

ABPDEA

Associação Brasileira

para a Proteção dos Direitos

Editoriais e Autorais

www.abpdea.org.br

Respeite o Autor

Não Faça Cópia

www.abpdea.org.br

Algoritmos

Lógica para Desenvolvimento de Programação de Computadores

Editora

ABPDEA

Associação Brasileira

para a Proteção dos Direitos

Editoriais e Autorais

www.abpdea.org.br

Respeite o Autor

Não Faça Cópia

www.abpdea.org.br

Seja Nosso Parceiro no Combate à Cópia Ilegal

A cópia ilegal é crime. Ao efetuá-la, o infrator estará cometendo um grave erro, que é inibir a produção de obras literárias, prejudicando profissionais que serão atingidos pelo crime praticado.

Junte-se a nós nesta corrente contra a pirataria. Diga não à cópia ilegal.

Seu Cadastro É Muito Importante para Nós

Ao preencher e remeter a ficha de cadastro constante no final desta publicação, você passará a receber informações sobre nossos lançamentos em sua área de preferência.

Conhecendo melhor nossos leitores e suas preferências, vamos produzir títulos que atendam suas necessidades.

Obrigado pela sua escolha.

Fale Conosco!

Eventuais problemas referentes ao conteúdo deste livro serão encaminhados ao(s) respectivo(s) autor(es) para esclarecimento, excetuando-se as dúvidas que dizem respeito a pacotes de softwares, as quais sugerimos que sejam encaminhadas aos distribuidores e revendedores desses produtos, que estão habilitados a prestar todos os esclarecimentos.

Os problemas só podem ser enviados por:

1. E-mail: producao@erica.com.br
2. Fax: (11) 6197.4060
3. Carta: Rua São Gil, 169 - Tatuapé - CEP 03401-030 - São Paulo - SP



**INVISTA EM VOCÊ.
LEIA LIVROS!**

José Augusto N. G. Manzano
Jayr Figueiredo de Oliveira

Algoritmos

Lógica para Desenvolvimento de Programação de Computadores

Editora Érica Ltda.
2005 - 17^a Edição

Conselho Editorial:

Diretor Editorial:	Antônio Marco Vicoari Cipelli
Diretor Comercial:	Paulo Roberto Alves
Diretor de Publicidade:	Waldir João Sandrini
Finalização de Capa:	Edson Antônio dos Santos
Editoração:	Rosana Ap. A. dos Santos
Desenhos:	Flávio Eugênio de Lima
Revisão Interna:	Graziela M. L. Gonçalves
Revisão Gramatical:	Marlene Teresa Santin Alves
Coordenação:	Rosana Arruda da Silva

Agradecimentos

Agradeço ao amigo e editor Antonio Marco, que desde o nosso primeiro contato confiou no meu trabalho, principalmente neste que ora se concretiza. Agradeço também aos amigos professores Nélio e Wilson, pela constante troca de idéias.

Augusto Manzano

Agradeço aos meus alunos que, por meio das exigências e anseios, cobraram um aprofundamento maior do meu universo de conhecimentos. Obrigado também aos colegas professores Fernando Almeida, Marcos Masetto, Onésimo de Oliveira, Newton Mattei, principalmente à Maria Cristina Sanches Amorim e Sérgio Sonnino pelas orientações e conselhos profissionais.

Jayr Figueiredo

Sobre os Autores

José Augusto N. G. Manzano

Brasileiro, nascido no Estado de São Paulo, Capital, em 26/04/1965, é professor e mestre. Atua na área de Tecnologia da Informação (Desenvolvimento e Treinamento) desde 1986. Atualmente exerce a atividade profissional de consultor e professor universitário, ministrando as disciplinas de Estrutura de Dados, Técnicas de Programação, Lógica de Programação, Introdução à Microinformática, Linguagens de Programação, Engenharia de Software, Tópicos Avançados em Processamento de Dados, Sistemas de Informação, Engenharia da Informação, Arquitetura de Computadores e Tecnologia WEB. Tem mais de quarenta obras publicadas na área em que atua.

Jayr Figueiredo de Oliveira

Bacharel em Administração de Empresas, com especializações em Administração, Didática em Ensino Superior, Análise de Sistemas, Ciência da Computação, MBA em Tecnologia, Conhecimento e Inovação. Mestre em Administração e Planejamento, Doutor em Educação-curriculum, Pós-doutorando em Ciências Sociais.

Vem atuando em empresas nacionais e multinacionais como profissional de Administração em Sistemas Informatizados desde 1977, tendo ocupado inúmeros cargos de alta gerência. É professor de cursos de graduação e pós-graduação desde 1982. Tem mais de dez obras publicadas na área em que atua.

Índice Analítico

Parte I - Introdução	01
Capítulo 1 - Abordagem Contextual	03
1.1 -Definições Básicas	03
1.2 -Necessidades do Uso da Lógica	04
1.3 -Aplicabilidade da Lógica no Auxílio do Desenvolvimento de Programas.....	04
1.4 -Diferenciação de Nomenclaturas	05
1.5 -Formas de Representação Gráfica	07
1.6 -Simbologias Básicas	08
1.7 -Simbologias Especiais	09
Capítulo 2 - Introdução à Lógica	11
2.1 -Princípios de Resolução de Problemas	11
2.2 -Particularidades entre Lógicas	14
2.2.1 - Linear	14
2.2.2 - Estruturada	15
2.2.3 - Modular	17
2.2.4 - Diagrama de Chapin	17
2.2.5 - Português Estruturado	18
Parte II - Técnicas Básicas de Programação	21
Capítulo 3 - Tipos de Dados e Instruções Primitivas	23
3.1 -Tipos de Informação	23
3.2 -Tipos de Dados	23
3.2.1 - Tipos Inteiros	23
3.2.2 - Tipos Reais	24
3.2.3 - Tipos Caractere	24
3.2.4 - Tipos Lógicos	24
3.3 -O Uso de Variáveis	24
3.4 -O Uso de Constantes	25
3.5 -Os Operadores Aritméticos	26

3.6 - As Expressões Aritméticas ou Fórmulas Matemáticas	26
3.7 - Instruções Básicas	27
3.7.1 - Algumas Regras antes de Começar	27
3.7.2 - Entrada, Processamento e Saída	28
3.8 - Exercício de Aprendizagem	32
3.9 - Exercício de Fixação	35
Capítulo 4 - Estruturas de Controle - A Tomada de Decisões	39
4.1 - Desvio Condicional Simples	39
4.2 - Operadores Relacionais	42
4.3 - Desvio Condicional Composto	42
4.4 - Desvios Condicionais Encadeados	44
4.5 - Operadores Lógicos	47
4.5.1 - Operador Lógico: .e.	48
4.5.2 - Operador Lógico: .ou.	49
4.5.3 - Operador Lógico: .não.	50
4.6 - Exercício de Aprendizagem	52
4.7 - Exercício de Fixação	57
Capítulo 5 - Estruturas de Controle - Laços ou Malhas de Repetição	61
5.1 - Repetição do Tipo: Teste Lógico no Início do Looping	62
5.2 - Repetição do Tipo: Teste Lógico no Fim do Looping	66
5.3 - Repetição do Tipo: Variável de Controle	70
5.4 - Consideração entre os Tipos de Estruturas de Looping	73
5.5 - Estruturas de Controle Encadeadas	73
5.5.1 - Encadeamento de Estrutura Enquanto com Enquanto	73
5.5.2 - Encadeamento de Estrutura Enquanto com Repita	74
5.5.3 - Encadeamento de Estrutura Enquanto com Para	75
5.5.4 - Encadeamento de Estrutura Repita com Repita	76
5.5.5 - Encadeamento de Estrutura Repita com Enquanto	77
5.5.6 - Encadeamento de Estrutura Repita com Para	78
5.5.7 - Encadeamento de Estrutura Para com Para	79
5.5.8 - Encadeamento de Estrutura Para com Enquanto	80
5.5.9 - Encadeamento de Estrutura Para com Repita	81
5.6 - Exercício de Aprendizagem	82
5.7 - Exercício de Fixação	90

Parte III - Estruturas Básicas de Dados - Tabelas em Memória	93
Capítulo 6 - Estrutura de Dados Homogêneas I	95
6.1 - Matrizes de uma Dimensão ou Vetores	95
6.2 - Operações Básicas com Matrizes do Tipo Vetor.....	97
6.2.1 - Atribuição de uma Matriz	98
6.2.2 - Leitura dos Dados de uma Matriz	98
6.2.3 - Escrita dos Dados de uma Matriz	100
6.3 - Exercício de Aprendizagem	101
6.4 - Exercício de Fixação	105
Capítulo 7 - Aplicações Práticas do Uso de Matrizes do Tipo Vetor	109
7.1 - Classificação dos Elementos de uma Matriz	111
7.2 - Métodos de Pesquisa em uma Matriz	118
7.2.1 - Método de Pesquisa Seqüencial	118
7.2.2 - Método de Pesquisa Binária	118
7.3 - Exercício de Aprendizagem	120
7.4 - Exercício de Fixação	128
Capítulo 8 - Estruturas de Dados Homogêneas II	131
8.1 - Matrizes com mais de uma Dimensão	131
8.2 - Operações Básicas com Matrizes de Duas Dimensões	132
8.2.1 - Atribuição de uma Matriz	133
8.2.2 - Leitura dos Dados de uma Matriz	133
8.2.3 - Escrita dos Dados de uma Matriz	134
8.3 - Exercício de Aprendizagem	135
8.4 - Exercício de Fixação	145
Capítulo 9 - Estruturas de Dados Heterogêneas	149
9.1 - Estrutura de um Registro	149
9.1.1 - Atribuição de Registros	150
9.1.2 - Leitura de Registros	151
9.1.3 - Escrita de Registros	153
9.2 - Estrutura de um Registro de Conjuntos	154
9.2.1 - Atribuição de Registros de Conjuntos	155
9.2.2 - Leitura de Registro de Conjuntos	155

9.2.3 - Escrita de Registro de Conjuntos	156
9.3 - Estrutura de um Conjunto de Registros	157
9.3.1 - Atribuição de Conjunto de Registros	157
9.3.2 - Leitura de Conjunto de Registros.....	158
9.3.3 - Escrita de Conjunto de Registros.....	159
9.4 - Exercício de Aprendizagem	161
9.5 - Exercício de Fixação	167
Parte IV - Programação Estruturada ou Modular	169
Capítulo 10 - Utilização de Sub-Rotinas	171
10.1 - As Sub-rotinas	171
10.2 - O Método Top-Down	172
Capítulo 11 - Aplicação Prática do Uso de Sub-Rotinas -Procedimentos ...	175
11.1 - Exercício de Aprendizagem	176
11.2 - Estrutura de Controle com Múltipla Escolha	181
11.3 - Variáveis Globais e Locais	184
11.3.1 - Escopo de Variáveis	186
11.3.2 - Refinamento Sucessivo	187
11.4 - Exercício de Fixação	190
Capítulo 12 - Utilização de Parâmetros	193
12.1 - Parâmetros Formais e Reais	193
12.2 - Passagem de Parâmetros	194
12.2.1 - Por Valor	194
12.2.2 - Por Referência	196
12.3 - Exercício de Aprendizagem	197
12.4 - Exercício de Fixação	204
Capítulo 13 - Aplicação Prática do Uso de Sub-rotinas - Funções	207
13.1 - Aplicação de Funções em um Programa	207
13.2 - Considerações a Respeito de Funções	209
13.3 - Exercício de Aprendizagem	209
13.4 - Exercício de Fixação	216
Parte V - Apêndices	217
A - Resoluções de Alguns Exercícios de Fixação	219
B - Exemplos de Codificação	225

PARTE I

Introdução

CAPÍTULO 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 seqüência 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/ou automatizados mecatronicamente. 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 desta obra é mostrar como desenvolver e aperfeiçoar melhor essa técnica, lembrando que para isso você deve ser persistente e praticá-la constantemente, chegando à exaustão sempre que julgar necessário.

1.3 - Aplicabilidade da Lógica no Auxílio do 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 seqüência 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 comandos 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 da programação;
- 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 apenas 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 diferentes. Mencionam-se em seguida os seus verdadeiros significados corretos em suas aplicações, mas lembre-se que para muitos elas terão a mesma finalidade, já que raros profissionais possuem uma formação adequada e capacitadora na área de Tecnologia da Informação, principalmente no que tange ao desenvolvimento de software.

Fluxograma é uma ferramenta usada e desenvolvida pelos profissionais da área de análise de sistemas (atualmente denominada como área de sistemas de informação), bem como, por alguns profissionais de Organização, Sistemas e Métodos. Tem como finalidade descrever o fluxo de ação de um determinado trabalho lógico, seja manual ou mecânico, especificando os suportes usados para os dados e para as informações. Usa símbolos convencionais (norma ISO 5807:1985), 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 (analista de sistemas de informação) e a serem realizados pelo programador por meio do desenvolvimento do raciocínio lógico (diagrama de blocos e codificação), o qual deverá solucionar o problema de um determinado usuário por meio de um programa a ser processado pelo computador. A figura 1.1 enfatiza este conceito.

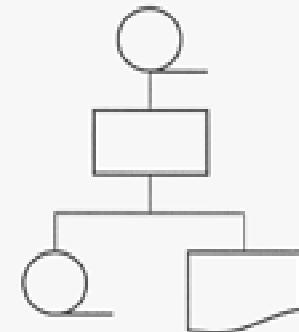


Figura 1.1 - Exemplo de um fluxograma.

Diagrama de Bloco (também poderia ser denominado *diagrama de fluxo* e não *fluxograma*) 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. Esses símbolos são internacionalmente aceitos e estão configurados, segundo as normas ANSI X3.5:1970 e ISO 1028:1973 (a qual foi reeditada como ISO 5807:1985). Após a elaboração do diagrama de blocos, será realizada a codificação do programa. A figura 1.2 mostra o exemplo de um diagrama de blocos.

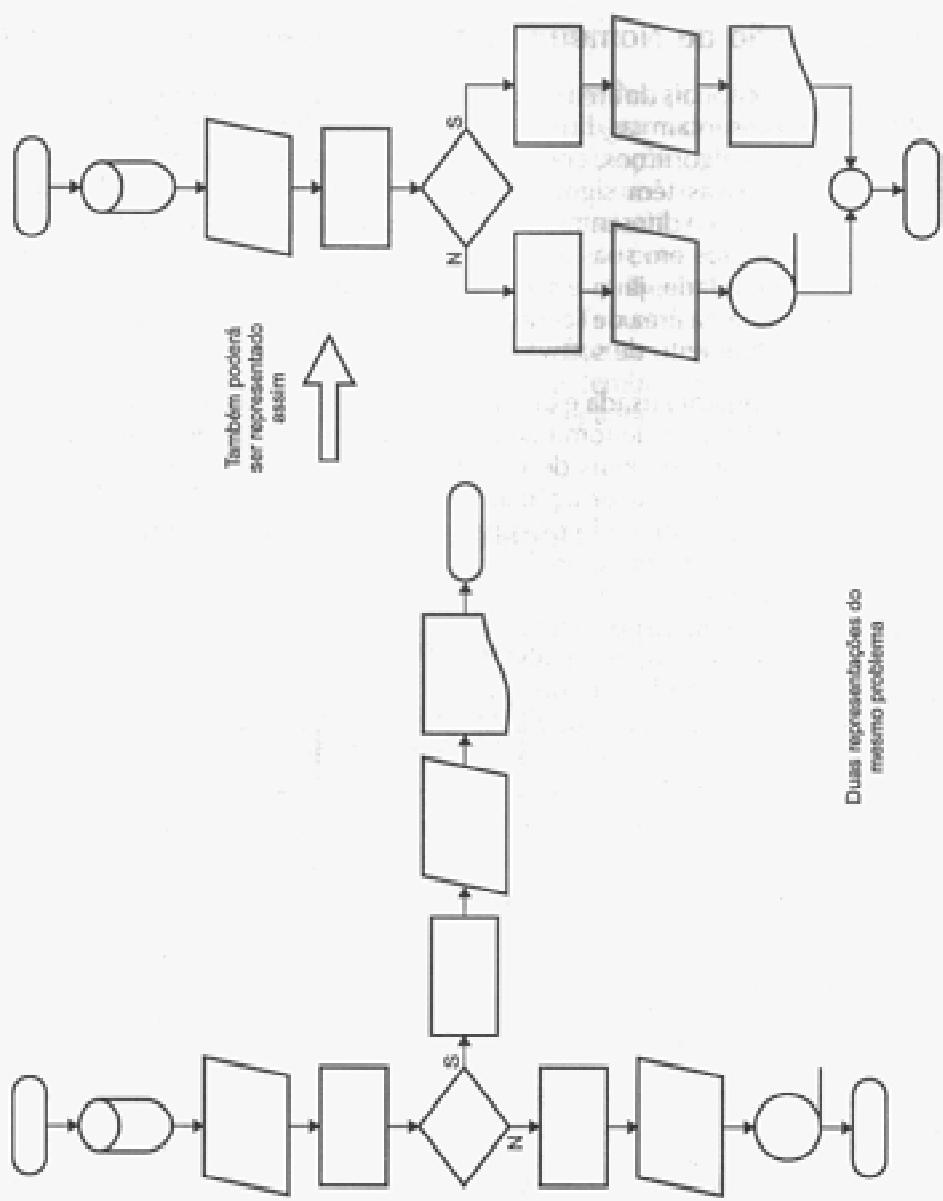


Figura 1.2 - Diagrama de Blocos.

Algoritmo é um processo de cálculo matemático ou da descrição sistemática da 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. Na área de desenvolvimento de software, é muito comum relacionar a palavra algoritmo com diagrama de bloco (neste caso, seria um algoritmo gráfico), 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, seja em uma folha de pagamento, bem como em qualquer situação que exija em resultado final "correto" e/ou "coerente". A figura 1.3 apresenta o exemplo de um algoritmo (forma gráfica) para calcular uma média escolar.

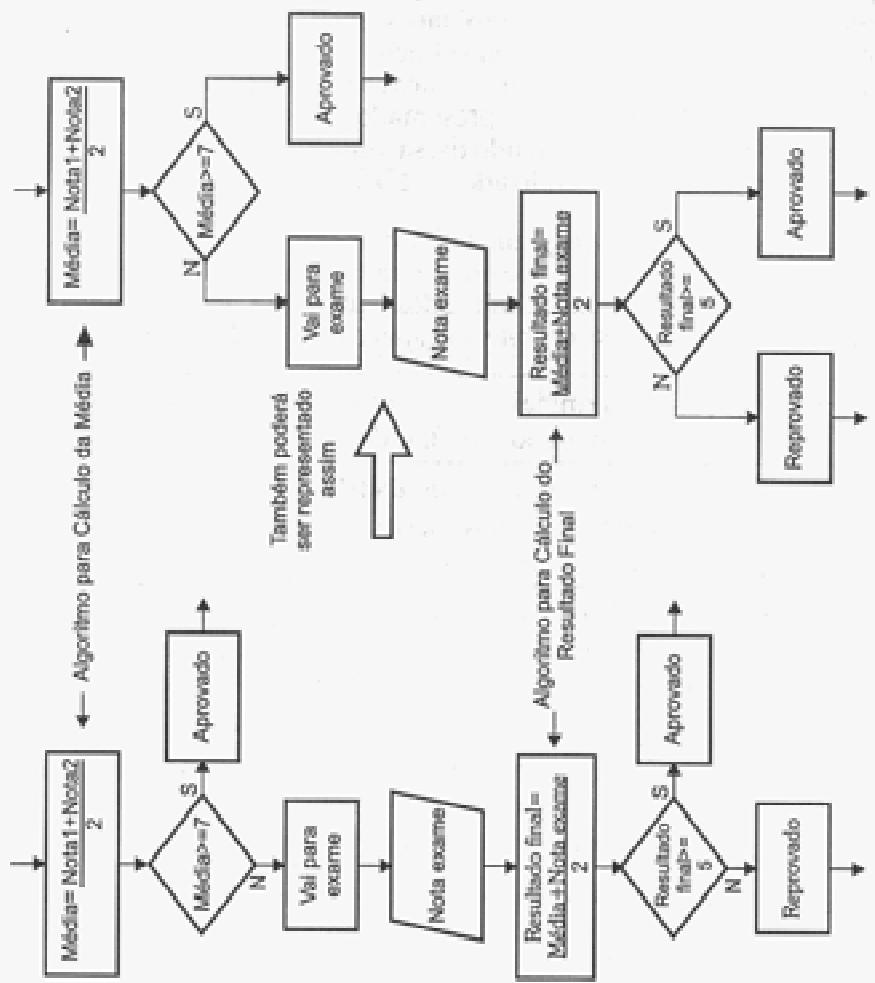


Figura 1.3 - Exemplo de um algoritmo.

1.5 - Formas de Representação Gráfica

As formas de representação gráfica (segundo norma ISO 5807:1985) são nada mais do que uma maneira mais simples e concisa de representar os dados sobre uma superfície plana, por meio de diferentes formas geométricas preestabelecidas, de modo a facilitar a visualização completa e imediata da linha de raciocínio lógico de um programador, quando da representação dos dados e do fluxo de ação de um programa de computador.

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.

A seguir, são mostrados os mais diversos símbolos, porém com o número excessivo de símbolos existentes e, quando combinados entre si, criam-se novos tipos de representações, aumentando consideravelmente esses símbolos. Entretanto, os profissionais, bem como as organizações, acabam criando suas próprias nomenclaturas. No decorrer dos exercícios apresentados nesta obra, serão utilizados os mais simples e convencionais, adotando dessa forma um padrão a ser seguido para facilitar sua compreensão e aprendizado.

1.6 - Simbologias Básicas

Estes são alguns dos símbolos mais conhecidos e utilizados ao longo do anos pelos profissionais de desenvolvimento de software da área de Tecnologia da Informação (TI).

	Terminal - símbolo utilizado como ponto para indicar o inicio 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 partitionar 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- Símbologias Especiais

Estes são os demais 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.

Outros sinais conhecidos pelos profissionais da área.

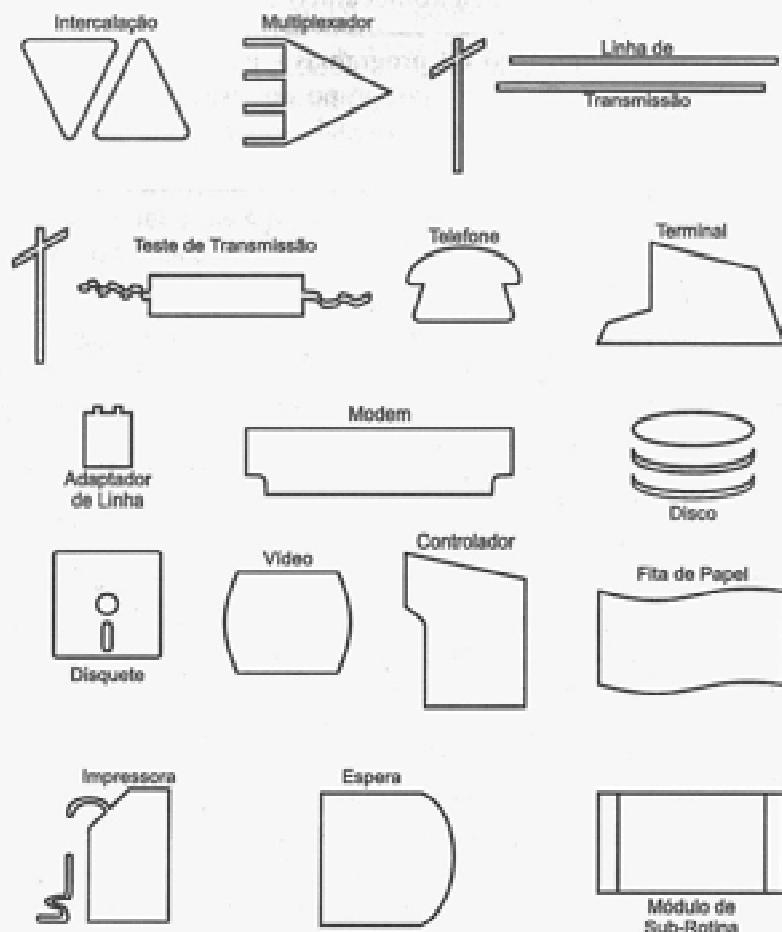


Figura 1.4 - Outros símbolos especiais.

Introdução à Lógica

2.1 - Princípios de Resolução de Problemas

Primeiramente, é importante entender e compreender 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 desta obra, 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.

Diferentes 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 detalhamentos 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.

Tome 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 2.1.



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 reaprovação do aluno. A figura 2.2 apresenta o diagrama de blocos com mais detalhes.

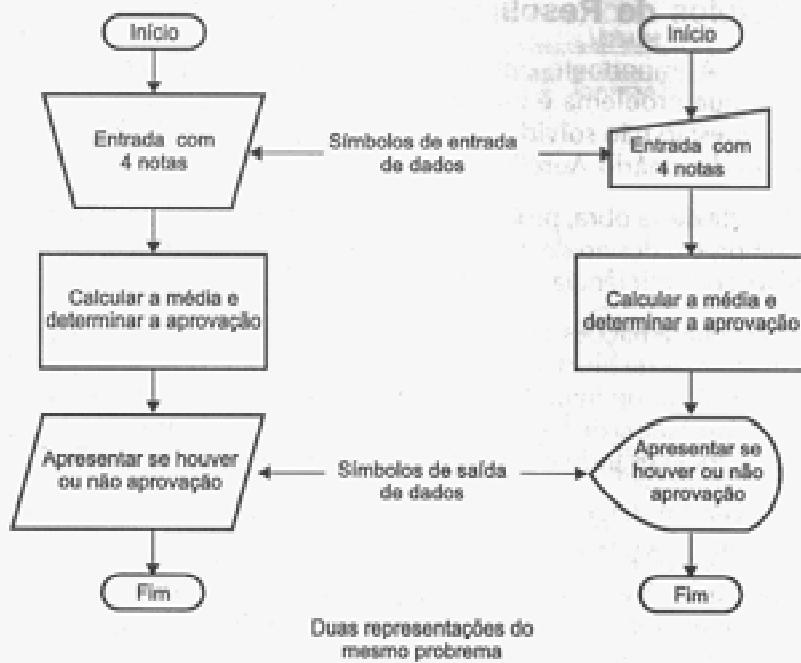


Figura 2.2 - Diagramas apresentando a entrada das notas e a saída se houve aprovação.

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 (sete), deve ser considerada no algoritmo. Com isso, inclui-se este bloco de decisão, como mostra a figura 2.3.

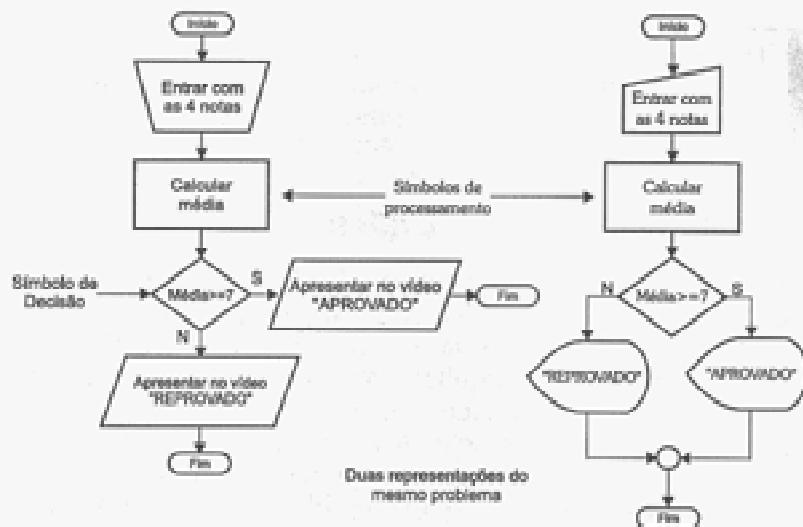


Figura 2.3 - Uso de uma condição em um diagrama de blocos.

Muitas vezes é preferível construir o diagrama de blocos trabalhando com variáveis. Este assunto será estudado no capítulo 3. A figura 2.4 apresenta um exemplo de um algoritmo utilizando variáveis.

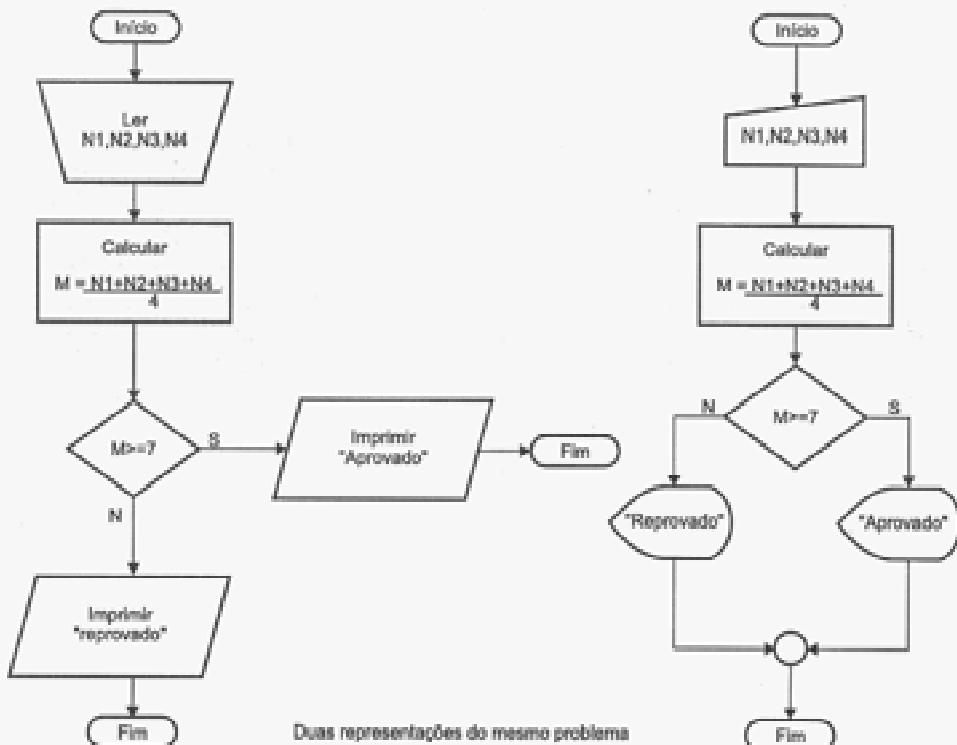


Figura 2.4 - Exemplo da utilização de variáveis.

OBS.: Em algumas figuras anteriores, são apresentados dois diagramas de bloco de formas diferentes, porém representando as mesmas ações e tendo os mesmos significados.

2.2 - Particularidades entre Lógicas

As representações gráficas de um diagrama de blocos podem ser feitas de várias maneiras e possuirem 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 para resolução dos problemas de processamento de dados. A seguir, são apresentados alguns tipos de procedimentos individualmente.

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 figura 2.5 apresenta um exemplo deste tipo de lógica.

