8ª EDIÇÃO
Revisada e Atualizada

PROGRAMANDO EM TURBO PASCAL 7.0

BRINDE
Programas do livro disponíveis na INTERNET.

José Augusto N. G. Manzano
Wilson Y. Yamatumi

& FREE PASCAL COMPILER

Guia Prático de Orientação e Desenvolvimento

- *Programação Seqüencial, Estruturada e Modular*
- *Tabelas, Matrizes e Arquivos*
- *Utilização de Impressora e de Relatórios*
- *Manipulação de Calendários e de Datas*
- *Rotinas CNPJ (CGC), CPF e PIS*
- *Exemplos Práticos de Aprendizagem*
- *Exercícios de Fixação com Respostas*
- *Compiladores Freeware e Shareware*

Inclui informações sobre Virtual Pascal.

ÉRICA
www.editoraerica.com.br

PROGRAMANDO EM TURBO PASCAL 7.0 & FREE PASCAL COMPILER

Guia Prático de Orientação e Desenvolvimento

UNIVERSITÁRIO - Centro Universitário - Biblioteca Silva Freire

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, automaticamente, 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, 159 - Tatuapé - CEP: 03401-030 - São Paulo - SP

Invista em você.
Visite uma livraria.

José Augusto N. G. Manzano

Wilson Y. Yamatumi

Ano: 2004 2003 **2002**

Edição: 13 12 11 10 9 **8**

Edição Revisada e Atualizada

Editora Érica Ltda.

Conselho Editorial:

Diretor Editorial:	*Antonio Marco Vicari Cipelli*
Diretor Comercial:	*Paulo Roberto Alves*
Diretor de Publicidade:	*Waldir João Sandrini*
Editoração:	*Graziela M. L. Gonçalves De Filippis*
Capa:	*Maurício S. de França*
Desenhos:	*Pedro Paulo Vieira Herruzo*
Revisão Interna:	*Érica Regina Antonio Pagano*
Revisão Gramatical:	*Marlene Teresa Santin Alves*
Revisão e Coordenação:	*Rosana Arruda da Silva*

Centro Universitário - UNIVATES - Biblioteca Silva Freire

Copyright © 1996 (edição original) e 2002 (edição atualizada) da Editora Érica Ltda.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Manzano, José Augusto Navarro Garcia, 1965-
Programando em Turbo Pascal 7.0: Guia Prático de Orientação e Desenvolvimento / José Augusto N. G. Manzano, Wilson Yoshiteru Yamatumi. - - 7 ed. rev., atual. e ampl. - São Paulo: Érica, 2001.

Bibliografia.
ISBN: 85-7194-777-5

1. Turbo Pascal (Linguagem de Programação para computadores) I. Yamatumi, Wilson Yoshiteru, 1976- II. Título

01-0662 CDD-005.133

Índices para Catálogo Sistemático:

1. Turbo Pascal 7.0: Linguagem de programação: Computadores eletrônicos: Processamento de dados 005.133

Todos os direitos reservados. Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfílmicos, fotográficos, reprográficos, fonográficos, videográficos, internet, e-books. Vedada a memorização e/ou recuperação total ou parcial em qualquer sistema de processamento de dados e a inclusão de qualquer parte da obra em qualquer programa juscibernético. Essas proibições aplicam-se também às características gráficas da obra e à sua editoração. A violação dos direitos autorais é punível como crime (art. 184 e parágrafos, do Código Penal, cf. Lei nº 6.895, de 17.12.80) com pena de prisão e multa, conjuntamente com busca e apreensão e indenizações diversas (artigos 102, 103 parágrafo único, 104, 105, 106 e 107 itens 1, 2 e 3 da Lei nº 9.610, de 19/06/98, Lei dos Direitos Autorais).

Os Autores e a Editora acreditam que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia, explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes dè sites e empresas, porventura mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis no site da Editora Érica para download.

"Algumas imagens utilizadas neste livro foram obtidas a partir do CorelDRAW 7, 8 e 9 e da Coleção do MasterClips/ MasterPhotos© da IMSI, 1985 Francisco Blvd. East, San Rafael, CA 94901-5506, USA."

BIBLIOTECA SILVA FREIRE
G: 60631
DATA: 23 / 11 / 004
UNIVAG - CENTRO UNIVERSITÁRIO

Editora Érica Ltda.
Rua São Gil, 159 - Tatuapé
CEP: 03401-030 - São Paulo - SP
Fone: (11) 295-3066 - Fax: (11) 6197-4060
Site: **www.editoraerica.com.br**

Biblioteca Silva Freire - UNIVAG
60631

FABRICANTES

SOFTWARE: TURBO PASCAL 7.0

INPRISE DO BRASIL

Rua Pedroso Alvarenga, 58 - 6º andar

CEP: 04531-000 - São Paulo - SP

Fone: (11) 3167-1722

Site (Brasil): www.borland.com.br

Site (USA): www.inprise.com

SOFTWARE: FREE PASCAL COMPILER

FLORIAN KLÄMPFL - Coordenador do Projeto

Contato pelo e-mail: florian@freepascal.org

Contatos em inglês ou alemão

Universidade Erlangen-Nürnberg, Alemanha

Site (Universidade): www.uni-erlangen.de

Site (Free Pascal): www.freepascal.org

Site (Free Pascal Brasil): www.freepascal.eti.br

REQUISITOS BÁSICOS DE HARDWARE E DE SOFTWARE

Turbo Pascal 7.0

- ✓ Computador padrão PC 286 ou superior (até Pentium de 400 Mhz);
- ✓ Mínimo de 1 MB de memória RAM disponível;
- ✓ MS-DOS 3.0 ou superior;
- ✓ Turbo Pascal 7.0;
- ✓ 6 MB de disco rígido em média para instalação;
- ✓ Mouse opcional.

FREE PASCAL COMPILER - MS-DOS/WINDOWS

- ✓ Computador padrão PC 486 ou superior;
- ✓ Mínimo de 4 MB de memória RAM em modo MS-DOS;
- ✓ Mínimo de 16 MB de memória RAM em modo Windows;
- ✓ MS-DOS 3.3 ou superior (MS-DOS) - Windows 95/98/NT;
- ✓ Free Pascal Compiler - MS-DOS/Windows;
- ✓ 8 MB de disco rígido em média para instalação;
- ✓ Mouse opcional.

DEDICATÓRIA

Dedico esta obra aos amigos, alunos e leitores, que com suas dúvidas e perguntas incentivam-me e fazem com que eu me aprimore cada vez mais, e assim possibilitam continuar o meu trabalho, com um grande abraço a todos.

Augusto Manzano

Dedico esta obra a todas as pessoas que direta ou indiretamente sempre me ajudaram até hoje; um cordial abraço.

Wilson Y. Yamatumi

Posso deitar-me, dormir e despertar,
pois é o Senhor quem me ampara.

Sl 3, 6

AGRADECIMENTOS

Agradeço ao amigo e editor Antonio Marco, que desde o nosso primeiro contato confiou no meu trabalho, e vem me dando a oportunidade de continuar a divulgar as minhas idéias por meio dos trabalhos que são publicados, em especial este livro que considero um dos mais importantes. Agradeço também a todos os funcionários da Editora Érica, pelo carinho, dedicação e apreço por minha pessoa.

Augusto Manzano

Agradeço ao amigo Augusto Manzano, pelo incentivo dado ao meu primeiro trabalho. Agradeço também aos amigos: Prof. Luiz Carlos Montevechi, Prof. Sérgio Sandrin e Prof. Reinaldo Madarazo.

Wilson Y. Yamatumi

PREFÁCIO

Esta obra baseia-se na experiência adquirida e acumulada no ensino de linguagens de programação há mais de quinze anos. Trata-se de um trabalho voltado tanto àqueles que estão iniciando o estudo da linguagem de programação Pascal, como àqueles que desejam explorar algumas de suas potencialidades básicas. Não é intenção nem pretensão dos autores abordar o assunto com extrema profundidade em relação aos recursos da linguagem Pascal, mas proporcionar uma visão técnica de forma que o leitor possa perceber o que é possível fazer com essa maravilhosa linguagem.

A intenção deste trabalho é fornecer ao leitor condições de dar os primeiros passos, com maior tranqüilidade, na tarefa de programar um computador por intermédio da linguagem de programação Pascal, estando assim preparado para alçar vôos mais altos.

A escolha da linguagem Pascal baseia-se no fato de ser muito poderosa e com uma sintaxe de fácil assimilação, além de ser utilizada em várias empresas de grande porte, e também no ensino de programação de computadores em várias escolas não só do Brasil. Outro ponto que levou à escolha dessa linguagem é o fato de ser um passaporte para o ambiente de desenvolvimento Delphi.

O ambiente Delphi faz uso da linguagem Object Pascal (linguagem Pascal Orientada a Objetos). O ambiente Turbo Pascal, a partir da versão 5.5, também é Orientado a Objetos, porém este assunto não será abordado nesta obra.

Para facilitar o aprendizado de qualquer linguagem, seria ideal que o leitor possuísse conhecimento básico de lógica de programação, para não correr o risco de desenvolver a lógica moldada sobre a linguagem Pascal.

É importante saber programar independentemente da linguagem a ser utilizada. Recomenda-se para este aprendizado o livro Algoritmos - Lógica para Desenvolvimento de Programação de Computadores, da Editora Érica, escrito pelos autores Augusto Manzano (também autor desta obra) e Jayr Figueiredo, o qual pode ser utilizado em conjunto com este.

O leitor vai perceber que alguns capítulos possuem exercícios de fixação e outros não. Os capítulos que possuem exercícios de fixação são estratégicos e apresentam técnicas de programação básicas, que necessitam ser conhecidas e entendidas por todos aqueles que desejam programar. Já os capítulos que não possuem exercícios de fixação são complementares e objetivam apresentar algum aprimoramento ao conhecimento da linguagem de programação Pascal.

Ao final é apresentada uma lista de referências bibliográficas, a qual pode e deve ser utilizada como complemento a este trabalho. Na lista encontra-se a indicação de outros livros e também de sites da Internet que podem facilmente ser consultados, a qualquer hora.

A todos um grande abraço e um bom aprendizado.

Os autores

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) há mais de dezessete anos. Atualmente, exerce a atividade profissional de consultor e professor universitário, ministrando as disciplinas de Estrutura de Dados, Técnicas de Programação (Lógica - Algoritmos), Introdução à Microinformática, Linguagens de Programação, Engenharia de Software e Tópicos Avançados em Processamento de Dados. Pela Editora Érica possui mais de trinta obras publicadas na área em que atua.

Wilson Y. Yamatumi

Brasileiro, nascido em São Bernardo do Campo, SP, em 10/07/76, é formado em Informática Industrial pela Escola Técnica Estadual "Jorge Street". Atualmente, ministra aulas particulares de Linguagens de Programação. Possui sólidos conhecimentos das linguagens de programação Pascal e C.

SOBRE O MATERIAL DISPONÍVEL NA INTERNET

O material disponível no site da Editora Érica (www.editoraerica.com.br) contém os programas desenvolvidos no livro e o gabarito dos exercícios.

Para utilizar os arquivos, é necessário que você possua os requisitos de hardware e de software descritos anteriormente.

Procedimento para Download

Acesse o site da Editora Érica: www.editoraerica.com.br. A transferência dos arquivos disponíveis pode ser feita de duas formas:

- ✓ **Por meio do módulo pesquisa**. Localize o livro desejado, digitando palavras--chaves (nome do livro ou do autor). Aparecerão os dados do livro e o arquivo para download, então dê um clique sobre o arquivo executável que será transferido.
- ✓ **Por meio do botão "Download"**. Na página principal do site, clique no item "Download". Será exibido um campo, no qual devem ser digitadas palavras-chaves (nome do livro ou do autor). Serão exibidos o nome do livro e o arquivo para download. Dê um clique sobre o arquivo executável que será transferido.

A seguir, é descrito o processo de instalação dos arquivos de programas desenvolvidos ao longo do livro. Os programas em questão são aqueles existentes junto da explicação de cada tópico, representados pelos exemplos de aprendizagem. O processo de instalação copia também dois arquivos de gabarito, sendo um em formato texto e o outro em formato *rtf* (que pode ser lido em qualquer processador de textos) para o diretório **\ERICAPAS**.

Para instalar os programas desenvolvidos no livro e o gabarito, é necessário ter o arquivo **PASAY.EXE**, o qual deve ser baixado do site da Editora Érica, conforme descrito anteriormente.

Descompacte o arquivo **PASAY.EXE** em um disquete formatado e verifique se foi criado o diretório **dados** (de outra forma não será possível proceder à instalação). Para isto, basta clicar duas vezes sobre o arquivo PASAY.EXE que automaticamente vai se descompactar dentro de um disquete na unidade A:\. Em seguida, execute o programa **INSTALA1** (compilado com o Turbo Pascal 7.0 - para computadores padrão 286, 386 e 486) ou **INSTALA2** (compilado em Free Pascal - para computadores padrão Pentium ou superior)[1]. Então,

---

[1] As fontes dos programas **INSTALA** também estão disponíveis no diretório **ERICAPAS** com os nomes **INSTALA_TP7** (versão Turbo Pascal) e **INSTALA_FPC** (versão Free Pascal). Desta forma, o leitor pode ter idéia de mais alguns detalhes que podem ser usados com a Linguagem Pascal.

aparecem a tela de apresentação e a tela de advertência. A figura A.1 apresenta a imagem da tela de advertência em questão.

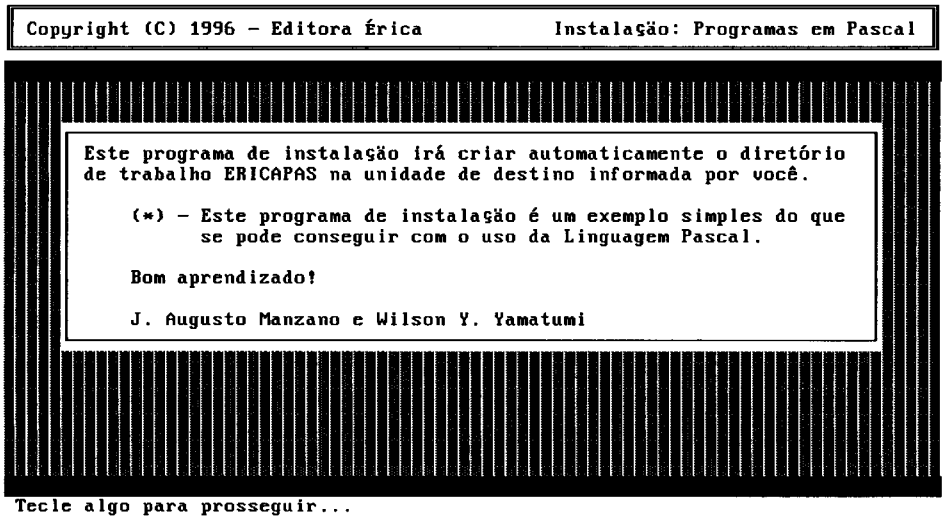


Figura 1 - Apresentação das orientações iniciais.

Após efetuar a leitura da tela de advertência, pressione alguma tecla de seu teclado para prosseguir com o processo de instalação. Será então apresentado um menu com duas opções: **Instalação** que dá início ao processo de instalação e **Encerrar** que finaliza o programa sem instalar os arquivos, como mostra a figura 2.

