



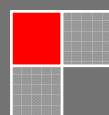
IFBA

CAMPUS
EUNÁPOLIS

CURSO TÉCNICO DE INFORMÁTICA

APOSTILA DE ALGORITMOS

Construir algoritmos é o objetivo fundamental de toda a programação, esta apostila ensina através de um processo lógico como resolver problemas computacionais através da construção de Algoritmos usando o Português Estruturado.



SUMÁRIO

PARTE 1: CONCEITOS INICIAIS	3
PARTE 2: ESTRUTURAS DE DECISÃO	15
PARTE 3: ESTRUTURAS DE REPETIÇÃO	34
PARTE 4: ESTRUTURA DE DADOS HOMOGÊNEAS: MATRIZES.....	49
PARTE 5: SUB-ROTINAS (PROCEDIMENTOS E FUNÇÕES)	63

LÓGICA DE
PROGRAMAÇÃO

PARTE 1:

Conceitos Iniciais

NESTE CAPÍTULO, SERÃO APRESENTADOS OS CONCEITOS INICIAIS
RELACIONADOS À LÓGICA DE PROGRAMAÇÃO TAIS COMO:

- ✚ SEQUÊNCIA LÓGICA
- ✚ ALGORITMO
- ✚ INSTRUÇÕES BÁSICAS
- ✚ VARIÁVEIS E TIPOS DE DADOS
- ✚ OPERADORES ARITMÉTICOS

1.1. CONCEITOS INICIAIS

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a seqüência lógica para o desenvolvimento. Estes pensamentos podem ser descritos como uma seqüência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Na linguagem comum, entende-se por instruções “um conjunto de regras ou normas definidas para a realização ou emprego de algo”. Em informática, porém, instrução é a informação que indica a um computador uma ação a executar. Uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem seqüencial lógica.

Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta. A partir daí, surge o conceito de Algoritmo. Um algoritmo é formalmente uma seqüência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma seqüência de instruções que dão cabo de uma meta específica.

1.2. ALGORITMOS

Os algoritmos são descritos em uma linguagem chamada **pseudocódigo**. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo, Visual Basic, estaremos gerando código em Visual Basic.

Sendo assim os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

Um algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação. Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais:

- ❖ Entrada: São os dados de entrada do algoritmo
- ❖ Processamento: São os procedimentos utilizados para chegar ao resultado final

❖ Saída: São os dados já processados

Imagine o seguinte problema: Calcular a média final dos alunos de uma escola. Os alunos realizarão quatro provas: P1, P2, P3 e P4, onde a média final é dada pela fórmula abaixo:

$$\frac{P1 + P2 + P3 + P4}{4}$$

Para montar o algoritmo proposto acima, devemos fazer três perguntas-chaves:

a) Quais são os dados de entrada?

R: Os dados de entrada são P1, P2, P3 e P4

b) Qual será o processamento a ser utilizado?

R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro)

c) Quais serão os dados de saída?

R: O dado de saída será a média final

“IDENTIFICAR OS ELEMENTOS QUE FORMAM UM ALGORITMO É O PONTO CHAVE PARA A RESOLUÇÃO DE UM PROBLEMA!”

1.3. TIPOS DE DADOS

Um computador nada mais é do que uma ferramenta utilizada para solucionar problemas que envolvam a manipulação de informações, as quais se classificam, grosso modo, em dois tipos básicos: **dados** e **instruções**. Os dados são representados por elementos advindos do mundo externo, os quais representam as informações que os seres humanos manipulam.

Os dados devem ser abstraídos para serem processados em um computador. Podemos dizer que os dados são as “ENTRADAS” de um algoritmo. Os dados podem ser categorizados em três tipos:

❖ Numéricos: valores inteiros e não reais;

- ❖ Caracteres: valores alfabéticos ou alfanuméricos os quais não serão utilizados em operações de cálculo matemático
- ❖ Lógicos: representados por valores dos tipos falsos ou verdadeiros

Os inteiros são dados numéricos positivos e negativos pertencentes ao conjunto de números inteiros, excluindo qualquer valor numérico fracionário. Como exemplos desse tipo de dado têm-se os valores: 35, 0, 234, -90, -10, entre outros. A representação do dado inteiro é feita em português estruturado pelo comando **inteiro**.

Os dados reais são os numéricos positivos e negativos pertencentes ao conjunto de números reais, incluindo todos os valores fracionários e também os valores inteiros. Como exemplo desse tipo de dado tem-se os valores: 35, 0, -90.90, 3.2987 etc. A representação do dado real é feita em português estruturado pelo comando **real**.

Os tipos caracteres são seqüências de valores delimitadas por aspas (""), formadas por letras (de A até Z), números (de 0 até 9) e símbolos (por exemplo, todos os símbolos imprimíveis existentes num teclado). O tipo de dado caractere é conhecido também como alfanumérico, string, literal ou cadeia de caracteres. Como exemplo tem-se os valores: "PROGRAMAÇÃO", "Rua Paulino Mendes", "010.589.987-90", "8843-7895". A representação do dado caractere é feita em português estruturado pelo comando **caractere**.

Por último, ainda existem o tipo de dado lógico que são os dados com valores que sugerem uma única opção entre duas possibilidades existentes, normalmente representados pelos valores **falso** ou **verdadeiro**. O tipo de dado lógico é também conhecido pela nomenclatura booleano. A representação do dado lógico é feita em português estruturado pelo comando **lógico**.

1.4. VARIÁVEIS

Todo dado a ser armazenado na memória de um computador deve ser previamente identificado, ou seja, primeiro é necessário saber o seu tipo para depois fazer o seu armazenamento adequado. Armazenado o dado, ele pode ser utilizado e manipulado a qualquer momento.

Imagine a memória de um computador como um grande arquivo com várias gavetas, e em cada gaveta é possível guardar um único valor por vez. Como em um arquivo, as gavetas devem ser identificadas como uma “etiqueta” contendo um nome.

Do ponto de vista computacional pode-se definir de forma bem simplista que uma variável é a representação de uma região de memória utilizada para armazenar um determinado valor por um espaço de tempo. O tempo de armazenamento de um valor está relacionado ao tempo de duração da execução de um programa.

O nome de uma variável é utilizado para sua identificação e representação dentro de um programa de computador e faz-se necessário estabelecer algumas regras de definição e uso de variáveis:

- ❖ Os nomes de identificação de uma variável podem utilizar um ou mais caracteres, limitando-se a restrições da própria linguagem formal de programação em uso. No caso do português estruturado essa restrição não existe.
- ❖ O primeiro caractere de identificação do nome de uma variável não pode ser numérico. O primeiro caractere de identificação do nome de uma variável deve ser sempre alfabético, os demais podem ser alfanuméricos.
- ❖ Na definição de um nome composto de uma variável não podem existir espaços em branco entre os nomes. Caso deseje separar nomes compostos, deve ser utilizado o caractere de separação “_” underline.
- ❖ Jamais uma variável pode ser definida com o mesmo nome de uma palavra que represente os comandos de uma linguagem de programação de computadores, ou seja, com palavras reservadas de uma linguagem.
- ❖ Não pode ser utilizado como nome de variável algum que já tenha sido usado para identificar o nome de um programa

São nomes válidos de variáveis: NOMEUSUARIO, NOME_USUÁRIO, FONE1, DELTA, _NOME1, etc. No entanto definições como: NOME USUARIO, 1TELEFONE, FONE\$, entre outras serão consideradas inválidas.

As palavras: INTEIRO, REAL, CARACTERE, LÓGICO, entre outras palavras reservadas são inválidas para a definição de nomes de variáveis por já estarem definidas como comandos de identificação de código em “português estruturado”.

Ainda existe um tipo de variável que denominada de Constante. Como sabemos constante é tudo que é fixo, estável, imutável, inalterado, contínuo, invariável, etc. Do

ponto de vista computacional, que é semelhante ao matemático ou científico, uma constante é uma grandeza numérica usada normalmente numa expressão aritmética ou matemática, que define um valor de equilíbrio que se mantém inalterado, independentemente das variáveis envolvidas na operação a ser realizada.

Como exemplo prático de uma constante, tem-se a constante matemática *pi*, que equivale ao valor aproximado de 3.14159265.

Todo algoritmo se inicia com o uso da palavra reservada **algoritmo** que indica o nome do algoritmo a ser desenvolvido. Exemplo:

```
algoritmo "Somar dois numeros"
algoritmo "Calcular IMC"
algoritmo "Pesquisa de Candidatos"
```

Após a identificação do algoritmo, tem-se o bloco de variáveis que é representado pelo comando **var**. Todas as variáveis utilizados em um problema deverá ser declaradas nessa estrutura. Exemplo:

```
var
    nome: caractere
    idade, peso: caractere
```

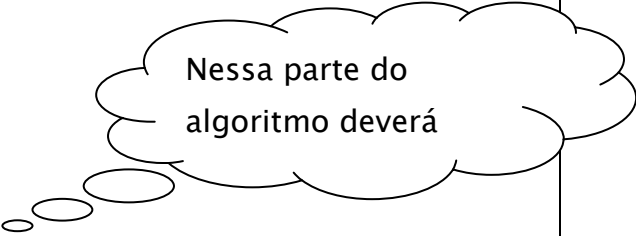
Depois do bloco var, existe o bloco onde todas as instruções do problema são resolvidas. Esse bloco é delimitado por Inicio e Fim que são representados pelos comandos **inicio** e **fimalgoritmo**

Como vimos para criar uma variável em um algoritmo, devemos declará-las no bloco **var** e identificá-las por meio de um nome e um tipo de dado. Veja exemplo completo de um algoritmo com declaração de variáveis:

```
algoritmo "Exemplo de Variáveis"

var
    nome: caractere
    idade, peso: inteiro
    altura: real
    ehdiabetico: logico

inicio
// Seção de Comandos
fimalgoritmo
```



Nessa parte do algoritmo deverá

1.5. OPERADORES ARITMÉTICOS

Os operadores aritméticos são as ferramentas responsáveis pelo estabelecimento das operações matemáticas a serem realizadas em um computador. Tanto variáveis como constantes são utilizadas na elaboração dos cálculos matemáticos.

São sete os principais operadores aritméticos presentes no português estruturado:

OPERADOR	OPERAÇÃO
+	ADIÇÃO
-	SUBTRAÇÃO
*	MULTIPLICAÇÃO
/	DIVISÃO
\	DIVISÃO INTEIRA
^	EXPONENCIAÇÃO
MOD	RESTO DA DIVISÃO

É bastante comum trabalharmos com expressões aritméticas ou fórmulas matemáticas, uma vez que a maior parte do trabalho computacional está relacionada e envolve a utilização de cálculos. Essas expressões são definidas pelo relacionamento entre as variáveis e constantes numéricas com a utilização dos operadores aritméticos.

Considere a fórmula: $AREA = \pi * RAIO^2$ para o cálculo da área de uma circunferência, em que estão presentes as variáveis AREA e RAIO, a constante π ($\pi = 3.14159$), os operadores aritméticos de multiplicação e exponenciação.

As expressões aritméticas em computação são escritas de uma forma um pouco diferente da forma conhecida em matemática. Por exemplo, à expressão:

$$X = \{ 43 . [55 : (30 + 2)] \}$$

A expressão acima seria escrita na forma computacional como:

$$X \leftarrow (43 * (55 / (30 + 2)))$$

Observe que as chaves e os colchetes são abolidos, utilizando-se em seu lugar apenas os parênteses. É também substituído o sinal de (=) pelo sinal de (\leftarrow). O sinal

(←) é utilizado para indicar que o valor de uma expressão aritmética ou fórmula matemática está sendo armazenado em uma variável.

Vamos a outro exemplo. Observe a fórmula abaixo que é utilizada para calcular a área de um triângulo:

$$A = \frac{b \cdot H}{2}$$

A fórmula para cálculo da área de um triângulo poderia ser computacionalmente definida como: **AREA ← (BASE * ALTURA) / 2**

AREA, BASE e ALTURA são variáveis declaradas dentro de um programa.

1.6. INSTRUÇÕES BÁSICAS

As instruções a serem implementadas em um computador para a execução de um determinado programa são representadas por um conjunto de palavras-chave ou comandos que formam a estrutura da linguagem de programação.

