

ÍNDICE

1 – Abordagem Contextual	4
1.1 – Definições Básicas	4
1.2 – Necessidade do Uso da Lógica	4
1.3 – Aplicabilidade da Lógica no Auxílio do Desenvolvimento de Programas.....	4
1.4 – Diferenciação de Nomenclaturas	5
1.5 – Formas de Representação Gráfica	7
1.6 – Simbologias Básicas	7
1.7 – Simbologias Especiais	8
2 – Introdução à Lógica	10
2.1 – Princípios de Resolução de Problemas	10
2.2 – Particularidades entre Lógicas	12
2.2.1 - Linear.....	13
2.2.2 - Estruturada.....	13
2.2.3 - Modular	15
2.2.4 – Diagrama de Chapin	15
2.2.5 – Português Estruturado.....	16
3 – Tipos de Dados e instruções Primitivas	17
3.1 – Tipos de Informação	17
3.2 – Tipos de Dados	17
3.2.1 - Tipos Inteiros.....	17
3.2.2 - Tipos Reais	17
3.2.3 - Tipos Literais.....	17
3.2.4 - Tipos Lógicos	17
3.3 - O Uso de Variáveis.....	17
3.4 - O Uso de Constantes.....	18
3.5 – Os Operadores Aritméticos.....	18
3.6 - As expressões aritméticas.....	19
3.7 - Instruções básicas	19
3.7.1 - Algumas regras antes de começar	19
3.7.2 - Entrada, processamento e saída	19
3.8 – Exercício de Aprendizagem.....	22
3.9 - Exercícios de Entrega Obrigatória.....	24
3.10 - Exercícios de Extras	26
4 – Estruturas de Controle – A Tomada de Decisões	27
4.1 – Desvio Condicional Simples.....	27
4.2 – Operadores Relacionais	29
4.3 Desvio Condicional Composto	29
4.4 – Desvios Condicionais Encadeados	30
4.5 Operadores Lógicos	33
4.5.1 Operador Lógico: E.....	33
4.5.2 Operador Lógico: OU	34
4.5.3 Operador Lógico: NAO	35
4.6 – Exercício de Aprendizagem.....	36
4.7 - Exercícios de Entrega Obrigatória.....	39
5 – Estruturas de Controle – Laços ou Malhas de Repetição	42
5.1 – Repetição do Tipo: Teste Lógico no Início do Looping	42
5.1.1 - Exercícios de Entrega Obrigatória.....	46
5.2 – Repetição do Tipo: Teste Lógico no Fim do Looping	47
5.2.1 - Exercícios de Entrega Obrigatória.....	49
5.3 – Repetição do Tipo: Variável de Controle	50
5.4 – Considerações entre os Tipos de Estrutura de Looping.....	52
5.5 – Estruturas de Controle Encadeadas.....	52

5.5.1 – Encadeamento de Estruturas Enquanto com Enquanto	53
5.5.2 – Encadeamento de Estrutura Enquanto com Repita	53
5.5.3 – Encadeamento de Estrutura Enquanto com Para	54
5.5.4 – Encadeamento de Estrutura Repita com Repita	55
5.5.5 – Encadeamento de Estrutura Repita com Enquanto	56
5.5.6 – Encadeamento de Estrutura Repita com Para	57
5.5.7 – Encadeamento de Estrutura Para com Para	58
5.5.8 – Encadeamento de Estrutura Para com Enquanto	59
5.5.9 – Encadeamento de Estrutura Para com Repita	60
5.6 – Exercício de Aprendizagem	60
5.6.1 - Exercícios de Entrega Obrigatória	66
6 – Estrutura de Dados Homogêneas I	67
6.1 – Matrizes de uma Dimensão ou Vetores	67
6.2 – Operações Básicas com Matrizes do Tipo Vetor	68
6.2.1 – Atribuição de uma Matriz	68
6.2.2 – Leitura dos Dados de uma Matriz	69
6.2.3 – Escrita dos Dados de uma Matriz	70
6.3 – Exercício de Aprendizagem	71
6.4 - Exercícios de Entrega Obrigatória	74
6.5 - Exercícios de Extras	75
7 – Aplicações Práticas do Uso de Matrizes do Tipo Vetor	75
7.1 – Classificação dos Elementos de uma Matriz	76
7.2 – Métodos de Pesquisa em Matriz	82
7.2.1 – Método de Pesquisa Sequencial	82
7.2.2 – Método de Pesquisa Binária	82
7.3 – Exercício de Aprendizagem	84
7.4 - Exercícios de Entrega Obrigatória	89
7.5 - Exercícios de Extras	90
8 – Estruturas de Dados Homogêneas II	90
8.1 – Matrizes com Mais de uma Dimensão	90
8.2 – Operações Básicas com Matrizes de Duas Dimensões	91
8.2.1 – Atribuição de uma Matriz	91
8.2.2 – Leitura dos Dados de uma Matriz	91
8.2.3 – Escrita de Dados de uma Matriz	93
8.3 – Exercício de Aprendizagem	93
8.4 - Exercícios de Entrega Obrigatória	100
8.5 - Exercícios de Extras	101
9 – Estruturas de Dados Heterogêneas	101
9.1 - Estrutura de um Registro	101
9.1.1 - Atribuição de Registros	102
9.1.2 - Leitura e Escrita de Registros	103
9.2 - Estrutura de um Registro de Vetor	103
9.2.1 - Atribuição de Registros de Vetores (Matrizes)	104
9.2.2 - Leitura e Escrita de Registros Contendo Vetores (Matrizes)	104
9.3 - Estrutura de um Vetor de Registros	104
9.3.1 - Atribuição de Vetor de Registros	105
9.3.2 - Leitura e Escrita de um Vetor de Registros	105
9.4 - Exercício de Aprendizagem	105
Algoritmo	106
Algoritmo	107
9.5 - Exercícios de Entrega Obrigatória	108
9.6 - Exercícios de Extras	109
10 – Utilização de Sub-rotinas	109
10.1 – As Sub-rotinas	109

10.2 – Método Top-Down.....	110
11 – Procedimentos (Sub-Rotinas)	111
11.1 – Exercício de Aprendizagem	111
11.2 – Estrutura de Controle com Múltipla Escolha.....	115
11.3 – Variáveis Globais e Locais	118
11.3.1 – Escopo de Variáveis.....	119
11.3.2 – Refinamento Sucessivo	120
11.4 - Exercícios de Entrega Obrigatória.....	121
11.5 - Exercícios de Extras	122
12 – Utilização de Parâmetros	123
12.1 - Parâmetros Formais e Reais	123
12.2 - Passagem de Parâmetros.....	123
12.2.1 – Por Valor.....	123
12.2.2 – Por Referência.....	124
12.3 – Exercício de Aprendizagem.....	125
12.4 - Exercícios de Entrega Obrigatória.....	129
12.5 - Exercícios de Extras	130
13 – Funções (Sub-Rotinas).....	130
13.1 – Aplicação de Funções em um Programa.....	131
13.2 – Considerações a Respeito de Funções	131
13.3 – Exercício de Aprendizagem.....	132
13.4 - Exercícios de Entrega Obrigatória.....	136
13.5 - Exercícios de Extras	137

1 – Abordagem Contextual

1.1 – Definições Básicas

Muitas pessoas gostam de falar ou julgar que possuem e sabem usar o raciocínio lógico, porém, quando questionadas direta ou indiretamente, perdem esta linha de raciocínio, pois este depende de inúmeros fatores para completá-lo, tais como: calma, conhecimento, vivência, versatilidade, experiência, criatividade, ponderação, responsabilidade, entre outros.

Para usar a lógica, é necessário ter domínio sobre o pensamento, bem como saber pensar, ou seja, possuir a “Arte de Pensar”. Alguns definem o raciocínio lógico como um conjunto de estudos que visa determinar os processos intelectuais que são as condições gerais do conhecimento verdadeiro. Isso é válido para a tradição filosófica clássica aristotélico-tomista. Outros preferem dizer que é a sequência coerente, regular e necessária de acontecimentos, de coisas ou fatos, ou até mesmo, que é a maneira do raciocínio particular que cabe a um indivíduo ou a um grupo. Estas são algumas definições encontradas no dicionário Aurélio, mas existem outras que expressam o verdadeiro raciocínio lógico dos profissionais da área de Tecnologia da Informação, tais como: um esquema sistemático que define as interações de sinais no equipamento automático do processamento de dados, ou o computador científico com o critério e princípios formais de raciocínio e pensamento.

Para concluir todas estas definições, pode-se dizer que lógica é a ciência que estuda as leis e critérios de validade que regem o pensamento e a demonstração, ou seja, ciência dos princípios formais do raciocínio.

1.2 – Necessidade do Uso da Lógica

Usar a lógica é um fator a ser considerado por todos, principalmente pelos profissionais da área da Tecnologia de Informação (programadores, analistas de sistemas e suporte), pois seu dia-a-dia dentro das organizações é solucionar problemas e atingir os objetivos apresentados por seus usuários com eficiência e eficácia, utilizando recursos computacionais e automatizados mecanicamente. Saber lidar com problemas de ordem administrativa, de controle, de planejamento e de estratégia requer atenção e boa performance de conhecimento de nosso raciocínio. Porém, é necessário considerar que ninguém ensina ninguém a pensar, pois todas as pessoas normais possuem esse “dom”. O objetivo deste trabalho é mostrar como desenvolver e aperfeiçoar melhor essa técnica, lembrando que para isso você deve ser persistente e praticá-la constantemente, chegando quase à exaustão sempre que julgar necessário.

1.3 – Aplicabilidade da Lógica no Desenvolvimento de Programas

Muitos programadores (principalmente os mais antigos profissionais desta área) preferem preparar um programa iniciando com um diagrama de blocos para demonstrar sua linha de raciocínio lógico. Esse diagrama, também denominado por alguns de fluxograma, estabelece a sequência de operações a se efetuar em um programa.

Essa técnica permite uma posterior codificação em qualquer linguagem de programação de computadores, pois na elaboração do diagrama de blocos não se atinge um detalhamento de instruções ou comando específicos, os quais caracterizam uma linguagem.

A técnica mais importante no projeto da lógica de programas é chamada programação estruturada, a qual consiste em uma metodologia de projeto, objetivando:

- agilizar a codificação da escrita de programas;
- facilitar a depuração da sua leitura;
- permitir a verificação de possíveis falhas apresentadas pelos programas;
- facilitar as alterações e atualizações dos programas.

E deve ser composta por quatro passos fundamentais:

- Escrever as instruções em seqüências ligadas entre si apenas por estruturas seqüenciais, repetitivas ou de selecionamento.
- Escrever instruções em grupos pequenos e combiná-las.
- Distribuir módulos do programa entre os diferentes programadores que trabalharão sob a supervisão de um programador sênior, ou chefe de programação.
- Revisar o trabalho executado em reuniões regulares e previamente programadas, em que compareçam programadores de um mesmo nível.

1.4 – Diferenciação de Nomenclaturas

É comum ouvir os profissionais da área de Tecnologia da Informação denominarem os símbolos que representam as linhas do raciocínio lógico de fluxograma, diagramas de blocos ou algoritmos, como se tivessem o mesmo significado. Cabe ressaltar que estas palavras têm significados diferenciados, as quais executam processos (tarefas) opostos e/ou diferentes. Mencionamos em seguida os seus verdadeiros significados corretos em suas aplicações, mas lembre-se que para muitos elas terão a mesma finalidade.

Fluxograma é uma ferramenta usada e desenvolvida pelos profissionais de análise de sistemas, bem como, por alguns profissionais de Organização, Sistemas e Métodos. Tem como finalidade descrever o fluxo, seja manual ou mecânico, especificando os suportes usados para os dados e as informações. Usa símbolos convencionais permitindo poucas variações. Representado por alguns desenhos geométricos básicos, os quais indicarão os símbolos de entrada de dados, do processamento de dados e da saída de dados, acompanhados dos procedimentos requeridos pelo analista de sistemas e a serem realizados pelo programador por meio do desenvolvimento do raciocínio lógico, o qual deverá solucionar o problema do programa a ser processado pelo computador.

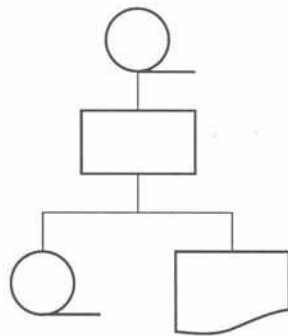


Figura 1.1 - Exemplo de um fluxograma.

Diagrama de Bloco (também poderia ser denominado diagrama de fluxo) é uma ferramenta usada e desenvolvida pelo profissional que está envolvido diretamente com a programação, tendo como objetivo descrever o método e a seqüência do processo dos planos num computador. Pode ser desenvolvido em qualquer nível de detalhe que seja necessário. Quando se desenvolve um diagrama para o programa principal, por exemplo, seu nível de detalhamento pode chegar até as instruções. Essa ferramenta usa diversos símbolos geométricos, os quais estabelecerão as seqüências de operações a serem efetuadas em um processamento computacional. Após a elaboração do diagrama de blocos, será realizada a codificação do programa.

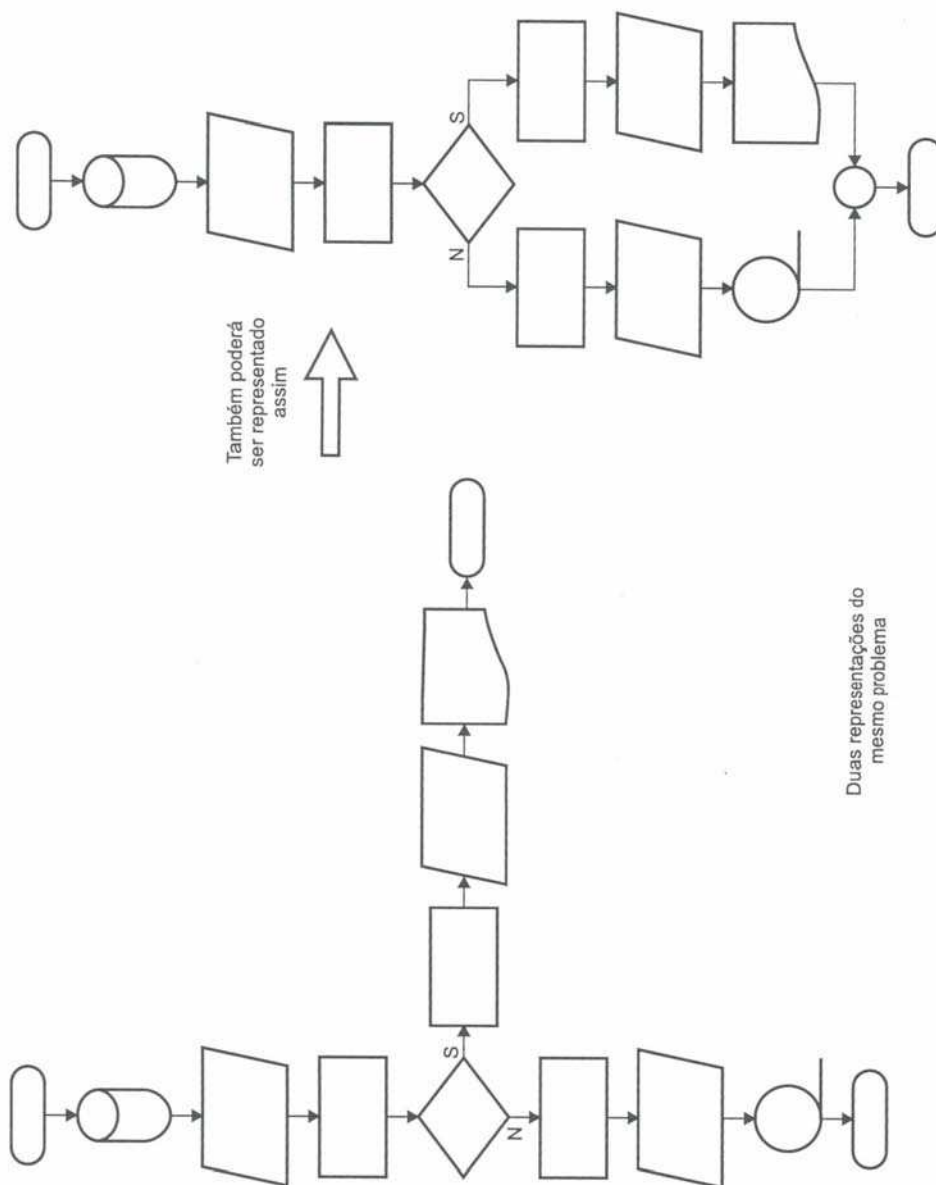


Figura 1.2 - Diagrama de Blocos.

Algoritmo é um processo de cálculo matemático ou de resolução de um grupo de problemas semelhantes. Pode-se dizer também que são regras formais para obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas. Em processamento de dados, é muito comum relacionar a palavra algoritmo com diagramação de bloco, já que muitas fórmulas estão dentro das simbologias de processos para a resolução de um determinado problema, seja na área contábil, seja na área financeira, bem como em qualquer situação que exija um resultado final “correto” ou “coerente”.

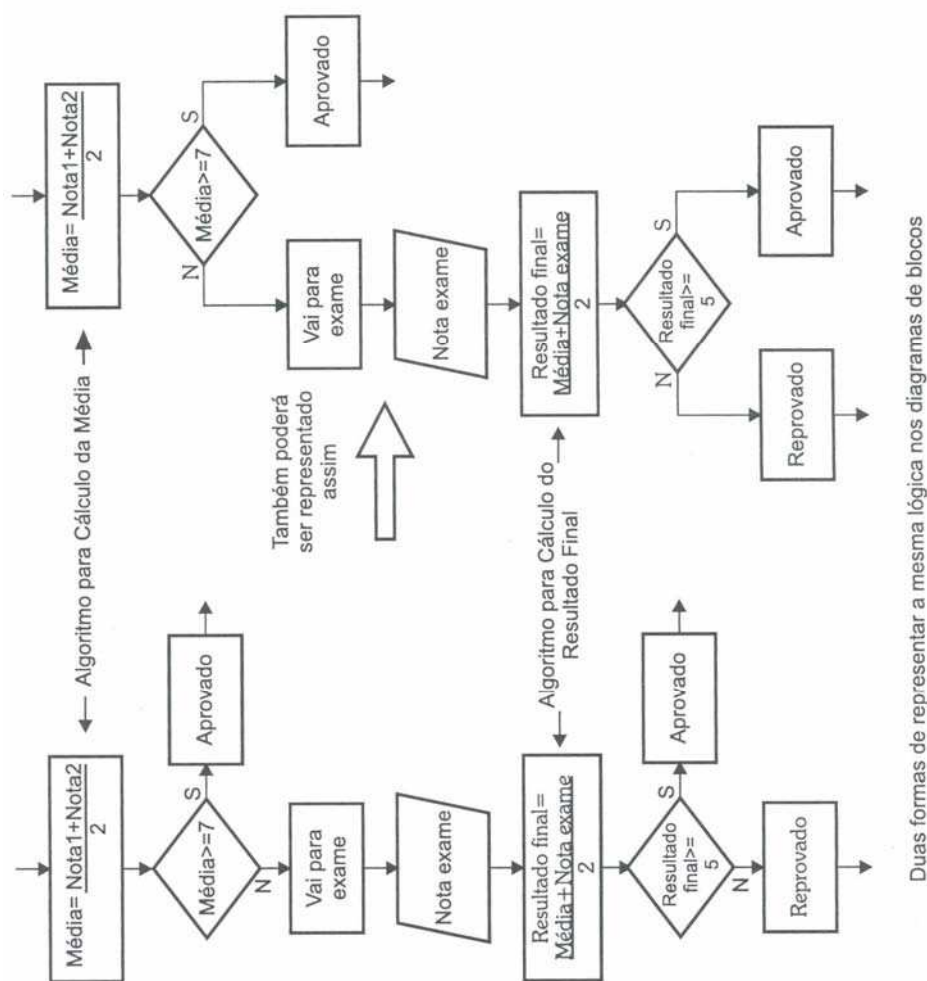


Figura 1.3 - Exemplo de um algoritmo.

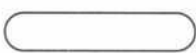






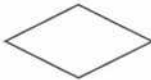


1.5 – Formas de Representação Gráfica

As formas de representação gráfica são nada mais do que uma maneira mais simples e concisa de representar os dados sobre uma superfície plana, por meio de diferentes formas, de modo a facilitar a visualização completa e imediata de dados ou fenômenos tabulados.

Sabe-se que existe uma grande variedade de símbolos usados nas mais diversas áreas administrativas, bem como dentro das áreas técnicas da Tecnologia da Informação, tais como: programação, teleprocessamento, análise de sistemas, etc.

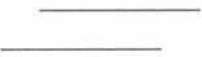

1.6 – Simbologias Básicas

A seguir alguns dos símbolos mais conhecidos e utilizados ao longo dos anos pelos profissionais de processamento de dados.

	Terminal - símbolo utilizado como ponto para indicar o início e/ou fim do fluxo de um programa
	Seta de fluxo de dados - permite indicar o sentido do fluxo de dados. Serve exclusivamente para conectar os símbolos ou blocos existentes.
	Processamento - símbolo ou bloco que se utiliza para indicar cálculos (algoritmos) a efetuar, atribuições de valores ou qualquer manipulação de dados que tenha um bloco específico para sua descrição.
	Entrada de dados ou operação manual - utilizado para ler os dados necessários ao programa fora de linha sem intervenção de dispositivos mecânicos.
	Entrada e saída de dados - símbolo em função de um dispositivo qualquer de entrada ou saída de dados, como fornecedor de informações para processamento, gravação e outros.
	Saída de dados em vídeo - utiliza-se este símbolo quando se quer mostrar dados na tela do vídeo.
	Saída de dados em impressora - é utilizado quando se deseja que os dados sejam impressos.
	Decisão - indica a decisão que deve ser tomada, indicando a possibilidade de desvios para diversos outros pontos do fluxo, dependendo do resultado de comparação e de acordo com situações variáveis.
	Conector - utilizado quando é preciso particionar o diagrama. Quando ocorrer mais de uma partição, é colocada uma letra ou número dentro do símbolo de conexão para identificar os pares de ligação.
	Conector - específico para indicar conexão do fluxo em outra página.

1.7 – Simbologias Especiais

Abaixo seguem outros símbolos normalmente usados em outras áreas, bem como também pela área de desenvolvimento de software.

	Operação manual - fora de linha sem intervenção de dispositivos eletromecânicos.
	Modificação de programas - indica a existência de uma instrução ou de um grupo de instruções que irão modificar o programa, o qual poderá estar descrito ou em análise.
	Cartão perfurado - todas as variedades apresentadas. Essa massa de cartões poderá ser usada como documentos escritos anteriormente.
	Preparação - refere-se a um determinado grupo de operações não incluídas na diagramação, bem como, na elaboração de uma chave que modificará a execução de um determinado programa.
	Teclado - serão as informações recebidas ou fornecidas de ou por um computador.
	Display - para informações exibidas por dispositivos visuais, vídeo ou monitor.
	Checagem de operação auxiliar - é uma operação de máquina suplementar à função principal de processamento.
	Disco magnético - memória de massa para armazenamento de dados (winchester).
	Tambor magnético - memória de massa para armazenamento de dados.
	Fita magnética - memória de massa para armazenamento de dados (streamer).
	Disquete - memória de armazenamento de dados.
	Linha de comunicação - permite a transmissão automática de informação em locais diferentes, por meio de linhas de comunicação.

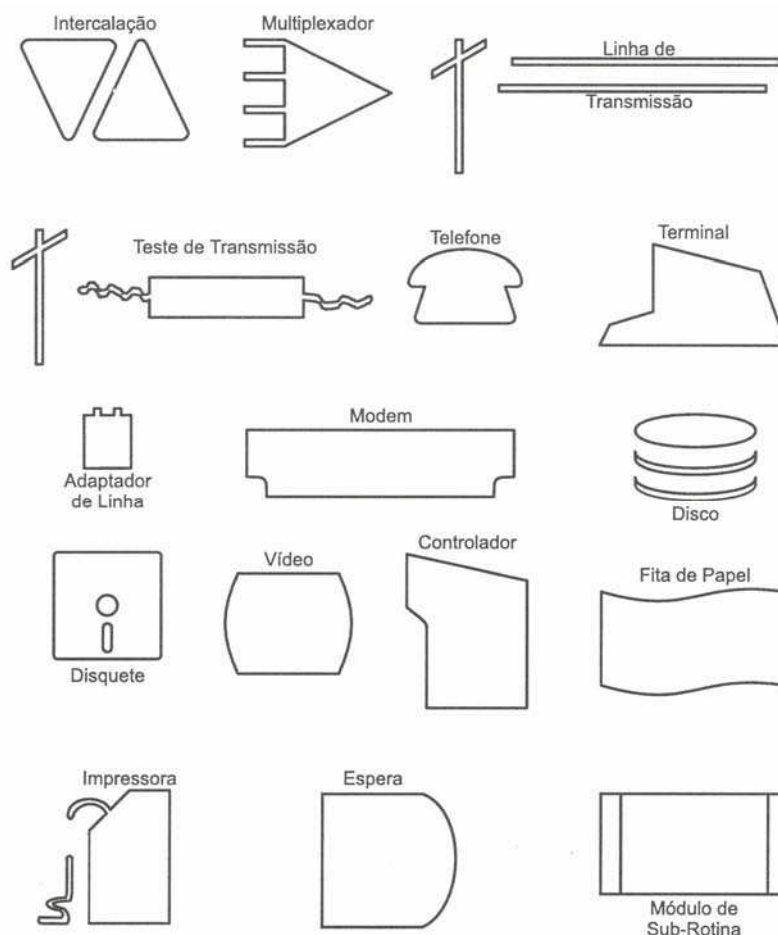


Figura 1.4 - Outros símbolos especiais.

2 – Introdução à Lógica

2.1 – Princípios de Resolução de Problemas

Primeiramente, é importante entender a palavra “problema”. Pode-se dizer que problema é uma proposta duvidosa, que pode ter numerosas soluções, ou questão não solvida e que é objeto de discussão, segundo definição encontrada no dicionário Aurélio.

Do ponto de vista deste trabalho, problema é uma questão que foge a uma determinada regra, ou melhor, é o desvio de um percurso, o qual impede de atingir um determinado objetivo com eficiência e eficácia.

Diferente das diagramações clássicas, que não fornecem grandes subsídios para análise, os diagramas de blocos são realmente o melhor instrumento para avaliação do problema do fluxo de informações de um dado sistema. Por esse motivo deve-se resolver um problema de lógica (principalmente se for da área de processamento eletrônico de dados) usando um procedimento de desenvolvimento.

Para desenvolver um diagrama correto, deve-se considerar como procedimentos prioritários os itens seguintes:

- Os diagramas devem ser feitos e quebrados em vários níveis. Os primeiros devem conter apenas as idéias gerais, deixando para as etapas posteriores os detalhes necessários;
- Para o desenvolvimento correto de um fluxograma, sempre que possível, deve ser desenvolvido de cima para baixo e da esquerda para direita;
- É incorreto e “proibido” ocorrer cruzamento das linhas de fluxo de dados.

Tomo como exemplo uma escola qualquer, cujo cálculo da média é realizado com as quatro notas bimestrais que determinam a aprovação ou reprovação dos seus alunos. Considere ainda que o valor da média deve ser maior ou igual a 7 para que haja aprovação. A primeira etapa deste problema via diagrama de blocos está na figura a seguir.

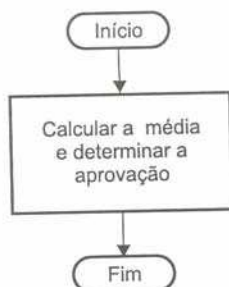


Figura 2.1 - Diagrama de bloco para o cálculo da média escolar.

A segunda etapa apresenta um detalhamento no que se refere à entrada e saída, ou seja, deve-se entrar com as quatro notas bimestrais para se obter, como resultado, o cálculo da média e assim definir a aprovação ou reprovação do aluno. A figura a seguir apresenta o diagrama de blocos com mais detalhes.

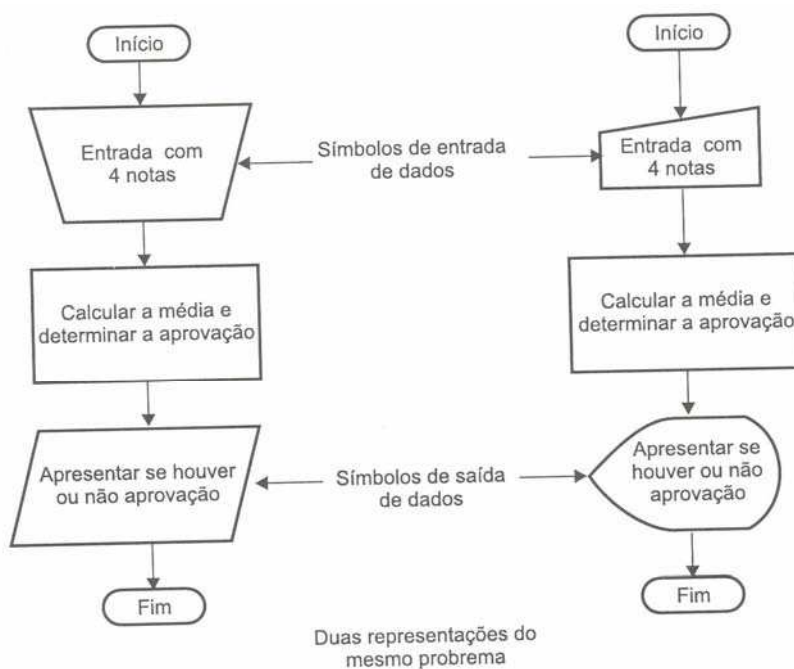


Figura 2.2 - Diagramas apresentando a entrada das notas e a saída se houve aprovação.

A terceira etapa consiste em trabalhar o termo “determinar a aprovação”. Para ser possível determinar algo, é necessário estabelecer uma condição. Assim sendo, uma condição envolve uma decisão a ser tomada segundo um determinado resultado. No caso, a média. Desta forma, a condição de aprovação: média maior ou igual a 7, deve ser considerada no algoritmo.

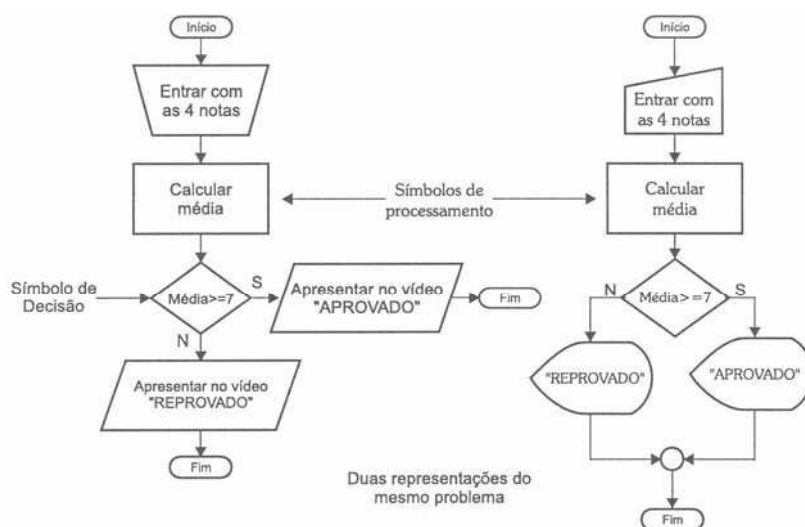


Figura 2.3 - Uso de uma condição em um diagrama de blocos.

Muitas vezes é preferível construir o diagrama de blocos trabalhando com variáveis. Este assunto será estudado mais adiante. A figura a seguir apresenta um exemplo de um algoritmo utilizando variáveis.

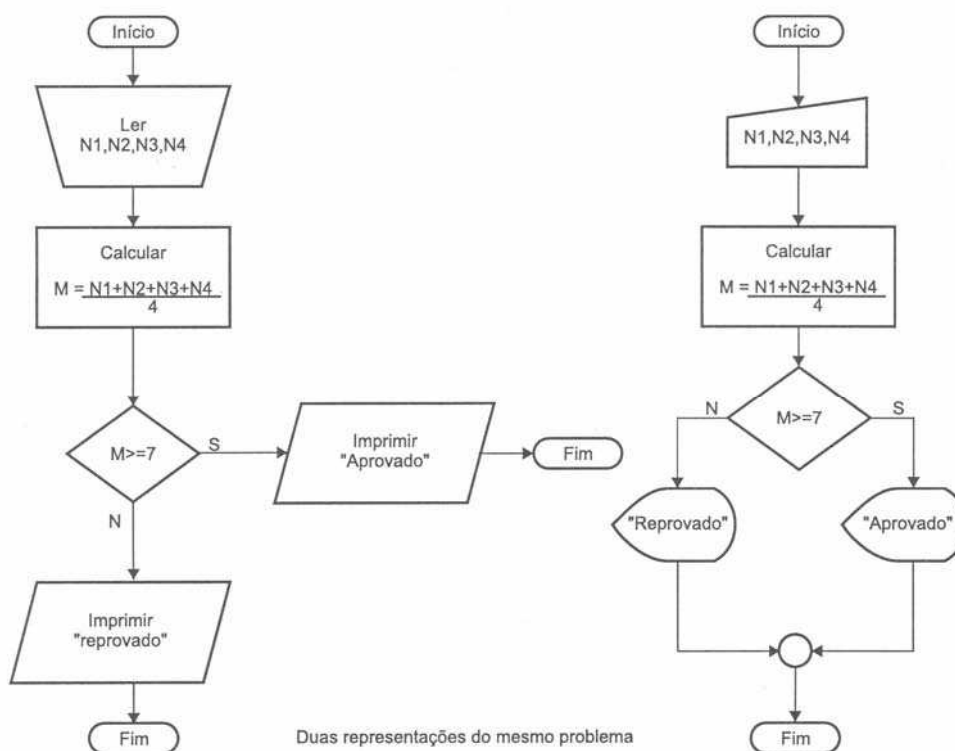


Figura 2.4 - Exemplo da utilização de variáveis.

2.2 – Particularidades entre Lógicas

As representações gráficas de um diagrama de blocos podem ser feitas de várias maneiras e possuem estruturas diferenciadas, porém isto não impede que a maneira de solucionar o problema seja eficiente. Segundo Verzello nos diz em uma de suas obras, assim como um arquiteto desenha e escreve especificações para descrever como uma tarefa (por exemplo, construção de um edifício) deverá ser efetuada e o engenheiro do projeto desenvolve um esquema detalhado das atividades de construção, um especialista em informação desenvolve um plano, que será comunicado a outros, de como o problema de processamento de dados deve ser resolvido.

Para auxiliarmos na sua compreensão, mostraremos como estes conceitos de estruturas, bem como as particularidades de conexões e dos procedimentos entre o método lógico, encaixam-se ou não na resolução dos problemas de processamento de dados.

2.2.1 - Linear

A técnica lógica linear é conhecida como um modelo tradicional de desenvolvimento e resolução de um problema. Não está ligada a regras de hierarquia ou de estruturas de linguagens específicas de programação de computadores. Devemos entender que este tipo de procedimento está voltado à técnica matemática, a qual permite determinar a atribuição de recursos limitados, utilizando uma coleção de elementos organizados ou ordenados por uma só propriedade, de tal forma que cada um deles seja executado passo a passo de cima para baixo, em que tenha um só “predecessor” e um só “sucessor”. A próxima figura apresenta um exemplo deste tipo de lógica.

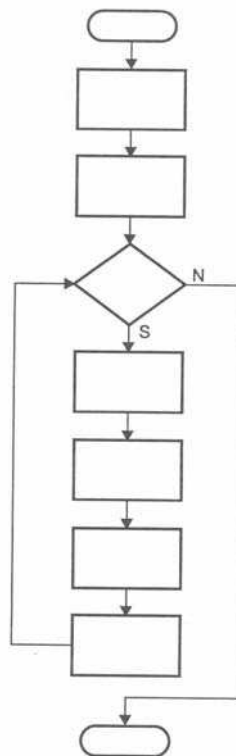


Figura 2.5 - Exemplo de lógica linear.

2.2.2 - Estruturada

A técnica da lógica estruturada é a mais usada pelos profissionais de processamento eletrônico de dados. Possui características e padrões particulares, os quais diferem dos modelos das linguagens elaboradas por seus fabricantes. Tem como pontos fortes para elaboração futura de um programa, produzi-lo com alta qualidade e baixo custo.

A sequência, a seleção e a iteração são as três estruturas básicas para a construção do diagrama de blocos. A figura seguinte apresenta um exemplo do tipo de lógica estruturada.

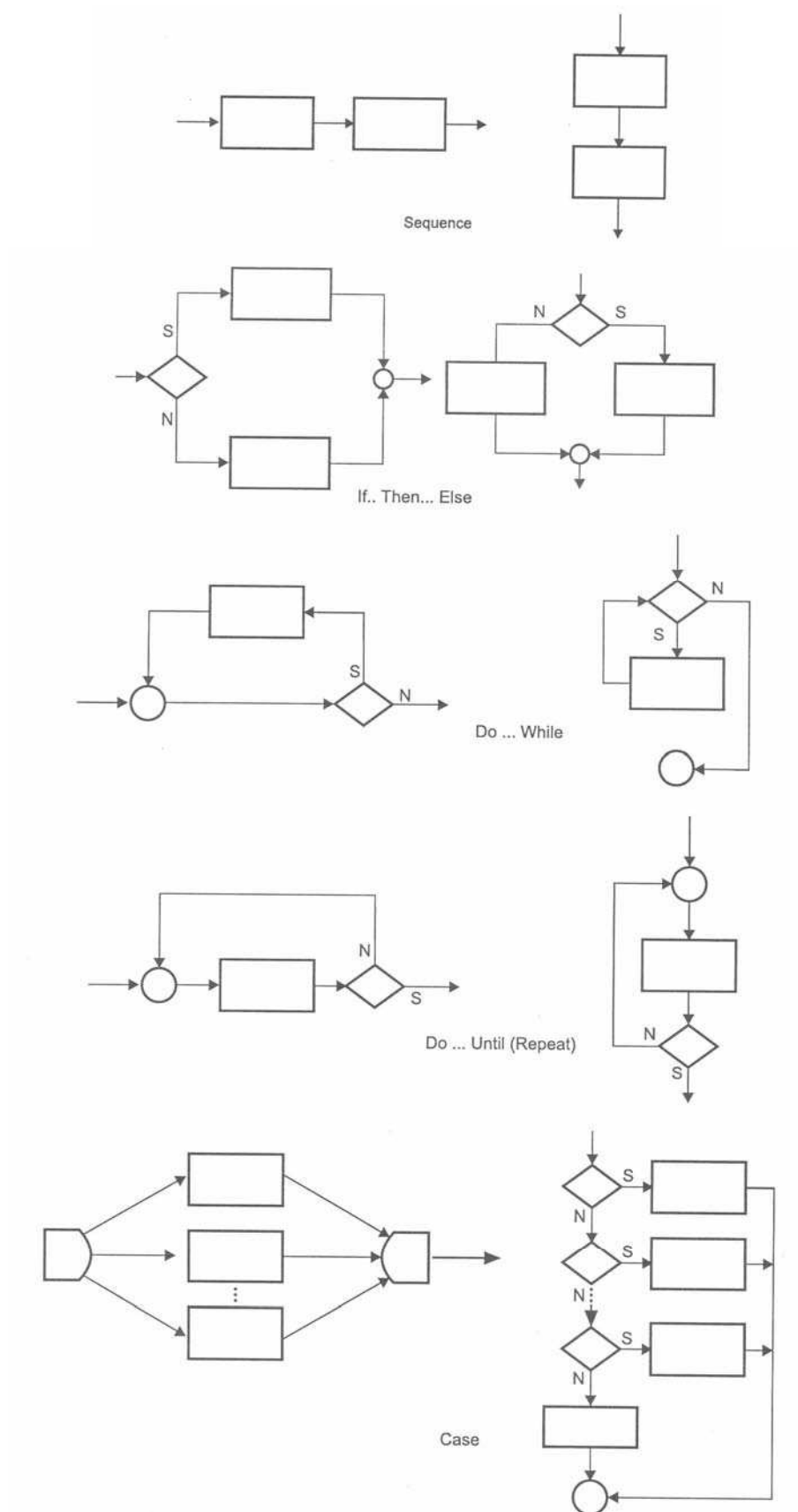


Figura 2.6 - Exemplo de lógica estruturada (continuação da página anterior).

2.2.3 - Modular

A técnica da lógica modular deve ser elaborada como uma estrutura de partes independentes, denominada de módulos, cujo procedimento é controlado por um conjunto de regras. Segundo James Martin, suas metas são as seguintes:

- Decompor um diagrama em partes independentes;
- Dividir um problema complexo em problemas menores e mais simples;
- Verificar a correção de um módulo de blocos, independentemente de sua utilização como uma unidade em um processo maior.

A modularização deve ser desenvolvida, se possível, em diferentes níveis. Poderá ser utilizada para separar um problema em sistemas, um sistema em programas e um programa em módulos. A figura seguinte apresenta um exemplo do tipo de lógica modular.

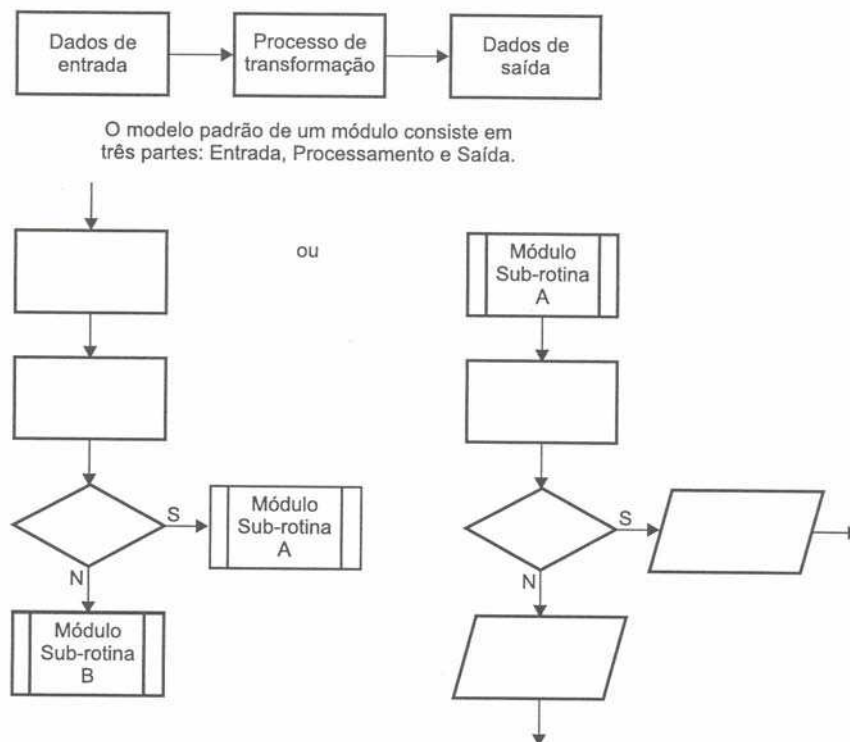


Figura 2.7 - Exemplo de lógica modular.