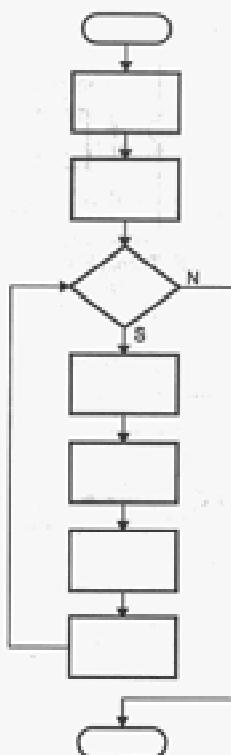


Figura 2.5 - Exemplo de lógica linear.

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 seqüência, a seleção e a iteração são as três estruturas básicas para a construção do diagrama de blocos. A figura 2.6 seguinte apresenta um exemplo do tipo de lógica estruturada.

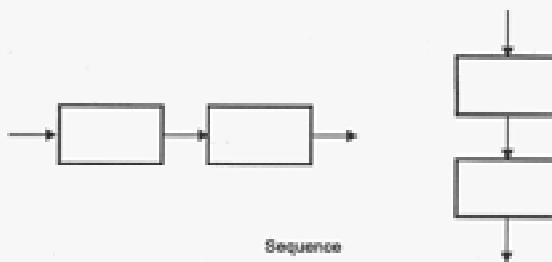


Figura 2.6 - Exemplo de lógica estruturada (continua na próxima página).

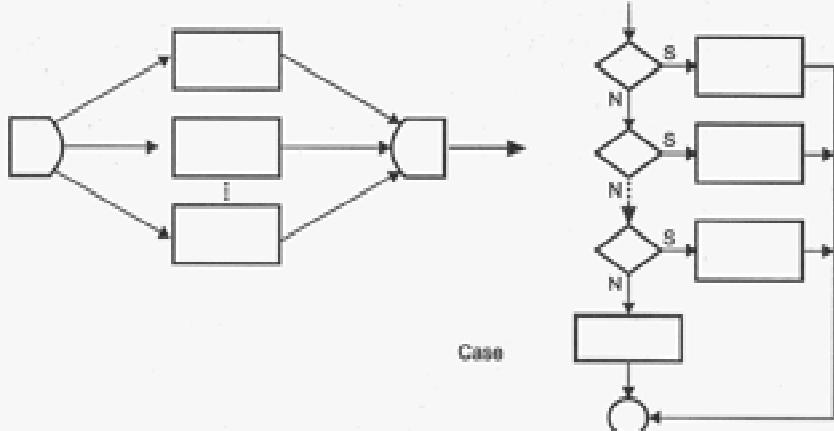
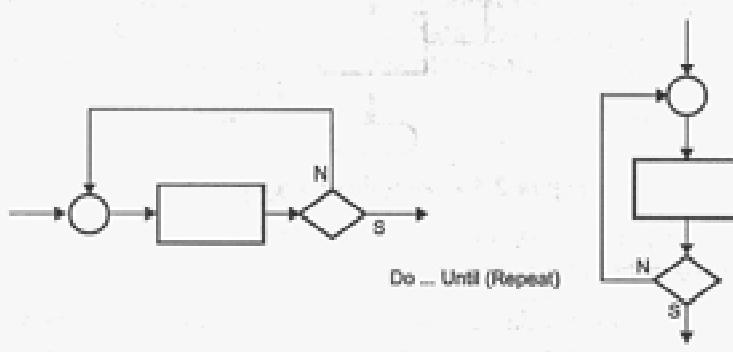
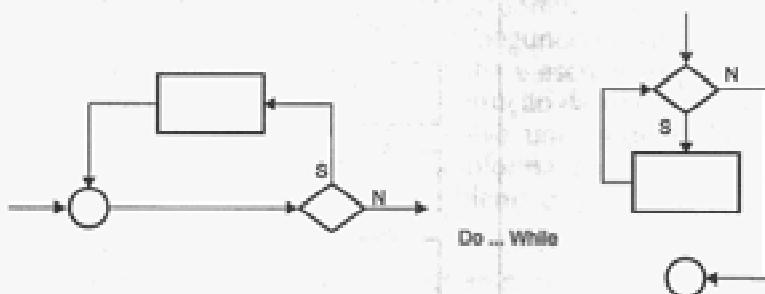
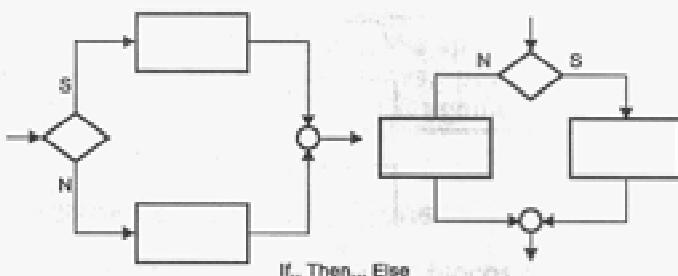


Figura 2.6 - Exemplo de lógica estruturada (continuação da página anterior).

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 2.7 apresenta um exemplo do tipo de lógica modular.

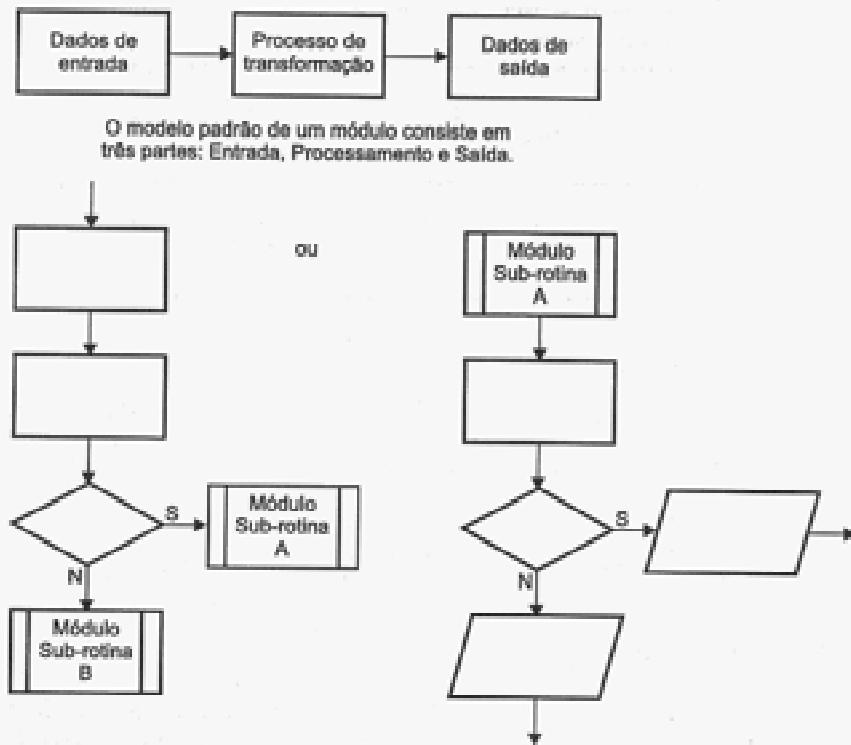


Figura 2.7 - Exemplo de lógica modular.

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

programa. 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 seqüência, seleção e repartição. 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 2.8 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*.

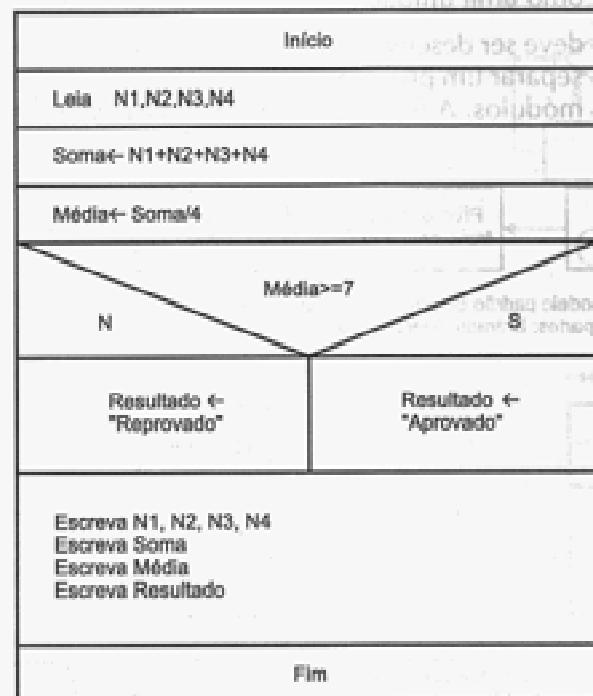


Figura 2.8 - Exemplo de diagrama de Chapin.

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 como *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). Nesta obra, estamos apresentando-a em português. A forma original de escrita é conhecida como *ínglês estruturado*, muito parecida com a notação da linguagem PASCAL. A PDL (neste caso, o *português estruturado*) é usada 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 de uma linguagem computacional (Clipper, C, Pascal, Delphi, Visual-Objects) e sua documentação. Abaixo, é apresentado um exemplo deste tipo de algoritmo.

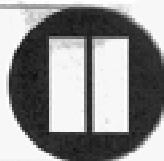
```
programa MÉDIA
var
    RESULTADO : caractere
    N1, N2, N3, N4 : real
    SOMA, MÉDIA : real
início
    leia N1, N2, N3, N4
    SOMA ← N1 + N2 + N3 + N4
    MÉDIA ← SOMA / 4
    se (MÉDIA >= 7) então
        RESULTADO ← "Aprovado"
    senão
        RESULTADO ← "Reprovado"
    fim_se
    escreva "Nota 1: ", N1
    escreva "Nota 2: ", N2
    escreva "Nota 3: ", N3
    escreva "Nota 4: ", N4
    escreva "Soma: ", SOMA
    escreva "Média: ", MÉDIA
    escreva "Resultado: ", RESULTADO
fim
```

A diferença entre uma linguagem de programação de alto nível utilizada em computação e uma PDL é que esta (seja escrita em português, inglês ou qualquer outro idioma) não pode ser compilada em um computador (por enquanto). Porém, existem "Processadores de PDL" que possibilitam traduzir essa linguagem numa representação gráfica de projeto, fornecendo uma variedade de informações, como: tabelas de referência cruzada, mapas de aninhamento, índice operacional do projeto, entre tantas outras.

ATENÇÃO

É importante lembrar que o diagrama de bloco e o pseudo-código (português estruturado) são as duas técnicas importantes para a documentação da solução de um problema computacional na forma de um programa de computador.

PARTE



Técnicas Básicas de Programação

CAPÍTULO 3

Tipos de Dados e Instruções Primitivas

A partir deste ponto de estudo, você terá um contato direto com a parte mais prática deste trabalho. Anteriormente nos preocupamos em proporcionar a você, leitor, um conhecimento teórico básico de alguns pontos que são por vezes causadores de dúvidas até a alguns profissionais de informática considerados experientes. Sendo assim, daqui para frente você terá um contato maior com a aplicação prática dos Algoritmos, Diagrama de Blocos, Testes de Mesa e por fim a codificação em pseudocódigo: "português estruturado". Mas ainda é necessário alguns conceitos operacionais.

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 a 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.

3.2.3 - Tipos Caracteres

São caracterizadas como tipos caracteres as 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, literal ou cadeia. Como exemplo deste tipo de dado, tem-se os valores: "PROGRAMAÇÃO", "Rua Alfa, 52 Ap. 1", "Fone: 574-9988", "04387-456", "", "7", 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 dois valores. Ele é chamado por alguns de **tipo booleano**, devido à contribuição do filósofo e matemático inglês George Boole na área da lógica matemática. Para facilitar a citação de um dado do tipo lógico, fica aqui declarado que estes deverão ser apresentados e delimitados pelo caractere ponto (.). Como exemplo deste tipo de dado têm-se os valores: .**Falso.**, .**F**, e .**N**. (para o valor lógico: falso) e .**Verdadeiro.**, .**V**, e .**S**. (para o valor lógico: verdadeiro)

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 (veja figura 3.1), 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.

Imagine a memória de um computador como um grande arquivo com várias gavetas, sendo que em cada gaveta é possível guardar apenas um único valor por vez, e essas gavetas, como em um arquivo, deverão estar identificadas com uma "etiqueta" contendo um nome.

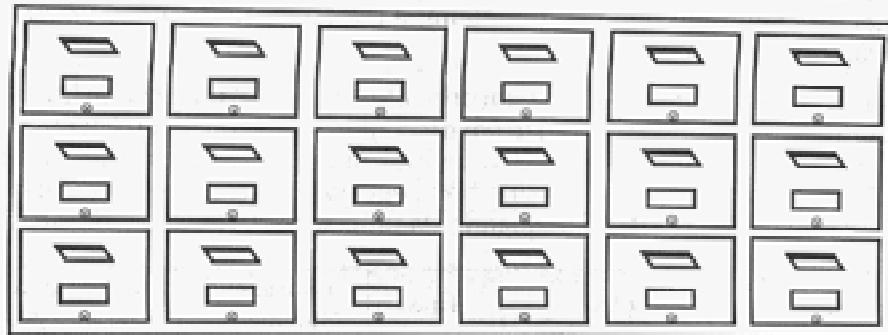


Figura 3.1 - Representação gráfica da memória de um computador nível de variáveis.

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;
- Não poderá ser nome de uma variável uma palavra reservada a uma instrução de programa;
- Não poderão ser utilizados outros caracteres a não ser letras e números.

São nomes válidos de variáveis: *NOMEUSUÁRIO*, *FONE1*, *X*, *DELTA25*, *Z4*, entre outros. São nomes inválidos de variáveis: *NOME USUÁRIO*, *1X*, *FONE#*, *ESCREVA* (considerando que seja esta uma palavra reservada à instrução de uma linguagem, no caso, o nosso "português estruturado").

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 sejam estabelecidas como do tipo real ou inteira, e para que isto ocorra é necessária a utilização de operadores aritméticos.

Os operadores aritméticos são 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	Tipo	Prioridade Matemática	Tipo de Retorno de Resultado
+	Manutenção de sinal	Unário	1	Positivo
-	Inversão de sinal	Unário	1	Negativo
\uparrow	Exponenciação	Binário	2	Inteiro ou real
/	Divisão	Binário	3	Real
div	Divisão	Binário	4	Inteiro
*	Multiplicação	Binário	3	Inteiro ou real
+	Adição	Binário	4	Inteiro ou real
-	Subtração	Binário	4	Inteiro ou real

3.6 - As Expressões Aritméticas ou Fórmulas Matemáticas

Será muito comum trabalharmos com **expressões aritméticas** ou **fórmulas matemáticas**, uma vez que a maior parte do 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: $\text{ÁREA} = \pi \cdot \text{RAIO}^2$ para o cálculo da área de uma circunferência, em que estão presentes as variáveis ÁREA e RAIO, a constante π ($\pi = 3.14159$) e os operadores aritméticos de multiplicação e também a operação de potência, elevando o valor da variável RAIO ao quadrado.

As expressões aritméticas em computação são escritas de uma forma um pouco diferente da forma conhecida em matemática, por exemplo, a expressão $X = \{43 \cdot [55 : (30 + 2)]\}$ será escrita na forma computacional como $X \leftarrow (43 * (55 / (30 + 2)))$.

Perceba que as chaves e colchetes são abolidos, utilizando-se em seu lugar apenas os parênteses. É também substituído o sinal de (=) igual pelo sinal de (←) implicado ou atribuído.

O sinal implicado ou atribuição (\leftarrow) 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 da área de uma circunferência, esta poderia ser escrita das seguintes maneiras: $\text{ÁREA} \leftarrow 3.14159 * \text{RAIO}^2 / 2$ ou $\text{ÁREA} \leftarrow 3.14159 * \text{RAIO} * \text{RAIO}$.

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:

$$\text{ÁREA} = \frac{\text{BASE} * \text{ALTURA}}{2}$$

Ela deveria ser escrita como: $\text{ÁREA} \leftarrow (\text{BASE} * \text{ALTURA}) / 2$.

3.7 - Instruções Básicas

As **Instruções** são representadas pelo conjunto de *palavras-chave* (vocabulário) de uma determinada linguagem de programação, que tem por finalidade comandar em um computador o seu funcionamento e a forma como os dados armazenados deverão ser tratados. Deve-se ainda considerar que existem várias linguagens de programação, como: Pascal, C, Basic, SmallTalk, entre outras, sendo que uma determinada instrução para realizar uma tarefa em um computador poderá ser escrita de forma diferente, dependendo da linguagem utilizada. Por exemplo, em português se diz *rua*, em inglês se diz *street* e em castelhano se diz *calle*. São termos diferentes para representar a mesma coisa.

Para a confecção deste livro adotamos a utilização de uma pseudolínguagem denominada por uns Portugol e por outros Português Estruturado. Um fato importante a ser considerado é que esta linguagem na verdade não existe, pois não foi criado para ela o compilador que executasse os seus comandos dentro de um computador, servindo apenas como um instrumento didático e permitindo dar ao neoprogramador a destreza necessária na montagem das estruturas de programa.

Deste ponto para a frente você terá contato com instruções do pseudocódigo português estruturado, tais como: *início*, *fim*, *var*, *programa*, *enquanto* *fim_enquanto*, *se*, *então*, *senão*, *fim_se* *para*, *fim_para*, *escreva*, *leia*, *faça*, *repita* e *até_que*, *conjunto*, *inteiro*, *real*, *caractere*, *lógico*, *tipo*, *registro*, *fim_registro*, *procedimento*, *função*, *caso*, *fim_caso*. Estas instruções, colocadas de forma estratégica, formarão os blocos de programa.

3.7.1 - Algumas Regras antes de Começar

No tópico anterior 3.3 O Uso de variáveis, você aprendeu o significado de uma variável e também teve contato com algumas regras para sua utilização. Porém, teremos que ter algum cuidado quando estivermos fazendo referência a uma instrução ou a uma variável. Desta forma teremos mais algumas regras utilizadas neste livro, a saber:

- Todo problema a ser resolvido será previamente entendido, passado para um algoritmo, para depois ser representado graficamente por meio de um diagrama de blocos e sua estruturação em código português estruturado;
- Toda referência feita a uma instrução será escrita em letra minúscula em formato negrito. As instruções não serão indicadas dentro dos diagramas de blocos;
- Toda referência feita a uma variável será escrita em letra maiúscula em formato itálico, sendo que serão sempre indicadas dentro dos diagramas de blocos;
- Qualquer valor atribuído a uma variável será feito com o símbolo ← (seta para esquerda), 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 consequentemente 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 têm.

O processo de execução de um programa ocorre segundo o exposto, após a entrada de dados com a instrução **leia** e a sua saída com a instrução **escreva**. O processamento será uma consequência da manipulação das variáveis de ação.

Uma entrada e uma saída poderão ocorrer dentro de um computador de diversas formas. Por exemplo, uma entrada poderá ser feita via teclado, modem, leitores óticos, disco, entre outros. Uma saída poderá ser feita em vídeo, impressora, disco, entre outras formas. Devido a esta grande variedade, nossos programas escritos em português estruturado farão menção às instruções **leia** e **escreva**.

Diagramas 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).

À medida que o leitor for conhecendo outras instruções e aprendendo a trabalhar melhor a sua lógica na resolução dos problemas, iremos passando pouco a pouco outros símbolos e sua forma de utilização. Neste ponto em que estamos, realmente não é tão importante conhecer cada símbolo.

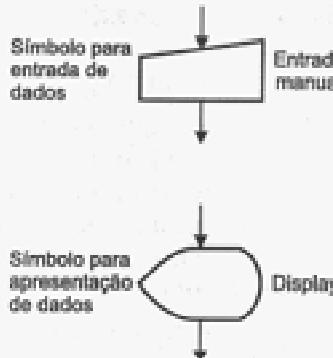


Figura 3.2 - Estrutura dos símbolos para as instruções leia e escreva.

Português Estruturado

leia <lista de dados>
escreva <lista de dados> ou informações>

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 a 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 do algoritmo, que poderá ser feita de duas formas: trabalhando com o estilo de diagrama de bloco tradicional (o mais utilizado por ser baseado na norma ISO 5807:1985, figura 3.3) ou com o diagrama de quadros (diagrama NS ou diagrama de Chapin), conforme apresenta a figura 2.8). Neste trabalho, é adotado o critério de trabalhar com o diagrama de bloco tradicional.



Ler de dois valores,
sua soma e apresentação

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 (algoritmo 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.

```
programa SOMA_NÚMEROS
var
    X : inteiro
    A : inteiro
    B : inteiro
```

Tendo relacionado todas as variáveis que serão utilizadas no programa com a 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.

```
início
    leia A
    leia B
    X ← A + B
    escreva X
fim
```

Após a leitura dos valores para as variáveis A e B, eles serão somados e implicados (\leftarrow) na variável X, a qual será apresentada com o valor da soma processada. A seguir, é apresentado o programa completo:

```

programa SOMA_NÚMEROS
var
  X : inteiro
  A : inteiro
  B : inteiro
início
  leia A
  leia B
  X ← A + B
  escreva X
fim

```

Daqui para frente o processo de montagem de um programa fica mais fácil; basta que o leitor siga as dicas anteriores.

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, apresentando 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 conterá 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 PI possui o valor 3.14159;
- 3 - Efetuar o cálculo da área, elevando ao quadrado o valor de R e multiplicando por PI;
- 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

Quando for mencionada uma operação de multiplicação num diagrama de bloco, será utilizado o símbolo \otimes para representar este tipo de cálculo.

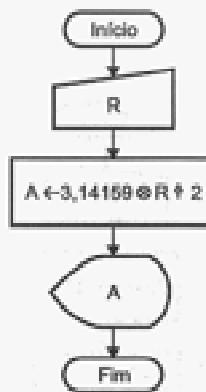


Figura 3.4 - Diagrama de bloco para o cálculo da área de uma circunferência.

Português Estruturado

```
programa AREA_CIRCULO
var
    A : real
    R : real
início
    leia R
    A ← 3.14159 * R ↑ 2
    escreva A
fim
```

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-se estabelecer qual será o seu salário bruto para efetuar o desconto e 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;

- Calcular o total de desconto (TD) com base no valor de PD dividido por 100;
- Calcular o salário líquido (SL), deduzindo o desconto do salário bruto;
- 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

```

programa SALARIO_PROFESSOR
var
  HT : inteiro
  VH, PD, TD, SB, SL : real
início
  leia HT
  leia VH
  leia PD
  SB ← HT * VH
  TD ← (PD/100) * SB
  SL ← SB - TD
  escreva SB
  escreva SL
fim
  
```

3.9 - Exercício de Fixação

1 - Indique com um X quais dos dados abaixo são do tipo **Inteiro**.

- () 1000
- () "0"
- () "-900"
- () .Verdadeiro.
- () -456
- () 34
- () "Casa 8"
- () 0
- () .Falso.
- () -1.56

2 - Indique com um X quais dos dados abaixo são do tipo **Real**.

- () -678
- () "0.87"
- () "-9.12"
- () .Verdadeiro.
- () -456
- () -99.8
- () "Cinco"
- () 45.8976
- () .Falso.
- () -1.56

3 - Indique com um X quais dos dados abaixo são do tipo **Literal**.

- () 678
- () "0.87"
- () "-9.12"
- () "Verdadeiro"
- () -456
- () -99.8
- () "Cinco"
- () 45.8976
- () .Falso.
- () 1.56

4 - Indique com um X quais dos dados abaixo são do tipo Lógico.

- () -678
- () "0.87"
- () "-9.12"
- () Verdadeiro.
- () -456
- () .V.
- () "Cinco"
- () Falso.
- () .F.
- () -1.56

5 - Assinale com um X os nomes válidos para uma variável.

- () ENDEREÇO
- () 21BRASIL
- () FONE\$COM
- () NOMEUSUÁRIO
- () NOME_USUÁRIO
- () NOME*USUÁRIO
- () END*A-6
- () CIDADE3
- () #CABEC

6 - Desenvolva os algoritmos, diagramas de bloco e codificação em português estruturado dos seguintes programas:

- a) Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é: $F \leftarrow (9 * C + 160) / 5$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
- b) 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.
- c) Calcular e apresentar o valor do volume de uma lata de óleo, utilizando a fórmula: $VOLUME \leftarrow 3.14159 * R^2 * ALTURA$.
- d) Efetuar o cálculo da quantidade de litros de combustível gastos em uma viagem, utilizando-se um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deverá fornecer o tempo gasto e a velocidade média durante viagem. Desta forma, será possível obter a distância percorrida com a fórmula $DISTÂNCIA \leftarrow TEMPO * VELOCIDADE$. Tendo o valor da

distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula: LITROS_USADOS ← DISTÂNCIA / 12. O programa deverá apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem.

- e) Efetuar o cálculo e a apresentação do valor de uma prestação em atraso, utilizando a fórmula: PRESTAÇÃO ← VALOR + (VALOR * (TAXA / 100) * TEMPO).
- f) Ler dois valores 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 valores numéricos inteiros e apresentar o resultado das adições e das multiplicações utilizando-se o conceito de propriedade distributiva para a máxima-combinação possível entre as quatro variáveis. Considerando-se o uso das variáveis A, B, C e D, deverá ser efetuada seis adições e seis multiplicações, ou seja, de forma geral deverá ser combinada a variável A com a variável B, a variável A com a variável C, a variável A com a variável D. Depois será necessário combinar a variável B com a variável C e a variável B com a variável D e por fim a variável C será combinada com a variável D.
- h) Elaborar um programa que calcule e apresente o volume de uma caixa retangular, por meio da fórmula:
$$\text{VOLUME} \leftarrow \text{COMPRIMENTO} * \text{LARGURA} * \text{ALTURA}.$$
- i) Efetuar a leitura de um número inteiro e apresentar o resultado do quadrado desse número.
- j) Ler dois valores inteiros (variáveis A e B) e apresentar o resultado do quadrado da diferença do primeiro valor (variável A) pelo segundo valor (variável B).
- k) Elaborar um programa que efetue a apresentação do valor da conversão em real (R\$) de um valor lido em dólar (US\$). O programa deverá solicitar o valor da cotação do dólar e também a quantidade de dólares disponível com o usuário.
- l) 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.
- m) Elaborar um programa que efetue a leitura de três valores inteiros (representados pelas variáveis A, B e C) e apresente como resultado final o valor da soma dos quadrados dos três valores lidos.

- n) Elaborar um programa que efetue a leitura de três valores inteiros (representados pelas variáveis A, B e C) e apresente como resultado final o valor do quadrado da soma dos três valores lidos.
- o) 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 da soma (variável S) do segundo com o quarto valor.
- p) 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).
- q) Elaborar um programa de computador que calcule e apresente o valor da área de uma circunferência (variável A). Para tanto, o programa deve solicitar o valor do raio (variável R) e fazer uso da fórmula de cálculo: $A \leftarrow 3.14159 * R^2$.
- r) 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.

CAPÍTULO 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 transformá-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...fim_se**. A instrução **se...então...fim_se** 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 **fim_se**. Sendo a condição *Falsa*, serão executadas as instruções que estejam após o comando **fim_se**.

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.

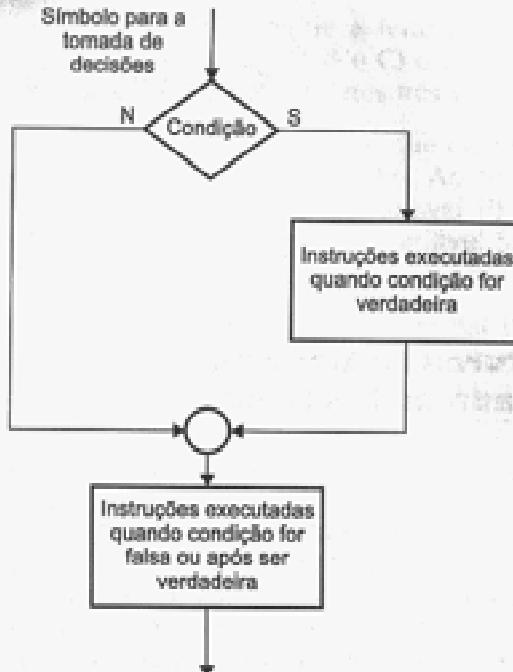


Figura 4.1 - Estrutura do símbolo para a instrução *se...então...fim_se*.

OBS: Também poderão ser utilizadas as letras V e F para determinar que a condição seja verdadeira ou falsa.

Português Estruturado

```

se (<condição>) então
    <instruções para condição verdadeira>
fim_se
    <instruções para condição falsa ou após ser verdadeira>
  
```

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". Veja o diagrama de blocos e a codificação em português estruturado.

Algoritmo

- 1 - Conhecer dois valores incógnitos (estabelecer variáveis A e B);
- 2 - Efetuar 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

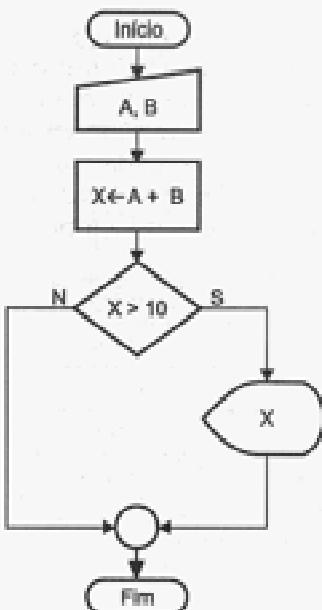


Figura 4.2 - Exemplo da utilização da estrutura se...então...fim_se.

Português Estruturado

```
programa SOMA_NÚMEROS
var
    X : inteiro
    A : inteiro
    B : inteiro
inicio
    leia A
    leia B
    X ← A + B
    se (X > 10) então
        escreva X
    fim_se
fim
```

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

2024/2 - 10/10/2024

Ao ser utilizada a instrução `se...então...fim_se`, ela implica na utilização de condições para verificar o estado de uma determinada variável quanto verdadeiro ou falso. Observe que para a condição do exemplo anterior foi utilizado o sinal de `>` (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
<code>=</code>	Igual a
<code>≠</code>	Diferente de
<code>></code>	Maior que
<code><</code>	Menor que
<code>>=</code>	Maior ou igual a
<code><=</code>	Menor ou igual a

4.3 - Desvio Condicional Composto

Há pouco você conheceu como fazer uso da instrução `se...então...fim_se` do tipo simples. Agora você aprenderá a fazer uso da instrução `se...então...senão...fim_se`, que sendo a condição *Verdadeira*, serão executadas todas as instruções que estejam posicionadas entre o `se...então` e a instrução `senão`. Sendo a condição *Falsa*, serão executadas as instruções que estejam entre o `senão` e a instrução `fim_se`.