Para o estudo de lógica de programação adotaremos uma estrutura de linguagem de programação denominada **portugol** ou **português estruturado**.

Para criar um programa que seja executável dentro de um computador, é preciso ter em mente três pontos de trabalho: a **entrada de dados**, o seu **processamento** e a **saída** deles.

O processo de execução de um programa ocorre segundo o exposto, após a entrada de dados com a instrução **leia** e a sua saída com a instrução **escreva**.

Uma entrada e uma saída podem acontecer dentro de um computador de diversas formas. Por exemplo, uma entrada pode ser feita via teclado, leitores óticos, etc. Uma saída pode ser feita em vídeo, impressora, disco, etc.

Para ler um valor do usuário, deverá ser feito conforme o exemplo abaixo:

<pre>leia (nome) leia (idade)</pre>

Ao executar a instrução acima, o **prompt de comando** exibirá uma linha para que o usuário escreva um valor na tela. Depois que o usuário informar esse valor e

digitar a tecla “ENTER”, o valor informado será gravado na variável e poderá ser utilizado em qualquer momento do programa

Para exibir alguma informação na tela, usamos o comando escreva. Veja o exemplo abaixo:

```
escreva("Digite um Valor")
escreval ("O RESULTADO DA OPERAÇÃO É:")
```

Ao utilizar o comando escreva, o conteúdo que estiver entre aspas duplas será exibido no prompt de comando. Quando utilizamos o comando **escreval**, significa que o texto será exibido na tela e será pulado uma linha para a próxima execução de um comando.

OBSERVAÇÃO IMPORTANTE: TODOS OS COMANDOS DE LEITURA E ESCRITA DE DADOS DEVERÃO ESTÁ LOCALIZADOS NO BLOCO QUE LIMITA O INICIO E FIM DE UM ALGORITMO.

Considere o seguinte exemplo: Deve ser criado um programa que faça a leitura de dois valores numéricos. Realize a operação de soma entre os dois valores e apresente o resultado obtido.

A ordem de execução do algoritmo acima seria:

1. Ler dois valores, no caso as variáveis A e B
2. Efetuar a soma das variáveis A e B, cujo resultado será representado pela variável X
3. Apresentar o valor da variável X após a operação de soma dos dois valores fornecidos

Para começar a escrever o código em Português Estruturado para o problema, devemos relacionar quais são as variáveis e os tipos de dados dessas variáveis. Veja como fica em Portugol:

```
algoritmo "SOMAR_NUMEROS"
var
x: inteiro
a: inteiro
b: inteiro
```

Após relacionar todas as variáveis que serão utilizadas no programa com a instrução **var**, passa-se para a fase de montagem do problema, que se localiza entre as instruções início e fim.

```
início
    leia (a)
    leia (b)
    x <- a + b
    escreva (x)
fimalgoritmo
```

1.7. EXEMPLOS

- a) Desenvolver a lógica para um programa que calcule a área de uma circunferência, e apresente a medida da área calculada.

```
algoritmo "AREA_CIRCULO"
var
    a: real
    r: real
início
    leia (r)
    a <- 3.14159 * r ^ 2
    escreva (a)
fimalgoritmo
```

- b) Construir um programa que calcule o salário líquido de um professor. Para elaborar o programa, você deve possuir alguns dados, tais como: valor da hora aula, número de horas trabalhadas no mês e percentual de desconto do INSS. Em primeiro lugar deve-se estabelecer o seu salário bruto para fazer o desconto e ter o valor do salário líquido.

```
algoritmo "SALARIO_PROFESSOR"
var
    HT: inteiro
    VH, PD, TD, SB, SL: real
início
```

```

leia (HT)
leia (VH)
leia (PD)
SB <- HT * VH
TD <- (PD/100) * SB
SL <- SB - TD
escreva (SB)
escreva (SL)
finalgoritmo

```

1.8. EXERCÍCIOS

QUESTÃO 01. Classifique os dados abaixo com **I** para Inteiro, **R** para Real, **C** para Caractere e **L** para Lógico.

Exemplo:

(R) -1,87

() 1000

() -475

() 0

() "009"

() "IF

BAHIA"

() 0.898

() -100

() "#9CJSH"

() 45685

() "Verdadeiro"

() Falso

() 92,32

() -10

() "Cinco"

() -283

QUESTÃO 02. Assinale com um X os nomes válidos para uma variável:

() NOME_USUARIO

() SEXO-USUARIO

() _1NOME

() SALARIOEMR\$

() NOME USUARIO

() _TELEFONE

() ENDEREÇO

() 21BRASIL

() TELEFONE&RESIDENCIAL

QUESTÃO 03. Calcule o valor de cada expressão abaixo.

a) $(20 - 15)/2$

d) $2 * (5 \setminus 20) + 30 / (15 * 2)$

b) $20 - 15 \setminus 2 * 3 + 9$

e) $35 \bmod 6 + 2 ^ 5$

c) $2 * 5 / 20 + 30 / 15 * 2$

f) $35 \setminus 8 + 6 - 2 * 10 / 3$

QUESTÃO 04. Desenvolva a codificação em português estruturado para os seguintes programas:





- a) Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é: $F \leftarrow (9 * C + 160) / 5$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
- b) Calcular e apresentar o valor do volume de uma lata de óleo, utilizando a fórmula: $VOLUME \leftarrow 3.14159 * R^2 * ALTURA$.
- c) Construir um programa que calcule e apresente em metros por segundo o valor da velocidade de um projétil que percorre uma determinada distância em quilômetros a um determinado espaço de tempo em minutos. Utilize a fórmula: $VELOCIDADE \leftarrow (DISTÂNCIA * 1000) / (TEMPO * 60)$.
- d) Elaborar um programa de computador que calcule e apresente o valor do volume de uma esfera. Utilize a fórmula: $VOLUME \leftarrow (4 / 3) * 3.14159 * (RAIO^3)$.
- e) Elaborar um programa que leia dois valores desconhecidos representados pelas variáveis A e B. Calcular e apresentar os resultados das quatro operações aritméticas básicas.
- f) Ler o valor correspondente ao salário mensal (variável SM) de um trabalhador e também o valor do percentual de reajuste (variável PR) a ser atribuído. Armazenar e apresentar o valor do novo salário (variável NS).

LÓGICA DE
PROGRAMAÇÃO

PARTE 2:

Estruturas de Decisão

NESTE CAPÍTULO, ESTUDAREMOS A ESTRUTURA DE CONTROLE PARA TOMADA DE DECISÕES. SERÃO ABORDADOS OS SEGUINTESS ASSUNTOS:

-  **DESVIO CONDICIONAL SIMPLES, COMPOSTO E ENCADEADO**
-  **OPERADORES RELACIONAIS**
-  **OPERADORES LÓGICOS**
-  **COMANDO DE SELEÇÃO MÚLTIPLA**

2.1. ESTRUTURAS DE DECISÃO

Até o momento estudamos como trabalhar com entradas, processamentos e saídas com a utilização de variáveis, constantes e operadores aritméticos. Apesar de já conseguir solucionar problemas e transformá-los em programas, esses recursos são limitados.

Imagine um programa que em determinado momento precisa tratar um dado para realizar um processamento mais adequado. Exemplo: Verificar se um aluno atingiu uma determinada média. Nesse caso, torna-se necessário utilizar os recursos de tomada de decisão.

2.2. DESVIO SIMPLES

A tomada de decisão simples utiliza a instrução:

```
se (condição a ser testada) então
    //trecho de código a ser executado caso a condição
    testada seja verdadeira
fimse
```

Nesta instrução, se a condição estabelecida for verdadeira, serão executadas todas as instruções definidas entre **se...então** e **fimse** e depois serão executadas todas as instruções existentes após **fimse**. Se a condição for falsa, serão executadas as instruções que estiverem definidas após a instrução **fimse**.

Como exemplo, vamos considerar o seguinte problema: “Ler dois valores numéricos, efetuar a adição e apresentar o seu resultado caso o valor somado seja maior que 10”. Os passos para a execução deste problema seriam:

1. Ler dois valores (A e B)
2. Efetuar a soma dos valores A e B e armazenar o resultado da soma.
3. Exibir o resultado da soma caso seja maior que 10

No português estruturado representamos os passos acima através do seguinte algoritmo abaixo:


```
algoritmo "Somar Numeros"
var
    valor1, valor2, resultado_soma: inteiro
inicio
    escreva ("Digite o valor do numero 1: ")
    leia (valor1)
    escreva ("Digite o valor do numero 2: ")
    leia (valor2)
    resultado_soma <- valor1 + valor2

    se (resultado_soma > 10) entao
        escreva (" O resultado é: ", resultado_soma)
fimse
finalgoritmo
```

No algoritmo acima, após a definição dos nomes e tipos de variáveis (bloco var), é solicitada a leitura dos valores para as variáveis valor1 e valor2 através do comando leia. Depois esses valores são implicados na variável resultado soma, a qual possui o resultado da adição dos dois valores. Em seguida, questiona-se no programa uma condição que permitirá imprimir o resultado da soma caso ela seja maior que 10, e não sendo, o programa é encerrado sem apresentar a referida soma, uma vez que a condição é falsa.

2.3. OPERADORES RELACIONAIS

Ao usar uma instrução de tomada de decisão, é necessário definir para ela uma condição, que é o estabelecimento de uma relação lógica entre dois elementos, podendo ser: variável X variável ou variável X constante. No exemplo anterior a relação lógica estabelecida foi se o valor armazenado na variável **resultado_soma** é maior que 10 (resultado_soma > 10).

Para que uma relação lógica seja definida, passa a ser necessário usar de um dos operadores relacionais existentes, os quais se encontram definidos na tabela a seguir:

São seis os operadores lógicos presentes no português estruturado:

OPERADOR	SIGNIFICADO
=	IGUAL
>	MAIOR QUE
<	MENOR QUE
>=	MAIOR OU IGUAL A
<=	MENOR OU IGUAL A
<>	DIFERENTE DE

Exemplos: $A > B$, $valorA \neq valorB$, $delta \leq 0$, $X = -20$

2.4. DESVIO CONDICIONAL COMPOSTO

No desvio condicional composto utilizamos a instrução:

```

se (condição a ser testada) entao
    //trecho de código a ser executado caso a condição
testada seja verdadeira
senao
    //trecho de código a ser executado caso a condição
seja falsa
fimse

```

Nesta instrução, se a condição estabelecida for verdadeira, serão executadas todas as instruções definidas entre **se...entao**. Se a condição estabelecida for falsa, serão executadas todas as instruções que estiverem definidas entre **senao** e **fimse**. Somente após a execução de uma das possibilidades anteriores é que o programa executará as instruções existentes após **fimse**.

Como exemplo da utilização desta estrutura condicional, considere o seguinte problema:

“Ler dois valores numéricos e efetuar a adição. Caso o valor somado seja maior ou igual a 10, deve ser apresentado somando a ele 5, caso o valor somado não seja maior ou igual a 10, deve ser apresentado subtraído dele 7”

No português estruturado representamos os passos acima através do seguinte algoritmo abaixo:

```
algoritmo "Somar Numeros"
var
    A, B, R, T: inteiro
inicio
    escreva ("Digite o valor de A: ")
    leia (A)
    escreva ("Digite o valor de B: ")
    leia (B)
    R <- A + B

    se (R > 10) entao
        T <- R + 5
    senao
        T <- R - 7
    escreva (" O resultado da operação é: ", T)
fimse

finalgoritmo
```

No algoritmo acima após a definição das variáveis, é solicitada a leitura dos valores para as variáveis A e B (*leia(A) e leia(B)*), em seguida a adição desses dois valores é armazenada na variável R (*R <- A + B*). Em seguida o programa testa uma condição que permitirá imprimir o resultado da soma adicionando 5, caso ela seja maior ou igual a 10; e não sendo, o programa apresenta o resultado subtraindo 7.