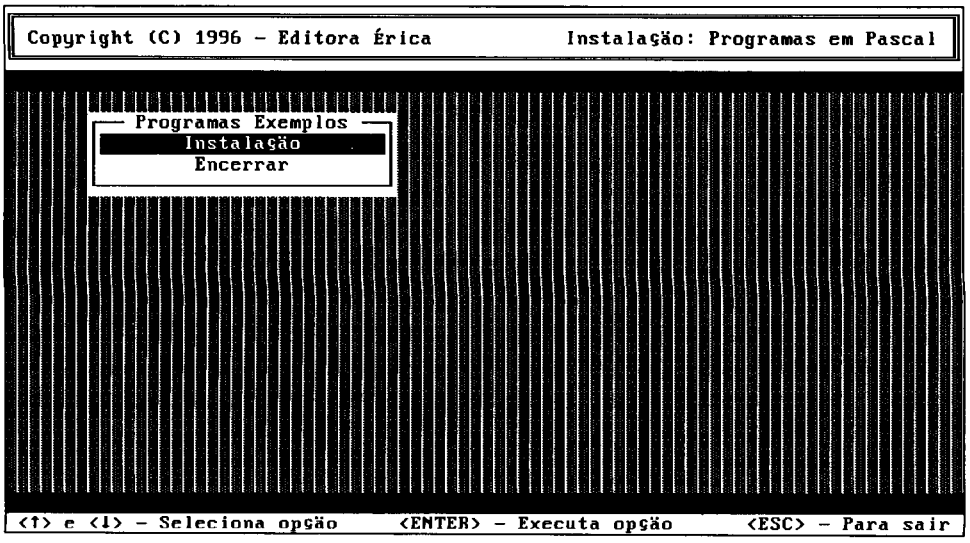


Figura 2 - Apresentação do menu de seleção.

Escolha **Instalar** e será apresentada uma caixa solicitando que se informe qual a unidade de origem (figura 3). Neste momento, você pode apenas digitar A ou B, que é uma das unidades de disco do seu computador. Selecione a sua unidade sem teclar **<Enter>**, pois a aceitação é processada automaticamente. Em seguida deve ser informada a unidade de destino, que pode ser C ou D (no caso de computadores com dois discos rígidos). Normalmente se digita C sem teclar **<Enter>**. Será apresentado o nome do diretório (**ERICAPAS**) que será criado automaticamente no processo de instalação.

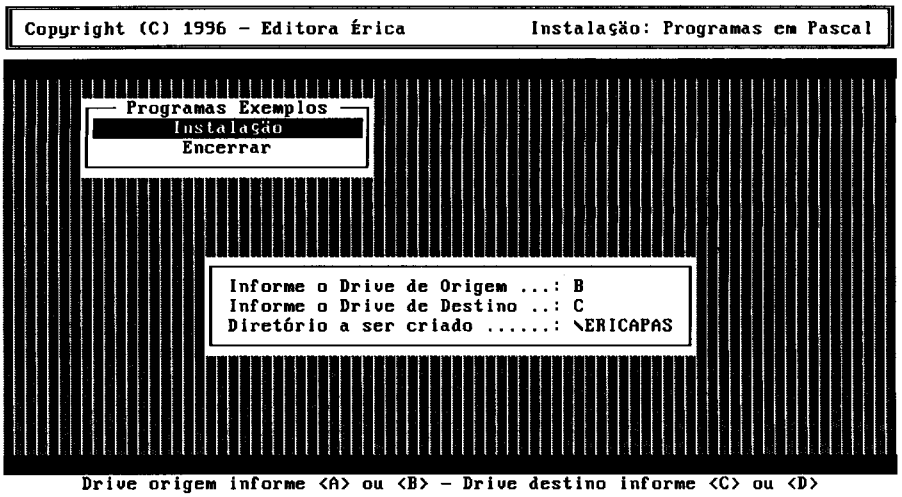


Figura 3 - Apresentação da caixa de instalação.

Após estes procedimentos, será dado início ao processo de instalação (figura 4) propriamente dito. Os programas serão copiados para o diretório da unidade indicada.

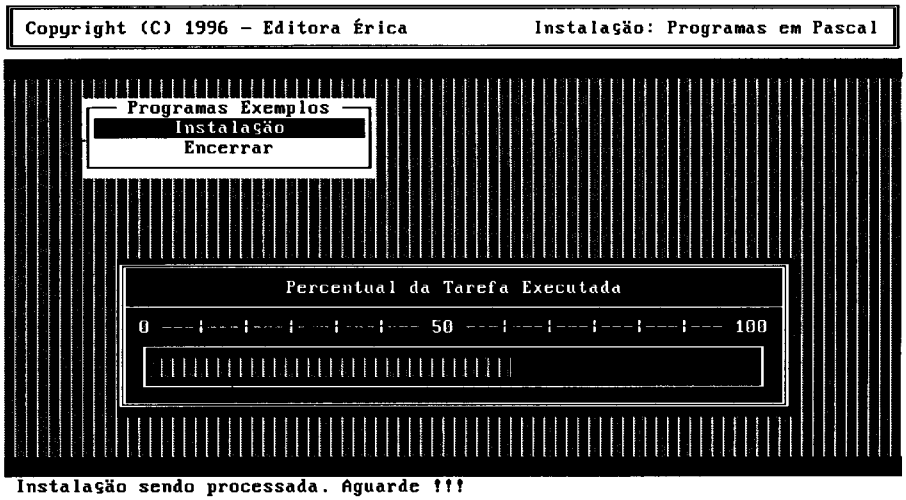


Figura 4 - Indicação do percentual de arquivos instalados.

UNIVAG

Ao término do processo o que não deve demorar muito, é apresentada uma mensagem (figura 5) indicando o fim do processo. Neste momento tecle algo para finalizar o programa de instalação.

Figura 5 - Final do processo de instalação.

ÍNDICE ANALÍTICO

Parte 1 - Conceitos Introdutórios 21

Capítulo 1 - Introdução 23
1.1 - Breve Histórico da Linguagem Pascal 24
1.2 - Notação Utilizada 26
1.3 - Aquisição do Free Pascal Compiler 27
1.4 - Instalação do Ambiente Pascal 29
1.4.1 - Instalação do Turbo Pascal 29
1.4.2 - Instalação do Free Pascal 33
1.5 - Considerações Operacionais 35

Capítulo 2 - O Ambiente de Trabalho 37
2.1 - O Menu de Comandos do Turbo Pascal 7.0 37
2.2 - O Menu de Comandos do Free Pascal Compiler 40
2.3 - Como Operar os Ambientes TP7 e FPC 42

Parte 2 - Programação Básica 49

Capítulo 3 - Tipos de Dados e Instruções Primitivas 51
3.1 - Tipos de Dados 51
3.1.1 - Tipos de Dados Inteiros 51
3.1.2 - Tipos de Dados Reais 52
3.1.3 - Tipos de Dados Caracteres 52
3.1.4 - Tipos de Dados Lógicos 52
3.2 - O Uso de Variáveis *52*
3.3 - O Uso de Constantes 53
3.4 - Os Operadores Aritméticos 53
3.5 - As Expressões Aritméticas 54
3.6 - Instruções Básicas 55
3.6.1 - Algumas Regras Antes de Começar 55
3.6.2 - A Entrada, Processamento e Saída de Dados 55
3.7 - Estrutura de um Programa em Pascal 55
3.7.1 - Cabeçalho do Programa 55
3.7.2 - Área de Declarações 56
3.7.3 - Corpo do Programa 57
3.7.4 - Diretório de Trabalho 57

3.8 - O Primeiro Programa em Pascal 57
3.9 - Exercício 64

Capítulo 4 - A Tomada de Decisões 65
4.1 - Desvio Condicional Simples 65
4.2 - Operadores Relacionais 68
4.3 - Desvio Condicional Composto 68
4.4 - Operadores Lógicos 70
4.4.1 - Operador Lógico: AND 70
4.4.2 - Operador Lógico: OR 71
4.4.3 - Operador Lógico: NOT 71
4.5 - Exercícios 74

Capítulo 5 - Laços, Loopings ou Malhas de Repetição 77
5.1 - Repetição com Teste Lógico no Início do Looping 78
5.2 - O Modo Debug 83
5.2.1 - Debug no Turbo Pascal 83
5.2.2 - Debug no Free Pascal 89
5.3 - Repetição do Tipo com Teste Lógico no Fim do Looping 94
5.4 - Repetição com Variável de Controle 97
5.5 - Consideração entre as Estruturas de Looping 99
5.6 - Exercício 100

Parte 3 - Tabelas em Memórias 101

Capítulo 6 - Estruturas de Dados Homogêneas I 103
6.1 - Matrizes de uma Dimensão ou Vetores 103
6.2 - Operações com Matrizes do Tipo Vetor 105
6.3 - Aplicações Práticas 109
6.3.1 - Classificação de Elementos 110
6.3.2 - Métodos de Pesquisa 115
6.4 - Exercício 123

Capítulo 7 - Estruturas de Dados Homogêneas II 125
7.1 - Matrizes com mais de Uma Dimensão 125
7.2 - Operações Básicas com Matrizes de Duas Dimensões 126
7.3 - Exercício 133

Capítulo 8 - Estruturas de Dados Heterogêneas 135
8.1 - Estrutura de Dados com o Uso do Tipo Registro 135
8.2 - Aplicação Prática do Tipo Registro 140
8.3 - Exercício 142

Parte 4 - Programação Estruturada ou Modular 143

Capítulo 9 - Utilização de Sub-rotinas 145
9.1 - As Sub-rotinas 145
9.2 - Tipos de Sub-rotina 145
9.3 - Utilização de Rotinas Embutidas: As Units 146

Capítulo 10 - Utilização de Procedures 151
10.1 - Aplicação Prática de Procedure 151
10.2 - Estrutura de Controle com Múltipla Escolha 155
10.3 - Variáveis Globais e Locais 157
10.3.1 - Escopo de Variáveis 158
10.4 - Refinamento Sucessivo 160
10.5 - Exercício 162

Capítulo 11 - Utilização de Parâmetros 163
11.1 - Parâmetros Formais e Reais 163
11.2 - Passagem de Parâmetros 164
11.2.1 - Por Valor 164
11.2.2 - Por Referência 165
11.3 - Passagem de Parâmetro em uma Sub-rotina 166
11.4 - Exercício 172

Capítulo 12 - Utilização de Function 173
12.1 - Aplicação Prática de Function 173
12.2 - Considerações a Respeito de Functions 174
12.3 - Exercícios 181

Capítulo 13 - Construir a Biblioteca de Sub-rotinas do Programador 183
13.1 - Sub-rotinas Dentro do Programa 183
13.2 - Sub-rotinas Fora do Programa 189

Parte 5 - Programação com Arquivos 195

Capítulo 14 - Arquivos 197
14.1 - Definição de um Arquivo 197
14.2 - Operações Executadas em um Arquivo 198
14.3 - Formas de Acesso em um Arquivo 199
14.3.1 - Arquivo de Acesso Seqüencial 199
14.3.2 - Arquivo de Acesso Direto 200
14.3.3 - Arquivo de Acesso Indexado 200

Capítulo 15 - Arquivos do Tipo Texto 201
15.1 - Criando um Arquivo 201
15.2 - Gravando Informações em um Arquivo 202
15.3 - Lendo Informações de um Arquivo 203
15.4 - Utilização de Arquivos do Tipo Texto 203
15.5 - Exercício 208

Capítulo 16 - Arquivos com Tipo Definido 209
16.1 - Criando um Arquivo 209
16.2 - Gravando Informações em um Arquivo 210
16.3 - Lendo Informações de um Arquivo 211
16.4 - Utilização de Arquivos com Tipo Definido 211
16.5 - Exercícios 213

Capítulo 17 - Arquivo com Tipo Definido de Registro 215
17.1 - Arquivo de Acesso Direto 215
17.2 - Utilização de Arquivo Direto 216
17.3 - Manipulação de Arquivo Direto 216
17.4 - Gerenciamento Escolar - Primeira Versão 217
17.5 - Gerenciamento Escolar - Segunda Versão 227

Parte 6 - Programação Avançada 237

Capítulo 18 - Variáveis Dinâmicas 239
18.1 - Conceito Básico 239
18.2 - O Tipo Pointer 240
18.3 - As Áreas de Memória 241
18.4 - Criar e Eliminar Variáveis Dinâmicas 242
18.4.1 - Procedimentos New/Dispose 242
18.4.2 - Procedimentos GetMem/FreeMem 248
18.5 - Eliminar Variáveis Dinâmicas - Mark/Release 250

Capítulo 19 - Alocação Dinâmica de Memória 253
19.1 - Estrutura Dinâmica de Dados Tipo Matriz 253
19.1.1 - Matriz com Uma Dimensão 253
19.1.2 - Matriz com Duas Dimensões 255
19.2 - Estrutura Dinâmica de Dados Tipo Lista 257
19.2.1 - Lista Ligada Homogênea 258
19.2.2 - Lista Ligada Heterogênea 263
19.3 - Exercício 268

Capítulo 20 - Utilização de Impressora 269
20.1 - Saída para a Impressora 269
20.2 - Códigos de Controle 270
20.3 - O Tamanho do Papel 272
20.4 - Posicionamento de Informação 274
20.5 - Impressão de Relatório 280

Capítulo 21 - Utilização de Datas 309
21.1 - Alguns Conceitos Importantes 309
21.1.1 - Dia Juliano 310
21.1.2 - Calendário Juliano 310
21.1.3 - Calendário Gregoriano 311
21.1.4 - Ano Bissexto 311
21.2 - Algoritmos 312
21.2.1 - Data em Dia Juliano 312
21.2.2 - Dia Juliano em Data 313
21.2.3 - Verificação de Ano Bissexto 315
21.3 - Biblioteca: Unidade Datas 316
21.3.1 - Conversão de Dia, Mês e Ano em Caractere 316
21.3.2 - Conversão de Dia, Mês e Ano em Número 317
21.3.3 - Verificação de Ano Bissexto 318
21.3.4 - Inversão do Formato Data 319
21.3.5 - Validações 320
21.3.6 - Conversão de Data e Dia Juliano 321
21.3.7 - Dia da Semana 323
21.3.8 - Entrada de Data e de Dia Juliano 325
21.3.9 - Unidade: Datas.tpu 326
21.4 - Relatório de Títulos Vencidos 333

Capítulo 22 - Apresentação Visual 337
22.1 - Utilização de Cores 337
22.2 - Utilização de Molduras e de Linhas Divisórias 341
22.2.1 - Moldura 342
22.2.2 - Linha Divisória Horizontal 346
22.3 - Menu com Barra Deslizante 351

Capítulo 23 - Programação em ANSI Pascal 355
23.1 - Programação em Modo ANSI 356
23.1.1 - O Driver ANSI.SYS 356
23.1.2 - Driver ANSI Alternativo 357

23.1.3 - Códigos de Controle ANSI 357
23.1.4 - Alguns Testes com Códigos ANSI 359
23.2 - Bibliotecas Genéricas 363
23.2.1 - Unidade CRTX 364
23.2.2 - Unidade DOSX 368
23.3 - Tratamento do Dispositivo de Impressão 369
23.4 - Tratamento do Cursor 371

Parte 7 - Apêndices 373

Apêndice A - Tabela ASCII 375

Apêndice B - Rotinas Comerciais 379
Cadastro de Pessoa Física 379
Programas de Integração Social 382
Cadastro Nacional da Pessoa Jurídica 385

Apêndice C - Compilar em Linha de Comando 393
Turbo Pascal 393
Free Pascal Compiler 394

Apêndice D - Compatibilidade entre Compiladores 397
Turbo Pascal 7.0 401
Free Pascal Compiler 401
QuickPascal 1.0 401
Virtual Pascal 2.1 401
Turbo Pascal 5.5 402
Irie Pascal 2.0 402
Conclusão 402