2.2.4 – Diagrama de Chapin

O diagrama foi desenvolvido por Nassi e Shneiderman e ampliado por Ned Chapin, os quais resolveram substituir o diagrama de blocos tradicional por um diagrama de quadros que permite apresentar uma visão hierárquica e estruturada da lógica do problema. A grande vantagem de usar este tipo de diagrama é a representação das estruturas que tem um ponto de entrada e um ponto de saída e são compostos pelas estruturas básicas de controle de sequência, seleção e repetição. Enquanto é difícil mostrar o embutimento e a recursividade com o diagrama de blocos tradicional, torna-se mais simples mostrá-los com o *diagrama de Chapin*, bem como codificá-los futuramente na conversão de pseudocódigos (português estruturado) ou qualquer linguagem de programação estruturada. A figura seguinte apresenta um exemplo do tipo de diagrama de Chapin para o algoritmo do cálculo da média escolar. Este tipo de diagrama também é denominado *diagrama de Shneiderman* ou diagrama N-S.

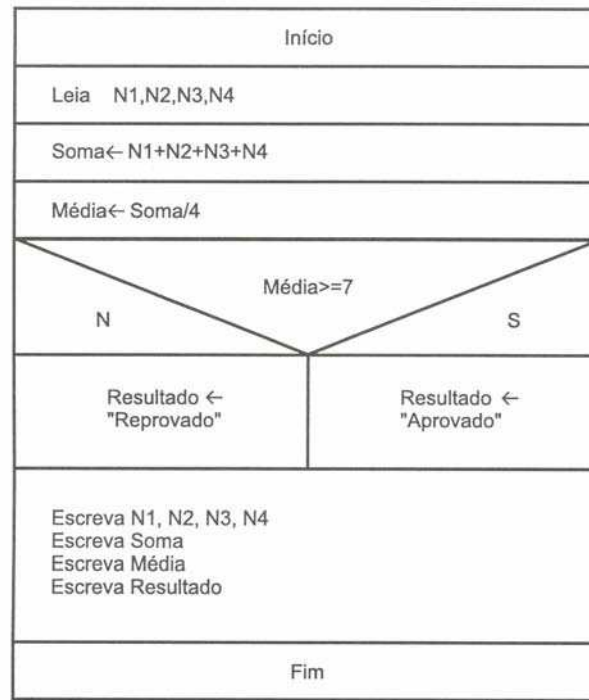


Figura 2.8 - Exemplo de diagrama de Chapin.

2.2.5 – Português Estruturado

Como foi visto até agora, o diagrama de blocos é a primeira forma de notação gráfica, mas existe outra, que é uma técnica narrativa denominada *pseudocódigo*, também conhecida com *português estruturado* ou chamada por alguns de *portugol*.

Esta técnica de algoritmização é baseada em uma PDL – *Program Design Language* (Linguagem de Projeto de Programação). Neste trabalho, estamos apresentando-a em português. A forma original de escrita é conhecida como *inglês estruturado*, muito parecida com a notação da linguagem PASCAL. A PDL (neste caso, o *português estruturado*) é usado como referência genérica para uma linguagem de projeto de programação, tendo como finalidade mostrar uma notação para elaboração de algoritmos, os quais serão utilizados na definição, criação e desenvolvimento em uma linguagem computacional (Cobol, Fortran, C, Pascal, Delphi, Visual-Basic, etc.). A seguir vemos um exemplo deste tipo de algoritmo.

```

algoritmo "media"
// Função: Calculo da media de um aluno exibindo
//          se foi aprovado ou reprovado
// Autor: Manzano
// Data: 9/7/2005
// Seção de Declarações

var
    resultado: caractere
    n1, n2, n3, n4: real
    soma, media: real

inicio
    leia(n1, n2, n3, n4)
    soma ← n1 + n2 + n3 + n4
    media ← soma / 4
    se (media >= 7) entao
        resultado ← "Aprovado"
    senao
        resultado ← "Reprovado"
    fimse
    escreva("Nota 1: ", n1)
    escreva("Nota 2: ", n2)
    escreva("Nota 3: ", n3)
    escreva("Nota 4: ", n4)
  
```



```
escreva("Soma: ", soma)
escreva("Media: ", media)
escreva("Resultado: ", resultado)
finalgoritmo
```

3 – Tipos de Dados e instruções Primitivas

3.1 – Tipos de Informação

Antes de iniciar o estudo de programação, é necessário considerar que um computador nada mais é do que uma ferramenta utilizada para solucionar problemas que envolvam a manipulação de informações, sendo que essas informações classificam-se grosso modo em dois tipos básicos: *dados* e *instruções*.

3.2 – Tipos de Dados

Os **dados** são representados pelas informações a serem tratadas (processadas) por um computador. Essas informações estão caracterizadas por três tipos de dados, a saber: dados numéricos (inteiros e reais), dados caracteres e dados lógicos.

3.2.1 - Tipos Inteiros

São caracterizados como tipos inteiros os dados numéricos positivos ou negativos, excluindo-se destes qualquer número fracionário. Como exemplo deste tipo de dado têm-se os valores: 35, 0, -56, entre outros.

3.2.2 - Tipos Reais

São caracterizados como tipos reais os dados numéricos positivos, negativos e números fracionários. Como exemplo deste tipo de dado têm-se os valores: 35, 0, -56, 1.2, -45.897, entre outros. (Observe que quando estamos tratando números em programas devemos utilizar a notação inglesa e substituir a vírgula pelo ponto flutuante).

3.2.3 - Tipos Literais

São caracterizados como tipos literais às seqüências contendo letras, números e símbolos especiais. Uma seqüência de caracteres deve ser indicada entre aspas (""). Este tipo de dado é também conhecido como: alfanumérico, string, caracter ou cadeia. Como exemplo deste tipo de dado, tem-se os valores: "media", "Rua Figueiredo Oliveira, 52 Apto 34", "Fone: 5560-6637", "0", "734-48", "Fernanda", entre outros.

3.2.4 - Tipos Lógicos

São caracterizados como tipos lógicos os dados com valores **verdadeiro** e **falso**, sendo que este tipo de dado poderá representar apenas um dos valores. Ele é chamado por alguns de **tipo booleano**, devido à contribuição do filósofo e matemático inglês George Boole na área da lógica matemática.

3.3 - O Uso de Variáveis

Tem-se como definição de variável tudo aquilo que é sujeito a variações, que é incerto, instável ou inconstante. E quando se fala de computadores, temos que ter em mente que o volume de informações a serem tratadas é grande e diversificado. Desta forma, os dados a serem processados serão bastante variáveis.

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

Para utilizar o conceito de variável, imagine que a memória de um computador é um grande arquivo com várias gavetas, sendo que cada gaveta pode apenas armazenar um único valor (seja ele numérico, lógico ou caractere). Se for um grande arquivo com várias gavetas, você há de concordar que é necessário identificar com um nome a gaveta que se pretende utilizar. Desta forma o valor armazenado pode ser utilizado a qualquer momento.

O nome de uma variável é utilizado para sua identificação e posterior uso dentro de um programa. Sendo assim, é necessário estabelecer algumas regras de utilização das variáveis:

- Nomes de uma variável poderão ser atribuídos com um ou mais caracteres;
- O primeiro caractere do nome de uma variável não poderá ser, em hipótese alguma, um número; sempre deverá ser uma letra;
- O nome de uma variável não poderá possuir espaços em branco;
- O nome de uma variável não poderá ser uma palavra reservada (uma instrução ou comando);
- Não poderão ser utilizados outros caracteres a não ser letras, números e sublinhado.

São nomes válidos de variáveis: NOMEDOUSUARIO, telefone, x, z, delta_25, z1, entre outros. São nomes **inválidos** de variáveis: NOME DO USUARIO, 25_delta, telefone#, escreva (é uma palavra reservada, no caso do nosso *Portugol*).

Devemos ainda considerar que dentro de um programa uma variável pode exercer dois papéis. Um de ação, quando é modificada ao longo de um programa para apresentar um determinado resultado, e o segundo de controle, a qual poderá ser “vigiada” e controlada durante a execução de um programa.

3.4 - O Uso de Constantes

Tem-se como definição de constante tudo aquilo que é fixo ou estável. E existirão vários momentos em que este conceito deverá estar em uso. Por exemplo, o valor 1.23 da fórmula seguinte é uma constante:

```
RESULTADO <- ENTRADA * 1.23.
```

3.5 – Os Operadores Aritméticos

Tanto variáveis como constantes poderão ser utilizadas na elaboração de cálculos matemáticos, ou seja, na elaboração de expressões aritméticas, desde que estejam estabelecidas como do tipo real ou inteira, e para que isto ocorra é necessário à utilização de operadores aritméticos.

Os operadores aritméticos são classificados em duas classificados em duas categorias, sendo **binários** ou **unários**. São binários quando atuam em operações de exponenciação, multiplicação, divisão, adição e subtração. São unários quando atuam na inversão de um valor, atribuindo a este o sinal positivo ou negativo. Veja em seguida, a tabela de prioridade matemática existente quando da utilização destes operadores:

Operador	Operação	Prioridade	Tipo	Resultado
+	Manutenção de sinal	1	Unário	Positivo
-	Inversão de sinal	1	Unário	Negativo
^	Exponenciação	2	Binário	Real
Div	Divisão inteira	3	Binário	Inteiro
Mod	Resto da divisão	3	Binário	Inteiro
/	Divisão	3	Binário	Real
*	Multiplicação	3	Binário	Inteiro ou Real
+	Adição	4	Binário	Inteiro ou Real
-	Subtração	4	Binário	Inteiro ou Real

3.6 - As expressões aritméticas

É muito comum lidarmos com expressões aritméticas, uma vez que a maior parte do todo trabalho computacional está relacionado e envolve a utilização de cálculos. Estas expressões são definidas pelo relacionamento existente entre variáveis e constantes numéricas por meio da utilização dos operadores aritméticos. Considere a fórmula: $AREA = \pi.RAIO^2$ para o cálculo da área de uma circunferência, onde estão presentes as variáveis AREA e RAIO, a constante pi (3.14159) e os operadores aritméticos de multiplicação e também a operação de potência, que eleva o valor da variável RAIO ao quadrado.

As expressões aritméticas escritas em computação seguem um formato um pouco diferente da forma conhecida em matemática. Por exemplo a expressão: $X = \{ 43 \cdot [55 : (30 + 2)] \}$ é escrita em Pascal como: `X <- (43 * (55 / (30 + 2)))`. Perceba que as chaves e colchetes são abolidos, utilizando-se em seu lugar apenas os parênteses. É também substituído o sinal de igual (=) pelos símbolos (<-) que estão no sentido de implicado ou atribuído.

O sinal implicado ou atribuído (<-) é utilizado para indicar que o valor de uma expressão aritmética ou fórmula matemática está sendo armazenado em uma variável. No caso da fórmula para o cálculo de área de um circunferência, ela é escrita em *Portugol* da seguinte forma: `AREA <- 3.14159 * RAIO * RAIO`.

E se a fórmula a ser utilizada fosse para efetuar o cálculo da área de um triângulo, em que é necessário efetuar a multiplicação da base pela altura e em seguida dividir pela constante 2, como ficaria? Observe abaixo a fórmula padrão:

$$AREA = \frac{BASE.ALTURA}{2}$$

Ela deveria ser escrita como: `AREA <- (BASE * ALTURA) / 2`.

3.7 - Instruções básicas

As **instruções** são representadas pelo conjunto de palavras-chaves (vocabulário) de uma linguagem de programação que tem por finalidade comandar, em um computador, o seu funcionamento e a forma como os dados armazenados devem ser tratados. Deve-se ainda considerar que existem várias linguagens de programação, como: Pascal, C, Basic, SmallTalk, Fortran, Cobol, Java, entre outras, sendo que uma determinada instrução para realizar uma tarefa em um computador poderá ser escrita de forma diferente, dependendo da linguagem utilizada.

3.7.1 - Algumas regras antes de começar

Anteriormente, você aprendeu o significado de uma variável e também teve contato com algumas regras para sua utilização. Porém, teremos que ter algum cuidado de diferenciar uma referência a uma instrução de a uma variável. Desta forma teremos mais algumas regras a seguir:

- Toda referência feita a uma instrução será escrita em letra minúscula em formato negrito;
- Qualquer valor atribuído a uma variável será feito com o símbolo <-, tanto no diagrama de blocos quanto em código *português estruturado*.

3.7.2 - Entrada, processamento e saída

Para criar um programa que seja executável dentro de um computador, deve-se ter em mente três pontos de trabalho: a entrada de dados, o seu processamento e a saída deles. Sendo assim, todo programa estará trabalhando com estes três conceitos. Se os dados forem entrados de forma errada, serão conseqüentemente processados de forma errada e resultarão em respostas erradas. Desta forma, dizer a alguém que foi erro do computador é ser um tanto “mediocre”. E isto é o que mais ouvimos quando nosso saldo está errado e vamos ao banco fazer uma reclamação, ou quando recebemos uma cobrança indevida. Se houve algum erro, é porque foi causado por falha humana. Realmente é impossível um computador errar por vontade própria, pois vontade é uma coisa que os computadores não tem.

Uma entrada e uma saída podem ocorrer dentro de um computador de diversas formas. Por exemplo, uma entrada pode ser feita por teclado, modem, leitores óticos, disco, entre outras formas. Devido à grande

variedade, nossos programas utilizam as instruções **leia()** (para a entrada de dados) e **escreva()** (para a saída de dados), considerando que toda entrada vai ser via teclado e a saída via vídeo.

Diagrama de Bloco

Para as instruções **leia()** e **escreva()** serão utilizados respectivamente os símbolos: *Teclado em linha* ou *Entrada manual* (para identificar uma entrada de dados via teclado) e *Exibição* ou *Display* (para identificar uma apresentação de dados via vídeo).

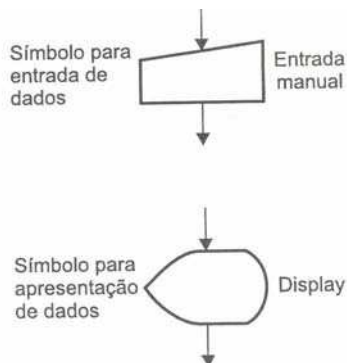


Figura 3.2 - Estrutura dos símbolos para as instruções *leia* e *escreva*.

Português Estruturado

```
leia(<lista de dados>)  
escreva(<lista de dados>)
```

Serão colocados em prática os conceitos estudados até este momento. Considere o seguinte exemplo de um problema: “Deverá ser criado um programa que efetue a leitura de dois valores numéricos. Faça a operação de soma entre os dois valores e apresente o resultado obtido”.

Note que sempre estaremos diante de um problema, o qual deverá ser resolvido primeiro por nós, para que depois seja resolvido por um computador. O que queremos dizer é que primeiro você deve entender bem o problema, para depois buscar a sua solução dentro de um computador, ou seja, você deverá “ensinar” a máquina a resolver o seu problema, por meio de um programa. Desta forma, o segredo de uma boa lógica está na compreensão adequada do problema a ser solucionado.

Com relação ao problema proposto, deverá ser primeiro muito bem interpretado. Isto ocorre com o auxílio de uma ferramenta denominada algoritmo, que deverá estabelecer todos os passos necessários a serem cumpridos na busca de uma solução para um problema. Lembre-se de que um algoritmo é na verdade uma “receita” de como fazer.

Para tanto, observe a estrutura do algoritmo com relação ao problema da leitura dos dois valores (que não conhecemos e também não precisamos conhecer, pois neste caso utilizaremos duas variáveis para trabalhar estas incógnitas A e B) e a sua respectiva soma (consequência dos valores informados, a qual também é uma incógnita e depende dos valores fornecidos; utilizaremos para esta a variável X).

Algoritmo

1. Ler dois valores, no caso variáveis A e B;
2. Efetuar a soma das variáveis A e B, implicando o seu resultado na variável X;
3. Apresentar o valor da variável X após a operação de soma dos dois valores fornecidos.

Perceba que o algoritmo é transcrição (interpretação) passo a passo de um determinado problema. É como ter um problema matemático: “João foi à feira com R\$ 20,00, comprou uma dúzia de laranjas por R\$ 5,00. Com quanto João voltou para casa?”. Na verdade, o que interessa não é o fato ocorrido com João e sim efetuar os cálculos necessários para se saber quanto sobrou na mão de João. Em processamento de dados é parecido, pois precisamos somente efetuar o levantamento das variáveis e saber o que fazer com elas.

Um detalhe a ser observado é que um algoritmo poderá ser feito de várias formas, não necessariamente como exposto acima, pois ele é a interpretação do problema. Acima está sendo utilizada a forma mais comum para iniciar o entendimento de um problema. A seguir, você terá contato com outras formas de

estabelecer algoritmos: o *Diagrama de Blocos* e o *Português Estruturado*. Todas estas formas têm em comum buscar a solução de um problema, separando-o em pequenas partes para facilitar a sua compreensão.

Diagrama de Bloco

Completada a fase de interpretação do problema e da definição das variáveis a serem utilizadas, passa-se para a fase de diagramação de bloco tradicional ou como o diagrama de quadros (diagrama N-S ou diagrama de Chapin). Neste trabalho, é adotado o critério de trabalhar com o diagrama de bloco tradicional.

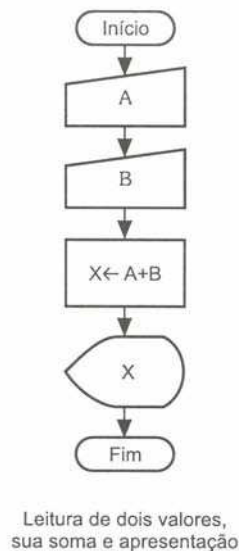


Figura 3.3 - Diagrama de bloco para a leitura, soma de dois valores e apresentação do resultado.

Observe a indicação de Início e Fim do diagrama com o símbolo *Terminal*. Este símbolo deverá estar sempre presente, indicando o ponto de início e fim de um diagrama de blocos. Note também a existência de uma seta na linha que liga um símbolo ao outro. Isto é necessário, pois desta forma sabe-se a direção que o processamento de um programa deverá seguir.

O símbolo retângulo significa *Processamento* e será utilizado para representar diversas operações, principalmente os cálculos matemáticos executados por um programa.

Português Estruturado

Tendo estabelecido os passos anteriores (algoritmos e diagrama de blocos), será efetuada a fase de codificação. Esta fase obedece ao que está definido no diagrama de blocos, pois é ele a representação gráfica da lógica de um programa. Porém, sempre deverá ser relacionado com todas as variáveis que serão utilizadas dentro do programa. Este relacionamento, além de definir os tipos de dados que serão utilizados, define também o espaço de memória que será necessário para manipular as informações fornecidas durante a execução de um programa.

Desta forma, são utilizadas no exemplo três variáveis: A, B e X, sendo que deverão ser relacionadas antes do seu uso, estabelecendo-se assim o seu respectivo tipo.

```
algoritmo "soma_numeros"
// Função : Efetuar a soma de dois valores e mostrar o resultado
// Autor : Manzano
// Data : 10/7/2005

// Seção de Declarações
var
  X: inteiro
  A: inteiro
  B: inteiro

inicio
```

```
// Seção de Comandos
  leia (A)
  leia (B)
  X <- A + B
  escreva (X)
finalgoritmo
```

Desta forma, são utilizadas no exemplo três variáveis: A, B e X, sendo que deverão ser declaradas antes do seu, estabelecendo-se assim o seu respectivo tipo.

Tendo relacionado todas as variáveis que serão utilizadas no programa com o instrução **var**, passa-se para a fase de montagem do que está estabelecido no diagrama de bloco, ou seja, de tudo que está relacionado entre os símbolos *Terminal* (indicação de início e fim do diagrama de bloco, sendo este, por conseguinte, um bloco de programa). Observe que o bloco de instruções de programa, indicado entre as instruções **início** e **fim**, é apresentado deslocado um pouco para a direita. Este estilo de escrita deve ser obedecido, para facilitar a leitura de um bloco de programa, recebendo o nome de endentação.

Após a leitura dos valores para as variáveis A e B, eles serão somados e implicados (<-) na variável X, a qual será apresentada com o valor da soma processada.

3.8 – Exercício de Aprendizagem

Abaixo são apresentados dois exemplos que aplicam os conceitos até aqui estudados. Sendo assim, olhe atentamente cada exemplo para perceber os seus detalhes.

1º Exemplo

Desenvolver a lógica para um programa que efetue o cálculo da área de uma circunferência, apresentado a medida da área calculada.

Algoritmo

Para efetuar o cálculo da área de uma circunferência é necessário conhecer a fórmula que executa este cálculo, sendo esta: $A = \pi.R^2$, em que A é a variável que conterà o resultado do cálculo da área, π é o valor de PI (3.14159, sendo uma constante na fórmula) e R o valor do raio. Sendo assim, basta estabelecer:

1. Ler um valor para o raio, no caso variável R;
2. Estabelecer que π possui o valor de 3.14159;
3. Efetuar o cálculo da área, elevando ao quadrado o valor de R e multiplicando por π ;
4. Apresentar o valor da variável A.

A fórmula para o cálculo da área passará a ser escrita como: $A \leftarrow 3.14159 * R^2$ ou se preferir:

$A \leftarrow 3.14159 * R * R$.

Diagrama de Bloco

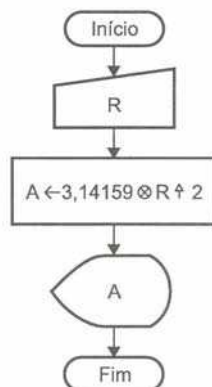


Figura 3.4 - Diagrama de bloco para o cálculo da área de uma circunferência.

Português Estruturado

```
algoritmo "area_circulo"  
// Função : Calcula a area de uma circunferencia  
// Autor : Manzano  
// Data : 10/7/2005  
  
// Seção de Declarações  
var  
  A: real  
  R: real  
  
inicio  
// Seção de Comandos  
  leia(R)  
  A  $\leftarrow$  3.14159*R^2  
  escreva(A)  
fimalgoritmo
```

2º Exemplo

Construir um programa que efetue o cálculo do salário líquido de um professor. Para fazer este programa, você deverá 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 estabelecer qual será o seu salário bruto para efetuar o desconto a ter o valor do salário líquido.

Algoritmo

1. Estabelecer a leitura da variável HT (horas trabalhadas no mês);
2. Estabelecer a leitura da variável VH (valor hora aula);
3. Estabelecer a leitura da variável PD (percentual de desconto);
4. Calcular o salário bruto (SB), sendo este a multiplicação das variáveis HT e VH;
5. Calcular o total de desconto (TD) com base no valor de PD dividido por 100;
6. Calcular o salário líquido (SL), deduzindo o desconto do salário bruto;
7. Apresentar os valores dos salários bruto e líquido: SB e SL.

Diagrama de Bloco

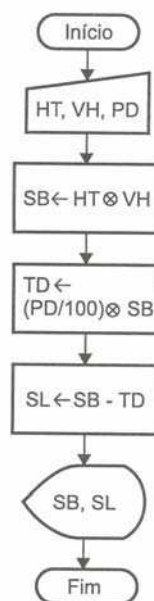


Figura 3.5 - Diagrama de bloco do programa de cálculo de salário.

Português Estruturado

```
algoritmo "Salario_Professor"
```

```
var
```

```
HT: inteiro
```

```
VH, PD, TD, SB, SL: real
```

```
inicio
```

```
leia(HT)
```

```
leia(VH)
```

```
leia(PD)
```

```
SB <- HT * VH
```

```
TD <- (PD/100) * SB
```

```
SL <- SB - TD
```

```
escreva(SB)
```

```
escreva(SL)
```

```
fimalgoritmo
```

3.9 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 01)

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

A. $a + b \frac{1}{3} =$

B. $43[55 \div (30 + 2\alpha)] =$

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

D. $2h - \left\{ \frac{45}{3h} - 4h(3-h) \right\}^2 =$

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

F. $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} =$

2) Escreva as expressões abaixo na forma convencional:

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

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

C. $((a+x) ^{(2+w)}-3*a)/2 =$

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

E. $(-5 + 96 * x - \text{raizq}(w-1)) ^{(1/y)}$

3) Classifique os conteúdos das variáveis a seguir de acordo com seu tipo, assinalando com I para

inteiros, R para reais, L para lógicos e C para literais:

- | | | |
|----------------|-------------------|----------------|
| a. () 0 | b. () Verdadeiro | c. () "Falso" |
| d. () 5.7 | e. () 45.8976 | f. () - 678 |
| g. () "cinco" | h. () "443" | i. () Falso |
| j. () "0" | k. () "Casa 8" | l. () 34 |

4) Assinale com um X os nomes de variáveis válidos:

- | | | |
|---------------------|-------------------|---------------------|
| a. () abc | b. () Verdadeiro | c. () 2lbrasil |
| d. () guarda-chuva | e. () etc. | f. () zero |
| g. () guarda_chuva | h. () leia | i. () nome*usuario |
| j. () #fone | k. () Casa 8 | l. () endereco |
| m. () "cinco" | n. () falso | o. () _falso |

5) No seguinte programa em Portugol existe algum erro? Onde?

```

algoritmo "Teste"
var
    Maria : caracter
    idade : inteiro
    letras : caracter
    Maria : real
inicio
    idade ← 23
    idade ← 678
    idade ← letra
    letras ← "ABC"
    letras ← 2
fimalgoritmo
    
```

6) Qual a diferença existente nas seguintes atribuições?

- | | |
|----------------|--------------|
| a) Letra ← "A" | b) Letra ← A |
| Nome ← "Joao" | Nome ← Joao |

7) Desenvolva os algoritmos, seus respectivos diagramas de bloco e sua codificação em Português Estruturado. Você deve gravar o exercício "a" como L01A, o exercício "b" como L01B, e assim por diante:

- 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.
- Ler uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius. A fórmula de conversão é $C \leftarrow (F - 32) * (5/9)$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
- Calcular e apresentar o valor do volume de uma lata de óleo, utilizando a fórmula:
 $Volume \leftarrow \pi * Raio^2 * Altura$
- Efetuar o cálculo da quantidade de litros de combustível gasta em uma viagem, utilizando um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto (TEMPO) e a velocidade média (VELOCIDADE) durante a viagem. Desta forma, será possível obter a distância percorrida com a fórmula $DISTANCIA \leftarrow TEMPO * VELOCIDADE$. Possuindo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula $LITROS_USADOS \leftarrow DISTANCIA / 12$. Ao final, o programa deve apresentar os valores da velocidade média (VELOCIDADE), tempo gasto na viagem (TEMPO), a distancia percorrida (DISTANCIA) e a quantidade de litros (LITROS_USADOS) utilizada na viagem.

- e) Efetuar o cálculo e a apresentação do valor de uma prestação em atraso, utilizando a fórmula $PRESTACAO \leftarrow VALOR + (VALOR * TAXA/100) * TEMPO$.
- f) Ler dois valores (inteiros, reais ou caracteres) para as variáveis A e B, e efetuar a troca dos valores de forma que a variável A passe a possuir o valor da variável B e a variável B passe a possuir o valor da variável A. Apresentar os valores trocados
- g) Ler quatro números inteiros e apresentar o resultado da adição e multiplicação, baseando-se na utilização do conceito da propriedade distributiva. Ou seja, se forem lidas as variáveis A, B, C, e D, devem ser somadas e multiplicadas A com B, A com C e A com D. Depois B com C, B com D e por fim C com D. Perceba que será necessário efetuar seis operações de adição e seis operações de multiplicação e apresentar doze resultados de saída.
- h) Elaborar um programa que calcule e apresente o volume de uma caixa retangular, por meio da fórmula $VOLUME \leftarrow COMPRIMENTO * LARGURA * ALTURA$.
- i) Ler dois inteiros (variáveis A e B) e imprimir o resultado do quadrado da diferença do primeiro valor pelo segundo.
- j) Elaborar um programa que efetue a apresentação do valor da conversão em real de um valor lido em dólar. O programa deve solicitar o valor da cotação do dólar e também a quantidade de dólares disponível com o usuário, para que seja apresentado o valor em moeda brasileira.
- k) Elaborar um programa que efetue a apresentação do valor da conversão em dólar de um valor lido em real. O programa deve solicitar o valor da cotação do dólar e também a quantidade de reais disponível com o usuário, para que seja apresentado o valor em moeda americana.
- l) Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final à soma dos quadrados dos três valores lidos.
- m) Elaborar um programa que efetue a leitura de três valores (A,B e C) e apresente como resultado final o quadrado da soma dos três valores lidos.

3.10 - Exercícios de Extras

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e sua codificação em Português Estruturado:

- a) Elaborar um programa de computador que efetue a leitura de quatro valores inteiros (variáveis A, B, C e D). Ao final o programa deve apresentar o resultado do produto (variável P) do primeiro com o terceiro valor, e o resultado do produto (variável P) do primeiro com o terceiro valor, e o resultado da soma (variável S) do segundo com o quarto valor.
- b) 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. Apresentar o valor do novo salário (variável NS).
- c) Em uma eleição sindical concorreram ao cargo de presidente três candidatos (A, B e C). Durante a apuração dos votos foram computados votos nulos e votos em branco, além dos votos válidos para cada candidato. Deve ser criado um programa de computador que efetue a leitura da quantidade de votos válidos para cada candidato, além de efetuar também a leitura da quantidade de votos nulos e votos em branco. Ao final o programa deve apresentar o número total de eleitores, considerando votos válidos, nulos e em branco; o percentual correspondente de votos válidos em relação à quantidade de eleitores; o percentual correspondente de votos válidos do candidato A em relação à quantidade de eleitores; o percentual correspondente de votos válidos do candidato B em relação à quantidade de eleitores; o percentual correspondente de votos válidos do candidato C em relação à quantidade de eleitores; o percentual correspondente de votos nulos em relação à quantidade de eleitores; e por último o percentual correspondente de votos em branco em relação à quantidade de eleitores.

4 – Estruturas de Controle – A Tomada de Decisões

Foi visto anteriormente como trabalhar com entradas, processamentos e saídas com a utilização de variáveis, constantes e operadores aritméticos. Apesar de já se conseguir solucionar problemas e transforma-los em programas, os recursos até aqui estudados são limitados, pois haverá momentos em que um determinado valor dentro de um programa necessitará ser tratado para se efetuar um processamento mais adequado. Imagine a seguinte situação: um programa que apresente a média escolar de um aluno. Até aqui, muito simples, mas além de calcular a média, o programa deve apresentar se ele está aprovado ou reprovado segundo a análise de sua média. Observe que aqui será necessário verificar a média do aluno para então *tomar uma decisão* no sentido de apresentar a sua real situação: aprovado ou reprovado

4.1 – Desvio Condicional Simples

Para solucionar o problema proposto, é necessário trabalhar uma nova instrução: **se...então...fimse**. A instrução **se...então...fimse** tem por finalidade tomar uma decisão. Sendo a condição **verdadeira**, serão executadas todas as instruções que estejam entre a instrução **se...então** e a instrução **fimse**. Sendo a condição **falsa**, serão executadas as instruções que estejam após o comando **fimse**.

Diagrama de Blocos

Observe no diagrama a existência das letras **S** e **N**, além das linhas com seta indicando a direção do processamento, colocadas juntamente com o símbolo de *Decisão*. O **S** representa **sim** e está posicionado para indicar que um determinado bloco de operações será executado quando a condição atribuída for verdadeira. O **N** está para **não** e será executado quando a condição for falsa. O símbolo do losango, ou melhor dizendo, *Decisão* deverá ser utilizado em situações em que haja a necessidade de usar uma decisão dentro do programa. uma decisão será tomada sempre com base em uma pergunta, como **RESPOSTA = "sim"**, e é esta pergunta que deverá estar indicada dentro do símbolo de losango.

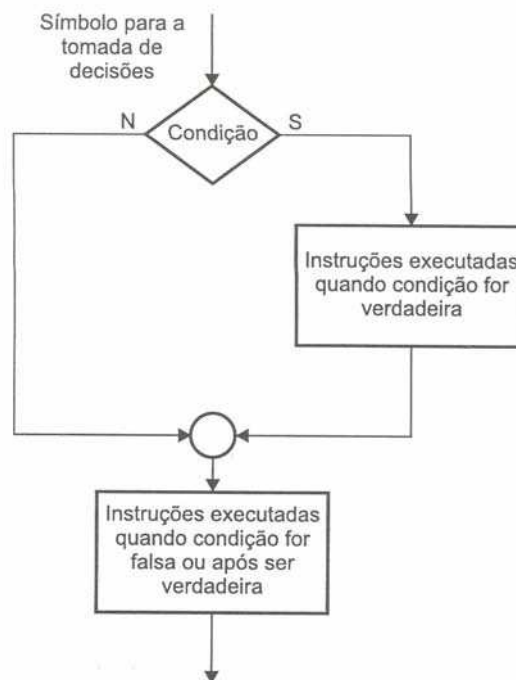


Figura 4.1 - Estrutura do símbolo para a instrução **se...então...fimse**.

Português Estruturado

```

se (<condição>) então
    <instruções para condição verdadeira>
fimse
  
```

Como um exemplo, considere o seguinte problema: "Ler dois valores numéricos, efetuar a adição e apresentar o seu resultado caso o valor somado seja maior que 10".

Algoritmo

1. Conhecer dois valores incógnitos (estabelecer variáveis A e B);
2. Efetuara a soma dos valores incógnitos A e B, implicando o valor da soma na variável X;
3. Apresentar o valor da soma contido na variável X, caso o valor de X seja maior que 10.

Diagrama de Blocos

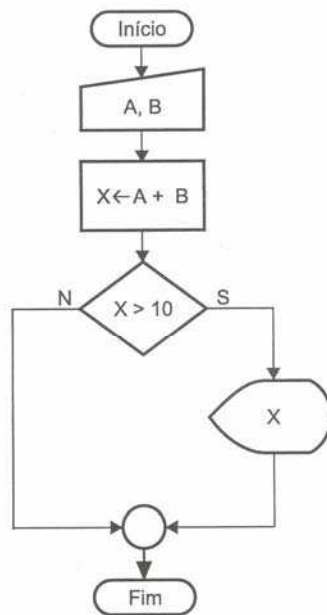


Figura 4.2 - Exemplo da utilização da estrutura se...então...fim_se.

Português Estruturado

```
algoritmo "Soma_numeros"
```

```
var
```

```
  X, A, B: inteiro
```

```
inicio
```

```
  leia(A)
```

```
  leia(B)
```

```
  X ← A + B
```

```
  se (X>10) entao
```

```
    escreva (X)
```

```
  fimse
```

```
fimalgoritmo
```

Observe que após a definição dos tipos de variáveis, é solicitada a leitura dos valores para as variáveis A e B, depois esses valores são implicados na variável X, a qual possui o resultado da adição dos dois valores. Neste ponto, é questionado no programa uma condição que permitirá imprimir o resultado da soma caso esta seja maior que 10, e não sendo, o programa é encerrado sem apresentar a referida soma, uma vez que a condição é falsa.

4.2 – Operadores Relacionais

Ao ser utilizada a instrução **se...entao...fimse**, ela implica na utilização de condições para verificar o estado de uma determinada variável quando verdadeiro ou falso. Observe que para a condição do exemplo anterior foi utilizado o sinal > (maior que) para verificar o estado da variável quanto ao seu valor. Sendo assim, uma condição também poderá ser verificada como: diferente de, igual a, menor que, maior ou igual a e menor ou igual a. Estas verificações são efetuadas com a utilização dos chamados operadores relacionais, conforme tabela seguinte:

Símbolo	Significado
=	igual a
<>	diferente
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

4.3 Desvio Condicional Composto

Já foi visto como fazer uso da instrução **se...entao...fimse**. Agora você aprenderá a fazer uso da instrução **se...entao...senao...fimse**, em que sendo a condição *Verdadeira*, é executada a instrução que estiver posicionada logo após a instrução **entao**. Sendo *Falsa*, é executada a instrução que estiver posicionada logo após a instrução **senao**.

Diagrama de Blocos

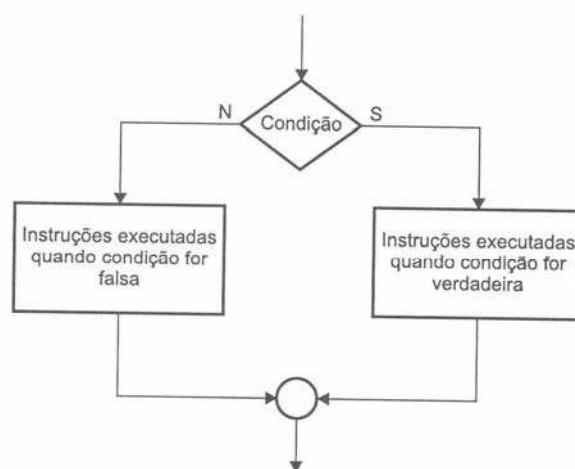


Figura 4.3 - Estrutura do símbolo para a instrução **se...entao...senao...fimse**.

Português Estruturado

```

se <(condição)> entao
    <instrução para condição verdadeira>
senao
    <instrução para condição falsa>
fimse
  
```

Para um exemplo da utilização desta estrutura considere o seguinte problema: “Ler dois valores numéricos, efetuar a adição. Caso o valor somado seja maior ou igual a 10, este deve ser apresentado somando-se a ele mais 5. Caso o valor somado não seja maior ou igual a 10, este deve ser apresentado subtraindo-se 7”.

Algoritmo

1. Conhecer dois valores (variáveis A e B);
2. Efetuar a soma dos valores A e B e implicar o valor da soma em X;
3. Verificar se X é maior ou igual a 10; caso sim, mostre X + 5, senão, mostre X - 7.

Diagrama de Blocos

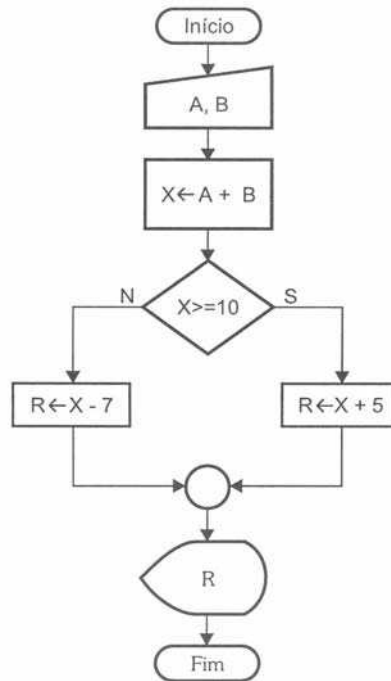


Figura 4.4 - Exemplo da utilização da estrutura se...então...senão...fim_se.

Português Estruturado

```

algoritmo "Soma_Numeros"
var
  X, A, B, R: inteiro

inicio
  leia(A, B)
  X ← A + B
  se (X ≥ 10) entao
    R ← X + 5
  senao
    R ← X - 7
  fimse
  escreva(R)
fimalgoritmo
  
```

Observe que após a definição dos tipos de variáveis, é solicitada a leitura dos valores para as variáveis A e B, depois esses valores são implicados na variável X, a qual possui o resultado da adição dos dois valores. Neste ponto, é questionado no programa uma condição que permitirá imprimir o resultado da soma adicionado de 5, caso esta seja maior ou igual a 10, e não sendo, o programa apresentará o resultado subtraindo 7.

4.4 – Desvios Condicionais Encadeados

Existem casos em que é necessário estabelecer verificação de condições sucessivas, em que uma determinada ação poderá ser executada se um conjunto anterior de instruções ou condições for satisfeito. Sendo a ação executada, ela poderá ainda estabelecer novas condições. Isto significa utilizar uma condição dentro de outra condição. Este tipo de estrutura poderá possuir diversos níveis de condição, sendo

chamadas de aninhamentos ou encadeamentos.

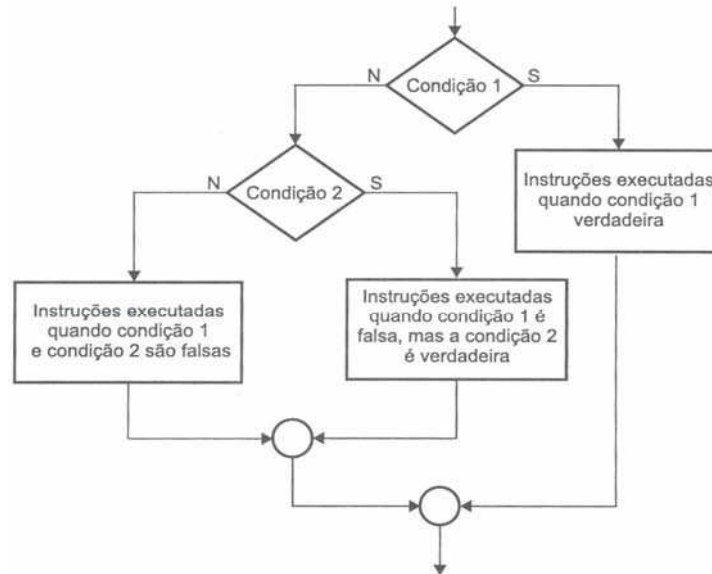


Figura 4.5 - Estrutura condicional composta ou encadeada.

Português Estruturado

Neste exemplo, está sendo adotado o encadeamento para a <condição1> falsa, mas, dependendo do problema a ser resolvido, poderá ser colocada no outro lado. Como poderá ocorrer de termos a necessidade de utilizar condição dos dois lados.

```

se (<condição1>) entao
  <instruções para condição1 verdadeira>
senao
  se (<condição2>) entao
    <instruções para condição2 verdadeira, porém condição1 falsa>
  senao
    <instruções para condição1 e condição2 falsa>
  fimse
fimse
  
```

Para um exemplo da utilização desta estrutura considere o seguinte problema: “Elaborar um programa que efetue o cálculo do reajuste de salário de um funcionário. Considere que o funcionário deverá 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 deverá ser de 5%”. Veja o algoritmo, diagrama de blocos e a codificação em Português Estruturado.