Diagrama de Blocos

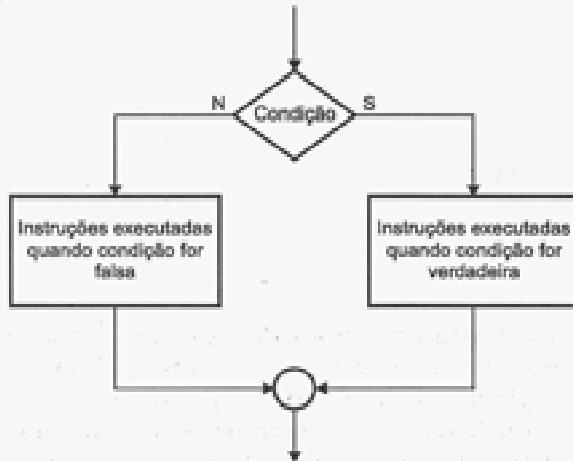


Figura 4.3 - Estrutura do símbolo para a instrução `se...então...senão...fim_se`.

Português Estruturado

```
se (<condição>) então
    <instruções para condição verdadeira>
senão
    <instruções para condição falsa>
fim_se
```

Para um exemplo da utilização desta estrutura considere o seguinte problema: "Ler dois valores numéricos e efetuar a adição. Caso o valor somado seja maior ou igual a 10, deverá ser apresentado somando a ele mais 5; caso o valor somado não seja maior ou igual a 10, este deverá ser apresentado subtraindo 7". Veja o diagrama de blocos e a codificação em português estruturado.

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 10; caso sim, calcule $R \leftarrow X+5$, senão calcule $R \leftarrow X-7$.

Diagrama de Blocos

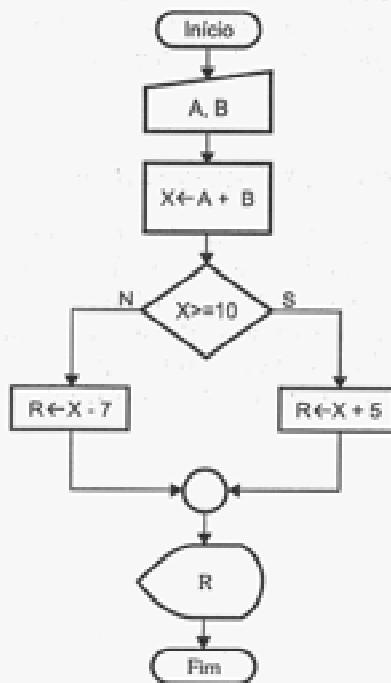


Figura 4.4 - Exemplo da utilização da estrutura se...então...senão...fim_se.

Português Estruturado

```
programa SOMA_NÚMEROS
var
    A : inteiro
    B : inteiro
    X : inteiro
    R : inteiro
inicio
    leia A,B
    X ← A + B
    se (X >= 10) então
        R ← X + 5
    senão
        R ← X - 7
    fim_se
    escreva R
fim
```

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.

Diagrama de Blocos

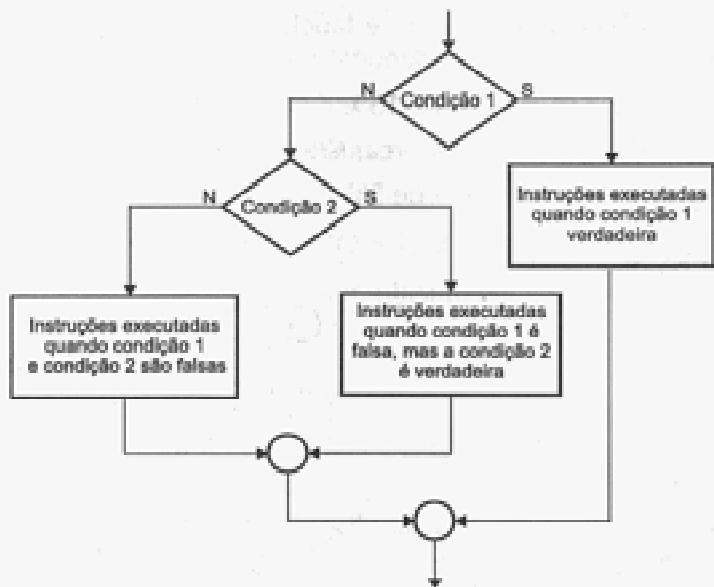


Figura 4.5 - Estrutura condicional composta ou encadeada.

Português Estruturado

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

```
se (<condição1>) então
    <instruções para condição1 verdadeira>
senão
    se (<condição2>) então
        <instruções para condição2 verdadeira, porém condição1 falsa>
    senão
        <instruções para condição1 e condição2 falsa>
    fim_se
fim_se
```

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 , reajuste será de 5%

Diagrama de Blocos

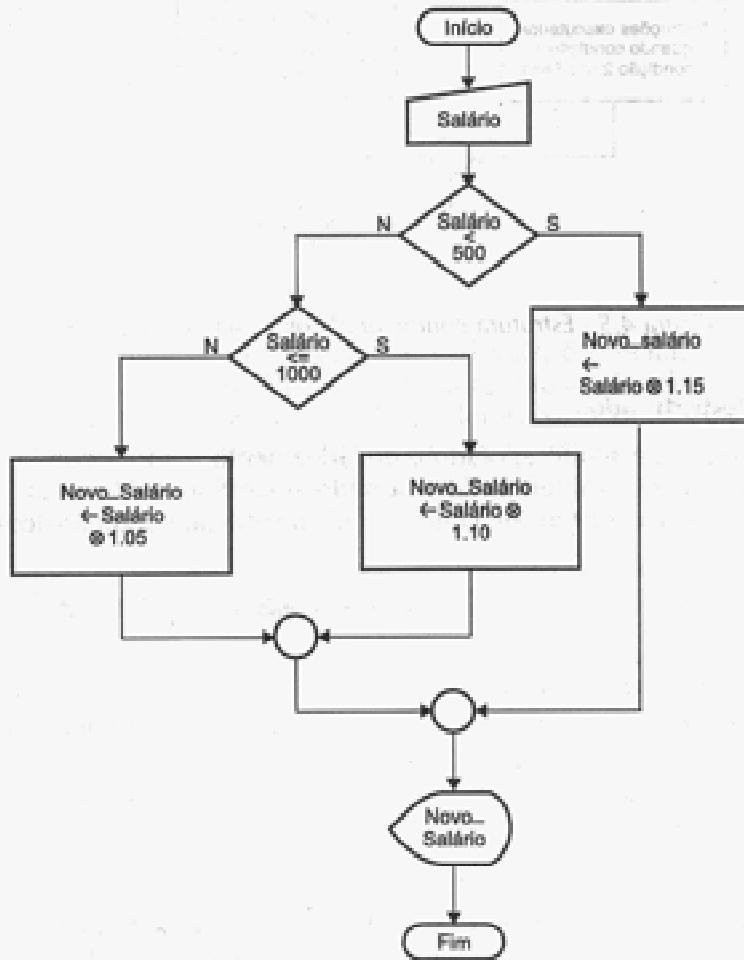


Figura 4.6 - Exemplo da utilização de uma estrutura condicional encadeada.

Observe que a referência feita na linha 5 do algoritmo não é escrita no diagrama de blocos e 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, consequentemente, acima de 1000.

Estas condições deverão ser encadeadas, pois todas as possibilidades de reajuste deverão ser cercadas. Sendo assim, observe o algoritmo abaixo:

- 1 - Definir uma variável para o salário reajustado: NOVO_SALÁRIO;
- 2 - Ler um valor para a variável SALÁRIO;
- 3 - Verificar se o valor de SALÁRIO < 500, se sim reajustar em 15%;
- 4 - Verificar se o valor de SALÁRIO <=1000, se sim reajustar em 10%;
- 5 - Verificar se o valor de SALÁRIO > 1000, se sim reajustar em 5%;
- 6 - Apresentar o valor reajustado, implicado em NOVO_SALÁRIO.

Português Estruturado

```
programa REAJUSTA_SALÁRIO
var
    NOVO_SALÁRIO : real
    SALÁRIO : real
inicio
    leia SALÁRIO
    se (SALÁRIO < 500) então
        NOVO_SALÁRIO ← SALÁRIO * 1.15
    senão
        se (SALÁRIO <= 1000) então
            NOVO_SALÁRIO ← SALÁRIO * 1.10
        senão
            NOVO_SALÁRIO ← SALÁRIO * 1.05
        fim_se
    fim_se
    escreva NOVO_SALÁRIO
fim
```

4.5 - Operadores Lógicos

Pode ser que você necessite, em algum momento, trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo na mesma instrução **se**, efetuando desta forma testes múltiplos. Para estes casos é necessário trabalhar com a utilização dos operadores lógicos, também conhecidos como operadores booleanos. Os operadores lógicos mais comuns são: **.e., .ou. e .não.**, e serão representados em português estruturado sempre entre pontos. Em alguns casos, o uso de operadores lógicos evita a utilização de instruções **se** 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. Abaixo, é apresentada a tabela de decisão para este tipo de operador:

Condição 1	Condição 2	Resultado
Falsa	Falsa	Falso
Verdadeira	Falsa	Falso
Falsa	Verdadeira	Falso
Verdadeira	Verdadeira	Verdadeiro

Diagrama de Blocos

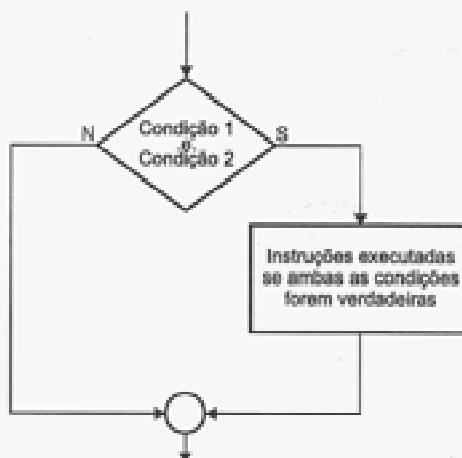


Figura 4.7 - Exemplo da utilização do operador lógico .e.

Português Estruturado

```
se (<condição1>) .e. (<condição2>) então
    <instruções executadas se condição1 e condição2 forem simultaneamente verdadeiras>
fim_se
```

O operador .e. faz com que somente seja executada uma determinada operação se todas as condições mencionadas forem simultaneamente verdadeiras. Veja o exemplo em Português Estruturado em seguida:

```
programa TESTA_LÓGICA_E
var
    NÚMERO : inteiro
inicio
    leia NÚMERO
```

```

se (NÚMERO >= 20) .e. (NÚMERO <= 90) então
    escreva "O número está na faixa de 20 a 90"
senão
    escreva "O número está fora da faixa de 20 a 90"
fim_se
fim

```

O exemplo mostra, por meio da utilização do operador **.e.**, que somente será apresentada a mensagem “**O número está na faixa de 20 a 90**”, caso o valor fornecido para a variável **NÚMERO** seja entre 20 e 90. Qualquer valor fornecido fora da faixa definida apresentará a mensagem “**O número não está na 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 (quando houver mais de um relacionamento) de uma condição necessita ser verdadeiro. Abaixo, é apresentada a tabela de decisão para este tipo de operador:

Condição 1	Condição 2	Resultado
Falsa	Falsa	Falso
Verdadeira	Falsa	Verdadeiro
Falsa	Verdadeira	Verdadeiro
Verdadeira	Verdadeira	Verdadeiro

Diagrama de Blocos

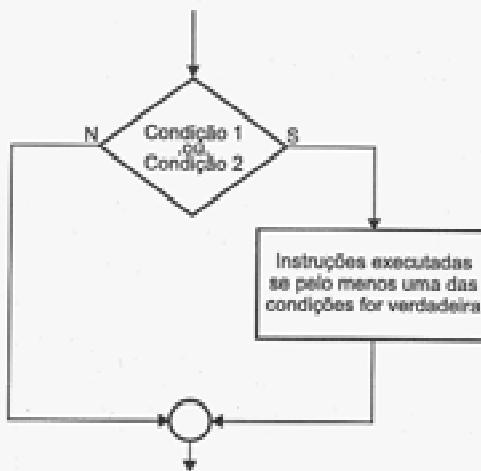


Figura 4.8 - Exemplo da utilização do operador lógico **.ou.**

Português Estruturado

```
se (<condição1>) .ou. (<condição2>) então
    <instruções executadas se cond1. verd. ou se cond2. verd.>
fim_se
```

O operador **.ou.** faz com que seja executada uma determinada operação se pelo menos uma das condições mencionadas for verdadeira. Veja o exemplo seguinte:

```
programa TESTA_LÓGICA_OU
var
    SEXO : caractere
início
    leia SEXO
    se (SEXO = "masculino") .ou. (SEXO = "feminino") então
        escreva "O seu sexo é válido"
    senão
        escreva "O seu sexo é inválido"
    fim_se
fim
```

O exemplo mostra, por meio da utilização do operador **.ou.**, que somente será apresentada a mensagem "**O seu sexo é válido**", caso o valor fornecido para a variável **SEXO** seja **masculino** ou **feminino**. Qualquer outro valor fornecido apresentará a mensagem "**O seu sexo é inválido**".

OBS.: Quando em um programa são trabalhados dados do tipo caractere, eles também são considerados valores. Não confundir com valores numéricos, pois existem basicamente três tipos de valores, sendo: lógicos, numéricos e caracteres. O termo valor está ligado ao conteúdo de uma variável, ou seja, ao valor que uma variável possui, seja ela do tipo que for.

4.5.3 - Operador Lógico: **.não.**

O operador do tipo **.não.** é utilizado quando houver a necessidade de estabelecer a inversão do resultado lógico de uma determinada condição. Se a condição for verdadeira, será considerada falsa. Se a condição for falsa, será considerada verdadeira. Abaixo, é representada a tabela de decisão para este tipo de operador:

Condição	Resultado
Verdadeira	Falso
Falso	Verdadeira

Diagrama de Blocos

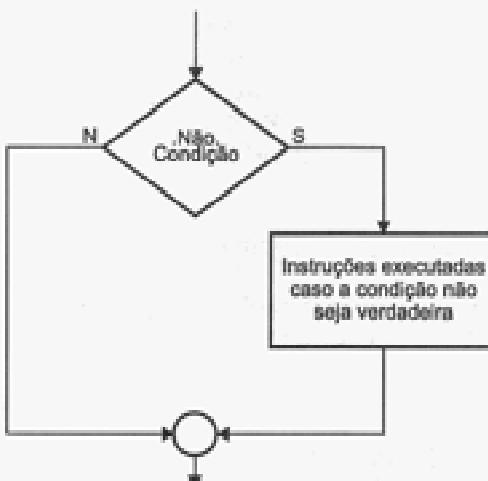


Figura 4.9 - Exemplo da utilização do operador lógico *.não.*

Português Estruturado

```
se .não. (<condição>) então
    <instruções executadas se condição não for verdadeira>
fim_se
```

O operador *.não.* faz com que seja executada uma determinada operação se a condição não for verdadeira, portanto, considerada falsa ou vice-versa. Veja o exemplo seguinte:

```
programa TESTA_LÓGICA_NÃO
var
    A, B, C, X : inteiro
inicio
    leia A, B, X
    se .não. (X > 5) então
        C ← (A + B) * X
    senão
        C ← (A - B) * X
    fim_se
    escreva C
fim
```

O exemplo acima mostra, por meio da utilização do operador *.não.*, que somente será efetuado o cálculo de $C \leftarrow (A + B) * X$, se o valor da variável X não for maior que 5. Qualquer valor de 5 para baixo efetuará o cálculo $C \leftarrow (A - B) * X$. Se forem informados os valores 5, 1 e 2, respectivamente, para as variáveis A , B e X , resultará

para a variável *C* o valor 12, pois o valor 2 da variável *X* é controlado pela instrução *se .não.* (*X > 5*) *então*, como sendo verdadeiro, uma vez que *não é maior que 5*. Sendo assim, os valores 5 e 1 são somados resultando 6 e multiplicados por 2 resultando 12. Mas se forem informados os valores 5, 1, e 6, respectivamente, para as variáveis *A*, *B* e *X*, resultará para a variável *C* o valor 24, pois o valor 6 da variável *X* é controlado pela instrução *se .não.* (*X > 5*) *então*, como sendo falso. Sendo assim, os valores 5 e 1 são subtraídos resultando 4 e multiplicados por 6 resultando 24.

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, e se for esta condição verdadeira, deverá ser indicado qual tipo de triângulo foi formado: isósceles, escaleno ou equilátero. Veja o algoritmo, diagrama de blocos e a codificação em português estruturado, prestando atenção na utilização dos operadores lógicos.

Algoritmo

Para estabelecer este algoritmo, é necessário em primeiro lugar saber o que realmente é um triângulo. Se você não souber o que é um triângulo, consequentemente não conseguirá resolver o problema. Triângulo é uma forma geométrica (polígono) composta por três lados, sendo que cada lado é menor que a soma dos outros dois lados. Perceba que isto é uma regra (uma condição) e deverá 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, serão então analisados os valores para estabelecer qual tipo de triângulo será 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 $A < > B$ e $B < > C$ 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. Se sim, saber se $A=B$ e se $B=C$, sendo verdade o triângulo é equilátero.
Se não, verificar $A=B$ 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, avisar a ocorrência.

Diagrama de Blocos

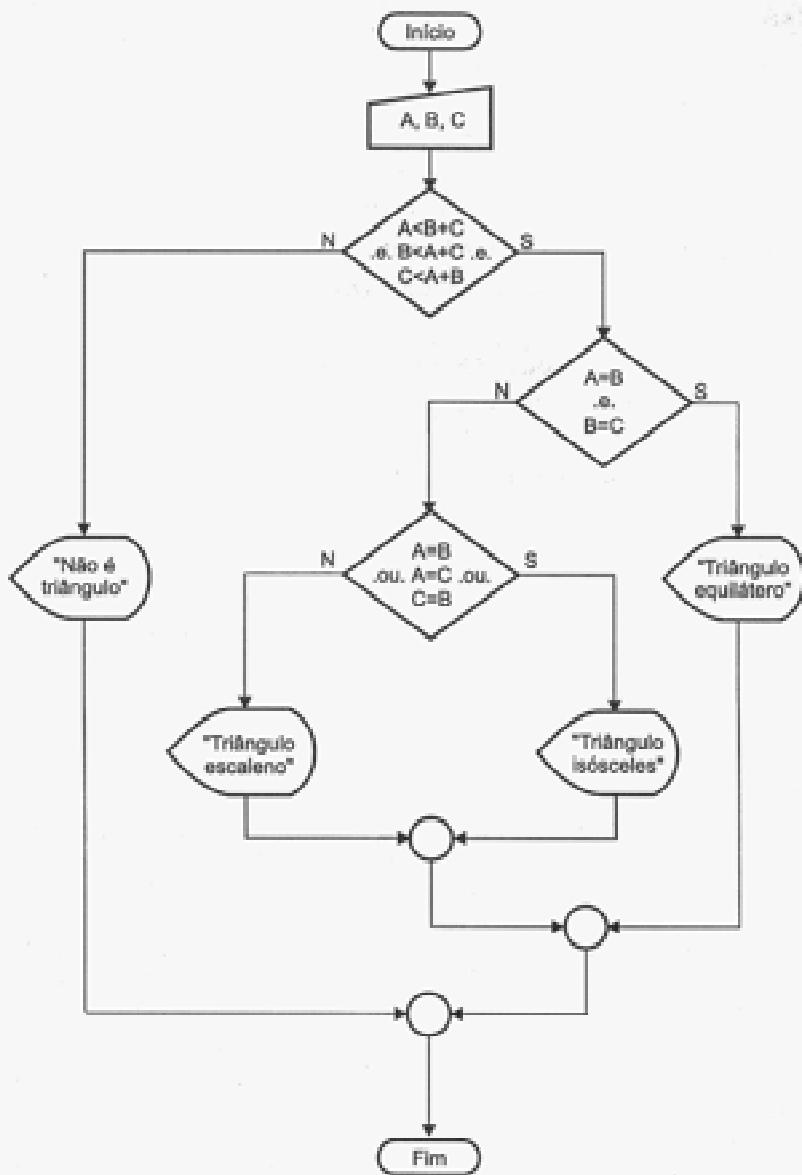


Figura 4.10 - Diagrama de blocos do programa triângulo.

Português Estruturado

```
programa TRIÂNGULO
var
  A, B, C : real
inicio
  leia A, B, C
  se (A < B + C) .e. (B < A + C) .e. (C < A + B) então
    se (A = B) .e. (B = C) então
      escreva "Triângulo Equilátero"
    senão
      se (A = B) .ou. (A = C) .ou. (C = B) então
        escreva "Triângulo Isósceles"
      senão
        escreva "Triângulo Escaleno"
      fim_se
    fim_se
  senão
    escreva "As medidas não formam um triângulo"
  fim_se
fim
```

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 4.11.

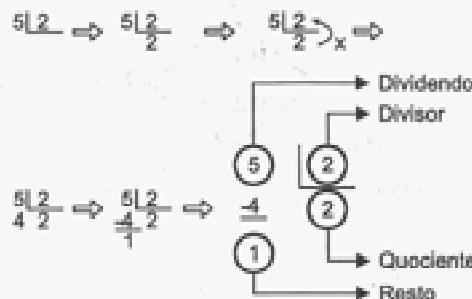


Figura 4.11 - Discriminação de uma operação de divisão com valores numéricos inteiros.

De acordo com o que está exposto na figura 4.11 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 respectivos valores, sendo: **Resto = 5 - 2 . 2**. Em seguida, basta efetuar o cálculo da multiplicação de $2 \cdot 2$, 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 calculo do resto da divisão da seguinte forma: **RESTO ← 5 - 2 * (5 div 2)**.

Observe a seguir o diagrama de blocos e o programa em Português Estruturado para verificar se um determinado valor numérico inteiro é ou não divisível por 4 e 5.

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

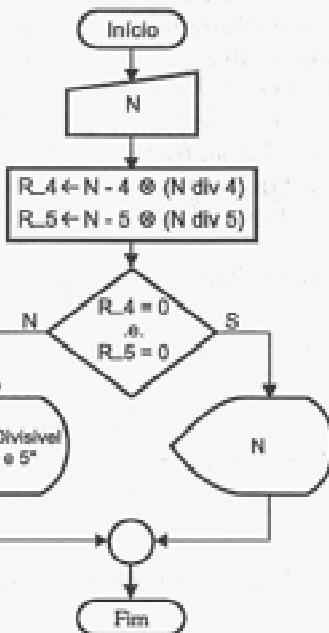


Figura 4.12 - Diagrama de blocos para verificar se N é divisível por 4 e 5.

Português Estruturado

```
programa DIVISÍVEL
var
  N, R_4, R_5 : inteiro
início
  leia N
  R_4 ← N - 4 * (N div 4)
  R_5 ← N - 5 * (N div 5)
  se (R_4 = 0) .e. (R_5 = 0) então
    escreva N
  senão
    escreva "Não é divisível por 4 e 5"
  fim_se
fim
```

4.7 - Exercício de Fixação

- 1 - Determine o resultado lógico das expressões mencionadas, assinalando se são verdadeiras ou falsas. Considere para as respostas os seguintes valores: $X = 1$, $A = 3$, $B = 5$, $C = 8$ e $D = 7$.

a- .não. ($X > 3$)

Verdadeiro (V)

Falso (F)

b- ($X < 1$) .e. .não. ($B > D$)

Verdadeiro (V)

Falso (F)

c- .não. ($D < 0$) .e. ($C > 5$)

Verdadeiro (V)

Falso (F)

d- .não. ($X > 3$) .ou. ($C < 7$)

Verdadeiro (V)

Falso (F)

e- ($A > B$) .ou. ($C > B$)

Verdadeiro (V)

Falso (F)

f- ($X \geq 2$)

Verdadeiro (V)

Falso (F)

g- ($X < 1$) .e. ($B \geq D$)

Verdadeiro (V)

Falso (F)

h- ($D < 0$) .ou. ($C > 5$)

Verdadeiro (V)

Falso (F)

i- .não. ($D > 3$) .ou. .não. ($B < 7$)

Verdadeiro (V)

Falso (F)

j- ($A > B$) .ou. .não. ($C > B$)

Verdadeiro (V)

Falso (F)

- 2 - Indique a saída dos trechos de programa em português estruturado, mostrado abaixo. Para as saídas considere os seguintes valores: A=2, B=3, C=5 e D=9. Não é necessário calcular os valores de X. Marque na resposta apenas a fórmula que será executada de acordo com a condição.

a - Resposta: _____

```
se .não. (D > 5) então
    X ← (A + B) * D
senão
    X ← (A - B) / C
fim_se
escreva X
```

b - Resposta: _____

```
se (A > 2) .e. (B < 7) então
    X ← (A + 2) * (B - 2)
senão
    X ← (A + B) / D * (C + D)
fim_se
escreva X
```

c - Resposta: _____

```
se (A = 2) .ou. (B < 7) então
    X ← (A + 2) * (B - 2)
senão
    X ← (A + B) / D * (C + D)
fim_se
escreva X
```

d - Resposta: _____

```
se (A > 2) .ou. .não. (B < 7) então
    X ← A + B - 2
senão
    X ← A - B
fim_se
escreva X
```

e- Resposta: _____

```
se .não. ( $A > 2$ ) .ou. .não. ( $B < 7$ ) então  
     $X \leftarrow A + B$   
senão  
     $X \leftarrow A / B$   
fim_se  
escreva X
```

f- Resposta: _____

```
se .não. ( $A > 3$ ) .e. .não. ( $B < 5$ ) então  
     $X \leftarrow A + B$   
senão  
     $X \leftarrow D / B$   
fim_se  
escreva X
```

g- Resposta: _____

```
se ( $C \geq 2$ ) .e. ( $B \leq 7$ ) então  
     $X \leftarrow (A + D) / 2$   
senão  
     $X \leftarrow D * C$   
fim_se  
escreva X
```

h- Resposta: _____

```
se ( $A \geq 2$ ) .ou. ( $C \leq 1$ ) então  
     $X \leftarrow (A + D) / 2$   
senão  
     $X \leftarrow D * C$   
fim_se  
escreva X
```

3 - Desenvolva os algoritmos, diagrama de blocos e codificação em português estruturado dos seguintes problemas:

- Ler dois valores numéricos inteiros e apresentar o resultado da diferença do maior valor pelo menor valor.
- Ler um valor numérico inteiro positivo ou negativo e apresentar o valor lido como sendo um valor positivo, ou seja, se o valor lido for menor ou igual a zero, ele deve ser multiplicado por -1.

- c) Ler os valores de quatro notas escolares de um aluno. Calcular a média aritmética e apresentar a mensagem “Aprovado” se a média obtida for maior ou igual a 5; caso contrário, apresentar a mensagem “Reprovado”. Informar junto de cada mensagem o valor da média obtida.
- d) Ler os valores de quatro notas escolares de um aluno. Calcular a média aritmética e apresentar a mensagem “Aprovado” se a média obtida for maior ou igual a 7; caso contrário, o programa deve solicitar a nota de exame do aluno e calcular uma nova média aritmética entre a nota de exame e a primeira média aritmética. Se o valor da nova média for maior ou igual a cinco, apresentar a mensagem “Aprovado em exame”; caso contrário, apresentar a mensagem “Reprovado”. Informar junto de cada mensagem o valor da média obtida.
- e) Ler três valores numéricos (representados pela variáveis A, B e C) e efetuar o cálculo da equação completa de segunda grau, utilizando a fórmula de Baskara (considerar todas as possíveis condições para delta: delta < 0, delta > 0 e delta = 0). Lembre-se de que é completa a equação de segundo grau que possui simultaneamente as variáveis A, B e C diferentes de zero.
- f) Ler três valores e apresentá-los dispostos em ordem crescente. Utilizar os conceitos de propriedade distributiva (exercício “g” do capítulo 3) e troca de valores entre variáveis (exercício “f” do capítulo 3).
- g) Ler quatro valores numéricos inteiros e apresentar os valores que são divisíveis por 2 e 3.
- h) Ler quatro valores numéricos inteiros e apresentar os valores que são divisíveis por 2 ou 3.
- i) Ler cinco valores numéricos inteiros, identificar e apresentar o maior e o menor valores informados. Não execute a ordenação dos valores.
- j) Ler um valor numérico inteiro e apresentar uma mensagem informando se o valor numérico informado é par ou ímpar.
- k) Ler um valor numérico inteiro que esteja na faixa de valores de 1 até 9. O programa deve apresentar a mensagem “O valor está na faixa permitida”, caso o valor informado esteja entre 1 e 9. Se o valor estiver fora da faixa, o programa deve apresentar a mensagem “O valor está fora da faixa permitida”.
- l) Ler um valor numérico inteiro qualquer e fazer a sua apresentação caso o valor não seja maior que três. Utilize apenas o operador lógico `não`, para a solução deste problema.
- m) Ler o nome e o sexo de uma pessoa e apresentar como saída uma das seguintes mensagens: “Ilmo. Sr.”, caso seja informado o sexo como masculino, ou “Ilma. Sra.”, caso seja informado o sexo como feminino. Apresentar também junto de cada mensagem de saudação o nome previamente informado.

CAPÍTULO 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 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 à variável de resposta R, e apresente o valor obtido, repetindo esta seqüência por cinco vezes, conforme mostrado abaixo em português estruturado:

```
programa PEDE_NÚMERO
var
    X : inteiro
    R : inteiro
inicio
    leia X
    R ← X * 3
    escreva R
    leia X
    R ← X * 3
    escreva R
    leia X
    R ← X * 3
    escreva R
    leia X
    R ← X * 3
```

```

escreva R
leia X
R ← X * 3
escreva R
fim

```

Para estes casos existem comandos 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...faça...fim_enquanto**.

A estrutura **enquanto...faça...fim_enquanto** 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.

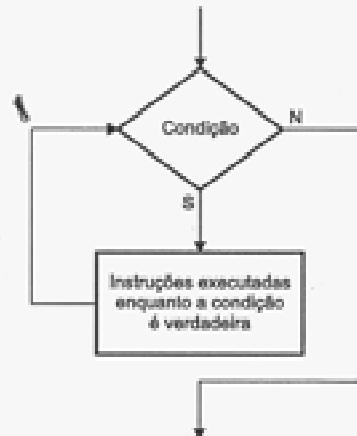


Figura 5.1 - Estrutura de funcionamento da instrução **enquanto...faça...fim_enquanto**.

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 do tipo contador, definida no passo 1;
- 7 - Quando contador for maior que 5, encerrar o processamento do looping.

Diagrama de Blocos

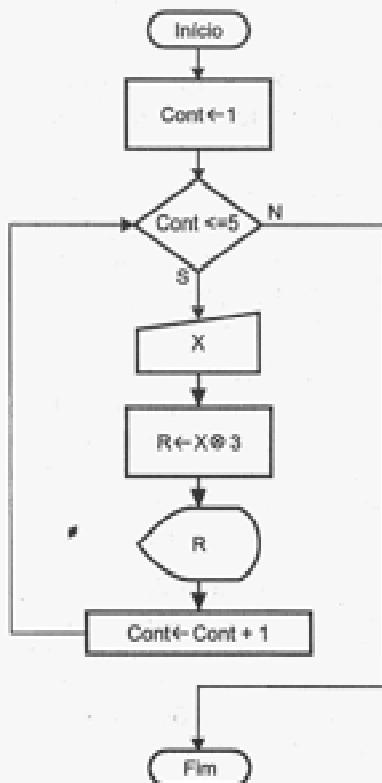


Figura 5.2 - Exemplo da utilização da estrutura enquanto...faça...fim_enquanto.