Apêndice E - Compiladores Freeware e Shareware 403

Índice Remissivo 405

Referências Bibliográficas 412

Parte 1

Conceitos Introdutórios

```
procedure CALC_ADICAO(A, B :
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Capítulo 1 Introdução

Esta obra foi escrita com a finalidade de introduzir o leitor nos conceitos de programação de computadores, fazendo uso da linguagem de programação Pascal.

Até a sétima edição a obra dava ênfase à distribuição Turbo Pascal 7.0 da empresa Borland. Porém, com o advento de computadores com processadores mais rápidos, o Turbo Pascal 7.0 "deixou de rodar" adequadamente, gerando erros de divisão por zero e de *runtime*. Apesar de a empresa Borland ter lançado um pequeno arquivo de correção (*upgrade*), este já não funciona adequadamente com a grande maioria dos computadores lançados a partir do ano 2000. Assim sendo, o compilador Turbo Pascal 7.0 funciona bem em computadores dotados de processadores 486 e alguns modelos de processadores Pentium, em torno dos 400 Mhz. No mercado mundial existem pessoas esperando o lançamento de uma nova versão do Turbo Pascal (talvez a versão 8.0) para computadores dotados de processadores mais rápidos.

Neste sentido, os autores desta obra decidiram adotar para o livro o uso do compilador **Free Pascal Compiler** a fim de evitar possíveis problemas de compatibilidade e conseguir atender assim a um público maior. Se o seu computador é baseado em processadores 286, 386, 485 ou Pentium até 400 Mhz, pode utilizar o Turbo Pascal 7.0, mas se o seu computador for de uma geração mais recente, é aconselhável a utilização do **Free Pascal Compiler**.

O programa **Free Pascal Compiler** é um compilador de linguagem Pascal bastante utilizado e roda muito bem em computadores com processadores acima do modelo Pentium de 400 Mhz, pois foi desenvolvido utilizando a tecnologia de 32 bits existente nos atuais processadores. O **Free Pascal Compiler** ou simplesmente **FPC** possui algumas vantagens estratégicas:

- ✓ O código compilado pelo **FPC** é o mesmo utilizado pelo Turbo Pascal 7.0 (poucas vezes ocorre alguma diferença, e quando elas existem são muito pequenas e de fácil alteração). Assim sendo, são compatíveis entre si e possuem um mecanismo de funcionamento bastante similar;
- ✓ Possui compatibilidade de código com algumas extensões utilizadas no ambiente de programação Delphi, além de ser distribuído em regime de *freeware* e possuir aberto o seu código-fonte.

O fato de o programa **FPC** ser distribuído livremente facilita sua aquisição, e por conseguinte evita que alunos, professores, instituições de ensino, empresas e usuários estejam lesando direitos autorais de uma empresa como a Borland, que merece todo o nosso respeito e consideração pela seriedade que sempre teve em relação à distribuição de ambientes de desenvolvimento para as mais diversas linguagens.

Para fazer o estudo desta obra, está sendo considerado que o leitor não possui conhecimento da sintaxe Pascal (seja ela manipulada no ambiente Turbo Pascal ou no ambiente FPC), mas tem familiaridade e possui sólidos conhecimentos de lógica de programação, pelo menos em nível básico.

Caso não seja esta a realidade do leitor, aconselha-se que faça um estudo do livro: Algoritmos - Lógica para o Desenvolvimento de Programação, da Editora Érica, para adquirir este conhecimento. Apesar deste requisito, cada técnica de programação Pascal abordada nesta obra é seguida de sua explicação técnica.

Vale salientar ainda que não é nosso objetivo fornecer neste livro um manual da linguagem Pascal, mas sim um trabalho que possibilite implementar as técnicas de programação básicas com esta linguagem Pascal.

1.1 - Breve Histórico da Linguagem Pascal

Tudo começa com os apontamentos de Niklaus Wirth[2] (figura 1.1), professor do Departamento de Informática da Escola Politécnica da Universidade de Zurique (Suíça), para ensinar seus alunos a programar nas linguagens PLI e ALGOL 60, principalmente, na década de 60. Foram esses apontamentos que deram origem à primeira notação da linguagem Pascal por volta de 1968 a 1970.

Figura 1.1 - Professor Niklaus Wirth.

[2] Niklaus Wirh é professor aposentado. Seus últimos projetos são as linguagens de programação Oberon e Lola. Possui vários artigos escritos e mais de quinze livros. Para maiores informações visite o site: www.cs.inf.ethz.ch/personal/wirth/

O Prof. Wirth procurou desenvolver uma forma de linguagem de programação que fosse de fácil assimilação e aceitação por parte dos alunos de programação, o que se concretizou. Essa linguagem foi batizada com o nome de Pascal, em homenagem a Blaise Pascal (figura 1.2), filósofo e matemático francês que viveu entre 1623 e 1662.

Figura 1.2 - Blaise Pascal.

Em 1970, é disponibilizado o primeiro compilador para a linguagem Pascal. Com o passar de alguns anos, a linguagem saiu dos muros acadêmicos e decolou no mercado mundial como uma poderosa linguagem de programação. Foi em novembro de 1983, que surgiu a primeira versão do Turbo Pascal, lançado pela Borland Internacional para microcomputadores de 8 bits, logo atualizado para a versão 2.0. Foi a partir da versão 2.0 que o Turbo Pascal passou a ser conhecido. Naquela época, a Borland era uma pequena e desconhecida *empresa de Scotts Valley, na Califórnia.*

A empresa Borland já lançou as versões 3.0 (1987), 4.0 (final de 1987), 5.0 (1988) e 5.5 (meados de 1989, implementação de recursos para Programação Orientada a Objetos - Object Pascal). Durante a década de 90, foram lançadas as versões 6.0, 7.0. Após a versão 7.0 do Turbo Pascal, a Borland lança o ambiente de desenvolvimento Delphi, que faz uso da linguagem Object Pascal para programação em ambiente Windows da Microsoft. A partir do ano 2000, a Borland inicia o projeto para lançar o ambiente Delphi também para o Sistema Operacional Linux, concretizando esta empreitada no segundo semestre de 2001 com o nome de Kylix.

De todas as linguagens existentes para microcomputadores, a linguagem Pascal é a que mais se destacou nos seus anos de existência. Durante os últimos anos, foram lançadas várias distribuições da linguagem Pascal para diversos tipos de sistema operacional, tais como: Apollo Domain Pascal, Berkeley Pascal, Bloodshed Dev-Pascal, Borland Pascal, Canterbury Pascal for Java, Compaq Pascal, Compas, DEC VAX Pascal, Dpas Pascal Compiler, FACILIS Pascal, Free Pascal Compiler, GNU Pascal, HP Pascal/64000, HP-UX Pascal (Hewlett Packard), Irie Pascal, Kyan Pascal, Microsoft Pascal, Microsoft QuickPascal, Oregon Software Pascal/2, P Pascal, P4 Pascal, Pascal Pro Compiler, Pascal XSC, PasOS2, Power Pascal,

Professional Pascal Compiler, Prospero Extended Pascal, Speed Pascal, Sun Pascal, SURPAS Pascal, SVS Pascal, Think Pascal (Macintosh), TI Pascal, TMT Pascal Lite, Turbo Pascal, UCSD Pascal (University of California at San Diego), VAX Pascal, Virtual Pascal, entre outras.

Dentre as várias distribuições concorrentes da Borland, o **FPC** é a mais ousada de todas elas, pois é desenvolvido por um grupo de pesquisadores que disponibiliza para acesso o código fonte do compilador, além de ambos os ambientes serem muito semelhantes. O **FPC** é um produto que muito se destaca na comunidade acadêmica. Está em uso desde meados da década de 90 e já foi matéria de revista especializada no Brasil.

Uma linguagem de programação não teria tantos distribuidores e interessados se fosse uma péssima linguagem. Este é um outro fator a ser também considerado e ponderado. Para saber um pouco mais sobre compiladores de linguagem Pascal *freeware* e *shareware,* consulte o apêndice D.

É fundamental que se faça um esclarecimento ao leitor com relação à forma como as pessoas mencionam a utilização de linguagens de programação. É muito comum ouvir que alguém programa nas linguagens: Free Pascal, Turbo Pascal, Turbo C, Delphi, Builder, Visual C, Visual Basic, etc. Na verdade, ninguém programa em Free Pascal, Turbo Pascal, Turbo C, Delphi, Builder, Visual C, Visual Basic, simplesmente pelo fato de essas linguagens não existirem.

O que existe, de fato, são as linguagens Pascal, C, Basic, (entre outras) as quais são integradas em ambientes de desenvolvimento. Por exemplo, Turbo Pascal é o ambiente de desenvolvimento Turbo (Borland) utilizado para facilitar o trabalho de um programador em linguagem Pascal. De forma análoga, existem os ambientes de desenvolvimento Visual (Microsoft) para as linguagens C e Basic Estruturado (Qbasic). Os ambientes de desenvolvimento Turbo Pascal e Delphi (Turbo Pascal for Windows) da empresa Borland são utilizados para programar em Pascal. O ambiente de desenvolvimento Turbo C e Builder (Borland), para a linguagem C e C++.

1.2 - NOTAÇÃO UTILIZADA

Neste trabalho, procurou-se manter ao máximo a notação oficial adotada nos manuais do ambiente de programação Turbo Pascal, porém algumas alterações se fizeram necessárias:

- ✓ Todas as variáveis utilizadas em um programa são escritas neste livro em caracteres maiúsculos e em itálico, e as instruções (comandos, procedures e functions) são escritas em caracteres minúsculos e em negrito.
- ✓ A definição de nomes de programas e sub-rotinas são escritas em caracteres maiúsculos, seguindo o mesmo critério de atribuição de nomes às variáveis.
- ✓ Na indicação das sintaxes, você encontrará alguns termos escritos entre os símbolos "<" e ">", os quais se caracterizam por informações obrigatórias que devem ser substituídas por algum elemento da linguagem Pascal.

- ✓ A informação que estiver mencionada entre os símbolos "[" e "]" caracteriza-se por uma informação opcional, a qual pode ser omitida para a utilização do recurso indicado.
- ✓ Todas as condições utilizadas nas instruções **if**, **while** e **repeat** estão definidas entre parênteses para melhor visualização.

1.3 - AQUISIÇÃO DO FREE PASCAL COMPILER

Devido ao fato de a versão 7.0 do compilador Turbo Pascal não estar mais à venda, muitas pessoas acabam por consegui-lo de forma ilegal. Existem basicamente duas formas legais de adquirir um compilador de linguagem de programação. A primeira é comprando o produto em um distribuidor de software, e a segunda fazendo um download (baixar o arquivo de algum site da Internet para o computador do usuário) de um compilador do tipo *freeware*, como é o caso do **FPC**.

Assim como o Sistema Operacional Linux é distribuído de forma livre (não confundir com distribuição gratuita), existem também vários outros produtos (*softwares*) distribuídos da mesma forma, e entre eles estão os compiladores de linguagens de programação de computador. Deste modo é possível conseguir um compilador de qualquer tipo de linguagem, e o mais importante, de forma legal, sem ferir o direito autoral de terceiros.

O **Free Pascal Compiler** é um projeto coordenado e desenvolvido pelo estudante de engenharia elétrica e pesquisador Florian Klämpfl, nascido em 1975 (figura 1.3), da Universidade de Erlangen-Nürnberg, Alemanha. Junto a Klämpfl estão associados outros estudantes e pesquisadores: Carl Eric Codère, Daniël Mantione, Jonas Maebe, Michael Van Canneyt, Peter Vreman, Pierre Muller e Marco van de Voort.

Figura 1.3 - Florian Klämpfl.

O ambiente **Free Pascal Compiler** é disponibilizado em versões para diversos sistemas operacionais: MS-DOS, Windows 95/98/Me/NT, Linux, OS/2, Solaris, FreeBSD, BeOS, QNX e

para o sistema Amiga. Esta obra está fazendo referência à versão disponível para MS-DOS (poderá também ser utilizada a versão para Windows).

Por ser o **FPC** um projeto acadêmico, esse compilador passa por rigorosos testes e sempre que possível é disponibilizada uma nova versão. Porém, não está isento de problemas (nem mesmo os produtos comerciais estão), mas possui um serviço de suporte técnico (apenas em inglês) que pode ser acessado por meio de seu site **www.freepascal.org** (opção **Community**).

Uma grande vantagem do **Free Pascal Compiler** e também um ponto que levou à sua escolha é o fato de o seu código-fonte ser aberto, disponibilizando ao leitor mais curioso uma rica fonte de informações (apesar de em nenhum momento esta obra fazer referência a isso). Por meio do código-fonte é possível observar e aprender como um compilador funciona, pois o compilador Free Pascal é escrito em linguagem Pascal com muito poucas rotinas escritas em *assembly*.

Para fazer o download (baixar) do compilador *Free Pascal*, é necessário acessar o endereço **www.freepascal.org** (site oficial em inglês) ou o endereço **www.freepascal.eti.br** (site oficial brasileiro autorizado).

Acesse o site **Free Pascal** (desejado) e após o carregamento dele, escolha no menu lateral à esquerda a opção **Download,** em seguida selecione um dos locais onde se encontra o compilador. Nesta etapa pode ser selecionado qualquer um dos sites relacionados. Sugere-se selecionar como padrão o primeiro *link* **Current site** se o acesso foi em inglês ou **Site local** se o acesso foi em português.

Observação: *Caso ocorra algum problema para baixar algum arquivo, sugere-se tentar baixá-lo por meio de um dos links sugeridos na opção* ***Links/Mirros*** *(tanto no site em inglês como no site em português).*

Após a seleção do *link*, é apresentada a relação de sistemas operacionais disponíveis. Neste momento selecione o link **Dos** e será exibida uma lista contendo os seguintes *links* de arquivo:

Arquivo	Descrição
dosxxxfull.zip	Esta opção caracteriza-se por disponibilizar para o DOS em modo 32 bits o compilador, os manuais, o código-fonte do compilador e todos os utilitários.
dosxxx.zip	Esta opção caracteriza-se por disponibilizar para o DOS em modo 32 bits apenas o compilador e os manuais. Ela não possui o código-fonte do compilador e os utilitários.
dosw32xxxfull.zip	Esta opção caracteriza-se por disponibilizar para o Windows o compilador, os manuais e todos os utilitários.

Para os objetivos desta obra, pode ser selecionado apenas o arquivo **dosxxx.zip**, por ser o de menor tamanho e por possuir o compilador de linguagem Pascal, tema desta obra. Mas pode também ser selecionado o arquivo **dosxxxfull.zip**. Selecione então o tipo de arquivo desejado para o ambiente MS-DOS.

Observação: *A versão Windows do compilador possui poucas diferenças em relação à versão DOS. Em tese são exatamente iguais. A diferença está no fato de que a versão para Windows deve ser utilizada para escrever programas para o ambiente Windows e não para ambiente DOS.*

1.4 - INSTALAÇÃO DO AMBIENTE PASCAL

A seguir, são apresentadas as instruções de instalação tanto do ambiente Turbo Pascal 7.0 como do ambiente Free Pascal Compiler.

1.4.1 - Instalação do Turbo Pascal

Para instalar o ambiente Turbo Pascal 7.0, é necessário ter em mãos o conjunto de disquetes disponibilizado no pacote, é composto por quatro unidades, porém serão instalados automaticamente apenas os dois primeiros discos. Para saber o que compõe cada disco, você pode consultar o arquivo FILELIST.DOC no diretório TP após o processo de instalação.

Para iniciar o processo de instalação, insira no drive de seu computador (A ou B) o primeiro disco **Install Disk**. Em seguida direcione o prompt do DOS para o drive a ser utilizado e digite **install** teclando **<Enter>** em seguida. Neste momento, será apresentada a tela de inicialização do processo de instalação, como mostra a figura 1.4. Desejando instalar, tecle **<Enter>**; caso deseje desistir da instalação, tecle **<Esc>**.

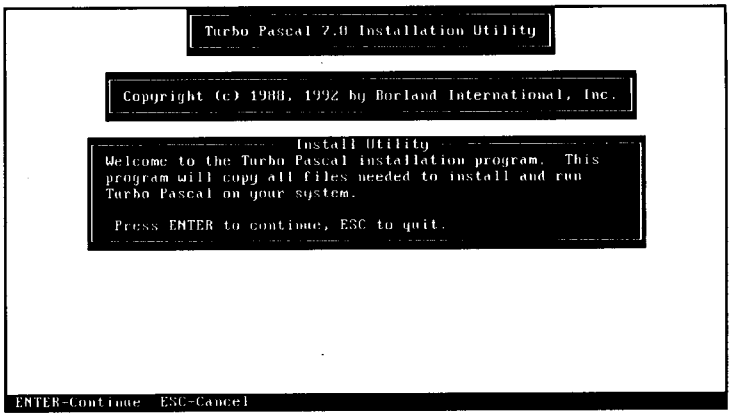


Figura 1.4 - Tela de inicialização do processo de instalação.

Caso seja teclado **<Enter>**, o programa solicita que se confirme o drive de origem, no caso o drive em que o disquete está inserido (figura 1.5). Geralmente basta teclar **<Enter>** para confirmar.

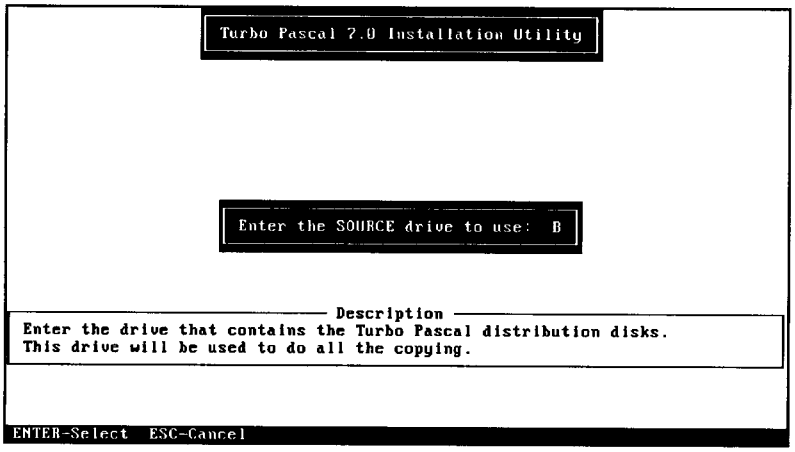


Figura 1.5 - Tela com a confirmação do drive de origem.

Após ter sido confirmado o drive de origem, é apresentado um menu solicitando a unidade em que será copiado o Turbo Pascal 7.0. Neste momento, mantenha selecionada a primeira opção, pois o programa será instalado no disco rígido, como indicado na figura 1.6.

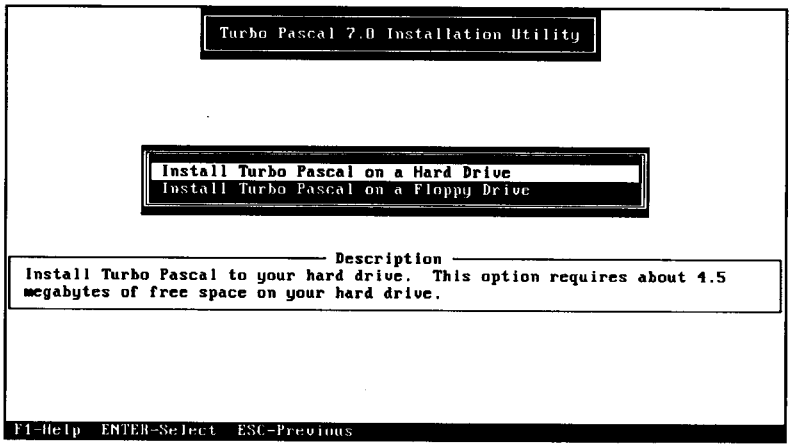


Figura 1.6 - Tela com a indicação da unidade de destino.

Após ter sido indicado o local de instalação, é apresentada uma outra tela, informando os diretórios que serão criados para instalar o programa. Desejando alterá-los, posicione a barra seletora sobre a opção **C:\TP** e tecle **<Enter>** para informar novo diretório. Neste caso, serão utilizados os diretórios sugeridos. Sendo assim, posicione a barra seletora na opção **Start Installation** e tecle **<Enter>** ou pressione a tecla de função **<F9>**. A figura 1.7 exibe esta ocorrência.