Algoritmo

Perceba que o problema em questão estabelece três condições para calcular o reajuste do salário do funcionário, sendo:

- Salário < 500, reajuste será de 15%
- Salário >= 500, mas <= 1000, reajuste será de 10%
- Salário > 1000, reajusto será de 5%

Estas condições deverão estar encadeadas, pois todas as possibilidades de reajuste deverão ser cercadas.

1. Definir uma variável para o salário reajustado: NOVO_SALARIO;
2. Ler um valor para a variável SALARIO;
3. Verificar se o valor do SALARIO < 500, se sim reajustar em 15%;
4. Verificar se o valor do SALARIO <= 1000, se sim reajustar em 10%;
5. Verificar se o valor do SALARIO > 1000, se sim reajustar em 5%;
6. Apresentar o valor reajustado, implicando em NOVO_SALARIO.

Diagrama de Blocos

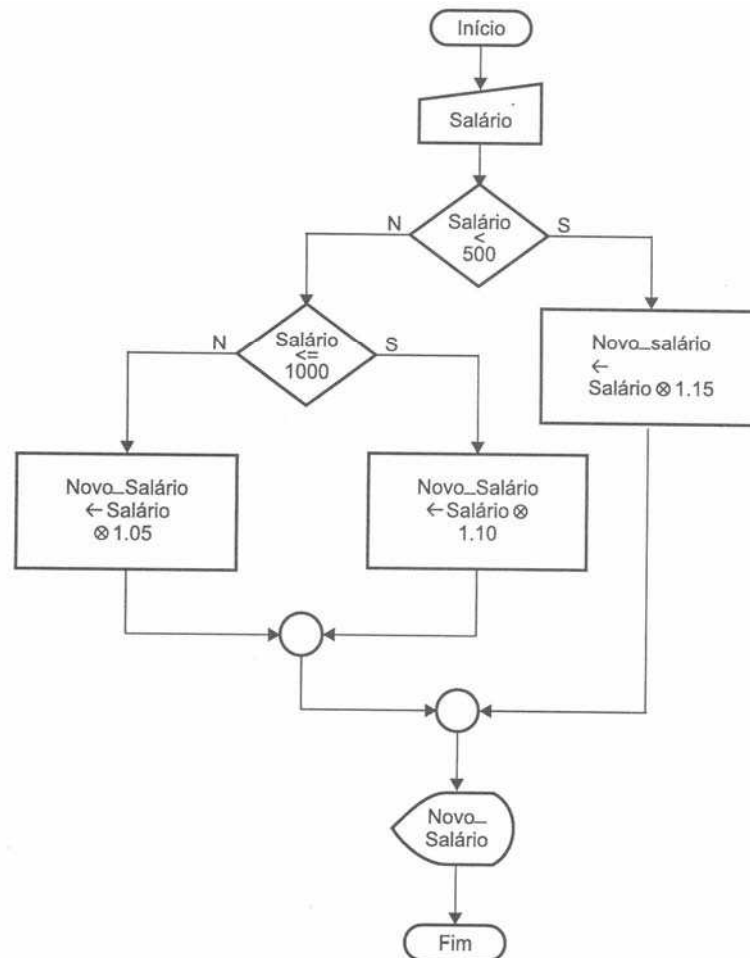


Figura 4.6 - Exemplo da utilização de uma estrutura condicional encadeada.

Observe que a referência feita na linha 5 do algoritmo não é escrita no diagrama de bloco e nem em Português Estruturado, uma vez que ela fica subentendida, ou seja, qualquer valor que não seja menor que 500 ou que não esteja situado na faixa de 500 a 1000 está, conseqüentemente, acima de 1000.

Português Estruturado

```

algoritmo "Reajusta_Salario"
var
    novo_salario: real
    salario: real

inicio
    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(novo_salario)
fimalgoritmo
    
```


4.5 Operadores Lógicos

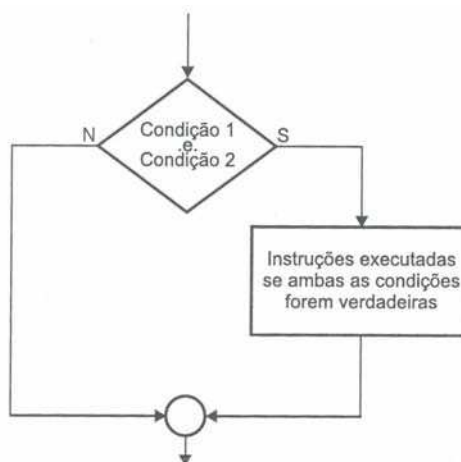
Há ocasiões em que é necessário trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo na mesma instrução **se...entao**, e efetuar deste modo testes múltiplos. Para esses casos é necessário empregar os operadores lógicos, também conhecidos como operadores booleanos. Os operadores lógicos são três: **E**, **OU** e **NAO**. Em alguns casos, o uso de operadores lógicos evita a utilização de muitas instruções **se...entao** encadeadas.

4.5.1 Operador Lógico: E

O operador do tipo **E** é utilizado quando dois ou mais relacionamentos lógicos de uma determinada condição necessitam ser verdadeiros. A seguir é apresentada a tabela-verdade para este tipo de operador:

E		
Condição 1	Condição 2	Resultado
Falso	Falso	Falso
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Verdadeiro	Verdadeiro	Verdadeiro

Diagrama de Blocos



Português Estruturado

```

se (<condição1> e (<condição2>) entao
  <instruções executadas se condição1 e condição2 verdadeira>
fimse
  
```

O operador **E** faz com que somente seja executada uma determinada operação se todas as condições mencionadas forem simultaneamente verdadeiras, gerando assim um resultado lógico verdadeiro. Veja o seguinte exemplo:

```

algoritmo "testa_logica_e"
var
  numero: inteiro

inicio
  leia(numero)
  se (numero >= 20) e (numero <=90) entao
    escreva("O numero esta na faixa de 20 a 90")
  senao
    escreva("O numero esta FORA da faixa de 20 a 90")
  fimse
fimalgoritmo
  
```

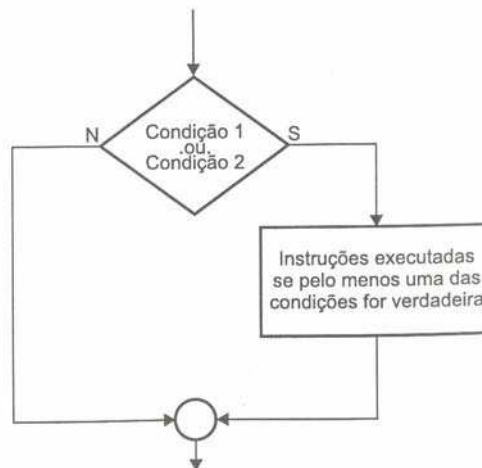
O exemplo mostra, por meio da utilização do operador **E**, que somente é apresentada à mensagem **O número esta na faixa de 20 a 90**, caso o valor fornecido para a variável *NUMERO* seja um valor entre 20 e 90. Qualquer valor fornecido fora da faixa definida, apresenta a mensagem **O número esta FORA da faixa de 20 a 90**.

4.5.2 Operador Lógico: OU

O operador do tipo **OU** é utilizado quando pelo menos um dos relacionamentos lógicos de uma condição necessita ser verdadeiro.

OU		
Condição 1	Condição 2	Resultado
Falso	Falso	Falso
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Diagrama de Blocos



Português Estruturado

```

se (<condição1>) ou (<condição2>) entao
    <instruções executadas se condição1 ou condição2 verdadeira>
fimse
  
```

O operador **OU** faz com que seja executada uma determinada operação se pelo menos uma das condições mencionadas gera um resultado lógico verdadeiro. Veja o exemplo seguinte:

```

algoritmo "testa_logica_ou"
var
    SEXO: literal

inicio
    leia(SEXO)
    se (SEXO = "masculino") ou (SEXO = "feminino") entao
        escreva("Seu sexo existe.")
    senao
        escreva("Desconheço seu sexo.")
    fimse
fim algoritmo
  
```

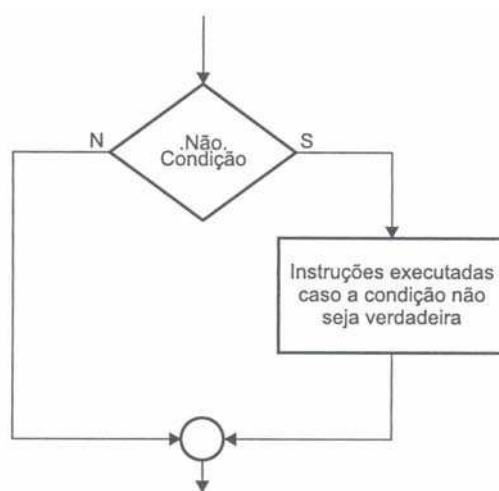
O exemplo mostra, por meio da utilização do operador **OU**, que somente é apresentada à mensagem "O seu sexo existe", caso o valor fornecido para a variável *SEXO* seja "masculino" ou "feminino" (em caracteres minúsculos). Qualquer outro valor fornecido apresenta a mensagem "Desconheço seu sexo".

4.5.3 Operador Lógico: NAO

O operador do tipo **NAO** é utilizado quando se necessita estabelecer que uma determinada condição deve não ser verdadeira ou deve não ser falsa. O operador **NAO** se caracteriza por inverter o estado lógico de uma condição. A seguir é apresentada a tabela-verdade para este tipo de operador.

NAO	
Condição	Resultado
Verdadeiro	Falso
Falso	Verdadeiro

Diagrama de Blocos



Português Estruturado

```

se nao(<condição1>) entao
    <instruções executadas se condição1 não for verdadeira>
fimse
  
```

O operador **NAO** faz com que seja executada uma determinada operação invertendo o resultado lógico da condição. Veja o exemplo a seguir:

```

algoritmo "Testa_Logica_NAO"
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)
fimalgoritmo
  
```

O exemplo descrito mostra, por meio da utilização do operador **NAO**, 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 cima efetua o cálculo $C \leftarrow (A-B) * X$.

4.6 – Exercício de Aprendizagem

Para demonstrar a utilização de operadores lógicos em um exemplo um pouco maior, considere os seguintes exemplos:

1º Exemplo

“Ler três valores para os lados de um triângulo, considerando lados como: A, B e C. Verificar se os lados fornecidos formam realmente um triângulo. Se for esta condição verdadeira, deve ser indicado qual tipo de triângulo foi formado: isósceles, escaleno ou equilátero.”

Algoritmo

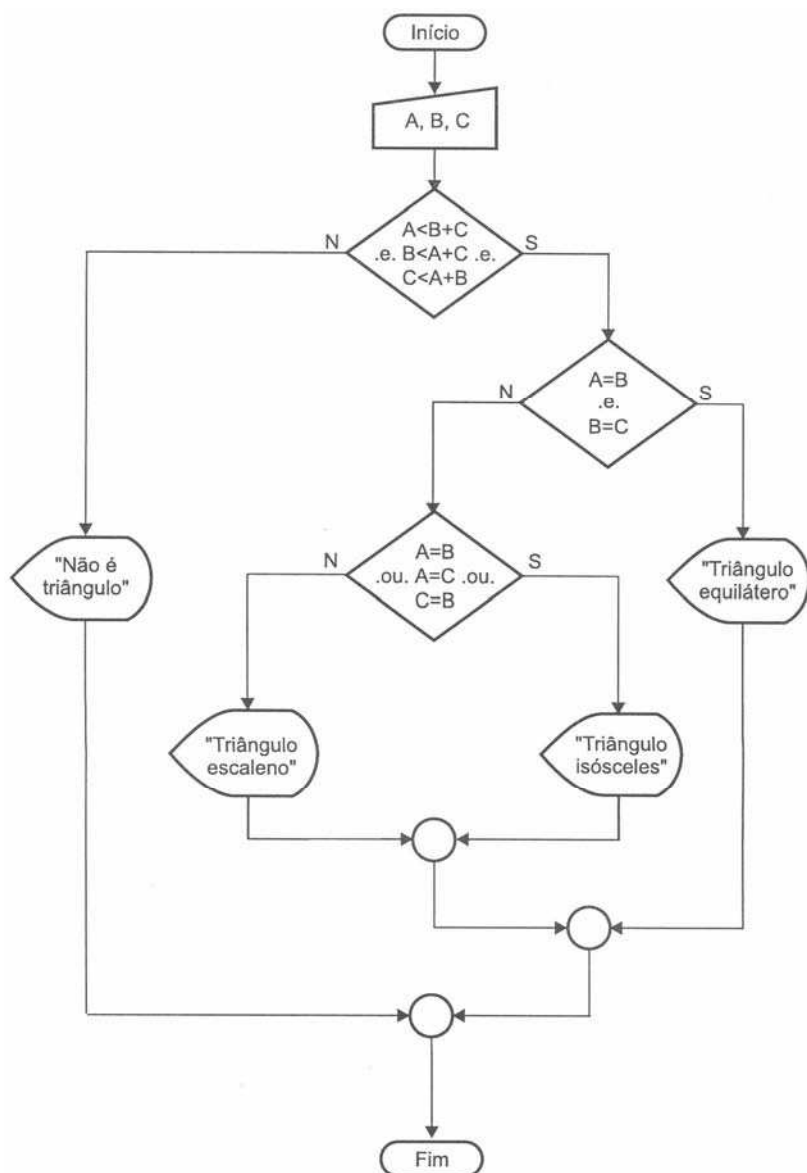
Para se estabelecer esse algoritmo é necessário em primeiro lugar saber o que realmente é um triângulo. Triângulo é uma forma geométrica (polígono) composta por três lados, em que cada lado é menor que a soma dos outros dois lados. Perceba que esta é uma regra (uma condição) e deve ser considerada. É um triângulo quando $A < (B + C)$, quando $B < (A + C)$ e quando $C < (A + B)$.

Tendo certeza de que os valores informados para os três lados formam um triângulo, são então analisados os valores para se estabelecer qual tipo de triângulo é formado: isósceles, escaleno ou equilátero.

Um triângulo é isósceles quando possui dois lados iguais e um diferente, sendo $A=B$ ou $A=C$ ou $B=C$; é escaleno quando possui todos os lados diferentes, sendo $A \neq B$ e $B \neq C$ e $C \neq A$ e é equilátero quando possui todos os lados iguais, sendo $A=B$ e $B=C$.

1. Ler três valores para os lados de um triângulo: A, B e C;
2. Verificar se cada lado é menor que a soma dos outros dois lados;
 - a. Se sim, saber se $A=B$ e se $B=C$ sendo verdade, o triângulo é equilátero;
 - b. Se não, verificar se $A=B$ ou se $A=C$ ou se $B=C$; sendo verdade, o triângulo é isósceles, caso contrário o triângulo é escaleno.
3. Caso os lados fornecidos não caracterizem um triângulo, avisa a ocorrência.

Diagrama de Blocos



Português Estruturado

```

algoritmo "Triangulo"
var
  a, b, c: real

inicio
leia(a, b, c)
se (a < b + c) e (b < a + c) e (c < a + b) entao
  se (a=b) e (b=c) entao
    escreva("Triangulo Equilatero")
  senao
    se (a=b) ou (a=c) ou (c=b) entao
      escreva("Triangulo Isosceles")
    senao
      escreva("Triangulo Escaleno")
    fimse
  fimse
senao
  escreva("As medidas nao formam um triangulo")
fimse
fimalgoritmo
  
```

2º Exemplo

Desenvolver um programa que efetue a leitura de um valor numérico inteiro e apresente-o caso este valor seja divisível por 4 e 5. Não sendo divisível por 4 e 5 o programa deverá apresentar a mensagem “Não é divisível por 4 e 5”.

Para resolver o problema proposto é necessário além do uso do operador lógico **E**, efetuar a verificação da condição do valor lido ser ou não divisível por 4 e 5. Para verificar se um valor numérico (dividendo) é divisível por outro valor numérico (divisor) é necessário levar em consideração que o resto da divisão deverá ser ZERO, considerando ainda que, o dividendo, o divisor e o quociente da divisão serão valores inteiros.

Anteriormente foi apresentado como efetuar operações aritméticas de divisão utilizando-se o operador “/” (barra). Este operador quando utilizado resulta em um quociente do tipo real. Por exemplo, se for efetuada a operação **5/2** (cinco dividido por dois), ter-se-á como quociente da divisão o valor **2.5**. Toda a divisão efetuada com o operador aritmético “/” (barra) tende a obter o resto da divisão igual a ZERO. Não servindo assim, para verificar a possibilidade de um valor numérico ser divisível por outro valor numérico.

Para detectar se um valor numérico inteiro é divisível por outro valor numérico inteiro deverá ser utilizado o operador aritmético de divisão “**div**”, que dará como resultado um valor numérico inteiro como quociente da divisão. O operador **div** não existe na ciência matemática, mas existente é considerado para operações computacionais. Desta forma, se for efetuada a operação **5 div 2**, ter-se-á como quociente da divisão o valor **2**. Sendo o quociente **2** isto indica que o resto da divisão está como o valor **1**. Desta forma, pode-se concluir que o valor numérico **5** (dividendo) não é divisível pelo valor numérico **2** (divisor), pois o resto da divisão não é ZERO, como mostra a figura a seguir.

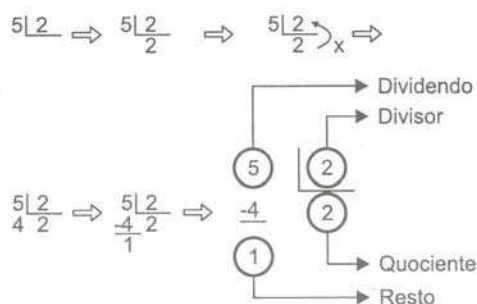


Figura 4.11 - Discriminação de uma operação de divisão com valores numéricos inteiros.

De acordo com o que está exposto na figura acima fica fácil efetuar o cálculo do resto da divisão entre dois valores numéricos inteiros. Para tanto, basta considerar a fórmula: **Resto = Dividendo – Divisor * Quociente**. Assim sendo, pode-se substituir a fórmula pelos seus valores, sendo: **Resto = 5 - 2*2**. Em seguida, basta efetuar o cálculo da multiplicação de **2*2**, para então subtrair o valor **4** do valor **5**, e assim obter o valor de resto igual a **1**.

A forma exposta no parágrafo anterior é a forma a ser utilizada em um programa de computador para determinar se um valor numérico é divisível por outro valor numérico. Supondo, que um programa deva verificar se um valor, no caso o valor numérico **5** é divisível pelo valor numérico **2**, este deveria efetuar o cálculo do resto da divisão da seguinte forma: **RESTO ← 5 – 2 * (5 div 2)**.

Algoritmo

1. Ler um número inteiro qualquer, no caso o número N;
2. Calcular o resto da divisão de N por 4, usar a variável R_4;
3. Calcular o resto da divisão de N por 5, usar a variável R_5;
4. Verificar se ambas as variáveis possuem o valor ZERO, se sim apresentar a variável N, se não apresentar a mensagem “Não é divisível por 4 e 5”.

Diagrama de Blocos

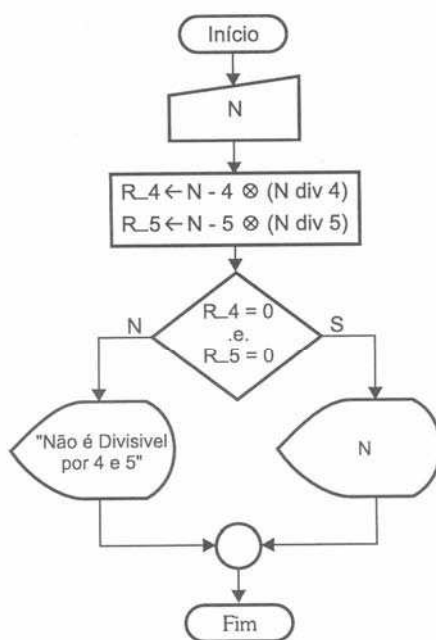


Figura 4.12 - Diagrama de blocos para verificar se N é divisível por 4 e 5.

Português Estruturado

```

algoritmo "Divisível"
var
    N, R_4, R_5: inteiro

inicio
    leia(N)
    R_4 ← N - 4 * (N div 4)
    R_5 ← N - 5 * (N div 5)
    se (R_4=0) e (R_5=0) entao
        escreva(N)
    senao
        escreva("Nao é divisível por 4 e 5")
    fimse
fimalgoritmo
    
```

É possível uma outra solução bem mais simples utilizando-se da instrução **mod** que fornece o resto da divisão inteira. Veja como ficaria esta solução:

```

algoritmo "Divisível"
var
    N, R_4, R_5: inteiro

inicio
    leia(N)
    se (N mod 4 = 0) e (N mod 5 = 0) entao
        escreva(N)
    senao
        escreva("Nao é divisível por 4 e 5")
    fimse
fimalgoritmo
    
```

4.7 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 02)

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

- A. $2 > 3 =$
- B. $(6 < 8) \text{ ou } (3 > 7) =$
- C. $\text{não } (2 < 3) =$
- D. $(5 \geq 6 \text{ ou } 6 < 7 \text{ ou não}(a+5-6=8)) \quad \{\text{onde } a = 5\}$

2) Determine o resultado lógico das expressões mencionadas, indicando se são verdadeiras ou falsas. Considere para as respostas os seguintes valores $x=1$, $a=3$, $b=5$, $c=8$ e $d=7$:

- A. $\text{não}(x > 3) =$
- B. $(x < 1) \text{ e } \text{não}(b > d) =$
- C. $\text{não}(d < 0) \text{ e } (c > 5) =$
- D. $\text{não}(x > 3) \text{ e } (c < 7) =$
- E. $(a > b) \text{ ou } (c > b) =$
- F. $(x \geq 2) =$
- G. $(x < 1) \text{ e } (b \geq d) =$
- H. $(d < 0) \text{ ou } (c > 5) =$
- I. $\text{não}(d > 3) \text{ ou } \text{não}(b < 7) =$
- J. $(a > b) \text{ ou } \text{não}(c > b) =$

3) Indique a saída dos trechos de programas em português estruturado, mostrados abaixo. Para as saídas considere os seguintes valores: $a=2$, $b=3$, $c=5$ e $d=9$.

A. Resposta: _____

```
se não(d > 5) então  
    escreva ("x <- (a + b) * d")  
senão  
    escreva ("x <- (a-b) / c")  
fimse
```

B. Resposta: _____

```
se (a > 2) e (b < 7) então  
    escreva ("x <- (a + 2) * (b - 2)")  
senão  
    escreva ("x <- (a + b) / d * (c + d)")  
fimse
```

C. Resposta: _____

```
se (a = 2) ou (b < 7) então  
    escreva ("x <- (a + 2) * (b - 2)")  
senão  
    escreva ("x <- (a + b) / d * (c + d)")  
fimse
```

D. Resposta: _____

```
se (a > 2) ou não(b < 7) então  
    escreva ("x <- a + b - 2")  
senão  
    escreva ("x <- a - b")
```


fimse

E. Resposta: _____

```
se nao(a > 2) ou nao(b < 7) entao
    escreva ("x <- a + b")
senao
    escreva ("x <- a / b")
fimse
```

F. Resposta: _____

```
se nao(a > 3) e nao(b < 5) entao
    escreva ("x <- a + d")
senao
    escreva ("x <- d / b")
fimse
```

G. Resposta: _____

```
se (c >= 2) e (b <= 7) entao
    escreva ("x <- (a + d) / 2")
senao
    escreva ("x <- d * c")
fimse
```

H. Resposta: _____

```
se (a >= 2) ou (c <= 1) entao
    escreva ("x <- (a + d) / 2")
senao
    escreva ("x <- d * c")
fimse
```

4) Desenvolva os algoritmos, seus respectivos diagramas de bloco e sua codificação em Português Estruturado (Você deve gravar o exercício “a” como L02A, o exercício “b” como L02B, e assim por diante):

- a. Ler dois valores numéricos inteiros e apresentar o resultado da diferença do maior pelo menor valor.
- b. Efetuar a leitura de um valor inteiro positivo ou negativo e apresentar o número lido como sendo um valor positivo, ou seja, o programa deverá apresentar o módulo de um número fornecido. Lembre-se de verificar se o número fornecido é menor que zero; sendo, multiplique-o por -1.
- c. Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir uma mensagem dizendo que o aluno foi aprovado, se o valor da média escolar for maior ou igual a 5. Se o aluno não foi aprovado, indicar uma mensagem informando esta condição. Apresentar junto das mensagens o valor da média do aluno para qualquer condição.
- d. Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir uma mensagem dizendo que o aluno foi aprovado, se o valor da média escolar for maior ou igual a 7. Se o valor da média for menor que 7, solicitar a nota de exame, somar com o valor da média e obter nova média. Se a nova média for maior ou igual a 5, apresentar uma mensagem dizendo que o aluno foi aprovado em exame. Se o aluno não foi aprovado, indicar uma mensagem informando esta condição. Apresentar com as mensagens o valor da média do aluno, para qualquer condição.
- e. Efetuar a leitura de três valores (variáveis A, B e C) e efetuar o cálculo da equação completa de segundo grau, apresentando as duas raízes, se para os valores informados for possível efetuar o referido cálculo. Lembre-se de que a variável A deve ser diferente de zero.
- f. Efetuar a leitura de três valores (variáveis A, B e C) e apresentá-los dispostos em ordem crescente.
- g. Efetuar a leitura de quatro números inteiros e apresentar os números que são divisíveis por 2 e 3.
- h. Efetuar a leitura de cinco números inteiros e identificar o maior e o menor valores.
- i. Elaborar um programa que efetue a leitura de um número inteiro e apresentar uma mensagem informando se o número é par ou ímpar.
- j. Elaborar um programa que efetue a leitura de um valor que esteja entre a faixa de 1 a 9. Após a

leitura do valor fornecido pelo usuário, o programa deverá indicar uma de duas mensagens: "O valor está na faixa permitida", caso o usuário forneça o valor nesta faixa, ou a mensagem "O valor está fora da faixa permitida", caso o usuário forneça valores menores que 1 ou maiores que 9.

- k. Elaborar um programa que efetue a leitura de um determinado valor inteiro, e efetue a sua apresentação, caso o valor não seja maior que três.
- l. Elaborar um programa que efetue a leitura do nome e do sexo de uma pessoa, apresentando com saída uma das seguintes mensagens: "Ilmo Sr.", se o sexo informado como masculino, ou a mensagem "Ilma Sra.", para o sexo informado como feminino. Apresente também junto da mensagem de saudação o nome previamente informado.

5 – Estruturas de Controle – Laços ou Malhas de Repetição

Existem ocasiões em que é necessário efetuar a repetição de um trecho de programa um determinado número de vezes. Neste caso, poderá ser criado um looping que efetue o processamento de um determinado trecho, tantas vezes quantas forem necessárias. Os loopings também são chamados de laços de repetição ou malhas de repetição.

Supondo um programa que deva executar um determinado trecho de instruções por cinco vezes. Com o conhecimento adquirido até este momento, o leitor com toda a certeza iria escrever o mesmo trecho, repetindo-o o número de vezes necessárias. Por exemplo, imagine um programa que peça a leitura de um valor para a variável X, multiplique esse valor por 3, implicando-o a variável de resposta R, e apresente o valor obtido, repetindo esta seqüência por cinco vezes, conforme mostrado abaixo em português estruturado:

```
algoritmo "Pede_Numero"
var
  X: inteiro
  R: inteiro

inicio
  leia(X)
  R <- X * 3
  escreval(R)
  leia(X)
  R <- X * 3
  escreval(R)
  leia(X)
  R <- X * 3
  escreval(R)
  leia(X)
  R <- X * 3
  escreval(R)
  leia(X)
  R <- X * 3
  escreval(R)
finalgoritmo
```

Para estes casos existem comando apropriados para efetuar a repetição de determinados trechos de programas o número de vezes que for necessário. A principal vantagem deste recurso é que o programa passa a ter um tamanho menor, podendo sua amplitude de processamento ser aumentada sem alterar o tamanho do código de programação. Desta forma, podem-se determinar repetições com números variados de vezes.

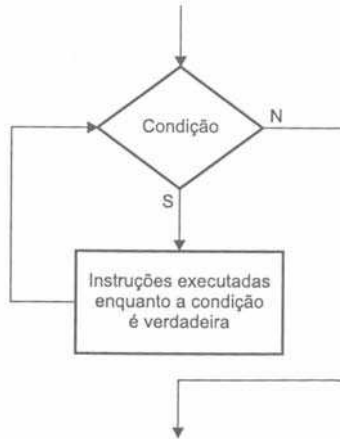
5.1 – Repetição do Tipo: Teste Lógico no Início do Looping

Caracteriza-se por uma estrutura que efetua um teste lógico no início de um looping, verificando se é permitido executar o trecho de instruções subordinado a esse looping. A estrutura em questão é denominada de **enquanto**, sendo conseguida com a utilização do conjunto de instruções **enquanto...faca...fimenquanto**.

A estrutura **enquanto...faca...fimenquanto** tem o seu funcionamento controlado por decisão. Sendo assim, poderá executar um determinado conjunto de instruções enquanto a condição verificada for **Verdadeira**. No momento em que esta condição se torna **Falsa**, o processamento da rotina é desviado para fora do looping. Se a condição for **Falsa** logo de início, as instruções contidas no looping são ignoradas.

Diagrama de Blocos

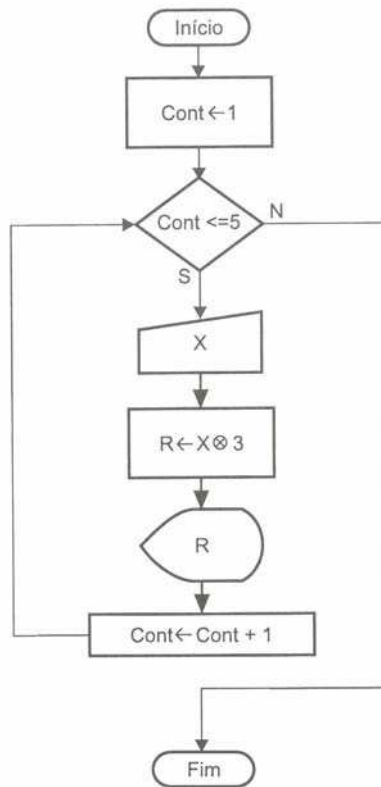
Cuidado para não confundir esta nova estrutura com a estrutura de decisão usada anteriormente. Aqui existe um retorno à condição após a execução do bloco de operações, até que a condição se torne falsa.



Algoritmo

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 3, implicando o resultado em R;
5. Apresentar o valor calculado contido na variável R;
6. Acrescentar +1 a variável contador, definida no passo 1;
7. Quando o contador for maior que 5, encerrar o processamento do looping.

Diagrama de Blocos



Português Estruturado

```

algoritmo "Looping_1A"
var
    X, R: inteiro
    CONT: inteiro

inicio
    CONT ← 1
    enquanto (CONT ≤ 5) faca
        leia(X)
        R ← X * 3
        escreva(R)
        CONT ← CONT + 1
    fimenquanto
fimalgoritmo
    
```

Além da utilização das variáveis X e R, foi necessário criar uma terceira variável (*CONT*) para controlar a contagem do número de vezes que o trecho de programa deverá ser executado.

Logo após o início do programa, a variável contador é atribuída com o valor 1 (*CONT* ← 1). Em seguida, a instrução **enquanto** (*CONT* ≤ 5) **faca** efetua a checagem da condição estabelecida, verificando que a condição é verdadeira, pois o valor da variável *CONT* neste momento é 1, é realmente menor ou igual a 5, e enquanto for deverá processar o looping.

Sendo assim, tem início a execução da rotina de instruções contidas entre as instruções **enquanto** e a instrução **fimenquanto**. Depois de efetuar a primeira leitura, cálculo e apresentação do valor calculado, o programa encontra a linha *CONT* ← *CONT* + 1, sendo assim resultará *CONT*=2.

Agora que a variável *CONT* possui o valor 2, o processamento do programa volta para a instrução **enquanto** (*CONT* ≤ 5) **faca**, uma vez que o valor da variável *CONT* é menor ou igual a 5. Será então executada a rotina de instruções, só que desta vez a variável *CONT* passará a possuir o valor 3. Desta forma o programa processará novamente a rotina de instruções, passando o valor de *CONT* para 4, que será verificado e sendo menor ou igual a 5, será executada mais uma vez a mesma rotina de instruções. Neste ponto, a variável *CONT* passa a possuir o valor 5. Perceba que a instrução **enquanto** (*CONT* ≤ 5) **faca** irá efetuar a checagem do valor da variável *CONT* que é 5 com a condição *CONT* ≤ 5. Veja que 5 não é menor que 5,

mas é igual. Sendo esta condição verdadeira, deverá a rotina ser executada mais uma vez. Neste momento, o valor da variável *CONT* passa a ser 6, que resultará para a instrução **enquanto** uma condição falsa. E por conseguinte, desviará o processamento para a primeira instrução após a instrução **fimenquanto**, que no caso é a instrução **fimalgoritmo**, dando o encerramento do programa.

Para ilustrar de forma um pouco diferente, imagine que o problema anterior deverá ser executado enquanto o usuário queira. Desta forma, em vez de possuir dentro da rotina um contador de vezes, pode-se possuir uma instrução pedindo que o usuário informe se deseja continuar ou não. Veja, a seguir, o exemplo desta nova situação.

Algoritmo

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, implicando o resultado em R;
5. Apresentar o valor calculado contido na variável R;
6. Quando a resposta for diferente de **sim**, encerrar o processamento.

Diagrama de Blocos

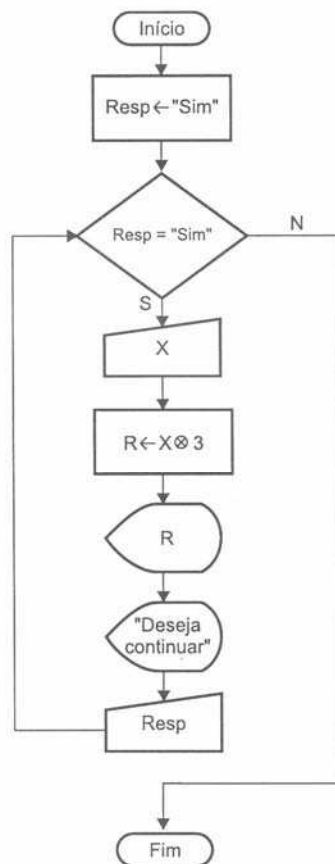


Figura 5.3 - Exemplo de um looping controlado pelo usuário.

Português Estruturado

```

algoritmo "Looping_1B"
var
  X, R: inteiro
  RESP: literal

inicio
  RESP ← "sim"

```

```

enquanto (RESP = "sim") faca
  leia(X)
  R <- X * 3
  escreval(R)
  escreval("Deseja continuar? ")
  leia(RESP)
fimenquanto
fimalgoritmo

```

Veja que o contador foi substituído pela variável *RESP*, que enquanto for igual a “sim” executará a rotina existente entre as instruções **enquanto** e **fimenquanto**. Neste caso, o número de vezes que a rotina irá se repetir será controlado pelo usuário e será encerrada somente quando alguma informação diferente de “sim” for fornecida para a variável *RESP*.

5.1.1 - Exercícios de Entrega Obrigatória (até ___/___/___) (Lista 03)

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado. Usar na resolução dos problemas apenas estruturas de repetição do tipo **enquanto** (Você deve gravar o exercício “a” como L03A, o exercício “b” como L03B, e assim por diante).

- a) Apresentar os resultados de uma tabuada de multiplicar (de 1 até 10) de um número qualquer.
- b) Apresentar o total da soma obtida dos cem primeiros números inteiros (1+2+3+4+...+98+99+100).
- c) Elaborar um programa que apresente no final o somatório dos valores pares existentes na faixa de 1 até 500.
- d) Apresentar todos os valores numéricos inteiros ímpares situados na faixa de 0 a 20. Para verificar se o número é ímpar, efetuar dentro da malha a verificação lógica desta condição com a instrução se, perguntando se o número é ímpar; sendo, mostre-o; não sendo, passe para o próximo passo.
- e) Apresentar os resultados das potências de 3, variando do expoente 0 até o expoente 15. Deve ser considerado que qualquer número elevado a zero é 1, e elevado a 1 é ele próprio. Observe que neste exercício não pode ser utilizado o operador de exponenciação do português (^).
- f) Elaborar um programa que apresente como resultado o valor de uma potência de uma base qualquer elevada a um expoente qualquer, ou seja, de B^E , em que B é o valor da base e E o valor do expoente. Observe que neste exercício não pode ser utilizado o operador de exponenciação do português (^).
- g) Escreva um programa que apresente a série de Fibonacci até o décimo quinto termo. A série de Fibonacci é formada pela seqüência: 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., etc. Esta série se caracteriza pela soma de um termo atual com o seu anterior subsequente, para que seja formado o próximo valor da seqüência. Portanto começando com os números 1, 1 o próximo termo é 1+1=2, o próximo é 1+2=3, o próximo é 2+3=5, o próximo 3+5=8, etc.
- h) Elaborar um programa que apresente os valores de conversão de graus Celsius em Fahrenheit, de 10 em 10 graus, iniciando a contagem em 10 graus Celsius e finalizando em 100 graus Celsius. O programa deve apresentar os valores das duas temperaturas. A fórmula de conversão é $F = \frac{9C + 160}{5}$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
- i) Elaborar um programa que efetue a leitura de 10 valores numéricos e apresente no final o total do somatório e a média aritmética dos valores lidos.
- j) Elaborar um programa que apresente os resultados da soma e da média aritmética dos valores pares situados na faixa numérica de 50 a 70.
- k) Elaborar um programa que possibilite calcular a área total de uma residência (sala, cozinha, banheiro, quartos, área de serviço, quintal, garagem, etc.). O programa deve solicitar a entrada do nome, a largura e o comprimento de um determinado cômodo. Em seguida, deve apresentar a área

do cômodo lido e também uma mensagem solicitando do usuário a confirmação de continuar calculando novos cômodos. Caso o usuário responda “NAO”, o programa deve apresentar o valor total acumulado da área residencial.

- l) Elaborar um programa que efetue a leitura de valores positivos inteiros até que um valor negativo seja informado. Ao final devem ser apresentados o maior e o menor valores informados pelo usuário.

5.2 – Repetição do Tipo: Teste Lógico no Fim do Looping.

Caracteriza-se por uma estrutura que efetua um teste lógico no fim de um looping. Esta estrutura é parecida com a **enquanto**. A estrutura em questão é denominada de **repita**, sendo conseguida com a utilização do conjunto de instruções **repita...ate**.

A estrutura **repita...ate** tem o seu funcionamento controlado por decisão. Porém, irá efetuar a execução de um conjunto de instruções *pelo menos uma vez* antes de verificar a validade da condição estabelecida. Diferente da estrutura **enquanto** que executa somente um conjunto de instruções, enquanto a condição é verdadeira.

Desta forma **repita** tem seu funcionamento em sentido contrário a **enquanto**, pois sempre irá processar um conjunto de instruções no mínimo uma vez até que a condição se torne **Verdadeira**. Para a estrutura **repita** um conjunto de instruções é executado enquanto a condição se mantém **Falsa** e até que ela seja **Verdadeira**.

Para exemplificar a utilização de **repita**, será utilizado o mesmo exemplo anterior: “Pedir a leitura de um valor para a variável X, multiplicar esse valor por 3, implicando-o à variável R, e apresentar o valor obtido, repetindo esta seqüência por cinco vezes”.

Algoritmo

1. Criar uma variável para servir como contador com valor inicial 1;
2. Ler um valor para a variável X ;
3. Efetuar a multiplicação do valor de X por 3, implicando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Acrescentar +1 a variável contador, definida no passo 1;
6. Repetir os passos 2, 3, 4 e 5 até que o contador seja maior que 5.

Diagrama de Blocos

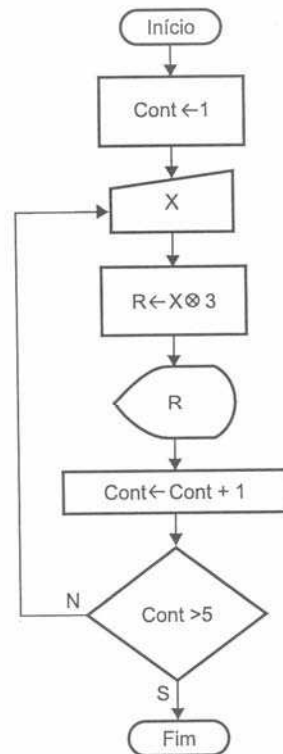


Figura 5.4 - Exemplo da utilização da instrução *repita...até_que*.

Português Estruturado

```

algoritmo "Looping_2A"
var
    X, R: inteiro
    CONT: inteiro

inicio
    CONT ← 1
    repita
        leia (X)
        R ← X * 3
        escreval(R)
        CONT ← CONT + 1
    ate (CONT > 5)
fimalgoritmo
    
```

Logo após o início do programa, a variável contador é atribuída com o valor 1 (`CONT ← 1`). Em seguida, a instrução **repita** indica que todo trecho de instruções situado até a instrução **ate** (`CONT > 5`) deverá ter seu processamento repetido até que a condição (`CONT > 5`) seja satisfeita.

Sendo assim, tem início a execução da rotina de instruções contidas entre as instruções **repita...ate**. Depois de efetuar a primeira execução das instruções o programa encontra a linha `CONT ← CONT + 1`; neste momento a variável `CONT` é somada a mais 1, passando a ter o valor 2. Em seguida é encontrada a linha com a instrução **ate** (`CONT > 5`), que efetuará o retorno à instrução **repita** para que seja reprocessada a seqüência de comandos, até que o valor da variável `CONT` seja maior que 5 e encerre o looping.

A seguir, é apresentado o exemplo em que não se utiliza o contador como forma de controle de execução de vezes de uma rotina em uma estrutura de repetição. Considere que será o usuário que encerrará o processamento segundo a sua vontade.