Português Estruturado

```
programa LOOPING_1A
var
  X, R : inteiro
  CONT : inteiro
início
  CONT ← 1
  enquanto (CONT <= 5) faça
    leia X
    R ← X * 3
    escreva R
    CONT ← CONT + 1
  fim_enquanto
fim
```

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) **faça** efetua a checagem da condição estabelecida, verificando que a condição é verdadeira, pois o valor da variável *CONT* que 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 **fim_enquanto**. Depois de efetuar a primeira leitura, cálculo e apresentação do valor calculado, o programa encontra a linha *CONT* ← *CONT* + 1, indicando o contador. Observe que a variável *CONT* possui neste momento o valor 1 e somado a mais 1 esta passa a ter o valor 2, ou seja, *CONT* ← *CONT* + 1, sendo assim *CONT* ← 1 + 1 que resultará *CONT* = 2.

Agora que a variável *CONT* possui o valor 2, o processamento do programa volta para a instrução **enquanto** (*CONT* <= 5) **faça**, 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) **faça** 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 **fim_enquanto**, que no caso é a instrução **fim**, dando o encerramento do programa.

Para ilustrar de forma um pouco diferente, imagine que o problema anterior deverá ser executado enquanto o usuário quiser. 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

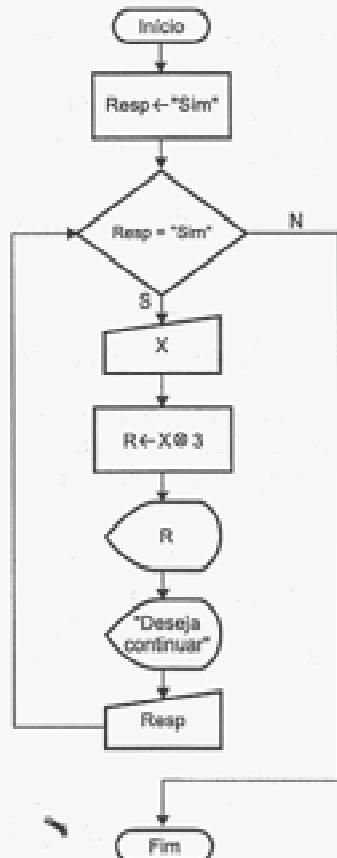


Figura 5.3 - Exemplo de um looping controlado pelo usuário.

Português Estruturado

```
programa LOOPING_1B
var
    X, R : inteiro
    RESP : caractere
início
    RESP ← "SIM"
    enquanto (RESP = "SIM") faça
        leia X
        R ← X * 3
        escreva R
        escreva "Deseja continuar?"
        leia RESP
    fim_enquanto
fim
```

Veja que o contador foi substituído pela variável *RESP*, que enquanto for igual a "SIM" executará a rotina existente entre as instruções **enquanto** e **fim_enquanto**. 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*.

Neste momento faça os exercícios de fixação 1 do tópico 5.7, antes de iniciar o estudo do próximo looping.

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...até_que**.

A estrutura **repita...até_que** 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 de resposta R, e apresentar o valor obtido, repetindo esta seqüência por cinco vezes".

Algoritmo

- 1 - Criar uma variável contador;
- 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 um ao contador;
- 6 - Repetir os passos 2, 3, 4 e 5 até que o contador seja maior que 5.

Diagrama de Blocos

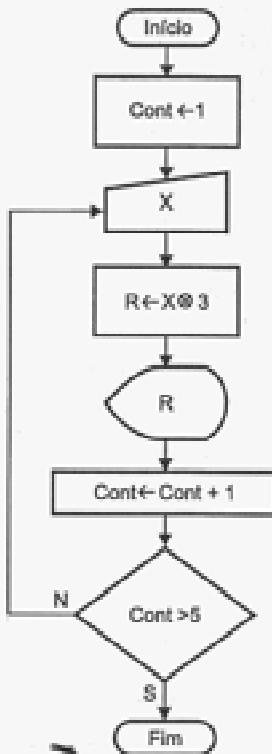


Figura 5.4 - Exemplo da utilização da instrução **repita...até que**.

Português Estruturado

```
programa LOOPING_2A
var
    X, R : inteiro
    CONT : inteiro
início
    CONT ← 1
    repita
        leia X
        R ← X * 3
        escreva R
        CONT ← CONT + 1
    até_que (CONT > 5)
fim
```

Logo após o início do programa, a variável contador é atribuída com o valor 1 ($CONT \leftarrow 1$). Em seguida, a instrução **repita** indica que todo trecho de instruções situado até a instrução **até_que** deverá ter o 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** e **até_que**. Depois de efetuar a primeira execução das instruções, o programa encontra a linha $CONT \leftarrow CONT + 1$; 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 **até_que** ($CONT > 5$), 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 não.

Diagrama de Blocos

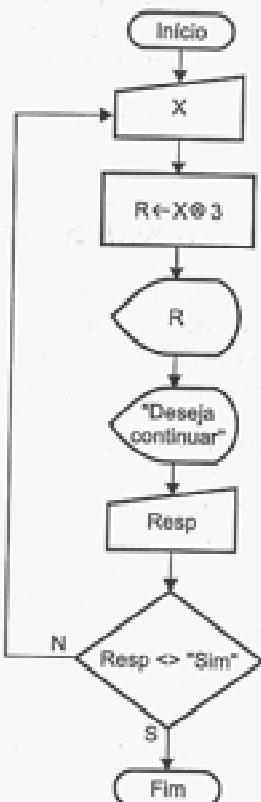


Figura 5.5 - Exemplo de um looping controlado pelo usuário.

Português Estruturado

```
programa LOOPING_2B
var
  X, R : inteiro
  RESP : caractere
inicio
  repita
    leia X
    R ← X * 3
    escreva R
    escreva “deseja continuar?”
    leia RESP
    até_que (RESP <> “SIM”)
fim
```

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, para somente no final ser verificada a condição, no caso *RESP* = "SIM".

Neste momento faça os exercícios de fixação do tópico 5.7, antes de iniciar o estudo do próximo looping.

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...até...passo...faça...fim_fora*.

A estrutura *para...de...até...passo...faça...fim_fora* 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 representação 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. Para este livro nós estamos adotando o formato representado abaixo:

Diagrama de Blocos

Com relação ao diagrama de blocos, será indicada a variável a ser controlada com a implicaçāo dos valores de inicio, 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.



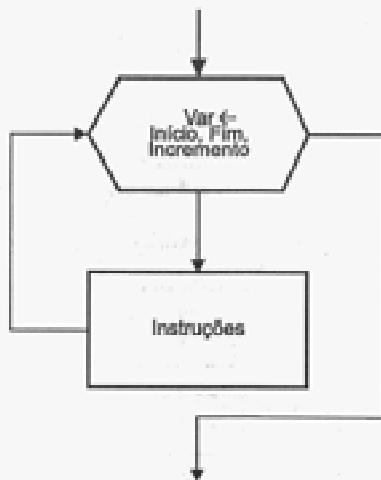


Figura 5.6 - Estrutura do símbolo da instrução para...fim_para.

Português Estruturado

```

para <variável> de <início> até <fim> passo <incremento> faça
    <instruções>
    fim_para

```

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

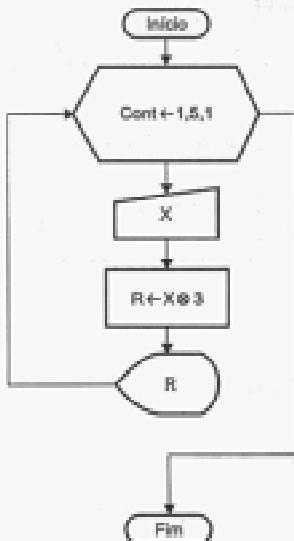


Figura 5.7 - Exemplo da utilização da estrutura para...de...até...passo...faça...fim_para.

Português Estruturado

```
programa LOOPING_2A
var
    X, R : inteiro
    CONT : inteiro
início
    para CONT de 1 até 5 passo 1 faça
        leia X
        R ← X * 3
        escreva R
    fim_para
fim
```

Será executado o conjunto de instruções entre a instrução **para** e a instrução **fim_para**, 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.

Neste momento faça o exercício de fixação 3 do tópico 5.7.

5.4 - Consideração 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**. Porem 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. Aguarde.

5.5.1 - Encadeamento de Estrutura Enquanto com Enquanto

Exemplo de encadeamento de estrutura **enquanto** com estrutura **enquanto**. A figura 5.8 mostra a estrutura do desenho do diagrama de blocos.

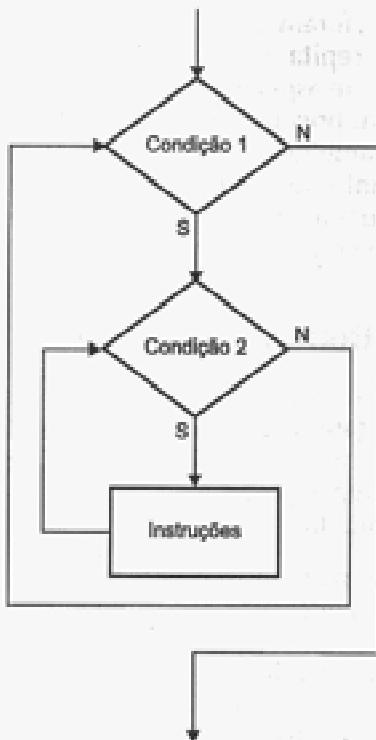


Figura 5.8 - Encadeamento de enquanto com enquanto.

Português Estruturado

```
enquanto (<condição1>) faça
    enquanto (<condição2>) faça
        <instruções>
    fim_enquanto
fim_enquanto
```

5.5.2 - Encadeamento de Estrutura Enquanto com Repita

Exemplo de encadeamento de estrutura **enquanto** com estrutura **repita**. A figura 5.9 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

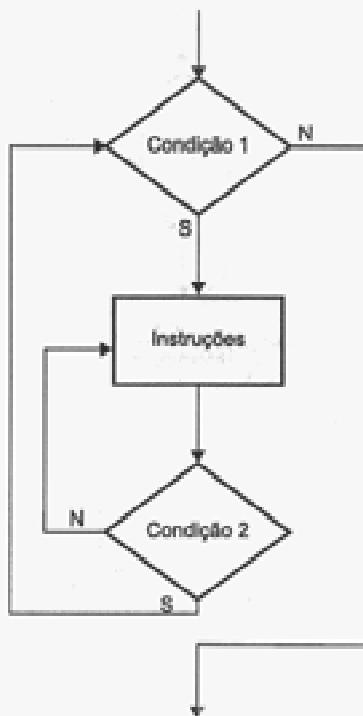


Figura 5.9 - Encadeamento de enquanto com repita.

Português Estruturado

```
enquanto (<condição1>) faça
    repita
        <instruções>
        até_que (<condição2>)
    fim_enquanto
```

5.5.3 - Encadeamento de Estrutura Enquanto com Para

Exemplo de encadeamento de estrutura enquanto com estrutura para. A figura 5.10 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

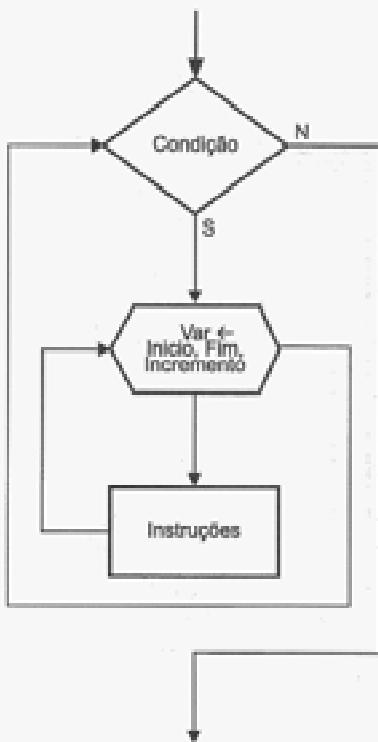


Figura 5.10 - Encadeamento de enquanto com para.

Português Estruturado

```
enquanto (<condição>) faça
    para <var> de <inicio> até <fim> passo <incr> faça
        <instruções>
    fim_para
fim_enquanto
```

5.5.4 - Encadeamento de Estrutura Repita com Repita

Exemplo de encadeamento de estrutura repita com estrutura repita. A figura 5.11 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

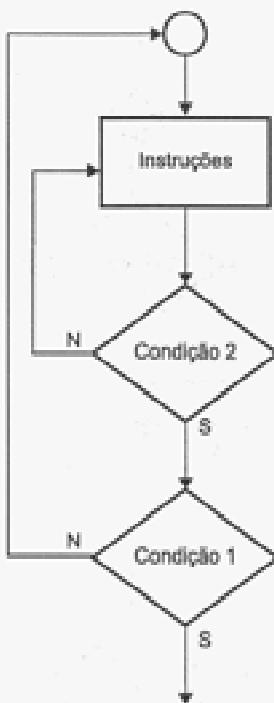


Figura 5.11 - Encadeamento de repita com repita.

Português Estruturado

```
repita
    repita
        <instruções>
    até_que (<condição2>)
até_que (<condição1>)
```

5.5.5 - Encadeamento de Estrutura Repita com Enquanto

Exemplo de encadeamento de estrutura repita com estrutura enquanto. A figura 5.12 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

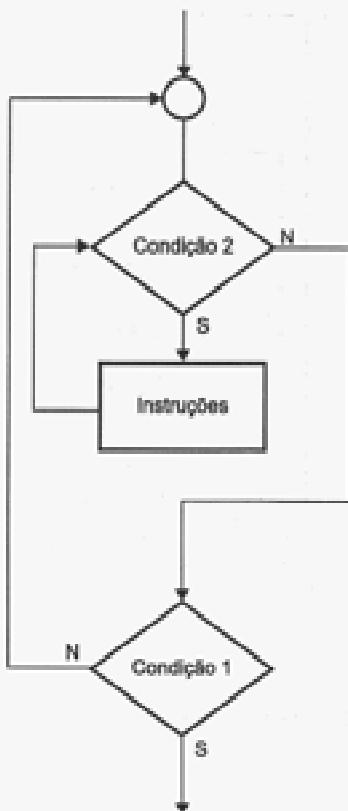


Figura 5.12 - Encadeamento de repita com enquanto.

Português Estruturado

```
repita
    enquanto (<condição2>) faça
        <instruções>
    fim_enquanto
    até_que (<condição1>)
```

5.5.6 - Encadeamento de Estrutura Repita com Para

Exemplo de encadeamento de estrutura repita com estrutura para. A figura 5.13 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

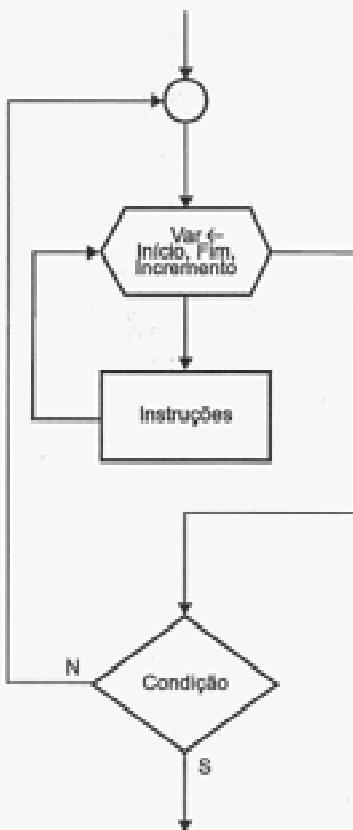


Figura 5.13 - Encadeamento de repita com para.

Português Estruturado

```
repita
    para <var> de <inicio> até <fim> passo <incr> faça
        <instruções>
    fim_para
até_QUE (<condição>)
```

5.5.7 - Encadeamento de Estrutura Para com Para

Exemplo de encadeamento de estrutura para com estrutura para. A figura 5.14 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

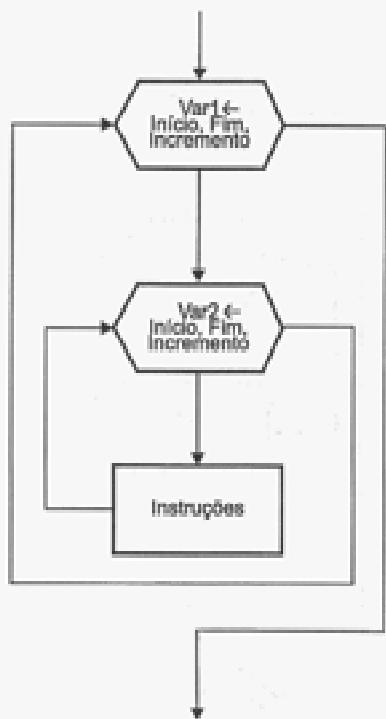


Figura 5.14 - Encadeamento de para com para.

Português Estruturado

```
para <var1> de <inicio> até <fim> passo <incr> faça
    para <var2> de <inicio> até <fim> passo <incr> faça
        <instruções>
    fim_para
fim_para
```

5.5.8 - Encadeamento de Estrutura Para com Enquanto

Exemplo de encadeamento de estrutura para com estrutura enquanto. A figura 5.15 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

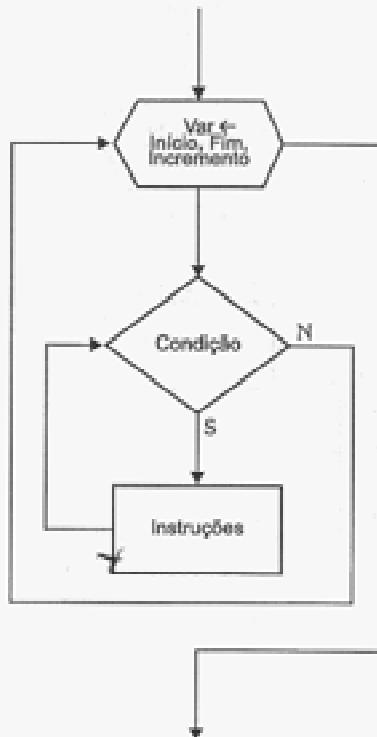


Figura 5.15 - Encadeamento de para com enquanto.

Português Estruturado

```
para <var> de <inicio> até <fim> passo <incr> faça
    enquanto (<condição>) faça
        <instruções>
    fim_enquanto
fim_para
```

5.5.9 - Encadeamento de Estrutura Para com Repita

Exemplo de encadeamento de estrutura `para` com estrutura `repita`. A figura 5.16 mostra a estrutura do desenho do diagrama de blocos.

Diagrama de Blocos

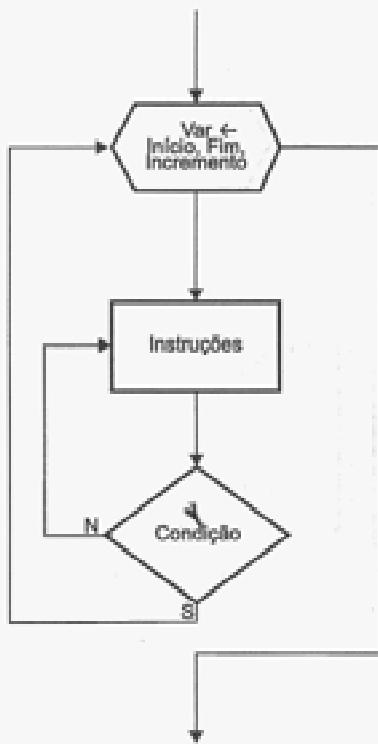


Figura 5.16 - Encadeamento de para com repita.

Português Estruturado

```
para <var> de <início> até <fim> passo <incr> faça
    repita
        <instruções>
    até_que (<condição>)
    fim_para
```

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 da fatorial do número 5, 5!". Desta forma, temos que $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$ ou $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$, equivalente a 120.

Fatorial é o produto dos números naturais desde 1 até o inteiro n. Sendo assim, o cálculo de uma 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. Veja em seguida:

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 factorial 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 factorial 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 indicada 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 ← FATORIAL * CONTADOR;
- 4 - Incrementar 1 à variável CONTADOR;
- 5 - Apresentar ao final o valor obtido.

Abaixo segue a resolução do problema do cálculo de factorial, 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

Resolução do problema usando a estrutura de repetição enquanto.

Diagrama de Blocos

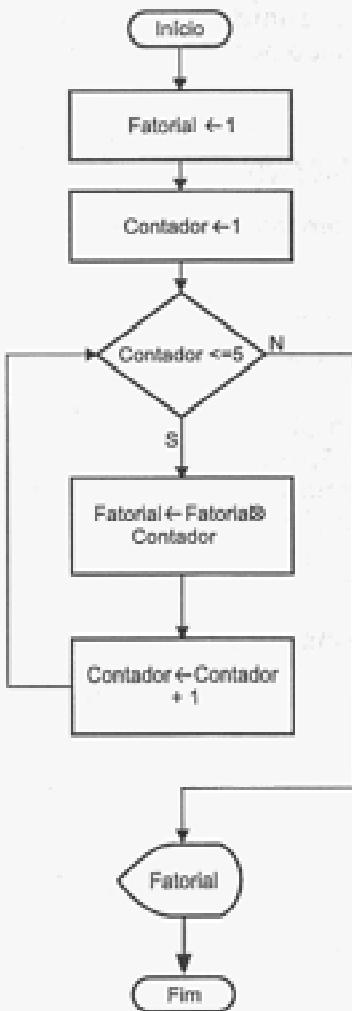


Figura 5.17 - Fatorial com estrutura enquanto.

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 da fatorial, e o segundo sendo utilizado para controlar a execução do looping e 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

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 \leftarrow Fatorial * Contador	Comentários
1	1	1	Valor inicial das variáveis e da fatorial
2	1	2	Cálculo da fatorial com o contador em 2
3	2	6	Cálculo da fatorial com o contador em 3
4	6	24	Cálculo da fatorial com o contador em 4
5	24	120	Cálculo da fatorial com o 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

```
programa FATORIAL_A
var
    CONTADOR : inteiro
    FATORIAL : inteiro
inicio
    FATORIAL  $\leftarrow$  1
    CONTADOR  $\leftarrow$  1
    enquanto (CONTADOR  $\leq$  5) faça
        FATORIAL  $\leftarrow$  FATORIAL * CONTADOR
        CONTADOR  $\leftarrow$  CONTADOR + 1
    fim_enquanto
    escreva "Fatorial de 5 é = ", FATORIAL
fim
```

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 separar as duas informações.

Exemplo 2 - Estrutura de repetição: Repita

Resolução do problema fazendo uso da estrutura de repetição repita.

Diagrama de Blocos

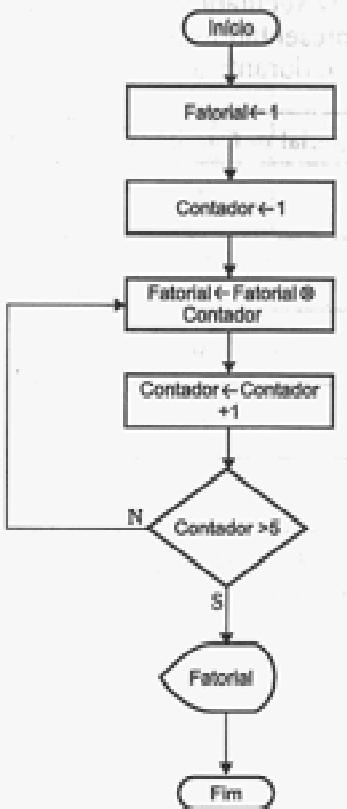


Figura 5.18 - Fatorial com estrutura repita.

Português Estruturado

```
programa FATORIAL_B
var
    CONTADOR : inteiro
    FATORIAL : inteiro
início
    FATORIAL ← 1
    CONTADOR ← 1
    repita
        FATORIAL ← FATORIAL * CONTADOR
        CONTADOR ← CONTADOR + 1
    até_que (CONTADOR > 5)
    escreva "Fatorial de 5 é = ", FATORIAL
fim
```

Exemplo 3 - Estrutura de repetição: Para

Resolução do problema fazendo uso da estrutura de repetição para.

Diagrama de Blocos

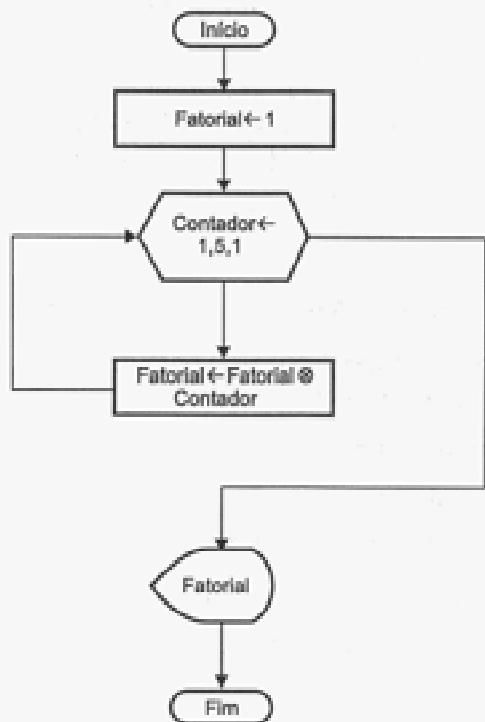


Figura 5.19 - Fatorial com estrutura para.

Português Estruturado

```
programa FATORIAL_C
var
    CONTADOR : inteiro
    FATORIAL : inteiro
inicio
    FATORIAL ← 1
    para CONTADOR de 1 até 5 passo 1 faça
        FATORIAL ← FATORIAL * CONTADOR
    fim_para
    escreva "Fatorial de 5 é = ", FATORIAL
fim
```

Exemplo 4

Observe que o algoritmo apresentado anteriormente para a resolução do problema de fatorial soluciona o cálculo apenas para a fatorial de 5. Seria melhor possuir a solução aberta para que um programa calcule a fatorial de um número qualquer, e que pudesse calcular outras 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 da fatorial. O exemplo seguinte faz uso do encadeamento de estruturas de repetição.

Algoritmo

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

Diagrama de Blocos

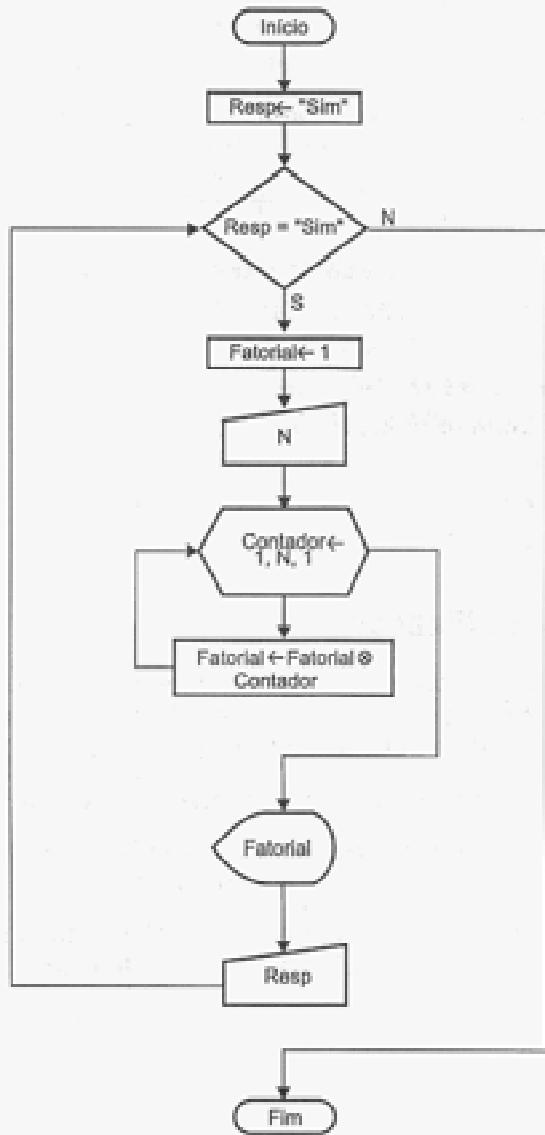


Figura 5.20 - Implementação do número de vezes e cálculo de qualquer fatorial.

Português Estruturado

```
programa FATORIAL_D
var
  CONTADOR : inteiro
  FATORIAL : inteiro
```

```

RESP      : caractere
N        : inteiro
inicio
  RESP ← "SIM"
  enquanto (RESP = "SIM") faça
    FATORIAL ← 1
    escreva "Fatorial de que número: "
    leia N
    para CONTADOR de 1 até N passo 1 faça
      FATORIAL ← FATORIAL * CONTADOR
    fim_para
    escreva "Fatorial de ", N , "é = ", FATORIAL
    escreva "Deseja continuar?"
    leia RESP
  fim_enquanto
fim

```

5.7 - Exercício de Fixação

1 - Desenvolva os algoritmos, seus respectivos diagramas e codificação em português estruturado. Usar na resolução dos problemas seguintes apenas a estrutura de repetição do tipo enquanto.

- Apresentar os quadrados dos números inteiros de 15 a 200.
- Apresentar os resultados de uma tabuada de um número qualquer, a qual deve ser impressa no seguinte formato:

Considerando como exemplo o fornecimento do número 2 para o número qualquer, ter-se-ia a seguinte situação:

2 X 1 = 2
 2 X 2 = 4
 2 X 3 = 6
 2 X 4 = 8
 2 X 5 = 10
 (...)
 2 X 10 = 20

- Apresentar o total da soma obtida dos cem primeiros números inteiros ($1+2+3+4+5+6+7+\dots+97+98+99+100$).
- 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 deverá ser iniciada com 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. Deverá ser apresentado, observando a seguinte definição:

$$3^0 = 1$$

$$3^1 = 3$$

$$3^2 = 9$$

(...)

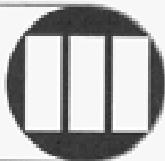
$$3^{15} = 14348907$$

- 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. Considere apenas a entrada de valores inteiros e positivos.
- 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 posterior com o seu anterior subsequente.
- 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 deverá apresentar os valores das duas temperaturas.
- k) 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 quadro colocam-se 8 grãos (tendo-se então 15 grãos) até atingir o sexagésimo quarto quadro (Este exercício foi baseado numa situação exposta no capítulo 16 do livro "O Homem que Calculava" de Malba Tahan, da Editora Record).

- I) 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.
 - m) Elaborar um programa que efetue a leitura de 10 valores numéricos e apresente no final o total do somatório e a média dos valores lidos.
 - n) Elaborar um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o total do somatório, a média 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 (menor que zero).
 - o) Elaborar um programa que apresente como resultado o valor da fatorial dos valores ímpares situados na faixa numérica de 1 a 10.
 - p) Elaborar um programa que apresente os resultados da soma e da média aritmética dos valores pares situados na faixa numérica de 50 a 70.
 - q) 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 "NÃO", o programa deve apresentar o valor total acumulado da área residencial.
 - r) 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.
 - s) 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.
- 2 - Desenvolva os algoritmos, seus respectivos diagramas e codificação em português estruturado dos exercícios elencados de a até s (exercício 1), usando a estrutura de repetição do tipo repita.
- 3 - Desenvolva os algoritmos, seus respectivos diagramas e codificação em português estruturado dos exercícios elencados de a até s (exercício 1), usando a estrutura de repetição do tipo para.