```
Turbo Pascal 7.0 Installation Utility

Turbo Pascal Base directory              : C:\TP
IDE/Command Line/Help directory          : C:\TP\BIN
Utilities directory                      : C:\TP\BIN
Units directory                          : C:\TP\UNITS
Source code directory                    : C:\TP\SOURCE
Borland Graphics Interface (BGI)         : C:\TP\BGI
Example programs directory               : C:\TP\EXAMPLES
Online documentation directory           : C:\TP\DOC

Start Installation

Description
Start the installation. This option will begin making subdirectories and
copying files to your hard drive.

F1-Help  F9-Start Installation  RETURN-Select  ESC-Previous
```

Figura 1.7 - Apresentação dos diretórios que serão criados para a instalação do programa.

Após serem confirmados os diretórios, o processo de instalação é iniciado, e são apresentados os arquivos que estão sendo copiados para o seu disco rígido (figura 1.8). Após alguns instantes é solicitada a inserção do disquete número dois, como indicado na figura 1.9. Neste momento retire o primeiro disco e insira o segundo, e em seguida pressione algo para o processo de instalação continuar.

```
Turbo Pascal 7.0 Installation Utility

Turbo Pascal Base directory              : C:\TP
IDE/Command Line/Help directory          : C:\TP\BIN
Utilities directory                      : C:\TP\BIN
Units directory                          : C:\TP\UNITS
Source code directory                    : C:\TP\SOURCE
Borland Graphics Interface (BGI)         : C:\TP\BGI
Example programs directory               : C:\TP\EXAMPLES
Online documentation directory           : C:\TP\DOC

    UNZIP.EXE
Writing files:
    C:\TP\UNZIP.EXE
Reading files:
    B:\README.COM, B:\FILELIST.DOC
Writing files:
    C:\TP\README.COM, C:\TP\FILELIST.DOC
Executing:
    C:\TP\UNZIP.EXE /o B:\SOURCE.ZIP C:\TP\SOURCE

ESC-Cancel
```

Figura 1.8 - Apresentação dos arquivos sendo instalados (copiados).

Ao final da instalação, é apresentado um aviso informando as alterações que são necessárias nos arquivos AUTOEXEC.BAT e CONFIG.SYS (figura 1.10). Neste ponto, o programa está instalado. Tecle algo para que sejam apresentadas as informações existentes no arquivo README, como indicado na figura 1.11.

```
Turbo Pascal 7.0 Installation Utility

Turbo Pascal Base directory                  : C:\TP
IDE/Command Line/Help directory              : C:\TP\BIN
Utilities directory                          : C:\TP\BIN
Units directory                              : C:\TP\UNITS
Source code director                         P\SOURCE
Borland Graphics Int                         P\BGI
Example programs dir                         P\EXAMPLES
Online documentation                         P\DOC

    Please insert disk
        Disk 2
    into drive B:
Press any key to continue

    C:\TP\UNZIP.EXE /o B
Executing:
    C:\TP\UNZIP.EXE /o B:\TPX.ZIP C:\TP\BIN
Executing:
    C:\TP\UNZIP.EXE /o B:\TURBO.ZIP C:\TP\BIN
Reading files:
    B:\README
Writing files:
    C:\TP\README

ESC Cancel
```

Figura 1.9 - Solicitação da inserção do segundo disco.

```
Turbo Pascal 7.0 Installation Utility

Turbo Pascal Base directory                  : C:\TP
IDE/Comm
Utilitie
Units di
Source c
Borland
Example
Online d

The Turbo Pascal 7.0 installation is now
complete.  Make sure the line
  FILES = 20
is in your CONFIG.SYS file, and C:\TP\BIN is in
your path.  For example, to run any of the Turbo
Pascal programs or utilities from anywhere on
your system, your AUTOEXEC.BAT file could
contain a path like the following:
  PATH=C:\DOS;C:\TP\BIN

   Press any key to view the README file.

    C:\TP\UM
Executing:
    C:\TP\UM
Executing:
    C:\TP\UM
Creating file:
    C:\TP\BIN\TPC.CFG
Creating file:
    C:\TP\BIN\THELP.CFG

Any Key Continue
```

Figura 1.10 - Informação das alterações necessárias para rodar o Turbo Pascal.

```
03-03-93 07:01 • C:\TP\README
                Welcome to Turbo Pascal 7.0
                ---------------------------

This README file contains important, last minute information
about Turbo Pascal 7.0. In addition to this file, there are
several files in the \TP\DOC directory that you should browse.
See FILELIST.DOC in the \TP directory for a complete list of all
files distributed with this package.

TABLE OF CONTENTS
-----------------
 1. How to get help
 2. Minimum system requirements
 3. Installation
 4. New features
 5. Release notes
    - Additional documentation
    - Demonstration and utility program notes
    - Compatibility and known problems
 6. Important note for Borland C++ and Paradox 4.0 users
 7. Turbo Pascal and the OS/2 Workplace Shell