Algoritmo

1. Iniciar o programa e o modo de laço repita;
2. Ler um valor para a variável X ;

3. Efetuar a multiplicação do valor de X por 3, implicando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Solicitar do usuário se este deseja ou não continuar o programa;
6. Repetir os passos 2, 3, 4 e 5 até que a resposta do usuário seja diferente de **sim**.

Diagrama de Blocos

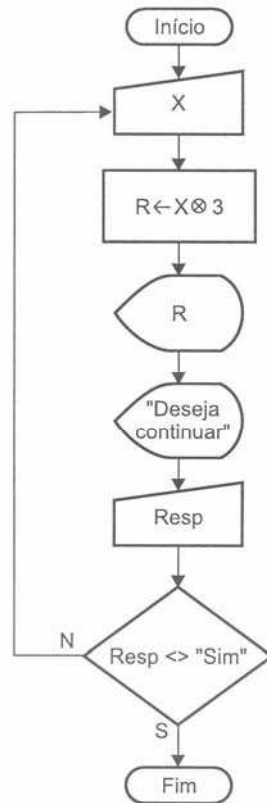


Figura 5.5 - Exemplo de um looping controlado pelo usuário.

Português Estruturado

```
algoritmo "Looping_2B"
var
  X, R: inteiro
  RESP: caracter

inicio
  RESP ← "sim"
  repita
    leia(X)
    R ← X * 3
    escreval(R)
    escreva("Deseja continuar? ")
    leia(RESP)
  ate (RESP <> "sim")
finalgoritmo
```

Veja que a variável *RESP* não precisou ser inicializada com algum valor, pois mesmo que o tivesse feito, não exerceria influência direta sobre a estrutura **repita**, uma vez que no mínimo são sempre executadas as instruções constantes no looping, pra somente no final ser verifica a condição, no caso *RESP* <> "sim".

5.2.1 - Exercícios de Entrega Obrigatória (até __/__/__) (Lista 04)

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado. Usar na resolução dos problemas apenas estruturas de repetição do tipo repita (Você deve gravar o exercício “a” como L04A, o exercício “b” como L04B, e assim por diante).

- a) Apresentar os quadrados dos números inteiros de 15 a 200.
- b) Elaborar um programa que apresente no final o somatório dos valores pares existentes na faixa de 1 até 500.
- c) Apresentar todos os números divisíveis por 4 que sejam menores que 200. Para verificar se o número é divisível por 4, efetuar dentro da malha a verificação lógica desta condição com a instrução se, perguntando se o número é divisível; sendo, mostre-o; não sendo, passe para o próximo passo. A variável que controlará o contador deve ser iniciada com o valor 1.
- d) Elaborar um programa que efetue o cálculo e no final apresente o somatório do número de grãos de trigo que se pode obter num tabuleiro de xadrez, obedecendo à seguinte regra: colocar um grão de trigo no primeiro quadro e nos quadros seguintes o dobro do quadro anterior. Ou seja, no primeiro quadro coloca-se 1 grão, no segundo quadro colocam-se 2 grãos (neste momento têm-se 3 grãos), no terceiro quadro colocam-se 4 grãos (tendo neste momento 7 grãos), no quarto colocam-se 8 grãos (tendo-se então 15 grãos) até atingir o sexagésimo quarto (64°) quadro. Utilize variáveis do tipo real como acumuladores.
- e) Elaborar um programa que efetue a leitura de 15 valores numéricos inteiros e no final apresente o total do somatório da fatorial de cada valor lido.
- f) Elaborar um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o total do somatório, a média aritmética e o total de valores lidos. O programa deve fazer as leituras dos valores enquanto o usuário estiver fornecendo valores positivos. Ou seja, o programa deve parar quando o usuário fornecer um valor negativo. Não se esqueça que o usuário pode entrar como primeiro número um número negativo, portanto, cuidado com a divisão por zero no cálculo da média.
- g) Elaborar um programa que apresente como resultado o valor do fatorial dos valores ímpares situados na faixa numérica de 1 a 10.
- h) Elaborar um programa que possibilite calcular a área total de uma residência (sala, cozinha, banheiro, quartos, área de serviço, quintal, garagem, etc.). O programa deve solicitar a entrada do nome, a largura e o comprimento de um determinado cômodo. Em seguida, deve apresentar a área do cômodo lido e também uma mensagem solicitando do usuário a confirmação de continuar calculando novos cômodos. Caso o usuário responda “NAO”, o programa deve apresentar o valor total acumulado da área residencial.
- i) Elaborar um programa que efetue a leitura de valores positivos inteiros até que um valor negativo seja informado. Ao final devem ser apresentados o maior e o menor valores informados pelo usuário.
- j) Elaborar um programa que apresente o resultado inteiro da divisão de dois números quaisquer. Para a elaboração do programa, não utilizar em hipótese alguma o conceito do operador aritmético DIV. A solução deve ser alcançada com a utilização de looping. Ou seja, o programa deve apresentar como resultado (quociente) quantas vezes o divisor cabe no dividendo.

5.3 – Repetição do Tipo: Variável de Controle

Anteriormente, foram vistas duas formas de elaborar looping. Uma usando o conceito **enquanto** e a outra usando o conceito **repita**. Foi visto também como elaborar rotinas que efetuaram a execução de um looping um determinado número de vezes com a utilização de um contador (por meio de uma variável de controle).

Porém, existe uma possibilidade de facilitar o uso de contadores finitos sem fazer uso das duas estruturas anteriores, deixando-as para utilização de loopings em que não se conhece de antemão o número de vezes que uma determinada sequência de instruções deverá ser executada. Os loopings que possuem um número

finito de execuções poderão ser processados por meio de estrutura de laços contados do tipo **para**, sendo conseguida com a utilização do conjunto de instruções **para...de...ate...passo...faca...fimpara**.

A estrutura **para...de...ate...passo...faca...fimpara** tem o seu funcionamento controlado por uma variável denominada contador. Sendo assim, poderá executar um determinado conjunto de instruções um determinado número de vezes.

Com relação à apresentação gráfica em um diagrama de blocos, existem várias formas adotadas por diversos profissionais no mercado, sendo assim, não existe um padrão de montagem do diagrama.

Diagrama de Blocos

Com relação ao diagrama de blocos, será indicada a variável a ser controlada com a implicação dos valores de início, fim e incremento, separados por vírgula. Para representar esta operação em um diagrama de blocos, será utilizado o símbolo denominado *Processamento predefinido* ou *Preparação*, representado pela figura de um hexágono.

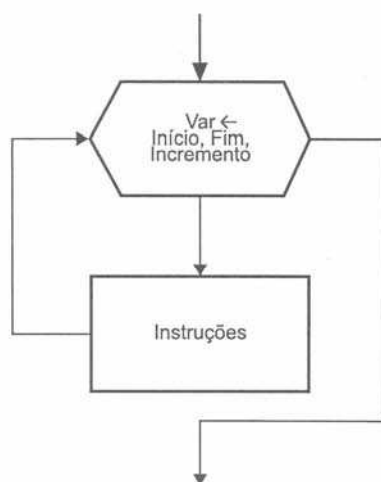


Figura 5.6 - Estrutura do símbolo da instrução **para...fimpara**.

Português Estruturado

```
para <variável> de <início> ate <fim> passo <incremento> faca  
    <instruções>  
fimpara
```

Para exemplificar, considere o problema anterior: “Pedir a leitura de um valor para a variável X, multiplicar esse valor por 3, implicando-o à variável de resposta R, e apresentar o valor obtido, repetindo esta seqüência por cinco vezes”.

Algoritmo

1. Definir um contador, variando de 1 a 5;
2. Ler um valor para a variável X;
3. Efetuar a multiplicação do valor de X por 3, implicando o resultado em R;
4. Apresentar o valor calculado, contido na variável R;
5. Repetir os passos 2, 3, 4 e 5 até que o contador seja encerrado.

Diagrama de Blocos

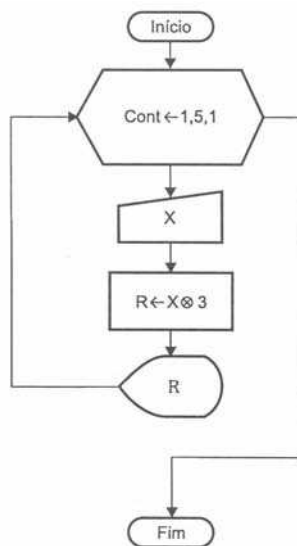


Figura 5.7 - Exemplo da utilização da estrutura para...de...até...passo...faça...fim_para.

Português Estruturado

```

algoritmo "Looping_2A"
var
  X, R: inteiro
  CONT: inteiro

inicio
  para CONT de 1 ate 5 passo 1 faca
    leia(X)
    R ← X * 3
    escreval(R)
  fimpara
fimalgoritmo
  
```

Será executado o conjunto de instruções entre **para** e a instrução **fimpara**, sendo a variável CONT (variável de controle) inicializada com valor 1 e incrementada de mais 1 por meio da instrução **passo** até o valor 5. Este tipo de estrutura de repetição poderá ser utilizado todas as vezes que houver a necessidade de repetir trechos finitos, em que se conhecem os valores inicial e final.

5.4 – Considerações entre os Tipos de Estrutura de Looping

No decorrer deste capítulo, foram apresentadas três estruturas de controle em nível de repetição: **enquanto**, **repita** e **para**, cada qual com sua característica de processamento. Dentro deste aspecto, deve-se notar que as estruturas mais versáteis são **enquanto** e **repita**, pois podem ser substituídas uma pela outra além de poderem substituir perfeitamente a estrutura **para**. Porém há de considerar-se que nem toda estrutura **enquanto** ou **repita** poderá ser substituída por uma estrutura **para**. Isto ocorre quando em uma estrutura utilizam-se condições que não envolvam o uso de variáveis de controle como contador.

5.5 – Estruturas de Controle Encadeadas

No capítulo anterior, foi discutido o fato de ocorrer o encadeamento das estruturas de decisão. Este fato pode também ocorrer com as estruturas de repetição. E neste ponto poderá ocorrer o encadeamento de um tipo de estrutura de repetição com outro tipo de estrutura de repetição. A existência destas ocorrências vai depender do problema a ser solucionado.

Devemos aqui salientar um pequeno detalhe. Nunca procure decorar estas regras, pois isto é impossível. Você precisa conhecer os comandos de entrada, processamento e saída, as estruturas de decisão e de repetição. Desta forma, conhecendo bem, você saberá utilizá-las no momento que for conveniente, pois na

resolução de um problema, ele “pedirá” a estrutura mais conveniente. E, você, conhecendo-as bem, saberá automaticamente o momento certo de utilizá-las.

Para exemplificar os tipos de aninhamento que poderão ser combinados, observe a seguir os exemplos em nível de diagrama de blocos e português estruturado. Ainda não serão apresentados exemplos da utilização prática destes aninhamentos. Mais adiante você terá contato com estes detalhes que ocorrerão de forma automática.

5.5.1 – Encadeamento de Estruturas Enquanto com Enquanto

Exemplo de encadeamento de estrutura **enquanto** com estrutura **enquanto**. A figura a seguir mostra a estrutura do desenho do diagrama de blocos.

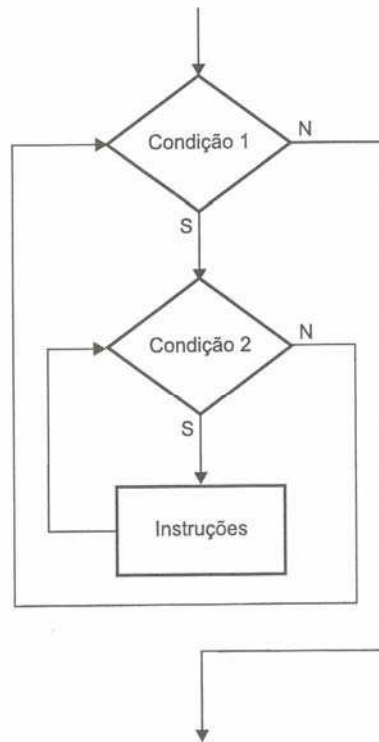
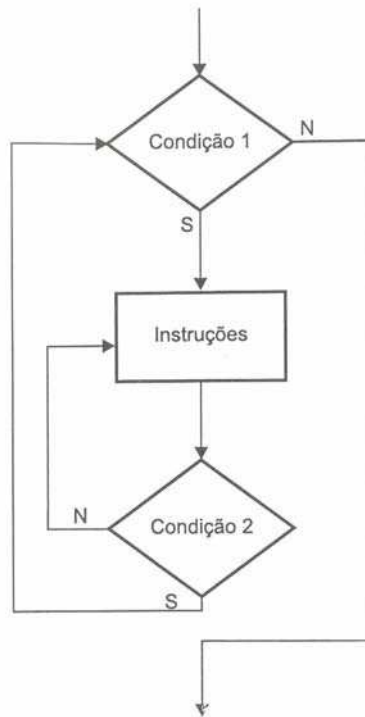


Figura 5.8 - Encadeamento de enquanto com enquanto.

Português Estruturado

```
enquanto (<condição1>) faca  
    enquanto (<condição2>) faca  
        <instruções>  
    fimenquanto  
fimenquanto
```

5.5.2 – Encadeamento de Estrutura Enquanto com Repita

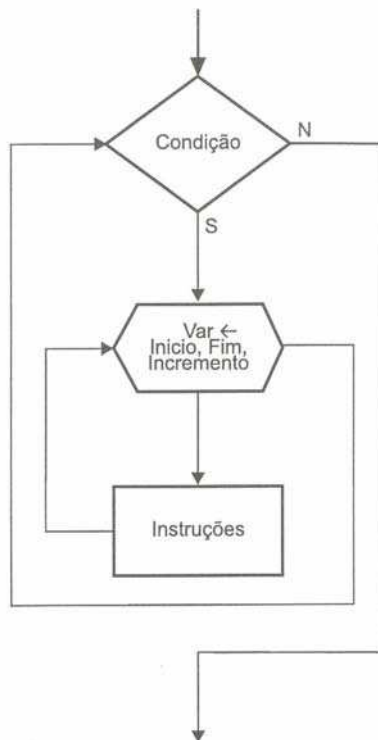


Português Estruturado

```

enquanto (<condição1>) faca
  repita
    <instruções>
  ate (<condição2>)
fimenquanto
  
```

5.5.3 – Encadeamento de Estrutura Enquanto com Para



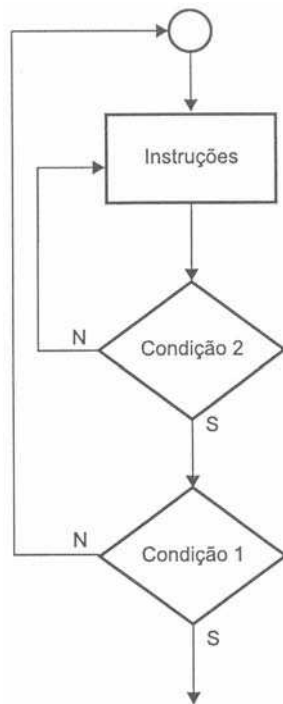
Português Estruturado

```

enquanto (<condição>) faca
  
```

```
para <var> de <início> ate <fim> passo <incremento> faca  
  <instruções>  
fimpara  
fimenquanto
```

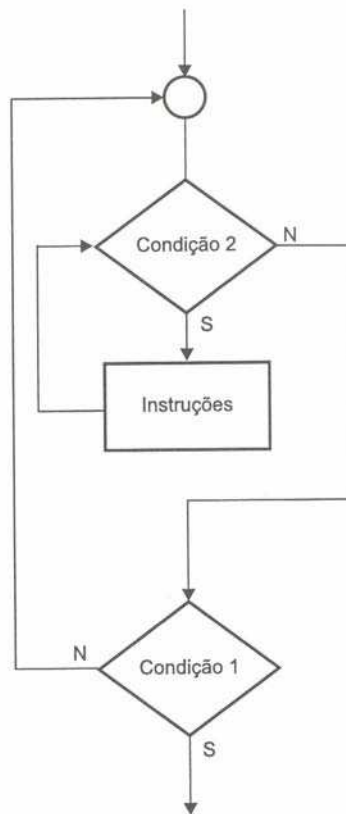
5.5.4 – Encadeamento de Estrutura Repita com Repita



Português Estruturado

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

5.5.5 – Encadeamento de Estrutura Repita com Enquanto



Português Estruturado

```
repita  
  enquanto (<condição2>) faca  
    <instruções>  
  fimenquanto  
ate (<condição1>)
```


5.5.6 – Encadeamento de Estrutura Repita com Para

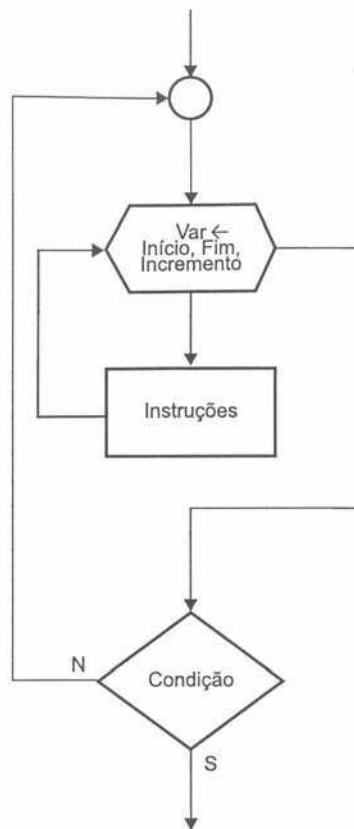


Figura 5.13 - Encadeamento de repita com para.

Português Estruturado

```
repita  
  para <var> de <início> ate <fim> passo <incremento> faca  
    <instruções>  
  fimpara  
ate (<condição>)
```

5.5.7 – Encadeamento de Estrutura Para com Para

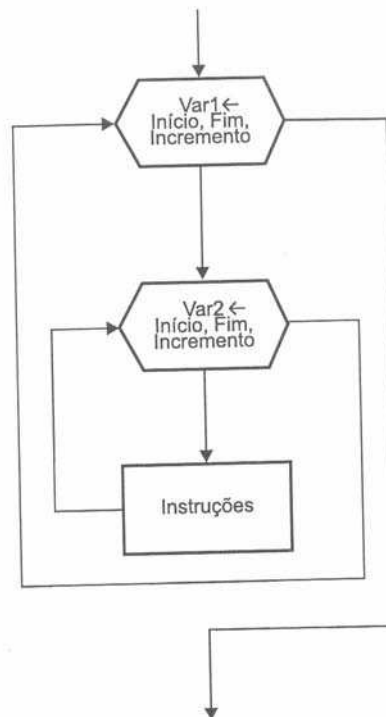


Figura 5.14 - Encadeamento de para com para.

Português Estruturado

```
para <var1> de <início> ate <fim> passo <incremento> faca  
  para <var2> de <início> ate <fim> passo <incremento> faca  
    <instruções>  
  fimpara  
fimpara
```

5.5.8 – Encadeamento de Estrutura Para com Enquanto

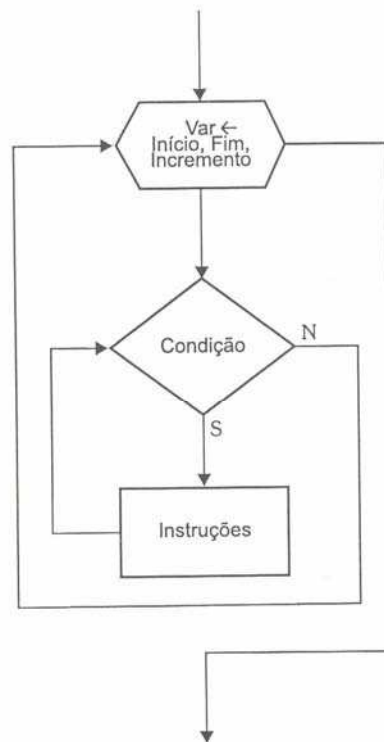


Figura 5.15 - Encadeamento de para com enquanto.

Português Estruturado

```
para <variavel> de <início> ate <fim> passo <incremento> faca  
  enquanto (<condição>) faca  
    <instruções>  
  fimenquanto  
fimpara
```

5.5.9 – Encadeamento de Estrutura Para com Repita

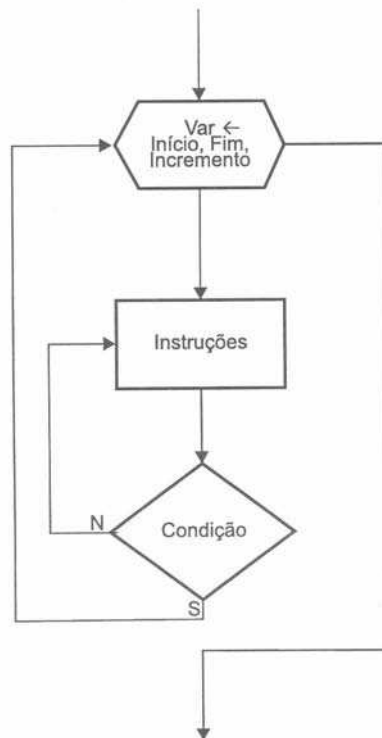


Figura 5.16 - Encadeamento de para com repita.

Português Estruturado

```
para <variavel> de <início> ate <fim> passo <incremento> faca  
  repita  
    <instruções>  
  ate (condição)  
fimpara
```

5.6 – Exercício de Aprendizagem

A seguir, são apresentados alguns exemplos da utilização das estruturas de repetição. Considere o seguinte problema: “Elaborar o algoritmo, diagrama de blocos e codificação em português estruturado de um programa que efetue o cálculo do fatorial do número 5, 5!”. Desta forma, temos que $5! = 5 \times 4 \times 3 \times 2 \times 1$ ou seja $5! = 120$.

Fatorial é o produto dos números naturais desde 1 até o inteiro n. Sendo assim, o cálculo de um fatorial é conseguido pela multiplicação sucessiva do número de termos. No caso do cálculo de uma fatorial de número 5, este é o número de termos a ser utilizado. Desta forma, o programa deverá executar as multiplicações sucessivamente e acumulá-las a fim de possuir o valor 120 após 5 passos. O número de passos deverá ser controlado por um contador.

Algoritmo

1. Inicializar as variáveis FATORIAL e CONTADOR com 1;
2. Multiplicar sucessivamente a variável FATORIAL pela variável CONTADOR;
3. Incrementar 1 à variável CONTADOR, efetuando o controle até 5;
4. Apresentar ao final o valor obtido.

Pelo fato de ter que efetuar o cálculo de uma fatorial de 5 (5!), isto implica que o contador deverá variar de 1 a 5, e por este motivo deverá ser a variável CONTADOR inicializada com valor 1. Pelo fato de a variável FATORIAL possuir ao final o resultado do cálculo da fatorial pretendida, esta deverá ser inicializada com valor 1. Se ela for inicializada com zero, não existirá resultado final, pois qualquer valor multiplicado por zero resulta zero.

Multiplicar sucessivamente implica em efetuar a multiplicação da variável CONTADOR com o valor atual da variável FATORIAL a cada passo controlado pelo looping. No caso, por cinco vezes. Abaixo é indicado esta ocorrência na linha 2 do algoritmo.

1. Inicializar as variáveis FATORIAL e CONTADOR com 1;
2. Repetir a execução dos passos 3 e 4 por cinco vezes;
3. $FATORIAL \leftarrow FATORIAL * CONTADOR$;
4. Incrementar 1 à variável CONTADOR;
5. Apresentar ao final o valor obtido.

A seguir a resolução do problema do cálculo de fatorial, utilizando as estruturas de repetição, sendo apresentados os diagramas de blocos e a codificação em português estruturado de cada estrutura.

Exemplo 1 – Estrutura de repetição: Enquanto

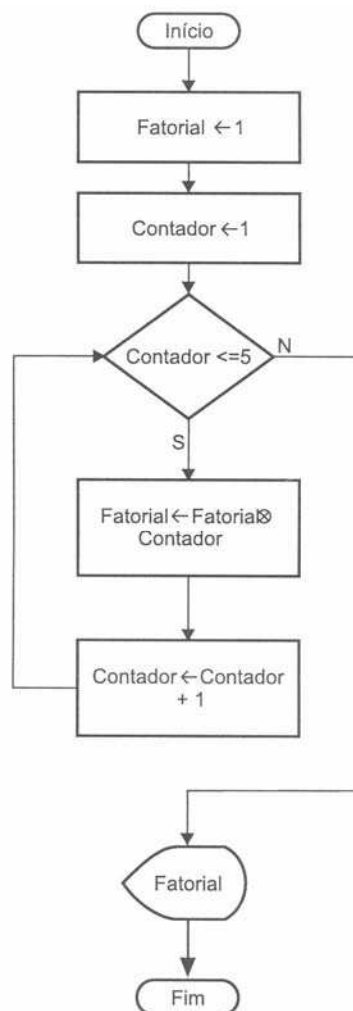


Figura 5.17 - Fatorial com estrutura enquanto.

Observe dentro do looping a indicação de dois contadores, sendo que o primeiro funciona como um acumulador, pois ele terá no final o valor do resultado do fatorial, e o segundo sendo utilizado para controlar a execução do looping a ser a base para o cálculo do acumulador.

Logo no início do diagrama de blocos, as variáveis CONTADOR e FATORIAL são igualadas em 1. Na primeira passagem dentro do looping, a variável FATORIAL é implicada pelo seu valor atual, no caso 1, multiplicado pelo valor da variável CONTADOR também com valor 1 que resulta 1. Em seguida a variável

CONTADOR é incrementada por mais 1, tendo agora o valor 2. Como 2 é menor ou igual a cinco, ocorre um novo cálculo. Desta vez a variável FATORIAL que possui o valor 1 é multiplicada pela variável CONTADOR que possui o valor 2, resultando 2 para o FATORIAL. Daí a variável CONTADOR é incrementada de mais 1, tendo agora o valor 3. Desta forma, serão executados os outros cálculos até que a condição se torne falsa e seja então apresentado o valor 120. Veja abaixo, a tabela com os valores das variáveis antes e durante a execução do looping:

CONTADOR	FATORIAL	FATORIAL <- FATORIAL*CONTADOR	Comentários
1	1	1	Valor inicial das variáveis
2	1	2	Cálculo do fatorial com contador em 2
3	2	6	Cálculo do fatorial com contador em 3
4	6	24	Cálculo do fatorial com contador em 4
5	24	120	Cálculo do fatorial com contador em 5

Perceba que quando a variável CONTADOR está com valor 5, a variável FATORIAL está com o valor 120. Neste ponto, o looping é encerrado e é apresentado o valor da variável FATORIAL.

Português Estruturado

```

algoritmo "Fatorial_A"
var
    CONTADOR : inteiro
    FATORIAL : inteiro

inicio
    FATORIAL <- 1
    CONTADOR <- 1
    enquanto (CONTADOR <=5) faca
        FATORIAL <- FATORIAL * CONTADOR
        CONTADOR <- CONTADOR + 1
    fimenquanto
    escreva("Fatorial de 5 é = ", FATORIAL)
fimalgoritmo

```

No código acima, estão sendo apresentadas pela instrução **escreva()** duas informações, sendo uma mensagem e uma variável. Note que isto é feito utilizando uma vírgula para separa as duas informações.

Exemplo 2 – Estrutura de repetição: Repita

Resolução do problema fazendo uso da estrutura de repetição **repita**.

Diagrama de Blocos

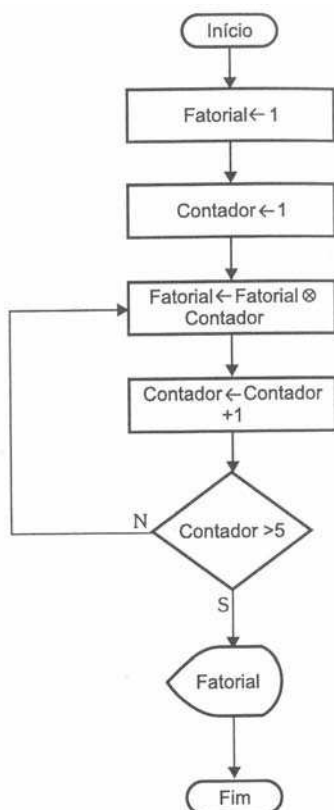


Figura 5.18 - Fatorial com estrutura *repita*.

Português Estruturado

```
algoritmo "Fatorial_B"
var
    CONTADOR : inteiro
    FATORIAL : inteiro

inicio
    FATORIAL <- 1
    CONTADOR <- 1
    repita
        FATORIAL <- FATORIAL * CONTADOR
        CONTADOR <- CONTADOR + 1
    ate (CONTADOR > 5)
    escreva("Fatorial de 5 é = ", FATORIAL)
finalgoritmo
```

Exemplo 3 – Estrutura de repetição: Para

Resolução do problema fazendo uso da estrutura de repetição **para**.

Diagrama de Blocos

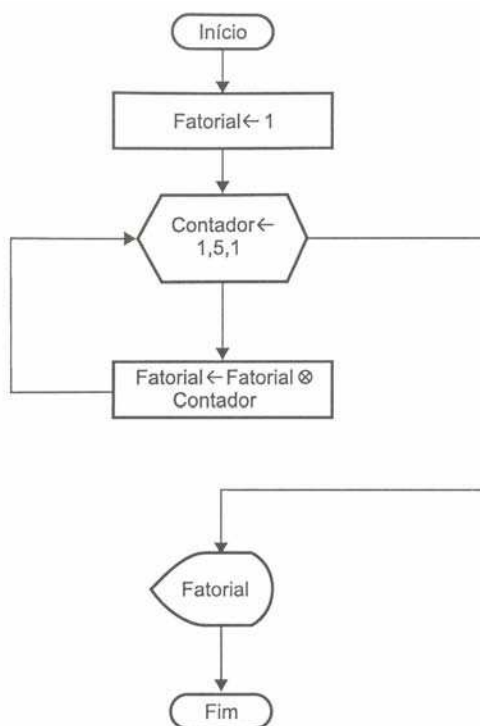


Figura 5.19 - Fatorial com estrutura para.

Português Estruturado

```

algoritmo "Fatorial_B"
var
  CONTADOR: inteiro
  FATORIAL: inteiro

inicio
  FATORIAL ← 1
  para CONTADOR de 1 ate 5 passo 1 faca
    FATORIAL ← FATORIAL * CONTADOR
  fimpara
  escreva("Fatorial de 5 é igual a ", FATORIAL)
fimalgoritmo
  
```

Exemplo 4

Observe que o algoritmo apresentado anteriormente para a resolução do problema de fatorial soluciona o cálculo apenas para o fatorial de 5. Seria melhor possuir a solução aberta para que um programa calcule o fatorial de um número qualquer, e que pudesse calcular outros fatoriais até que o usuário não mais deseje utilizar o programa. Sendo assim, o programa deverá pedir ao usuário a sua continuidade ou não.

No código já existente, deverão ser criadas duas variáveis, sendo uma para a confirmação da continuidade da execução do programa e outra para determinar o cálculo do valor do fatorial. O exemplo seguinte faz uso do encadeamento de estruturas de repetição.

1. Inicializar as variáveis FATORIAL e CONTADOR com 1;
2. Declarar as variáveis RESP (resposta) para confirmação e N para receber o limite de valor para o cálculo do fatorial;
3. Enquanto RESP do usuário for sim, executar os passos 3, 4, 5, 6 e 7;
4. Repetir a execução dos passos 4 e 5 por N vezes;
5. FATORIAL ← FATORIAL * CONTADOR;
6. Incrementar 1 à variável CONTADOR;
7. Apresentar ao final o valor obtido.

Diagrama de Blocos

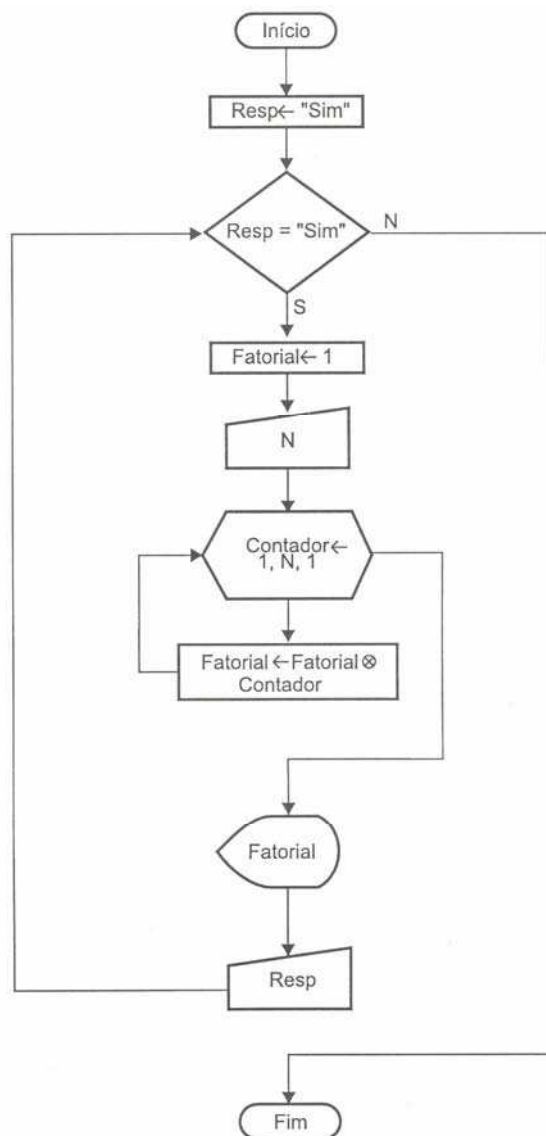


Figura 5.20 - Implementação do número de vezes e cálculo de qualquer fatorial.

Português Estruturado

algoritmo "Fatorial_D"

var

CONTADOR: inteiro

FATORIAL: inteiro

RESP: literal

N: inteiro

inicio

RESP ← "sim"

enquanto (RESP = "sim") **faca**

FATORIAL ← 1

escreva("Fatorial de que numero: ")

leia(N)

para CONTADOR **de** 1 **ate** N **passo** 1 **faca**

FATORIAL ← FATORIAL * CONTADOR

fimpara

escreval("Fatorial de ", N, " é igual a ", FATORIAL)

escreva("Deseja continua?")

leia(RESP)

fimenquanto

fimalgoritmo

5.6.1 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 05)

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado. Usar na resolução dos problemas apenas estruturas de repetição do tipo para (Você deve gravar o exercício “a” como L05A, o exercício “b” como L05B, e assim por diante).

- a) Apresentar os quadrados dos números inteiros de 15 a 200.
- b) Apresentar os resultados de uma tabuada de multiplicar (de 1 até 10) de um número qualquer.
- c) Apresentar o total da soma obtida dos cem primeiros números inteiros (1+2+3+4+...+98+99+100).
- d) Elaborar um programa que apresente no final o somatório dos valores pares existentes na faixa de 1 até 500.
- e) Apresentar todos os valores numéricos inteiros ímpares situados na faixa de 0 a 20. Para verificar se o número é ímpar, efetuar dentro da malha a verificação lógica desta condição com a instrução se, perguntando se o número é ímpar; sendo, mostre-o; não sendo, passe para o próximo passo.
- f) Apresentar todos os números divisíveis por 4 que sejam menores que 200. Para verificar se o número é divisível por 4, efetuar dentro da malha a verificação lógica desta condição com a instrução se, perguntando se o número é divisível; sendo, mostre-o; não sendo, passe para o próximo passo. A variável que controlará o contador deve ser iniciada com o valor 1.
- g) Apresentar os resultados das potências de 3, variando do expoente 0 até o expoente 15. Deve ser considerado que qualquer número elevado a zero é 1, e elevado a 1 é ele próprio. Observe que neste exercício não pode ser utilizado o operador de exponenciação do português (^).
- h) Elaborar um programa que apresente como resultado o valor de uma potência de uma base qualquer elevada a um expoente qualquer, ou seja, de B^E , em que B é o valor da base e E o valor do expoente. Observe que neste exercício não pode ser utilizado o operador de exponenciação do português (^).
- i) Escreva um programa que apresente a série de Fibonacci até o décimo quinto termo. A série de Fibonacci é formada pela seqüência: 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., etc. Esta série se caracteriza pela soma de um termo atual com o seu anterior subsequente, para que seja formado o próximo valor da seqüência. Portanto começando com os números 1, 1 o próximo termo é 1+1=2, o próximo é 1+2=3, o próximo é 2+3=5, o próximo 3+5=8, etc.
- j) Elaborar um programa que apresente os valores de conversão de graus Celsius em Fahrenheit, de 10 em 10 graus, iniciando a contagem em 10 graus Celsius e finalizando em 100 graus Celsius. O programa deve apresentar os valores das duas temperaturas. A fórmula de conversão é $F = \frac{9C + 160}{5}$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
- k) Elaborar um programa que apresente como resultado o valor do fatorial dos valores ímpares situados na faixa numérica de 1 a 10.

6 – Estrutura de Dados Homogêneas I

Durante os pontos estudados nos capítulos 4 e 5, percebeu-se que o poder de programação tornou-se maior, antes limitado somente ao que foi estudado no capítulo 3. Porém, tendo o domínio das técnicas anteriores, ainda correr-se-á o risco de não conseguir resolver alguns tipos de problema, pois foram trabalhadas até aqui apenas variáveis simples que somente armazenam um valor por vez.

Neste capítulo, será apresentada uma técnica de programação que permitirá trabalhar com o agrupamento de várias informações dentro de uma mesma variável. Vale salientar que esse agrupamento ocorrerá obedecendo sempre ao mesmo tipo de dado, e por esta razão é chamado de estrutura de dados homogênea. Agrupamentos de tipos de dados diferentes serão estudados mais adiante quando forem abordadas as estruturas de dados heterogêneas.

A utilização deste tipo de estrutura de dados recebe diversos nomes, como: variáveis indexadas, variáveis compostas, variáveis subscriptas, arranjos, vetores, matrizes, tabelas em memória ou arrays (do inglês). São vários os nomes encontrados na literatura voltada para o estudo de técnicas de programação que envolvem a utilização das estruturas homogêneas de dados. Por nós, serão definidas como matrizes.

As matrizes (tabelas em memória principal) são tipos de dados que podem ser “construídos” à medida que se fazem necessários, pois não é sempre que os tipos básicos (real, inteiro, caractere e lógico) e/ou variáveis simples são suficientes para representar a estrutura de dados utilizada em um programa.

6.1 – Matrizes de uma Dimensão ou Vetores

Este tipo de estrutura em particular é também denominado por alguns profissionais como matrizes unidimensionais. 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 constituída por constantes inteiras e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar as variáveis simples.

Para ter uma idéia de como utilizar matrizes em um determinada situação, considere o seguinte problema: “Calcular a média geral de uma turma de 8 alunos. A média a ser obtida deve ser a média geral das médias de cada aluno obtida durante o ano letivo”. Desta forma será necessário somar todas as média e dividi-las por 8. A tabela seguinte apresenta o número de alunos, suas notas bimestrais e respectivas médias anuais. É da média de cada aluno que será efetuado o cálculo da média da turma.

Aluno	Nota 1	Nota 2	Nota 3	Nota 4	Média
1	4.0	6.0	5.0	3.0	4.5
2	6.0	7.0	5.0	8.0	6.5
3	9.0	8.0	9.0	6.0	8.0
4	3.0	5.0	4.0	2.0	3.5
5	4.0	6.0	6.0	8.0	6.0
6	7.0	7.0	7.0	7.0	7.0
7	8.0	7.0	6.0	5.0	6.5
8	6.0	7.0	2.0	9.0	6.0

Agora basta escrever um programa para efetuar o cálculo das 8 médias de cada aluno. Para representar a média do primeiro aluno será utilizada a variável MD1, para o segundo MD2 e assim por diante. Então se tem:

```
MD1 = 4.5
MD2 = 6.5
MD3 = 8.0
MD4 = 3.5
MD5 = 6.0
```

```
MD6 = 7.0
MD7 = 6.5
MD8 = 6.0
```

Com o conhecimento adquirido até este momento, seria então elaborado um programa que efetuaria a leitura de cada nota, a soma delas e a divisão do valor da soma por 8, obtendo-se desta forma a média conforme exemplo abaixo em português estruturado:

```
algoritmo "Media_Turma"
var
  MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8: real
  SOMA, MEDIA: real

inicio
  SOMA <- 0
  leia(MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8)
  SOMA <- MD1 + MD2 + MD3 + MD4 + MD5 + MD6 + MD7 + MD8
  MEDIA <- SOMA / 8
  escreva(MEDIA)
finalgoritmo
```

Perceba que para receber a média foram utilizadas oito variáveis. Com a técnica de matrizes poderia ter sido utilizada apenas uma variável com a capacidade de armazenar oito valores.

6. 2 – Operações Básicas com Matrizes do Tipo Vetor

Uma matriz de uma dimensão ou vetor será, neste trabalho, representada por seu nome e seu tamanho (dimensão) entre colchetes. Desta forma seria uma matriz MD[1..8], sendo seu nome MD, possuindo um tamanho de 1 a 8. Isto significa que poderão ser armazenados em MD até oito elementos. Perceba que na utilização de variáveis simples existe uma regra: uma variável somente pode conter um valor por vez. No caso das matrizes, poderão armazenar mais de um valor por vez, pois são dimensionadas exatamente para este fim. Desta forma poder-se-á manipular uma quantidade maior de informação com pouco trabalho de processamento. Deve-se apenas considerar que com relação à manipulação dos elementos de uma matriz, eles ocorrerão de forma individualizada, pois não é possível efetuar a manipulação de todos os elementos do conjunto ao mesmo tempo.

No caso do exemplo do cálculo da média dos 8 alunos, ter-se-ia então uma única variável indexada (a matriz) contendo todos os valores das 8 notas. isto seria representado da seguinte forma:

```
MD[1] = 4.5
MD[2] = 6.5
MD[3] = 8.0
MD[4] = 3.5
MD[5] = 6.0
MD[6] = 7.0
MD[7] = 6.5
MD[8] = 6.0
```

Observe que o nome é um só. O que muda é a informação indicada dentro dos colchetes. A esta informação dá-se o nome de **índice**, sendo este o endereço em que o elemento está armazenado. É necessário que fique bem claro que elemento é o conteúdo da matriz, neste caso os valores das notas. No caso de MD[1] = 4.5, o número 1 é o índice; o endereço cujo elemento é 4.5 está armazenado.