Vamos a outro exemplo:

Construa um algoritmo que leia o preço de um produto (P) e apresente a mensagem: “Em promoção”, caso o preço seja maior ou igual a R\$ 50,00. Caso contrário, deve apresentar a mensagem: “Preço Normal”.

```
algoritmo "PROMOÇÃO"
var
    preco :real
inicio
    escreva ("DIGITE O PREÇO DO PRODUTO")
    leia (preco)
```

```

    se (preco >= 50) entao
        escreva ("PROMOÇÃO")
    senao
        escreva ("PREÇO NORMAL")
    fimse
fimalgoritmo

```

Para encerrar mais um exemplo: Num determinado Estado, para transferências de veículos, o DETRAN cobra uma taxa de 1% para carros fabricados antes de 1990 e uma taxa de 1.5% para os fabricados de 1990 em diante, taxa esta incidindo sobre o valor de tabela do carro. O algoritmo abaixo lê o ano e o preço do carro e a seguir calcula e imprime imposto a ser pago.

```

algoritmo "Detran"
var
    valorCarro, impostoPagar: real
    anoCarro: inteiro
inicio

    escreva ("INFORME O ANO DO VEÍCULO: ")
    leia (anoCarro)
    escreva ("INFORME O PREÇO DO VEÍCULO EM R$: ")
    leia (valorCarro)

    se (anoCarro < 1990) entao
        impostoPagar <- valorCarro * 0.01
    senao
        impostoPagar <- valorCarro * 0.015
    fimse

    escreval ("O VALOR DE IMPOSTO A PAGAR É DE R$: ",
impostoPagar)

fimalgoritmo

```

2.5. DESVIO CONDICIONAL ANINHADO

Existem casos em que é necessário estabelecer algumas verificações lógicas de condições definidas sucessivamente. A partir do momento em que uma determinada ação é executada, ela pode também levar a outras condições, de forma que não haja

limites. Dessa forma existe a possibilidade de usar uma condição dentro de outra condição, o que leva a uma estrutura de decisão encadeada ou aninhada.

Podemos representar um desvio condicional aninhado no português estruturado da seguinte forma:

```
se (condição 1 a ser testada) entao
    //trecho de código a ser executado caso a condição 1
    testada seja verdadeira
senao
    //trecho de código a ser executado caso a condição 1
    seja falsa
        se (condição 2 a ser testada) entao
            //trecho de código a ser executado caso a
            condição 2 testada seja verdadeira
        senao
            //trecho de código a ser executado caso a condição 2
            seja falsa
        fimse
    fimse
```

Observe que dentro da estrutura senão da primeira condição, existe uma outra estrutura **se**. Uma observação neste ponto é que o programador deverá encerrar as estruturas de condição uma de cada vez, neste caso primeiro colocamos o **fimse** da segunda condição e por último o **fimse** da primeira condição.

Considere o seguinte problema: “Elaborar um programa que efetue o cálculo do reajuste de salário de um funcionário. Considere que o funcionário deve receber um reajuste de 15% caso seu salário seja menor que 500. Se o salário for maior ou igual a 500, mas menor ou igual a 1000, seu reajuste será de 10%; caso seja ainda maior que 1000, o reajuste será de 5%.”

No português estruturado representamos os passos acima através do seguinte algoritmo abaixo:

```
algoritmo "Reajusta Salario"
var
    salario,novo_salario: real
```

```
inicio
    escreva ("Digite o salario atual do empregado: ")
    leia (salario)

    se (salario < 500) entao
        novo_salario <- salario * 1.15
    senao
        se (salario <= 1000) entao
            novo_salario <- salario * 1.10
        senao
            novo_salario <- salario * 1.05
        fimse
    fimse

    escreva ("O empregado teve o salario reajustado para: ", novo_salario)

fimalgoritmo
```

Agora outro exemplo:

Escreva um algoritmo que funcione como uma máquina de calcular operando da seguinte maneira:

- a) primeiro deve pedir os dois operandos;
- b) logo a seguir deve pedir o tipo de operação (+, -, * ou /)
- c) após a escolha do operador, deve apresentar o resultado indicando que tipo de operação foi executado.

Ex: $a + b = 6$;

- d) se o operador não for um dos especificados, mensagem deve informar o fato.

```
algoritmo "Calculadora"
var
    op1, op2: real
    operacao: caractere
inicio

    escreva ("INFORME O OPERANDO 01: ")
    leia (op1)
```

```

    escreva ("INFORME O OPERANDO 02: ")
    leia (op2)

    escrevaL ("DIGITE: ")
    escrevaL (" + :PARA FAZER UMA OPERAÇÃO DE ADIÇÃO.")
    escrevaL (" - :PARA FAZER UMA OPERAÇÃO DE SUBTRAÇÃO")
    escrevaL ("  *  :PARA FAZER UMA OPERAÇÃO DE
MULTIPLICAÇÃO")
    escrevaL (" / :PARA FAZER UMA OPERAÇÃO DE DIVISÃO")
    escreva ("OPERANDO: ")

    leia (operacao)

    se (operacao = "+") entao
        escreva("A ADIÇÃO DO OPERANDO 1 + OPERANDO 2 = ",
op1 + op2)
    senao
        se (operacao = "-") entao
            escreva("A SUBTRAÇÃO DO OPERANDO 1 - OPERANDO
2 = ", op1 - op2)
        senao
            se (operacao = "*") entao
                escreva("A MULTIPLICAÇÃO DO OPERANDO 1 *
OPERANDO 2 = ", op1 * op2)
            senao
                escreva("A DIVISÃO DO OPERANDO 1 /
OPERANDO 2 = ", op1 / op2)
            fimse
        fimse
    fimse

    fimse

    fimalgoritmo

```

2.6. OPERADORES LÓGICOS

Em algum momento pode haver a necessidade de trabalhar com mais de uma condição dentro de uma única decisão, e fazer alguns testes lógicos múltiplos. Quando houver a necessidade de utilizar mais de uma condição para uma mesma tomada de decisão deve ser utilizado um recurso denominado **operador lógico**.

Os operadores lógicos mais comuns são: e (conjunção), ou (disjunção) e o não (negação)

2.6.1. OPERADOR “E”

O operador lógico de conjunção “e” é utilizado quando duas ou mais condições de uma determinada decisão necessitam ser verdadeiras para obter-se resultado lógico verdadeiro; caso contrário, o resultado do valor lógico retornado será falso. Se a primeira condição possui valor lógico falso, a segunda condição não precisa ser avaliada.

Dessa forma o resultado lógico é verdadeiro quando todas as condições envolvidas na decisão são verdadeiras.

No português estruturado, este operador é representado da seguinte forma:

```
se (condição1 a ser testada) e (condição2 a ser
testada) entao
    //trecho de código a ser executado caso a condição1 e
a condição2 sejam verdadeira
senao
    //trecho de código a ser executado caso uma ou todas
as condições sejam falsas
fimse
```

OBSERVAÇÃO: Para que o resultado lógico seja verdadeiro, todas as condições têm que ser verdadeiras.

Considere o seguinte problema: **A Justiça Eleitoral de um município desenvolveu um programa para cadastramento de eleitores. Esse programa possui uma rotina para verificar se um eleitor é obrigado a votar ou não. Consideremos que todos os eleitores são alfabetizados e o programa foi desenvolvido para a eleição de 2008. O programa solicita do eleitor o ano de seu nascimento e informa se ele está obrigado ou não a participar das eleições.**

No problema acima, o algoritmo deverá calcular a idade do eleitor e verificar se sua idade está entre 18 e 70 anos.

Com a utilização do operador “e”, o exemplo mostra que somente será apresentado que o eleitor é obrigado a participar das eleições caso sua idade seja maior ou igual a dezoito e menor ou igual a 70 anos.

Veja como fica agora a representação acima no português estruturado.

```
algoritmo "VERIFICA ELEITOR"
var
ano_nascimento, idade: inteiro
inicio
escreva ("Informe o ano de nascimento do eleitor: ")
leia (ano_nascimento)
idade <- 2008 - ano_nascimento
se (idade >= 18) e (idade <= 70) entao
    escreva ("Eleitor obrigado a participar das
eleições")
senao
    escreva ("O eleitor não é obrigado a participar das
eleições")
fimse
fimalgoritmo
```

2.6.2. OPERADOR “OU”

O operador lógico de disjunção “ou” é utilizado quando pelo menos uma de duas ou mais condições de uma determinada decisão necessita ser verdadeiro para obter-se um resultado lógico verdadeiro, caso contrário o valor do resultado lógico retornado será falso.

Dessa forma se a primeira condição possuir valor lógico verdadeiro a segunda condição não precisa ser avaliada. Sendo assim o resultado lógico é verdadeiro quando pelo menos uma das condições envolvidas na decisão é verdadeira.

No português estruturado, este operador é representado da seguinte forma:

```
se (condição1 a ser testada) ou (condição2 a ser
testada) entao
    //trecho de código a ser executado caso a condição1 ou
a condição2 sejam verdadeira
senao
```

```
//trecho de código a ser executado caso todas as  
condições sejam falsas  
fimse
```

OBSERVAÇÃO: Para que o resultado lógico seja verdadeiro, apenas uma das condições tem que ser verdadeiras.

Considere o seguinte exemplo: **Em um programa de computador é feito um teste para verificar se o usuário informou o seu estado civil de forma correta no sistema. O sistema apenas aceita os seguintes valores: solteiro e casado.**

Com a utilização do operador lógico “ou”, o algoritmo verifica se um dos valores acima foi digitado. Caso um dos valores seja verdadeiro o sistema emite uma mensagem: “Estado civil válido”; caso os dois valores sejam falsos é mostrado uma mensagem informando que o estado civil é inválido.

Veja como fica a representação do problema acima no português estruturado.

```
algoritmo "VERIFICA ESTADO CIVIL"  
  
var  
estado_civil: caractere  
inicio  
escreva ("Informe o estado civil: ")  
leia (estado_civil)  
  
se (estado_civil = "Solteiro") ou (estado_civil =  
"Casado") entao  
    escreva ("Estado civil válido")  
senao  
    escreva ("Estado civil inválido")  
fimse  
  
fimalgoritmo
```

2.6.3. OPERADOR “NAO”

Por último temos o operador lógico “*nao*”. Este operador faz com que seja executada uma determinada ação de uma decisão invertendo o seu resultado lógico. Se a condição for **verdadeira** e possuir a sua frente o operador “*nao*”, esta será automaticamente considerada **falsa**, o inverso ocorrerá para uma condição **falsa** que possua a sua frente o operador “*nao*”, que será automaticamente considerada **verdadeira**.

Resumindo, o operador “*nao*” faz com que seja executada uma determinada operação, invertendo o resultado lógico da condição.

No português estruturado, este operador é representado da seguinte forma:

```
se nao (condição 1 a ser testada) entao
// instruções executadas se a condição não for
verdadeiras
senao
//trecho de código a ser executado se a condição for
verdadeira
fimse
```

OBSERVAÇÃO: “*se nao*” é diferente de “*senao*”

O exemplo a seguir mostra a utilização do operador não, que somente será efetuado o cálculo de $C \leftarrow (A+B) * X$, se o valor da variável X não for maior que 5. Qualquer valor de 5 para baixo será efetuado o cálculo $C \leftarrow (A + B) * X$.

```
algoritmo "TESTE LÓGICO NÃO"

var
A,B,C,X : inteiro
inicio
leia (A, B, X)
se nao (x > 5) entao
    C ← (A+B) * X
senao
    C ← (A-B) * X
fimse
escreva (C)
```

2.6.4. PRIORIDADE DOS OPERADORES LÓGICOS

Os operadores lógicos possibilitam o uso de mais de uma condição para a tomada de uma única decisão. Para usar adequadamente os operadores lógicos, é necessário levar em conta sua ordem de prioridade. Veja na tabela abaixo a ordem padrão de prioridade.

OPERADOR	OPERAÇÃO	PRIORIDADE
Não	Negação	1
E	Conjunção	2
Ou	Disjunção	3

Por exemplo, a expressão lógica:

$(A = B)$ e não $(A > 5)$, deve ser avaliada a partir de não $(A > 5)$ e somente depois de saber seu resultado é que pode ser realizada avaliação do restante da expressão lógica.