Command▶                    Keys: ↑↓ • PgUp PgDn ESC=Exit F1=Help
```

Figura 1.11 - Apresentação do arquivo README com informações adicionais.

Biblioteca Silva Freire - UNIVAG
60631

Para interromper a apresentação das informações do arquivo README, tecle **<Esc>** e será então apresentado o prompt do DOS. Faça as alterações necessárias nos arquivos AUTOEXEC.BAT e CONFIG.SYS e dê um novo boot no seu microcomputador. Após a apresentação do prompt, digite **TURBO** para executar o Turbo Pascal 7.0. A figura 1.12 apresenta a tela do Turbo Pascal.

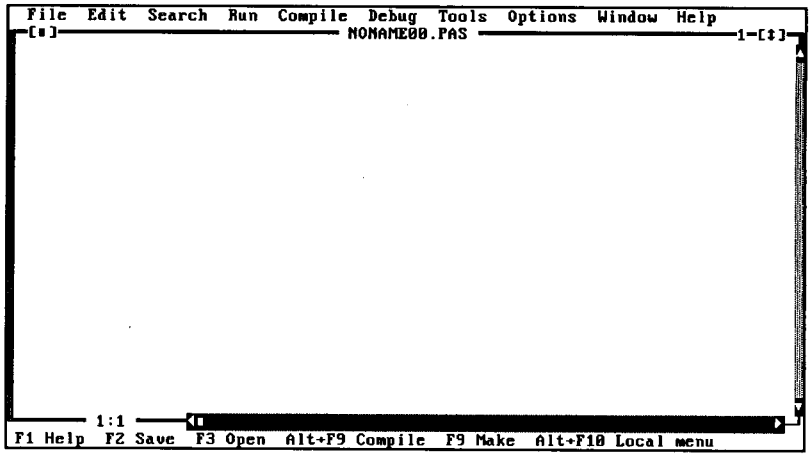


Figura 1.12 - Tela do programa Turbo Pascal.

1.4.2 - Instalação do Free Pascal

Após ter baixado o arquivo desejado, ele deve ser descompactado com um programa do tipo **BraZip** (www.brazip.com.br), **WinZip** (www.winzip.com) ou outro que possa descompactar o formato **.zip**. Recomenda-se o uso do **BraZip** por ser brasileiro, estar em português e conseguir trabalhar com mais de quinze formatos de compactação diferentes. No momento da execução da descompactação, torna-se necessário fornecer o local em que o processo ocorrerá. Sugere-se então descompactar o **FPC** no diretório **C:\FPC** (com o objetivo de padronizar o processo de instruções em seguida).

Após a descompactação, é necessário acessar o diretório **C:\FPC** (considera-se que esta ação será executada na linha de comando do *prompt* do MS-DOS). Execute então **CD\FPC** e tecle **<Enter>**. Em seguida execute o comando **Install** e tecle **<Enter>**. Neste momento, será apresentada a tela inicial do processo de instalação informando o local em que o compilador será instalado, como mostra a figura 1.13.

Normalmente não se altera nenhuma das opções. Neste sentido basta acionar com o ponteiro do *mouse* o botão **Continue** para que o processo de instalação seja iniciado. Serão instalados o compilador e os manuais[3] em formato **.htm** e **.pdf**.

---

[3] Para ter acesso aos manuais do programa, acesse o diretório **CD\PP\DOC** no qual são encontrados o arquivo **FAQ**, o arquivo **FPCTOC** (manual em HTML) e os manuais em formato **PDF** (prog, ref, units e user).

Universitá
Biblioteca

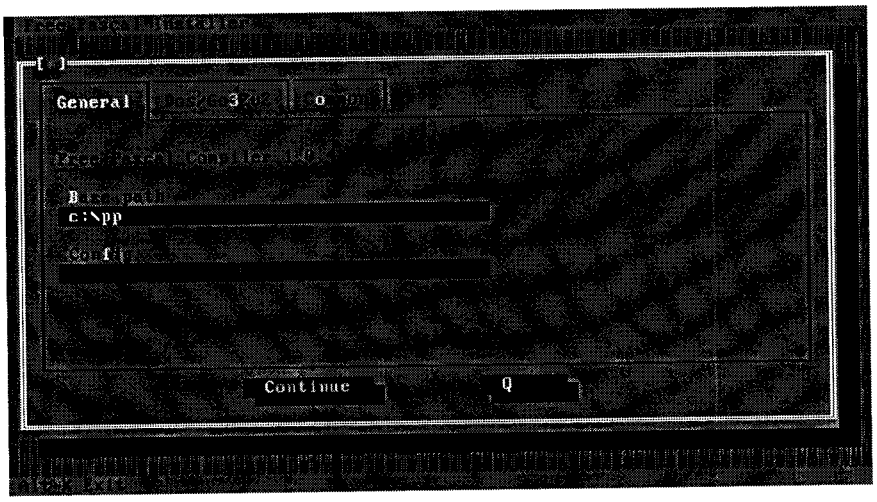


Figura 1.13 - Tela de inicialização do processo de instalação.

Ao término do processo será apresentada uma tela com uma caixa de mensagem **Installation Successfull** (figura 1.14) informando que o comando **PATH** do arquivo AUTOEXEC.BAT deve ser alterado com o caminho do compilador (para o comando ter efeito, é necessário executar o arquivo AUTOEXEC.BAT), e para rodar o programa por meio de sua IDE, basta executar no *prompt* do MS-DOS o comando **FP**.

Dependendo da versão do Windows em uso, o arquivo AUTOEXEC.BAT não será alterado automaticamente. Neste caso o processo de configuração deve ser efetuado manualmente a partir do disco de inicialização do sistema.

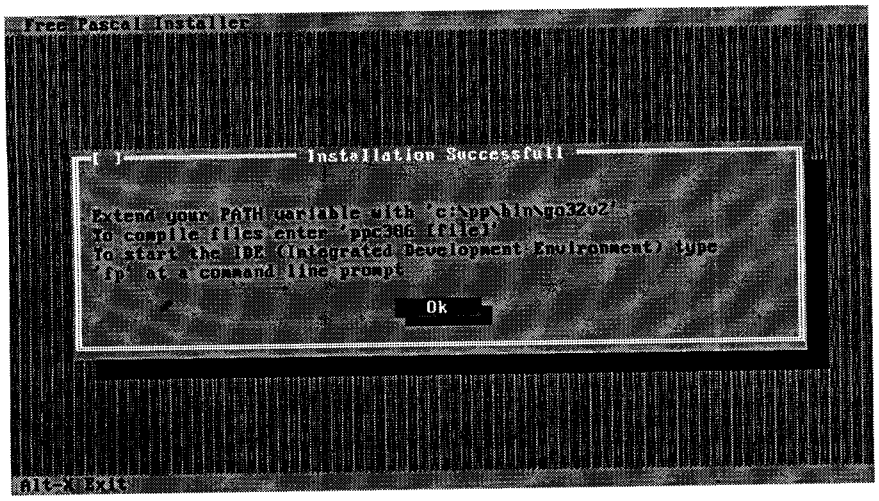


Figura 1.14 - Tela do término da instalação.

Para encerrar o processo de instalação, acione com o ponteiro do mouse o botão **OK** ou tecle **<Enter>**. Após a apresentação do *prompt*, digite o endereço de diretório **CD\PP\BIN**

GO32V2 e em seguida digite **FP** para executar o ambiente de edição de programas IDE do Free Pascal Compiler. A figura 1.15 apresenta a tela do Free Pascal.

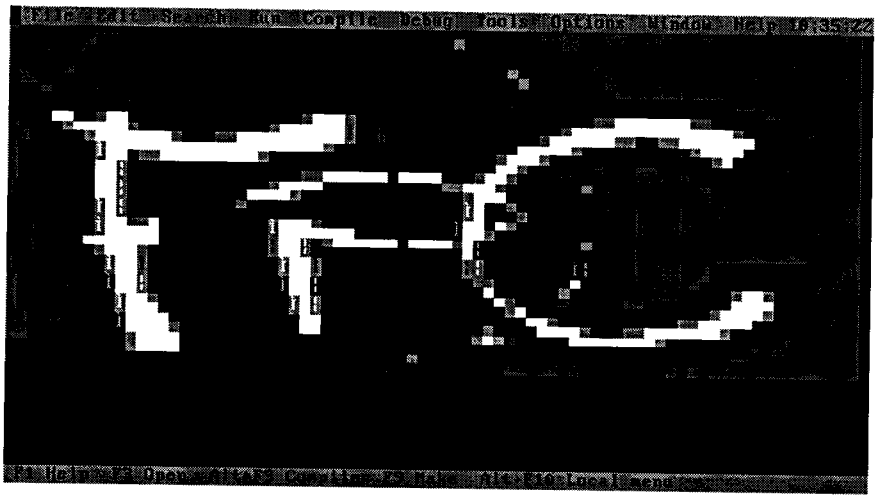


Figura 1.15 - Tela do programa Free Pascal.

1.5 - Considerações Operacionais

O programa **TURBO** do Turbo Pascal 7.0 e o programa **FP** do Free Pascal caracterizam-se por serem as IDEs de seus respectivos compiladores. IDE (Integrated Development Environment - Ambiente Integrado de Desenvolvimento) é a ferramenta responsável por integrar em uma única interface um editor de texto com a capacidade de executar os comandos de compilação.

Tanto o **TURBO** quanto o **FP** são ferramentas de acesso aos seus respectivos compiladores. Visualmente são muito semelhantes, mas operacionalmente possuem algumas diferenças. As diferenças existentes entre elas não prejudicam o trabalho com ambos os compiladores se assim o leitor desejar, pois os comandos básicos são os mesmos e estão nos mesmos lugares.

Capítulo 2 O Ambiente de Trabalho

Os ambientes de trabalho dos compiladores Turbo Pascal 7.0 e Free Pascal Compiler integram os recursos de um compilador com um editor de textos, selecionando opções por intermédio de um menu. O objetivo deste capítulo é apresentar ao leitor, de forma básica, como se encontram organizadas as opções de acesso ao menu de cada ambiente.

2.1 - O Menu de Comandos do Turbo Pascal 7.0

O Turbo Pascal 7.0 possui um menu de comandos com dez opções que possibilitam executar diversas tarefas operacionais. O acesso ao menu pode ser conseguido de três formas diferentes:

1. A primeira forma pode ser conseguida com o pressionamento da tecla de função **<F10>** e em seguida usando as teclas de setas para movimentar o cursor sobre as opções desejadas, teclando **<Enter>** sobre a opção escolhida.
2. A segunda pode ser com o pressionamento da tecla **<Alt>** + a letra que estiver grafada em maiúsculo, que é a primeira letra de cada opção do menu. Para acionar o comando de um determinado menu, basta pressionar a letra que se encontra grafada na cor vermelha (ambiente padrão).
3. A terceira forma é com a utilização de um mouse, cujo ponteiro deve ser posicionado sobre a opção desejada e em seguida ser dado um clique.

Para sair do menu ou de qualquer caixa de diálogo que venha a ser acionada, basta pressionar a tecla **<Esc>**. O menu do Turbo Pascal 7.0 apresenta os seguintes comandos:

File

Esta opção possibilita executar operações de controle com arquivos. Desta forma, é possível: Criar uma nova janela de trabalho (**New**), Abrir um programa existente (**Open**), Salvar um programa em disco (**Save**), Salvar um programa em disco com outro nome (**Save as**), Mudar o diretório de trabalho (**Change dir**), Imprimir o arquivo da janela ativa (**Print**),

Configurar o uso de outra impressora (**Print setup**), Sair temporariamente para o sistema operacional (**DOS shell**), Finalizar a execução do Turbo Pascal (**Exit**).

Edit

Esta opção possibilita executar operações do editor do programa, sendo possível remover, movimentar e copiar textos que estejam selecionados. Desta forma, é possível: Desfazer (**Undo**) e Refazer (**Redo**) operações efetuadas com a edição, Remover um texto previamente selecionado (**Cut**), Copiar um texto selecionado do editor para uma área de transferência (**Copy**), Copiar um texto da área de transferência para o editor (**Paste**), Remover o texto selecionado sem transferi-lo para a área de transferência (**Clear**), Apresentar o conteúdo existente na área de transferência (**Show clipboard**).

Search

Esta opção executa operações de busca, troca e deslocamento dentro de um programa. Desta forma, é possível: Localizar uma seqüência de caracteres em um programa (**Find**), Substituir uma seqüência de caracteres por outra (**Replace**), Repetir a última busca (**Search again**), Posicionar-se em uma determinada linha do programa (**Go to line number**), Mostrar o último erro de compilação, quando ocorrer (**Show last compiler error**), Posicionar-se na última posição de erro encontrada pelo compilador (**Find error**), Localizar uma sub-rotina dentro do programa no momento de depuração de um programa (**Find procedure**).

Run

Esta opção coloca em execução o programa da janela ativa. Desta forma, é possível: Rodar todo o programa (**Run**), Rodar o programa passo a passo com exceção das sub-rotinas existentes (**Step over**), Rodar o programa passo a passo inclusive as suas sub-rotinas (**Trace into**), Rodar o programa até a posição em que está o cursor (**Go to cursor**), Interromper a execução de um programa durante sua depuração (**Program reset**), Efetuar a passagem de parâmetros (**Parameters**).

Compile

Esta opção compila o programa. Desta forma, é possível: Compilar o programa da janela ativa (**Compile**), Recompilar apenas os programas alterados (**Make**), Recompilar todos os programas (**Build**), Determinar se o programa será compilado somente em memória ou disco (**Destination**), Determinar numa lista de arquivo qual será o arquivo principal que será carregado primeiro no processo de compilação (**Primary file**), Obter informações a respeito da última compilação executada (**Information**).

Debug

Esta opção possibilita depurar o programa, para facilitar a localização de erros lógicos. Desta forma, é possível: Colocar ou retirar um ponto de parada (exame) quando da depuração de um programa (**Breakpoint**), Apresentar uma janela com a seqüência de chamadas efetuadas de sub-rotinas (**Call stack**), Visualizar a janela de registradores da CPU (**Register**), Abrir a janela de acompanhamento de valores nas variáveis do programa (**Watch**), Abrir uma janela para apresentar as telas de saída do programa em execução ou depuração (**Output**), Exibir na sua totalidade a tela do usuário (**User screen**), Efetuar a avaliação de expressões, constantes ou variáveis (**Evaluate/modify**), Incluir expressões na tela de vigia (**Add watch**), Incluir um ponto de parada quando for executado o programa (**Add breakpoint**).

Tools

Esta opção possibilita a utilização de ferramentas configuradas pelo usuário. Desta forma, é possível: Abrir uma janela para a apresentação de mensagens (**Messagens**), Visualizar a próxima mensagem (**Go to next**) ou a mensagem anterior (**Go to previous**) da janela de mensagens, Efetuar a busca de seqüências de caracteres em programas gravados por meio do utilitário GREP.EXE (**Grep**).

Options

Esta opção permite configurar a forma de trabalho do ambiente do Turbo Pascal. Desta forma, é possível: Alterar o estado das diretivas de compilação (**Compile**), Definir o tamanho de memória (**Memory sizes**), Usar ou não o recurso de link (**Linker**), Estabelecer critérios de depuração (**Debugger**), Determinar os diretórios de trabalho (**Directories**), Efetuar a manutenção do menu de ferramentas (**Tools**), Efetuar mudanças no ambiente de trabalho (Preferências, Editor, Mouse, Inicialização e Cores) do Turbo Pascal conforme a necessidade do usuário (**Environment**), Abrir o arquivo de configuração (**Open**) e sua gravação (**Save** ou **Save as**).

Window

Este comando possibilita o controle das janelas que estejam abertas. Desta forma, é possível: Ordenar as janelas lado a lado (**Tile**) ou em cascata (**Cascade**), Fechar todas as janelas abertas (**Close all**), Redesenhar a janela ativa (**Refresh display**), Movimentar ou alterar o tamanho de uma janela (**Size/Move**), Alterar o tamanho de uma janela para o máximo ou para o tamanho preestabelecido (**Zoon**), Selecionar a próxima janela como ativa (**Next**) ou a janela anterior (**Previous**), Fechar a janela ativa (**Close**), Apresentar a listagem das janelas que estejam abertas (**List**).

Help

Este comando permite executar o modo de ajuda do Turbo Pascal de qualquer parte do programa com a tecla de função **<F1>** ou **<Ctrl>** + **<F1>** para visualizar explicações de instruções que estejam marcadas com o posicionamento do cursor sobre elas. Pesquisas podem ser feitas utilizando o sumário (**Contents**), por meio da listagem alfabética de todas as instruções (**Index**), entre outras formas.

2.2 - O MENU DE COMANDOS DO FREE PASCAL COMPILER

O Free Pascal possui um menu de comandos com dez opções que possibilitam executar diversas tarefas operacionais. O acesso ao menu pode ser conseguido de três formas diferentes:

1. A primeira forma pode ser com o pressionamento da tecla de função **<F10>** e em seguida usando as teclas de setas para movimentar o cursor sobre as opções desejadas, teclando **<Enter>** sobre a opção escolhida.
2. A segunda pode ser com o pressionamento da tecla **<Alt>** + a letra que estiver grafada em maiúsculo, que é a primeira letra de cada opção do menu. Para acionar o comando de um determinado menu, basta pressionar a letra que se encontra grafada na cor vermelha (ambiente padrão).
3. A terceira forma pode ser com a utilização de um mouse, cujo ponteiro deve ser posicionado sobre a opção desejada e em seguida ser dado um clique.

Para sair do menu ou de qualquer caixa de diálogo que venha a ser acionada, basta pressionar a tecla **<Esc>**. O menu do Free Pascal Compiler apresenta os seguintes comandos:

File

Esta opção possibilita executar operações de controle com arquivos. Desta forma, é possível: Criar uma nova janela de trabalho (**New**), Criar uma aplicação por meio de máscara de programa predefinida (**New from template**), Abrir um programa existente (**Open**), Salvar um programa em disco (**Save**), Salvar um programa em disco com outro nome (**Save as**), Mudar o diretório de trabalho (**Change dir**), Sair temporariamente para o sistema operacional (**Command shell**), Finalizar a execução do programa (**Exit**).

Edit

Esta opção executa operações do editor do programa, sendo possível remover, movimentar e copiar textos que estejam selecionados. Desta forma, é possível: Desfazer (**Undo**) e Refazer (**Redo**) operações efetuadas com a edição, Remover um texto previamente selecionado (**Cut**), Copiar um texto selecionado do editor para uma área de transferência (**Copy**), Copiar um texto da área de transferência para o editor (**Paste**), Remover o texto

selecionado sem transferi-lo para a área de transferência (**Clear**), Apresentar o conteúdo existente na área de transferência (**Show clipboard**).

Search

Esta opção executa operações de busca, troca e deslocamento dentro de um programa. Desta forma, é possível: Localizar uma seqüência de caracteres em um programa (**Find**), Substituir uma seqüência de caracteres por outra (**Replace**), Repetir a última busca (**Search again**), Posicionar-se em uma determinada linha do programa (**Go to line number**), Localizar uma sub-rotina dentro do programa no momento de depuração de um programa (**Find procedure**).

Run

Esta opção possibilita colocar em execução o programa da janela ativa. Desta forma, é possível: Rodar todo o programa (**Run**), Rodar o programa passo a passo com exceção das sub-rotinas existentes (**Step over**), Rodar o programa passo a passo inclusive as suas sub--rotinas (**Trace into**), Rodar o programa até a posição em que está o cursor (**Goto cursor**), Interromper a execução de um programa durante sua depuração (**Program reset**), Efetuar a passagem de parâmetros (**Parameters**).

Compile

Esta opção compila o programa. Desta forma, é possível: Compilar o programa da janela ativa (**Compile**), Recompilar apenas os programas alterados (**Make**), Recompilar todos os programas (**Build**), Determinar se o programa será compilado somente em memória ou disco (**Destination**), Determinar o padrão de compilação para um sistema operacional específico (**Target**), Determinar numa lista de arquivos o principal que será carregado primeiro no processo de compilação (**Primary file**), Obter informações a respeito da última compilação executada (**Information**).

Debug

Esta opção depura o programa, para facilitar a localização de erros lógicos. Desta forma, é possível: Abrir uma janela para apresentar as telas de saída do programa em execução ou depuração (**Output**), Exibir na sua totalidade a tela do usuário (**User screen**), Colocar ou retirar um ponto de parada (exame) quando da depuração de um programa (**Breakpoint**), Apresentar uma janela com a seqüência de chamadas efetuadas de sub-rotinas (**Call stack**), Visualizar a janela de registradores da CPU (**Register**), Incluir expressões na tela de vigia (**Add watch**), Abrir a janela de acompanhamento de valores nas variáveis do programa (**Watches**), Editar os pontos de parada (**Breakpoint List**).

Centro Universitário - UNIVAS Biblioteca Silva Freire

Tools

Esta opção possibilita a utilização de ferramentas configuradas pelo usuário. Desta forma, é possível: Abrir uma janela para a apresentação de mensagens (**Messagens**), Visualizar a próxima mensagem (**Goto next**) ou a mensagem anterior (**Goto previous**) da janela de mensagens, Efetuar a busca de seqüências de caracteres em programas gravados por meio do utilitário GREP.EXE (**Grep**), Acessar a calculadora (**Calculator**), Acessar a tabela ASCII (**Ascii table**).

Options

Esta opção permite configurar a forma de trabalho do ambiente do Turbo Pascal. Desta forma, é possível: Selecionar o modo de execução do programa (**Mode**), Alterar o estado das diretivas de compilação (**Compiler**), Definir o tamanho de memória (**Memory sizes**), Usar ou não o recurso de link (**Linker**), Estabelecer critérios de depuração (**Debugger**), Determinar os diretórios de trabalho (**Directories**), Especificar configurações no modo de navegação do ambiente (**Browser**), Efetuar a manutenção do menu de ferramentas (**Tools**), Efetuar mudanças no ambiente de trabalho (Preferências, Editor, Mouse, Inicialização e Cores) do Turbo Pascal conforme a necessidade do usuário (**Environment**), Abrir o arquivo de configuração (**Open**) e sua gravação (**Save** ou **Save as**).

Window

Este comando controla as janelas que estejam abertas. Desta forma, é possível: Ordenar as janelas lado a lado (**Tile**) ou em cascata (**Cascade**), Fechar todas as janelas abertas (**Close all**), Movimentar ou alterar o tamanho de uma janela (**Size/Move**), Alterar o tamanho de uma janela para o máximo ou para o tamanho preestabelecido (**Zoon**), Selecionar a próxima janela como ativa (**Next**) ou a janela anterior (**Previous**), Esconder as janelas ativas (**Hide**), Fechar a janela ativa (**Close**), Apresentar a listagem das janelas que estejam abertas (**List**), Redesenhar a janela ativa (**Refresh display**).

Help

Este comando permite executar o modo de ajuda do Turbo Pascal de qualquer parte do programa com a tecla de função **<F1>** ou **<Ctrl>** + **<F1>**. Pesquisas podem ser feitas utilizando o sumário (**Contents**), por meio da listagem alfabética de todas as instruções (**Index**).

2.3 - Como Operar os Ambientes TP7 e FPC

Este tópico tem por objetivo dar uma idéia ao programador iniciante do funcionamento do ambiente IDE do Turbo Pascal e do Free Pascal. Assim sendo, são dadas instruções

básicas de como operar o editor de textos[4]. Desta forma, serão enfatizados três comandos do menu, em particular, **File**, **Edit** e **Window**.

Ao ser carregado o editor de texto do Turbo Pascal 7.0 (TURBO) pela primeira vez, ele apresentará a janela **NONAME00.PAS** aberta, como mostra a figura 2.1. Com relação ao editor de texto do Free Pascal (FP), ele mostrará primeiro a tela de apresentação; execute em seguida o comando de menu **File/New** e será apresentada a janela **noname01.pas**, como mostra a figura 2.2.

Figura 2.1 - Tela do editor de texto do Turbo Pascal.

---

4 Tenha cuidado para não confundir editor de texto com processador de texto. É muito comum ouvirmos as pessoas chamarem de editores de texto produtos como: MS-Word, AMI Pro, WordPerfect, entre outros. Na verdade, esses produtos são processadores de texto, pois além de efetuarem as operações básicas de edição, também fazem a formatação de textos e parágrafos. Coisas que os simples editores não conseguem fazer. Um editor possibilita apenas escrever o texto sem formatação, gravá-lo apenas em formato ASCII (American Standard Code for Information Interchange - Código Americano Padrão para Troca de Informações - lê-se asqui-i e não asqui 2, ver Apêndice A), efetuar cópia e remoção de partes do texto, efetuar busca e troca de caracteres, efetuar localização de caracteres e em algumas vezes, dependendo do programa, desfazer sempre a última operação executada. No caso do Turbo Pascal, o que temos é um editor de textos utilizado para a digitação do programa e sua execução, a qual pode ser feita por meio da linha de comando do DOS (Prompt).

Figura 2.2 - Tela do editor de texto do Free Pascal.

Em primeiro lugar, você aprenderá a usar os recursos de edição existentes no teclado. Para tanto, digite o trecho de linhas seguinte exatamente como está escrito em qualquer um dos editores, teclando **<Enter>** sempre ao final de cada linha.