6.2.1 – Atribuição de uma Matriz

Anteriormente, foram utilizadas várias instruções em português estruturado para poder definir e montar um programa. No caso da utilização de matrizes, será definida a instrução **vetor** que indicará em português estruturado a utilização de uma matriz, tendo como sintaxe: **VARIÁVEL: VETOR[<dimensão>] de <tipo de dado>**, sendo que <dimensão> será a indicação dos valores inicial e final do tamanho do vetor e <tipo de dado> se o vetor em questão irá utilizar valores reais, inteiros, lógicos ou caracteres.

6.2.2 – Leitura dos Dados de uma Matriz

A leitura de uma matriz é processada passo a passo, um elemento por vez. A instrução de leitura é **leia()** seguida da variável mais o índice. A seguir, são apresentados diagramas de blocos e codificação em português estruturado da leitura das notas dos 8 alunos, cálculo da média e a sua apresentação.

Diagrama de Blocos

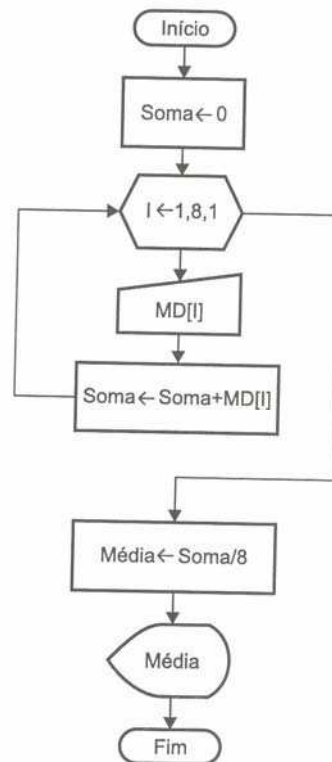


Figura 6.1 - Diagrama de blocos para leitura dos elementos de uma matriz tipo vetor.

Português Estruturado

```

algoritmo "Media_turma"
var
    MD: vetor[1..8] de real
    SOMA, MEDIA: real
    I: inteiro

inicio
    SOMA ← 0
    para I de 1 ate 8 passo 1 faca
        leia(MD[I])
        SOMA ← SOMA + MD[I]
    fimpara
    MEDIA ← SOMA / 8
    escreva(MEDIA)
fimalgoritmo
    
```

Veja que o programa ficou mais compacto, além de possibilitar uma mobilidade maior, pois se houver a necessidade de efetuar o cálculo para um número maior de alunos, basta dimensionar a matriz e mudar o valor final da instrução **para**. Observe que no exemplo anterior, a leitura é processada uma por vez. Desta forma, a matriz é controlada pelo número do índice que faz com que cada entrada aconteça em uma posição diferente da outra. Assim sendo, a matriz passa a ter todas as notas. A tabela seguinte, mostra como ficarão os valores armazenados na matriz.

Matriz: MD	
Índice	Elemento
1	4.5
2	6.5
3	8.0
4	3.5
5	6.0
6	7.0
7	6.5
8	6.0

Tenha cuidado para não confundir o índice com o elemento. Índice é o endereço de alocação de uma unidade da matriz, enquanto elemento é o conteúdo armazenado em um determinado endereço.

6.2.3 – Escrita dos Dados de uma Matriz

O processo de escrita de uma matriz é bastante parecido com o processo de leitura de seus elementos. Para esta ocorrência deverá ser utilizada a instrução **escreva()** seguida da indicação da variável e seu índice. Supondo que após a leitura das 8 notas, houvesse a necessidade de apresentá-las antes da apresentação do valor da média.

Diagrama de Blocos

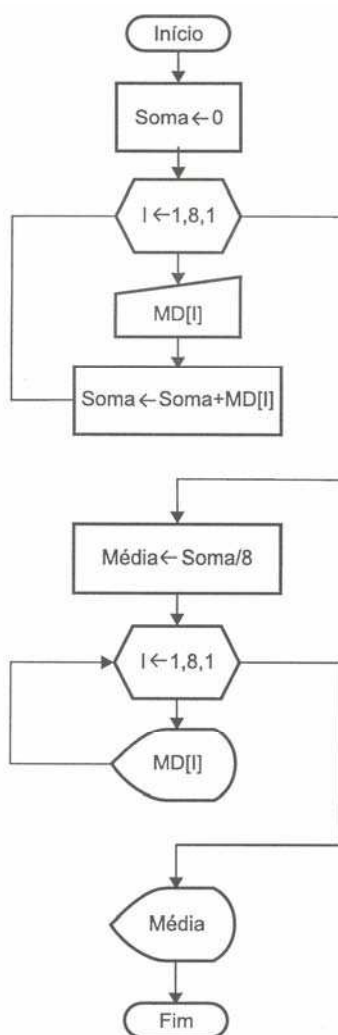


Figura 6.2 - Diagrama de bloco para escrita dos elementos de uma matriz tipo vetor.

Português Estruturado

```
algoritmo "Media_turma"
var
  MD: vetor[1..8] de real
  SOMA, MEDIA: real
  i: inteiro

inicio
  SOMA <- 0
  para i de 1 ate 8 passo 1 faca
    leia(MD[i])
    SOMA <- SOMA + MD[i]
  fimpara
  para i de 1 ate 8 passo 1 faca
    escreval(MD[i])
  fimpara
  MEDIA <- SOMA / 8
  escreva(MEDIA)
fimalgoritmo
```

6.3 – Exercício de Aprendizagem

1º Exemplo

Desenvolver um programa que efetue a leitura de dez elementos de uma matriz A tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: se o valor do índice for par, o valor deverá ser multiplicado por 5, sendo ímpar, deverá ser somando com 5. Ao final mostrar o conteúdo da matriz B.

Algoritmo

Este exemplo de resolução estará mostrando como fazer o tratamento da condição do índice.

1. Iniciar o contador de índice, variável I com 1 em um contador até 10;
2. Ler os 10 valores, um a um;
3. Verificar se o índice é par se sim multiplica por 5, se não soma 5. Criar a matriz B;
4. Apresentar os conteúdos das duas matrizes.

Diagrama de Blocos

Deverá ser perguntado se o valor do índice I em um determinado momento é para (ele será par quando dividido por 2 obtiver resto igual a zero). Sendo a condição verdadeira, será implicada na matriz B[i] a multiplicação do elemento da matriz A[i] por 5. Caso o valor do índice I seja ímpar, será implicada na matriz B[i] a soma do elemento da matriz A[i] por 5.

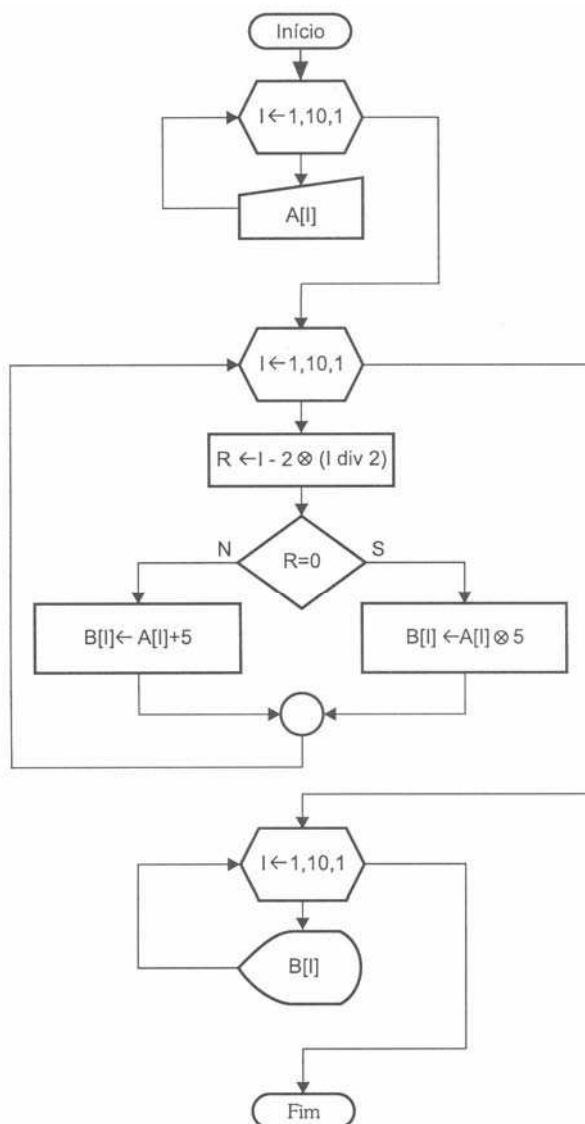


Figura 6.3 - Diagrama de blocos para o 1º exemplo.

Português Estruturado

```

algoritmo "Indice_par_ou_impar"
var
  A, B: vetor[1..10] de real
  i, r: inteiro

inicio
  para i de 1 ate 10 passo 1 faca
    leia(A[i])
  fimpara
  para i de 1 ate 10 passo 1 faca
    r ← i - 2*(i div 2)
    se (r = 0) entao
      B[i] ← A[i] * 5
    senao
      B[i] ← A[i] + 5
    fimse
  fimpara
  para i de 1 ate 10 passo 1 faca
    escreval(B[i])
  fimpara
fimalgoritmo
  
```


2º Exemplo

Desenvolver um programa que efetue a leitura de cinco elementos de uma matriz A do tipo vetor. No final, apresente o total da soma de todos os elementos que sejam ímpares.

Algoritmo

Perceba que em relação ao primeiro exemplo, este apresenta uma diferença: o primeiro pedia para verificar se o índice era par ou ímpar. Neste exemplo, está sendo solicitado que analise a condição do elemento e não do índice. Já foi alertado anteriormente para se tomar cuidado para não confundir elemento com índice. Veja a solução.

1. Iniciar o contador de índice, variável i como 1 em um contador até 5;
2. Ler os 5 valores, um a um;;
3. Verificar se o elemento é ímpar; se sim efetuar a soma dos elementos;
4. Apresentar o total somado de todos os elementos ímpares da matriz.

diagrama de Blocos

Observe que quando se faz menção ao índice indica-se a variável que controla o contador de índice, e no caso do exemplo anterior, a variável i. Quando se faz menção ao elemento, indica-se: A[i], pois desta forma está se pegando o valor armazenado e não a sua posição de endereço.

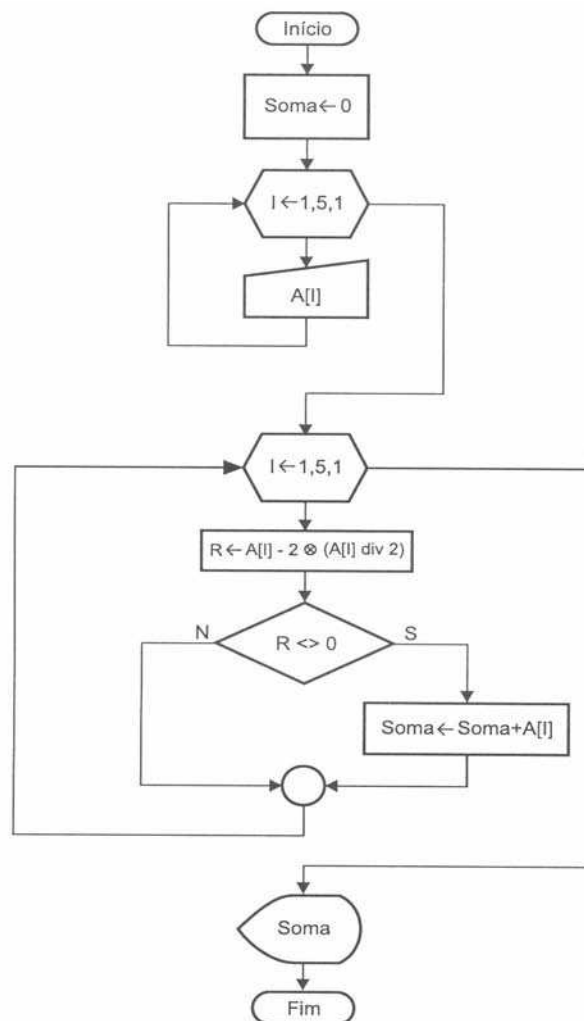


Figura 6.4 - Diagrama de blocos para o 2º exemplo.

Português Estruturado

```

algoritmo "Elemento_impair"
var
  A: vetor[1..5] de inteiro
  i, R, SOMA: inteiro

inicio
  soma <- 0
  para i de 1 ate 5 passo 1 faca
    leia(A[i])
  fimpara
  para i de 1 ate 5 passo 1 faca
    R <- A[i] - 2*(A[i] div 2)
    se (R <> 0) entao
      SOMA <- SOMA + A[i]
    fimse
  fimpara
  escreva(SOMA)
fimalgoritmo

```

6.4 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 06)

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado (Você deve gravar o exercício “a” como L06A, o exercício “b” como L06B, e assim por diante).

- a) Ler 10 elementos de uma matriz tipo vetor e apresentá-los.
- b) Ler 8 elementos em uma matriz A tipo vetor. Construir uma matriz B de mesma dimensão com os elementos da matriz A multiplicados por 3. O elemento B[i] deverá ser implicado pelo elemento A[i]*3, o elemento B[2] implicado pelo elemento A[2]*3 e assim por diante, até 8. Apresentar o vetor B.
- c) Ler duas matrizes A e B do tipo vetor com 20 elementos. Construir uma matriz C, onde cada elemento de C é a subtração do elemento correspondente de A com B. Apresentar a matriz C.
- d) Ler 15 elementos de uma matriz tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguintes lei de formação: “Todo elemento de B deverá ser o quadrado do elemento de A correspondente”. Apresentar as matrizes A e B.
- e) Ler duas matrizes A e B do tipo vetor com 15 elementos cada. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deverá ter o dobro de elementos, ou seja, 30. Apresentar a matriz C.
- f) Ler duas matrizes do tipo vetor, sendo A com 20 elementos e B com 30 elementos. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deverá ter a capacidade de armazenar 50 elementos. Apresentar a matriz C.
- g) Ler 20 elementos de uma matriz A tipo vetor e construir uma matriz B de mesma dimensão com os mesmo elementos da matriz A, sendo que deverão estar invertidos. Ou seja, o primeiro elemento de A passa a ser o último de B, o segundo elemento de A passa a ser o penúltimo elemento de B e assim por diante. Apresentar as matrizes A e B lado a lado.
- h) Ler três matrizes (A, B e C) de uma dimensão com 5 elementos cada. Construir uma matriz D, sendo esta a junção das três outras matrizes. Desta forma D deverá ter o triplo de elementos, ou seja, 15. Apresentar os elementos da matriz D.
- i) Ler 15 elementos reais para uma matriz A de uma dimensão do tipo vetor. Construir uma matriz B de mesmo tipo e dimensão, observando a seguinte lei de formação: “Todo elemento da matriz A que possuir índice par deverá ter seu elemento dividido por 2; caso contrário, o elemento da matriz A deverá ser multiplicado por 1.5”. Apresentar as matrizes A e B lado a lado.
- j) Ler duas matrizes A e B de uma dimensão com 6 elementos. A matriz A deverá aceitar apenas a

entrada de valores pares, enquanto a matriz B deverá aceitar apenas a entrada de valores ímpares. A entrada das matrizes deverá ser validada pelo programa e não pelo usuário. Construir uma matriz C de forma que a matriz C seja a junção das matrizes A e B, de modo que a matriz C contenha 12 elementos. Apresentar a matriz C.

6.5 - Exercícios de Extras

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado.

- a) Ler duas matrizes A e B de uma dimensão com 12 elementos. A matriz A deverá aceitar apenas a entrada de valores que sejam divisíveis por 2 ou 3, enquanto a matriz B deverá aceitar apenas a entrada de valores que não sejam múltiplos de 5. A entrada das matrizes deverá ser validada pelo programa e não pelo usuário. Construir uma matriz C de forma que a matriz C seja a junção das matrizes A e B, e de forma que a matriz C contenha 24 elementos. Apresentar a matriz C.
- b) Ler uma matriz A do tipo vetor com 15 elementos. Construir uma matriz B de mesmo tipo, sendo que cada elemento da matriz B seja o fatorial do elemento correspondente da matriz A. Apresentar as matrizes A e B.
- c) Ler 5 elementos (valores reais) para temperaturas em graus Celsius em uma matriz A de uma dimensão do tipo vetor. Construir uma matriz B de mesmo tipo e dimensão, em que cada elemento da matriz B deverá ser a conversão da temperatura em graus Fahrenheit do elemento correspondente da matriz A. Apresentar as matrizes A e B lado a lado. A fórmula de conversão é
$$F = \frac{9C + 160}{5}$$
, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
- d) Solicitar 20 nomes quaisquer que serão armazenados em uma matriz do tipo vetor. Ordene este vetor em ordem alfabética. Exiba na tela os 20 nomes na ordem. Em seguida solicite um nome para pesquisa. Caso o nome fornecido exista no vetor, informar a sua localização (seu índice).

7 – Aplicações Práticas do Uso de Matrizes do Tipo Vetor

A utilização de matrizes em programação é bastante ampla. Podem ser utilizadas em diversas situações, tornando bastante útil e versátil esta técnica de programação. Para ter uma idéia, considere o seguinte problema: “Criar um programa que efetue a leitura dos nomes de 20 pessoas e em seguida apresente-os na mesma ordem em que foram informados”.

Algoritmo

1. Definir a variável i do tipo inteiro para controlar a malha de repetição;
2. Definir a matriz NOME do tipo literal para 20 elementos;
3. Iniciar o programa, fazendo a leitura dos 20 nomes;
4. Apresentar após a leitura, os 20 nomes.

Diagrama de Blocos

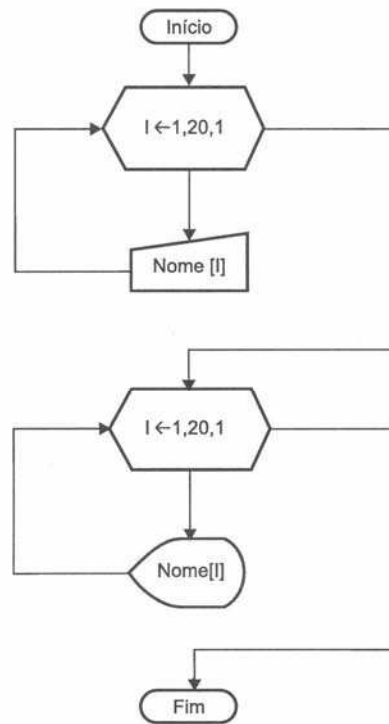


Figura 7.1 - Diagrama de blocos para ler e escrever 20 nomes de pessoas.

Português Estruturado

```

algoritmo "Lista_Nome"
var
    NOME: vetor[1..20] de literal
    i: inteiro

inicio
    para i de 1 ate 20 faca
        leia(NOME[i])
    fimpara
    para i de 1 ate 20 faca
        escreval(NOME[i])
    fimpara
fimalgoritmo
    
```

O programa anterior executa a leitura e a escrita de 20 nomes. Perceba que será apresentada a relação dos nomes da mesma forma em que foi informada para o trecho de entrada.

7.1 – Classificação dos Elementos de uma Matriz

Tendo construído o programa de entrada e saída dos 20 nomes na matriz, seria bastante útil que, antes de apresentá-los, o programa efetuasse o processamento da classificação alfabética, apresentando os nomes em ordem, independentemente daquela em que foram informados, facilitando desta forma a localização de algum nome, quando for efetuada uma pesquisa visual.

Existem vários métodos para obter a ordenação de elementos de uma matriz. Aqui será apresentado um método bastante simples de ordenação que consiste na comparação de cada elemento com todos os elementos subsequentes. Sendo o elemento comparado menor para ordenação decrescente, ou maior para ordenação crescente que o atual, ele será trocado de posição com o outro. A ordenação considerada é alfabética, devendo essa ser crescente, ou seja, de A até Z.

A seguir, é apresentado o programa completo com a parte de entrada de dados na matriz, o processamento da ordenação e a apresentação dos dados ordenados. Mas somente o trecho de ordenação será comentado, uma vez que a entrada e a saída dos elementos já foram comentados anteriormente.

Algoritmo

1. Definir a variável *i* do tipo inteira para controlar a malha de repetição;
2. Definir a matriz NOME do tipo literal para 20 elementos;
3. Iniciar o programa, fazendo a leitura dos 20 nomes;
4. Colocar em ordem crescente os elementos da matriz;
5. Apresentar após a leitura, os 20 nomes ordenados.

É importante que se estabeleça adequadamente o quarto passo do algoritmo, ou seja, como fazer para colocar os elementos em ordem crescente. Foi informado que basta comparar um elemento com os demais subseqüentes. Isto é verdade, mas necessita ser feito com alguma cautela.

Imagine um problema um pouco menor: “Colocar em ordem crescente 5 valores numéricos armazenados numa matriz A e apresentá-los”. Considere para esta explicação os valores apresentados na tabela a seguir:

Matriz A	
Índice: <i>i</i>	Elemento: <i>A[i]</i>
1	9
2	8
3	7
4	5
5	3

Os valores estão armazenados na ordem: 9, 8, 7, 5 e 3 e deverão ser apresentados na ordem 3, 5, 7, 8 e 9.

Para efetuar o processo de troca, é necessário aplicar o método de propriedade distributiva. Sendo assim, o elemento que estiver em *A[1]* deverá ser comparado com os elementos que estiverem em *A[2]*, *A[3]*, *A[4]* e *A[5]*. Depois, o elemento que estiver em *A[2]* não necessita ser comparado com o elemento que estiver em *A[1]*, pois já foram anteriormente comparados, passando a ser comparado somente com os elementos que estiverem em *A[3]*, *A[4]* e *A[5]*. Na seqüência, o elemento que estiver em *A[3]* é comparado com os elementos que estiverem em *A[4]* e *A[5]* e por fim o elemento que estiver em *A[4]* é comparado com o elemento que estiver em *A[5]*.

Seguindo este conceito, basta comparar o valor do elemento armazenado em *A[1]* com o valor do elemento armazenado em *A[2]*. Se o primeiro for maior que o segundo, então trocam-se os seus valores. Como a condição de troca é verdadeira, o elemento 9 de *A[1]* é maior que o elemento 8 de *A[2]*, passa-se para *A[1]* o elemento 8 e para *A[2]* passa-se o elemento 9, desta forma os valores dentro da matriz passaram a ter a seguinte formação:

Matriz A	
Índice: <i>i</i>	Elemento: <i>A[i]</i>
1	8
2	9
3	7
4	5
5	3

Seguindo a regra de ordenação, o atual valor de *A[1]* deverá ser comparado com o próximo valor após a sua última comparação. Sendo assim, deverá ser comparado com o próximo valor após a sua última comparação. Sendo assim, deverá ser comparado com o valor existente em *A[3]*. O atual valor do elemento de *A[1]* é maior que o valor do elemento de *A[3]*. Ou seja, 8 é maior que 7. Efetua-se então a sua troca, ficando *A[1]* com o elemento de valor 7 e *A[3]* com o elemento de valor 8. Assim sendo, os valores da matriz passam a ter a seguinte formação:

Matriz A	
Índice: i	Elemento: A[i]
1	7
2	9
3	8
4	5
5	3

Agora deverão ser comparados os valores dos elementos armazenados nas posições A[1] e A[4]. O valor do elemento 7 de A[1] é maior que o valor do elemento 5 de A[4]. Eles são trocados, passando A[1] a possuir o elemento 5 e A[4] a possuir o elemento 7. A matriz passa a ter a seguinte formação:

Matriz A	
Índice: i	Elemento: A[i]
1	5
2	9
3	8
4	7
5	3

Observe que até aqui os elementos comparados foram sendo trocados de posição, estando agora em A[1] o elemento de valor 5 e que será mudado mais uma vez por ser maior que o valor do elemento 3 armazenado em A[5]. Desta forma, a matriz passa a ter a seguinte formação:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	9
3	8
4	7
5	5

A partir deste ponto o elemento de valor 3 armazenado em A[1] não necessitará mais ser comparado. Assim sendo, deverá ser pego o atual valor do elemento da posição A[2] e ser comparado sucessivamente com todos os outros elementos restantes. Desta forma, queremos dizer que o valor do elemento armazenado em A[2] deverá ser comparado com os elementos armazenados em A[3], A[4] e A[5], segundo a regra da aplicação de propriedade distributiva.

Comparando o valor do elemento 9 da posição A[2] com o elemento 8 da posição A[3] e efetuando a troca de forma que 8 esteja em A[2] e 9 esteja em A[3], a matriz passa a ter a seguinte formação:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	8
3	9
4	7
5	5

Em seguida o atual valor do elemento de A[2] deve ser comparado com o valor do elemento de A[4]. 8 é maior que 7 e são trocados, ficando A[2] com 7 e A[4] com 8. Desta forma, a matriz passa a ter a seguinte formação:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	7
3	9
4	8
5	5

Será então dada continuidade a processo de comparação e troca. O atual valor do elemento na posição A[2] é 7 e será comparado com o valor do elemento A[5] que é 5. São estes trocados, passando A[2] ficar com o elemento 5 e A[5] ficar com o elemento 7, conforme indicado no esquema abaixo:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	5
3	9
4	8
5	7

Note que até este ponto a posição A[2] foi comparada com todas as posições subseqüentes a ela, não tendo mais nenhuma comparação para ela. Agora será efetuada a comparação da próxima posição com o restante. No caso, de A[3] com A[4] e A[5]. Sendo assim, o valor do elemento da posição A[3] será comparado com o valor da posição A[4]. Como 9 é maior que 8 eles serão trocados:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	5
3	8
4	9
5	7

A seguir, será comparado o valor do elemento da posição A[3] com o valor do elemento da posição A[5]. Sendo o primeiro maior que o segundo, ocorre a troca. Desta forma A[3] passa a possuir o elemento 7 e A[5] passa a possuir o elemento 8, como indicado em seguida:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	5
3	7
4	9
5	8

Tendo sido efetuadas todas as comparações de A[3] com A[4] e A[5], fica somente a última comparação que é A[4] com A[5], cujos valores são trocados, passando A[4] possuir o elemento de valor 8 e A[5] possuir o elemento de valor 9, como mostrado em seguida:

Matriz A	
Índice: i	Elemento: A[i]
1	3
2	5
3	7
4	8
5	9

Desta forma, pode-se notar que a referida ordenação foi executada, apresentando os elementos da matriz em ordem crescente.

Para dados do tipo literal o processo é o mesmo, uma vez que cada letra possui um valor diferente da outra. A letra “A”, por exemplo, tem valor menor que a letra “B”, e assim por diante. Se a letra “A” maiúscula for comparada com a letra “a” minúscula, terão valores diferentes. Cada caractere é guardado dentro da memória de um computador segundo o valor de um código, que recebe o nome de **ASCII** (American Standard Code for Information Interchange – Código Americano Padrão para Troca de Informações). E é com base nesta tabela que o processo de ordenação trabalha, pois cada caractere tem um peso, um valor previamente determinado, segundo este padrão.

Diagrama de Blocos

A seguir são apresentados o diagrama de blocos da entrada, processamento de ordenação e apresentação dos nomes ordenados. Atente para dois pontos:

O primeiro a ser observado é a utilização de uma segunda variável para controlar o índice subsequente no processo de ordenação, no caso a variável “j”. Observe que a variável i é iniciada pela instrução **para** como: **i de 1**, e no segundo pela instrução **para** que está sendo encadeada a primeira e iniciando a variável j, como: **j de i+1**. Isto implica na seguinte seqüência:

Quando i for	j será
1	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
2	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
3	4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
4	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
5	6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
6	7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
7	8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
8	9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
9	10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
10	11, 12, 13, 14, 15, 16, 17, 18, 19, 20
11	12, 13, 14, 15, 16, 17, 18, 19, 20
12	13, 14, 15, 16, 17, 18, 19, 20
13	14, 15, 16, 17, 18, 19, 20
14	15, 16, 17, 18, 19, 20
15	16, 17, 18, 19, 20
16	17, 18, 19, 20
17	18, 19, 20
18	19, 20
19	20

Observe que somente quando a variável j atinge o valor 20 é que este looping se encerra, retornando ao looping da variável i, acrescentando mais um em i até i atinja o seu limite e ambos os loopings sejam encerrados.

Quando a variável i for 1, a variável j será 2 e contará até 20. Ao final deste ciclo, a variável i é acrescentada de mais 1 tornando-se 2; assim sendo a variável j passa a ser 3. Quando a variável j voltar a ser 20 novamente, a variável i passa a ser 3 e a variável j passa a ser 4. Este ciclo irá ser executado até que por

fim a variável i seja 19 e a variável j seja 20, e será comparado o penúltimo elemento com o seu elemento subsequente, no caso, o último.

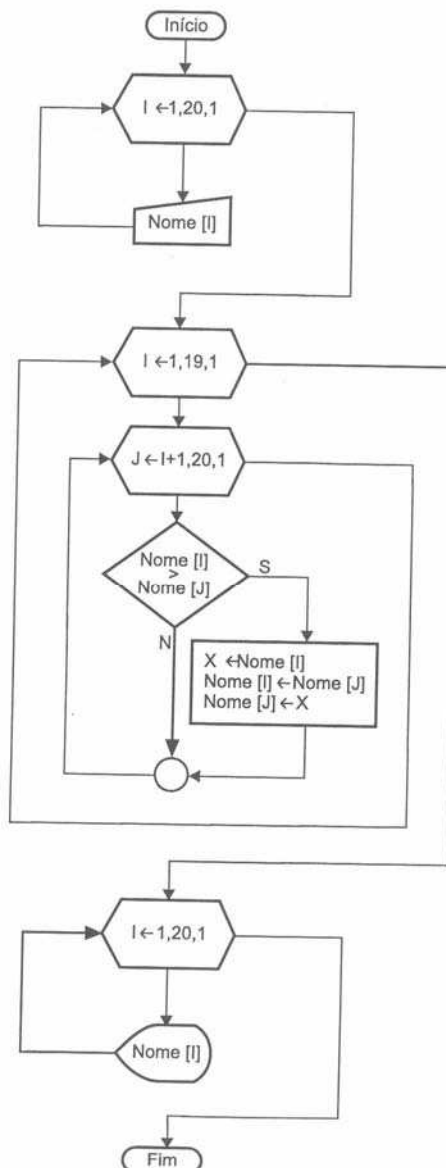
O segundo ponto a ser observado é o fato da utilização do algoritmo de troca, utilizado junto da instrução de decisão **se** $NOME[i] > NOME[j]$ **então**. Após a verificação desta condição, sendo o primeiro nome maior que o segundo, efetua-se então a sua troca com o algoritmo:

```

X ← NOME[i]
NOME[i] ← NOME[j]
NOME[j] ← X

```

Considere o vetor $NOME[i]$ como o valor "Carlos" e o vetor $NOME[j]$ com o valor "Alberto". Ao final $NOME[i]$ deverá estar com "Alberto" e $NOME[j]$ deverá estar com "Carlos". Para conseguir este efeito, é necessária a utilização de uma variável de apoio, a qual será chamada X. Para que o vetor $NOME[i]$ fique livre para receber o valor do vetor $NOME[j]$, X deverá ser implicado pelo valor do vetor $NOME[i]$. Assim sendo, X passa a ser "Carlos". Neste momento pode-se implicar o valor de $NOME[j]$ em $NOME[i]$. Desta forma o vetor $NOME[i]$ passa a possuir o valor "Alberto". Em seguida o vetor $NOME[j]$ é implicado pelo valor que está em X. Ao final deste processo, ter-se-á $NOME[i]$ com "Alberto" e $NOME[j]$ com "Carlos".



Português Estruturado

```

algoritmo "Ordena_Nomes"
var
    NOME: vetor[1..20] de literal
    i, j: inteiro
    X: literal

inicio

    {Rotina e entrada de dados}

    para i de 1 ate 20 faca
        escreva("Digite o nome numero",i," : ")
        leia(NOME[i])
    fimpara

    {Rotina de processamento de ordenação}

    para i de 1 ate 19 faca
        para j de i+1 ate 20 faca
            se NOME[i] > NOME[j] entao
                X <- NOME[i]
                NOME[i] <- NOME[j]
                NOME[j] <- X
            fimse
        fimpara
    fimpara

    {Rotina de saída com dados ordenados"}

    para i de 1 ate 20 faca
        escreval(i, " ", NOME[i])
    fimpara
fimalgoritmo

```

Um detalhe utilizado neste exemplo foram os comentários colocados entre as chaves. Comentários deste tipo servem para documentar o programa, facilitando a interpretação de um determinado trecho.

7.2 – Métodos de Pesquisa em Matriz

Quando se trabalha com matrizes, elas poderão gerar grandes tabelas, dificultando localizar um determinado elemento de forma rápida. Imagine uma matriz possuindo 4.000 elementos (4.000 nomes de pessoas). Será que você conseguiria encontrar rapidamente um elemento desejado de forma manual, mesmo estando a lista de nomes em ordem alfabética? Certamente que não. Para solucionar este tipo de problema, você pode fazer pesquisas em matrizes com o uso de programação. Serão apresentados dois métodos para efetuar pesquisa em uma matriz, sendo o primeiro o método seqüencial e o segundo o método de pesquisa binária.

7.2.1 – Método de Pesquisa Seqüencial

O primeiro método consiste em efetuar a busca da informação desejada a partir do primeiro elemento seqüencialmente até o último. Localizando a informação no caminho, ela é apresentada. Este método de pesquisa é lento, porém eficiente nos casos em que uma matriz encontra-se com seus elementos desordenados.

7.2.2 – Método de Pesquisa Binária

O segundo método de pesquisa é em média mais rápido que o primeiro, porém exige que a matriz esteja

previamente classificada, pois este método “divide” a lista em duas partes e “procura” saber se a informação a ser pesquisada está acima ou abaixo da linha de divisão. Se estiver acima, por exemplo, toda a metade abaixo é desprezada. Em seguida, se a informação não foi encontrada, é novamente dividida em duas partes, e pergunta se aquela informação está acima ou abaixo, e assim vai sendo executada até encontrar ou não a informação pesquisada. Pelo fato de ir dividindo sempre em duas partes o volume de dados é que o método recebe a denominação de pesquisa binária. Para exemplificar a utilização deste tipo de pesquisa, imagine a seguinte tabela.

Índice	Nome
1	André
2	Carlos
3	Frederico
4	Golias
5	Sílvia
6	Sílvio
7	Waldir

A tabela está representando uma matriz do tipo vetor com 7 elementos literais. Deseja-se neste momento efetuar uma pesquisa de um dos seus elementos. No caso, será escolhido o elemento Waldir (por acaso o último registro da tabela).

O processo binário consiste em pegar o número de registro e dividir por dois. Sendo assim, $7 \text{ div } 2 = 3$; o que nos interessa é somente o valor do quociente inteiro. Então a tabela fica dividida em duas partes como em seguida:

Primeira parte após primeira divisão

Índice	Nome
1	André
2	Carlos
3	Frederico

Segunda parte após primeira divisão

Índice	Nome
4	Golias
5	Sílvia
6	Sílvio
7	Waldir

Estando a tabela dividida em duas partes, deverá ser verificado se a informação Waldir está na primeira ou na segunda parte. Detecta-se que Waldir está na segunda parte. Desta forma despreza-se a primeira e divide-se em duas partes novamente a segunda parte da tabela. Como são 4 elementos divididos por 2, resultam duas tabelas, cada uma com dois elementos, conforme em seguida:

Primeira parte após a segunda divisão

Índice	Nome
4	Golias
5	Sílvia

Segunda parte após a segunda divisão

Índice	Nome
6	Sílvio
7	Waldir

Após esta segunda divisão, verifica-se em qual das partes Waldir está situado. Veja que está na segunda

parte; despreza-se a primeira e divide-se a segunda parte novamente por dois, sobrando um elemento em cada parte:

Primeira parte após a terceira divisão

Índice	Nome
6	Sílvia

Segunda parte após a terceira divisão

Índice	Nome
7	Waldir

Após a terceira divisão, Waldir é encontrado na segunda parte da tabela. Se estivesse sendo pesquisado o elemento Whashington, este não seria apresentado por não existir.

7.3 – Exercício de Aprendizagem

A seguir, são demonstrados dois exemplos, fazendo uso dos dois métodos de pesquisa apresentados neste capítulo.

1º Exemplo

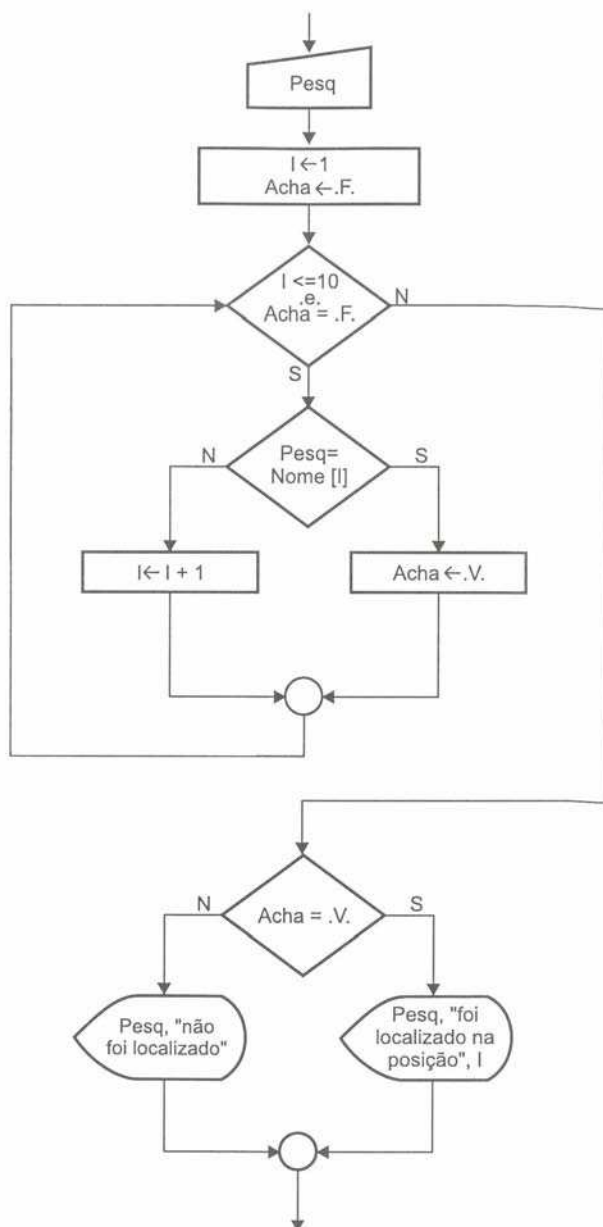
Considerando a necessidade de trabalhar com uma matriz com 10 nomes, seguem abaixo o algoritmo, diagrama de blocos e codificação em português estruturado para efetuar uma pesquisa seqüencial na referida matriz.

Algoritmo

1. Iniciar um contador e pedir a leitura de 10 nomes;
2. Criar um looping que efetue a pesquisa enquanto o usuário assim o desejar. Durante a fase de pesquisa, deverá ser solicitada a informação a ser pesquisada. Essa informação deverá ser comparada com o primeiro elemento; sendo igual mostra, caso contrário avança para o próximo. Se não achar em toda lista, informar que não existe elemento pesquisado; se existir deverá mostrá-lo;
3. Encerrar a pesquisa quando desejado.

Diagrama de Blocos

O diagrama seguinte concentra-se somente no trecho de pesquisa seqüencial, uma vez que os demais algoritmos já são conhecidos.



Português Estruturado

```

algoritmo "Pesquisa_Sequencial"
var
  NOME: vetor[1..10] de literal
  i: inteiro
  PESQ, RESP: literal
  ACHA: logico

inicio
  para i de 1 ate 10 faca
    escreva("Digite o nome numero",i," : ")
    leia(NOME[i])
  fimpara
  RESP ← "sim"
  enquanto (RESP = "sim") faca
    escreva("Entre o nome a ser pesquisado: ")
    leia(PESQ)
    i ← 1
    ACHA ← falso
    enquanto (i ≤ 10) e (ACHA = falso) faca
      se (PESQ = NOME[i]) entao
        ACHA ← verdadeiro
  
```

```

senao
    i <- i + 1
fimse
fimenquanto
se (ACHA = verdadeiro) entao
    escreval(PESQ, " foi localizado na posicao ", i)
senao
    escreval(PESQ, " não foi localizado")
fimse

escreva("Deseja continuar? ")
leia(RESF)
fimenquanto
finalgoritmo

```

Anteriormente foi montada a rotina de pesquisa empregada dentro de um contexto. Observe o trecho seguinte que executa a pesquisa com seus comentários:

```

01:    leia(PESQ)
02:    i <- 1
03:    ACHA <- falso
04:    enquanto (i <= 10) e (ACHA = falso) faca
05:        se (PESQ = NOME[i]) entao
06:            ACHA <- verdadeiro
07:        senao
08:            i <- i + 1
09:        fimse
10:    fimenquanto
11:    se (ACHA = verdadeiro) entao
12:        escreval(PESQ, " foi localizado na posicao ", i)
13:    senao
14:        escreval(PESQ, " não foi localizado")
15:    fimse

```

Na linha 1, é solicitado que se informe o nome a ser pesquisado na variável PESQ, em seguida, na linha 2, é setado o valor do contador de índice como 1 e na linha 3, a variável ACHA é setada como tendo um valor falso. A linha 4 apresenta instrução **enquanto** indicando que enquanto o valor da variável i for menor ou igual a 10 e simultaneamente o valor da variável ACHA for falso, deverá ser processado o conjunto de instruções situadas nas linhas 5, 6, 7, 8 e 9.

Neste ponto, a instrução **se** da linha 5 verifica se o valor da variável PESQ é igual ao valor da variável indexada NOME[1], e se for igual, é sinal que o nome foi encontrado. Neste caso, a variável ACHA passa a ser verdadeira, forçando a execução da linha 11, uma vez que uma das condições do laço **enquanto** da linha 4 se tornou falsa. Na linha 11, é confirmado se a variável ACHA está mesmo com o valor verdadeiro. Sendo esta condição verdadeira, é apresentada a mensagem da linha 12.

Caso na linha 5 seja verificado que o valor de PESQ não é igual a NOME[1], será então incrementado 1 na variável i. Será executada a próxima verificação de PESQ com NOME[2], e assim por diante. Caso o processamento chegue até ao final e não seja encontrado nada, a variável ACHA permanece com valor falso. Quando analisada pela linha 11, será então falsa e apresentará a mensagem da linha 14.

