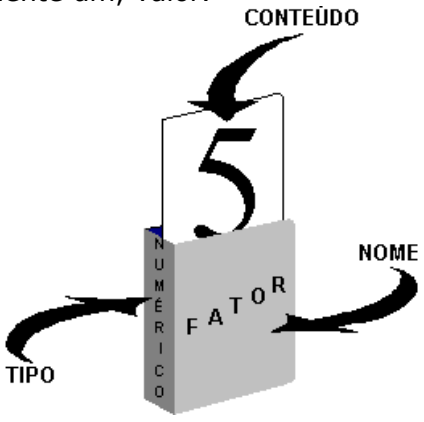
onde as palavras **algoritmo** e **fimalgoritmo** fazem parte da sintaxe da linguagem e sempre delimitam o **inicio** e fim de um algoritmo; a < declaração de variáveis> é a seção ou parte do algoritmo onde descrevemos os tipos de dados que serão usados na lista de comandos. Por exemplo, poderíamos definir 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 utilizamos o comando de declaração de variáveis que faz parte da nossa linguagem que tem a seguinte forma:

```
declare <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 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	**enquanto**	**fimrepita**	**outrocaso**
algoritmo	**entao**	**fimse**	**para**
arquivo	**escolha**	**inicio**	**passo**
ate	**escreva**	**interrompa**	**pausa**
caracter	**faca**	**leia**	**repita**
caso	**falso**	**literal**	**se**
cronometro	**fimalgoritmo**	**logico**	**senao**
debug	**fimenquanto**	**mensagem**	**timer**
declare	**fimescolha**	**nao**	**verdadeiro**
e	**fimpara**	**numerico**	**xou**
eco		**ou**	

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

TIPO	DESCRIÇÃO
numerico	Representa valores inteiros e reais (com ponto separador da parte decimal). Exemplos: 10, 15.5, -14.67
logico	Representa valores lógicos (VERDADEIRO ou FALSO).
literal	Representa texto (seqüência ou cadeia de caracteres) entre aspas duplas. Exemplo "Esta é uma cadeia de caracteres", "B", "1234"

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

declare salario: **numerico**
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
```

Múltiplas Linhas

Para tal, basta colocar o comentário entre chaves **{ }**

Exemplo 7.2

ALGORITMO "Boletim"

{este algoritmo pega o nome de um aluno, seu código e as suas notas, e verifica se ele passou de ano}

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>
```

Exemplo 8.1

```
X <- 3.5
Y <- 4
escreva "O valor de X é", X, " e o valor de Y é ", Y
escreva "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
```

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

```
leia "Digite seu nome", nome
leia "Digite sua agencia", NumeroAgencia
leia "Digite sua conta", 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
```

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.  Declare A,B,Media : Numerico
3.  Inicio
4.  Escreva "Programa que calcula a média aritmética de dois valores."
5.  Leia "Digite um valor : ",  A
6.  Leia "Digite outro valor : ", B
7.  Media <- (A+B)/2
8.  Escreva "A média dos dois valores é : ", Media
9.  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.  declare  a, b, c, MH: numerico
3.  declare  NOME: literal
4.  inicio
5.  leia "Entre com o nome do aluno: ", nome
6.  escreva "Entre com as notas das três provas"
7.  leia "Digite a primeira nota: ", a
8.  leia "Digite a segunda nota: ", b
9.  leia "Digite a terceira nota: ", c
10. MH <- 3/(1/a + 1/b +1/c)
11. escreva "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 "calculaAreaCirculo"
2.  declare Area, Raio, Pi: numerico
3.  inicio
4.  leia "Entre com o raio: ", Raio
5.  Pi <- 3.1415
6.  Area <- Pi*Raio^2
7.  escreva "A area do circulo com o raio ", Raio, " é ", Area
8.  fimalgoritmo
```

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, então 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