Pode ocorrer o fato de algum exercício não poder ser solucionado com este tipo de estrutura de repetição.

PARTE



Estruturas Básicas de Dados - Tabelas em Memória

CAPÍTULO 6

Estruturas 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) 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 de 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 uma determinada situação, considere o seguinte problema: "Calcular e apresentar 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 médias e dividi-las por 8. A tabela seguintes 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 tem-se:

```
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:

```
programa MÉDIA_TURMA  
var  
    MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8 : real  
    SOMA, MÉDIA : real
```

início

```
SOMA ← 0  
leia MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8  
SOMA ← MD1 + MD2 + MD3 + MD4 + MD5 + MD6 + MD7 + MD8  
MÉDIA ← SOMA / 8  
escreva MÉDIA  
fim
```

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 um 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 poderá-se 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 **conjunto** que indicará em português estruturado a utilização de uma matriz, tendo como sintaxe: VARIÁVEL : **conjunto[<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 diagrama de blocos e codificação em português estruturado da leitura das notas dos 8 alunos, cálculo da média e a sua apresentação.

Diagrama de Blocos

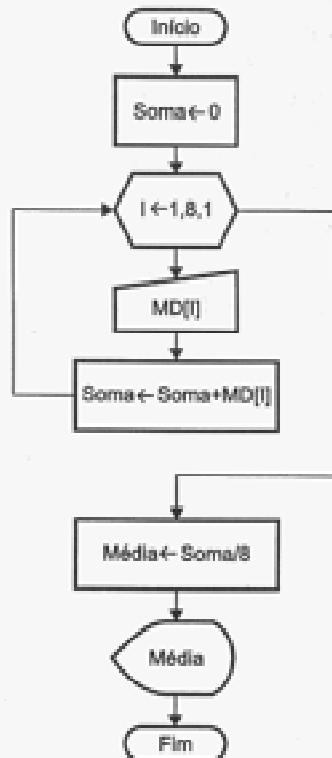


Figura 6.1 - Diagrama de blocos para leitura dos elementos de uma matriz tipo vetor.

```

programa MÉDIA_TURMA
var
  MD : conjunto[1..8] de real
  SOMA, MÉDIA : real
  I : inteiro
inicio
  SOMA ← 0
  para I de 1 até 8 passo 1 faça
    leia MD[I]
    SOMA ← SOMA + MD[I]
  fim_para
  MÉDIA ← SOMA / 8
  escreva MÉDIA
fim

```

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. Abaixo são apresentados o diagrama de blocos e a codificação em português estruturado da escrita das notas dos 8 alunos antes de ser apresentado o cálculo da média.

Diagrama de Blocos

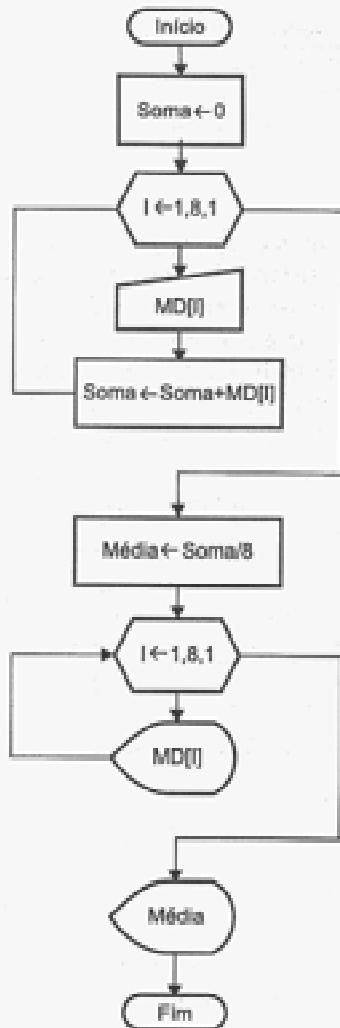


Figura 6.2 - Diagrama de bloco para escrita dos elementos de uma matriz tipo vetor.

Português Estruturado

```
programa MÉDIA_TURMA
var
    MD : conjunto[1..8] de real
    SOMA, MÉDIA : real
    I : inteiro
início
    SOMA ← 0
    para I de 1 até 8 passo 1 faça
        leia MD[I]
        SOMA ← SOMA + MD[I]
    fim_para
    MÉDIA ← SOMA / 8
    para I de 1 até 8 passo 1 faça
        escreva MD[I]
    fim_para
    escreva MÉDIA
fim
```

6.3 - Exercício de Aprendizagem

Para demonstrar a utilização de uma matriz em um exemplo um pouco maior, considere como problemas os exemplos apresentados em seguida:

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 somado 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 como 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 par (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.

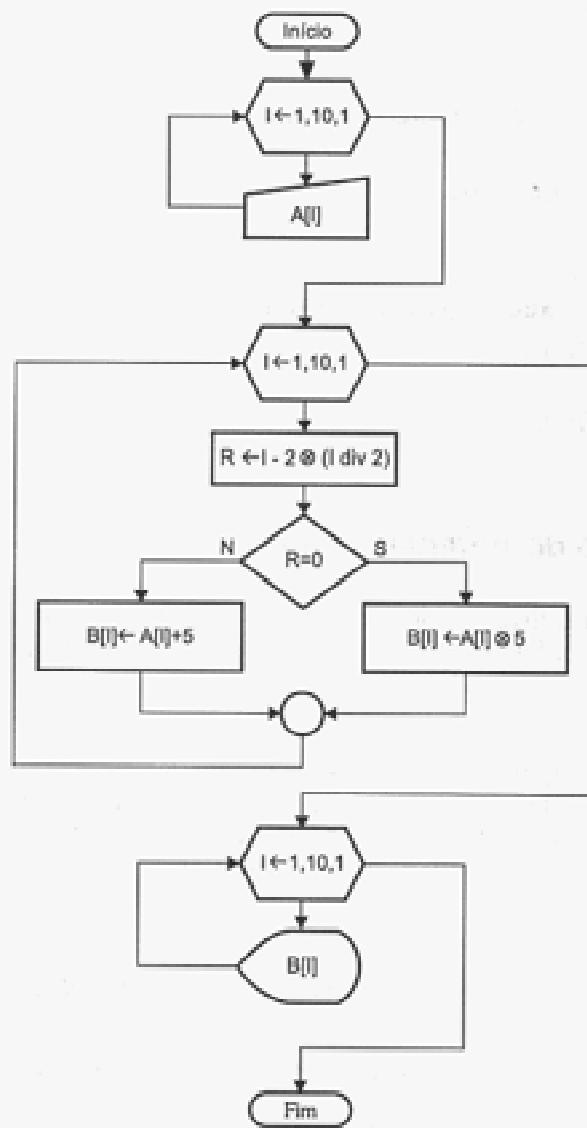


Figura 6.3 - Diagrama de blocos para o 1º exemplo.

Português Estruturado

```
programa ÍNDICE_PAR_OU_IMPAR
var
  A, B : conjunto[1..10] de real
  I, R : inteiro
início
  para I de 1 até 10 passo 1 faça
    leia A[I]
  fim_para
  para I de 1 até 10 passo 1 faça
    R ← I - 2 * (I div 2)
    se (R=0) então
      B[I] ← A[I] * 5
    senão
      B[I] ← A[I] + 5
    fim_se
  fim_para
  para I de 1 até 10 passo 1 faça
    escreva B[I]
  fim_para
fim
```

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 elemento 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.

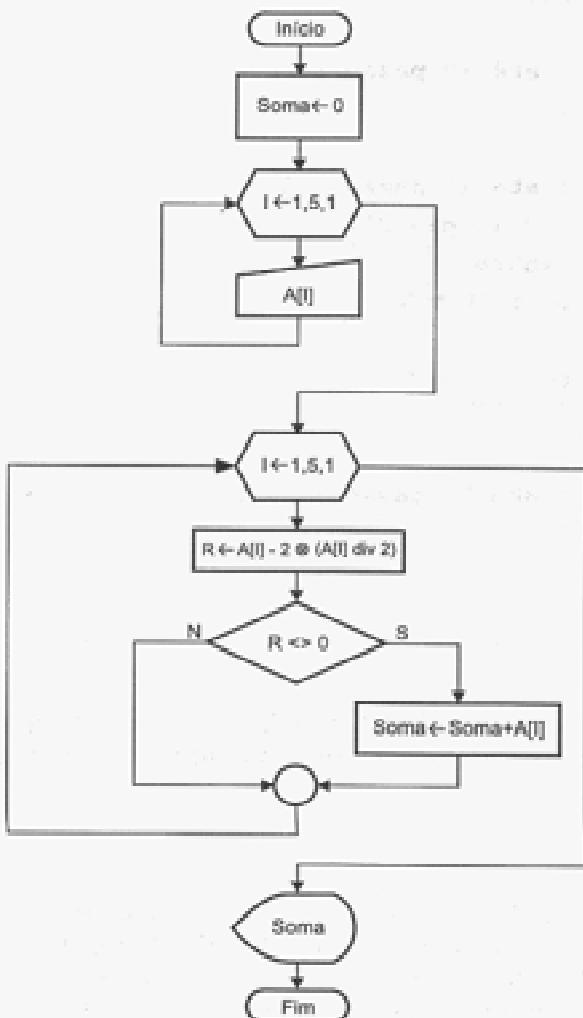


Figura 6.4 - Diagrama de blocos para o 2º exemplo.

Português Estruturado

```
programa ELEMENTO_IMPAR
var
  A           : conjunto[1..5] de inteiro
  R, I, SOMA : inteiro
```

```

    inicio
        SOMA ← 0
        para I de 1 até 5 passo 1 faça
            leia A[I]
        fim_para
        para I de 1 até 5 passo 1 faça
            R ← A[I] - 2 * (A[I] div 2)
            se (R<>0) então
                SOMA ← SOMA + A[I]
            fim_se
        fim_para
        escreva SOMA
    fim

```

6.4 - Exercício de Fixação

- 1) Desenvolva os algoritmos, diagrama de blocos e codificação em português estruturado dos seguintes exercícios:
 - 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[1] deverá ser implicado pelo elemento A[1] * 3, o elemento B[2] implicado pelo elemento A[2] * 3 e assim por diante, até 8. Apresentar a matriz B.
 - c) Ler duas matrizes A e B do tipo vetor com 20 elementos. Construir uma matriz C, onem que de 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 A do tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: "Todo elemento de B deverá ser o quadrado do elemento de A correspondente". Apresentar as matrizes A e B.
 - e) 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.
 - f) 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.

- g) 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.
- h) Ler 20 elementos de uma matriz A tipo vetor e construir uma matriz B de mesma dimensão com os mesmos 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 de B e assim por diante. Apresentar as matrizes A e B.
- i) 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.
- j) Ler uma matriz A do tipo vetor com 20 elementos. Construir uma matriz B do mesmo tipo da matriz A, sendo que cada elemento de B seja o somatório do elemento correspondente da matriz A. Se o valor do elemento de A [1] for 5, B [1] deverá ser 15 e assim por diante. Apresentar a matriz B.
- k) Ler uma matriz A do tipo vetor com 10 elementos positivos. Construir uma matriz B de mesmo tipo e dimensão, em que cada elemento da matriz B deverá ser o valor negativo do elemento correspondente da matriz A. Desta forma, se em A[1] estiver armazenado o elemento 8, deverá estar em B[1] o valor -8, e assim por diante. Apresentar os elementos da matriz B.
- l) Ler uma matriz A tipo vetor com 10 elementos. Construir uma matriz B de mesmo tipo, sendo que cada elemento de B deverá ser a metade de cada elemento de A. Apresentar os elementos das matrizes A e B.
- m) Elaborar um programa que efetue o cálculo de uma tabuada de um número qualquer e armazene os resultados em uma matriz A de uma dimensão para 10 elementos. Apresentar os valores armazenados na matriz.
- n) Ler 20 elementos (valores reais) para temperaturas em graus Celsius em uma matriz A de uma dimensão do tipo vetor. O programa deverá apresentar a menor, a maior e a média das temperaturas lidas.
- o) Ler 25 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.
- p) Ler 12 elementos inteiros 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 for ímpar deverá ser multiplicado por 2; caso contrário, o elemento da matriz A deverá permanecer constante". Apresentar a matriz B.

- q) 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 a matriz B.
- r) Ler 6 elementos (valores inteiros) para as matrizes A e B de uma dimensão do tipo vetor. Construir as matrizes C e D de mesmo tipo e dimensão, sendo que a matriz C deverá ser formada pelos elementos de índice ímpar das matrizes A e B, e a matriz D deverá ser formada pelos elementos de índice par das matrizes A e B. Apresentar as matrizes C e D.
- s) 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.
- t) Ler duas matrizes A e B de uma dimensão com 10 elementos. A matriz A deverá aceitar apenas a entrada de valores que sejam divisíveis por 2 e 3, enquanto a matriz B deverá aceitar apenas a entrada de valores que 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, de modo que a matriz C contenha 20 elementos. Apresentar a matriz C.
- u) 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, de forma que a matriz C contenha 24 elementos. Apresentar a matriz C.
- v) Ler uma matriz A de uma dimensão do tipo vetor com 30 elementos do tipo inteiro. Ao final apresentar a quantidade de valores pares e ímpares existentes na referida matriz.
- w) Ler duas matrizes A e B de uma dimensão do tipo vetor com dez elementos inteiros cada. Construir uma matriz C de mesmo tipo e dimensão que seja formada pelo quadrado da soma dos elementos correspondentes nas matrizes A e B.
- x) Ler uma matriz A de uma dimensão com 6 elementos do tipo real. Construir uma matriz B, onde cada posição de índice ímpar da matriz B

deve ser atribuído com um elemento de índice par existente na matriz A e cada posição de índice par da matriz B deve ser atribuído com um elemento de índice ímpar existente na matriz A. Apresentar as duas matrizes.

- y) Ler uma matriz A de uma dimensão com 15 elementos numéricos inteiros. Apresentar o total de elementos pares existentes na matriz.
- z) Ler uma matriz A de uma dimensão com 10 elementos numéricos inteiros. Apresentar o total de elementos ímpares existentes na matriz e também o percentual do valor total de números ímpares em relação a quantidade total de elementos armazenados na matriz.

CAPÍTULO 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 outra idéia, considere o seguinte problema: "Criar um programa que efetue a leitura dos nomes de 20 pessoas e em seguida apresentá-los na mesma ordem em que foram informados".

Algoritmo

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

Diagrama de Blocos

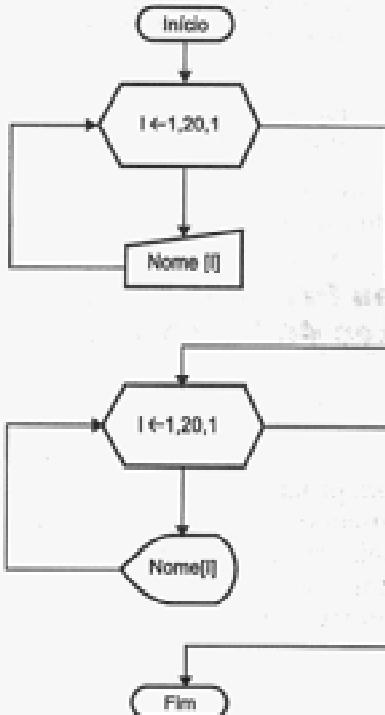


Figura 7.1 - Diagrama de blocos para ler e escrever 20 nomes de pessoas.

Português Estruturado

```
programa LISTA_NOME
var
  NOME : conjunto[1..20] de caractere
  I : inteiro
início
  para I de 1 até 20 passo 1 faça
    leia NOME[I]
  fim_para
  para I de 1 até 20 passo 1 faça
    escreva NOME[I]
  fim_para
fim
```

O programa anterior executa a leitura e a escrita dos 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. Nesta obra, 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 comentadas anteriormente.

Algoritmo

Neste algoritmo, será comentado passo a passo como funciona o processo de ordenação de dados em nível de processamento. Veja os passos em seguida:

- 1 - Definir a variável I do tipo Inteira para controlar a malha de repetição;
- 2 - Definir a matriz NOME do tipo caractere 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 subsequentes. 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 ao lado:

Matriz: A	
Índice	Elemento
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. Convertendo a tabela acima no formato matricial, ter-se-ia então:

```
A[1] = 9  
A[2] = 8  
A[3] = 7  
A[4] = 5  
A[5] = 3
```

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 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 passam a ter a seguinte formação:

```
A[1] = 8  
A[2] = 9  
A[3] = 7  
A[4] = 5  
A[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 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:

```
A[1] = 7  
A[2] = 9  
A[3] = 8  
A[4] = 5  
A[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:

A[1] = 5
A[2] = 9
A[3] = 8
A[4] = 7
A[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:

A[1] = 3
A[2] = 9
A[3] = 8
A[4] = 7
A[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:

A[1] = 3
A[2] = 8
A[3] = 9
A[4] = 7
A[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:

A[1] = 3
A[2] = 7
A[3] = 9
A[4] = 8
A[5] = 5

Será então dado continuidade ao 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:

A[1] = 3
A[2] = 5
A[3] = 9
A[4] = 8
A[5] = 7

Note que até este ponto a posição A[2] foi comparada com todas as posições subsequentes 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]. Serão estes trocados, ficando A[3] com 8 e A[4] com 9, conforme em seguida:

A[1] = 3
A[2] = 5
A[3] = 8
A[4] = 9
A[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:

A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 9
A[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:

A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8
A[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 caractere 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 à 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, ..., 20
2	3, 4, 5, 6, 7, 8, ..., 20
3	4, 5, 6, 7, 8, ..., 20
4	5, 6, 7, 8, ..., 20
5	6, 7, 8, ..., 20
6	7, 8, ..., 20
...	..., 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 que 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 **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] com 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.

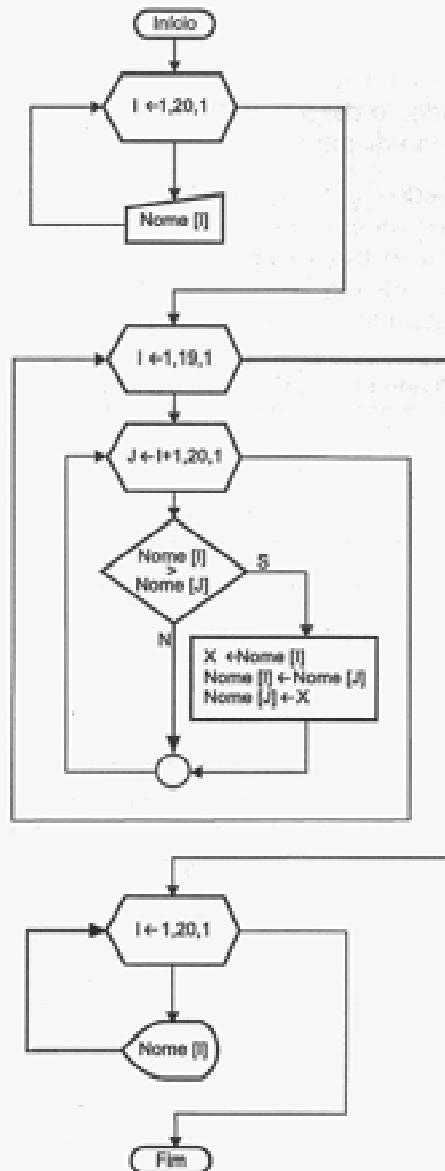


Figura 7.2 - Diagrama de blocos para a ordenação dos nomes.

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

Observe em seguida, a codificação em português estruturado da rotina de ordenação. Atente para o detalhe das instruções para encadeadas. Lembre-se de que no caso de encadeamento, será executada primeiro a rotina mais interna, no caso a rotina de contagem da variável J, passando o processamento para a rotina mais externa, quando a rotina mais interna fechar o seu ciclo.

```
programa LISTA_NOME_ORDERADA
var
    NOME : conjunto[1..20] de caractere
    I, J : inteiro
    X     : caractere
inicio
    (Rotina de entrada de dados)
    para I de 1 até 20 passo 1 faça
        leia NOME[I]
    fim_para
    (Rotina de processamento de ordenação)
    para I de 1 até 19 passo 1 faça
        para J de I + 1 até 20 passo 1 faça
            se (NOME[I] > NOME[J]) então
                X ← NOME[I]
                NOME[I] ← NOME[J]
                NOME[J] ← X
            fim_se
        fim_para
    fim_para
    (Rotina de saída com dados ordenados)
    para I de 1 até 20 passo 1 faça
        escreva NOME[I]
    fim_para
fim
```

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 uma Matriz

Quando se trabalha com matrizes, elas poderão gerar grandes tabelas, dificultando localizar um determinado elemento de forma rápida. Imagine uma matriz possuindo 4000 elementos (4000 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	Nomes
1	André
2	Carlos
3	Frederico
4	Golias
5	Silvia
6	Sílvio
7	Waldir

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

O processo binário consiste em pegar o número de registros 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	Nomes
1	André
2	Carlos
3	Frederico

Segunda parte após primeira divisão

Índice	Nomes
4	Golias
5	Sílvia
6	Silvio
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 segunda divisão

Índice	Nomes
4	Golias
5	Sílvia

Segunda parte após segunda divisão

Índice	Nomes
6	Silvio
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 terceira divisão

Índice	Nomes
6	Silvio

Segunda parte após terceira divisão

Índice	Nomes
7	Waldir

Após a terceira divisão, Waldir é encontrado na segunda parte da tabela. Se estiver sendo pesquisado o elemento Washington, 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

O algoritmo abaixo indicado estabelece a entrada de 10 nomes e a apresentação de nomes que venham a ser solicitados durante a fase de pesquisa.

- 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 o 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.

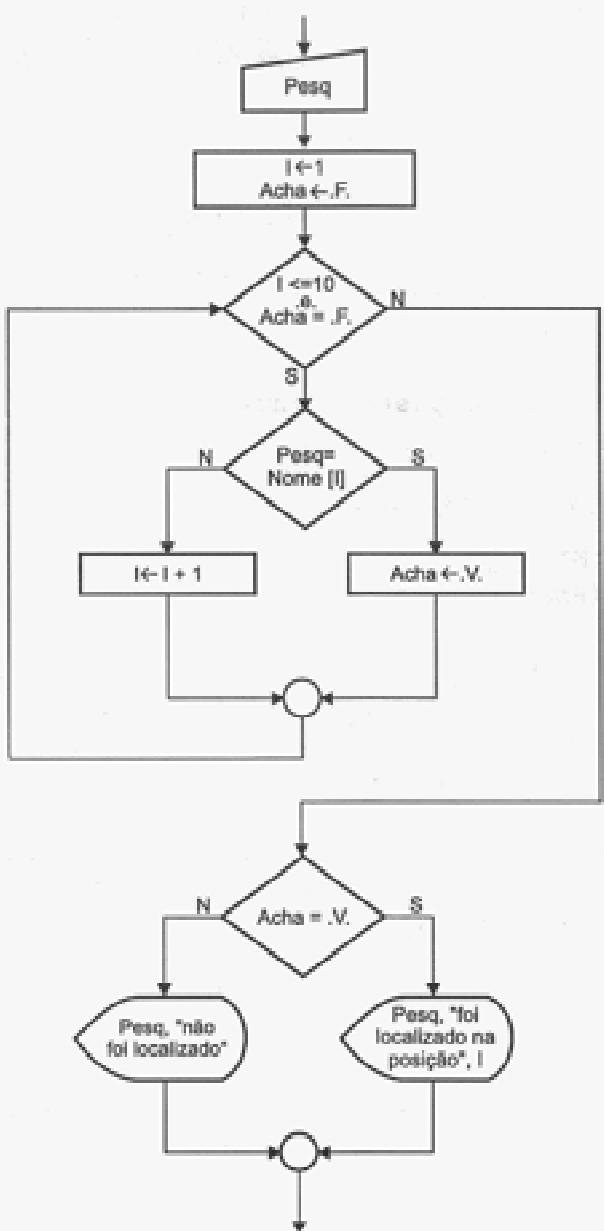


Figura 7.3 - Diagrama de blocos para pesquisa seqüencial.

Português Estruturado

O programa seguinte demonstra a utilização da rotina de pesquisa seqüencial dentro de um contexto prático.

```
programa PESQUISA_SEQUENCIAL
var
    NOME : conjunto[1..10] of caractere
    I    : inteiro
    PESQ, RESP : caractere
    ACHA : lógico
início
    para I de 1 até 10 passo 1 faça
        leia NOME[I]
    fim_para

    RESP = "SIM"
    enquanto (RESP = "SIM") faça

        (Início da rotina de pesquisa)
        escreva "Entre o nome a ser pesquisado: "
        leia PESQ
        I ← 1
        ACHA ← Falso.
        enquanto (I <= 10) .e. (ACHA = .Falso.) faça
            se (PESQ = NOME[I]) então
                ACHA ← Verdadeiro.
            senão
                I ← I + 1
            fim_se
        fim_enquanto
        se (ACHA = Verdadeiro.) então
            escreva PESQ, " foi localizado na posição ", I
        senão
            escreva PESQ, " não foi localizado."
        fim_se
        (Fim da rotina de pesquisa)

        escreva "Deseja continuar? "
        leia RESP
    fim_enquanto
fim
```

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

```
1 -  leia PESQ
2 -  I ← 1
3 -  ACHA ← .Falso.
4 -  enquanto (I <= 10) .e. (ACHA = .Falso.) faça
5 -    se (PESQ = NOME[I]) então
6 -      ACHA ← .Verdadeiro.
7 -    senão
8 -      I ← I + 1
9 -    fim_se
10 -   fim_enquanto
11 -   se (ACHA = .Verdadeiro.) então
12 -     escreva PESQ, "foi localizado na posição", I
13 -   senão
14 -     escreva PESQ, "não foi localizado"
15 -   fim_se
```

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 a 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 estava "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

O algoritmo abaixo indica estabelece a entrada de 10 nomes e apresentação somente de nomes que venham a ser solicitados durante a fase de pesquisa.

- 1 - Iniciar um contador, pedir a leitura de 10 nomes e colocá-los em ordem alfabética;
- 2 - Criar um laço de repetição 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 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 o primeiro exemplo, este também se concentra somente no trecho de pesquisa binária, uma vez que os demais algoritmos já são conhecidos.

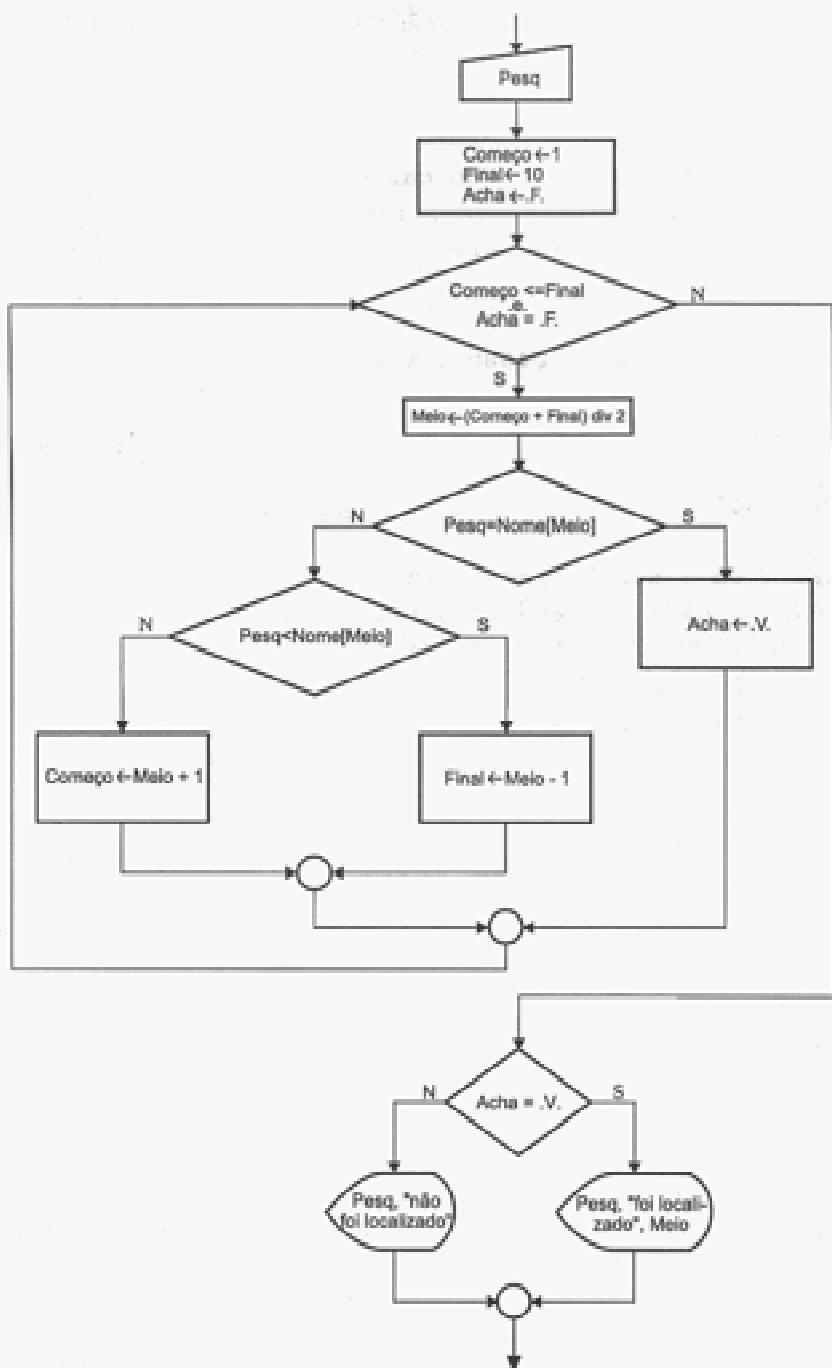


Figura 7.4 - Diagrama de blocos para pesquisa binária.

Português Estruturado

O programa abaixo demonstra a utilização da rotina de pesquisa binária dentro de um contexto prático.