Observe que a variável ACHA exerce um papel importante dentro da rotina de pesquisa, pois serve como um pivô, estabelecendo um valor verdadeiro quando uma determinada informação é localizada. Este tipo de tratamento de variável é conhecido pelo nome de FLAG (bandeira). A variável ACHA é o flag, podendo dizer que ao começar a rotina a bandeira esta “abaixada” – falsa; quando a informação é encontrada, a bandeira é “levantada”- verdadeira, indicando a localização da informação desejada.

2º Exemplo

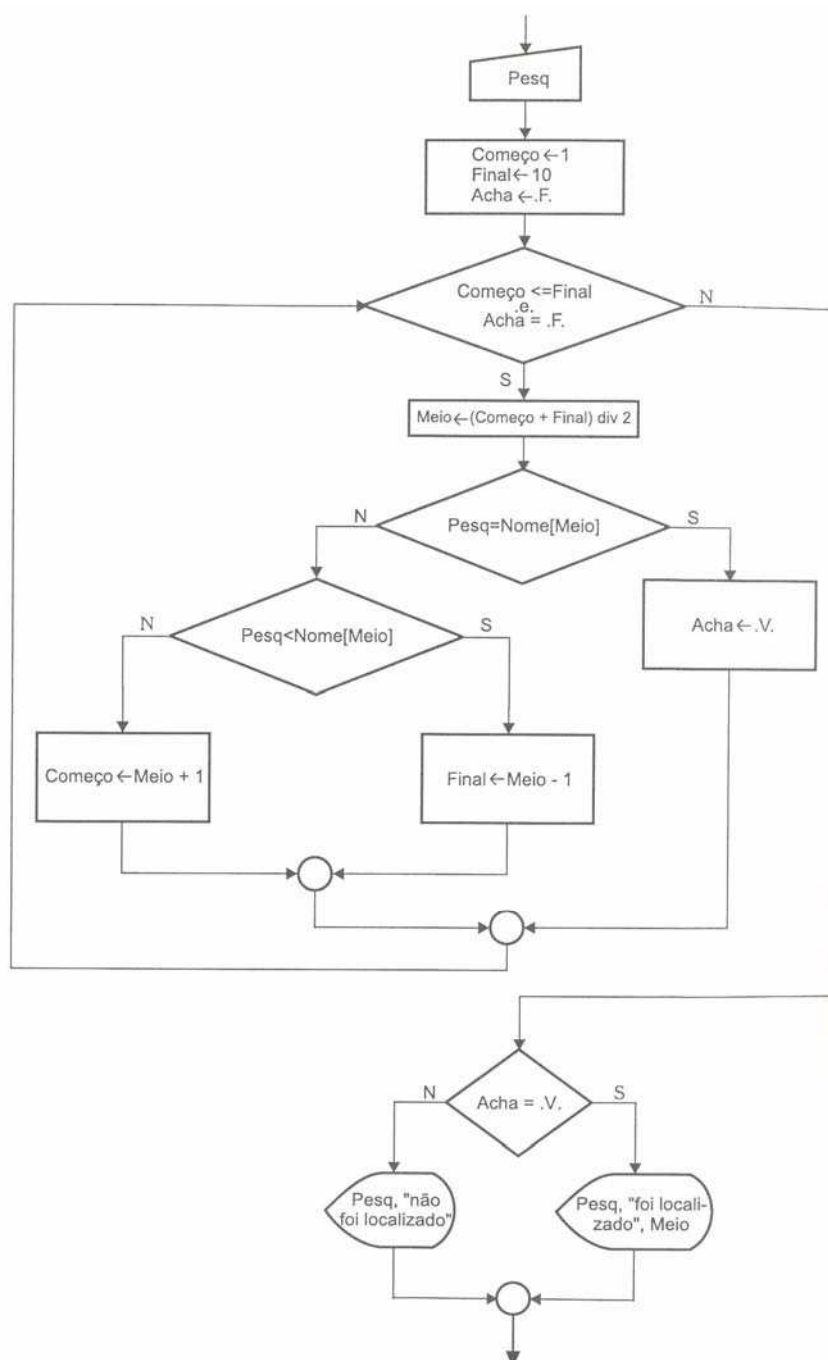
Considerando a necessidade de trabalhar com uma matriz com 10 nomes, seguem abaixo o algoritmo, diagrama de blocos e codificação em português estruturado para efetuar uma pesquisa binária na referida matriz.

Algoritmo

1. Iniciar um contador e pedir a leitura de 10 nomes e colocá-los em ordem alfabética;
2. Criar um laço de repetição que efetua a pesquisa enquanto o usuário assim o desejar. Durante a fase de pesquisa deverá ser solicitada a informação a ser pesquisada. Essa informação deverá ser comparada utilizando-se o método de pesquisa binária. Sendo igual mostra, caso contrário avança para o próximo. Se não achar em toda lista, informar que não existe o elemento pesquisado; se existir deverá mostrá-lo;
3. Encerrar a pesquisa quando desejado.

Diagrama de Blocos

Como no primeiro exemplo, este também se concentra somente no trecho de pesquisa binária, uma vez que os demais algoritmos já são conhecidos.



Português Estruturado

algoritmo "Pesquisa_Binaria"

```

var
  NOME: vetor[1..10] de literal
  i, j, COMECO, FINAL, MEIO: inteiro
  PESQ, RESP, X: literal
  ACHA: logico

inicio
  para i de 1 ate 10 faca
    leia(NOME[i])
  fimpara

  {Ordenação}
  para i de 1 ate 9 faca
    para j de i+1 ate 10 faca
      se (NOME[i] > NOME[j]) entao
        X <- NOME[i]
        NOME[i] <- NOME[j]
        NOME[j] <- X
      fimse
    fimpara
  fimpara

  {Mostra registros ordenados}
  para i de 1 ate 10 faca
    escreval("NOME[" , i, " ] = " , NOME[i])
  fimpara

  {Pesquisa}
  RESP <- "sim"
  enquanto (RESP="sim") faca
    escreva("Entre com o nome a ser pesquisado: ")
    leia(PESQ)
    COMECO <- 1
    FINAL <- 10
    ACHA <- falso
    enquanto (COMECO <= FINAL) e (ACHA = falso) faca
      MEIO <- (COMECO + FINAL) div 2
      se (PESQ = NOME[MEIO]) entao
        ACHA <- verdadeiro
      senao
        se (PESQ < NOME[MEIO]) entao
          FINAL <- MEIO - 1
        senao
          COMECO <- MEIO + 1
        fimse
      fimse
    fimenquanto
    se (ACHA = verdadeiro) entao
      escreval(PESQ, " foi localizado na posição " , MEIO)
    senao
      escreval(PESQ, " não foi localizado")
    fimse
    escreva("Deseja continuar? ")
    leia(RESP)
  fimenquanto
fimalgoritmo

```

Anteriormente foi montada a rotina de pesquisa empregada dentro de um contexto. Observe abaixo o trecho que executa a pesquisa com comentários:

```

01:      leia(PESQ)
02:      COMECO <- 1
03:      FINAL <- 10
04:      ACHA <- falso
05:      enquanto (COMECO <= FINAL) e (ACHA = falso) faca
06:        MEIO <- (COMECO + FINAL) div 2
07:        se (PESQ = NOME[MEIO]) entao
08:          ACHA <- verdadeiro
09:        senao
10:          se (PESQ < NOME[MEIO]) entao
11:            FINAL <- MEIO - 1

```



```

12:      senao
13:          COMECO <- MEIO + 1
14:      fimse
15:  fimse
16:  fimenquanto
17:  se (ACHA = verdadeiro) entao
18:      escreval(PESQ, " foi localizado na posição ", MEIO)
19:  senao
20:      escreval(PESQ, " não foi localizado")
21:  fimse

```

Na linha 1, é solicitado que seja informado o dado a ser pesquisado. As linhas 2 e 3 inicializam as variáveis de controle COMECO com 1 e FINAL com 10. A linha 4 inicializa o flag ACHA. A linha 5 apresenta a instrução que manterá a pesquisa em execução enquanto o COMECO for menor ou igual ao FINAL e simultaneamente o flag ACHA seja falso. Assim sendo, o processamento irá dividir a tabela ao meio conforme instrução na linha 6, em que 1 que é o começo da tabela é somado com 10 que é o fim da tabela, resultando no total 11 que dividido por 2 resultará 5 que é o meio da tabela.

Nesta ponto, a tabela está dividida em duas partes. A instrução da linha 8 verifica se o valor fornecido para PESQ é igual ao valor armazenado na posição NOME[5]. Se for, o flag é setado como verdadeiro sendo em seguida apresentada a mensagem da linha 18. Se condição de busca não for igual, poderá ocorrer uma de duas situações.

A primeira situação em que a informação pesquisada está numa posição acima da atual no caso NOME[5], ou seja, o valor da variável PESQ é menor que o valor de NOME[5] (linha 10). Neste caso, deverá a variável FINAL ser implicada pelo valor da variável MEIO subtraída de 1 (linha 11), ficando a variável FINAL com valor 4. Se for esta a situação ocorrida, será processada a linha 5 que efetuará novamente o looping pelo fato de o valor 1 da variável COMECO ser menor um, igual ao valor 4 da variável FINAL. A instrução da linha 6 divide a primeira parte da tabela em mais duas partes, desta forma, é somado com 4, resultando 5, e dividido por 2 resultará 2 (sendo considerada a parte inteira do resultado da divisão), que é o meio da primeira parte da tabela.

A segunda situação poderá ocorrer caso a informação pesquisada esteja abaixo do meio da tabela, ou seja, o valor da variável PESQ é maior que o valor de NOME[5] (linha 10). Neste caso, deverá a variável COMECO ser implicada pelo valor da variável MEIO somado com 1 (linha 13), ficando a variável COMECO com valor 6. Se for esta a situação ocorrida, será processada a linha 5 que efetuará novamente o looping pelo fato de o valor 6 da variável COMECO ser menor um, igual ao valor 10 da variável FINAL. A instrução da linha 6 divide a segunda parte da tabela em mais duas partes, desta forma, 6 é somado com 10, resultando 16 que dividido por 2 resultará 8, que é o meio da segunda parte da tabela.

Tanto na primeira como na segunda situação, será sempre pego uma das metades da tabela. A vantagem deste método está exatamente na metade desprezada, pois ela não será checada novamente.

7.4 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 07)

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado (Você deve gravar o exercício “a” como L07A, o exercício “b” como L07B, e assim por diante).

- a) Ler 12 elementos de uma matriz tipo vetor, coloca-los em ordem decrescente e apresentar os elementos ordenados.
- b) Ler 8 elementos em uma matriz A tipo vetor. Construir uma matriz B de mesma dimensão com os elementos da matriz A multiplicados 5. Montar uma rotina de pesquisa seqüencial, para pesquisar os elementos armazenados na matriz B.
- c) Ler uma matriz A tipo vetor que conterá 15 números inteiros. Construir uma matriz B de mesmo tipo, sendo que cada elemento de B deverá ser a metade (parte inteira) de cada elemento de A. Apresentar os elementos das matrizes A em ordem decrescente e os elementos da matriz B em ordem crescente.

7.5 - Exercícios de Extras

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado.

- a) Ler 20 elementos de uma matriz tipo vetor e construir uma matriz B de mesma dimensão com os mesmos elementos de A acrescentados de mais 2. Colocar os elementos da matriz B em ordem crescente. Montar uma rotina de pesquisa binária, para pesquisar os elementos armazenados na matriz B.
- b) Ler uma matriz A do tipo vetor com 20 elementos negativos. Construir uma matriz B de mesmo tipo e dimensão, sendo que cada elemento da matriz B deverá ser o valor positivo do elemento correspondente da matriz A. Desta forma, se em A[1] estiver armazenado o elemento -3, deverá estar em B[1] o valor 3, e assim por diante. Apresentar os elementos da matriz B em ordem decrescente.
- c) Ler 30 elementos de uma matriz A do tipo veto. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: todo elemento de B deverá ser o cubo do elemento de A correspondente. Montar uma rotina de pesquisa seqüencial, para pesquisar os elementos armazenados na matriz B.
- d) Ler três matrizes A, B e C de uma dimensão do tipo vetor com 15 elementos inteiros cada. Construir uma matriz D de mesmo tipo e dimensão que seja formada pela soma dos elementos correspondentes às matrizes A, B e C. Montar uma rotina de pesquisa binária, para pesquisar os elementos existentes na matriz D.

8 – Estruturas de Dados Homogêneas II

Nos capítulos 6 e 7, você teve contato com a utilização de variáveis indexadas (matrizes) do tipo vetor, ou seja, as matrizes de uma dimensão. Aprendeu a trabalhar com a classificação dos seus dados e a efetuar pesquisa dentro de sua estrutura. neste capítulo, será enfatizado o uso de matrizes com duas dimensões, conhecidas também por matrizes bidimensionais ou arranjos (arrays).

Pelo fato de utilizar-se de uma estrutura de dados homogênea, todos os elementos de uma matriz deverão ser do mesmo tipo.

8.1 – Matrizes com Mais de uma Dimensão

Anteriormente, houve contato com o uso de uma única variável indexada com apenas uma dimensão (uma coluna e várias linhas), quando foi utilizado o exemplo para efetuar o cálculo da média geral das médias dos oito alunos. A partir deste ponto, serão apresentadas tabelas com mais colunas, sendo assim, teremos variáveis no sentido horizontal e vertical.

Com o conhecimento adquirido até este ponto, você tem condições suficientes para elaborar um programa que efetue a leitura das notas dos alunos, o cálculo da média de cada aluno e no final apresentar a média do grupo, utilizando-se apenas de matrizes unidimensionais. Porém, há de considerar-se que o trabalho é grande, uma vez que se necessita manter um controle de cada índice em cada matriz para um mesmo aluno.

Para facilitar o trabalho com estruturas deste porte é que serão utilizadas matrizes com mais dimensões. A mais comum é a matriz de duas dimensões por se relacionar diretamente com a utilização de tabelas. Matrizes com mais de duas dimensões são utilizadas com menos freqüência, mas poderão ocorrer momentos em que se necessite trabalhar com um número maior de dimensões, porém estas são fáceis de utilizar se o leitor estiver dominando bem a utilização de uma matriz com duas dimensões.

Um importante aspecto a ser considerado é que na manipulação de uma matriz tipo vetor é utilizado uma única instrução de looping (**enquanto**, **para** ou **repita**). No caso de matrizes com mais dimensões, deverá ser utilizado o número de loopings relativo ao tamanho de suas dimensão. Desta forma, uma matriz de duas dimensões deverá ser controlada com dois loopings, de três dimensões deverá ser controlada por três loopings e assim por diante.

Em matrizes de mais de uma dimensão os seus elementos são também manipulados de forma individualizada, sendo a referência feita sempre por meio de dois índices: o primeiro para indicar a linha e o segundo para indicar a coluna. Desta forma, TABELA[2,3] indica que está sendo feita uma referência ao elemento armazenado na linha 2 coluna 3. Pode-se considerara que uma matriz com mais de uma dimensão é também um vetor, sendo válido para este tipo de matriz tudo o que já foi utilizado anteriormente para as matrizes de uma dimensão.

8.2 – Operações Básicas com Matrizes de Duas Dimensões

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. Desta forma é uma matriz de duas dimensões TABELA[1..8,1..5], onde seu nome é TABELA, possuindo um tamanho de 8 linhas (de 1 a 8) e 5 colunas (de 1 a 5), ou seja, é uma matriz de 8 por 5 (8x5). Isto significa que podem ser armazenados em TABELA até 40 elementos (posições) que podem ser utilizadas para armazenamento de seus elementos.

Matriz: TABELA

		Coluna				
		↓				
		1	2	3	4	5
Linha →	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

8.2.1 – Atribuição de uma Matriz

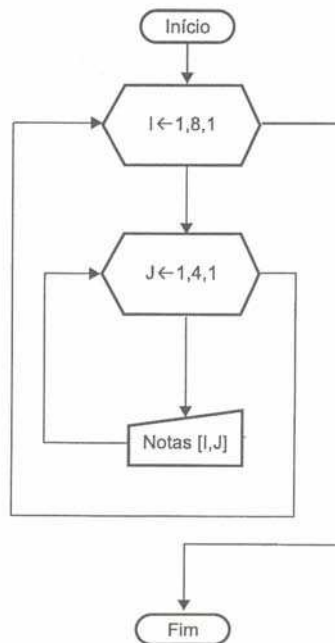
Uma matriz de duas dimensões é atribuída pela instrução **vetor** já utilizada para definir o uso de uma matriz de uma dimensão, sendo bastante parecidas em sua referência. A sintaxe é: **VARIÁVEL: vetor[<dimensão1>,<dimensão2>] de <tipo de dado>**, em que <dimensão1> e <dimensão2> são a indicação do tamanho da tabela e <tipo de dado> o tipo da matriz, que poderá ser formada por valores reais, inteiros, lógicos ou literais.

8.2.2 – Leitura dos Dados de uma Matriz

A leitura de uma matriz de duas dimensões, assim como das matrizes de uma dimensão é processada passo a passo, um elemento por vez, sendo utilizada a instrução **leia()** seguida da variável mais os seus índices. A seguir são apresentados o diagrama de blocos e codificação em português estruturado da leitura das 4 notas bimestrais de 8 alunos, sem considerar o cálculo da média.

Diagrama de Blocos

Observe que está sendo considerada a leitura das 4 notas de 8 alunos. Assim sendo, a tabela em questão armazena 32 elementos. Um detalhe a ser considerado é a utilização de duas variáveis para controlar os dois índices de posicionamento de dados na tabela. Anteriormente, foi utilizada a variável *i* para controlar as posições dos elementos dentro da matriz, ou seja, a posição em nível de linha. Neste exemplo, a variável *i* continua tendo o mesmo efeito e a segunda variável, a *j*, está controlando a posição da coluna.



Analisando o diagrama de blocos, temos a inicialização das variáveis *i* e *j* como 1, ou seja, a leitura será efetuada na primeira linha da primeira coluna. Em seguida é iniciado em primeiro lugar o looping da variável *i* para controlar a posição em relação às linhas e depois é iniciado o looping da variável *j* para controlar a posição em relação às colunas.

Veja que, ao serem iniciados os valores para o preenchimento da tabela, eles são colocados na posição NOTAS[1,1], lembrando que o primeiro valor dentro dos colchetes representa a linha e o segundo representa a coluna. Assim sendo, será então digitado para o primeiro aluno a sua primeira nota. Depois é incrementado mais 1 em relação à coluna, sendo colocada para a entrada a posição NOTAS[1,2], linha 1 e coluna 2. Desta forma, será digitado para o primeiro aluno a sua segunda nota.

Quando o contador de coluna, o looping da variável *j*, atingir o valor 4, ele será encerrado. Em seguida o contador da variável *i* será incrementado com mais 1, tornando-se 2. Será então inicializado novamente o contador *j* em 1, permitindo que seja digitado um novo dado na posição NOTAS[2,1].

O mecanismo de preenchimento estender-se-á até que o contador de linhas atinja o seu último valor, no caso 8. Esse looping é o principal, tendo a função de controlar o posicionamento na tabela por aluno. O segundo looping, mais interno, controla o posicionamento das notas.

Português Estruturado

```

algoritmo "Ler_Elementos"
var
  NOTAS: vetor[1..8, 1..4] de real
  i, j: inteiro

inicio
  para i de 1 ate 8 faca
    para j de 1 ate 4 faca
      leia(NOTAS[i,j])
    fimpara
  fimpara

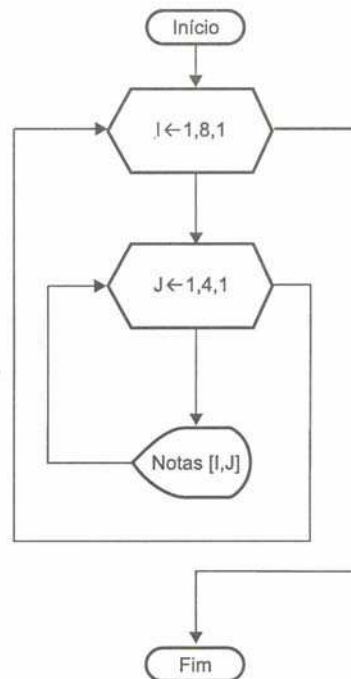
```

fimalgoritmo

8.2.3 – Escrita de Dados de uma Matriz

O processo de escrita é bastante parecido com o processo de leitura de seus elementos. Supondo que após a leitura das notas dos 8 alunos, houvesse a necessidade de efetuar a apresentação das notas.

Diagrama de Blocos



Português Estruturado

```

algoritmo "Ler_Elementos"
var
  NOTAS: vetor[1..8, 1..4] de real
  i, j: inteiro

inicio
  para i de 1 ate 8 faca
    para j de 1 ate 4 faca
      escrever(NOTAS[i, j])
    fimpara
  fimpara
fimalgoritmo
  
```

8.3 – Exercício de Aprendizagem

Para demonstrar a utilização de matrizes de duas dimensões, considere os exemplos apresentados em seguida:

1º Exemplo

Desenvolver um programa de agenda que cadastre o nome, endereço, CEP e telefone de 6 pessoas. Ao final, o programa deverá apresentar os seus elementos dispostos em ordem alfabética, independentemente

da forma em que foram digitados.

Algoritmo

Para resolver este problema, você precisa uma tabela com 10 linhas (pessoas) e 5 colunas (dados pessoais). Assim sendo, imagine esta tabela como sendo:

	Nome 1	Endereço 2	CEP 3	Bairro 4	Telefone 5
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Em cada coluna é indicado o seu número, sendo 5 colunas, uma para cada informação pessoal e o número de linha totalizando um conjunto de 10 informações.

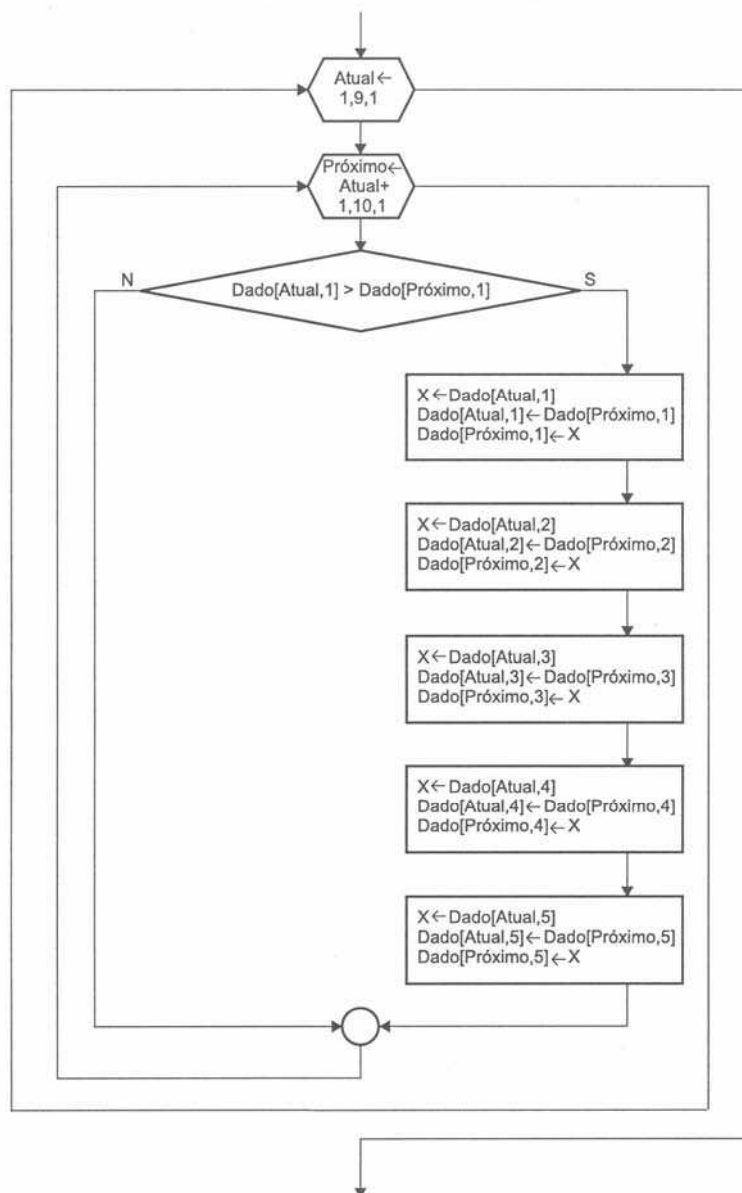
Nesta tabela, são utilizados dois elementos numéricos, o CEP e o Telefone, mas como não são executados cálculos com esses números, eles são armazenados como caracteres.

Depois de cadastrar todos os elementos, é iniciado o processo de classificação alfabética pelo nome de cada pessoa. Este método já foi anteriormente estudado, bastando aplica-lo neste contexto. Porém, após a comparação do primeiro nome com o segundo, sendo o primeiro maior que o segundo, deverão ser trocados, mas os elementos relacionados ao nome também deverão ser trocados no mesmo nível de verificação, ficando para o final o trecho de apresentação de todos os elementos.

Diagrama de Blocos

Neste exemplo, não estão sendo utilizados para a entrada de dados dois loopings para controlar o posicionamento dos elementos na matriz. Note que as referências feitas ao endereço das colunas são citadas como constantes, durante a variação do valor da variável *i*.

Com relação à ordenação de elementos de uma matriz de duas dimensões, o processo é o mesmo utilizado para ordenar matrizes de uma dimensão. Se você sabe fazer a ordenação de um estilo de matriz, sabe fazer a ordenação de qualquer estilo, seja ela da dimensão que for. Observe no trecho de ordenação, a troca de posição de todos os elementos assim que os nomes são comparados e verificados que estão fora de ordem. Perceba que assim que o nome é trocado de posição, os demais elementos relacionados a ele na mesma linha também o são.



Para a apresentação dos dados ordenados estão sendo utilizados os dois loopings para controlar linha e coluna.

Português Estruturado

```

algoritmo "Agenda"
var
  dado: vetor [1..6, 1..4] de literal
  i, j, atual, proximo: inteiro
  x: literal

inicio
  // Rotina de entrada de dados
  para i de 1 ate 6 passo 1 faca
    escreva("Nome.....: ")
    leia(dado[i, 1])
    escreva("Endereco.: ")
    leia(dado[i, 2])
    escreva("CEP.....: ")
    leia(dado[i, 3])
    escreva("Telefone.: ")
    leia(dado[i, 4])
  
```

```

    escreval()
fimpara

// Rotina de ordenação e troca de todos os elementos
para atual de 1 ate 5 faca
    para proximo de atual+1 ate 6 faca
        se (dado[atual,1]) > (dado[proximo,1]) entao
            // Troca Nome
            x <- dado[atual,1]
            dado[atual,1] <- dado[proximo,1]
            dado[proximo,1] <- x

            // Troca Endereco
            x <- dado[atual,2]
            dado[atual,2] <- dado[proximo, 2]
            dado[proximo,2] <- x

            // Troca CEP
            x <- dado[atual, 3]
            dado[atual,3] <- dado[proximo,3]
            dado[proximo,3] <- x

            // Troca Telefone
            x <- dado[atual,4]
            dado[atual,4] <- dado[proximo,4]
            dado[proximo,4] <- x
        fimse
    fimpara
fimpara

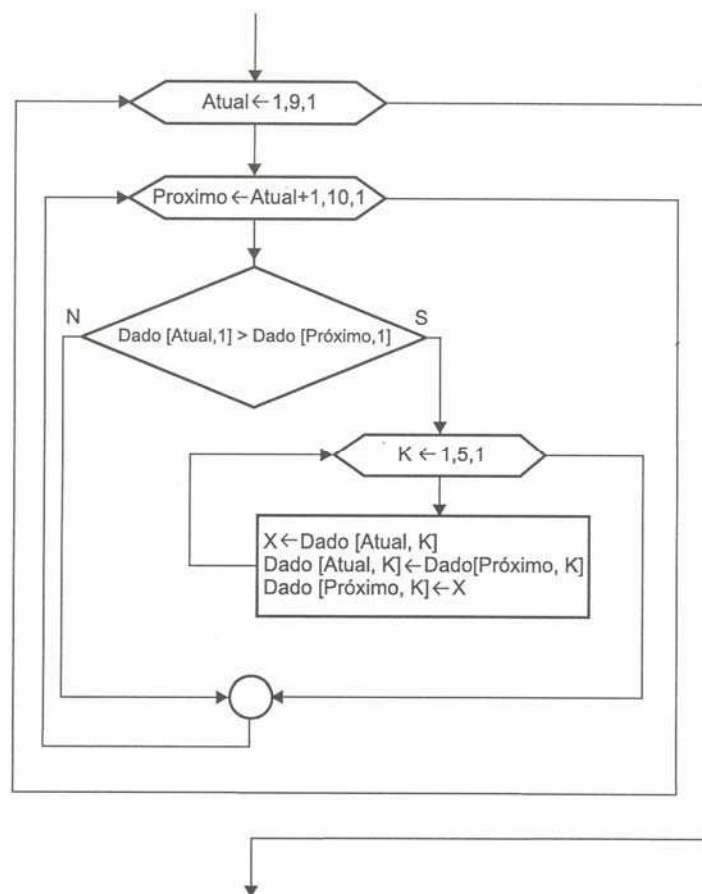
// Rotina de saída dos dados
para i de 1 ate 6 faca
    para j de 1 ate 4 faca
        escreva(dado[i,j], " ")
    fimpara
    escreval()
fimpara
finalgoritmo

```

O trecho de ordenação do programa AGENDA pode ser simplificado com a inserção de um looping para administrar a troca após a verificação da condição: **se** (dado[atual,1]>DADO[próximo,1]) **entao**.

Diagrama de Blocos

Neste segundo exemplo está sendo levado em consideração apenas para o diagrama de blocos o trecho correspondente a ordenação.



Português Estruturado

algoritmo "Agenda"

var

dado: **vetor** [1..6, 1..4] **de literal**

i, j, k, atual, proximo: **inteiro**

x: **literal**

inicio

// Rotina de entrada de dados

para i **de** 1 **ate** 6 **passo** 1 **faca**

escreva("Nome.....: ")

leia(dado[i, 1])

escreva("Endereco.: ")

leia(dado[i, 2])

escreva("CEP.....: ")

leia(dado[i, 3])

escreva("Telefone.: ")

leia(dado[i, 4])

escreval()

fimpara

// Rotina de ordenação e troca de todos os elementos

para atual **de** 1 **ate** 5 **faca**

para proximo **de** atual+1 **ate** 6 **faca**

se (dado[atual,1]) > (dado[proximo,1]) **entao**

para k **de** 1 **ate** 4 **faca**

 x ← dado[atual,k]

 dado[atual,k] ← dado[proximo,k]

 dado[proximo,k] ← x

fimpara

fimse

fimpara

fimpara

```
// Rotina de saída dos dados
para i de 1 ate 6 faca
  para j de 1 ate 4 faca
    escreva(dado[i,j], " ")
  fimpara
  escreval()
fimpara
finalgoritmo
```

2º Exemplo

Desenvolver um programa que efetue a leitura dos nomes de 8 alunos e também de suas 4 notas bimestrais. Ao final, o programa deverá apresentar o nome de cada aluno classificado em ordem alfabética, bem como suas média e a média geral dos 8 alunos.

Algoritmo

Neste exemplo é apresentado um problema cuja solução será utilizar duas matrizes para a entrada de dados. Já é sabido que uma matriz trabalha somente com dados de mesmo tipo (homogêneos). E neste caso, em particular, será necessário ter uma matriz tipo vetor para armazenar os nomes e a outra tipo tabela para armazenar as notas, uma vez que os tipos de dados a serem manipulados são diferentes.

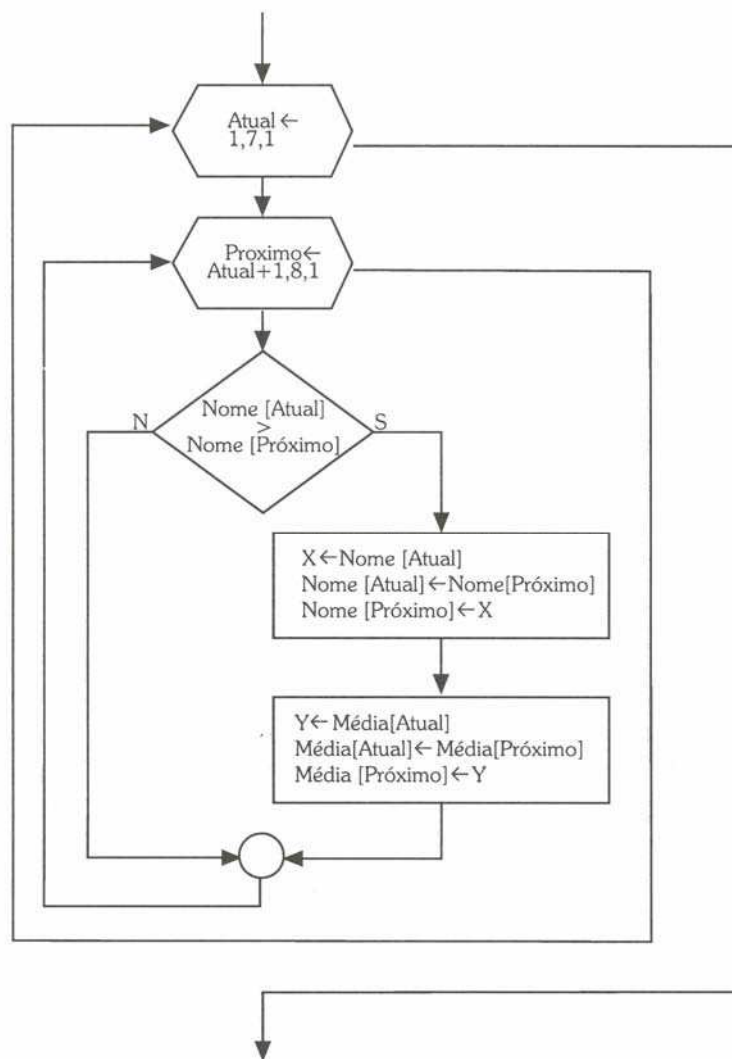
O programa deverá pedir o nome do aluno e em seguida as quatro notas, calcular a média e armazená-la numa terceira matriz de uma dimensão, para então apresentar o nome de cada aluno e sua respectiva média, bem como, a média do grupo.

Logo no início, a variável **SOMA_MD** é inicializada com valor zero. Esta variável será utilizada para armazenar a soma das médias de cada aluno durante a entrada de dados.

Depois a instrução **para i de 1 ate 8 faca** inicializa o primeiro looping que tem por finalidade controlar o posicionamento dos elementos no sentido linear. Neste ponto, a variável **SOMA_NT** é inicializada com o valor zero para o primeiro aluno. Esta variável irá guardar a soma das quatro notas de cada aluno durante a execução do segundo looping. Neste momento, é solicitado antes do segundo looping o nome do aluno.

Toda vez que o segundo looping é encerrado, a matriz **MEDIA** é alimentada com o valor da variável **SOMA_NT** dividido por 4. Deste modo, tem-se o resultado da média do aluno cadastrado. Em seguida é efetuado o cálculo da soma das médias de cada aluno na variável **SOMA_MD**, que posteriormente servirá para determinar o cálculo da média do grupo. É neste ponto, que o primeiro looping repete o seu processo para o próximo aluno, e assim irá transcorrer até o último aluno.

Após a disposição dos alunos por ordem alfabética de nome, é dado início à apresentação dos nomes de cada aluno e suas respectivas médias. Ao final, a variável **MEDIA_GP** determina o cálculo da média do grupo (média das médias), através do valor armazenado na variável **SOMA_MD** dividido por 8 (total de alunos).



Português Estruturado

```

algoritmo "Agenda"
var
x: literal
i, j, atual, proximo: inteiro
y, soma_nt, soma_md, media_gp: real
nota: vetor[1..8, 1..4] de real
media: vetor[1..8] de real
nomes: vetor[1..8] de literal

inicio
soma_md <- 0
para i de 1 ate 8 faca
    soma_nt <- 0
    escreva("Aluno ", i)
    leia(nomes[i])
    para j de 1 ate 4 faca
        escreva("nota ", j, ": ")
        leia(nota[i, j])
        soma_nt <- soma_nt + nota[i, j]
    fimpara
    media[i] <- soma_nt / 4
    soma_md <- soma_md + media[i]
fimpara

// Rotina de ordenação e troca de todos os elementos
para atual de 1 ate 7 faca
    para proximo de atual+1 ate 8 faca

```

```

se (nomes[atual]) > (nomes[proximo]) entao
  // Troca Nome
  x <- nomes[atual]
  nomes[atual] <- nomes[proximo]
  nomes[proximo] <- x
  y <- media[atual]
  media[atual] <- media[proximo]
  media[proximo] <- y
fimse
fimpara
fimalgoritmo

media_gp <- soma_md / 8
// Rotina de saída dos dados
para i de 1 ate 8 faca
  escreval("Aluno .: ", nomes[i])
  escreval("Media .: ", media[i])
fimpara
escreva("Media Geral: ", media_gp)
fimalgoritmo

```

8.4 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 08)

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado (Você deve gravar o exercício “a” como L08A, o exercício “b” como L08B, e assim por diante).

- Ler duas matrizes A e B, cada uma de duas dimensões com 5 linhas e 3 colunas. Construir uma matriz C de mesma dimensão, que é formada pela soma dos elementos da matriz A com os elementos da matriz B. Apresentar a matriz C.
- Ler duas matrizes A e B, cada uma com uma dimensão para 7 elementos. Construir uma matriz C de duas dimensões, em que a primeira coluna deverá ser formada pelos elementos da matriz A e a segunda coluna deverá ser formada pelos elementos da matriz B. Apresentar a matriz C.
- Ler 20 elementos para uma matriz qualquer, considerando que ela tenha o tamanho de 4 linhas por 5 colunas, em seguida apresentar a matriz.
- Ler uma matriz A de uma dimensão com 10 elementos. Construir uma matriz C de duas dimensões com três colunas, em que a primeira coluna da matriz C é formada pelos elementos da matriz A somados com mais 5, a segunda coluna é formada pelo valor do cálculo da fatorial de cada elemento correspondente da matriz A e a terceira e última coluna deverá ser formada pelos quadrados dos elementos correspondentes da matriz A. Apresentar a matriz C. (Observe que fatorial de zero é igual a 1.)
- Ler uma matriz A de duas dimensões com 10 linhas e 10 colunas. Apresentar o somatório dos elementos situados na diagonal principal (posições A[1,1], A[2,2], A[3,3], A[4,4] e assim por diante) da referida matriz. (Claro que não é para efetuar a soma deste modo: SOMA <- A[1,1] + A[2,2] + A[3,3] + ... + A[10,10])
- Ler uma matriz A de duas dimensões com 7 linhas e 7 colunas. Construir uma matriz B de mesma dimensão, sendo que cada elemento da matriz B deverá ser o somatório de cada elemento correspondente da matriz A com o os seus índices (ou seja, se A[1,2] possui o valor 8, B[1,2] deverá possuir o valor 11, correspondente a 8+1+2), com exceção para os valores situados nos índices ímpares da diagonal principal (B[1,1], B[3,3], B[5,5], B[7,7]), os quais deverão ser o fatorial de cada elemento correspondente da matriz A. Apresentar ao final a matriz A e B lado a lado.
- Ler uma matriz A de duas dimensões com 4 linhas e 4 colunas. Ao final apresentar o total de elementos pares existentes dentro da matriz. (Observe que é o total de Elementos e não a soma total do conteúdo dos elementos.)

- h) Ler uma matriz A de duas dimensões com 10 linhas e 7 colunas. Ao final apresentar o total de elementos pares e o total de elementos ímpares existentes dentro da matriz. Apresentar também o percentual de elementos pares e ímpares em relação ao total de elementos da matriz. Supondo a existência de 20 elementos pares e 50 elementos ímpares, ter-se-ia 28.6 % de elementos pares e 71,4% de elementos ímpares.
- i) Desenvolver um programa que efetua a leitura dos nomes de 5 alunos e também de suas duas notas semestrais. Ao final deverá ser apresentado o nome de cada aluno classificado em ordem numérica crescente de suas médias anuais.

8.5 - Exercícios de Extras

1) Desenvolva os algoritmos, seus respectivos diagramas de bloco e codificação em português estruturado.

- a) Ler duas matrizes A e B, cada uma com uma dimensão para 12 elementos. Construir uma matriz C de duas dimensões, sendo que a primeira coluna da matriz C deverá ser formada pelos elementos da matriz A multiplicados por 2 e a segunda coluna deverá ser formada pelos elementos da matriz B subtraídos de 5. Apresentar a matriz C.
- b) Ler uma matriz A de duas dimensões com 8 linhas e 6 colunas. Construir uma matriz B de uma dimensão que seja formada pela soma de cada linha da matriz A. Ao final apresentar o somatório dos elementos da matriz B.
- c) Elaborar um programa que efetue a leitura de 20 valores inteiros em uma matriz A de duas dimensões com 4 linhas e 5 colunas. Construir uma matriz B de uma dimensão para 4 elementos que seja formada pelo somatório dos elementos correspondentes de cada linha da matriz A. Construir também uma matriz C de uma dimensão para 5 elementos que seja formada pelo somatório dos elementos correspondentes de cada coluna da matriz A. Ao final o programa deverá apresentar o total do somatório dos elementos da matriz B com o somatório dos elementos da matriz C.
- d) Ler duas matrizes A e B de duas dimensões com 5 linhas e 5 colunas. A matriz A deverá ser formada por valores que não seja divisíveis por 3, enquanto a matriz B deverá ser formada por valores que não sejam divisíveis por 6. As entradas dos valores nas matrizes deverão ser validadas pelo programa então pelo usuário. Construir e apresentar uma matriz C de mesma dimensão e número de elementos que contenha a soma dos elementos da matrizes A e B.

9 – Estruturas de Dados Heterogêneas

Nos três tópicos anteriores, foram estudadas técnicas de programação que envolvem o uso de estruturas de dados homogêneas, com a utilização de matrizes de uma e duas dimensões. Observou-se que somente foi possível trabalhar com um tipo de dado por matriz. No momento em que se precisou trabalhar com dois tipos de dados diferentes foi necessária a utilização também de duas matrizes, sendo uma de cada tipo.

9.1 - Estrutura de um Registro

Com a utilização de registro é possível trabalhar com vários tipos de dados diferentes (os campos) em uma mesma estrutura. Por esta razão, este tipo de dado é considerado heterogêneo.