Já a expressão $(A = 1)$ ou $(A \geq 4)$ e $(A \leq 9)$ será resolvida primeiro a parte da expressão submetida ao operador “e” para depois ser resolvida a parte da expressão submetida ao operador “ou”. Caso exista a necessidade de primeiro ser executada a avaliação lógica da expressão “ou”, neste caso a avaliação lógica deve ser definida entre parênteses como sendo: $((A = 1)$ ou $(A \geq 4))$ e $(A \leq 9)$

Para testar o aprendizado, considere o exemplo a seguir:

Sabendo que $A=5$, $B=4$ e $C=3$ e $D=6$, informe se as expressões abaixo são verdadeiras ou falsas.

- a) $(A > C)$ e $(C \leq D)$ ()
- b) $(A+B) > 10$ ou $(A + B) = (C + D)$ ()
- c) $(A \geq C)$ e $(D \geq C)$ ()

Respostas:

- a)V
- b)V

c)F

2.7. COMANDOS DE SELEÇÃO MÚLTIPLA

Em algumas situações ao chegarmos a uma determinada instrução de um algoritmo devemos selecionar um dentre alguns trechos a seguir, tendo como base para esta escolha um conjunto de valores.

Exemplo: Testar o valor do estado civil de um determinado usuário (Solteiro,Casado,Divorciado,Viúvo), Testar o tipo sanguíneo (A,B,O,AB), entre outros exemplos.

Para lidar com casos deste tipo foi criado o comando de seleção múltipla, que substitui as instruções de “se...senao” aninhados.

Em um determinado algoritmo é necessário escrever um código para testar o tipo sanguíneo informado pelo usuário. Poderíamos resolver esse problema utilizando uma estrutura de tomada de decisão de forma aninhada, conforme o exemplo abaixo:

```
se tipoSanguineo = "A" entao
senao
    se tipoSanguineo = "B" entao
    senao
        se tipoSanguineo = "AB" entao
        senao
            fimse
        fimse
    fimse
fimse
```

Como resolver esse problema utilizando Comandos de Seleção Múltipla? Este comando consiste de uma expressão (o seletor) e uma lista de comandos. Ou seja, caso o valor de uma variável for igual ao que uma lista de alternativas testa, então o comando desta variável será executado.

Depois da execução o controle vai para o próximo comando após o **CASO**. Se não existir nenhum rótulo que satisfaça essa condição, o efeito do comando fica indefinido.

O comando de seleção múltipla possui a seguinte representação no português estruturado:

```
escolha (variável)
caso valor11, valor12, ..., valor1n
//seqüência-de-comandos-1
caso valor21, valor22, ..., valor2m
//seqüência-de-comandos-2
...
outrocaso
//seqüência-de-comandos-extra
fimescolha
```

O valor de expressão é calculado e comparado seqüencialmente com cada uma das opções valor1, valor2, etc.

Se uma das opções for igual ao valor da expressão, a execução continua a partir dos comandos associados a esse valor.

Se nenhuma das opções for igual ao valor da expressão, a opção **outrocaso** será selecionada e a execução irá continuar a partir dos comandos associados à mesma.

Veja agora como ficaria o exemplo do Tipo Sanguíneo:

```
escolha tipoSanguineo
caso "A"
    escreva ("A")
caso "B"
    escreva ("B")
caso "AB"
    escreva ("C")
caso "O"
    escreva ("O")
outrocaso
    escreva ("Tipo sanguíneo inválido")
fimescolha
```

2.8. EXERCÍCIOS

QUESTÃO 01. Um usuário deseja um algoritmo onde possa escolher que tipo de média deseja calcular a partir de 3 notas. Faça um algoritmo que leia as notas, a opção escolhida pelo usuário e calcule a média.

As opções podem ser:

1 - aritmética

2 - ponderada (3, 3,4)

QUESTÃO 02. Faça um algoritmo que leia um nº inteiro e mostre uma mensagem indicando se este número é par ou ímpar, e se é positivo ou negativo.

QUESTÃO 03. A prefeitura de Eunápolis abriu uma linha de crédito para seus servidores. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto. Através do salário bruto e do valor da prestação, informar se o empréstimo pode ou não ser concedido.

QUESTÃO 04. Escreva um programa em que calcule o valor a ser pago por produtos de uma loja, considerando o preço final da compra e a escolha da condição de pagamento. Os códigos da tabela abaixo devem ser seguidos para ler qual a condição de pagamento escolhida e a efetuação do cálculo.

Código	Condição de Pagamento
1	À vista em dinheiro ou cheque, recebe 10% de desconto
2	À vista no cartão de crédito, recebe 5% de desconto
3	Em 2 vezes, preço normal de etiqueta sem juros
4	Em 3 vezes, preço normal de etiqueta mais juros de 10%

QUESTÃO 05. Escreva um algoritmo que leia as idades de 2 homens e de 2 mulheres (*considere que as idades dos homens serão sempre diferentes entre si, bem como as das mulheres*). Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova, e o produto das idades do homem mais novo com a mulher mais velha.

QUESTÃO 06. Ler os valores de quatro notas escolares de um aluno. Calcular a média aritmética e apresentar a mensagem “Aprovada” se a média obtida for maior ou igual a 7; caso contrário, o programa deve solicitar a nota do exame de recuperação e calcular uma nova média aritmética entre a nota de exame de recuperação e a primeira média aritmética. Se o valor da nota da nova média for maior ou igual a sete,

apresentar a mensagem “Aprovada na Recuperação”, caso contrário apresentar a mensagem “Reprovado”. Informar junto com a mensagem o valor da média obtida.

QUESTÃO 07. Uma fruteira está vendendo frutas com a seguinte tabela de preços:

	Até 5 Kg	Acima de 5 Kg
Morango	R\$ 2,50 por Kg	R\$ 2,20 por Kg
Maçã	R\$ 1,80 por Kg	R\$ 1,50 por Kg

Se o cliente comprar mais de 8 Kg em frutas ou o valor total da compra ultrapassar R\$ 25,00, receberá ainda um desconto de 10% sobre este total. Escreva um programa para ler a quantidade (em Kg) de morangos e a quantidade (em Kg) de maçãs adquiridas e escreva o valor a ser pago pelo cliente.

QUESTÃO 08. Determine o resultado das expressões lógicas (V ou F). Considere para as respostas os seguintes valores: $X = 1$, $A = 3$, $B = 5$, $C = 8$, $D = 7$

- a) não $(X > 3)$ ()
- b) $(X < 1)$ e não $(B > D)$ ()
- c) não $(X > 3)$ ou $(C < 7)$ ()
- d) $(A > B)$ ou $(C > B)$ ()
- e) $(A > B)$ ou não $(C > B)$ ()
- f) não $(D > 3)$ ou não $(B < 7)$ ()
- g) $(D < 0)$ ou $(C > 5)$ ()

QUESTÃO 09. Ler três valores para os lados de um triângulo, considerando lados como: A, B e C. Verificar se os lados formam um triângulo e se for verdadeiro, deve ser indicado qual tipo de triângulo foi formado: isósceles, escaleno ou equilátero.

Obs.: um triângulo é uma forma geométrica composta de três lados e o valor de cada lado deve ser menor que a soma dos valores dos outros dois lados.

QUESTÃO 10. Desenvolver um programa que leia um valor numérico inteiro e apresente-o caso seja divisível por 4 e 5. Não sendo divisível por 4 e 5, o programa deve apresentar a mensagem: “Não é divisível por 4 e 5”.

Obs.: Para resolver esse problema é necessária a utilização do operador aritmético mod que permite obter o resto da divisão de um valor por outro.

QUESTÃO 11. A OMS – Organização Mundial de Saúde adotou o seguinte critério de classificação para verificar se o peso das pessoas está dentro dos valores permitidos (normais) ou não.

Valor Apurado	Situação
Até 26	Normal
Entre 26 e 30 (inclusive)	Acima do Peso
Acima de 30	Obeso

Para obter estes valores da classificação a operação que se deve realizar é peso dividido pela altura ao quadrado. O valor obtido desta operação deve ser aplicado à tabela apresentada acima. Apresentar a situação de cada pessoa que está sendo consultada.

QUESTÃO 12. Calcular o valor do salário e, o desconto de Imposto de Renda, com base na tabela abaixo.

Base de Cálculo (R\$)	Alíquotas (%)	Parcela a Deduzir (R\$)
Até 1.058,00	7,65	----
De 1.058,01 até 2.115,00	15,00	158,70
Acima de 2.115,01	27,50	423,08

Apresentar o salário bruto, o salário líquido o valor de desconto de IR de cada funcionário e, o total de retenção na fonte realizado por esta empresa.

QUESTÃO 13. Leia mês e ano do teclado e mostre o número de dias correspondente ao mês lido.

QUESTÃO 14. Criar um programa para ler um número de 1 a 7 e informar o dia da semana correspondente, sendo domingo o dia de número 1. Se o número não corresponder a um dia da semana, é mostrada uma mensagem de erro.

LÓGICA DE
PROGRAMAÇÃO

PARTE 3:

Estruturas de Repetição






NESTE CAPÍTULO, ESTUDAREMOS UMA ESTRUTURA DE CONTROLE CONHECIDA COMO LOOPS OU MALHA DE REPETIÇÃO. A ESTRUTURA DE REPETIÇÃO ESTUDADA SERÁ:

- REPETIÇÃO COM TESTE LÓGICO NO INÍCIO E NO FINAL DO LAÇO
- REPETIÇÃO DA VARIÁVEL DE CONTROLE

3.1. ESTRUTURAS DE REPETIÇÃO

Existem ocasiões em que é necessário repetir um trecho do programa um determinado número de vezes. Neste caso, pode ser criado um laço que faça o processamento de um determinado trecho tantas vezes quantas forem necessárias. Os laços também são chamados “*loopings*” ou malhas de repetição.

Imagine um programa que executa um determinado trecho de instruções por cinco vezes. Com o conhecimento adquirido até o momento, repetiríamos o trecho o número de vezes necessário.

<code>leia (x)</code> <code>resultado <- x * 3</code> <code>escreva (resultado)</code>		1x
<code>leia (x)</code> <code>resultado <- x * 3</code> <code>escreva (resultado)</code>		2x
<code>leia (x)</code> <code>resultado <- x * 3</code> <code>escreva (resultado)</code>		3x
<code>leia (x)</code> <code>resultado <- x * 3</code> <code>escreva (resultado)</code>		4x
<code>leia (x)</code> <code>resultado <- x * 3</code> <code>escreva (resultado)</code>		5x

Observe que no exemplo acima, um mesmo trecho de código é repetido por 5 vezes. Imagine a situação caso seja necessário fazer a leitura e processamento de 1000 valores para o exemplo. **Teríamos que repetir esse trecho de código 1000 vezes?**

Para estes casos existem comandos apropriados para repetir determinados trechos de programas o número de vezes que for necessário. Dessa forma é possível determinar repetições com número variados de vezes, dependendo da forma de laço a ser utilizada.

3.2. REPETIÇÃO COM TESTE LÓGICO NO INÍCIO DO LAÇO

Nessa estrutura é feito um teste lógico no início de um laço, verificando se é permitido executar o trecho de instruções subordinado a esse laço. Essa estrutura é denominada de **enquanto**.

No português estruturado essa estrutura é representada da seguinte forma:

```
enquanto (<condição>) faca  
< instruções para condição >  
fimenquanto
```

A estrutura **enquanto...faca...fimenquanto** tem o seu funcionamento controlado por decisão. Sendo assim, pode executar um determinado conjunto de instruções enquanto a condição for verdadeira. No momento em que a condição for falsa, o processamento da rotina é desviado para fora do laço. Se a condição for falsa logo de início, as instruções contidas no laço são ignoradas.

Considere que em um determinado programa de computador, serão lidos por cinco vezes do usuário um determinado valor. Após o usuário informar um número, o computador deverá mostrar o produto por 5 desse número. Esse processo poderia ser definido conforme a seqüência de passos abaixo:

1. Criar uma variável para servir como contador com valor inicial 1.
2. Enquanto o valor do contador for menor ou igual a 5, processar os passos 3,4 e 5.
3. Ler um valor para a variável X
4. Efetuar a multiplicação do valor de X por 5, colocando o resultado em R.
5. Apresentar o valor calculado que está na variável E.
6. Acrescentar 1 à variável do tipo contador, definida no passo 1
7. Quando contador for maior que 5, encerrar o processamento do laço

Observe que na seqüência de passos acima estamos utilizando uma variável como uma espécie de contador, de forma que possa contar quantas vezes o laço deverá se repetir, neste caso 5 vezes.