```
programa PESQUISA_BINÁRIA
var
  NOME : conjunto[1..10] de caractere
  I, J, COMEÇO, FINAL, MEIO : inteiro
  PESQ, RESP, X : caractere
  ACHA : lógico
início
  para I de 1 até 10 passo 1 faça
    leia NOME[I]
  fim_para

  (Ordenação)

  para I de 1 até 9 passo 1 faça
    para J de I + 1 até 10 passo 1 faça
      se (NOME[I] > NOME[J]) então
        X ← NOME[I]
        NOME[I] ← NOME[J]
        NOME[J] ← X
      fim_se
    fim_para
  fim_para
  (Trecho de pesquisa)

  RESP = "SIM"
  enquanto (RESP = "SIM") faça
    escreva "Entre o nome a ser pesquisado: "
    leia PESQ
    COMEÇO ← 1
    FINAL ← 10
    ACHA ← .Falso.
    enquanto (COMEÇO <= FINAL) .e. (ACHA = .Falso.) faça
      MEIO ← (COMEÇO + FINAL) div 2
      se (PESQ = NOME[MEIO]) então
        ACHA ← .Verdadeiro.
```

```

senão
    se (PESQ < NOME[MEIO]) então
        FINAL ← MEIO - 1
    senão
        COMEÇO ← MEIO + 1
    fim_se
fim_se
fim_enquanto
se (ACHA = .Verdadeiro.) então
    escreva PESQ, " foi localizado na posição ", MEIO
senão
    escreva PESQ, " não foi localizado"
fim_se
escreva "Deseja continuar? "
leia RESP
(Fim do trecho de pesquisa)
fim_enquanto
fim

```

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

```

1 - leia PESQ
2 - COMEÇO ← 1
3 - FINAL ← 10
4 - ACHA ← .Falso.
5 - enquanto (COMEÇO <= FINAL) .e. (ACHA = .Falso.) faça
6 -     MEIO ← (COMEÇO + FINAL) div 2
7 -     se (PESQ = NOME[MEIO]) então
8 -         ACHA ← .Verdadeiro.
9 -     senão
10 -         se (PESQ < NOME[MEIO]) então
11 -             FINAL ← MEIO - 1
12 -         senão
13 -             COMEÇO ← MEIO + 1
14 -         fim_se
15 -     fim_se
16 - fim_enquanto
17 - se (ACHA = .Verdadeiro.) então
18 -     escreva PESQ, "foi localizado na posição", MEIO
19 - senão
20 -     escreva PESQ, "não foi localizado"
21 - fim_se

```

Na linha 1, é solicitado que seja informado o dado a ser pesquisado. As linhas 2 e 3 inicializam as variáveis de controle COMEÇO 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 COMEÇO 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.

Neste ponto, a tabela está dividida em duas partes. A instrução da linha 7 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 COMEÇO 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 COMEÇO ser implicada pelo valor da variável MEIO somado com 1 (linha 13), ficando a variável COMEÇO 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 COMEÇO 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 - Exercício de Fixação

- 1) Desenvolva os algoritmos, diagrama de blocos e codificação em português estruturado dos seguintes exercícios:
 - a) Ler 12 elementos de uma matriz tipo vetor, colocá-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 por 5. Montar uma rotina de pesquisa binária, para pesquisar os elementos armazenados na matriz B.
- c) 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 os elementos da matriz B ordenados de forma crescente.
- d) Ler uma matriz A com 12 elementos. Após sua leitura, colocar os seus elementos em ordem crescente. Depois ler uma matriz B também com 12 elementos, colocar os elementos de B em ordem crescente. Construir uma matriz C, em que cada elemento de C é a soma do elemento correspondente de A com B. Colocar em ordem decrescente a matriz C e apresentar os seus valores.
- e) Ler duas matrizes do tipo vetor 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 os elementos da matriz C em ordem decrescente.
- f) Ler 30 elementos de uma matriz A do tipo vetor. 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.
- g) Ler 20 elementos de uma matriz A 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.
- h) Ler uma matriz A do tipo vetor com 20 elementos negativos. Construir uma matriz B de mesmo tipo e dimensão, segundo 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.
- i) Ler uma matriz A tipo vetor com 15 elementos. Construir uma matriz B de mesmo tipo, sendo que cada elemento de B deverá ser a metade de cada elemento de A. Apresentar os elementos das matrizes A em ordem decrescente e os elementos da matriz B em ordem crescente.

- j) Ler duas matrizes A e B do tipo vetor com 15 elementos cada. Construir duas outras matrizes C e D de mesmo tipo, sendo que cada elemento da matriz C deverá ser o somatório do elemento correspondente da matriz A, e cada elemento da matriz D deverá ser o fatorial do elemento correspondente da matriz B. Em seguida construir uma matriz E, que deverá conter a diferença dos elementos das matrizes C e D com a soma dos elementos das matrizes A e B. Apresentar os elementos da matriz E em ordem crescente.
- k) Ler duas matrizes A e B de uma dimensão do tipo vetor com dez elementos inteiros cada. Construir uma matriz C de mesmo tipo e dimensão que seja formada pelo soma dos quadrados de cada elemento correspondente das matrizes A e B. Apresentar a matriz C em ordem decrescente.
- l) Ler três matrizes A, B e C de uma dimensão do tipo vetor com 15 elementos reais cada. Construir uma matriz D de mesmo tipo e dimensão que seja formada pelo cubo da soma dos elementos correspondente às matrizes A, B e C. Apresentar a matriz D em ordem crescente.
- m) Ler duas matrizes A e B de uma dimensão do tipo vetor com 12 elementos reais cada. Construir uma matriz C de mesmo tipo e dimensão que seja formada pelo produto de cada elemento correspondente às matrizes A e B. Montar uma rotina de pesquisa seqüencial, para pesquisar os elementos existentes na matriz C.
- n) 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.

CAPÍTULO 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 utilizada 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 sua 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 considerar 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 (8 x 5). Isto significa que podem ser armazenados em TABELA até 40 elementos. A figura 8.1 apresenta a matriz TABELA com a indicação dos endereços (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				

Figura 8.1 - Exemplo da matriz TABELA com suas posições.

8.2.1 - Atribuição de uma Matriz

Uma matriz de duas dimensões é atribuída pela instrução **conjunto** já utilizada para definir o uso de uma matriz de uma dimensão, sendo bastante parecidas em sua referência. A sintaxe é: VARIÁVEL : **conjunto**[<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 caracteres.

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.

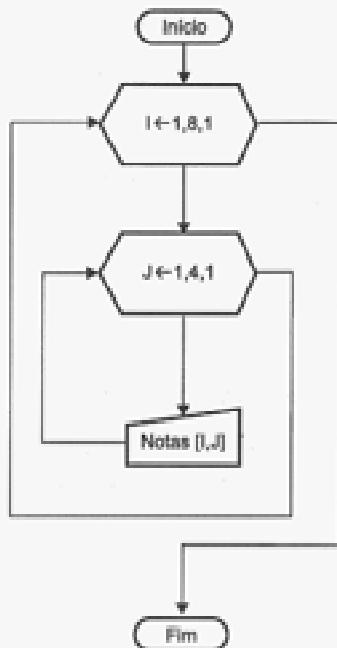


Figura 8.2 - Diagrama de blocos para leitura dos elementos de uma matriz tipo tabela.

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

```
programa LER_ELEMENTOS
var
    NOTAS : conjunto[1..8,1..4] de real
    I, J : inteiro
início
    para I de 1 até 8 passo 1 faça
        para J de 1 até 4 passo 1 faça
            leia NOTAS[I,J]
        fim_para
    fim_para
fim
```

8.2.3 - Escrita dos 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. A seguir, são apresentados o diagrama de blocos a a codificação em português estruturado da escrita das 4 notas dos 8 alunos.

Diagrama de Blocos

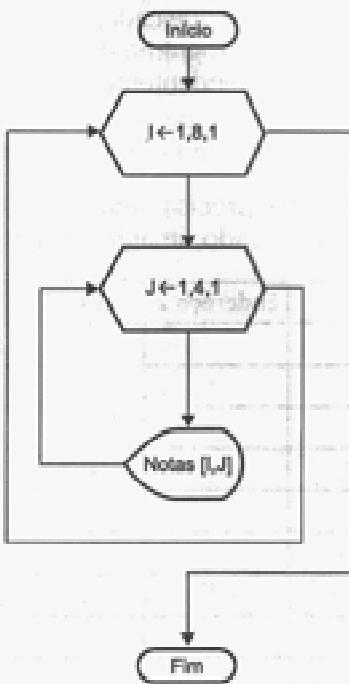


Figura 8.3 - Diagrama de blocos para escrita dos elementos de uma matriz tipo tabela.

Português Estruturado

```
programa ESCREVER_ELEMENTOS
var
  NOTAS : conjunto[1..8,1..4] de real
  I, J : inteiro
início
  para I de 1 até 8 passo 1 faça
    para J de 1 até 4 passo 1 faça
      escreva NOTAS[I,J]
    fim_para
  fim_para
fim
```

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, bairro e telefone de 10 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 aplicá-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.

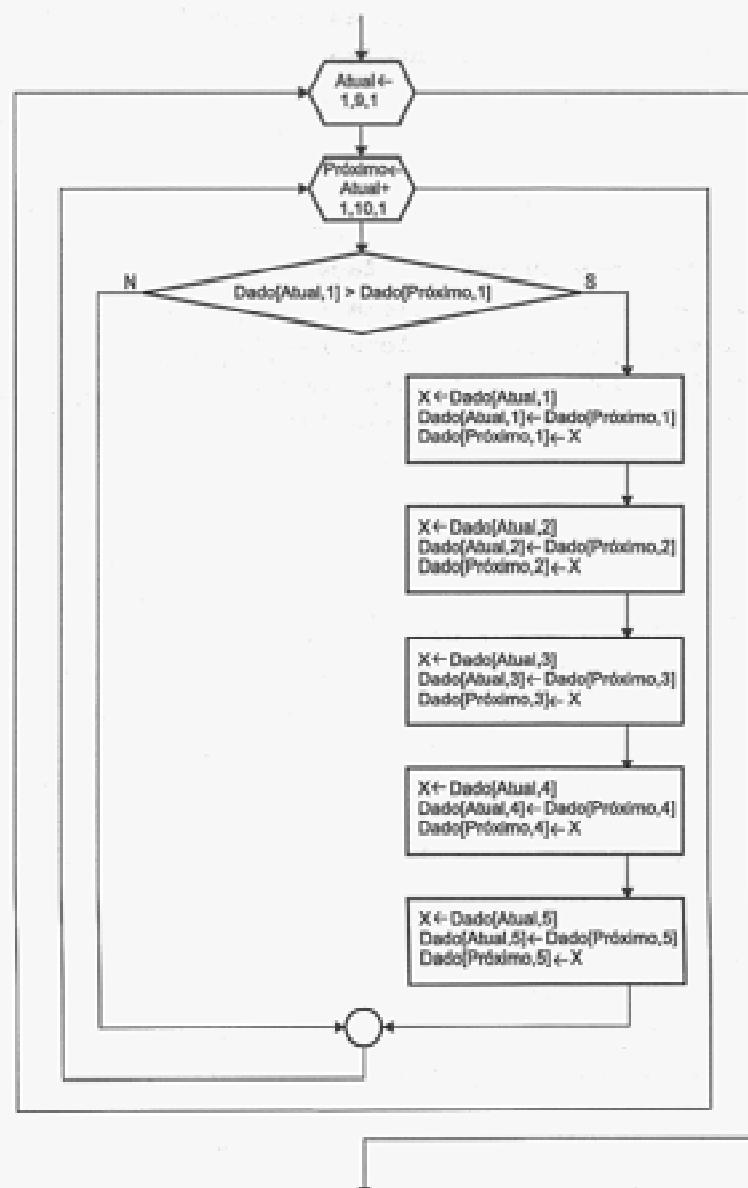


Figura 8.4 - Diagrama de blocos para o trecho de ordenação do programa de agenda.

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

Português Estruturado

programa AGENDA

var

DADO : conjunto[1..10,1..5] de caractere

I, J, ATUAL, PRÓXIMO : inteiro

X : caractere

inicio

{Rotina de entrada}

para I de 1 até 10 passo 1 faça

escreva "Nome: " leia DADO[I,1]

escreva "Endereço ..: " leia DADO[I,2]

escreva "CEP: " leia DADO[I,3]

escreva "Bairro: " leia DADO[I,4]

escreva "Telefone ..: " leia DADO[I,5]

fim_para

{Rotina de ordenação e troca de todos os elementos}

para ATUAL de 1 até 9 passo 1 faça

para PRÓXIMO de ATUAL + 1 até 10 passo 1 faça

se (DADO[ATUAL,1] > DADO[PRÓXIMO,1]) então

(Troca Nome)

X ← DADO[ATUAL,1]

DADO[ATUAL,1] ← DADO[PRÓXIMO,1]

DADO[PRÓXIMO,1] ← X

(Troca Endereço)

X ← DADO[ATUAL,2]

DADO[ATUAL,2] ← DADO[PRÓXIMO,2]

DADO[PRÓXIMO,2] ← X

(Troca CEP)

X ← DADO[ATUAL,3]

DADO[ATUAL,3] ← DADO[PRÓXIMO,3]

DADO[PRÓXIMO,3] ← X

(Troca Bairro)

$X \leftarrow DADO[ATUAL, 4]$

$DADO[ATUAL, 4] \leftarrow DADO[PRÓXIMO, 4]$

$DADO[PRÓXIMO, 4] \leftarrow X$

(Troca Telefone)

$X \leftarrow DADO[ATUAL, 5]$

$DADO[ATUAL, 5] \leftarrow DADO[PRÓXIMO, 5]$

$DADO[PRÓXIMO, 5] \leftarrow X$

fim_se

fim_para

fim_para

(Rotina de saída)

para I de 1 até 10 passo 1 faça

 para J de 1 até 5 passo 1 faça

 escreva DADO[I, J]

 fim_para

fim_para

fim

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 $(DADO[ATUAL, 1] > DADO[PRÓXIMO, 1])$ então.

Diagrama de Blocos

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

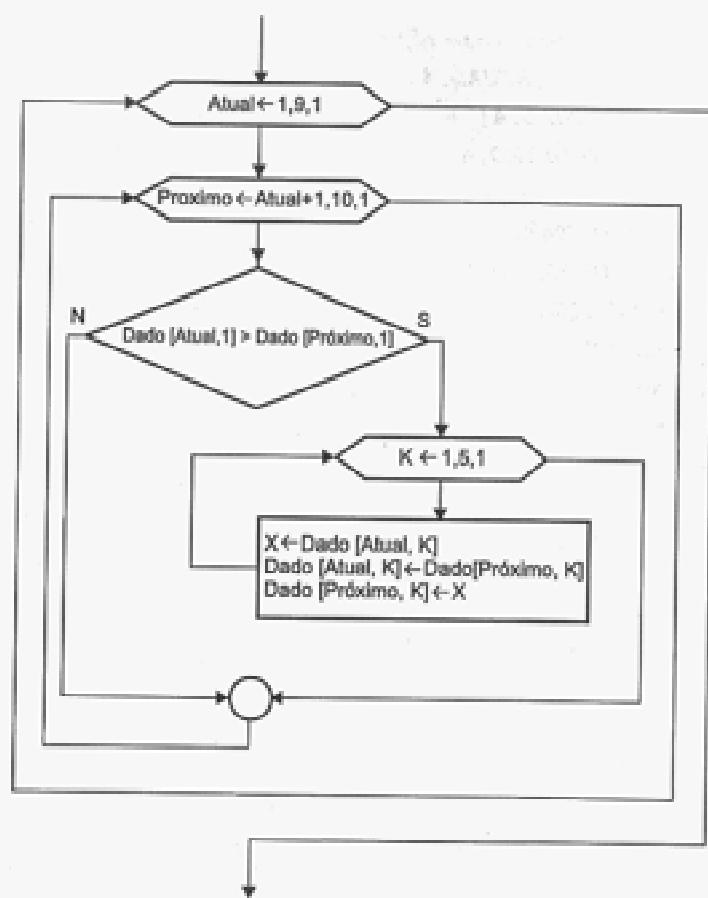


Figura 8.5 - Diagrama de blocos para o trecho de ordenação simplificado do programa de agenda.

Português Estruturado

```

programa AGENDA
var
  DADO : conjunto[1..10,1..5] de caracteres
  I, J, K, ATUAL, PRÓXIMO : inteiro
  X : caractere
início
  (Rotina de entrada)

  para I de 1 até 10 passo 1 faça
    escreva "Nome .....: " leia DADO[I,1]
    escreva "Endereço ..: " leia DADO[I,2]

```

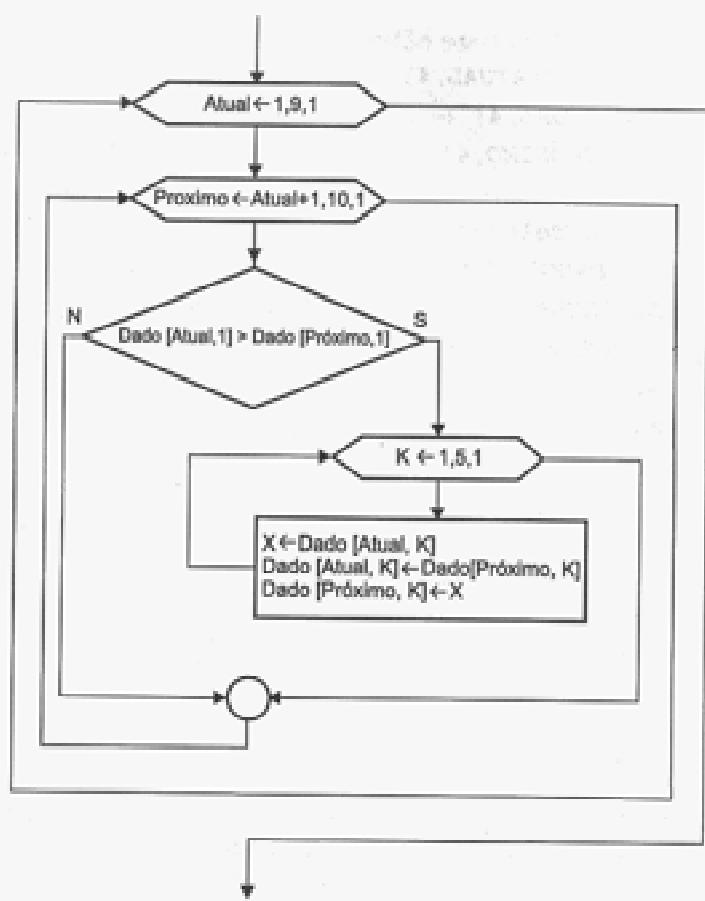


Figura 8.5 - Diagrama de blocos para o trecho de ordenação simplificado do programa de agenda.

Português Estruturado

```

programa AGENDA
var
  DADO : conjunto[1..10,1..5] de caracteres
  I, J, K, ATUAL, PRÓXIMO : inteiro
  X : caractere
inicio
  (Rotina de entrada)

  para I de 1 até 10 passe 1 faça
    escreva "Nome .....: " leia DADO[I,1]
    escreva "Endereço ..: " leia DADO[I,2]

```

```
escreva "CEP .....: " leia DADO[I,3]
escreva "Bairro ....: " leia DADO[I,4]
escreva "Telefone ..: " leia DADO[I,5]
fim_para
```

(Rotina de ordenação e troca de todos os elementos)
(com looping para administrar a troca de elementos)

```
para ATUAL de 1 até 9 passo 1 faça
    para PRÓXIMO de ATUAL + 1 até 10 passo 1 faça
        se (DADO[ATUAL,1] > DADO[PRÓXIMO,1]) então
            para K de 1 até 5 passo 1 faça
                X ← DADO[ATUAL,K]
                DADO[ATUAL,K] ← DADO[PRÓXIMO,K]
                DADO[PRÓXIMO,K] ← X
            fim_para
        fim_se
    fim_para
fim_para
```

(Rotina de saída)

```
para I de 1 até 10 passo 1 faça
    para J de 1 até 5 passo 1 faça
        escreva DADO[I,J]
    fim_para
fim_para
fim
```

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 médias 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. Considere para tanto as duas tabelas seguintes:

	Nome
1	
2	
3	
4	
5	
6	
7	
8	

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

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 / de 1 até 8 passo 1 faça** 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 MÉDIA 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).

Diagrama de Blocos

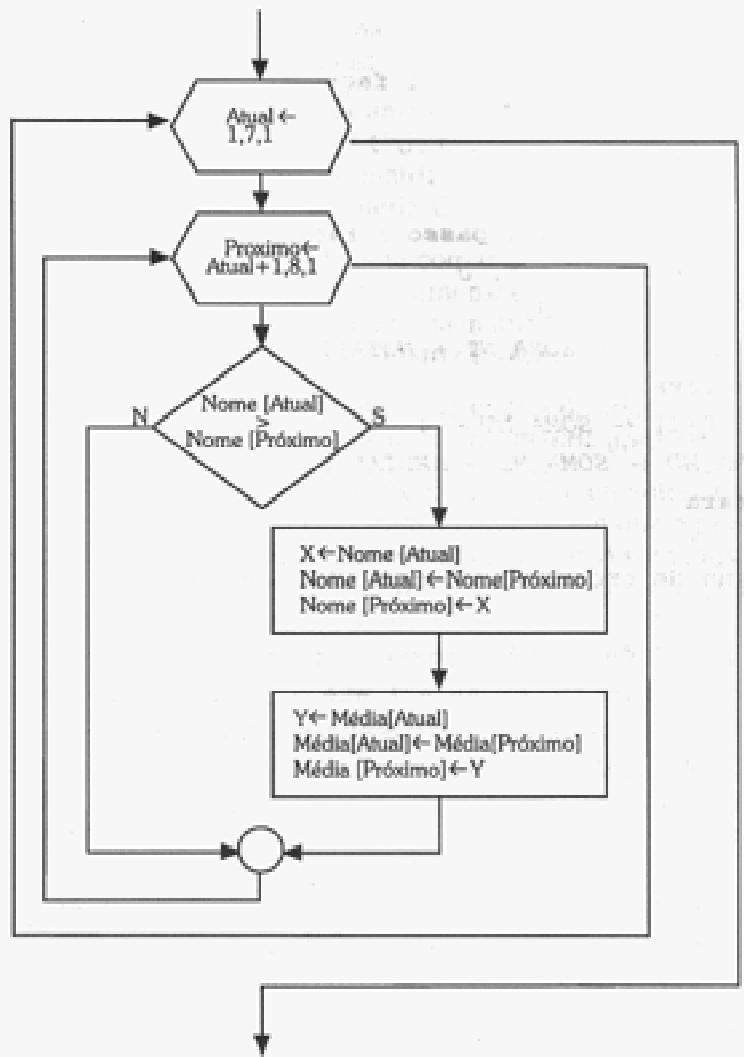


Figura 8.6 - Diagrama de blocos para o trecho de ordenação do programa de notas.

Português Estruturado

```

programa CALC_MÉDIA
var
  X : caractere
  I, J, ATUAL, PRÓXIMO : inteiro
  Y, SOMA_NT, SOMA_MD, MÉDIA_GP : real
  NOTA : conjunto[1..8,1..4] de real
  MÉDIA : conjunto[1..8] de real
  
```

```

NOMES : conjunto[1..8] de caractere
início
    SOMA_MD ← 0
    para I de 1 até 8 passo 1 faça
        SOMA_NT ← 0
        escreva "Aluno ", I
        leia NOMES[I]
        para J de 1 até 4 passo 1 faça
            escreva "Nota ", J
            leia NOTA[I,J]
            SOMA_NT ← SOMA_NT + NOTA[I,J]
        fim_para
        MÉDIA[I] ← SOMA_NT / 4
        SOMA_MD ← SOMA_MD + MÉDIA[I]
    fim_para

```

(Rotina de ordenação e troca de elementos)

```

para ATUAL de 1 até 7 passo 1 faça
    para PRÓXIMO ← ATUAL + 1 até 8 passo 1 faça
        se (NOMES[ATUAL] > NOMES[PRÓXIMO]) então
            X ← NOMES[ATUAL]
            NOMES[ATUAL] ← NOMES[PRÓXIMO]
            NOMES[PRÓXIMO] ← X
            Y ← MÉDIA[ATUAL]
            MÉDIA[ATUAL] ← MÉDIA[PRÓXIMO]
            MÉDIA[PRÓXIMO] ← Y
        fim_se
    fim_para
fim_para
MÉDIA_GP ← SOMA_MD / 8
para I de 1 até 8 passo 1 faça
    escreva "Aluno .: ", NOMES[I]
    escreva "Media .: ", MÉDIA[I]
fim_para
escreva "Media Geral : ", MÉDIA_GP
fim

```

8.4 - Exercício de Fixação

- 1) Desenvolva os algoritmos, diagrama de blocos e codificação em português estruturado dos seguintes exercícios:
- 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 seja formada pela soma dos elementos da matriz A com os elementos da matriz B. Apresentar os elementos da 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.
 - 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.
 - Ler uma matriz A de duas dimensões com 5 linhas e 4 colunas. Construir uma matriz B de mesma dimensão, onde cada elemento da matriz B deverá ser o fatorial de cada elemento correspondente da matriz A. Apresentar ao final as matrizes A e B.
 - Ler uma matriz A de duas dimensões com 4 linhas e 5 colunas, armazenando nessa matriz os valores das temperaturas em graus Celsius. Construir uma matriz B de mesma dimensão, sendo que cada elemento da matriz B deverá ser o valor da temperatura em graus Fahrenheit de cada elemento correspondente da matriz A. Apresentar ao final as matriz A e B.
 - Ler uma matriz A de duas dimensões com 5 linhas e 5 colunas. Construir uma matriz B de mesma dimensão, sendo que cada elemento da matriz B deverá ser o dobro de cada elemento correspondente da matriz A, com exceção para os valores situados na diagonal principal (posições B[1,1], B[2,2], B[3,3], B[4,4] e B[5,5]) os quais deverão ser o triplo de cada elemento correspondente da matriz A. Apresentar ao final a matriz B.

- i) 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 exceção para os valores situados nos índices ímpares da diagonal principal ($B[1,1]$, $B[3,3]$, $B[5,5]$ e $B[7,7]$), os quais deverão ser o fatorial de cada elemento correspondente da matriz A. Apresentar ao final a matriz B.
- j) Ler uma matriz A de duas dimensões com 6 linhas e 5 colunas. Construir uma matriz B de mesma dimensão, que deverá ser formada do seguinte modo: para cada elemento par da matriz A deverá ser somado 5 e para cada elemento ímpar da matriz A deverá ser subtraído 4. Apresentar ao final as matrizes A e B.
- k) Ler uma matriz A de duas dimensões com 4 linhas e 4 colunas. Apresentar o somatório dos elementos situados na diagonal principal (posições $A[1,1]$, $A[2,2]$, $A[3,3]$, $A[4,4]$) da referida matriz.
- l) Ler uma matriz A de duas dimensões com 15 linhas e 15 colunas. Apresentar o somatório dos elementos pares situados na diagonal principal da referida matriz.
- m) Ler uma matriz A de duas dimensões com 5 linhas e 5 colunas. Apresentar o somatório dos elementos situados nas posições de linha e coluna ímpares da diagonal principal ($A[1,1]$, $A[3,3]$, $A[5,5]$) da referida matriz.
- n) Ler uma matriz A de duas dimensões com 7 linhas e 7 colunas. Ao final apresentar o total de elementos pares existentes dentro da matriz.
- o) 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.
- p) 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.
- q) 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.

- r) Ler quatro matrizes A, B, C e D de uma dimensão com 4 elementos. Construir uma matriz E de duas dimensões do tipo 4 x 4, sendo que a primeira linha da matriz E deverá ser formada pelo dobro dos valores dos elementos da matriz A, a segunda linha da matriz E deverá ser formada pelo triplo dos valores dos elementos da matriz B, a terceira linha da matriz E deverá ser formada pelo quadruplo dos valores dos elementos da matriz C e a quarta linha da matriz E deverá ser formada pela fatorial dos valores dos elementos da matriz D. Apresentar a matriz E.
- s) Ler duas matrizes A e B, cada uma de duas dimensões com 5 linhas e 6 colunas. A matriz A deverá apenas aceitar a entrada de valores pares, enquanto que a matriz B deverá aceitar apenas a entrada de valores ímpares. As entradas dos valores nas matrizes A e B deverão ser validadas pelo programa e não pelo usuário. Construir uma matriz C de mesma dimensão, que seja formada pela soma dos elementos da matriz A com os elementos da matriz B. Apresentar os elementos da matriz C.
- t) Ler duas matrizes A e B de duas dimensões com 4 linhas e 5 colunas. A matriz A deverá ser formada por valores que sejam divisíveis por 3 e 4, enquanto a matriz B deverá ser formada por valores que sejam divisíveis por 5 ou 6. As entradas dos valores nas matrizes deverão ser validadas pelo programa e não pelo usuário. Construir e apresentar uma matriz C de mesma dimensão e número de elementos que conteria a subtração dos elementos da matriz A em relação aos elementos da matriz B.
- u) Ler duas matrizes A e B de duas dimensões com 4 linhas e 5 colunas. A matriz A deverá ser formada por valores que sejam divisíveis por 3 ou 4, enquanto a matriz B deverá ser formada por valores que sejam divisíveis por 5 e 6. As entradas dos valores nas matrizes deverão ser validadas pelo programa e não pelo usuário. Construir e apresentar uma matriz C de mesma dimensão e número de elementos que contenha o valor da multiplicação dos elementos da matriz A com os elementos correspondentes da matriz B.
- v) 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 sejam 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 e não pelo usuário. Construir e apresentar uma matriz C de mesma dimensão e número de elementos que contenha a soma dos elementos das matrizes A e B.

Cadastro de Notas Escolares	
Nome	<input type="text"/>
Primeira Nota	<input type="text"/>
Segunda Nota	<input type="text"/>
Terceira Nota	<input type="text"/>
Quarta Nota	<input type="text"/>

Figura 9.1 - Exemplo do layout de um registro com seus campos.

Sendo assim, o registro está 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 e 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...fim_registro**, conforme sintaxe indicada abaixo.

Português Estruturado

```

tipo
    <identificador> = registro
        <lista dos campos e seus tipos>
    fim_registro
var
    <variáveis> : <identificador>

```

Em que *identificador* é o nome do tipo registro em caracteres maiúsculos, em itálico, como as variáveis, 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 caractere.

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 está sendo 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 : caractere
    NOTA1 : real
    NOTA2 : real
    NOTA3 : real
    NOTA4 : real
  fim_registro

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.

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

9.1.2 - Leitura 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 diagrama de blocos e o código em português estruturado:

Diagrama de Blocos



Figura 9.2 - Exemplo de leitura de um registro.

Português Estruturado

```
programa LEITURA
  tipo
    CAD_ALUNO = registro
      NOME : caractere
      NOTA1 : real
      NOTA2 : real
      NOTA3 : real
      NOTA4 : real
    fim_registro
  var
    ALUNO : cad_aluno
  inicio
    leia ALUNO.NOME
    leia ALUNO.NOTAA1
    leia ALUNO.NOTAA2
    leia ALUNO.NOTAA3
    leia ALUNO.NOTAA4
  fim
```

Uma leitura de registros também pode ser feita como **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 um campo em específico.

9.1.3 - Escrita de Registros

O processo de escrita de um registro é feito com a instrução **escreva** de forma semelhante ao processo de leitura. Assim sendo, observe o diagrama de blocos e o código em português estruturado abaixo:

Diagrama de Blocos



Figura 9.3 - Exemplo de escrita de um registro.

Português Estruturado

```
programa ESCRITA
tipo
  CAD_ALUNO = registro
    NOME : caractere
    NOTA1 : real
    NOTA2 : real
```

```

NOTA3 : real
NOTA4 : real
fim_registro

var
ALUNO : cad_aluno
inicio
escreva ALUNO.NOME
escreva ALUNO.NOTA1
escreva ALUNO.NOTA2
escreva ALUNO.NOTA3
escreva ALUNO.NOTA4
fim

```

A escrita de registros também pode ser feita como **escreva ALUNO**, desde que se deseje fazer uma escrita de forma genérica de todos os campos. Note que a estrutura de registro apresentada permite somente a leitura e escrita de um único conjunto de campos para um registro. Mais adiante, será apresentado como fazer para conseguir ler e escrever mais de um registro.

9.2 - Estrutura de um Registro de Conjuntos

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 9.4 mostra um layout desta nova proposta.

Layout do formato de um registro com seus campos

Cadastro de Notas Escolares				
Nome	<input type="text"/>			
Notas	1	2	3	4
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Figura 9.4 - Exemplo do layout de um registro de conjunto com seus campos.

9.2.1 - Atribuição de Registros de Conjuntos

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 = conjunto[1..4] de real  
    CAD_ALUNO = registro  
        NOME : caractere  
        NOTA : bimestre  
    fim_registro
```

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 de Registro de Conjuntos

A leitura de um registro de conjunto é efetuada com a instrução leia geralmente dentro de um laço de repetição. Assim sendo, observe o diagrama de blocos e o código em português estruturado abaixo, que representam a leitura do nome e das quatro notas bimestrais do aluno.