```
A programacao de computadores consiste em primeiro lugar entender
adequadamente o plobrema propostp.


Em seguida, adotar criterios e metodos de algoritimizacao
para entao codificar o programa na linguagem desejada.

                                                    Augusto Manzano
```

As palavras anteriores escritas de forma incorreta e sem acento devem ser reproduzidas como estão grafadas. Os erros com troca de letras ou com elas fora de posição são propositais e serão o motivo para efetuar a correção depois de escrito. As figuras 2.3 e 2.4 mostram respectivamente o texto digitado nos editores TURBO e FP. Procure deixar o texto escrito no seu computador com a mesma aparência das figuras apresentadas.

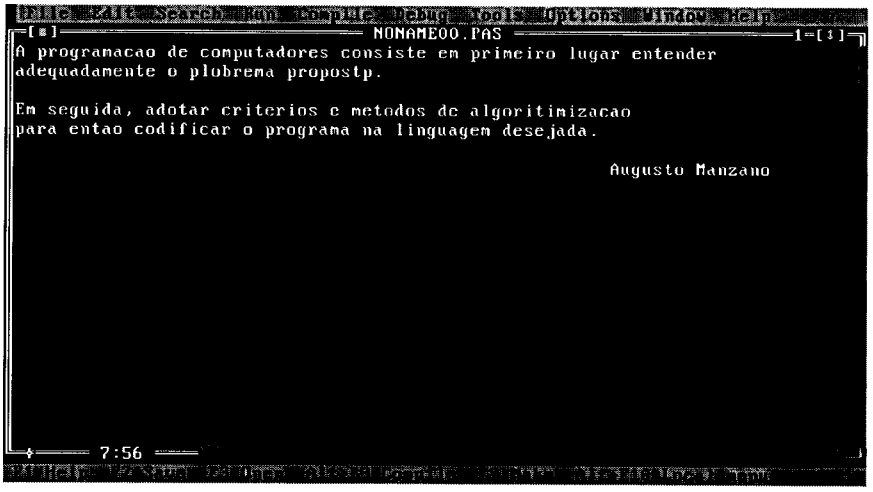


Figura 2.3 - Tela com o texto digitado no Editor TURBO.

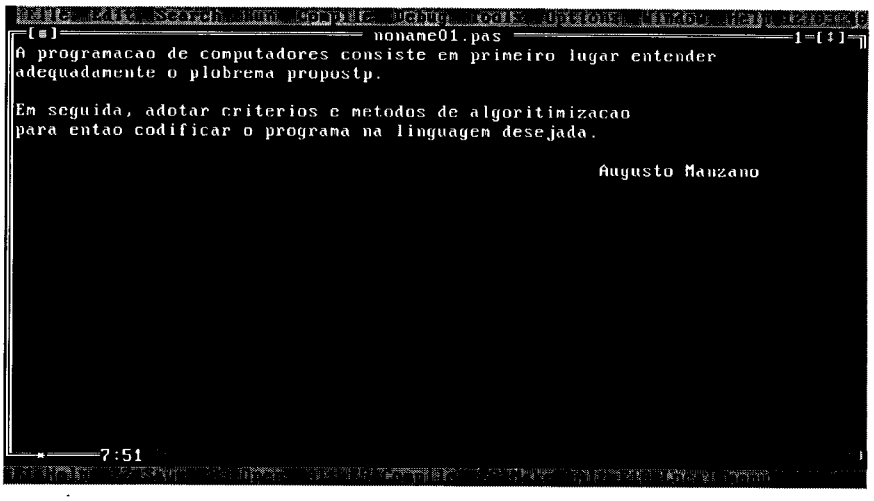


Figura 2.4 - Tela com o texto digitado no Editor FP.

Para movimentar o cursor na tela, você deve fazer uso das teclas de setas de movimentação. Desta forma, posicione o cursor na segunda linha de texto, abaixo da letra **l** da palavra **plobrema**. Em seguida digite a letra **r** que será inserida entre a letra **p** e a letra **l**, ficando a palavra escrita como: **prlobrema**. Neste momento, o cursor estará abaixo da letra **l**, a qual pode ser removida com a utilização da tecla **<Delete>** ou **<Del>**. Após remover a letra **l**, a palavra fica escrita como: **probrema**.

Você pode fazer da mesma forma a troca da letra **r** por **l** da palavra "problema", porém o fato de inserir uma letra e ter que remover a letra grafada errada pode se tornar trabalhoso, caso haja um número de erros grande. Uma forma de minimizar este trabalho é inserir a letra correta sobre a outra letra. Para tanto, aperte a tecla **<Insert>** ou **<Ins>**. O cursor ficará

grande, então posicione-o sobre a letra **r** da palavra **probrema** e em seguida digite a letra **l**. Note que agora você terá a palavra escrita corretamente como: **problema**.

Em seguida substitua a letra **p** da palavra **propostp** pela letra **o**, de forma que fique escrita como **proposto**. Depois é a vez de remover a letra **i** que se encontra entre a letra **t** e a letra **m** da palavra **algoritimizacao**. Assim sendo, a palavra deve ficar **algoritmizacao**. As figuras 2.5 e 2.6 mostram os textos escritos sem erros, respectivamente para os editores TURBO e FP.

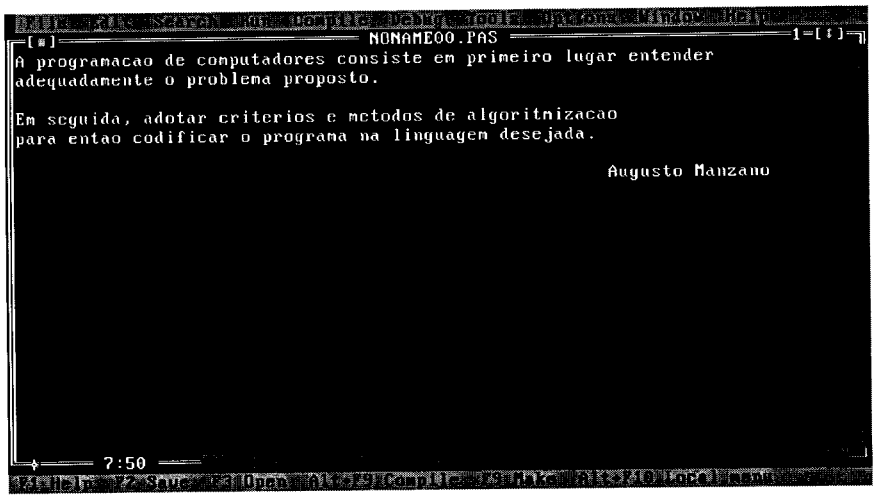


Figura 2.5 - Tela com o texto escrito de forma correta no Editor TURBO.

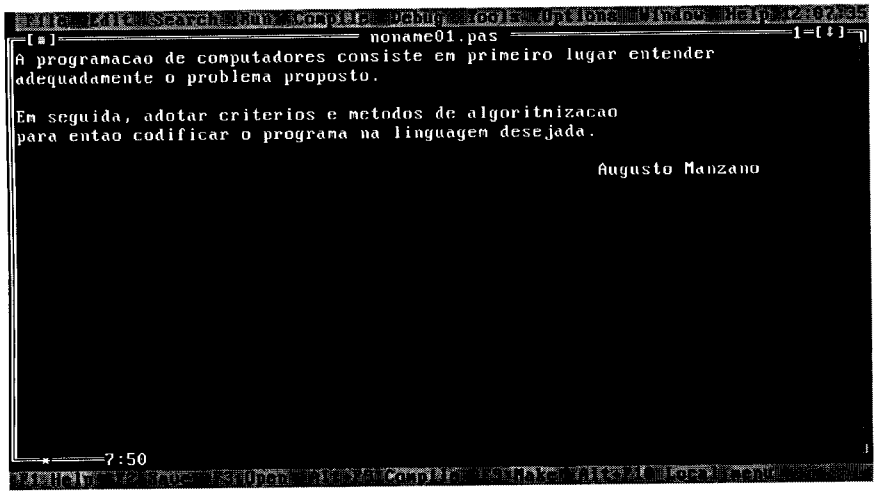


Figura 2.6 - Tela com o texto escrito de forma correta no Editor FP.

A partir deste momento que o texto encontra-se corrigido, é apropriado efetuar a sua gravação. Para tanto, execute o comando **File/Save** com o mouse, ou pelo teclado com o

conjunto de teclas **<Alt>** + **<F>** e em seguida teclando **<S>**, ou ainda com a execução da tecla de função **<F2>** para ambos os editores.

Será então apresentada a caixa de diálogo **Save File As.** Digite o nome **TEXTO1** no campo **Save file as** do editor TURBO ou digite **TEXTO1** no campo **Name** do editor FP. Em seguida tecle **<Enter>** ou com o mouse selecione o botão **OK**. Observe que as indicações **NONAME00** (editor TURBO) ou **noname01** (editor FP) foram substituídas pelo nome de arquivo informado no momento da gravação.

Em seguida, após gravar o texto, você aprenderá a fazer uso do comando **Edit**. Para tanto, posicione o cursor na primeira letra do primeiro parágrafo, no caso na letra **A**, em seguida, pressione a tecla **<Shift>** e fique segurando. Simultaneamente acione por cinco vezes a tecla com o desenho da seta para baixo. Desta forma, o texto ficará selecionado.

Execute em seguida o comando **Edit/Copy** para copiar o conteúdo do texto para a área de transferência. Depois posicione o cursor após a última linha digitada, tecle **<Enter>** para inserir uma linha em branco e em seguida execute o comando **Edit/Paste**. Note que o texto selecionado e copiado foi duplicado na posição em que o cursor estava.

Este recurso é bastante útil quando se necessita repetir trechos iguais dentro de um programa, ou mesmo aproveitar parte de um programa em outros programas. Se você utilizar o comando **Edit/Cut** no lugar de **Edit/Paste,** fará a retirada do texto selecionado, transferindo-o para a área de transferência.

Após esta alteração, é importante fazer a gravação do arquivo. Em vez de executar o comando **File/Save**, você pode pressionar a tecla de função **<F2>** que surtirá o mesmo efeito.

Em seguida, após a gravação, será fechada a janela atual. Para tanto, execute o comando **Window/Close**. Observe que a janela de trabalho será retirada do editor, ficando a tela com a apresentação do menu e da linha de *status*.

Para abrir novamente o arquivo **TEXTO1.PAS**, execute o comando **File/Open** e na caixa de diálogo **Open a File** selecione o arquivo desejado e tecle **<Enter>** ou dê um clique no botão **OK**.

Nos ambientes Turbo Pascal e Free Pascal, é possível fazer uso de teclas de atalho, as quais visam um manuseio mais rápido dos comandos mais utilizados. Na documentação do Turbo Pascal, essas teclas são referenciadas com o nome **Hot Keys** e são acionadas pelo conjunto de duas ou três teclas que devem ser selecionadas simultaneamente, como foi descrito no processo de salvamento.

Após abrir o arquivo **TEXTO1.PAS**, efetue a execução do comando **File/New**, e será apresentada uma segunda janela de trabalho com o nome **NONAME00.PAS** (no editor TURBO) e com o nome **noname02.pas** (no editor FP). Perceba que em ambos os ambientes a segunda janela sobrepôs a primeira. Para visualizar a primeira janela, execute o comando **Window/List** (em ambos os ambientes) que apresentará a caixa de diálogo **Window List**, selecione a janela **TEXTO1.PAS** e tecle **<Enter>** ou clique no botão **OK**.

A partir do momento em que se sabe selecionar uma das janelas abertas, você pode visualizar as duas ao mesmo tempo. Para tanto, execute o comando **Window/Tile** (em ambos os ambientes), para a tela ser dividida em duas partes.

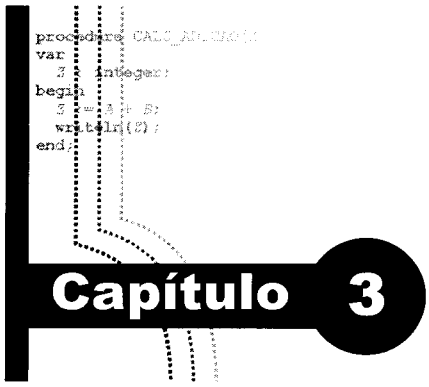
Tendo duas janelas, você pode transferir informações de uma para a outra, fazendo uso dos comandos **Edit/Copy** e **Edit/Paste**. Para ficar experiente nesta operação, nada melhor do que treinar. Faça por conta própria alguns exercícios, transferindo partes de um texto de uma janela para a outra e vice-versa.

Parte 2

Programação Básica

```
procedure CALC_ADICAO(A,
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Centro Universitário - UNIVAG
Biblioteca
Silva Freire



Tipos de Dados e Instruções Primitivas

Para desenvolver um programa que seja executável dentro de um computador, é necessário ter em mente que o trabalho de gerenciamento de dados ocorre em três níveis:

1. entrada dos dados;
2. processamento dos dados;
3. saída de dados.

Este trabalho ocorre por meio da execução de instruções de comando, que são ordenadas de forma lógica para executar uma determinada ação.

3.1 - Tipos de Dados

Os **dados** são representados pelas informações a serem processadas por um computador. A linguagem Pascal fornece ao programador um conjunto de tipos de dados predefinidos, podendo ser: numéricos (inteiros e reais), caracteres e lógicos.

3.1.1 - Tipos de Dados Inteiros

São caracterizados por tipos inteiros os dados numéricos positivos ou negativos, excluindo qualquer número fracionário. Em Pascal, este tipo de dado pode ser referenciado por um dos seguintes identificadores:

Tipo de dado inteiro	Faixa de abrangência
shortint	de -128 até 127
integer	de -32.768 a 32.767
longint	de -2.147.483.648 até 2.147.483.647
byte	de 0 até 255
word	de 0 até 65535

Nos programas deste livro, serão utilizados os tipos de identificadores **integer** e **byte**.