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

```
1.   algoritmo "calculaArea"
2.   declare Area, Raio: numerico
3.   inicio
4.   escreva "Entre com raio do círculo "
5.   leia Raio
6.   se Raio > 0 entao
7.      Area <- 3.1415*(Raio^2)
8.      escreva "A área do círculo de raio ", Raio, " é ", Area
9.   fimse
10.  se Raio <= 0 entao
11.     escreva "Raio negativo!"
12.  fimse
13.  fimalgoritmo
```

Observe que se a condição do primeiro `se` é **verdadeira**, a condição do segundo `se` é **falsa** e vice-versa, e o conjunto de instruções a ser executado **se** `Raio <= 0` (apenas a instrução **escreva** `"Raio 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 "calculaArea"
2.  declare Area, Raio: numerico
3.  inicio
4.  escreva "Entre com raio do círculo "
5.  leia Raio
6.  se Raio > 0 entao
7.     Area <- 3.1415*(Raio^2)
8.     escreva "A área do círculo de raio ", Raio, " é ", Area
9.  senao
10.    escreva "Raio negativo!"
11. fimse
12. 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.  declare Dinheiro: numerico
3.  inicio
4.  escreva "*** Serviço Informatizado de Sugestões ***"
5.  escreva "Quanto dinheiro você tem sobrando?"
6.  leia Dinheiro
7.  se Dinheiro >= 10 entao
8.     escreva "Vá ao cinema hoje à noite."
9.  senao
10.    escreva "Fique em casa vendo TV."
11. fimse
12. escreva "Obrigado e volte sempre."
13. 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				
declare a,b,c: **numerico**	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				

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.  declare  a, b, c, MH: numerico
3.  declare  NOME: literal
4.  inicio
5.  leia "Entre com o nome do aluno: ", nome
6.  escreva "Entre com as notas das três provas"
7.  leia "Digite a primeira nota: ", a
8.  leia "Digite a segunda nota: ", b
9.  leia "Digite a terceira nota: ", c
10. MH <- 3/(1/a + 1/b +1/c)
11. escreva "A média harmônica do aluno: ", NOME, " é ", MH
12. leia "Entre com o nome do aluno: ", nome
13. escreva "Entre com as notas das três provas"
14. leia "Digite a primeira nota: ", a
15. leia "Digite a segunda nota: ", b
16. leia "Digite a terceira nota: ", c
17. MH <- 3/(1/a + 1/b +1/c)
18. escreva "A média harmônica do aluno: ", NOME, " é ", MH
19. leia "Entre com o nome do aluno: ", nome
20. escreva "Entre com as notas das três provas"
21. leia "Digite a primeira nota: ", a
22. leia "Digite a segunda nota: ", b
23. leia "Digite a terceira nota: ", c
24. MH <- 3/(1/a + 1/b +1/c)
25. escreva "A média harmônica do aluno: ", NOME, " é ", MH
26. 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:

```
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.    declare i: numerico
3.    inicio
4.    i<- 1
5.    repita
6.       escreva i
7.       i<- i + 1
8.    ate i > 10
9.    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.    declare  a, b, c, MH, i: numerico
3.    declare  NOME: literal
4.    inicio
5.    i <- 1
6.    Repita
7.       leia "Entre com o nome do aluno: ", nome
8.       escreva "Entre com as notas das três provas"
9.       leia "Digite a primeira nota: ", a
10.      leia "Digite a segunda nota: ", b
11.      leia "Digite a terceira nota: ", c
12.      MH <- 3/(1/a + 1/b +1/c)
13.      escreva "A média harmônica do aluno: ", NOME, " é ", MH
14.      i <- i + 1
15.   ate i > 3
16.   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 1.