Veja como ficaria a implementação desse algoritmo no Português Estruturado:

```
algoritmo "Exemplo Enquanto...Faca"
var
    contador, valor: inteiro
inicio
    contador <- 1
    enquanto contador <= 5 faca
        escreva("Valor: ")
        leia (valor)
        escreval ("Multiplicado por 5 = ", valor * 5)
        contador <- contador + 1
    fimenquanto
fimalgoritmo
```

Na situação acima, a variável contador assume valor inicial 1 (**contador<-1**). Todo o código a ser repetido é controlado pela estrutura: **enquanto contador <=5 faca**. Dentro da estrutura de repetição escrevemos o código que será repetido. Antes de finalizar o código a ser repetido é necessário alterar o valor da variável contador, que neste caso tem seu valor acrescido em 1 (**contador <- contador + 1**). Isso é necessário para que o valor da variável contador assuma um novo valor cada vez que entrar no laço de repetição, de forma que ao final a variável contador possua valor 5 e o laço de repetição seja finalizado.

Para ilustrar de forma um pouco diferente, imagine que o problema anterior deverá ser executado enquanto o usuário queira. Em vez de possuir dentro da rotina um contador de vezes, é possível ter uma instrução pedindo que o usuário informe se deseja continuar ou não a repetição.

Esse processo poderia ser definido conforme a seqüência de passos abaixo:

1. Criar uma variável para ser utilizada como resposta
2. Enquanto a resposta for sim, executar os passos 3,4 e 5
3. Ler um valor para a variável X
4. Efetuar a multiplicação do valor de X por 3, colocando o resultado em R
5. Apresentar o valor calculado que está na variável R
6. Quanto a resposta for diferente de sim, apresentar o processamento.

Veja como ficaria a implementação desse algoritmo no Português Estruturado:

```

algoritmo "Exemplo Enquanto...Faca"
var
    valor: inteiro
    resp :caractere
inicio
    resp <- "S"
enquanto resp = "S" faca
    escreva("Valor: ")
    leia (valor)
    escreval("Multiplicado por 3 = ", valor * 3)
    escreva ("Deseja Continuar? S - para Sim / N -
para não: ")
    leia (resp)
fimenquanto
fimalgoritmo
  
```

Na situação acima, a variável **resp** assume valor inicial **"S"** (**resp<-"S"**). Todo o código a ser repetido é controlado pela estrutura: **enquanto resp = "S" faca**. Dentro da estrutura de repetição escrevemos o código que será repetido. Antes de finalizar o código a ser repetido é necessário perguntar ao usuário se ele deseja ou não continuar a repetir o código. Isso é necessário para que o valor da variável **resp** assuma um novo valor cada vez que entrar no laço de repetição, de forma que o laço de repetição seja encerrada quando o valor da variável **resp** receber um valor diferente de **"S"**.

3.3. REPETIÇÃO COM TESTE LÓGICO NO FIM DO LAÇO

Nesta estrutura é realizado um teste lógico no fim de um laço. Ela é parecida com a estrutura **enquanto**. Essa estrutura é denominada de **repita**, a qual é conseguida com a utilização do conjunto de instruções **repita ... ate**

Representação em português estruturado:

```

repita
< instruções para condição >
ate (<condição>)
  
```

A estrutura **repita...ate** tem seu funcionamento controlado por decisão, porém executa um conjunto de instruções pelo menos uma vez antes de verificar a validade

da condição estabelecida. Desta forma repita tem seu funcionamento em sentido contrário a **enquanto**, pois sempre processa em conjunto de instruções no mínimo uma vez até que a condição seja verdadeira. Se a condição for falsa, o laço continua. Se a condição for verdadeira, o laço será encerrado

Para ilustrar a utilização de **repita**, será usado o mesmo exemplo anterior: “Pedir a leitura de um valor para a variável Valor, multiplicar esse valor por 3 e mostrar o resultado para o usuário. Continuar essa seqüência até que o usuário responda que não deseja continuar a inserir novos valores”. Com a utilização do comando **repita**, a variável **RESP** não precisa ser inicializada com nenhum valor, uma vez que no mínimo serão sempre executadas as instruções constantes no laço, para somente no final ser verificada a condição, no caso **RESP = “N”**.

Veja como ficaria a implementação desse algoritmo no Português Estruturado:

```
algoritmo "Exemplo Com Repita...Ate"
var
    valor: inteiro
    resp :caractere
inicio
repita
    escreva("Valor: ")
    leia (valor)
    escreval("Multiplicado por 3 = ", valor * 3)
    escreva ("Deseja Continuar? S - para Sim / N -
para não: ")
    leia (resp)
ate Resp = "N"
fimalgoritmo
```

3.4. REPETIÇÃO COM VARIÁVEL DE CONTROLE

Anteriormente foram vistas duas formas de elaborar laços. Uma utilizando **enquanto** e a outra usando **repita**. Foi visto também como elaborar programas que usaram um laço um determinado número de vezes com a utilização de um contador (por meio de uma variável de controle)

Existe uma possibilidade de facilitar o uso de laços sem usar as duas estruturas anteriores, por meio de estrutura de laços conhecida como “para”, sendo conseguida com a utilização do conjunto de instruções **para ... de ... ate ... passo ... faca ... fimpara**

Essa estrutura tem o seu funcionamento controlado por uma variável denominada contador que inicia e termina com um determinado valor informado pelo programador.

Representação em português estruturado

```
para <variável> de <inicio> ate <fim> passo <incremento> faca  
    < instruções>  
fimp
```

Para exemplificar vamos considerar o problema trabalhado nos exemplos anteriores: “Pedir a leitura de um valor para a variável X, multiplicar esse valor por 3, colocá-lo na variável de resposta R e apresentar o valor obtido, repetindo esta sequência por cinco vezes.”

No decorrer do estudo sobre laços ou malhas de repetição foram apresentados três estruturas de controle baseadas nos laços: enquanto, repita e para, cada qual com sua característica de processamento. Dentro desse aspecto, deve-se notar que as estruturas mais versáteis são **enquanto** e **repita**, pois podem ser substituída uma pela outra além de poderem substituir a estrutura **para**. É preciso considerar que nem toda estrutura **enquanto** ou **repita** pode ser substituída por uma estrutura **para**. Isso ocorre quando em uma estrutura utilizam-se condições que não envolvam o uso de variáveis de controle como contador.

3.5. ESTRUTURAS DE CONTROLE DE REPETIÇÃO ANINHADAS

Durante o estudo das estruturas de decisão foi discutido o fato de ocorrer o encadeamento das estruturas de decisão.

Exemplo:

```
se sexo = "Masculino" entao  
se idade < 18 entao  
    //instruções  
senao  
    se idade >=18 e idade <=25 entao  
        //instruções  
    senao  
        //instruções  
fimse  
fimse
```


Esse fato pode também ocorrer com as estruturas de laço. E nesse ponto pode ocorrer o encadeamento de um tipo de estrutura de laço com outro tipo de estrutura de laço.

Essas ocorrências vão depender do problema a ser selecionado. Dessa forma podemos ter uma estrutura **repita...ate** dentro de uma estrutura **enquanto...faca**, ou uma estrutura **para...faca** dentro de uma estrutura **enquanto...faca**, etc.

Alguns exemplos dessas estruturas encadeadas são:

1. enquanto ... faca

```
enquanto (condição1) faca
    //instruções
enquanto (condição2) faca
    //instruções
fimenquanto
fimenquanto
```

```
enquanto (condição1) faca
    //instruções
    repita
    //instruções
    ate (condição 2)
fimenquanto
```

```
enquanto (condição1) faca
    //instruções
    para <var> de <inicio> ate <fim> passo <incr> faca
    //instruções
    fimpara
fimenquanto
```

2. repita ... ate

```
repita
    //instruções
    repita
    //instruções
    ate (condição 2)
```

```
ate (condição 1)
```

```
repita
//instruções
    enquanto (condição 2) faca
    //instruções
fimenquanto
ate (condição 1)
```

```
repita
    //instruções
para <var> de <inicio> ate <fim> passo <incr> faca
//instruções
fimpara
ate (condição1)
```

3. para ... de ... ate ... faca

```
para <var1> de <inicio> ate <fim> passo <incr> faca
//instruções
    para <var2> de <inicio> ate <fim> passo <incr> faca
    //instruções
    fimpara
fimpara
```

```
para <var1> de <inicio> ate <fim> passo <incr> faca
//instruções
    enquanto (condição 1) faca
    //instruções
    fimenquanto
fimpara
```

```
para <var1> de <inicio> ate <fim> passo <incr> faca
//instruções
    repita
    //instruções
    ate (condição 1)
fimpara
```

3.6. EXERCÍCIOS DE APRENDIZAGEM

Exemplo 01. Escrever um programa que apresente o total da soma dos cem primeiros números inteiros (1+2+3+4+5...+100)

Para resolver o problema acima, teríamos que obter cada um dos números de 1 a 100 e somá-los à medida que cada número for encontrado. Para isso escreveremos uma estrutura de repetição usando uma variável **contador** que assumirá cada um desses valores, e para cada valor encontrado iremos armazená-lo na variável **soma**. A variável soma é responsável por acumular a soma dos valores encontrados.

OBS: No problema a variável contador é incrementada de 1 em 1 (contador <- contador + 1) para encontrar os números de 1 a 100.

Veja a resolução deste problema:

```
algoritmo "soma"
var
contador, soma: inteiro
inicio
contador <- 1
soma <- 0

enquanto contador <= 100 faca
soma <- soma + contador
contador <- contador + 1
fimenquanto

escreva("SOMA DOS 100 PRIMEIROS NUMEROS: ", soma)
finalgoritmo
```

Exemplo 02. Faça um algoritmo para ler uma quantidade de números. Depois de ler todos os números o algoritmo deve apresentar na tela o maior dos números lidos e a média dos números lidos.

Para resolver o problema acima, teríamos que obter vários números e somá-los à medida que cada número for informado pelo usuário. Para isso escreveremos uma estrutura de repetição usando uma variável **resposta** que confirmará ou não se o

usuário continuará inserindo novos valores. Utilizaremos também uma variável para guardar a soma dos valores informados e uma variável maior, que tem a função de verificar se um número informado é maior que o atual valor da variável maior. Caso verdadeiro, substituiremos o valor. Ao final, quando o usuário decidir não inserir mais dados, o algoritmo exibirá os resultados.

Veja a resolução deste problema:

```
algoritmo "soma"
var
soma, numero, maior, contador: inteiro
resposta: caractere
inicio
soma <- 0
maior <- 0
contador <- 0

repita
escreva("INFORME UM NÚMERO: ")
leia(numero)

se numero > maior entao
    maior <- numero
fimse

soma <- soma + numero
contador <- contador + 1

escreva("DESEJA CONTINUAR A INSERIR NOVOS VALORES: ")
leia(resposta)

ate resposta = "n"

escreval("MAIOR NÚMERO: ", maior)
escreval("MÉDIA: ", soma/contador)

fimalgoritmo
```

Exemplo 03. Desenvolver um algoritmo que leia a altura e o sexo (M ou F) de 15 pessoas. Este programa deverá calcular e mostrar:

- A menor altura do grupo;
- A média de altura das mulheres;
- O número de homens;
- O sexo da pessoa mais alta.

Para resolver o problema acima, teremos que criar uma estrutura de repetição para solicitar cada um dos dados(altura e sexo). A cada entrada de dados o algoritmo calculará os resultados: menor altura, média, número de homens e sexo da pessoa mais alta. Ao final da leitura dos 15 dados, o algoritmo apresentará os resultados.