3.1.2 - Tipos de Dados Reais

São caracterizados por tipos reais os dados numéricos positivos, negativos e números fracionários e também os inteiros. Em Pascal, este tipo de dado pode ser referenciado por um dos seguintes identificadores:

Tipo de dado real	Faixa de abrangência
real	de 2,9 E-39 até 1,7 E+38
single	de 1,5 E-45 até 3,4 E+38
double	de 5,0 E-324 até 1,7 E+308
extended	de 3,4 E-4.932 até 1,1 E+4.932
comp	de -9,2 E+18 até 9,2 E+18

Este tipo de dado permite trabalhar com uma representação de valores em ponto flutuante que consiste em uma mantissa (parte fracionária). O tipo **single** usa uma mantissa de 7 a 8 dígitos, o tipo **real** usa uma mantissa de 11 dígitos, o tipo **double**, uma mantissa de 15 a 16 dígitos e os tipos **extended** e **comp** utilizam uma mantissa de 19 a 20 dígitos. Nos programas deste livro, será utilizado o tipo de identificador **real**.

3.1.3 - Tipos de Dados Caracteres

São caracterizados por tipos caracteres as seqüências contendo letras, números e símbolos especiais. Uma seqüência de caracteres deve ser representada na linguagem Pascal entre apóstrofos (' '). Este tipo de dado é referenciado pelo identificador **string**, podendo armazenar de 1 até 255 caracteres. Existe ainda o tipo **char**, utilizado da mesma forma que o tipo **string**, porém com uma pequena diferença, que é usado para strings de apenas um caractere.

3.1.4 - Tipos de Dados Lógicos

São caracterizados por tipos lógicos os dados com valores **true** (verdadeiro) e **false** (falso), sendo que este tipo de dado somente pode representar um dos dois valores. Ele é chamado por alguns de tipo booleano. Este tipo de dado é referenciado pelo identificador **boolean**. Mais adiante, você apreenderá como trabalhar com o tipo lógico de dado.

3.2 - O Uso de Variáveis

Variável é, no sentido de programação, uma região de memória de um computador, previamente identificada, que tem por finalidade armazenar as informações (dados) de um programa temporariamente. Uma variável armazena apenas um valor por vez. Sendo considerado como valor o conteúdo de uma variável, um valor está relacionado ao tipo de dado de uma variável, podendo ser um valor numérico, lógico ou caractere.

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

- ✓ Nomes de uma variável podem ser atribuídos com um ou mais caracteres;
- ✓ O primeiro caractere do nome de uma variável não pode ser, em hipótese alguma, um número; sempre deve ser uma letra;
- ✓ O nome de uma variável não pode possuir espaços em branco;
- ✓ Não pode ser nome de uma variável uma palavra reservada a uma instrução ou identificador da linguagem Pascal;
- ✓ Não podem ser utilizados outros caracteres, a não ser letras e números, com exceção do caractere underscore "_", que pode ser utilizado para simular a separação de duas palavras, como: NOME_ALUNO.

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, sendo que ela pode ser "vigiada" e controlada durante a execução de um programa (este tipo de variável será estudado nos capítulos que abordam a tomada de decisões e o uso de loopings).

Todo dado a ser armazenado na memória de um computador por meio de uma variável, utilizando a linguagem Pascal, deve ser previamente declarado, ou seja, primeiro é necessário saber qual o seu tipo para depois fazer o seu armazenamento. Estando armazenado o dado, ele pode ser utilizado e manipulado a qualquer momento, durante a execução do programa.

3.3 - 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.

3.4 - Os Operadores Aritméticos

Tanto variáveis como constantes podem ser utilizadas na elaboração de cálculos matemáticos com 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, radiciação, multiplicação, divisão, adição e subtração. São unários quando atuam na inversão de um valor, atribuindo a este o sinal positivo ou negativo. A tabela seguinte apresenta um resumo dos operadores aritméticos utilizados na linguagem Pascal:

Operador	Operação	Tipo	Resultado
+	Manutenção de sinal	Unário	Positivo
-	Inversão de sinal	Unário	Negativo
Exp(Ln(x)*n)	Exponenciação: x^n	Binário	Real
Sqrt(x)	Raiz quadrada	Unário	Real
Exp(Ln(x)*(1/n))	Raiz de índice qualquer: $\sqrt[n]{x}$	Binário	Real
Div	Divisão	Binário	Inteiro
Mod	Resto de divisão	Binário	Inteiro
/	Divisão	Binário	Real
*	Multiplicação	Binário	Inteiro ou Real
+	Adição	Binário	Inteiro ou Real
-	Subtração	Binário	Inteiro ou Real

3.5 - As Expressões Aritméticas

Será muito comum trabalharmos com expressões aritméticas, uma vez que, na sua maioria, o trabalho computacional está relacionado com a utilização de cálculos. Essas expressões são definidas pelo relacionamento existente entre variáveis e constantes numéricas por meio da utilização dos operadores aritméticos. Considere a fórmula: $AREA = \pi \cdot RAIO^2$ para o cálculo da área de uma circunferência, em que estão presentes as variáveis *AREA* e *RAIO*, a constante π (pi = 3.14159) e os operadores aritméticos de multiplicação, e também a operação de potência, elevando o valor da variável *RAIO* ao quadrado.

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

O sinal implicado ou atribuído (:=) é utilizado para indicar que o valor de uma expressão aritmética ou fórmula matemática está sendo armazenado em uma variável. No caso da fórmula para o cálculo da área de uma circunferência, seria escrita em Pascal da seguinte forma: AREA := 3.14159 * (RAIO * RAIO) ou AREA := 3.14159 * Exp (ln(Raio) * 2).

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 em seguida a fórmula padrão:

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

A fórmula deve ser escrita assim: ***AREA*** := (***BASE*** * ***ALTURA***) / 2.

3.6 - INSTRUÇÕES BÁSICAS

As **instruções** são representadas pelo conjunto de *palavras-chave* (vocabulário) de uma linguagem de programação que tem por finalidade comandar, em um computador, o seu funcionamento e a forma como os dados armazenados devem ser tratados.

3.6.1 - Algumas Regras Antes de Começar

Anteriormente, você aprendeu o significado de uma variável e também teve contato com algumas regras para sua utilização, porém devemos ter algum cuidado quando estivermos fazendo referência a uma instrução ou a uma variável. Desta forma, nesta obra, ter-se-á mais algumas regras, apresentadas em seguida:

- ✓ Toda referência feita a uma instrução da linguagem Pascal será escrita em letra minúscula em negrito;
- ✓ Toda referência feita a uma variável será escrita em letra maiúscula em formato itálico.

3.6.2 - A Entrada, Processamento e Saída de Dados

Uma entrada e uma saída podem ocorrer dentro de um computador de diversas formas. Por exemplo, uma entrada pode ser feita via teclado, modem, leitores óticos, disco, entre outros. Uma saída pode ser feita em vídeo, impressora, disco, entre outras formas. Devido a esta grande variedade, nossos programas utilizarão as instruções **readln()** (para a entrada de dados) e **writeln()** (para a saída de dados), considerando que toda entrada será via teclado e saída, via vídeo.

3.7 - ESTRUTURA DE UM PROGRAMA EM PASCAL

Todo programa escrito em Pascal é subdividido em três áreas distintas, a saber: cabeçalho do programa, área de declarações e corpo do programa.

3.7.1 - Cabeçalho do Programa

Essa área é utilizada para fazer a identificação do programa com um nome, sendo que para sua formação, ele segue as mesmas regras utilizadas para nomear as variáveis. Um detalhe a ser observado é que não pode existir nenhuma variável no programa que seja atribuída com o mesmo nome dado ao programa. Por exemplo, você atribuiu ao programa o nome SOMA e também atribuiu este mesmo nome a uma variável no programa. Quando for

solicitada a execução do programa, ele apresentará uma mensagem informando a ocorrência. No caso, será apresentada a mensagem: **Error 4: Duplicate identifier (soma).**

O cabeçalho de um programa é atribuído pela instrução **program** seguida de um nome, e ao final do nome, deve-se colocar o símbolo (;) ponto-e-vírgula, como exemplificado em seguida:

```
program CALCULA_AREA;
```

3.7.2 - Área de Declarações

Essa área é utilizada para validar o uso de qualquer tipo de identificador que não seja predefinido, estando subdividida em sete subáreas, a saber:

```
uses

label

const

type

var

procedure

function
```

Inicialmente, interessa-nos apenas a subárea **var**, pois cada uma possui o seu modo de ser referenciada. A subárea **var** é utilizada na declaração das variáveis que serão utilizadas durante a execução de um programa, bem como, também o seu tipo. Desta forma, a linguagem Pascal efetua a reserva de espaço de trabalho na memória para que as variáveis sejam operadas.

A declaração das variáveis é atribuída pela instrução **var** seguida da relação de variáveis. Após os nomes de cada variável, deve ser utilizado o símbolo (:) dois pontos, e após é mencionado o tipo de dado que a variável irá receber, seguido de ponto-e-vírgula, conforme o exemplo seguinte:

```
var
  NOME : string;
  IDADE : integer;
  ALTURA : real;
```

Caso sejam as variáveis de mesmo tipo, elas podem ser relacionadas separadas por vírgula, conforme indicado em seguida:

```
var
  A, B, C : integer;
```

3.7.3 - Corpo do Programa

O programa propriamente dito, em Pascal, está escrito na área denominada corpo do programa. Essa área tem início com a instrução **begin** e é finalizada pela instrução **end** seguida do símbolo (.) ponto final. O uso destas instruções caracteriza o que é chamado de bloco, como indicado em seguida:

```
begin
   <instruções>;
   (...)
end.
```

Na área denominada corpo do programa, podem existir, dependendo da necessidade, vários blocos.

3.7.4 - Diretório de Trabalho

Antes de iniciar o estudo prático desta obra, sugere-se criar um diretório de trabalho para que os programas sejam gravados em um único lugar reservado para esta finalidade. Não é conveniente gravar os programas desenvolvidos juntamente com os arquivos do compilador.

Para tanto, crie a partir da raiz do disco rígido em uso o diretório de trabalho denominado **ESTUDO**. Em seguida, execute o comando **File/Change dir** (existente em ambos os ambientes de programação - Turbo Pascal e Free Pascal), digite no campo **Directory name** a informação **C:\ESTUDO** e acione o botão **OK**.

Todas as vezes que ocorrer a entrada no ambiente de programação Turbo Pascal ou Free Pascal será necessário executar o comando **File/Change dir** para selecionar o diretório de trabalho.

3.8 - O Primeiro Programa em Pascal

Para colocar em prática o exposto até este momento, considere o seguinte exemplo de um problema:

"Deve ser desenvolvido um programa que efetue a leitura de dois valores numéricos. Faça a operação de adição de dois valores e apresente o resultado obtido".

Note que sempre estaremos diante de um problema que deve 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 deve "ensinar" a máquina a resolvê-lo 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 da leitura dos dois valores (os quais não são e não necessitam ser conhecidos e que serão representados por duas variáveis para trabalhar estas

incógnitas A e B) e a sua respectiva adição (conseqüência dos valores informados, a qual também é uma incógnita e depende dos valores fornecidos, e será representada pela variável X), o programa em questão deve:

Algoritmo

1. Ler um valor para a variável A;
2. Ler outro valor para a variável B;
3. Efetuar a adição das variáveis A e B, implicando o seu resultado na variável X;
4. Apresentar o valor da variável X após a operação de adição dos dois valores fornecidos.

Observe que tem-se neste exemplo as três fases de trabalho de um programa: a fase de entrada retratada nos passos 1 e 2, o processamento retratado no passo 3 e por fim a saída retratada no passo 4.

Completada a fase de interpretação do problema e da definição das variáveis a serem utilizadas, por meio de um algoritmo, passa-se para a fase de diagramação. Não é o intuito deste trabalho ensinar a confecção de diagramas de blocos, sendo assim será passado para a codificação em linguagem Pascal, uma vez que está sendo considerado que o leitor já possui este conhecimento, tendo sua atenção vinculada apenas ao aprendizado da sintaxe da linguagem.

Tendo estabelecido os passos anteriores, será efetuada a codificação do programa. Desta forma, são utilizados no exemplo três variáveis: A, B e X, sendo que devem ser declaradas segundo o seu tipo. Observe a utilização da instrução **program** com o nome do programa ADICIONA_NUMEROS seguido de um ponto-e-vírgula indicando o final da linha, formando assim o cabeçalho do programa:

```
program ADICIONA_NUMEROS;
var
  X : integer;
  A : integer;
  B : integer;
```

Tendo relacionado as variáveis que serão utilizadas no programa com a instrução **var**, passa-se para a fase de montagem do programa propriamente dito. Observe que o bloco de instruções entre **begin** e **end** está 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 identação.

```
begin
  readln(A);
  readln(B);
  X := A + B;
  writeln(X);
end.
```

Após a leitura dos valores para as variáveis A e B, eles serão adicionados e implicados na variável X, a qual será apresentada com o valor da adição processada. Veja que a instrução

begin não é seguida de ponto-e-vírgula, pois ela abre um bloco de instruções que deve ser encerrado com a instrução **end** que, quando seguida de ponto final, indica o fim do programa. A seguir, é apresentado o programa completo:

```
program ADICIONA_NUMEROS;
var
  X : integer;
  A : integer;
  B : integer;
begin
  readln(A);
  readln(B);
  X := A + B;
  writeln(X);
end.
```

Daqui para frente o processo de montagem de um programa fica mais fácil, bastando que sejam seguidas as dicas anteriores. As figuras 3.1 e 3.2 mostram respectivamente como deve ficar a apresentação do programa digitado nos editores de texto do Turbo Pascal e do Free Pascal.

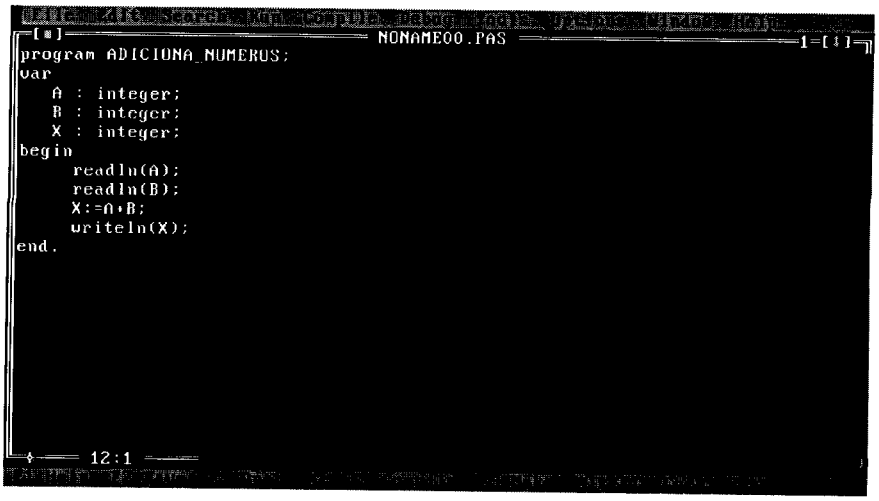


Figura 3.1 - Apresentação de um programa digitado no Turbo Pascal.

```
[■]═══════════ noname01.pas ═══════════1=[↕]═╗
program ADICIONA NUMEROS;
var
   A : integer;
   B : integer;
   X : integer;
begin
     readln(A);
     readln(B);
     X:=A+B;
     writeln(X);
end.
═══*═══12:1 -
```

Figura 3.2 - Apresentação de um programa digitado no Free Pascal.

> ***Observação:*** *No editor de textos, as instruções serão escritas em caracteres minúsculos, enquanto as variáveis e nome do programa serão escritos com caracteres maiúsculos. Note que isto não é uma obrigação; trata-se apenas de uma padronização.*

Para executar o programa desenvolvido, é necessário estar com o compilador Pascal carregado na memória do computador, com o programa digitado e devidamente gravado[5].

Para gravar o programa em operação, pressione as teclas **<Alt>** + **<F>** e em seguida pressione a letra **<S>** ou selecione os comandos no menu **File/Save**, será então apresentada uma caixa de diálogo na qual deve ser fornecido o nome para gravar no campo **Save file as** (para o Turbo Pascal) ou no campo **Name** (para o Free Pascal).

Digite **ADICAO** e tecle **<Enter>**. Para executar o processo de salvamento, pode também ser pressionada a tecla **<F2>**, como informado na linha de status do editor, antes de acionar o menu do programa.