Diagrama de Blocos

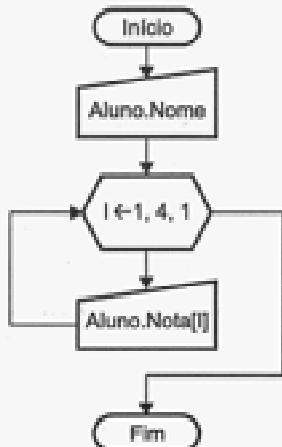


Figura 9.5 - Exemplo de leitura de um registro de conjunto.

Português Estruturado

```
programa LEITURA
tipo
    BIMESTRE = conjunto[1..4] de real
    CAD_ALUNO = registro
        NOME : caractere
        NOTA : bimestre
    fim_registro
var
    ALUNO : cad_aluno
    I : inteiro
inicio
    leia ALUNO.NOME
    para I de 1 até 4 passo 1 faça
        leia ALUNO.NOTA[I]
    fim_para
fim
```

9.2.3 - Escrita de Registro de Conjuntos

O processo de escrita de um registro de conjunto é feito com a instrução **escreva** de forma semelhante ao processo de leitura. Assim sendo, observe em seguida o diagrama de blocos e o código em português estruturado.

Diagrama de Blocos

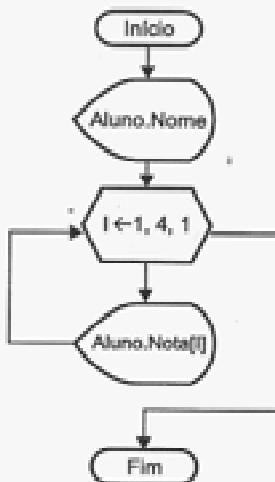


Figura 9.6 - Exemplo de escrita de um registro de conjunto.

```
programa ESCRITA
    tipo
        BIMESTRE = conjunto[1..4] de real
        CAD_ALUNO = registro
            NOME : caractere
            NOTA : bimestre
        fim_registro
    var
        ALUNO : cad_aluno
        I : inteiro
    inicio
        escreva ALUNO.NOME
        para I de 1 até 4 passo 1 faça
            escreva ALUNO.NOTA[I]
        fim_para
    fim
```

9.3 - Estrutura de um Conjunto 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 leitor terá contato com o conjunto 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 Conjunto de Registros

Para declarar um conjunto 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. Para exemplificar o que está sendo exposto, considere que você deve fazer um 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 os oito alunos:

```

tipo
    BIMESTRE = conjunto[1..4] de real
    CAD_ALUNO = registro
        NOME : caractere
        NOTA : bimestre
    fim_registro

var
    ALUNO : conjunto[1..8] de cad_aluno

```

Observe que após a instrução var, é indicada a variável de registro ALUNO, sendo esta um conjunto 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 é um conjunto de quatro valores reais.

9.3.2 - Leitura de Conjunto de Registros

A leitura será feita 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 diagrama de blocos e o código em português estruturado.

Diagrama de Blocos

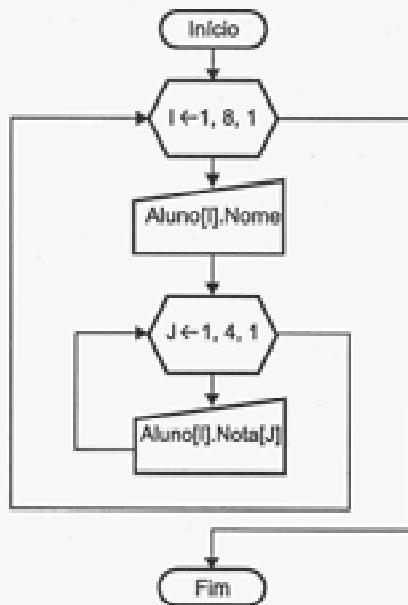


Figura 9.7 - Exemplo de leitura de um conjunto de registros.

Português Estruturado

```
programa LEITURA
tipo
    BIMESTRE = conjunto[1..4] de real
    CAD_ALUNO = registro
        NOME : caractere
        NOTA : bimestre
    fim_registro
var
    ALUNO : conjunto[1..8] de cad_aluno
    I, J : inteiro
inicio
    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]
        fim_para
    fim_para
fim
```

Veja que o looping da variável I controla o número de alunos da turma, no caso 8, e o looping da variável J controla o número de notas, até 4 por aluno. Para cada movimentação de mais um na variável I existem quatro movimentações na variável J.

9.3.3 - Escrita de Conjunto de Registros

O processo de escrita de um conjunto de registros é similar aos modos anteriores já estudados. Assim sendo, observe em seguida o diagrama de blocos e o código em português estruturado.

Diagrama de Blocos

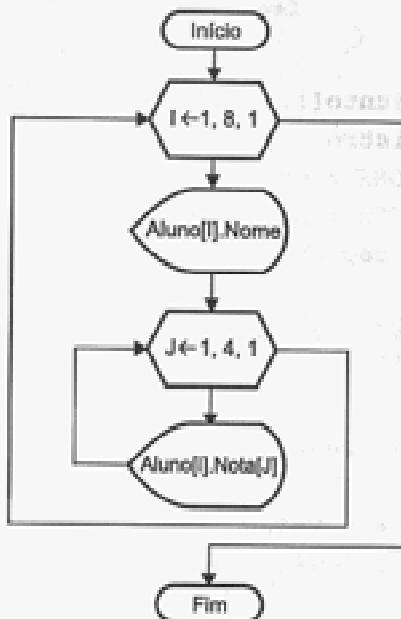


Figura 9.8 - Exemplo de escrita de um conjunto de registros.

Português Estruturado

```
programa ESCRITA
tipo
    BIMESTRE = conjunto[1..4] de real
    CAD_ALUNO = registro
        NOME : caractere
        NOTA : bimestre
    fim_registro

var
    ALUNO : conjunto[1..8] de cad_aluno
    I, J : inteiro
início
    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]
        fim_para
    fim_para
fim
```

9.4 - Exercício de Aprendizagem

Para demonstrar a utilização de programas com tabelas de dados heterogêneos, considere os seguintes problemas:

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 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 PRÓXIMO. 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.

Diagrama de Blocos

No diagrama seguinte, é apresentada a comparação efetuada, sendo $\text{ALUNO[ATUAL].NOME} > \text{ALUNO[PRÓXIMO].NOME}$. Observe que está sendo questionado se o nome do aluno atual é maior que o nome do próximo aluno. A condição sendo verdadeira é efetuada a troca não só do nome, mas de todos os elementos que estão armazenados na tabela. Isto é possível, uma vez que a variável X é do mesmo tipo que a tabela ALUNO.

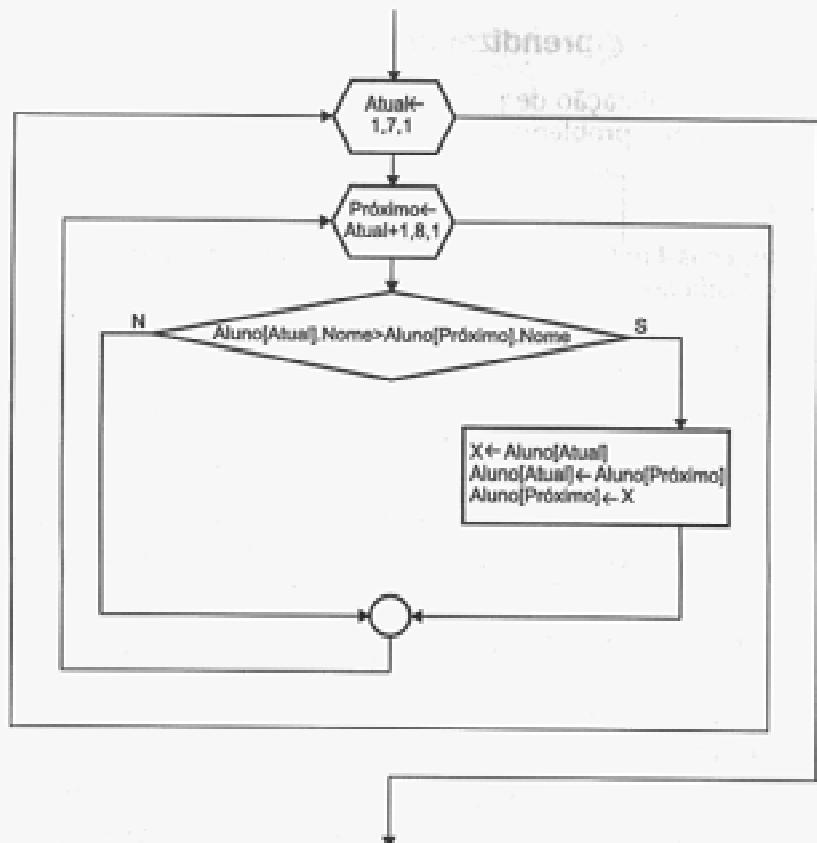


Figura 9.9 - Exemplo da rotina de ordenação com dados heterogêneos.

Português Estruturado

```

programa LEITURA_ORDERAÇÃO_ESCRITA
tipo
    BIMESTRE = conjunto[1..4] de real
    CAD_ALUNO = registro
        NOME : caractere
        NOTA : bimestre
    fim_registro
var
    ALUNO : conjunto[1..8] de cad_aluno
    I, J, ATUAL, PRÓXIMO : inteiro
    X : cad_aluno
início
    para J de 1 até 8 passo 1 faça

```

```

leia ALUNO[J].NOME
para I de 1 até 4 passo 1 faça
    leia ALUNO[J].NOTA[I]
fim_para
fim_para

(Rotina de ordenação de dados heterogêneos)

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
        fim_se
    fim_para
fim_para

para J de 1 até 8 passo 1 faça
    escreva ALUNO[J].NOME
    para I de 1 até 4 passo 1 faça
        escreva ALUNO[J].NOTA[I]
    fim_para
fim_para
fim

```

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 válido, o programa deverá apresentar o cargo e o respectivo salário. Caso seja 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	Continuo	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 caractere e o número de salários 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; sem sim, repetir os passos 3, 4 e 5; se não, encerrar o programa.

Diagrama de Blocos

Observe que o diagrama concentra-se na lógica no que tange à pesquisa, pois os dados da tabela anterior deverão ser fornecidos pelo operador.

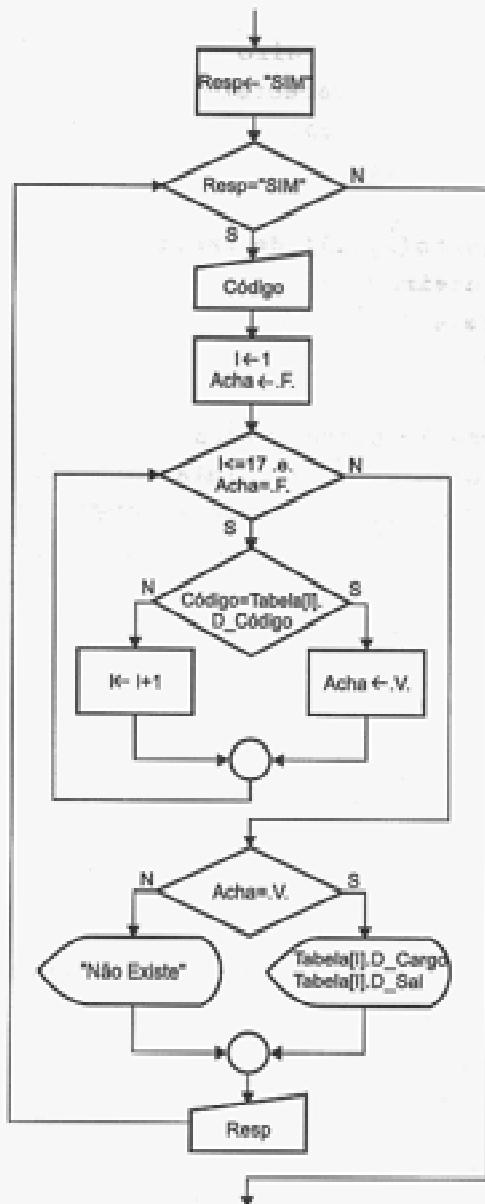


Figura 9.10 - Exemplo da rotina de pesquisa sequencial em uma tabela.

Português Estruturado

```
programa TABELA_DE_SALÁRIOS
    tipo
        DADOS = registro
            D_CÓDIGO : inteiro
            D_CARGO : caractere
            D_SAL : real
        fim_registro
    var
        TABELA : conjunto[1..17] de dados
        I, CÓDIGO : inteiro
        RESP : caractere
        ACHA : lógico
    inicio
        para I de 1 até 17 passo 1 faça
            escreva "Código ..... : " leia TABELA[I].D_CÓDIGO
            escreva "Cargo ..... : " leia TABELA[I].D_CARGO
            escreva "Salário ..... : " leia TABELA[I].D_SAL
        fim_para
        (Trecho de pesquisa seqüencial)

        RESP ← "SIM"
        enquanto (RESP = "SIM") faça
            escreva "Qual código - 1 a 17"
            leia CÓDIGO
            I ← 1
            ACHA ← .Falso.
            enquanto (I <= 17) ..e.. (ACHA = .Falso.) faça
                se (CÓDIGO = TABELA[I].D_CÓDIGO) então
                    ACHA ← .Verdadeiro.
                senão
                    I ← I + 1
                fim_se
            fim_enquanto
            se (ACHA = .Verdadeiro.) então
                escreva "Cargo: ", TABELA[I].D_CARGO
                escreva "Salário: ", TABELA[I].D_SAL
            senão
```

```
    escreva "Cargo Inexistente"  
fim_se  
    escreva "Deseja continuar pesquisa?"  
leia RESP  
  
    fim_enquanto  
fim
```

9.5 - Exercício de Fixação

- 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:
 - 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:
 - a) Cadastrar os 20 registros (após o cadastro efetuar a classificação por nome).
 - b) Pesquisar os 20 registros, de cada vez, pelo campo nome (usar o método binário, 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:
- Cadastrar os 15 registros.
 - Apresentar os registros (nome e altura) das pessoas menores ou iguais a 1.5m.
 - Apresentar os registros (nome e altura) das pessoas que sejam maiores que 1.5m.
 - Apresentar os registros (nome e altura) das pessoas que sejam maiores que 1.5m e menores que 2.0m.
 - Apresentar todos os registros com a média extraída de todas as alturas armazenadas.
 - Sair do programa de cadastro.
- 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 seguinte etapas:
- Cadastrar os 20 empregados.
 - Classificar os registros por número de matrícula.
 - Pesquisar um determinado empregado pelo número de matrícula (método binário).
 - Apresentar de forma ordenada os registros dos empregados que recebem salários acima de \$1.000.
 - Apresentar de forma ordenada os registros dos empregados que recebem salários abaixo de \$1.000.
 - Apresentar de forma ordenada os registros dos empregados que recebem salários iguais a \$1.000.
 - Sair do programa de cadastro.

PARTE N

**Programação
Estruturada ou
Modular**

CAPÍTULO 10

Utilização de Sub-rotinas

A partir deste capítulo será estudada a aplicação de sub-rotinas em algoritmos, também conhecidas pela denominação módulos subprogramas, ou subalgoritmos. Na realidade, não importa como são chamadas, o que importa é a forma como funcionam e como devem ser aplicadas em um programa, e é isto que o leitor 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 - O 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 figura 10.1 apresenta um exemplo desta estrutura.

A utilização do método "de cima para baixo" permite que seja efetuado cada módulo de 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.

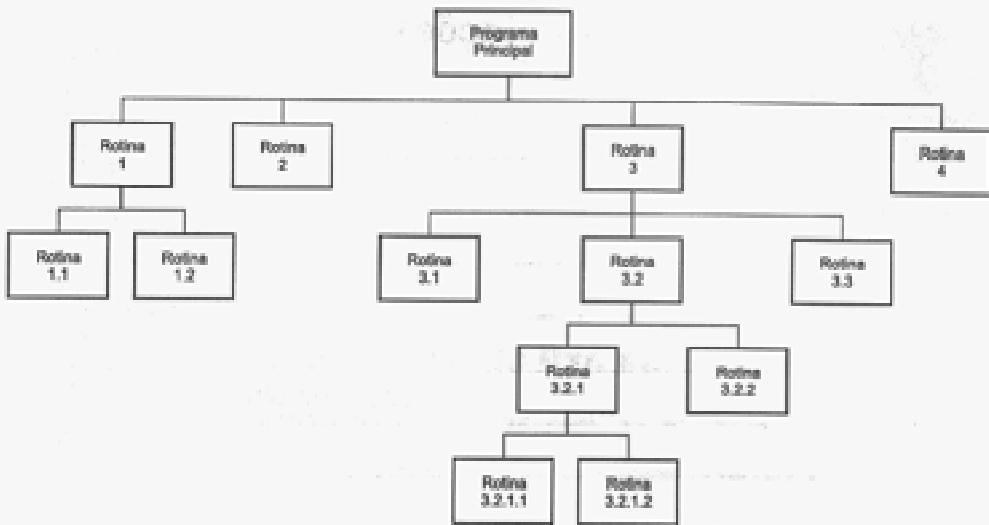


Figura 10.1 - Organização hierárquica de programa com o método Top-Down.

A utilização de sub-rotinas e o uso do método Top-Down na programação permitem 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. Se o leitor comparar esta proposta com o que foi estudado anteriormente, verá suas vantagens. Lembre-se dos programas anteriores todos os seus algoritmos de saída obrigavam de certa forma efetuar primeiro a entrada dos dados.

Anotações

CAPÍTULO 11

Aplicação Prática do Uso de Sub-Rotinas - Procedimentos

Será estudado neste capítulo um tipo de sub-rotina conhecido como *Procedimento*. Existe um outro tipo conhecido como *Função* que será estudado no capítulo 13. Entre estes dois tipos de sub-rotina 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: **Procedimento e Função**. O importante no uso prático destes dois tipos de sub-rotina é distinguir as diferenças entre elas e como utilizá-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.

Português Estruturado

```
procedimento <nome do procedimento>
  var
    <variáveis>
  inicio
    <instruções>
  fim
```

A melhor maneira de entender como trabalhar com uma sub-rotina é fazer a sua aplicação em um programa mais complexo. Para tanto, imagine o seguinte problema:

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 a 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. A rotina principal efetuará o controle das quatro rotinas secundárias que, por sua vez, pedirão a leitura de dois valores, farão a operação e apresentarão o resultado obtido. A figura 11.2 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.



Figura 11.1 - Hierarquia das rotinas do programa calculadora.

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
 - 5 - Fim de Programa

- 2 - Ao ser selecionado um valor, 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 mesmas variáveis, mas elas não serão executadas ao mesmo tempo para todas as operações. Serão utilizadas em separado e somente para a rotina escolhida.

Diagramas 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.

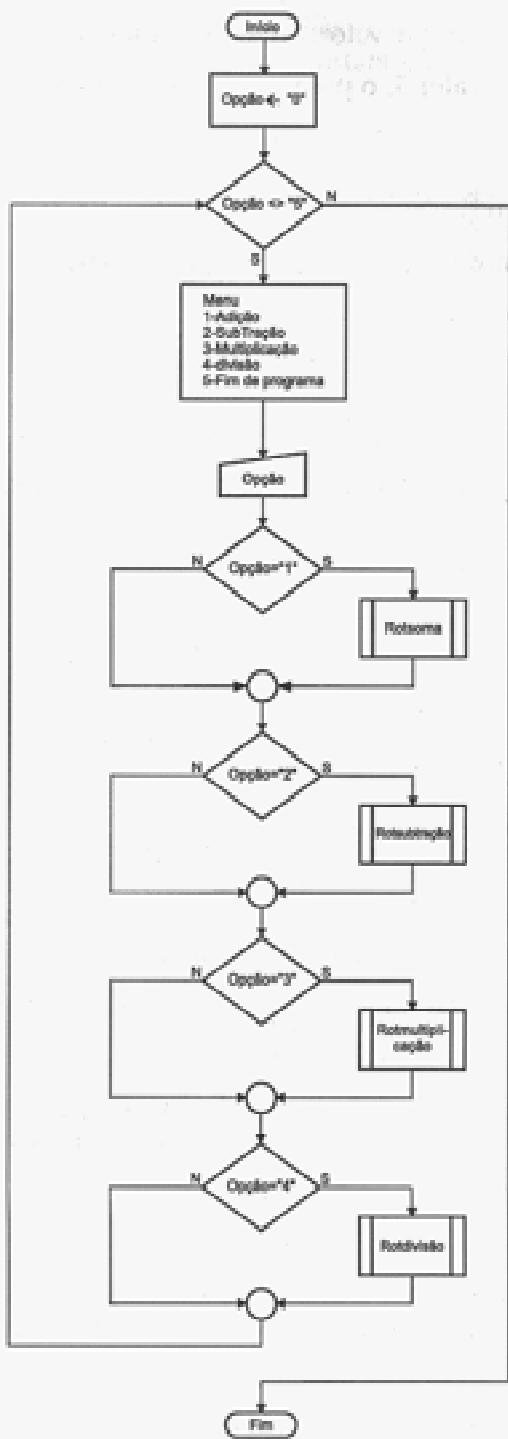
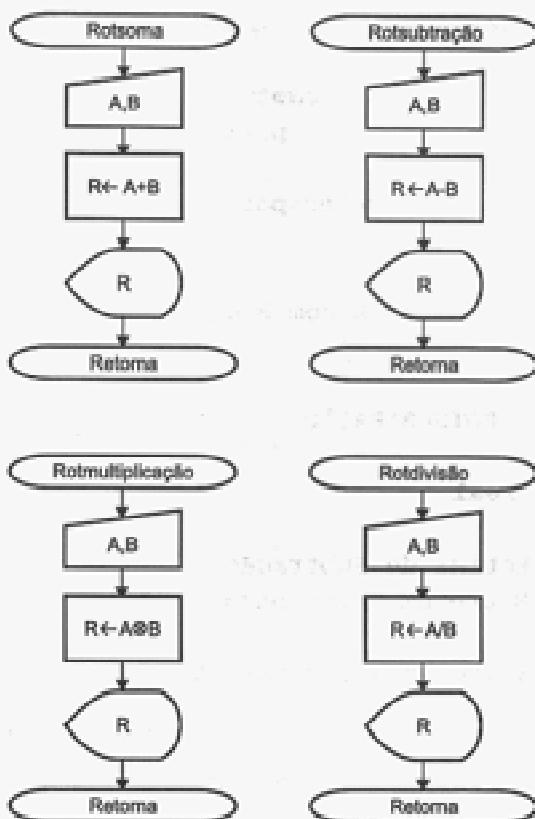


Figura 11.2 - Diagrama de blocos para o programa calculadora com sua sub-rotina.
(continua na próxima página)



*Figura 11.3 - Diagramas de blocos para o programa calculadora com sua sub-rotina.
(continuação da página anterior)*

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 **OPÇÃO** 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.

```

programa CALCULADORA
var
    OPÇÃO : caractere

    (Sub-rotinas de cálculos)
procedimento ROTSOMA
var

```

R, A, B : real

início

 escreva "Rotina de Adição"

 escreva "Entre um valor para A: "

 leia A

 escreva "Entre um valor para B: "

 leia B

R ← *A* + *B*

 escreva "A soma de A com B é = ", *R*

fim

procedimento ROTSUBTRAÇÃO

var

R, A, B : real

início

 escreva "Rotina de Subtração"

 escreva "Entre um valor para A: "

 leia A

 escreva "Entre um valor para B: "

 leia B

R ← *A* - *B*

 escreva "A subtração de A com B é = ", *R*

fim

procedimento ROTMULTIPLICAÇÃO

var

R, A, B : real

início

 escreva "Rotina de Multiplicação"

 escreva "Entre um valor para A: "

 leia A

 escreva "Entre um valor para B: "

 leia B

R ← *A* * *B*

 escreva "A multiplicação de A com B é = ", *R*

fim

procedimento ROTDIVISÃO

var

R, A, B : real

início

 escreva "Rotina de Divisão"

```

escreva "Entre um valor para A: "
leia A
escreva "Entre um valor para B: "
leia B
R ← A / B
escreva "A divisão de A com B é = ", R
fim

```

(Programa Principal)

```

inicio
OPÇÃO ← "0"
enquanto (OPÇÃO <> "5") faça
    escreva "1 - Adição"
    escreva "2 - Subtração"
    escreva "3 - Multiplicação"
    escreva "4 - Divisão"
    escreva "5 - Fim de Programa"
    escreva "Escolha uma opção: "
    leia OPÇÃO
    se (OPÇÃO = "1") então
        rotadicao
    fim_se
    se (OPÇÃO = "2") então
        rotsubtracao
    fim_se
    se (OPÇÃO = "3") então
        rotmultiplicacao
    fim_se
    se (OPÇÃO = "4") então
        rotdivisao
    fim_se
fim_enquanto
fim

```

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 sequência grande de instruções do tipo se, sejam estas uma após a outra ou mesmo encadeadas, poderá ser simplificada com a utilização da instrução **caso...fim_caso**, que possui a seguinte sintaxe:

```
caso <variável>
  seja <valor1> faça <operação1>
  seja <valor2> faça <operação2>
  seja <valorN> faça <operaçãoN>
  senão <operação>
fim_caso
```

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

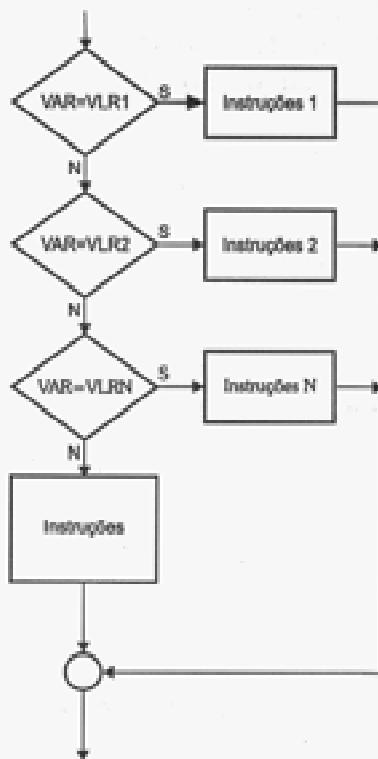


Figura 11.4 - Diagrama de Blocos da Instrução **caso...fim_caso**.

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

Diagrama de Blocos

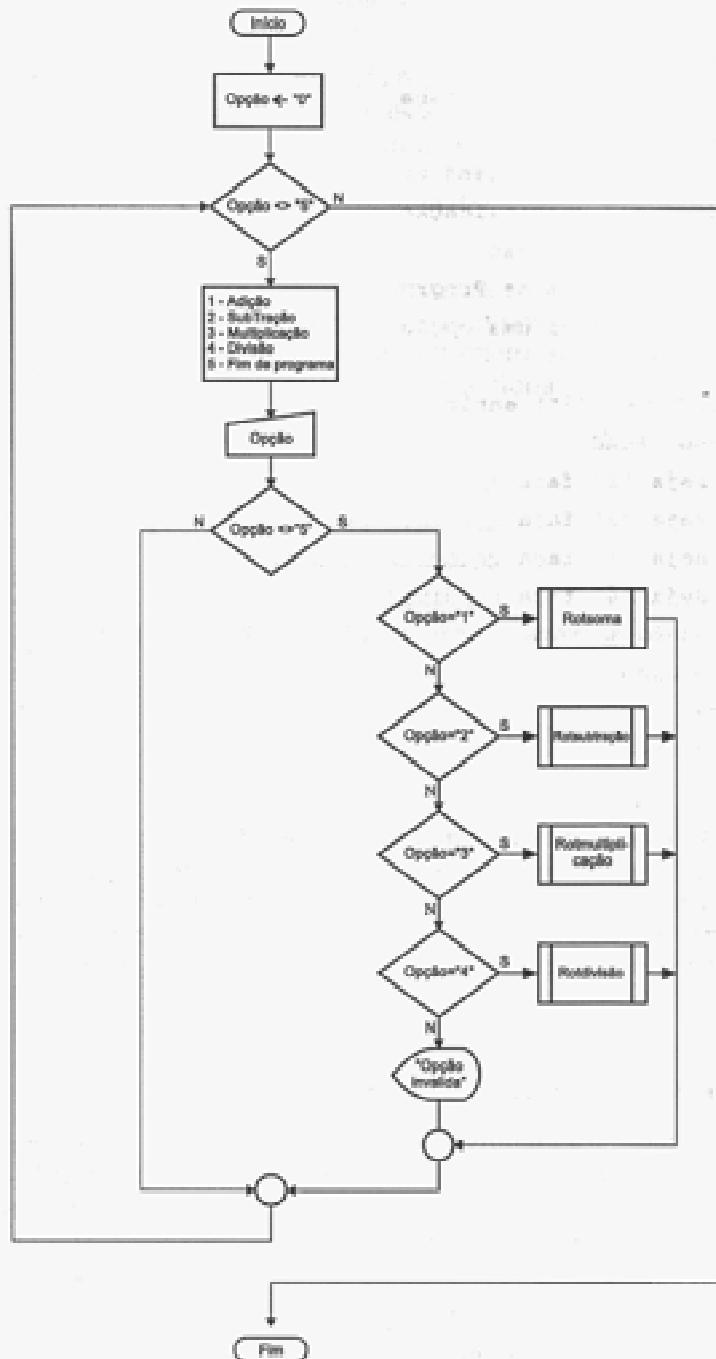


Figura 11.5 - Diagrama de Blocos do Programa Principal Usando a Instrução caso...fim_caso.

```
início
    OPÇÃO ← "0"
    enquanto (OPÇÃO <> "5") faça
        escreva "1 - Adição"
        escreva "2 - Subtração"
        escreva "3 - Multiplicação"
        escreva "4 - Divisão"
        escreva "5 - Fim de Programa"
        escreva "Escolha uma opção: "
        leia OPÇÃO
        se (OPÇÃO <> "5") então
            caso OPÇÃO
                seja "1" faça rot soma
                seja "2" faça rotsubtracão
                seja "3" faça rotmultiplicacão
                seja "4" faça rotdivisão
                senão escreva "Opção inválida - Tente novamente"
            fim_caso
        fim_se
    fim_enquanto
fim
```

Antes do uso da instrução **caso...fim_caso**, está sendo verificado se o valor da opção informado é realmente diferente de 5. Sendo, será verificado se é 1, 2, 3 ou 4; não sendo nenhum dos valores válidos, a instrução **senão** informará que a tentativa foi inválida. O uso desta nova instrução agiliza o processamento das instruções, ou seja, é muito mais rápido utilizar a instrução **caso...fim_caso** do que aninhar diversas instruções do tipo **se**.