```
1.    algoritmo "DemonstraRepeticao"
2.    declare i: numerico
3.    inicio
4.    i <- 10
5.    Repita
6.      escreva i
7.      i <- i - 2
8.    ate i <= 0
9.    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.    declare Contador: numerico
3.    inicio
4.    repita
5.      Contador <- 1
6.      Contador <- Contador + 1
7.    ate Contador = 10
8.    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.    declare Soma: numerico
3.    inicio
4.    Soma <- 1
5.    repita
6.      Soma <- Soma + 2
7.    ate Soma = 10
8.    escreva soma
9.    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.  declare Soma: numerico
3.  inicio
4.  Soma <- 1
5.  repita
6.    Soma <- Soma + 2
7.  ate Soma > 10
8.  escreva soma
9.  fimalgoritmo
```

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.

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

```
1.  algoritmo "comRepita"
2.  declare i: numerico
3.  inicio
4.  repita
5.    leia i
6.    se i >=0 entao
7.      escreva i^0.5
8.    fimse
9.  ate i<0
10. 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.    declare i, numero: numerico
3.    inicio
4.    para i de 1 ate 5 passo 1 faca
5.       leia numero
6.       se numero>0 entao
7.          escreva numero
8.       fimse
9.    fimpara
10.   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.    declare i, N: numerico
3.    inicio
4.    leia N
5.    para i de 1 ate N faca
6.       escreva i
7.    fimpara
8.    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.

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 "Inviavel"
2.    declare nome1, nome2, nome3, nome4, nome5, nome6, nome7 ,....,
nome499, nome500: literal
3.    inicio
4.    leia nome1,nome2,...,nome500
5.    ...
6.    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:

```
declare <identificador>[<tamanho>]: < tipo >
```

Exemplo 16.2

```
7.    declare nomes[5]: literal
8.    declare idades[5]: numerico
```

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 `vet` de 6 posições e o escreve. A seguir, ele conta quantos valores de `vet` são negativos e escreva esta informação.

```
1.  algoritmo "vetores"
2.  declare vet[6]: numerico
3.  declare i, conta_neg: numerico
4.  inicio
5.  conta_neg <- 0
6.  para i de 1 ate 6 faca
7.     leia vet[i]
8.     se vet[i] < 0 entao
9.        conta_neg <- conta_neg + 1
10.    fimse
11.  fimpara
12.  para i de 1 ate 6 faca
13.    escreva vet[i]
14.  fimpara
15.  escreva "Total de números negativos: ", conta_neg
16.  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:

```
declare <identificador>[<tamanho 1>,<tamanho 2>]: < tipo >
```

Exemplo 16.5

```
1.  declare M[2,3]: numerico
```

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

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

```
1.    algoritmo "Matriz"
2.    declare Matriz[3,3]: numerico
3.    declare i, j, somaLinha3, somaColuna2, somaDiagPrinc, somaDiagsecu,
somaTudo: numerico
4.    inicio
5.    somaLinha3 <- 0
6.    somaColuna2 <- 0
7.    somaDiagPrinc <- 0
8.    somaDiagsecu <- 0
9.    somaTudo <- 0
10.   para i de 1 ate 3 faca
11.      para j de 1 ate 3 faca
12.         leia matriz[i,j]
13.               somaTudo <- matriz[i,j] + somaTudo
14.         se i=3 entao
15.            somaLinha3 <- matriz[i,j]+ somaLinha3
16.         fimse
17.         se j=2 entao
18.            somaColuna2 <- matriz[i,j]+ somaColuna2
19.         fimse
20.         se i=j entao
21.            somaDiagPrinc <- matriz[i,j]+ somaDiagPrinc
22.         fimse
23.         se j=4-i entao
24.            somaDiagsecu <- matriz[i,j]+ somaDiagsecu
25.         fimse
26.      fimpara
27.   fimpara
28.   para i de 1 ate 3 faca
29.      para j de 1 ate 3 faca
30.         escreva matriz[i,j]
31.      fimpara
32.   fimpara
33.   escreva "Soma de todos os elementos é ", somaTudo
34.   escreva "Soma dos elementos da linha 3 é ", somaLinha3
35.   escreva "Soma dos elementos da coluna 2 é ", somaColuna2
36.   escreva "Soma dos elementos da diagonal principal é ",
somaDiagPrinc
37.   escreva "Soma dos elementos da diagonal secundária é ",
somaDiagsecu
38.   fimalgoritmo
```

17 - Referências

http://www.apoioinformatica.inf.br/
http://www.consiste.dimap.ufrn.br/~david/
http://www.inf.pucrs.br/%7Egidio/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