OBS: Apesar do exemplo abaixo ter sido desenvolvido com a estrutura **repita ... ate**, é comum os programadores utilizarem a estrutura **para ... de ... ate** nessas situações.

```
algoritmo "Questão 05"

var
contador, total_homens, qtd_m: inteiro
sexo, sexo_mais_alto, resposta: caractere
altura, menor_altura, soma_altura_mulheres, maior_altura: real

inicio

menor_altura <- 3
maior_altura <- 0

repita

    escreva("ALTURA: ")
    leia(altura)
    escreva("SEXO: ")
    leia(sexo)

    se altura < menor_altura entao
        menor_altura <- altura
    fimse

    se sexo = "F" entao
        soma_altura_mulheres <- soma_altura_mulheres +
altura
```

```
        qtd_m <- qtd_m + 1
    senao
        total_homens <- total_homens + 1
    fimse

    se altura > maior_altura entao
        sexo_mais_alto <- sexo
    fimse

    escreva ("DESEJA CONTINUAR A INSERIR DADOS?")
    leia (resposta)

ate resposta = "N"

escreval ("MENOR ALTURA: ",menor_altura)
escreval ("MÉDIA ALTURA DAS MULHRES: ",soma_altura_mulheres
/ qtd_m)
escreval ("NÚMERO DE HOMENS: ",total_homens)
escreval ("SEXO PESSOA MAIS ALTA: ", sexo_mais_alto)

fimalgoritmo
```

3.7. EXERCÍCIOS

QUESTÃO 01. Apresentar a tabuada (multiplicação de 1 a 9) de um número qualquer informado pelo usuário, a qual deve ser impressa no seguinte formato:

- 2 X 1 = 2
- 2 X 2 = 4
- 2 X 3 = 6
-

QUESTÃO 02. Elaborar um programa que apresente os valores de conversão de graus Celsius em Fahrenheit, de 10 em 10, iniciando a contagem em 10 graus e finalizando em 100 graus Celsius. O programa deve apresentar os valores das duas temperaturas.

Fórmula de conversão: $F \leftarrow (9 \times C + 160) / 5$.

QUESTÃO 03. Elaborar um programa que efetue o cálculo da fatorial do número 5. Fatorial é o produto dos números naturais desde 1 até o inteiro n.

QUESTÃO 04. Com base no exemplo anterior, elaborar um programa que efetue o cálculo da fatorial de um número qualquer informado pelo usuário.

QUESTÃO 05. Elaborar um programa que apresente no final o somatório dos valores pares existentes em uma faixa de valores informado pelo usuário. Exemplo: de 1 até 500

QUESTÃO 06. Escreva um algoritmo utilizando a estrutura **para...ate** que lê um valor n inteiro e positivo e que calcula a seguinte soma:

$$S = 1 + 1/3 + 1/5 + 1/7 + \dots + 1/n$$

O algoritmo deve escrever cada termo gerado e o valor final de S .

QUESTÃO 07. Faça um algoritmo que mostre os 20 primeiros termos de uma série e calcule o somatório destes termos.

Os primeiros termos da série são: 1, -1/2, 1/3, -1/4, 1/5 etc

QUESTÃO 08. Faça um algoritmo que receba 10 idades, pesos e alturas, calcule e mostre:

- A média das idades das 10 pessoas;
- A quantidade de pessoas com peso superior a 90 quilos e altura inferior a 1.50;
- A porcentagem de pessoas com idade entre 10 e 30 anos entre as pessoas que medem mais de 1.90.
- A maior e menor idade

QUESTÃO 09. Em um cinema, certo dia, cada espectador respondeu a um questionário, que perguntava a sua idade (ID) e a opinião em relação ao filme (OP), seguindo os seguintes critérios ao lado. Construa um programa que, lendo esses dados, calcule e apresente:

- Média de idade das pessoas que responderam a pesquisa
- Total e Porcentagem de cada uma das respostas

OBSERVAÇÃO: Considere que participaram da pesquisa 20 pessoas

QUESTÃO 10. A prefeitura de uma cidade deseja fazer uma pesquisa entre seus habitantes. Faça um programa para coletar dados sobre o salário e número de filhos de cada habitante e após as leituras, escrever:

- Média de salário da população

- Média do número de filhos
- Menor salário dos habitantes
- Maior salário dos habitantes

OBS: Solicitar 5 entrada de dados

LÓGICA DE
PROGRAMAÇÃO

PARTE 4:

Estrutura de

Dados

Homogêneas:

Matrizes

**NESTE CAPÍTULO, SERÁ APRESENTADA A ESTRUTURA DE DADOS ARRAYS
(UMA DIMENSÃO E DUAS DIMENSÕES)**

4.1. ESTRUTURA DE DADOS HOMOGÊNEAS

A utilização de estrutura de dados homogênea é uma técnica de programação que permite trabalhar com o agrupamento de vários dados de um mesmo tipo dentro de uma mesma variável. Essa estrutura recebe diversos nomes como: conjuntos, arranjos, vetores, matrizes, arrays (inglês)

4.2. ESTRUTURA DE DADOS HOMOGÊNEAS DE UMA DIMENSÃO

Esse tipo de estrutura também recebe o nome de matriz unidimensional, vetores ou arrays em inglês. Sua utilização mais comum está vinculada à criação de tabelas. Caracteriza-se por ser definida uma única variável dimensionada com um determinado tamanho.

A dimensão de uma matriz é formada por constantes inteiras e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar variáveis simples.

Para ter uma idéia de como utilizar matrizes em uma determinada situação, considere o seguinte problema: **“Calcular e apresentar a média geral de uma turma de oito alunos. A média a ser obtida deve ser a média geral de cada aluno obtida durante o ano letivo”**. Desta forma será necessário somar todas as médias e dividi-las por oito. Agora basta escrever um programa para fazer o cálculo das 8 médias de cada aluno, que pode ser representada por oito variáveis na qual cada variável armazena a média do aluno.

Exemplo

```
var  
md1,md2,md3,md4,md5,md6,md7,md8:real
```

Veja o programa completo para calcular a média do aluno a partir de 8 notas sem utilizar vetores:

```
algoritmo "Media Final"  
var  
    md1,md2,md3,md4,md5,md6,md7,md8:real  
inicio  
leia (md1,md2,md3,md4,md5,md6,md7,md8)  
soma <-md1 + md2 + md3 + md4 + md5 + md6 +md7 +md8
```

```
media <- soma / 8  
escreva (media)
```

Para receber a média foram utilizadas oito variáveis. Com a técnica de vetores poderia ter sido utilizada apenas uma variável com a capacidade de armazenar oito valores. Dessa forma teríamos um vetor representado pelo seu nome e seu tamanho.

Neste caso poderíamos criar um vetor da seguinte forma:

```
media: vetor [1..8] de real
```

Sendo assim para representarmos uma informação em um vetor, devemos nos referir ao índice, ou seja, o endereço em que o elemento (conteúdo) está armazenado.

Exemplo:

```
media[1]  
media[2]  
media[3]  
...  
media[8]
```

Para fazer referência a uma posição particular ou elemento no array, especificamos o nome do array e o número da posição do elemento no array.

O primeiro elemento em um array é o elemento um¹. Dessa forma, o primeiro elemento do array é referenciado como: exemploArray[1], o segundo elemento é representando como exemploArray[2] e assim sucessivamente.

Os arrays ocupam espaço na memória, dessa forma o programador deverá especificar o tipo de cada elemento e o número de elementos exigidos pelo array. Pode-se reservar memória para vários arrays com uma única declaração

¹ Em linguagens de programação, como por exemplo C# o primeiro elemento de um Array é o elemento 0 (zero) e o último elemento é TAMANHO DO ARRAY – 1.

4.2.1 DEFINIÇÃO E ATRIBUIÇÃO

Como podemos observar, um vetor é definido no mesmo local que uma variável comum, ou seja, dentro do bloco de variáveis “**var**”. Em seguida utilizaremos a instrução “**vetor**” que indica que a utilização de um tipo de dado homogêneo.

Depois definimos a dimensão que será a indicação dos valores inicial e final do tamanho do vetor e por último o tipo de dados que o vetor irá utilizar.

Exemplos:

```
nome: vetor[1..10] de caractere  
salario: vetor[1..15] de real  
idade: vetor[1..60] de inteiro
```

4.2.2 LEITURA E ESCRITA DE DADOS

A leitura de um vetor é processada passo a passo, um elemento por vez. A instrução de leitura é feita da mesma forma que uma variável comum, através do comando “**leia**” seguida da variável mais o índice.

Exemplo:

```
leia (idade[1])  
leia (salario[2])  
leia (nome[4])
```

O processo de escrita de dados de um vetor é semelhante ao processo de leitura de seus elementos. Utilizamos a instrução “**escreva**” seguida da indicação da variável e seu índice.

Exemplo:

```
escreva( idade[3])  
escreva (salario[6])  
escreva (nome[7])
```

Podemos ainda utilizar uma estrutura de repetição para ler e escrever todos os elementos de uma matriz. Veja como ficaria a leitura dos dados

```
para i de 1 ate 8 faca
    leia (media[i])
    soma <-soma + media[i]
fimpara
```

Agora veja como ficaria para escrever todos os dados na tela:

```
para i de 1 ate 8 faca
    escreva(media[i])
fimpara
```

No exemplo anterior a leitura é processada uma por vez. Desta forma o vetor é controlado pelo número do índice que faz com que cada entrada aconteça em uma posição diferente da outra. Ao final da leitura, dizemos que a matriz está preenchida.

Se houver a necessidade de calcular um número maior de valores, basta dimensionar a matriz e mudar o valor final da instrução “**para**” para o valor desejado.

OBS: Tenha cuidado para não confundir o índice com o elemento. O índice é o endereço, enquanto o elemento é o conteúdo armazenado em um determinado endereço.

4.2.3 EXEMPLOS RESOLVIDOS DE MATRIZES UNIDIMENSIONAIS

EXEMPLO 01. Ler 8 elementos de um vetor A. Construir um vetor B de mesma dimensão com os elementos do vetor A multiplicados por 3. Dessa forma o elemento B[1] deve ser impicado pelo elemento A[1] * 3, o elemento B[2] deve ser impicado pelo elemento A[2] * 3. No final você deverá apresentar a matriz B.

```
algoritmo "EXEMPLO 01"
var
a:vetor[1..8] de inteiro
b:vetor[1..8] de inteiro
posicao: inteiro
inicio
```

```

para posicao de 1 ate 8 faca
    escreva("DIGITE O ", posicao, " ° VALOR: ")
    leia(a[posicao])
    b[posicao] <- a[posicao] * 3
fimpara

para posicao de 1 ate 8 faca
    escreval(a[posicao], " * 3 = ", b[posicao])
fimpara

fimalgoritmo

```

EXEMPLO 02. Ler dois vetores A e B com 20 elementos. Construir um vetor C, sendo que cada elemento de C é a subtração de um elemento do vetor A com um elemento correspondente do vetor B, ou seja, a operação $C[1] \leftarrow A[1] - B[1]$.

Ao final apresentar os elementos do vetor C.

```

algoritmo "semnome"
var
a:vetor[1..5] de inteiro
b:vetor[1..5] de inteiro
c:vetor[1..5] de inteiro
j: inteiro
inicio

    para j de 1 ate 5 faca
        escreva("a[,j,] = ")
        leia(a[j])

        escreva("b[,j,] = ")
        leia(b[j])

        c[j] <- a[j] - b[j]
    fimpara
    escreval("O VETOR C FORMADO É: ")

    para j de 1 ate 5 faca
        escreval("c[,j,] = ",c[j])

```

```
fimpara
```

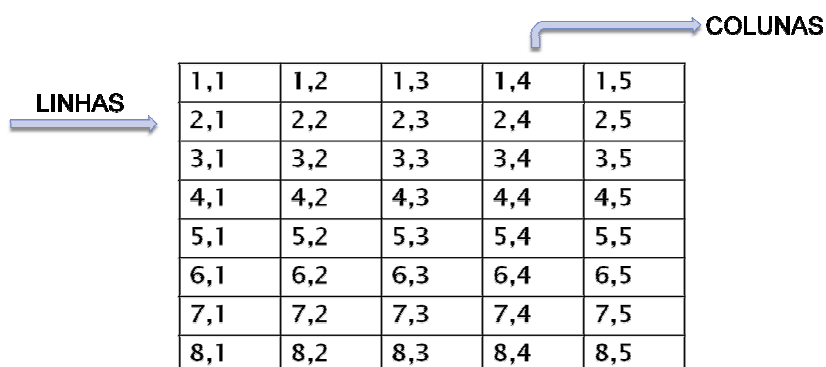
```
fimalgoritmo
```

4.3. ESTRUTURA DE DADOS HOMOGÊNEAS MULTIDIMENSIONAIS

Em determinados problemas são utilizados estruturas de vetores com mais de uma dimensão. O mais comum é a matriz de duas dimensões por se relacionar com a utilização de tabelas. Matrizes com mais de duas dimensões são utilizadas com menos frequência.