Em um exercício anterior se solicitou que fosse informado o nome de um aluno e de suas 4 notas bimestrais, o que obrigou a utilização de duas matrizes, sendo uma para conter os nomes por seus valores serem do tipo literal, e a outra para conter as notas por seus valores serem do tipo real. Imagine como seria mais fácil agrupar os dois tipos de dados em uma mesma estrutura. É exatamente isto que se consegue

fazer com a utilização de registros. A figura abaixo mostra um exemplo do layout de um registro com as suas informações, as quais recebem o nome de campos.

Cadastro de Notas Escolares

Nome.....: _____

Primeira Nota....: _____

Segunda Nota....: _____

Terceira Nota....: _____

Quarta Nota.....: _____

Sendo assim, o registro anterior é formado pelos campos: Nome, Primeira Nota, Segunda Nota, Terceira Nota e Quarta Nota e pode ser chamado de Aluno.

9.1.1 - Atribuição de Registros

Os tipos registro devem ser declarados ou atribuídos antes das variáveis, pois pode ocorrer a necessidade de declarar uma variável com o tipo registro anteriormente atribuído. A declaração de um registro é citada no algoritmo em português estruturado, mas não no diagrama de blocos, que só fará menção à utilização de um determinado campo da estrutura heterogênea definida.

Para que seja declarado um tipo registro em português estruturado, deve ser utilizada a instrução tipo em conjunto com a instrução **registro...fimregistro**, conforme sintaxe indicada a seguir.

```

tipo
    <identificador> = registro
                        <lista dos campos e seus tipos>
                        fimregistro

var
    <variáveis> : <identificador>
    
```

Em que *identificador* é o nome do tipo registro em caracteres maiúsculos e *lista dos campos e seus tipos* é a relação de variáveis que serão usadas como campos, bem como o seu tipo de estrutura de dados, podendo ser real, inteiro, lógico ou literal.

Após a instrução **var**, deverá ser indicada a variável tipo registro e a declaração do seu tipo de acordo com um identificador definido anteriormente. Perceba que a instrução **tipo** deverá ser utilizada antes da instrução **var**, pois ao definir um tipo de variável, pode-se fazer uso deste tipo definido.

Note que para a sintaxe anterior não é apresentada sua forma gráfica no diagrama de blocos. Isto ocorre uma vez que este tipo de citação não é indicado dentro do diagrama. Observe que todas as variáveis citadas com a instrução **var** também não são indicadas, ou seja, tudo o que é indicado antes da instrução **inicio** em português estruturado não é mencionado de forma direta dentro do diagrama de blocos.

Tomando como exemplo a proposta de criar um registro denominado ALUNO, cujos campos são NOME, NOTA1, NOTA2, NOTA3 e NOTA4, ele deve ser assim declarado:

```

tipo
    CAD_ALUNO = registro
                NOME : literal
                NOTA1: real
                NOTA2: real
                NOTA3: real
                NOTA4: real
                fimregistro

var
    ALUNO : CAD_ALUNO
    
```

Observe que é especificado um registro denominado CAD_ALUNO, o qual é um conjunto de dados heterogêneos (um campo tipo caractere e quatro do tipo real). Desta forma é possível guardar em uma mesma estrutura vários tipos diferentes de dados.

Grosso modo, pode-se dizer que um tipo registro é também um vetor (matriz de uma dimensão), pois se tem a variável ALUNO do tipo CAD_ALUNO como um vetor com os índices NOME, NOTA1, NOTA2, NOTA3 e NOTA4.

9.1.2 - Leitura e Escrita de Registros

A leitura de um registro é efetuada com a instrução *leia* seguida do nome da variável registro e seu campo correspondente separado por um caractere "." (ponto). Assim sendo, observe em seguida o código em português estruturado:

```

Algoritmo "Leitura"
tipo
    CAD_ALUNO = registro
                NOME : literal
                NOTA1: real
                NOTA2: real
                NOTA3: real
                NOTA4: real
    fimregistro

var
    ALUNO : CAD_ALUNO
inicio
    leia(ALUNO.NOME)
    leia(ALUNO.NOTA1)
    leia(ALUNO.NOTA2)
    leia(ALUNO.NOTA3)
    leia(ALUNO.NOTA4)
fimalgoritmo

```

Uma leitura de registros também pode ser feita com o comando *leia (ALUNO)*. Neste caso, está sendo feita uma leitura do tipo genérica, em que todos os campos estão sendo referenciados implicitamente. A forma explícita apresentada anteriormente é mais legível, uma vez que se faz referência a cada um dos campos em específico.

O processo de escrita de um registro é feito com a instrução *escreva()* de forma semelhante ao processo de leitura.

9.2 - Estrutura de um Registro de Vetor

No tópico anterior, foi apresentado o conceito de trabalhar com um registro. No ponto de aprendizado em que o leitor se encontra, é possível que esteja se perguntando: Será que não é possível definir um vetor ou mesmo uma matriz dentro de um registro, para não ser necessário utilizar somente os tipos primitivos de dados? Isto é realmente possível. Considere ainda o exemplo do registro ALUNO, em que temos o campo NOME tipo caractere e mais quatro campos tipo real para o armazenamento de suas notas, sendo NOTA1, NOTA2, NOTA3 e NOTA4. Veja que é possível definir um vetor chamado NOTA com quatro índices, um para cada nota. A figura a seguir mostra um layout desta nova proposta.

Cadastro de Notas Escolares

Nome.....: _____

Nota			
1	2	3	4

9.2.1 - Atribuição de Registros de Vetores (Matrizes)

Tomando como exemplo a proposta de criar um registro denominado ALUNO, cujas notas serão informadas em um vetor, ele deve ser assim declarado:

```
tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : CAD_ALUNO
```

Observe que ao ser especificado o registro CAD_ALUNO, existe nele um campo chamado NOTA do tipo BIMESTRE, sendo BIMESTRE a especificação de um tipo de conjunto matricial de uma única dimensão com capacidade para quatro elementos. Veja que o tipo bimestre foi anteriormente definido, pois se caracteriza por um tipo criado, assim como o tipo CAD_ALUNO atribuído à variável de registro ALUNO.

9.2.2 - Leitura e Escrita de Registros Contendo Vetores (Matrizes)

A leitura de um registro de vetores é efetuada com a instrução *leia()*, enquanto a escrita é feita com a instrução *escreva()*, geralmente dentro de um laço de repetição. Assim sendo, observe o código em português estruturado abaixo que representa a leitura do nome e das quatro notas bimestrais do aluno e a seguinte exibição destes valores.

algoritmo "Leitura e Escrita"

```
tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : CAD_ALUNO
    i      : inteiro

inicio
    // Leitura
    leia(ALUNO.NOME)
    para i de 1 até 4 passo 1 faça
        leia(ALUNO.NOTA[i])
    fimpara

    // Escrita
    escreva(ALUNO.NOME)
    para i de 1 até 4 passo 1 faça
        escreva(ALUNO.NOTA[i])
    fimpara
fimalgoritmo
```

9.3 - Estrutura de um Vetor de Registros

Com as técnicas de programação anteriormente apresentadas, passou-se a ter uma mobilidade bastante grande, podendo trabalhar de uma forma mais adequada com diversos problemas, principalmente os que envolvem a utilização de dados heterogêneos, facilitando a construção de programas mais eficientes. Porém, os programas apresentados até aqui com a utilização de registros só fizeram menção à leitura e escrita de um único registro.

Neste momento, o estudante terá contato com um vetor de registros que permite a construção de programas, em que é possível fazer a entrada, processamento e saída de diversos registros.

9.3.1 - Atribuição de Vetor de Registros

Para declarar um vetor de registros, é necessário, em primeiro lugar, possuir a definição de um registro, ou seja, é necessário ter o formato de um único registro para então definir o número de registros que será utilizado pelo programa que leia o nome e as quatro notas escolares de 8 alunos. Isto já é familiar. Veja em seguida, a definição do tipo registro e também a definição do conjunto de registros para 8 alunos:

```
tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : vetor [1..8] de CAD_ALUNO
```

Observe que após a instrução **var**, é indicada a variável de registro **ALUNO**, sendo esta um vetor de 8 registros do tipo **CAD_ALUNO** que, por sua vez, é formado de dois tipos de dados: o nome como caractere e a nota como **BIMESTRE**. **BIMESTRE**, por sua vez, é um vetor de quatro valores reais.

9.3.2 - Leitura e Escrita de um Vetor de Registros

A leitura e a escrita serão efetuadas de forma semelhante às anteriores. No entanto, serão utilizados dois laços, pois além de controlar a entrada das quatro notas de cada aluno, é necessário controlar a entrada de oito alunos. Esta estrutura é bastante similar a uma matriz de duas dimensões. Assim sendo, observe em seguida o código em português estruturado.

```
algoritmo "Leitura e Escrita"
tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : vetor [1..8] de CAD_ALUNO
    i, j : inteiro

inicio
    // Leitura
    para i de 1 até 8 passo 1 faça
        leia(ALUNO[i].NOME)
        para j de 1 até 4 passo 1 faça
            leia(ALUNO[i].NOTA[j])
        fimpara
    fimpara

    // Escrita
    para i de 1 até 8 passo 1 faça
        escreva(ALUNO[i].NOME)
        para j de 1 até 4 passo 1 faça
            escreva(ALUNO[i].NOTA[j])
        fimpara
    fimpara
fimalgoritmo
```

9.4 - Exercício de Aprendizagem

1º Exemplo

Efetuar a leitura das 4 notas bimestrais de 8 alunos, apresentando no final os dados dos alunos classificados por nome.

Algoritmo

O algoritmo de ordenação é o mesmo de antes e será aplicado da mesma forma, mas se faz necessário estabelecer alguns critérios.

Por se tratar de uma ordenação, é necessário estabelecer mais duas variáveis, as quais podem ser ATUAL e PROXIMO. Deverá ser também estabelecida uma variável de auxílio à troca, a qual poderá ser a variável X, porém deverá ser do tipo registro.

A ordenação será efetuada com base no nome de cada aluno e quando estiver fora da ordem, os dados deverão ser trocados de posição.

Algoritmo "Leitura Ordenação Escrita"

```

tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro

var
    ALUNO : vetor [1..8] de CAD_ALUNO
    i, j, atual, próximo : inteiro
    x: CAD_ALUNO

inicio
    // Leitura
    para i de 1 até 8 passo 1 faça
        leia(ALUNO[i].NOME)
        para j de 1 até 4 passo 1 faça
            leia(ALUNO[i].NOTA[j])
        fimpara
    fimpara

    // Ordenação
    para atual de 1 até 7 passo 1 faça
        para próximo de atual+1 até 8 passo 1 faça
            se (ALUNO[atual].NOME > ALUNO[próximo].NOME) então
                X <- ALUNO[atual]
                ALUNO[atual] <- ALUNO[próximo]
                ALUNO[próximo] <- X
            fimse
        fimpara
    fimpara

    // Escrita
    para i de 1 até 8 passo 1 faça
        escreva(ALUNO[i].NOME)
        para j de 1 até 4 passo 1 faça
            escreva(ALUNO[i].NOTA[j])
        fimpara
    fimpara
fimalgoritmo

```

2º Exemplo

Deverá ser criado um programa que efetue a leitura de uma tabela de cargos e salários. Em seguida, o programa deverá solicitar que seja fornecido o código de um determinado cargo. Esse código deverá estar entre 1 e 17. O operador do programa poderá fazer quantas consultas desejar. Sendo o código inválido, o programa deve avisar o operador da ocorrência. Para dar entrada no código de cargos/salários, observe a tabela seguinte:

Código	Cargo	Salário
1	Analista de Salários	9.00
2	Auxiliar de Contabilidade	6.25
3	Chefe de Cobrança	8.04
4	Chefe de Expedição	8.58
5	Contador	15.60
6	Gerente de Divisão	22.90

7	Escriturário	5.00
8	Faxineiro	3.20
9	Gerente Administrativo	10.30
10	Gerente Comercial	10.40
11	Gerente de Pessoal	10.29
12	Gerente de Treinamento	10.68
13	Gerente Financeiro	16.54
14	Contínuo	2.46
15	Operador de Micro	6.05
16	Programador	9.10
17	Secretária	7.31

Algoritmo

Observe os breves passos que o programa deverá executar. No tocante à pesquisa, será usado o método de pesquisa seqüencial:

1. A tabela em questão é formada por três tipos de dados: o código como inteiro, o cargo como literal e o salário como real. Criar um registro com tal formato;
2. Cadastrar os elementos da tabela. Para facilitar, o código será fornecido automaticamente no momento do cadastramento;
3. Criar um looping para executar as consultas enquanto o operador desejar;
4. Pedir o código do cargo; se válido, apresentar o cargo e o salário;
5. Se o código for inexistente, apresentar mensagem ao operador;
6. Saber do usuário se ele deseja continuar com as consultas; se sim, repetir os passos 3, 4 e 5; se não, encerrar o programa.

Algoritmo "Tabela_de_Salarios"

tipo

```
DADOS = registro
        CODIGO : inteiro
        CARGO  : literal
        SAL    : real
        Fimregistro
```

var

```
TABELA: vetor [1..17] de DADOS
i, codigo: inteiro
resp: literal
acha: lógico
```

inicio

```
// Entrada de dados
para i de 1 até 17 passo 1 faça
    TABELA[i].CODIGO <- i
    escreva("Código ....:", TABELA[i].CODIGO)
    leia(TABELA[i].CARGO)
    escreva("Cargo: ....:", TABELA[i].CARGO)
    leia(TABELA[i].SAL)
    escreva("Salário: ...:", TABELA[i].SAL)
fimpara

// Trecho de pesquisa seqüencial
resp <- "sim"
enquanto (resp = "sim") faça
    escreva("Qual o código (1 a 17)?")
    leia(codigo)
    i <- 1
    acha <- falso
    enquanto (i <= 17) e (acha = falso) faça
        se (codigo = TABELA[i].CODIGO) então
            acha <- verdadeiro
        senão
            i <- i + 1
    fimse
    fimenquanto
    se (acha = verdadeiro) então
        escreva("Cargo:      ", TABELA[i].CARGO)
        escreva("Salário:    ", TABELA[i].SAL)
```

```
senão
    escreva("Cargo INEXISTENTE").
fimse
escreva("Deseja continuar a pesquisa?")
leia(resp)
fimenquanto
fimalgoritmo
```

9.5 - Exercícios de Entrega Obrigatória (até ____/____/____) **(Lista 09)**

1) Considerando a necessidade de desenvolver uma agenda que contenha nomes, endereços e telefones de 10 pessoas, defina a estrutura de registro apropriada, o diagrama de blocos e a codificação de um programa que por meio do uso de um menu de opções, execute as seguintes etapas (Este programa deverá ser salvo com o nome L09_1):

- a) Cadastrar os 10 registros.
- b) Pesquisar um dos 10 registros de cada vez pelo campo nome (usar o método seqüencial).
- c) Classificar por ordem de nome os registros cadastrados.
- d) Apresentar todos os registros.
- e) Sair do programa de cadastro.

2) Considerando a necessidade de um programa que armazene o nome e as notas bimestrais de 20 alunos do curso de Técnicas de Programação, defina a estrutura de registro apropriada, o diagrama de blocos e a codificação de um programa que, por meio do uso de um menu de opções, execute as seguintes etapas (Este programa deverá ser salvo com o nome L09_2):

- a) Cadastrar os 20 registros (após o cadastro efetuar a classificação por nome).
- b) Pesquisar os 20 registros, de cada vez, pelo campo nome. Nesta pesquisa o programa deverá também apresentar a média do aluno e as mensagens: "Aprovado" caso sua média seja maior ou igual a 5, ou "Reprovado" para média abaixo de 5.
- c) Apresentar todos os registros, médias e a mensagem de aprovação ou reprovação.
- d) Sair do programa de cadastro.

3) Elaborar um programa que armazene o nome e a altura de 15 pessoas, por meio do uso de registros. O programa deverá ser manipulado por um menu que execute as seguintes etapas (Este programa deverá ser salvo com o nome L09_3):

- a) Cadastrar os 15 registros.
- b) Apresentar os registros (nome e altura) das pessoas menores ou iguais a 1.5m.
- c) Apresentar os registros (nome e altura) das pessoas que sejam maiores que 1.5m.
- d) Apresentar os registros (nome e altura) das pessoas que sejam maiores que 1.5m e menores que 2.0m.
- e) Apresentar a média extraída de todas as alturas armazenadas.
- f) Sair do programa.

4) Considerando os registros de 20 funcionários, contendo os campos: matrícula, nome e salário, desenvolver um programa que, por meio de um menu, execute as seguintes etapas (Este programa deverá ser salvo com o nome L09_4):

- a) Cadastrar os 20 empregados e classificar os registros por número de matrícula.
- b) Pesquisar um determinado empregado pelo número de matrícula.
- c) Apresentar de forma ordenada (por matrícula) os registros dos empregados que recebem salários acima de R\$1.000,00.

- d) Apresentar de forma ordenada (por matrícula) os registros dos empregados que recebem salários abaixo de R\$1.000,00.
- e) Apresentar de forma ordenada (por matrícula) os registros dos empregados que recebem salários iguais a R\$1.000,00.
- f) Sair do programa.

9.6 - Exercícios de Extras

1) Como citado em aula, o programa Visualg não trabalha com estruturas heterogêneas, porém é possível com um esforço maior de lógica transformar algoritmos que usam estruturas heterogêneas para algoritmos que usam somente estruturas homogêneas. Caso você goste de desafios que tal transformar os algoritmos anteriores que usam estruturas heterogêneas para estruturas homogêneas.

Para auxiliar, veja abaixo como podem ser convertida a declaração das estruturas:

Estrutura Heterogênea:

```
tipo
  CAD_SALARIO = vetor[1..4] de real
  CAD_EMPREGADO = registro
    NOME: literal
    NOTA: CAD_SALARIO
  fimregistro

var
  empregado: vetor [1..5] de CAD_EMPREGADO
```

Estrutura Homogênea:

```
var
  empregado_NOME: vetor [1..5] de literal
  empregado_SALARIO: vetor [1..5,1..4] de real
```

10 – Utilização de Sub-rotinas

A partir deste capítulo será estudada a aplicação de sub-rotinas em algoritmos, também conhecidas pela denominação de módulos subprogramas, ou subalgoritmos. Na realidade, não importa como são chamadas, o que importa é a forma com funcionam e como devem ser aplicadas em um programa, e é isto que você aprenderá a partir deste ponto.

Neste capítulo, será introduzido o conceito da criação estruturada de programas, pois para escrever um programa de computador necessita-se de estudo (levantamento de todas as necessidades e detalhes do que deverá ser feito) e metodologia (regras básicas que deverão ser seguidas). Sem a aplicação de métodos não será possível resolver grandes problemas; quando muito, pequenos problemas.

10.1 – As Sub-rotinas

No geral, problemas complexos exigem algoritmos complexos. Mas sempre é possível dividir um problema grande em problemas menores. Desta forma, cada parte menor tem um algoritmo mais simples, e é esse trecho menor que é chamado de sub-rotina. Uma sub-rotina é na verdade um programa, e sendo um programa poderá efetuar diversas operações computacionais (entrada, processamento e saída) e deverá ser tratada como foram os programas projetados até este momento. As sub-rotinas são utilizadas na divisão de algoritmos complexos, permitindo assim possuir a modularização de um determinado problema, considerado grande e de difícil solução.

Ao trabalhar com esta técnica, pode-se deparar com a necessidade de dividir uma sub-rotina em outras tantas quantas forem necessárias, buscando uma solução mais simples de uma parte do problema maior. O processo de dividir sub-rotinas em outras é denominado *Método de Refinamento Sucessivo*.

10.2 – Método Top-Down

O processo de programar um computador torna-se bastante simples quando aplicado o método de utilização de sub-rotinas (módulos de programas). Porém, a utilização dessas sub-rotinas deverá ser feita com aplicação do método *top-down*.

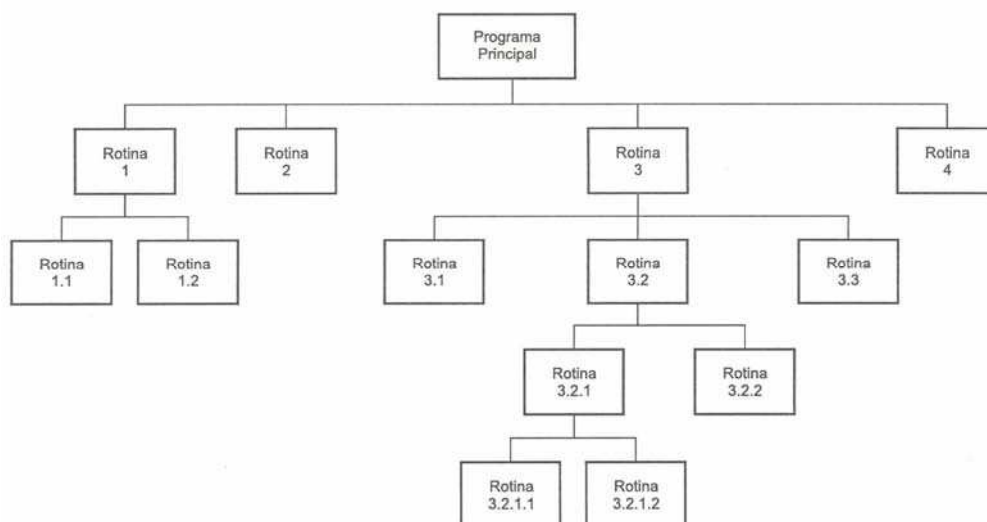
Um método bastante adequado para a programação de um computador é trabalhar com o conceito de programação estruturada, pois a maior parte das linguagens de programação utilizadas atualmente também são, o que facilita a aplicação deste processo de trabalho. O método mais adequado para a programação estruturada é o *Top-Down* (de cima para baixo) o qual se caracteriza basicamente por:

- Antes de iniciar a construção do programa, o programador deverá ter em mente as tarefas principais que este deverá executar. Não é necessário saber como funcionarão, somente saber quantas são.
- Conhecidas todas as tarefas a serem executadas, tem-se em mente como deverá ser o *programa principal*, o qual vai controlar todas as outras tarefas distribuídas em suas sub-rotinas.
- Tendo definido o programa principal, é iniciado o processo de detalhamento para cada sub-rotina. Desta forma são definidos vários algoritmos, um para cada rotina em separado, para que se tenha uma visão do que deverá ser executado em cada módulo de programa. Existem programadores que estabelecem o número máximo de linhas de programa que uma rotina deverá possuir. Se o número de linhas ultrapassa o limite preestabelecido, a rotina em desenvolvimento é dividida em outra sub-rotina (é neste ponto que se aplica o método de refinamento sucessivo).

O método Top-Down faz com que o programa tenha uma estrutura semelhante a um organograma.

A utilização do método “de cima para baixo” permite que seja efetuado cada módulo do programa em separado. Desta forma, cada um pode ser testado separadamente garantindo que o programa completo esteja sem erro ao seu término.

Outro detalhe a ser considerado é que muitas vezes existem em um programa trechos de códigos que são repetidos várias vezes. Esses trechos poderão ser utilizados como sub-rotinas, proporcionando um programa menor e mais fácil de ser alterado num futuro próximo.



A utilização de sub-rotinas e o uso do método Top-Down na programação permite ao programador elaborar rotinas exclusivas. Por exemplo, uma rotina somente para entrada, outra para a parte de processamento e outra para a saída dos dados.

11 – Procedimentos (Sub-Rotinas)

Será estudado neste capítulo um tipo de sub-rotina conhecido como *procedimento*. Existe um outro tipo conhecido como *Função* que será estudado posteriormente. Entre estes dois tipos de sub-rotinas existem algumas diferenças, mas o conceito é o mesmo para ambas. Desta forma, será usada uma nova instrução que identifique em português estruturado cada tipo de sub-rotina, sendo elas: **Procedimento** e **Função**. O importante no uso prático destes dois tipos de sub-rotinas é distinguir as diferenças entre elas e como utiliza-las no momento mais adequado.

Um procedimento é um bloco de programa contendo início e fim e será identificado por um nome, por meio do qual será referenciado em qualquer parte do programa principal ou do programa chamador da rotina. 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 efetuou a chamada da sub-rotina.

Com relação à criação da rotina, será idêntica a tudo o que já foi estudado sobre programação. Na representação do diagrama de blocos, não há quase nenhuma mudança, a não ser pela troca das identificações **início** e **fim** nos símbolos de *terminal* e o novo símbolo *sub-rotina*, que é idêntico ao símbolo de processamento, porém se caracteriza pelas linhas paralelas às bordas esquerda e direita, devendo ser utilizado no programa chamador. A sintaxe em português estruturado será também idêntica ao estudo anterior. Observe em seguida, o código em português estruturado.

```
procedimento <nome do procedimento>
var
    <variáveis>

inicio
    <instruções>
fimprocedimento
```

11.1 – Exercício de Aprendizagem

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

Algoritmo

Note que esse programa deverá ser um conjunto de cinco rotinas, sendo uma principal e quatro secundárias que, por sua vez, pedirão a leitura de dois valores, farão a operação e apresentarão o resultado obtido. A figura a seguir apresenta um organograma com a idéia de hierarquização das rotinas do programa. A quinta opção não se caracteriza por ser uma rotina, apenas a opção que vai encerrar o looping de controle do menu.



Tendo uma idéia da estrutura geral do programa, será escrito em separado cada algoritmo com os seus detalhes de operação. Primeiro o programa principal e depois as outras rotinas, de preferência na mesma ordem em que estão mencionadas no organograma.

Programa Principal

1. Apresentar um menu de seleção com cinco opções: 1 - Adição, 2 - Subtração, 3 - Multiplicação, 4 – Divisão e 5 – Fim de Programa.\\;
2. Ao ser selecionado um valor do menu, a rotina correspondente deverá ser executada;
3. Ao escolher o valor 5, o programa deverá ser encerrado.

Rotina 1 - Adição

1. Ler dois valores, no caso variáveis A e B;
2. Efetuar a soma das variáveis A e B, implicando o seu resultado na variável R;
3. Apresentar o valor da variável R.

Rotina 2 - Subtração

1. Ler dois valores, no caso variáveis A e B;
2. Efetuar a subtração das variáveis A e B, implicando o seu resultado na variável R;
3. Apresentar o valor da variável R.

Rotina 3 - Multiplicação

1. Ler dois valores, no caso variáveis A e B;
2. Efetuar a multiplicação das variáveis A e B, implicando o seu resultado na variável R;
3. Apresentar o valor da variável R.

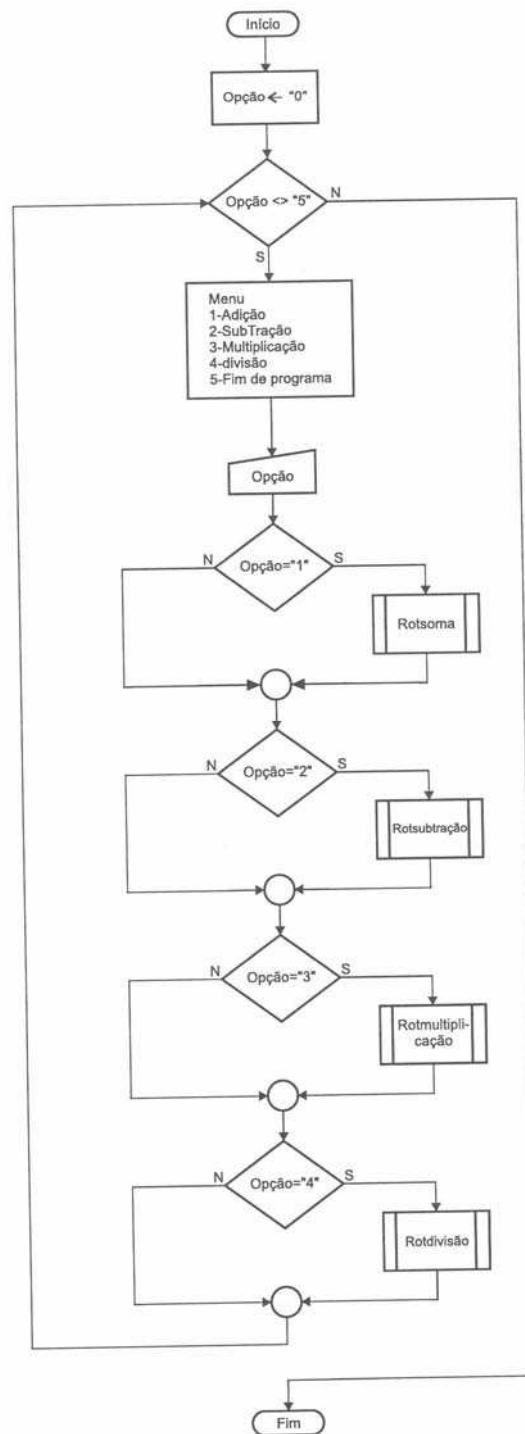
Rotina 4 - Divisão

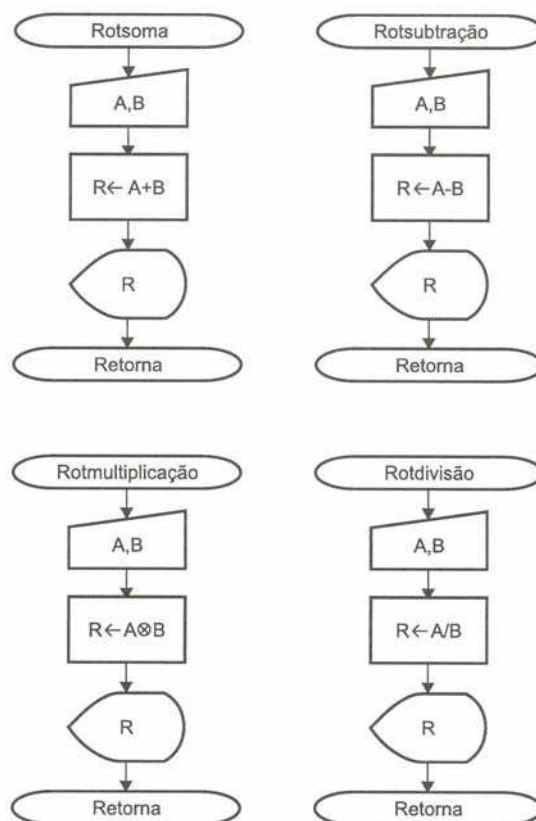
1. Ler dois valores, no caso variáveis A e B;
2. Efetuar a divisão das variáveis A e B, implicando o seu resultado na variável R;
3. Apresentar o valor da variável R.

Observe que em cada rotina serão utilizadas as mesma variáveis, mas elas não serão executadas ao mesmo tempo para todas as operações. Serão utilizadas em separado e somente para a rotina escolhida.

Diagrama de Blocos

Perceba que na diagramação cada rotina é definida em separado como um programa independente. O que muda é a forma de identificação do símbolo *terminal*. Em vez de se utilizarem os termos **início** e **fim**, utilizam-se o nome da sub-rotina para iniciar e a palavra **retornar** para encerrar. Com relação ao módulo principal, ele faz uso do símbolo *sub-rotina* que indica a chamada de uma sub-rotina.





Português Estruturado

Em código português estruturado, serão mencionadas em primeiro lugar as sub-rotinas e por último o programa principal. Quando no programa principal ou rotina chamadora for referenciada uma sub-rotina, ela será sublinhada para facilitar sua visualização. observe no programa a variável OPCA0 para controlar a opção do operador que é do tipo caractere. Outro detalhe a ser observado é o nome de cada rotina mencionado junto da verificação da instrução **se** no módulo de programa principal.

```

algoritmo "Calculadora1"
var
opcao: caracter

//Sub-rotinas de calculo
procedimento ROTSOMA
var
    r, a, b: real
inicio
    escreval("Rotina de SOMA")
    escreval("Entre um valor para A: ")
    leia(a)
    escreva("Entre um valor para B: ")
    leia(b)
    r ← a + b
    escreval("A soma de A com B é = ", r)
    escreval()
fimprocedimento

procedimento ROTSUBTRACAO
var
    r, a, b: real
inicio
    escreva("Rotina de SUBTRACAO")
    escreva("Entre um valor para A: ")
    leia(a)
    escreva("Entre um valor para B: ")
    leia(b)
    r ← a - b
    
```

```

    escreval("A subtração de A com B é = ", r)
    escreval()
fimprocedimento

procedimento ROTMULTIPLICACAO
var
    r, a, b: real
inicio
    escreva("Rotina de MULTIPLICACAO")
    escreva("Entre um valor para A: ")
    leia(a)
    escreva("Entre um valor para B: ")
    leia(b)
    r <- a * b
    escreval("A multiplicação de A com B é = ", r)
    escreval()
fimprocedimento

procedimento ROTDIVISAO
var
    r, a, b: real
inicio
    escreva("Rotina de DIVISAO")
    escreva("Entre um valor para A: ")
    leia(a)
    escreva("Entre um valor para B: ")
    leia(b)
    r <- a / b
    escreval("A divisao de A com B é = ", r)
    escreval()
fimprocedimento

//Programa principal
inicio
    opcao <- "0"
    enquanto (opcao <> "5") faca
        escreval("1 - Adição")
        escreval("2 - Subtração")
        escreval("3 - Multiplicação")
        escreval("4 - Divisão")
        escreval("5 - Fim do programa")
        escreval("Escolha uma opcao: ")
        leia(opcao)
        se (opcao = "1") entao
            ROTSOMA
        fimse
        se (opcao = "2") entao
            ROTSUBTRACAO
        fimse
        se (opcao = "3") entao
            ROTMULTIPLICACAO
        fimse
        se (opcao = "4") entao
            ROTDIVISAO
        fimse
    fimenquanto
fimalgoritmo

```

11.2 – Estrutura de Controle com Múltipla Escolha

O programa anterior apresenta a aplicação da técnica de programação estruturada, permitindo assim construir programas mais elaborados. Porém, no tocante ao selecionamento das sub-rotinas foi utilizada a instrução **se**. Observe que se o programa possuir um menu com 15 opções, deverão ser definidas 15 instruções do tipo **se** para verificar a escolha do operador.

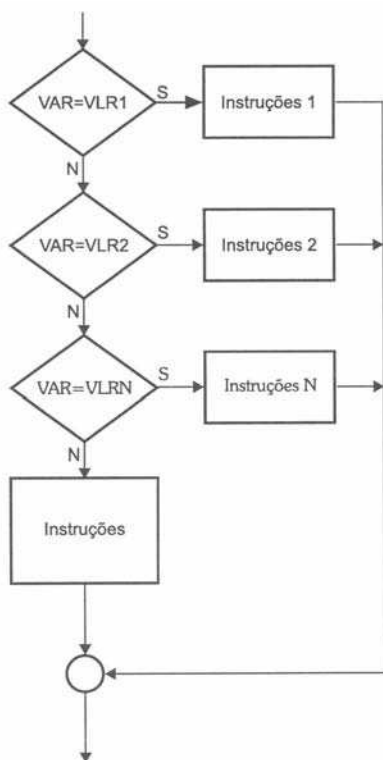
Quando houver a necessidade de construir um programa no qual seja necessário utilizar uma seqüência grande de instruções do tipo **se**, sejam estas uma após a outra ou mesmo encadeadas, poderá ser

simplificado com a utilização da estrutura **escolha...fimescolha**, que possui a seguinte sintaxe:

```
escolha <variável>  
  caso <valor 1>  
    <operação 1>  
  caso <valor 2>  
    <operação 2>  
  caso <valor N>  
    <operação N>  
  outrocaso  
    <operação>  
fimescolha
```

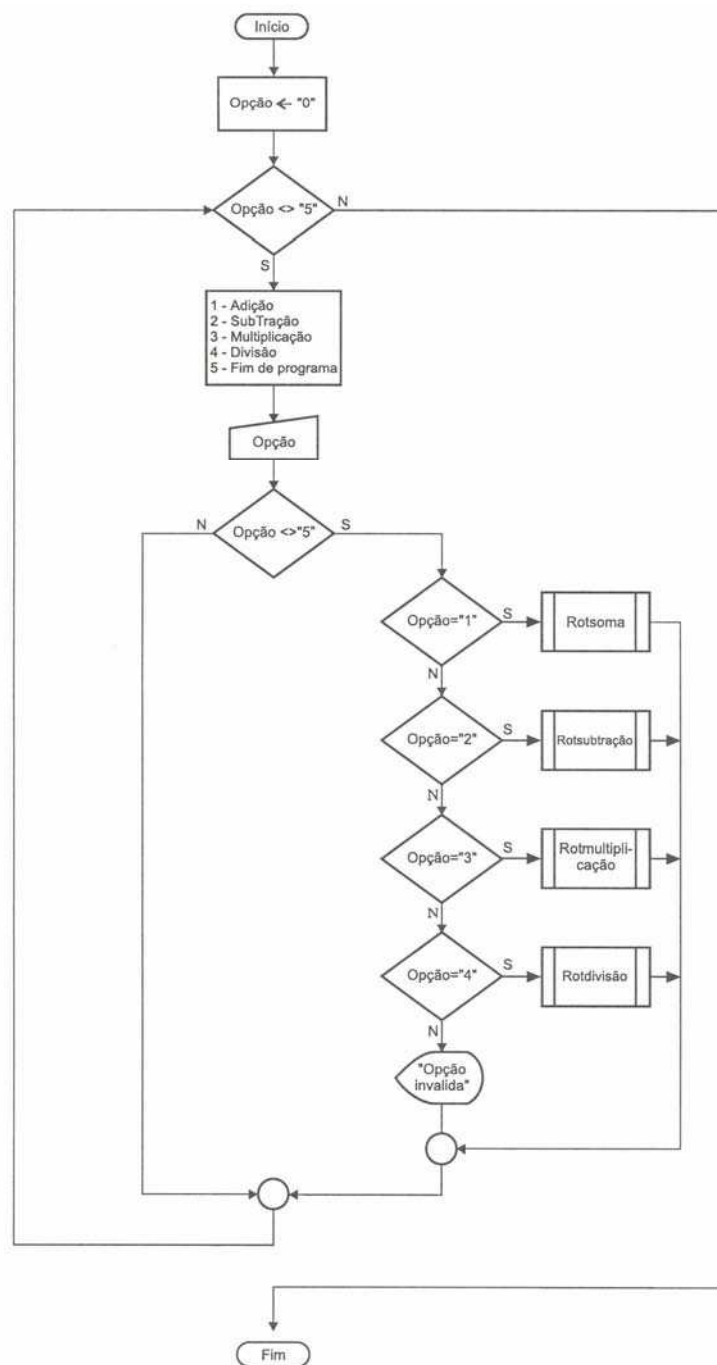
Em que <variável> será a variável a ser controlada, <valor> será o conteúdo de uma variável sendo verificado e <operação> poderá ser a chamada de uma sub-rotina, a execução de qualquer operação matemática ou de qualquer outra instrução.

Diagrama de Blocos



Desta forma, a rotina principal do programa calculadora poderá ser escrita fazendo uso da instrução **escolha...fimescolha** no selecionamento de uma opção escolhida pelo operador, conforme em seguida:

Diagrama de Blocos



Português Estruturado

```

//Programa principal
inicio
    opcao <- "0"
    enquanto (opcao <> "5") faca
        escreval("1 - Adição")
        escreval("2 - Subtração")
        escreval("3 - Multiplicação")
        escreval("4 - Divisão")
        escreval("5 - Fim do programa")
        escreval("Escolha uma opcao: ")
        leia(opcao)
        se (opcao <> "5") entao
            escolha opcao
                caso "1"
                    ROTSOMA
                caso "2"

```

```

    ROTSUBTRACAO
caso "3"
    ROTMULTIPLICACAO
caso "4"
    ROTDIVISAO
outrocaso
    escreval("Opção inválida - Tente novamente.")
fimescolha
fimse
fimenquanto
fimalgoritmo

```

11.3 – Variáveis Globais e Locais

No programa anterior foram utilizadas variáveis dentro das sub-rotinas, no caso as variáveis A, B e R, e fora das sub-rotinas, no caso a variável OPCAO, sem que houvesse a preocupação de agrupar as variáveis de uma forma coerente segundo o seu tipo. São dois os tipos de variáveis utilizadas em um programa: as variáveis *Globais* e as *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. Assim sendo, este tipo de variável passa a ser visível a todas as sub-rotinas hierarquicamente subordinadas à rotina principal, que poderá ser o próprio programa principal ou uma outra sub-rotina. No programa anterior, a variável OPCAO é global, apesar de não estar sendo utilizada nas demais sub-rotinas.

Dependendo da forma como se trabalha com as variáveis, é possível economizar espaço memória, tornando o programa mais eficiente. Considere abaixo dois exemplos em português estruturado, sendo o primeiro sem uso de sub-rotina e o segundo com sub-rotina, fazendo uso de variável global e local:

Exemplo 1

```

algoritmo "Troca_valores_exemplo_1"
var
    X, A, B: inteiro

inicio
    escreva("A= ")
    leia(A)
    escreva("B= ")
    leia(B)

    X <- A
    A <- B
    B <- X

    escreval()
    escreval("A=", A)
    escreval("B=", B)
fimalgoritmo

```

Exemplo 2

```

algoritmo "Troca_valores_exemplo_2"
var
    A, B: inteiro

procedimento TROCA
var
    X: inteiro
inicio
    X <- A
    A <- B
    B <- X
fimprocedimento

```

```

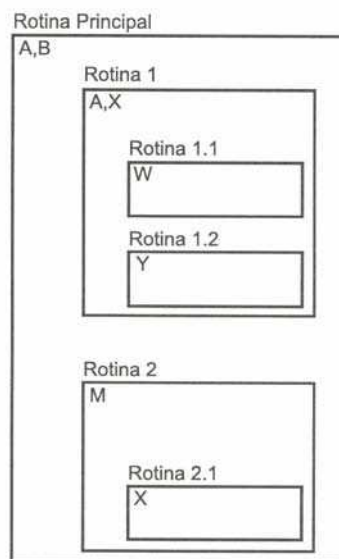
inicio
  escreva("A= ")
  leia(A)
  escreva("B= ")
  leia(B)
  TROCA
  escreval()
  escreval("A=", A)
  escreval("B=", B)
fimalgoritmo

```

Observe que apesar de o segundo exemplo ser um pouco maior que o primeiro, ele possibilita uma economia de espaço em memória bastante interessante, pois o espaço a ser utilizado pela variável X é somente solicitado quando a sub-rotina TROCA é executada. Terminada a execução, a variável X é desconsiderada, ficando em uso somente os espaços reservados para as variáveis globais A e B.