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 OPÇÃO, 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 OPÇÃO é global, apesar de não estar sendo utilizada nas demais sub-rotinas.

Uma variável é considerada *Local* quando é declarada dentro de uma sub-rotina e é somente válida dentro da rotina à qual está declarada. Desta forma, as demais sub-rotinas e programa principal não poderão fazer uso daquelas variáveis como *Global*, pois não visualizam a existência delas. No programa anterior, as variáveis A, B e R são locais. Por este motivo estão sendo declaradas repetidas vezes dentro de cada sub-rotina.

Dependendo da forma como se trabalha com as variáveis, é possível economizar espaço em 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:

```
programa TROCA_VALORES_EXEMPLO_1
```

```
var
```

```
    X : inteiro
```

```
    A : inteiro
```

```
    B : inteiro
```

```
início
```

```
    leia A, B
```

```
    X ← A
```

```
    A ← B
```

```
    B ← X
```

```
    escreva A, B
```

```
fim
```

```
programa TROCA_VALORES_EXEMPLO_2
```

```
var
```

```
    A : inteiro
```

```
    B : inteiro
```

```
procedimento TROCA
```

```
var
```

```
    X : inteiro
```

```
início
```

```
    X ← A
```

```
    A ← B
```

```
    B ← X
```

```
fim
```

```
início
```

```
    leia A, B
```

```
    Troca
```

```
    escreva A, B
```

```
fim
```

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 gráfico apresentado na figura 11.5.

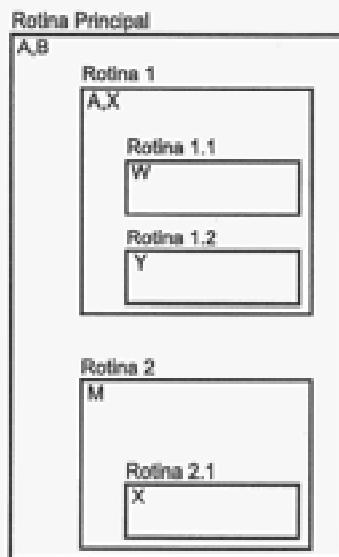


Figura 11.6 - Representação do escopo de variáveis.

Com relação ao gráfico da figura 11.5, 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.

Um cuidado importante a ser considerado com relação ao escopo de uma variável é o fato de uma variável ser declarada antes ou depois de uma seqüência de sub-rotinas. Se declarada antes, ela será global a todas as sub-rotinas existentes após a sua declaração. Porém, se for declarada após uma seqüência de sub-rotinas, ela será global do ponto em que está para baixo e local com relação às sub-rotinas definidas acima dela. Por exemplo, o programa de troca apresentado anteriormente tem as variáveis A e B globais para todo o programa. Mas se as variáveis A e B fossem declaradas após a sub-rotina, seriam locais, conforme é demonstrado em seguida:

```
programa TROCA_VALORES
```

ABRIR F1

(Rotina com erro de declaração de variáveis)

```
procedimento TROCA
```

Lembrar

```
var
```

X : inteiro

```
inicio
```

X ← A

A ← B

B ← X

```
fim
```

```
var
```

A : inteiro

B : inteiro

```
inicio
```

leia A, B

Troca

escreva A, B

```
fim
```

Isto caracteriza um erro, pois as variáveis A e B para a sub-rotina TROCA não são válidas por não estarem definidas. Desta forma, estas variáveis precisam ser declaradas dentro da sub-rotina.

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 calculadora apresentado anteriormente permite que seja aplicada esta técnica, pois existem nas quatro sub-rotinas de cálculo, instruções que efetuam as mesmas tarefas. Por exemplo: a entrada e a 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 a outra para 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 saída. 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.

```
programa CALCULADORA
var
    OPÇÃO : caractere
    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
    fim
procedimento SAÍDA
    inicio
        escreva "O resultado de A com B é = ", R
    fim

    (Sub-rotinas de cálculos)

procedimento ROTSUMA
    inicio
        escreva "Rotina de Adição"
        entrada
        R ← A + B
        saída
    fim

procedimento ROTSUBTRAÇÃO
    inicio
        escreva "Rotina de Subtração"
```

```
entrada
R ← A - B
saida
fim
```

procedimento ROTMULTIPLICAÇÃO

```
início
  escreva "Rotina de Multiplicação"
  entrada
  R ← A * B
  saida
```

```
fim
```

procedimento ROTDIVISÃO

```
início
  escreva "Rotina de Divisão"
  entrada
  R ← A / B
  saida
fim
```

(Programa Principal)

```
início
  OPÇÃO ← "0"
  enquanto (OPÇÃO <> "5") faça
    escreva "1 - Adição"
    escreva "2 - Subtração"
    escreva "3 - Multiplicação"
    escreva "4 - Divisão"
    escreva "5 - Fim de Programa"
    escreva "Escolha uma opção: "
    leia OPÇÃO
    se (OPÇÃO <> "5") então
      caso OPÇÃO
        seja "1" faça rotsoma
        seja "2" faça rotsubtracão
        seja "3" faça rotmultiplicacão
        seja "4" faça rotdivisão
        senão escreva "Opção inválida - Tente novamente"
      fim_caso
    fim_se
  fim_enquanto
fim
```

11.4 - Exercício de Fixação

Com base nas técnicas de programação apresentadas neste capítulo, desenvolva a solução para os seguintes exercícios:

- 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 do conceito de sub-rotinas, apresente um menu e suas respectivas rotinas para a execução das seguintes etapas:

 - a) Cadastrar os 10 registros.
 - b) Pesquisar os 10 registros, um 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 do conceito de sub-rotinas, apresente um menu e suas respectivas rotinas para a execução das seguintes etapas:

 - a) Cadastrar os 20 registros (após o cadastro efetuar a classificação por nome).
 - b) Pesquisar os 20 registros, um de cada vez, pelo campo nome (usar o método binário; 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 reaprovação.
 - d) Apresentar apenas os registros e médias dos alunos aprovados.
 - e) Apresentar apenas os registros e médias dos alunos reprovados.
 - f) Sair do programa de cadastro.

- 3 - Elaborar um programa que armazene o nome e a altura de 15 pessoas com o uso de registros. O programa deverá ser manipulado por meio do uso de sub-rotinas tanto na apresentação do menu, como também de suas rotinas, para a execução das seguintes etapas:
- 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 todos os registros com a média extraída de todas as alturas armazenadas.
 - f) Sair do programa de cadastro.
- 4 - Considerando os registros de 20 funcionários, contendo os campos matrícula, nome e salário, desenvolver um programa que utilizando conceito de sub-rotinas apresente um menu e suas respectivas rotinas para a execução das seguintes etapas:
- a) Cadastrar os 20 empregados.
 - b) Classificar os registros por número de matrícula.
 - c) Pesquisar um determinado empregado pelo número de matrícula (método binário).
 - d) Apresentar de forma ordenada os registros dos empregados que recebem salários acima de \$1.000.
 - e) Apresentar de forma ordenada os registros dos empregado que recebem salários abaixo de \$1.000.
 - g) Apresentar de forma ordenada os registros dos empregados que recebem salários iguais a \$1.000.
 - h) Sair do programa de cadastro.

Anotações

CAPÍTULO 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
início
  Z ← A + B
  escreva Z
fim
```

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
    leia Y
    calcsoma(X, Y)
    leia W
    leia T
    calcsoma(W, T)
    calcsoma(8, 2)
fim
```

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:

Diagramas de Blocos

Quando se utilizam parâmetros em uma sub-rotina, eles deverão ser mencionados nos diagramas de blocos, conforme o exemplo da figura 12.1. Note a indicação **FATORIAL(N)** no símbolo de Terminal.

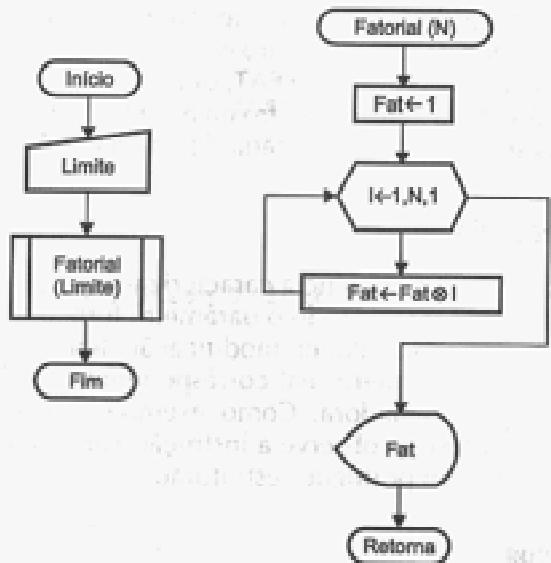


Figura 12.1 - Diagramas de blocos com a sub-rotina FATORIAL.

Português Estruturado

```

programa CALC_FATORIAL_V1
procedimento FATORIAL(N : inteiro)
var
    I, FAT : inteiro
inicio
    FAT ← 1
    para I de 1 até N passo 1 faça
        FAT ← FAT * I
    fim_para
    escreva FAT
fim
var
    LIMITE : inteiro
inicio
    escreva "Qual factorial: "
    leia LIMITE
    fatorial(LIMITE)
fim

```

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 factorial 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 “presa” 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:

Diagramas 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 de programa um resultado.

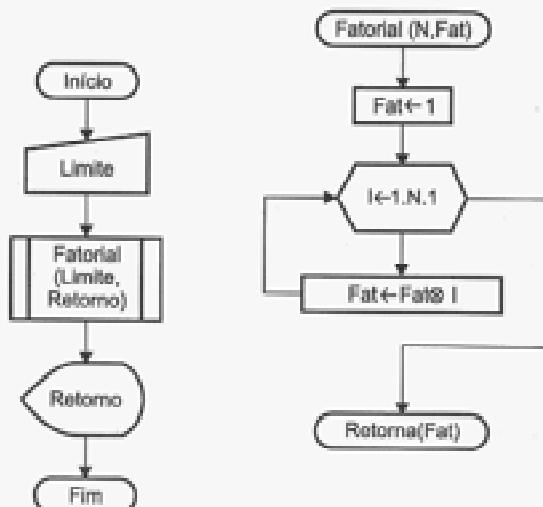


Figura 12.2 - Diagramas de blocos com a sub-rotina FATORIAL.

```

programa CALC_FATORIAL_V2
procedimento FATORIAL(N : inteiro, var FAT : inteiro)
var
  I : inteiro
inicio
  FAT ← 1
  para I de 1 até N passo 1 faça
    FAT ← FAT * I
  fim_para
fim

var
  LIMITE, RETORNO : inteiro
inicio
  escreva "Qual fatorial: "
  leia LIMITE
  factorial(LIMITE, RETORNO)
  escreva RETORNO
fim

```

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 neste exemplo continua sendo do tipo passagem de parâmetro por valor, pois ela 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 é do tipo passagem de parâmetro por referência e possui no final o valor acumulado do cálculo da factorial. 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. Então, a instrução `escreva RETORNO` imprime 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, considere a leitura de dez valores em uma matriz de uma dimensão (vetor) e coloque-os em ordem crescente. A ordenação deverá ser executada com sub-rotina.

O processo de ordenação já é conhecido de exemplos anteriores. 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 essa troca só ocorre após a verificação de uma condição, em que 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, este problema poderá ser resolvido de duas formas básicas, ou seja, a criação de duas sub-rotinas, sendo uma para a troca e a outra para a verificação da condição, ou uma única sub-rotina com verificação de condição e troca. Tanto uma forma quanto a outra deverão acompanhar os seguintes passos:

- 1 - Criar duas sub-rotinas, sendo uma para a troca e outra para verificação da condição;
- 2 - Para a troca, estabelecer três variáveis: X como auxiliar e A, 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 serão por referência.

1º 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 com a instrução var.

Diagramas de Blocos

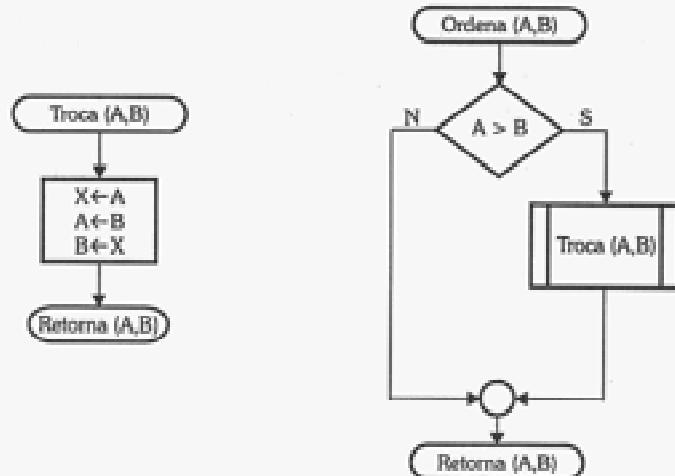


Figura 12.3 - Diagramas de blocos para sub-rotinas de troca e de ordenação.

Português Estruturado

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

procedimento ORDENA(var A, B : inteiro)
inicio
  se (A > B) então
    troca(A, B)
  fim_se
fim.
```

2^a 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

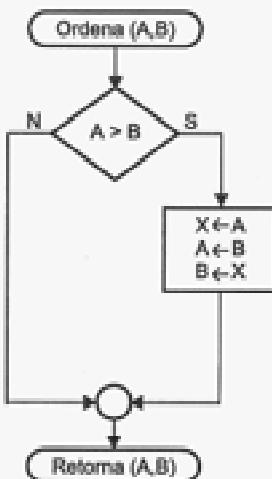


Figura 12.4 - Diagrama de blocos para sub-rotina de ordenação.

Português Estruturado

```
procedimento ORDENA(var A, B : inteiro)
var
  X : inteiro
início
  se (A > B) então
    X ← A
    A ← B
    B ← X
  fim_se
fim
```

A seguir, é apresentado um exemplo de um programa em código português estruturado, 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.

```
programa ORDENA_NÚMEROS
var
  VETOR : conjunto[1..10] de inteiro
  I, J : inteiro

procedimento ORDENA(var A, B : inteiro)
var
  X : inteiro
início
  se (A > B) então
    X ← A
    A ← B
    B ← X
  fim_se
fim
início
  (ENTRADA DE DADOS)

  para I de 1 até 10 passo 1 faça
    leia VETOR[I]
  fim_para

  (ORDENAÇÃO)

  para I de 1 até 9 passo 1 faça
    para J de I + 1 até 10 passo 1 faça
```

```

    ordena(VETOR[I], VETOR[J])
    fim_para
fim_para

(SAÍDA DE DADOS)
```

```

para I de 1 até 10 passo 1 faça
    escreva VETOR[I]
fim_para
fim
```

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 caractere, 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. Veja em seguida, como ficará este exemplo.

Diagrama de Blocos

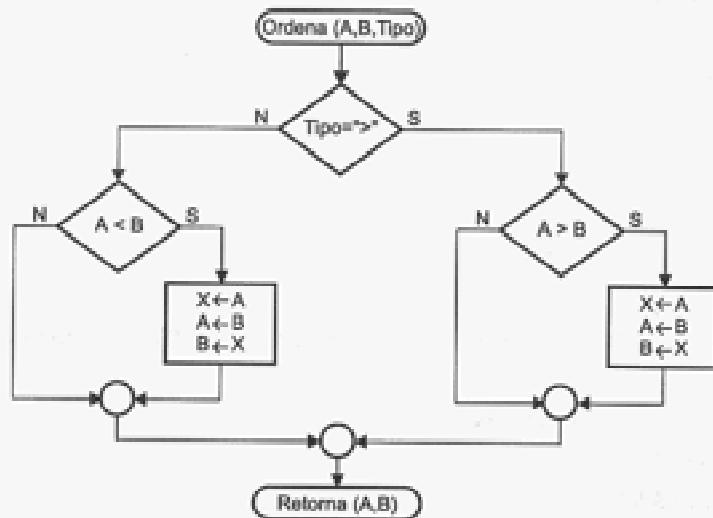


Figura 12.5 - Diagrama de blocos para a sub-rotina de ordenação crescente ou decrescente.

Português Estruturado

```
procedimento ORDENA(var A, B : inteiro, TIPO : caractere)
var
    X : inteiro
inicio
    se (TIPO = ">") então
        se (A > B) então
            X ← A
            A ← B
            B ← X
        fim_se
    senão
        se (A < B) então
            X ← A
            A ← B
            B ← X
        fim_se
    fim_se
fim
```

Com este exemplo é adequado aplicar a primeira forma de solução apresentada anteriormente, pois a troca de valores repete por duas vezes, sendo uma para o processo crescente e outra para o decrescente. Sendo assim, a rotina escrita em código português estruturado ficará:

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

```
procedimento ORDENA(var A, B : inteiro, TIPO : caractere)
inicio
    se (TIPO = ">") então
        se (A > B) então
            troca(A, B)
        fim_se
fim
```

```

senão
    se (A < B) então
        troca(A, B)
    fim_se
fim_se
fim

```

A seguir, é apresentado o algoritmo em código português estruturado, fazendo uso da sub-rotina de ordenação anteriormente apresentada.

```

programa ORDENA_NÚMEROS_V2
var
    VETOR : conjunto[1..10] de inteiro
    I, J : inteiro
    OPÇÃO : caractere

procedimento TROCA(var A, B : inteiro)
var
    X : inteiro
início
    X ← A
    A ← B
    B ← X
fim

procedimento ORDENA(var A, B : inteiro, TIPO : caractere)
início
    se (TIPO = ">") então
        se (A > B) então
            troca(A, B)
        fim_se
    senão
        se (A < B) então
            troca(A, B)
        fim_se
    fim_se
fim
início

(ENTRADA DE DADOS)
escreva "Digite > p/ crescente ou < p/ decrescente:"
leia OPÇÃO

```

```
para I de 1 até 10 passo 1 faça
    leia VETOR[I]
fim_para
```

(ORDENAÇÃO)

```
para I de 1 até 9 passo 1 faça
    para J de I + 1 até 10 passo 1 faça
        ordena(VETOR[I], VETOR[J], OPÇÃO)
    fim_para
fim_para
```

(SAÍDA DE DADOS)

```
para I de 1 até 10 passo 1 faça
    escreva VETOR[I]
fim_para
fim
```

Neste capítulo, você teve a oportunidade de aprender a organizar melhor a idéia de desenvolvimento de programas, não só com menus e rotinas distintas, mas aprendeu a utilizar o conceito de sub-rotinas para simplificar operações que muitas vezes se repetem ao longo de um programa.

12.4 - Exercício de Fixação

- 1) Desenvolva os algoritmos dos problemas abaixo indicados e suas sub-rotinas do tipo procedimento. Cada problema deverá ser resolvido usando passagem de parâmetro por valor e por referência.
 - a) 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.
 - b) Elaborar um programa que possua uma sub-rotina que efetue e permita apresentar o somatório dos N primeiros números inteiros, definidos por um operador. ($1+2+3+4+5+6+7+\dots+N$).
 - c) 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 subseqüente. Apresentar o resultado.

- d) 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 POTÊNCIA(2,3), deverá ser apresentado o valor 8.
- e) Elaborar um programa que efetue a leitura de um número inteiro e apresente uma mensagem informando se o número é par ou ímpar.
- f) 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 efetue a leitura de três valores (A, B e C) e apresente como resultado final o quadrado da soma dos três valores lidos.
- h) 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.
- i) 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.
- j) 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".
- k) Elaborar um programa que por meio de sub-rotina efetue a apresentação da mensagem: "Este valor é divisível por 2 ou 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".
- l) Elaborar um programa que por meio de sub-rotina efetue a apresentação da mensagem: "Este valor não 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".
- m) 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.
- n) 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.

- o)** Elaborar um programa que por meio de sub-rotina apresente o resultado do valor de uma fatorial de um número qualquer.
- p)** 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 \$10 e \$20. 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.

CAPÍTULO 13

Aplicação Prática do Uso de Sub-Rotinas - Funções

Uma **Função** também é um bloco de programa, como são os procedimentos, contendo início e fim e sendo identificada por um nome, pelo 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. Observe em seguida, o código em português estruturado de uma função.

Português Estruturado

```
função <nome da função>(<parâmetros>) : <tipo da função>
var
    <variáveis>
início
    <instruções>
fim
```

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 levados em consideração os 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 a factorial 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 até N passo 1 faça
    FAT ← FAT * I
  fim_para
fim

```

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 factorial. Veja em seguida, como isto é feito:

```

função FATORIAL(N : inteiro) : inteiro
var
  I, FAT : inteiro
inicio
  FAT ← 1
  se (N = 0) então
    FATORIAL ← 1
  senão
    para I de 1 até N passo 1 faça
      FAT ← FAT * I
    fim_para
    FATORIAL ← FAT
  fim_se
fim

```

Observe que o nome da função, no caso FATORIAL, é também o nome da variável interna que recebe o valor acumulado da variável FAT, caso o número fornecido para o parâmetro N não seja zero. Pois se for zero, o valor da função FATORIAL é igualado a 1, que é o valor da factorial de zero. Desta forma, uma função retorna um valor pelo seu próprio nome, pois esse nome é usado dentro do corpo da função para a recepção do valor calculado.

Outro detalhe a ser considerado numa função, além do seu nome ser usado como variável de retorno, é o fato da definição do seu tipo, no caso, função 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. Observe que poderá ocorrer o fato de entrar um parâmetro de um tipo e retorná-lo como outro tipo.

13.2 - Considerações a Respeito de Funções

Você pode até estar achando que o número de linhas da função FATORIAL apresentada acima 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
```

Caso se deseje obter o resultado da factorial de 5; o limite é colocado na sub-rotina que efetuará o cálculo e transferirá para a variável RETORNO que será em seguida escrita. Perceba que são necessárias duas linhas de programa.

Considerando que seja utilizada a função FATORIAL, ela deve ser mencionada na rotina chamadora como:

```
escreva fatorial(5)
```

Destra forma o valor a ser calculado, no caso 5, é transferido para a sub-rotina que após efetuar o cálculo devolve o resultado na própria função. Isto facilita a utilização deste valor para qualquer operação computacional, enquanto a sub-rotina de procedimento equivalente exige o uso de uma variável para receber o retorno de um resultado transferido por passagem de parâmetro por referência.

Observe que na função foi utilizada a passagem de parâmetro por valor, no caso a variável N, pois o retorno está sendo efetuado na própria sub-rotina (função) e não em uma variável, como foi o caso do procedimento equivalente.

Falando em parâmetro, vale salientar que uma função também poderá receber a passagem de parâmetro por valor ou referência. Vai depender muito do problema computacional a ser resolvido. Se for necessário somente informar um valor e este gerar e retornar um resultado para função (sendo esta a forma mais comum), então utiliza-se a passagem de parâmetro por valor.

Porém, poderá ocorrer em alguns casos, além de se obter o retorno da função, obter-se o valor alterado do parâmetro. Neste caso, a passagem do parâmetro deverá ser feita por referência.

13.3 - Exercício de Aprendizagem

Para demonstrar a utilização de programas 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.

Diagramas de Bloco

Na diagramação, não há muita novidade na representação gráfica, como mostrado em seguida na figura 31.1.

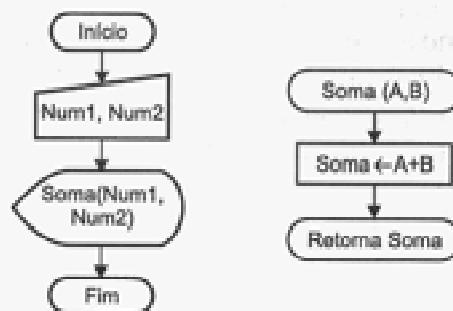


Figura 13.1 - Diagramas da função soma e do programa principal.

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 soma dos dois valores e implicada à variável SOMA que efetua o retorno à função, o qual também é do tipo real.

```

função SOMA(A, B : real) : real
  inicio
    SOMA ← A + B
  fim

  var
  NUM1, NUM2 : real
  inicio
    escreva "Informe o 1o. valor: "
    leia NUM1
    escreva "Informe o 2o. valor: "
    leia NUM2
    escreva "Soma = ", soma(NUM1, NUM2)
  fim

```

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.

Diagramas de Blocos

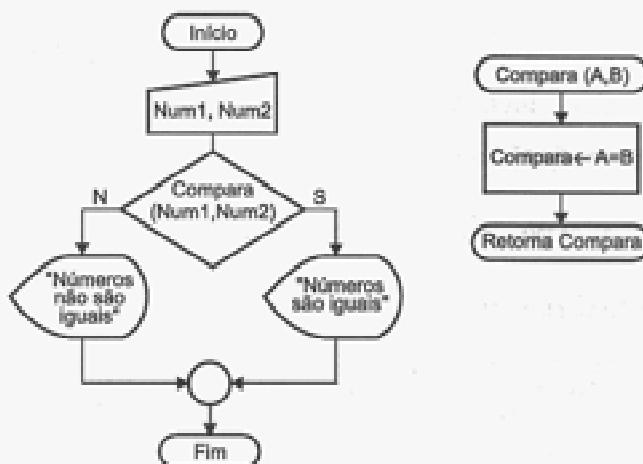


Figura 13.2 - Diagramas da função comparação e do programa principal.

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.

```
programa COMPARAÇÃO

função COMPARA(A, B : real) : lógico
início
  COMPARA ← A = B
fim

var
  NUM1, NUM2 : real
início
  escreva "Informe o 1o. valor: "
  leia NUM1
  escreva "Informe o 2o. valor: "
  leia NUM2
```

```

se (compara(NUM1, NUM2)) então
    escreva "Números iguais"
senão
    escreva "Números diferentes"
fim_se
fim

```

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 saída 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 a soma dos dois valores;
- 2 - Se o operador for "-", faz-se a subtração dos dois valores;
- 3 - Se o operador for "*", faz-se a multiplicação dos dois valores;
- 4 - Se o operador for "/", faz-se a 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.

```

função CALCULO(A, B : real, OPERADOR : caractere) : real
início
    caso OPERADOR
        seja "+" faça CALCULO ← A + B
        seja "-" faça CALCULO ← A - B
        seja "*" faça CALCULO ← A * B
        seja "/" faça CALCULO ← A / B
    fim_caso
fim

```

A seguir, é apresentado somente o algoritmo em português estruturado do programa calculadora, aplicando o uso da nova função.

programa CALCULADORA

var

OPÇÃO : caractere

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

fim

(Função para o cálculo das operações)

função CÁLCULO(A, B : real, OPERADOR : caractere) : real

inicio

caso OPERADOR

seja "+" faça CÁLCULO ← A + B

seja "--" faça CÁLCULO ← A - B

seja "***" faça CÁLCULO ← A * B

seja "/" faça CÁLCULO ← A / B

fim_caso

fim

procedimento SAÍDA

inicio

escreva "O resultado de A com B é = ", R

fim

(Sub-rotinas de cálculos)

procedimento ROTSSOMA

inicio

escreva "Rotina de Adição"

entrada

R ← cálculo(A, B, "+")

saida

fim

procedimento ROTSUBTRAÇÃO**inicio**

escreva "Rotina de Subtração"

entrada $R \leftarrow \text{cálculo}(A, B, "-")$ saida**fim****procedimento ROTMULTIPLICAÇÃO****inicio**

escreva "Rotina de Multiplicação"

entrada $R \leftarrow \text{cálculo}(A, B, "\times")$ - também use a opção * saida**fim****procedimento ROTDIVISÃO****inicio**

escreva "Rotina de Divisão"

entrada $R \leftarrow \text{cálculo}(A, B, "/")$ - também use a opção / saida**fim**

(Programa Principal)

inicio OPÇÃO $\leftarrow "0"$

enquanto (OPÇÃO <> "5") faça

escreva "1 - Soma"

escreva "2 - Subtração"

escreva "3 - Multiplicação"

escreva "4 - Divisão"

escreva "5 - Fim de Programa"

escreva "Escolha uma opção: "

leia OPÇÃO

se (OPÇÃO <> "5") então

caso OPÇÃO

seja "1" faça rotsoma

seja "2" faça rotsubtracão

seja "3" faça rotmultiplicacão

seja "4" faça rotdivisão

senão escreva "Opção inválida - Tente novamente"

fim_caso

fim_se

fim_enquanto

fim

13.4 - Exercício de Fixação

- 1) Desenvolva os algoritmos dos problemas abaixo, usando o conceito de sub-rotinas de funções.
- Elaborar um programa que possua uma sub-rotina que apresente o somatório dos N primeiros números inteiros, definidos por um operador ($1+2+3+4+5+6+7+\dots+N$).
 - 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.
 - Criar um algoritmo que efetue o cálculo e apresente o valor de uma prestação em atraso. Para tanto, utilize a fórmula $\text{PREST} = \text{VALOR} + (\text{VALOR} * (\text{TAXA}/100) * \text{TEMPO})$.
 - 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 $\text{POTÊNCIA}(2,3)$, deverá ser apresentado o valor 8.
 - 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.
 - 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.
 - Elaborar um programa que efetue a apresentação do valor da conversão em real (R\$) de um valor lido em dólar (US\$). O programa deverá solicitar o valor da cotação do dólar e também a quantidade de dólares disponível com o usuário.
 - 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.
 - Elaborar um programa que com o uso de uma sub-rotina do tipo função apresente o valor de uma temperatura em graus Celsius. O programa deverá ler a temperatura em graus Fahrenheit.

PARTE V

Apêndices

APÊNDICE A

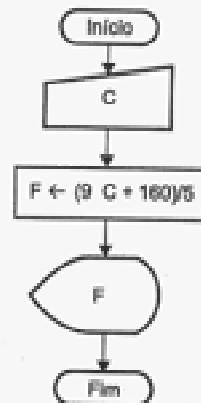
Resoluções de Alguns Exercícios de Fixação

A título de ilustração, são apresentados neste apêndice alguns dos exercícios de fixação resolvidos, objetivando dar algumas idéias de como os demais exercícios deverão ser resolvidos.

Capítulo 3 - Exercício 6a.

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.

```
programa TEMPERATURA
var
  C, F : real
inicio
  leia C
  F ← (9 * C + 160) / 5
  escreva F
fim
```



Capítulo 4 - Exercício 3c.

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 com uma das mensagens o valor da média do aluno para qualquer condição.