Uma matriz de duas dimensões está sempre fazendo menção a linhas e colunas e é representada por seu nome e seu tamanho (dimensão) entre colchetes. Considere o exemplo de uma matriz chamada TABELA, com um tamanho de oito linhas (de 1 a 8) e 5 colunas (de 1 a 5), ou seja, é uma matriz de 8 por 5 (8 x 5). Isso significa que podem ser armazenados em TABELA até 40 elementos.

MATRIZ TABELA:



1,1	1,2	1,3	1,4	1,5
2,1	2,2	2,3	2,4	2,5
3,1	3,2	3,3	3,4	3,5
4,1	4,2	4,3	4,4	4,5
5,1	5,2	5,3	5,4	5,5
6,1	6,2	6,3	6,4	6,5
7,1	7,2	7,3	7,4	7,5
8,1	8,2	8,3	8,4	8,5

4.3.1 DEFINIÇÃO E ATRIBUIÇÃO

Uma matriz de duas dimensões é atribuída pela instrução vetor A sintaxe é:

```
nome: vetor [linha,colunas] de tipo de dado
```

Exemplos:

4.3.2 LEITURA E ESCRITA DE DADOS

```
tabela: vetor[1..5,1..3] de inteiro  
medias: vetor[1..10,1..4] de real
```

Para fazer referência a uma posição particular ou elemento no array, especificamos o nome do array e o número da posição do elemento no array. A leitura de uma matriz de duas dimensões é processada passo a passo, um elemento por vez, sendo utilizada a instrução "leia" seguida da variável mais os seus índices.

Exemplo:

```
leia(tabela[1,3])
```

Na instrução acima, será armazenado na linha 1 e coluna 3 um valor informado.

O processo de escrita é bastante parecido com o processo de leitura de seus elementos. Supondo que após a leitura dos dados, houvesse a necessidade de apresentar os dados lidos. Para apresentar um dado devemos especificar os seus índices (linha e coluna)

Exemplo:

```
escreva(tabela[2,2])
```

No exemplo acima será mostrado na tela o valor do elemento que está na linha 2 e coluna 2.

Um importante aspecto a ser considerado é que na manipulação de uma matriz do tipo vetor é utilizada uma única instrução de laço (para, enquanto ou repita). No caso de matrizes com mais dimensões, deve ser utilizado o número de laços relativos ao tamanho de sua dimensão. Desta forma uma matriz de duas dimensões deve ser controlada com dois laços, de três dimensões por 3 laços e assim por diante.

Considere que em uma turma existam 10 alunos e para cada aluno deverá ser armazenado as médias referentes às quatro unidades. Sendo assim, para resolver esse problema poderíamos utilizar uma matriz de dimensão 10 x 4 (10 linhas representando os 10 alunos e 4 colunas representando cada uma das notas).

Para realizar a leitura dos dados, será necessário realizar um mecanismo para ler as 4 médias de cada aluno. Para isso usaremos duas estruturas de repetição conforme exemplo abaixo:

```
para i de 1 ate 10 faca
    para j de 1 ate 4 faca
        leia(notas[i,j])
    fimpara
fimpara
```

Para a escrita dos dados, também deverá ser utilizado o mesmo mecanismo apresentado nos slides anterior. Veja exemplo:

```
para i de 1 ate 10 faca
    para j de 1 ate 4 faca
        escreva(notas[i,j])
    fimpara
fimpara
```

4.3.3 EXEMPLOS RESOLVIDOS DE MATRIZES MULTIDIMENSIONAIS

EXEMPLO 01. Ler dois vetores A e B, cada um com uma dimensão para 5 elementos. Construir em seguida uma matriz C de duas dimensões, de forma que a primeira coluna da matriz C deve ser formada pelos elementos da matriz A multiplicados por 2 e a segunda coluna deve ser formada pelos elementos da matriz B subtraídos de 5. Apresentar ao final a matriz C.

```
algoritmo "EXEMPLO 01"
var
s:vetor[1..2,1..5] de inteiro
i,j,total_pares: inteiro

inicio

escreval("LENDO OS VALORES DO VETOR...")
para i de 1 ate 5 faca
```

```

        escreva("a[" , i , " , " , j , " ] = ")
        leia(a[i])

fimpara

//preenchendo a primeira e segunda linha da matriz C
para j de 1 ate 5 faca
    c[1,j] <- a[j] * 2
    c[2,j] <- b[j] - 5

fimpara

escreval("ESCREVENDO OS NOVOS VALORES DA MATRIZ ...")

para i de 1 ate 2 faca
    para j de 1 ate 5 faca
        escreval("c[" , i , " , " , j , " ] = " , c[i,j])
    fimpara
fimpara

fimalgoritmo

```

EXEMPLO 02. Escreva um algoritmo que lê uma matriz M(5,5) e calcula as somas:

- da linha 4 de M
- da coluna 2 de M
- da diagonal principal
- da diagonal secundária
- de todos os elementos da matriz M

Escrever essas somas e a matriz.

```

algoritmo "EXEMPLO 02"
var
a:vetor[1..4,1..4] de inteiro
i,j,soma_diagonal_principal,soma_diagonal_secundaria,soma_l
inha_4,soma_coluna_2,soma_elementos: inteiro

inicio

```

```
escreval("LENDO OS VALORES DA MATRIZ...")
para i de 1 ate 4 faca
    para j de 1 ate 4 faca
        escreva("a[" , i , " , " , j , " ] = ")
        leia(a[i,j])
    fimpara
fimpara

//soma_diagonal_principal
para i de 1 ate 4 faca
    soma_diagonal_principal <- soma_diagonal_principal +
a[i,i]
fimpara

j <- 4
//soma_diagonal_secundaria
para i de 1 ate 4 faca
    soma_diagonal_secundaria <- soma_diagonal_secundaria +
a[i,j]
    j <- j -1
fimpara

//soma_linha_4
para i de 1 ate 4 faca
    soma_linha_4 <- soma_linha_4 + a[4,i]
fimpara

//soma_coluna_2
para i de 1 ate 4 faca
    soma_coluna_2 <- soma_coluna_2 + a[i,2]
fimpara

para i de 1 ate 4 faca
    para j de 1 ate 4 faca
        soma_elementos <- soma_elementos + a[i,j]
    fimpara
fimpara

escreval("A SOMA DOS ELEMENTOS DA LINHA 4 É =
",soma_linha_4)
```

```
escreval("A SOMA DOS ELEMENTOS DA COLUNA 2 É =  
",soma_coluna_2)  
escreval("A SOMA DOS ELEMENTOS DA DIAGONAL PRINCIPAL DA  
MATRIZ É = ",soma_diagonal_principal)  
escreval("A SOMA DOS ELEMENTOS DA DIAGONAL SECUNDÁRIA DA  
MATRIZ É = ",soma_diagonal_secundaria)  
escreval("A SOMA DOS ELEMENTOS DA MATRIZ É =  
",soma_elementos)  
fimalgoritmo
```

4.4. EXERCÍCIOS

QUESTÃO 01. Escreva um algoritmo que leia dois vetores de 10 posições e faça a multiplicação dos elementos de mesmo índice, colocando o resultado em um terceiro vetor. Mostre o vetor resultante.

QUESTÃO 02. Faça um algoritmo que leia um vetor S[20] e uma variável A. A seguir, mostre o produto da variável A pelo vetor.

QUESTÃO 03. Escreva um algoritmo que leia e mostre um vetor de 20 números. A seguir, conte quantos valores pares existem no vetor.

QUESTÃO 04. Faça um programa que leia dois vetores: F[20] e G[20]. Calcule e mostre, a seguir, o produto dos valores de F por G.

QUESTÃO 05. Faça um programa que leia um vetor K[30]. Troque a seguir, todos os elementos de ordem ímpar do vetor com os elementos de ordem par imediatamente posteriores.

QUESTÃO 06. Escreva um programa que leia um vetor de 20 posições e mostre- o. Em seguida, troque o primeiro elemento com o último, o segundo com o penúltimo, o terceiro com o antepenúltimo, e assim sucessivamente. Mostre o novo vetor depois da troca.

QUESTÃO 07. Dados dois vetores, um contendo a quantidade e o outro o preço de 20 produtos, elabore um algoritmo que calcule e exiba o faturamento que é igual a quantidade x preço. Calcule e exiba também o faturamento total que é o somatório de todos os faturamentos, a média dos faturamentos e quantos faturamentos estão abaixo da média.

QUESTÃO 08. Ler uma matriz A de duas dimensões com 3 linhas e 4 colunas. Construir um vetor B de uma dimensão que seja formado pela soma de cada linha da matriz A. Ao final apresentar o somatório dos elementos da matriz B.

QUESTÃO 09. 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 M

Escrever essas somas e a matriz.

QUESTÃO 10. Desenvolver um programa de agenda que cadastre o nome, endereço, CEP, bairro e telefone de 10 pessoas. Após a leitura dos dados, o programa deverá permitir que seja consultado os dados da agendar a partir de um nome.

Exemplo: Se o usuário digitar REGILAN, caso REGILAN exista na agenda deverá apresentar o endereço, CEP, bairro e telefone do mesmo

QUESTÃO 11. O Museu de Carros Antigos da Bahia está conduzindo um levantamento dos carros antigos do estado. Para cada carro são fornecidas as seguintes informações: o fabricante (um código inteiro de zero a trinta), o ano do carro (de 1900 a 1950) e as condições do carro (inteiros de 1 a 4 para ruim, regular, boa e excelente, respectivamente). Ao todo o museu de carros antigos da Bahia tem em seu acervo 2000 carros.

Faça um algoritmo que:

- a) Leia um conjunto de dados e depois forneça as seguintes estatísticas:
- b) Os números de carros feitos antes de 1910 cujas condições são classificadas como boas ou excelentes;
- c) Fabricante mais popular, escolhido a partir do número registrado de todos os fabricantes;
- d) Identificar o fabricante cujos carros parecem estar nas melhores condições médias

QUESTÃO 12. Os Obesos Anônimos estão conduzindo um estudo da efetividade de seus programas de redução de peso. Cinquenta membros foram selecionados ao acaso para o estudo.

Nos últimos doze meses, um registro de peso foi feito para cada elemento no início do mês. Pede-se um algoritmo para analisar estas informações, determinando:

- a) A variação média de peso de todos os elementos no período de 12 meses;
- b) O número de elementos cuja variação total de peso excedeu a média;
- c) O número de casos durante o ano no qual um elemento perdeu mais do que a variação média durante um único mês.

LÓGICA DE
PROGRAMAÇÃO

PARTE 5: Sub- rotinas (Procedimentos e Funções)

NESTE CAPÍTULO, SERÃO APRESENTADOS OS CONCEITOS INICIAIS
RELACIONADOS À IMPLEMENTAÇÃO DE SUB-ROTINAS:

-  PROCEDIMENTOS
-  FUNÇÕES

5.1. SUB-ROTINAS

No geral, problemas complexos exigem algoritmos complexos. Mas sempre é possível dividir um problema em partes menores. Cada parte menor tem um algoritmo mais simples, e é esse trecho que se chama sub-rotina. Uma sub-rotina é na verdade um programa, e sendo um programa pode efetuar diversas operações computacionais (entrada, processamento e saída).

As sub-rotinas são utilizadas na divisão de algoritmos complexos, permitindo a modularização de um determinado problema. Ao se trabalhar com essa técnica, pode ser necessário dividir uma sub-rotina em outras tantas quantas forem necessárias, buscando uma solução mais simples e uma parte do problema maior.