11.3.1 – Escopo de Variáveis

O escopo de uma variável ou sua abrangência está vinculado a sua visibilidade (*Global* ou *Local*) em relação às sub-rotinas de um programa, sendo que a sua visibilidade está relacionada a sua hierarquia. Mas existe um detalhe importante a ser considerado, pois uma variável poderá ser considerada global para todas as sub-rotinas inferiores a uma rotina principal, e dentro de uma dessas sub-rotinas a mesma variável poderá estar sendo utilizada como local. Observe o esquema apresentado a seguir.



Com relação ao gráfico anterior, as variáveis A e B da rotina principal são num primeiro momento globais às sub-rotinas 1 e 2. Porém, dentro da sub-rotina 1, a variável A é definida novamente, assumindo assim um contexto local para esta sub-rotina (é como se tivesse utilizando uma nova variável, no caso A'), mas será global para as sub-rotinas 1.1 e 1.2 com relação à variável A' que também terá como global a variável X. As variáveis W e Y que respectivamente pertencem às sub-rotinas 1.1 e 1.2 são locais. A variável B é global para as sub-rotinas 1, 1.1 e 1.2.

Para a sub-rotina 2 as variáveis A e B da rotina principal são globais e serão também visualizadas como globais para a rotina 2.1. A rotina 2 possui uma variável local M que é considerada global para a rotina 2.1 que faz referência à variável X local a esta sub-rotina, não tendo nenhuma referência com a variável X da sub-rotina 1.

11.3.2 – Refinamento Sucessivo

O refinamento sucessivo é uma técnica de programação que possibilita dividir uma sub-rotina em outras sub-rotinas. Deve ser aplicado com muito critério para que o programa a ser construído não se torne desestruturado e difícil de ser compreendido por você ou por outras pessoas.

O programa calculador apresentado anteriormente permite que seja aplicada esta técnica, pois existem nas quatro sub-rotinas de cálculo, instruções que efetuam as mesmas tarefas. Por exemplo: entrada e saída são efetuadas com as mesmas variáveis. Observe que as variáveis A, B e R são definidas quatro vezes, uma em cada sub-rotina.

A solução é definir as variáveis A, B e R como globais e construir mais duas sub-rotinas, uma para entrada e outra para a saída. As quatro sub-rotinas atuais serão diminuídas em número de linhas, pois tudo o que se repete nas sub-rotinas será retirado.

Observe a declaração das variáveis A, B e R como globais e a definição e chamada das duas novas sub-rotinas ENTRADA e SAIDA. Perceba que nas sub-rotinas de cálculos não está sendo declarada nenhuma variável, uma vez que agora estão fazendo uso do conceito de variáveis globais.

```
algoritmo "Calculadora3"
var
  OPCA0: character
  R, A, B: real

{Sub-rotinas de entrada e saída}
procedimento ENTRADA
inicio
  escreva("Entre um valor para A: ")
  leia(A)
  escreva("Entre um valor para B: ")
  leia(B)
fimprocedimento

procedimento SAIDA
inicio
  escreval("O resultado de A com B é = ", R)
  escreval()
fimprocedimento

{Sub-rotinas de calculo}
procedimento ROTSOMA
inicio
  escreval("Rotina de SOMA")
  ENTRADA
  R <- A + B
  SAIDA
fimprocedimento

procedimento ROTSUBTRACAO
inicio
  escreval("Rotina de SUBTRACAO")
  ENTRADA
  R <- A - B
  SAIDA
fimprocedimento

procedimento ROTMULTIPLICACAO
inicio
  escreval("Rotina de MULTIPLICACAO")
  ENTRADA
  R <- A * B
  SAIDA
fimprocedimento

procedimento ROTDIVISAO
inicio
  escreval("Rotina de DIVISAO")
```



```

ENTRADA
R <- A / B
SAIDA
fimprocedimento

{Programa principal}
inicio
OPCAO <- "0"
enquanto (OPCAO <> "5") faca
  escreval("1 - Adição")
  escreval("2 - Subtração")
  escreval("3 - Multiplicação")
  escreval("4 - Divisão")
  escreval("5 - Fim do programa")
  escreval("Escolha uma opcao: ")
  leia(OPCAO)
  se (OPCAO <> "5") entao
    escolha OPCA0
    caso "1"
      ROTSOMA
    caso "2"
      ROTSUBTRACAO
    caso "3"
      ROTMULTIPLICACAO
    caso "4"
      ROTDIVISAO
    outrocaso
      escreval("Opção inválida - Tente novamente.")
    fimescolha
  fimse
fimenquanto
fimalgoritmo

```

11.4 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 10)

Utilizando-se das técnicas de programação top-down, da utilização de subalgoritmos (procedimentos) e a da estrutura **escolha . . . caso** efetue os exercícios a seguir.
Atenção observe que o método de resolução é **diferente** do da lista 09.

1) Considerando a necessidade de desenvolver uma agenda que contenha nomes, endereços e telefones de 10 pessoas, defina a estrutura de registro apropriada, o diagrama de blocos e a codificação de um programa que por meio do uso de um menu de opções, execute as seguintes etapas (Este programa deverá ser salvo com o nome L10_1):

- a) Cadastrar os 10 registros.
- b) Pesquisar um dos 10 registros de cada vez pelo campo nome (usar o método seqüencial).
- c) Classificar por ordem de nome os registros cadastrados.
- d) Apresentar todos os registros.
- e) Sair do programa de cadastro.

2) Considerando a necessidade de um programa que armazene o nome e as notas bimestrais de 20 alunos do curso de Técnicas de Programação, defina a estrutura de registro apropriada, o diagrama de blocos e a codificação de um programa que, por meio do uso de um menu de opções, execute as seguintes etapas (Este programa deverá ser salvo com o nome L10_2):

- a) Cadastrar os 20 registros (após o cadastro efetuar a classificação por nome).
- b) Pesquisar os 20 registros, de cada vez, pelo campo nome. Nesta pesquisa o programa deverá também apresentar a média do aluno e as mensagens: "Aprovado" caso sua média seja maior ou igual a 5, ou "Reprovado" para média abaixo de 5.

- c) Apresentar todos os registros, médias e a mensagem de aprovação ou reprovação.
- d) Sair do programa de cadastro.

3) Elaborar um programa que armazene o nome e a altura de 15 pessoas, por meio do uso de registros. O programa deverá ser manipulado por um menu que execute as seguintes etapas (Este programa deverá ser salvo com o nome L10_3):

- a) Cadastrar os 15 registros.
- b) Apresentar os registros (nome e altura) das pessoas menores ou iguais a 1.5m.
- c) Apresentar os registros (nome e altura) das pessoas que sejam maiores que 1.5m.
- d) Apresentar os registros (nome e altura) das pessoas que sejam maiores que 1.5m e menores que 2.0m.
- e) Apresentar a média extraída de todas as alturas armazenadas.
- f) Sair do programa.

4) Considerando os registros de 20 funcionários, contendo os campos: matrícula, nome e salário, desenvolver um programa que, por meio de um menu, execute as seguintes etapas (Este programa deverá ser salvo com o nome L10_4):

- a) Cadastrar os 20 empregados e classificar os registros por número de matrícula.
- b) Pesquisar um determinado empregado pelo número de matrícula.
- c) Apresentar de forma ordenada (por matrícula) os registros dos empregados que recebem salários acima de R\$1.000,00.
- d) Apresentar de forma ordenada (por matrícula) os registros dos empregados que recebem salários abaixo de R\$1.000,00.
- e) Apresentar de forma ordenada (por matrícula) os registros dos empregados que recebem salários iguais a R\$1.000,00.
- f) Sair do programa.

11.5 - Exercícios de Extras

1) Como citado em aula, o programa Visualg não trabalha com estruturas heterogêneas, porém é possível com um esforço maior de lógica transformar algoritmos que usam estruturas heterogêneas para algoritmos que usam somente estruturas homogêneas. Caso você goste de desafios que tal transformar os algoritmos anteriores que usam estruturas heterogêneas para estruturas homogêneas.

Para auxiliar, veja abaixo como podem ser convertida a declaração das estruturas:

Estrutura Heterogênea:

```
tipo
  CAD_SALARIO = vetor[1..4] de real
  CAD_EMPREGADO = registro
    NOME: literal
    NOTA: CAD_SALARIO
  fimregistro

var
  empregado: vetor [1..5] de CAD_EMPREGADO
```

Estrutura Homogênea:

```
var
  empregado_NOME: vetor [1..5] de literal
  empregado_SALARIO: vetor [1..5,1..4] de real
```

12 – Utilização de Parâmetros

Os parâmetros têm por finalidade servir como um ponto de comunicação bidirecional entre uma sub-rotina e o programa principal ou uma outra sub-rotina hierarquicamente de nível mais alto. Desta forma, é possível passar valores de uma sub-rotina ou rotina chamadora à outra sub-rotina e vice-versa, utilizando parâmetros que podem ser *formais* ou *reais*.

12.1 - Parâmetros Formais e Reais

Serão considerados parâmetros *formais* quando forem declarados por meio de variáveis juntamente com a identificação do nome da sub-rotina, os quais serão tratados exatamente da mesma forma que são tratadas as variáveis globais ou locais. Considere como exemplo de parâmetros *formais* o código em português estruturado da sub-rotina apresentado abaixo:

```
Procedimento CALCSOMA (A, B: inteiro)
var
    Z : inteiro
inicio
    Z <- A + B
    escreva(Z)
fimprocedimento
```

Observe que a variável Z é local e está sendo usada para armazenar a soma das variáveis A e B que representam os parâmetros formais da sub-rotina CALCSOMA.

Serão considerados parâmetros *reais* quando substituírem os parâmetros formais, quando da utilização da sub-rotina por um programa principal ou por uma rotina chamadora. Considere como exemplo de parâmetros *reais* o código em português estruturado do programa que faz uso da sub-rotina CALCSOMA apresentado em seguida:

```
inicio
    leia(x, y)
    CALCSOMA(x, y)
    leia(w, t)
    CALCSOMA(w, t)
    CALCSOMA(8, 2)
fimalgoritmo
```

No trecho acima, toda vez que a sub-rotina CALCSOMA é chamada, faz-se uso de parâmetros *reais*. Desta forma, são parâmetros *reais* as variáveis x, y, w e t, pois seus valores são fornecidos pela instrução **leia()** e também os valores 8 e 2.

12.2 - Passagem de Parâmetros

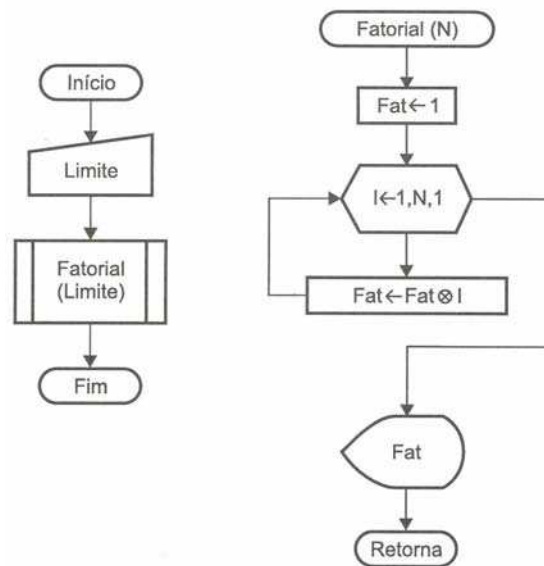
A passagem de parâmetro ocorre quando é feita uma substituição dos parâmetros formais pelos reais no momento da execução da sub-rotina. Esses parâmetros são passados por variáveis de duas formas: por valor e por referência.

12.2.1 – Por Valor

A passagem de parâmetro por valor caracteriza-se pela não-alteração do valor do parâmetro real quando o parâmetro formal é manipulado dentro da sub-rotina. Assim sendo, o valor passado pelo parâmetro real é copiado para o parâmetro formal, que no caso assume o papel de variável local da sub-rotina. Desta forma, qualquer modificação que ocorra na variável local da sub-rotina não afetará o valor do parâmetro real

correspondente, ou seja, o processamento é executado somente dentro da sub-rotina, ficando o resultado obtido “preso” na sub-rotina. Como exemplo deste tipo de parâmetro considere o mostrado em seguida:

Diagrama de Blocos



Português Estruturado

```

algoritmo "Calc_Fatorial_V1"

procedimento FATORIAL(N: inteiro)
var
    i, FAT: inteiro
inicio
    FAT ← 1
    para i de 1 ate N faca
        FAT ← FAT * i
    fimpara
    escreva(FAT)
fimprocedimento

var
    LIMITE: inteiro
inicio
    escreva("Qual o fatorial: ")
    leia(LIMITE)
    FATORIAL(LIMITE)
fimalgoritmo
    
```

Neste exemplo, é indicado o uso da passagem de parâmetro por valor. No caso, a variável N é o parâmetro formal, que receberá o valor fornecido à variável LIMITE por meio da sub-rotina FATORIAL. Esse valor estabelece o número de vezes que o looping deve ser executado. Dentro do procedimento é encontrada a variável FAT que irá realizar um efeito de acumulador, tendo ao final do looping o valor da fatorial do valor informado para o parâmetro N. Ao término do looping, a instrução **escreva(FAT)** imprime o valor da variável FAT, que somente é válida dentro da sub-rotina e por esta razão ficará “preso” dentro da mesma. A passagem de parâmetro por valor é utilizada somente para a entrada de um determinado valor.

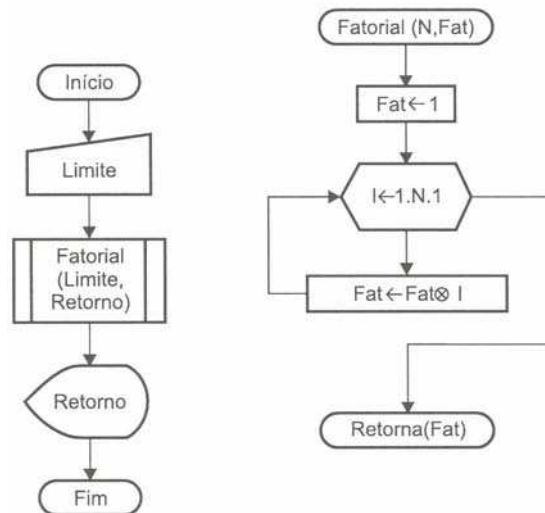
12.2.2 – Por Referência

A passagem de parâmetro por referência caracteriza-se pela ocorrência de alteração do valor do parâmetro real quando o parâmetro formal é manipulado dentro da sub-rotina. Desta forma, qualquer modificação feita no parâmetro formal implica em alteração no parâmetro real correspondente. A alteração efetuada é

devolvida para a rotina chamadora. Como exemplo deste tipo de parâmetro considere o mostrado abaixo, e observe a instrução **var** sendo utilizada junto da declaração do parâmetro em português estruturado.

Diagrama de Blocos

Ao utilizar a passagem de parâmetros por referência, eles deverão ser indicados não só no início da sub-rotina, mas também no símbolo *terminal* junto da palavra **retorna**, uma vez que este tipo de parâmetro devolve para o módulo principal do programa um resultado.



Português Estruturado

```

algoritmo "Calc_Fatorial_V2"

procedimento FATORIAL(N: inteiro; var FAT: inteiro)
var
  i: inteiro
inicio
  FAT <- 1
  para i de 1 ate N faca
    FAT <- FAT * i
  fimpara
fimprocedimento

var
  LIMITE, RETORNO: inteiro
inicio
  escreva("Qual o fatorial: ")
  leia(LIMITE)
  FATORIAL(LIMITE, RETORNO)
  escreva(RETORNO)
fimalgoritmo

```

Neste exemplo, é indicado o uso da passagem de parâmetro por referência (variável *FAT* por meio da instrução **var** na declaração do nome da sub-rotina). A variável *N* no exemplo continua sendo do tipo passagem de parâmetro por valor, pois recebe o valor fornecido à variável *LIMITE*, por meio da sub-rotina *FATORIAL*. Esse valor estabelece o número de vezes que o *looping* deve ser executado. Dentro do procedimento, é encontrada a variável *FAT* que é do tipo passagem de parâmetro por referência e possui no final o valor acumulado do cálculo da fatorial. Ao término do *looping*, o valor da variável *FAT* é transferido para fora da rotina, ou seja, é transferido para a variável *RETORNO* do programa principal, a qual apresenta o valor recebido de dentro da sub-rotina por meio da variável *FAT*. A passagem de parâmetro por referência é utilizada para que se tenha a saída de um determinado valor de dentro de uma sub-rotina.

12.3 – Exercício de Aprendizagem

Para demonstrar o uso de sub-rotina com passagem de parâmetro, considere a leitura de 10 valores em uma matriz de uma dimensão (vetor) e os coloque em ordem crescente. A ordenação deve ser executada com uma sub-rotina apropriada para este fim.

O processo de ordenação já é conhecido. A proposta é que se crie uma sub-rotina que faça a ordenação dos valores. Você deve lembrar que uma ordenação ocorre com a troca de valores entre duas variáveis, ou a troca de elementos entre dois índices de uma matriz, e que a troca só ocorre após a verificação de uma condição, onde se pergunta se o primeiro valor é maior que o segundo (no caso crescente), e se for, efetua-se a troca.

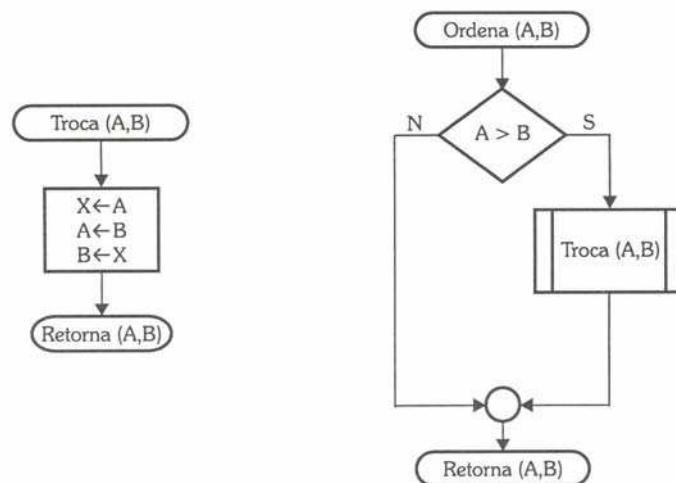
Note que são duas operações bem distintas: uma é a troca e a outra a verificação da condição. Assim sendo, são concebidas duas sub-rotinas, uma para a troca e a outra para a verificação da condição:

1. Criar duas sub-rotinas, uma para a troca e outra para verificação da condição;
2. Para a troca, estabelecer 3 variáveis: X como auxiliar e A e B como valores;
3. Para verificação de condição comparar com duas variáveis A e B;
4. Pelo fato de as sub-rotinas retornarem uma resposta, seus parâmetros vão ser por referência.

1ª Solução

O diagrama de blocos especifica apenas as sub-rotinas de troca e de verificação da condição para a troca de valores. Assim sendo, o código abaixo apresenta apenas os algoritmos para as sub-rotinas de troca e verificação da condição para a troca de valores. Como os parâmetros a serem utilizados são por referência, estão sendo declarados como a instrução **var**.

Diagrama de Blocos



Português Estruturado

```

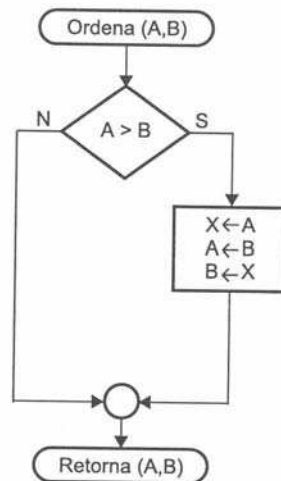
procedimento TROCA(var A, B: inteiro)
var
  X: inteiro
inicio
  X ← A
  A ← B
  B ← X
fimprocedimento

procedimento ORDENA(var A, B: inteiro)
inicio
  se (A > B) entao
    TROCA(A,B)
  fimse
fimprocedimento
    
```

2ª Solução

Esta solução é parecida com a primeira e, quando utilizada, surtirá o mesmo efeito. A diferença está na forma de sua escrita, pois neste exemplo se utiliza apenas uma sub-rotina para efetuar a verificação da condição e a troca dos valores.

Diagrama de Blocos



Português Estruturado

```

procedimento ORDENA(var A, B: inteiro)
var
    X: inteiro
inicio
    se (A > B) entao
        X ← A
        A ← B
        B ← X
    fimse
fimprocedimento
    
```

A seguir, é apresentado um exemplo de um programa, fazendo uso da sub-rotina de ordenação. O programa em questão faz a leitura de dez valores, ordena-os e em seguida apresenta a lista classificada.

```

algoritmo "Ordena_Numeros"
var
    VET: vetor[1..10] de inteiro
    i, j: inteiro

procedimento ORDENA(var A, B: inteiro)
var
    X: inteiro
inicio
    se (A > B) entao
        X ← A
        A ← B
        B ← X
    fimse
fimprocedimento

{Programa principal}
inicio
    {entrada de dados}
    para i de 1 ate 10 faca
        leia(VET[i])
    fimpara

    {Ordenação}
    para i de 1 ate 9 faca
    
```

```

para j de i+1 ate 10 faca
  ORDENA(VET[i], VET[j])
fimpara
fimalgoritmo

{Saída de Dados}
para i de 1 ate 10 faca
  escreval(VET[i])
fimpara

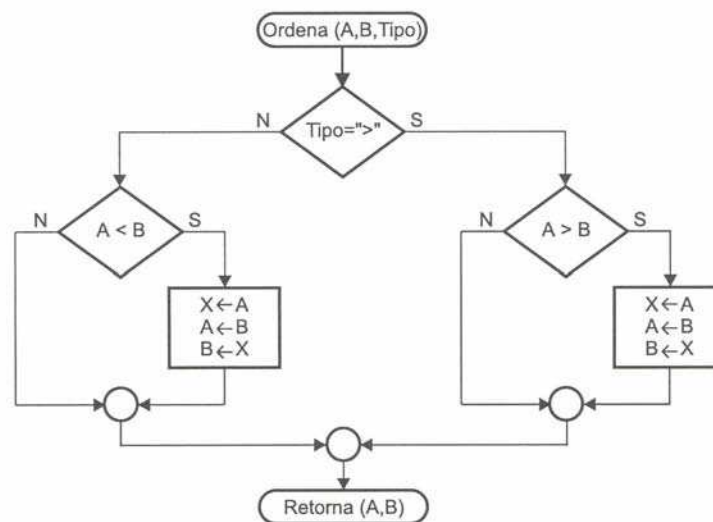
```

Observe que a criação de sub-rotina de ordenação deixou mais limpa a rotina de processamento do programa. É isto que deve ser levado em consideração quando se utilizam sub-rotinas.

Uma sub-rotina, se bem projetada, poderá ser de grande valia. Por exemplo, uma sub-rotina de ordenação que você possa informar um parâmetro para que a ordenação seja crescente ou decrescente. isto é simples; basta definir para a sub-rotina um novo parâmetro.

Para resolver este problema, será usada uma passagem de parâmetro por valor, em que a variável será do tipo caracter, sendo este informado no momento da ordenação por meio dos sinais > (maior que) ou < (menor que). Se quiser crescente o parâmetro será >; se decrescente, o parâmetro será <. A questão é apenas utilizar uma decisão dentro da rotina que identifique o sinal de classificação.

Diagrama de Blocos



Português Estruturado

```

algoritmo "Ordena_Numeros_v2"
var
  VET: vetor[1..10] de inteiro
  i, j: inteiro
  OPCA0: caracter

procedimento TROCA(var A, B: inteiro)
var
  X: inteiro
inicio
  X <- A
  A <- B
  B <- X
fimprocedimento

procedimento ORDENA(var A, B: inteiro; TIPO: caracter)
inicio
  se (TIPO = ">") entao

```



```

    se (A > B) entao
        TROCA(A,B)
    fimse
senao
    se (A < B) entao
        TROCA(A,B)
    fimse
fimse
fimprocedimento

{Programa principal}
inicio
    {entrada de dados}
    escreva("Digite > para crescente ou < para decrescente: ")
    leia(OPCAO)
    para i de 1 ate 10 faca
        leia(VET[i])
    fimpara

    {Ordenação}
    para i de 1 ate 9 faca
        para j de i+1 ate 10 faca
            ORDENA(VET[i], VET[j], OPCA0)
        fimpara
    fimpara

    {Saída de Dados}
    para i de 1 ate 10 faca
        escreval(VET[i])
    fimpara
fimalgoritmo

```

12.4 - Exercícios de Entrega Obrigatória (até ____/____/____) (Lista 11)

1) Desenvolva os algoritmos dos problemas abaixo e suas sub-rotinas do tipo **PROCEDIMENTO**. Cada problema deverá ser resolvido usando passagem de parâmetro por valor e por referência, ou seja, serão necessários dois algoritmos por exercício (Você deve gravar o exercício “a” como L11A, o exercício “b” como L11B, e assim por diante).

- Criar um algoritmo que efetue o cálculo de uma prestação em atraso. Para tanto, utilize a fórmula $PREST = VALOR + (VALOR * (TAXA/100) * TEMPO)$. Apresentar o valor da prestação.
- Elaborar um programa que possua uma sub-rotina que efetue e permita apresentar o somatório dos N primeiros números inteiros, definidos por um operador $(1+2+3+4+...+N)$.
- Escreva um programa que utilize uma sub-rotina para calcular a série de Fibonacci de N termos. A série de Fibonacci é formada pela seqüência: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... etc. Esta série caracteriza-se pela soma de um termo posterior com o seu anterior subsequente. Apresentar o resultado.
- Desenvolva um algoritmo de programa que crie uma sub-rotina para calcular e apresentar o valor de uma potência de um número qualquer. Ou seja, ao informar para a sub-rotina o número e sua potência, deverá ser apresentado o seu resultado. Por exemplo, se for mencionado no programa principal a sub-rotina POTENCIA(2, 3), deverá ser apresentado o valor 8.
- Elaborar um programa que efetue a leitura de um número inteiro e apresente uma mensagem informando se o número é par ou ímpar.
- Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final a soma dos quadrados dos três valores lidos.

- g) Elaborar um programa que por meio de sub-rotina efetue a apresentação do valor da conversão em real (R\$) de um valor lido em dólar (US\$). Deverá ser solicitado por meio do programa principal o valor da cotação do dólar e a quantidade de dólar disponível.
- h) Elaborar um programa que por meio de sub-rotina efetue a apresentação da mensagem: “Este valor é divisível por 2 e 3”. Deverá ser solicitado pelo programa principal o valor a ser verificado. Caso o valor não atenda à condição desejada, a sub-rotina deverá apresentar a mensagem: “Valor inválido”.
- i) Elaborar um programa que por meio de uma sub-rotina apresente como resultado um número positivo, mesmo que a entrada tenha sido feita com um valor negativo.
- j) Elaborar um programa que por meio de sub-rotina apresente o resultado do valor de um fatorial de um número qualquer fornecido pelo usuário.

12.5 - Exercícios de Extras

1) Desenvolva os algoritmos dos problemas abaixo e suas sub-rotinas do tipo PROCEDIMENTO. Cada problema deverá ser resolvido usando passagem de parâmetro por valor e por referência (ou seja, serão necessários dois algoritmos por exercício).

- a) Elaborar um programa que utilizando uma sub-rotina apresente o valor de uma temperatura em graus Fahrenheit. O programa deverá ler a temperatura em graus Celsius. Lembre-se da fórmula de conversão: $F = \frac{9C + 160}{5}$.
- b) Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final o quadrado da soma dos três valores lidos.
- c) Elaborar um programa que efetue a leitura do nome e sexo de um indivíduo. Por meio de uma sub-rotina o programa deverá apresentar a mensagem “Ilmo. Sr.”, caso o sexo seja masculino, e “Ilma Sra.”, caso o sexo seja feminino. Apresentar também junto de cada mensagem o nome do indivíduo.
- d) 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 de cada produto é dado, respectivamente, para os produtos A e B como sendo de R\$ 10,00 e R\$ 20,00. Elaborar um programa que por meio de sub-rotina calcule e apresente o valor da despesa do freguês na compra dos produtos. Lembre-se que o freguês poderá levar mais de uma unidade de um determinado produto.

13 – Funções (Sub-Rotinas)

Uma *Função* também é um bloco de programa, como são os procedimentos, contendo início e fim e sendo identificada por um nome, por meio do qual também será referenciada em qualquer parte do programa principal. Uma sub-rotina de função é na verdade muito parecida com uma sub-rotina de procedimento. A sintaxe em português estruturado é também idêntica ao estudo anterior.

Português Estruturado

```
funcao <nome da função> (parâmetros): <tipo da função>
var
    <variáveis>
inicio
    <instruções>
```

```
retorne <valor>
fimfuncao
```

A sua principal diferença está no fato de uma função retornar um determinado valor, que é retornado no próprio nome da função. Quando se diz <valor>, devem ser considerados valores numéricos, lógicos ou literais (caracteres).

13.1 – Aplicação de Funções em um Programa

Os procedimentos pela passagem de parâmetro por referência permitem que sejam retornados valores à rotina chamadora, e desta forma podem ser impressos, atribuídos a uma variável, servirem em operações aritméticas, entre outras. Acontece que com o uso de sub-rotinas de funções, esta tarefa fica mais simples.

Como exemplo, considere o procedimento seguinte usado para calcular o fatorial de um número qualquer. Observe que temos que fornecer dois parâmetros, sendo um N para a entrada do valor a ser calculado e FAT para a saída do resultado obtido.

```
procedimento FATORIAL(N: inteiro; var FAT:inteiro)
var
  i: inteiro
inicio
  FAT <- 1
  para i de 1 ate N faca
    FAT <- FAT * i
  fimpara
fimprocedimento
```

Se a sub-rotina acima for escrita em forma de função, será necessária apenas a informação de um parâmetro, no caso o valor do parâmetro N do qual deverá ser calculada a fatorial.

```
funcao FATORIAL(N: inteiro):inteiro
var
  i, FAT: inteiro
inicio
  FAT <- 1
  para i de 1 ate N faca
    FAT <- FAT * i
  fimpara
  retorne FAT
fimfuncao
```

Observe que o nome da função, no caso FATORIAL, é quem receberá o valor acumulado da variável FAT. Outro detalhe a ser considerado numa função é o fato da definição do seu tipo, no caso, **funcao** FATORIAL(n: inteiro):inteiro, em que N está sendo considerado como inteiro dentro dos parênteses. Isto significa que o valor fornecido pelo parâmetro N é inteiro, porém existe uma segunda definição de tipo fora dos parênteses, que é o tipo de retorno da função. Desta forma entra um valor inteiro com o parâmetro N e sai um valor inteiro para FATORIAL (através do comando **retorne** FAT). Observe que é perfeitamente normal entrar um parâmetro de um tipo e retornar um outro tipo.

13.2 – Considerações a Respeito de Funções

Você pode até estar achando que o número de linhas da função FATORIAL apresentada anteriormente é maior que o *procedimento* FATORIAL. Isto é verdade, mas no tocante ao uso no programa principal, ou qualquer outra rotina, a função é mais simples de ser mencionada.

Considerando que seja utilizado o procedimento FATORIAL, ele deve ser mencionado na rotina chamadora como:

```
FATORIAL(5, RETORNO)
escreva(RETORNO)
```

Entretanto considerando que seja utilizada a função FATORIAL, ela deve ser mencionada como:

escreva(FATORIAL(5))

13.3 – Exercício de Aprendizagem

Para demonstrar a utilização de programa com sub-rotinas do tipo função, considere os seguintes problemas:

1º Exemplo

Deverá ser criado um programa que faça uso de uma sub-rotina de função que retorne o valor da soma de dois números fornecidos como parâmetros.

Algoritmo

O problema proposto é bem simples, pois a função em questão recebe dois valores e retorna a soma deles. Desta forma o programa principal pedirá os valores e chamará a função para ter o retorno do cálculo. Assim sendo:

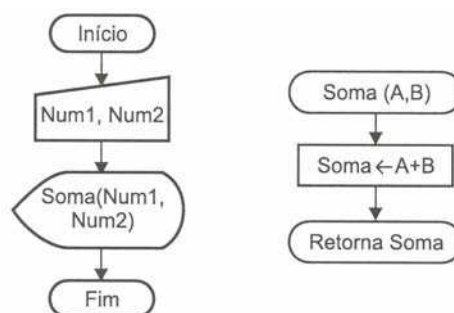
Programa Principal

1. Pedir a leitura de dois valores, no caso variáveis NUM1 e NUM2;
2. Chamar a função de soma, fornecendo como parâmetros as duas variáveis;
3. Apresentar o resultado retornado.

Sub-rotina para Soma

1. Receber dois valores fornecidos por meio de parâmetros;
2. Efetuar o cálculo entre os dois valores, no caso a soma;
3. Retornar para a função o resultado obtido.

Diagrama de Bloco



Português Estruturado

algoritmo "Soma"

```
funcao SOMA(a, b: real): real
var
    total: real
inicio
    total <- a + b
    retorne total
fimfuncao
```

```
var
    num1, num2: real
```

```

inicio
  escreva("Qual o primeiro valor: ")
  leia(num1)
  escreva("Qual o segundo valor: ")
  leia(num2)
  escreval("A soma é ", SOMA(num1, num2))
fimalgoritmo

```

2º Exemplo

Deverá ser criado um programa que por intermédio de uma sub-rotina do tipo função efetue a leitura de dois valores reais e apresente como saída uma mensagem informando se os números são iguais ou diferentes.

Algoritmo

Assim como o primeiro, este problema também é simples, pois a função em questão recebe dois valores, compara-os e retorna se são iguais ou diferentes. Desta forma, o programa principal pedirá os valores e chamará a função para ter o retorno da condição de igualdade.

Programa principal

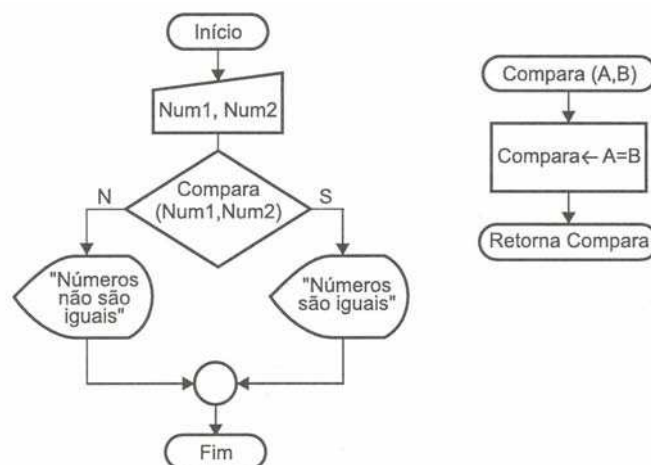
1. Pedir a leitura de dois valores, no caso variáveis NUM1 e NUM2;
2. Chamar a função de comparação de igualdade, fornecendo os dois valores;
3. Apresentar o resultado retornado.

Sub-rotina para Comparação

1. Receber dois valores fornecidos por meio de parâmetros;
2. Efetuar a comparação entre os valores para determinar se são iguais ou diferentes;
3. Retornar para a função o resultado obtido.

Observe que este tipo de problema já indica a entrada de parâmetros de um tipo e o retorno da resposta da função de outro tipo.

Diagrama de Blocos



Português Estruturado

O programa principal efetua a leitura das variáveis NUM1 e NUM2, transferindo os seus valores para os parâmetros formais A e B do tipo real, que assumem seus respectivos valores tornando-se parâmetros reais. Em seguida é processada a comparação dos dois valores e implicado à função o retorno da condição, desta forma o retorno desta função deverá ser lógico.

```

algoritmo "Comparacao"

```

```

var
num1, num2: real

funcao COMPARA(A, B: real): logico
inicio
    retorne ( A = B )
fimfuncao

inicio
    escreva("Qual o primeiro valor: ")
    leia(num1)
    escreva("Qual o segundo valor: ")
    leia(num2)
    se ( COMPARA(num1,num2) ) entao
        escreval("Os numeros são IGUAIS")
    senao
        escreval("Os numeros são DIFERENTES")
    fimse
fimalgoritmo

```

3º Exemplo

Anteriormente foi estudado o uso de sub-rotinas do tipo procedimento no programa calculadora do capítulo 11. Neste programa, as sub-rotinas de cálculo foram simplificadas com o uso das sub-rotinas de ENTRADA e SAIDA que são operações genéricas de todas as rotinas de cálculo. Deverá agora ser efetuada uma sub-rotina calculadora que faça qualquer uma das quatro operações, segundo a rotina de cálculo selecionada pelo operador.

A proposta agora é criar uma sub-rotina de função que efetue o cálculo, segundo o parâmetro de operação fornecido. Assim, essa sub-rotina deverá receber três parâmetros, sendo os dois números mais o operador para cálculo.

1. Se o operador for "+", faz-se à soma dos dois valores;
2. Se o operador for "-", faz-se à subtração dos dois valores;
3. Se o operador for "*", faz-se à multiplicação dos dois valores;
4. Se o operador for "/", faz-se à divisão dos dois valores.

Observe que na estrutura de controle com múltipla escolha, foi omitida a instrução **senão**. Isto é possível e poderá ser utilizado quando não se deseja estabelecer uma operação ou execução para alguma opção inválida. Perceba que isto, neste exemplo, não é necessário, uma vez que o terceiro parâmetro será indicado dentro de cada sub-rotina de cálculo.

```

funcao CALCULO(A, B: real; operador: caractere): real
var
    total: real

inicio
    escolha operador
    caso "+"
        total <- A + B
    caso "-"
        total <- A - B
    caso "*"
        total <- A * B
    caso "/"
        total <- a / b
    fimescolha
    retorne total
fimfuncao

```

A seguir é apresentada somente o algoritmo em português estruturado do programa Calculador, aplicando o uso da nova função:

```

algoritmo "Calculadora"
var
    opcao: caracter
    r, a, b: real

```

```

funcao CALCULO(A, B: real; operador: caractere): real
var
    total: real

inicio
    escolha operador
    caso "+"
        total <- A + B
    caso "-"
        total <- A - B
    caso "*"
        total <- A * B
    caso "/"
        total <- a / b
    fimescolha
    retorne total
fimfuncao

procedimento ENTRADA
inicio
    escreva("Entre um valor para A: ")
    leia(A)
    escreva("Entre um valor para B: ")
    leia(B)
fimprocedimento

procedimento SAIDA
inicio
    escreval("O resultado de A com B é = ", r)
    escreval()
fimprocedimento

procedimento ROTSOMA
inicio
    escreval("Rotina de SOMA")
    ENTRADA
    r <- CALCULO(A, B, "+")
    SAIDA
fimprocedimento

procedimento ROTSUBTRACAO
inicio
    escreval("Rotina de SUBTRACAO")
    ENTRADA
    r <- CALCULO(A, B, "-")
    SAIDA
fimprocedimento

procedimento ROTMULTIPLICACAO
inicio
    escreval("Rotina de MULTIPLICACAO")
    ENTRADA
    r <- CALCULO(A, B, "*")
    SAIDA
fimprocedimento

procedimento ROTDIVISAO
inicio
    escreval("Rotina de DIVISAO")
    ENTRADA
    r <- CALCULO(A, B, "/")
    SAIDA
fimprocedimento

//Programa principal
inicio
    opcao <- "0"
    enquanto (opcao <> "5") faca
        escreval("1 - Adição")
        escreval("2 - Subtração")

```

```
escreval("3 - Multiplicação")
escreval("4 - Divisão")
escreval("5 - Fim do programa")
escreval("Escolha uma opcao: ")
leia(opcao)
se (opcao <> "5") entao
    escolha opcao
    caso "1"
        ROTSOMA
    caso "2"
        ROTSUBTRACAO
    caso "3"
        ROTMULTIPLICACAO
    caso "4"
        ROTDIVISAO
    outrocaso
        escreval("Opção inválida - Tente novamente.")
    fimescolha
fimse
fimenquanto
fimalgoritmo
```

13.4 - Exercícios de Entrega Obrigatória (até ____/____/____) **(Lista 12)**

1) Desenvolva os algoritmos dos problemas abaixo, usando o conceito de sub-rotinas de **FUNÇÕES** (Você deve gravar o exercício “a” como L12A, o exercício “b” como L12B, e assim por diante).

- a) Escreva um programa que utilize uma sub-rotina para calcular a série de Fibonacci de N termos. A série de Fibonnacci é formada pela seqüência: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... etc. Está série caracteriza-se pela soma de um termo posterior com o seu anterior subsequente. Apresentar o resultado.
- b) Criar um algoritmo que efetue o cálculo de uma prestação em atraso. Para tanto, utilize a fórmula $PREST = VALOR + (VALOR * (TAXA/100) * TEMPO)$. Apresentar o valor da prestação.
- c) Desenvolva um algoritmos de programa que crie uma sub-rotina para calcular e apresentar o valor de uma potência de um número qualquer. Ou seja, ao informar para a sub-rotina o número e sua potência, deverá ser apresentado o seu resultado. Por exemplo, se for mencionado no programa principal a sub-rotina POTENCIA(2,3), deverá ser apresentado o valor 8.
- d) Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final o quadrado da soma dos três valores lidos.
- e) Elaborar um programa que por meio de sub-rotina efetue a apresentação do valor da conversão em real (R\$) de um valor lido em dólar (US\$). Deverá ser solicitado por meio do programa principal o valor da cotação do dólar e a quantidade de dólar disponível.
- f) Elaborar um programa que com o uso de sub-rotina do tipo função apresente o valor de uma temperatura em graus Celsius. O programa deverá ler a temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius. Sendo que a fórmula de conversão é $C \leftarrow (F - 32) * (5/9)$.

13.5 - Exercícios de Extras

1) Desenvolva os algoritmos dos problemas abaixo, usando o conceito de sub-rotinas de **FUNÇÕES**.

- a) Elaborar um programa que possua uma sub-rotina que efetue e permita apresentar o somatório dos N primeiros números inteiros, definidos por um operador ($1+2+3+4+\dots+N$).
- b) Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final a soma dos quadrados dos três valores lidos.
- c) Elaborar um programa que efetue a apresentação do valor da conversão em dólar (US\$) de um valor lido em real (R\$). O programa deverá solicitar o valor da cotação do dólar e também a quantidade de reais disponível com o usuário.