O processo de dividir sub-rotinas em outras é denominado Método de Refinamento Sucessivo

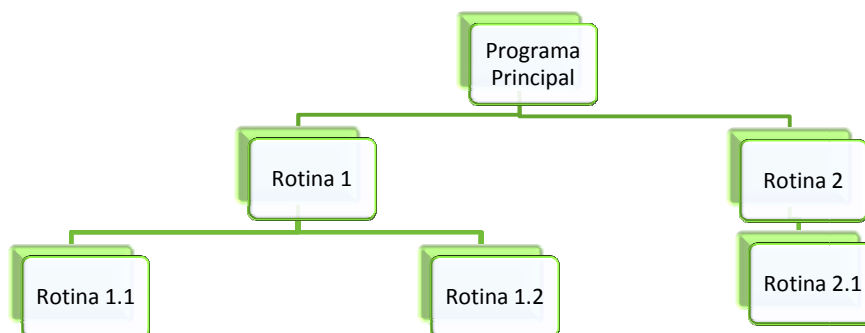


Figura 1 - Método de refinamento sucessivo

Existem dois tipos de rotinas. Entre esses dois tipos existem algumas diferenças, mas o conceito é o mesmo para ambas. Para cada uma das sub-rotinas será usada uma nova instrução que identifique em português estruturado uma sub-rotina, sendo: **procedimento** e **função**.

5.2. PROCEDIMENTOS

Um procedimento é um bloco de programa contendo início e fim e é identificado por um nome. Quando uma sub-rotina é chamada por um programa, ela é executada e ao seu término o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que fez a chamada da sub-rotina.

Veja como é representado um procedimento no Português Estruturado:


```
procedimento <nome do procedimento>  
var  
    //variáveis  
inicio  
    //instruções  
fimprocedimento
```

OBSERVAÇÃO: “O NOME DO PROCEDIMENTO OBEDECE AS MESMAS REGRAS DE NOMENCLATURA DAS VARIÁVEIS.”

Exemplo: Criar um programa calculadora que apresente um menu de seleções no programa principal. Esse menu deve dar ao usuário a possibilidade de escolher uma entre quatro operações aritméticas. Escolhida a opção desejada, deve ser solicitada a entrada de dois números, e processada a operação, deve ser exibido o resultado.

Para o problema acima, podemos dividi-lo em cinco rotinas:

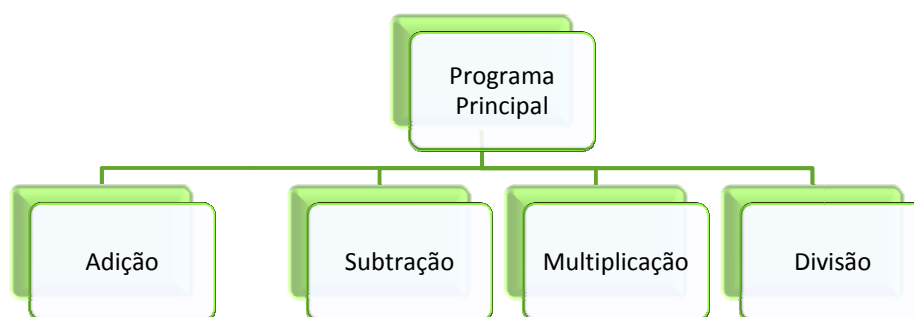


Figura 2 - Exemplo sub-rotina

Tendo idéia da estrutura geral do programa, será escrito em separado cada um dos algoritmos com seus detalhes de operação. Primeiro será escrito as sub-rotinas responsáveis por cada operação e posteriormente o programa principal.

OBS: A ordem da escrita das sub-rotinas não interfere no resultado final do algoritmo

- Rotina Principal: Apresenta um menu com 5 opções
- Rotina adição: Sub-rotina responsável por somar dois números
- Rotina subtração: Sub-rotina responsável por subtrair dois números
- Rotina multiplicação: Sub-rotina responsável por multiplicar dois números

- Rotina divisão: Sub-rotina responsável por dividir dois números

Veja agora a implementação completa do exemplo:

```
procedimento adicao()
var
a,b: real
inicio
escreva ("Informe o primeiro valor: ")
leia (a)
escreva ("Informe o segundo valor: ")
leia (b)

escreval ("O resultado da adição é: ", a + b)

fimprocedimento

procedimento subtracao()
var
a,b: real
inicio
escreva ("Informe o primeiro valor: ")
leia (a)
escreva ("Informe o segundo valor: ")
leia (b)

escreval ("O resultado da subtração é: ", a - b)

fimprocedimento

procedimento multiplicacao()
var
a,b: real
inicio
escreva ("Informe o primeiro valor: ")
leia (a)
escreva ("Informe o segundo valor: ")
leia (b)
```

```
escreval ("O resultado da multiplicação é: ", a * b)

fimprocedimento

procedimento divisao()

var
a,b: real
inicio
escreva ("Informe o primeiro valor: ")
leia (a)
escreva ("Informe o segundo valor: ")
leia (b)

escreval ("O resultado da divisão é: ", a / b)

fimprocedimento

algoritmo "CALCULADORA"

var
opcao: inteiro
inicio

repita

escreval ("MENU DE OPÇÕES: ")
escreval ("Digite 1 para adição")
escreval ("Digite 2 para subtração")
escreval ("Digite 3 para multiplicação")
escreval ("Digite 4 para divisão")
escreval ("Digite 5 para sair")
escreva ("OPÇÃO: ")
leia (opcao)

escolha opcao
caso 1
    adicao()
caso 2
    subtracao()
caso 3
```

```
        multiplicacao()  
caso 4  
        divisao()  
outrocaso  
escreval ("Digite um valor disponivel no menu de  
opções!")  
fimescolha  
  
ate (opcao = 5)  
fimalgoritmo
```

OBSERVAÇÃO IMPORTANTE: Em português estruturado mencionam-se em primeiro lugar as sub-rotinas e por último o programa principal. Sendo assim, durante a escrita do código, primeiro o programador escreve todas as rotinas que serão usadas no programa principal.

No programa apresentado foram utilizadas variáveis dentro do programa principal e dentro das sub-rotinas

- Variável opcao: programa principal
- Variáveis a e b: sub-rotinas

São dois os tipos de variáveis utilizadas em um programa: as variáveis **Globais** e **Locais**. Uma variável é considerada Global quando é declarada no início do algoritmo principal de um programa, podendo ser utilizada por qualquer sub-rotina subordinada ao algoritmo principal. Nesse caso, a variável **opcao** definida na rotina principal é considerada uma variável GLOBAL

Uma variável é considerada local quando é declarada dentro de uma sub-rotina e somente é válida dentro da sub-rotina onde foi declarada. Portanto as variáveis a e b somente estão visíveis nas suas sub-rotinas.

5.3. PARÂMETROS

Os parâmetros são utilizados como um ponto de comunicação entre uma sub-rotina e o programa principal. Através da passagem de parâmetros, podemos passar o valor de uma variável que está no programa principal para a sub-rotina, ou passar o valor de uma variável que está em uma sub-rotina para outra sub-rotina. Os

parâmetros são declarados por meio de variáveis juntamente com a identificação do nome da sub-rotina.

Veja como é feita a declaração de parâmetros no português estruturado:

```
procedimento nome (parametro: tipo de dado)  
var  
//variáveis  
inicio  
//instruções  
fimprocedimento
```

Veja o exemplo de uma sub-rotina com a utilização de parâmetros:

```
procedimento somar (a,b: inteiro)  
var  
resultado: inteiro  
inicio  
resultado <- a + b  
escreva ("O resultado da adição para os valores informados  
é: ",resultado)  
fimprocedimento
```

Com o procedimento pronto, é hora de chamá-la na rotina principal. Veja como fica:

```
algoritmo "uso de rotina"  
var  
valor1,valor2: inteiro  
inicio  
  
leia (valor1)  
leia (valor2)  
somar (valor1,valor2)  
leia (valor1)  
leia (valor2)  
somar(valor1,valor2)  
  
fimalgoritmo
```

5.4. FUNÇÕES

Uma função também é um bloco de programa, como são os procedimentos, contendo início e fim e identificada por um nome, pelo qual também será referenciada em qualquer parte do programa principal.

Uma função é muito parecida com um procedimento, a sua diferença está no fato de uma função retornar um determinado valor, no próprio nome da função. Quando se diz valor, devem ser levados em consideração os valores numéricos, lógicos e caracteres.

Veja como é representado uma função no Português Estruturado

```
funcao <nome da funcao>(parametros) : <tipo de retorno>
var
//variáveis
inicio
//instruções
fimfuncao
```

OBSERVAÇÕES: O nome da função obedece às mesmas regras de nomenclatura das variáveis. O tipo de retorno se refere ao tipo de dado retornado, que pode ser caractere, numérico, lógico

O **nome da função** obedece às mesmas regras de nomenclatura das variáveis. Por outro lado, os <parâmetros> é uma seqüência de: **<parâmetros>: <tipo-de-dado>**

Por sua vez, a seqüência de parâmetros é uma seqüência de nomes de parâmetros (também obedecem à mesma regra de nomenclatura de variáveis) separados por vírgulas.

O valor retornado pela função será do tipo especificado na sua declaração (logo após os dois pontos). Em alguma parte da função (de modo geral, no seu final), este valor deve ser retornado através do comando **retorne**.

De modo análogo ao programa principal, a seção de declaração interna começa com a palavra-chave **var**, e continua com a seguinte sintaxe: **<lista-de-variáveis>: <tipo-de-dado>**

Para exemplificar vamos construir uma função sem parâmetros que soma dois valores. Ela também utiliza uma variável local **aux** para armazenar provisoriamente o resultado deste cálculo, antes de atribuí-lo à variável global **res**:

```
funcao soma: inteiro
var
aux: inteiro
inicio
// n, m e res são variáveis globais
aux <- n + m
retorne aux
fimfuncao
```

No programa principal deve haver os seguintes comandos:

```
n <- 4
m <- -9
res <- soma
escreva(res)
```

Agora iremos realizar a mesma tarefa, só que utilizando passagem de parâmetros para a função:

```
funcao soma (x,y: inteiro): inteiro
inicio
retorne x + y
fimfuncao
```

No programa principal deve haver os seguintes comandos:

```
n <- 4
m <- -9
res <- soma(n,m)
escreva(res)
```

5.5. EXERCÍCIOS

QUESTÃO 01. Criar um procedimento que calcule uma prestação em atraso. Para tanto, utilize a fórmula $PREST = VALOR + (VALOR * (TAXA/100) * TEMPO)$. Apresentar o valor da prestação.

QUESTÃO 02. Elaborar um programa que por meio de uma sub-rotina de procedimento apresente o valor da conversão em real (R\$) de um valor lido em dólar. Deve ser solicitado por meio do programa principal o valor da cotação do dólar e a quantidade de dólar disponível.

QUESTÃO 03. Faça um procedimento que recebe a idade de um nadador por parâmetro e retorna, também por parâmetro, a categoria desse nadador de acordo com a classificação abaixo:

- a) 5 a 7 anos: Infantil A
- b) 8 a 10 anos: Infantil B
- c) 11-13 anos Juvenil A
- d) 14-17 anos Juvenil B
- e) Maiores de 18 anos (inclusive) Adultos

QUESTÃO 04. Desenvolva um programa que crie uma função para calcular e apresentar o valor de uma potência de um número qualquer. Ou seja, ao informar para a função o número e sua potência, deve ser apresentado o seu resultado. Por exemplo, se for mencionado no programa principal a função `Potencia(2,3)`, deve ser apresentado o valor 8.

QUESTÃO 05. Um determinado estabelecimento fará uma venda com descontos nos produtos A e B. Se forem comprados apenas os produtos A ou apenas os produtos B, o desconto será de 10%. Caso sejam comprados os produtos A e B, o desconto será de 15%. O custo da unidade e a quantidade de cada produto deverá ser informado ao programa. Elaborar um programa que por meio de uma função calcule e apresente o valor da despesa do freguês na compra dos produtos.