Depois pressione as teclas **<Alt>** + **<R>** e tecle a letra **<R>** ou pressione as teclas **<Ctrl>** + **<F9>**. Possuindo um mouse, basta dar um clique no comando **Run** da barra de menu e em seguida dar outro clique na opção **Run**. Pode ainda acionar a tecla de função **<F10>** e com as setas do teclado selecionar o comando **Run** no menu, ou acionar a tecla **R**. As figuras 3.3 e 3.4 mostram respectivamente o selecionamento do comando de execução por meio do menu para o Turbo Pascal e para o Free Pascal.

---

5 No caso do ambiente Turbo Pascal o programa digitado pode ser executado sem ser previamente gravado.

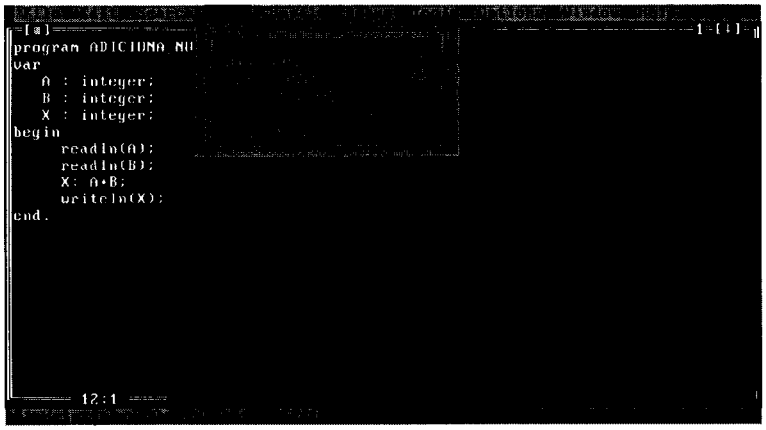


Figura 3.3 - Execução do comando Run/Run no Turbo Pascal.

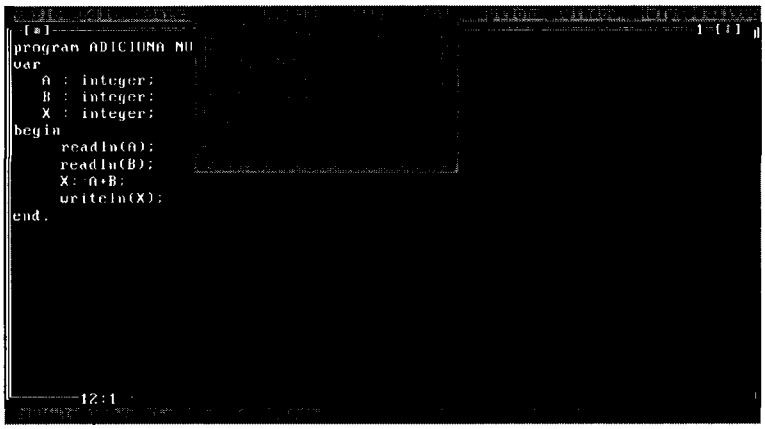


Figura 3.4 - Execução do comando Run/Run no Free Pascal.

Observação: *Caso na tela do editor não seja apresentada a janela para digitação do programa, você deve abri-la executando no menu os comandos* ***File/New.***

Aparecerá uma tela em branco com um cursor piscando ao lado esquerdo superior. Digite um valor inteiro qualquer e em seguida tecle **<Enter>**. O cursor será posicionado na próxima linha. Digite então um segundo valor teclando **<Enter>**, e será apresentado em seguida o resultado da adição dos dois valores fornecidos.

Apesar de ter sido apresentado o resultado da adição, você não conseguiu visualizá-lo, pois após a execução do programa ADICIONA_NUMEROS, o processamento retornou ao editor de textos do Pascal. Neste momento, para visualizar o resultado obtido, pressione as teclas **<Alt> + <F5>**. Para voltar ao editor, pressione qualquer tecla.

De volta novamente ao editor de textos do compilador Pascal, pressione as teclas **<Alt> + <W>** e em seguida a letra **<C>** para fechar o programa atual, ou então execute os comandos do menu **Window/Close.** Pode também ser utilizado o conjunto de teclas **<Alt> + <F3>**.

Como um segundo exemplo, será desenvolvido um programa que efetuará o cálculo do salário líquido de um profissional que trabalha por hora. Para fazer este programa, você deve possuir alguns dados, tais como: valor da hora de trabalho, número de horas trabalhadas no mês e o percentual de desconto do INSS. O programa em questão deve apresentar o valor do salário bruto, o valor descontado e o valor do salário líquido. Para tanto, pressione as teclas **<ALT>** + **<F>** e em seguida pressione a letra **<N>** para abrir uma nova janela de trabalho, ou execute os comandos do menu **File/New**. Depois de digitar o programa, grave-o com o nome **SALARIO**.

Algoritmo

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

Programa em Pascal

```
program SALARIO;
var
  HT, VH, PD, TD, SB, SL : real;
begin
  readln(HT);
  readln(VH);
  readln(PD);
  SB := HT * VH;
  TD := (PD/100) * SB;
  SL := SB - TD;
  writeln(SB);
  writeln(TD);
  writeln(SL);
end.
```

Ao ser executado este programa, possivelmente você ficará um pouco perdido, não sabendo exatamente o momento adequado de digitar um determinado valor (fato também acontecido com o primeiro exemplo). Isto ocorre, pois o programa não possui nenhuma indicação para quem está digitando os dados de entrada, podendo ser melhorado com mensagens que façam o programa "conversar" com o usuário.

Outro detalhe a ser observado é o fato da utilização de tipos de dados reais. Se durante a execução do programa forem fornecidos os valores 10, 100 e 5, respectivamente, para as variáveis HT, VH e PD, estes irão resultar na saída os valores 1.0000000000E+03, 5.0000000000E+01 e 9.5000000000E+02, respectivamente, para as variáveis SB, TD e SL.

Desta forma, a leitura destes valores torna-se um pouco difícil, pois eles estão impressos em notação científica e desta forma necessitam ser convertidos na forma decimal.

Para saber o valor decimal correspondente de um número indicado em notação científica, é necessário multiplicá-lo por dez e elevá-lo ao expoente indicado. Desta forma, o valor 9.5000000000E+02 em notação científica corresponde a 950 em decimal, pois 9.5 foi multiplicado por 10 elevado à potência 2, ou seja, 9.5 x 10^2. Existe uma outra forma de apresentar este tipo de valor em Pascal, conforme exemplificado em seguida:

```
program SALARIO;
var
  HT, VH, PD, TD, SB, SL : real;
begin
  write('Quantas horas de trabalho? '); readln(HT);
  write('Qual o valor da hora? '); readln(VH);
  write('Qual o percentual de desconto? '); readln(PD);
  SB := HT * VH;
  TD := (PD/100) * SB;
  SL := SB - TD;
  writeln('Salario bruto ....: ', SB:7:2);
  writeln('Desconto .........: ', TD:7:2);
  writeln('Salario liquido ..: ', SL:7:2);
end.
```

Faça as alterações propostas e execute novamente o programa. Observe a melhora que estas pequenas alterações proporcionaram. Em seguida pressione as teclas **<Alt>** + **<F>** e selecione a letra **<A>** ou selecione no menu **File/Save as**, para gravar o programa com outro nome, no caso SALARIO2. Execute o programa e observe a sua nova forma visual e também o formato dos números na saída dos dados.

Observação: *As instruções **writeln** e **readln** também podem ser escritas sem as letras **ln**, desta forma seria válido também escrever **write** e **read**. A colocação de **ln** indica **line new**, ou seja, nova linha. Em particular, as instruções **read** devem ser mencionadas com as letras **ln**, pois isto evita alguns pequenos problemas de entrada de dados. No exemplo, as instruções **write** utilizadas permitem manter o cursor na mesma linha após a exibição das mensagens, mas seria interessante que você fizesse um teste, colocando as letras **ln** nas instruções **write** para sentir o efeito visual que esta alteração ocasiona.*

Para tornar a visualização da saída dos dados numéricos do tipo real mais adequada, foram utilizados na instrução **writeln** dois valores separados com dois pontos, em vez de multiplicar o valor em notação científica por 10 e elevando à potência indicada, como mostrado em seguida:

```
writeln('Salario bruto ....: ', SB:7:2);
writeln('Desconto .........: ', TD:7:2);
writeln('Salario liquido ..: ', SL:7:2);
```

O primeiro valor, após a variável, corresponde ao tamanho total de posições de um número real, e o segundo valor corresponde ao número de casas após o ponto (mantissa) mais

o ponto que será subtraído do tamanho total. O valor 7 na linha que apresentará o valor do salário bruto indica que é possível apresentar de forma tabulada valores numéricos até 9999.99. Após este valor, a apresentação ocorre de forma desalinhada.

Por exemplo, se você deseja digitar valores para o salário bruto em torno de 12000, é necessário indicar a saída como sendo: writeln('Salario bruto .....: ', SB:8:2. Das oito posições reservadas, duas posições à direita serão usadas para tabular a apresentação dos valores decimais. As nove posições restantes serão usadas para apresentar o ponto à esquerda dos valores decimais e os oito valores pertencentes à porção inteira do valor.

3.9 - EXERCÍCIO

Desenvolva a codificação dos programas apresentados em seguida. Não se esqueça de ir gravando cada programa. Como sugestão você pode gravar o exercício "a" como EXERC03A, o exercício "b" como EXERC03B, e assim por diante.

a. Ler quatro números e apresentar o resultado dois a dois da adição e multiplicação, baseando-se na utilização da propriedade distributiva, ou seja, se forem lidas as variáveis A, B, C e D, devem ser somadas e multiplicadas A com B, A com C e A com D; B com C, B com D; e por fim C com D. Perceba que ao final deverão ser apresentados 12 resultados: 6 adições e 6 multiplicações.

b. Efetuar o cálculo da quantidade de litros de combustível gasta em uma viagem, utilizando um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto e a velocidade média durante a viagem. Desta forma, será possível obter a distância percorrida com a fórmula: DISTANCIA = TEMPO * VELOCIDADE. Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula: LITROS = DISTANCIA / 12. O programa deve apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem.

c. Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é: F = (9 * C + 160) / 5, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

d. Ler uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius. A fórmula de conversão é: C = (F - 32) * (5 / 9), sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

e. Calcular e apresentar o valor do volume de uma lata de óleo, utilizando a fórmula: V = 3.14159 * R * R * A, em que a variável V representa o valor de volume, a variável R o valor do raio da circunferência e a variável A o valor da altura.

f. Ler dois valores para as variáveis A e B, 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 das variáveis A e B após o processamento da troca.

```
procedure CALC_ADICAO(A, B : ...
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Capítulo 4 — A Tomada de Decisões

Até este momento, foi visto 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 em alguns momentos, um determinado valor dentro de um programa necessitará ser tratado para que seja efetuado um processamento mais adequado.

Imagine 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 o aluno está aprovado ou reprovado segundo a análise de sua média. Perceba que neste ponto, é necessário verificar o valor da média do aluno para então tomar uma decisão no sentido de apresentar uma mensagem indicando: aprovado ou reprovado. A variável que conterá o valor da média passa a ser a de controle, apesar de essa mesma variável ser de ação, uma vez que o valor da média é variável.

4.1 - Desvio Condicional Simples

Para solucionar o problema proposto, é necessário trabalhar uma nova instrução **if...then** que tem por finalidade tomar uma decisão e efetuar um desvio no processamento, dependendo, é claro, da condição atribuída ser *Verdadeira* ou *Falsa*.

Sendo a condição *Verdadeira*, será executada a instrução que estiver escrita após a instrução **if...then.** Caso seja necessário executar mais de uma instrução para uma condição verdadeira, elas devem estar mencionadas dentro de um bloco. Lembre-se que um bloco é definido com o uso das instruções **begin** e **end**.

Sendo a condição Falsa, serão executadas as instruções que estejam após a instrução considerada verdadeira *ou as instruções após o bloco considerado verdadeiro. Desta forma* a instrução **if...then** pode ser escrita:

```
if <(condição)> then
  <instrução para condição verdadeira>;
<instrução para condição falsa ou após condição ser verdadeira>;
```

Caso venha a existir mais de uma instrução verdadeira para uma determinada condição, estas, como já mencionado, devem estar inseridas em um bloco, como indicado abaixo. Observe que é considerado um bloco o conjunto de instruções entre um **begin** e um **end**.

```
if <(condição)> then
  begin
    <instrução1 para condição verdadeira>;
    <instrução2 para condição verdadeira>;
    <instrução3 para condição verdadeira>;
    <instruçãoN para condição verdadeira>;
  end;
<instrução para condição falsa ou após condição ser verdadeira>;
```

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

Algoritmo

1. Conhecer dois valores incógnitos (estabelecer variáveis A e B);
2. Efetuar a adição 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 seja esse valor maior que 7.

Programa em Pascal

```
program ADICIONA_NUMEROS;
var
  X, A, B : integer;
begin
  write('Informe um valor para a variavel A: '); readln(A);
  write('Informe um valor para a variavel B: '); readln(B);
  writeln;
  X := A + B;
  if (X > 10) then
    writeln('O valor da variavel X equivale a: ', X);
end.
```

Observe que após a definição dos tipos de variáveis, é solicitada a leitura dos valores para as variáveis A e B. A instrução **writeln;** é utilizada para pular uma linha em branco. 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 com a instrução **if...then** que permitirá imprimir o resultado da adição caso esta seja maior que 10. Não sendo, o programa é encerrado sem apresentar o referido resultado, uma vez que a condição é falsa. Grave este programa com o nome ADIC_NR1.

Para uma melhor compreensão, considere o seguinte problema: "Ler dois valores inteiros, e independentemente da ordem em que foram entrados, eles devem ser impressos em ordem crescente, ou seja, se forem fornecidos 5 e 3, respectivamente, deverão ser apresentados 3 e 5. O programa em questão deve efetuar a troca dos valores entre as duas variáveis."

Algoritmo

1. Conhecer dois valores inteiros e incógnitos (estabelecer variáveis A e B);
2. Verificar se o valor de A é maior que o valor de B;
   - **2.1.** Se for verdadeiro, efetuar a troca(*) de valores entre as variáveis;
   - **2.2.** Se for falso, pedir para executar o que está estabelecido no passo 3;
3. Apresentar os valores das duas variáveis

(*) Para efetuar a troca, utilizar uma terceira variável, no caso X, por exemplo, de forma que X seja igual ao valor de A, liberando A para receber o valor de B e em seguida B estar livre para receber o valor que está em X, que anteriormente era o valor de A.

Programa em Pascal

```
program ORDENA;
var
  X, A, B : integer;
begin
  write('Informe um valor para a variavel A: '); readln(A);
  write('Informe um valor para a variavel B: '); readln(B);
  writeln;
  if (A > B) then
    begin
      X := A;
      A := B;
      B := X;
    end;
  writeln('Os valores ordenados sao:');
  write(A, ' ', B);
end.
```

Note a utilização das instruções **begin** e **end** após a utilização da instrução **if...then**. Isto se fez necessário, pois a troca de valores é conseguida com a execução de três operações, sendo que essas três operações devem acontecer quando a condição for *Verdadeira*. Sendo a condição *Falsa*, serão apresentados os valores como entrados, uma vez que já estão em ordem. Grave este programa com o nome ORDENA.

4.2 - Operadores Relacionais

No exemplo anterior, foi utilizado o sinal de > (maior que) para verificar o estado da variável quanto ao seu valor, ou seja, a sua condição. Sendo assim, uma condição também pode ser verificada como: diferente, igual, menor que, maior ou igual que e menor ou igual que. Estas verificações são efetuadas com a utilização dos chamados operadores relacionais, conforme tabela seguinte:

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

4.3 - Desvio Condicional Composto

Anteriormente, foi visto como fazer uso da instrução **if...then** do tipo simples. Agora você aprenderá a fazer uso da instrução **if...then...else**, que sendo a condição *Verdadeira,* será executada a instrução que estiver posicionada entre a instrução **if...then** e a **else**. Sendo a condição *Falsa,* será executada a instrução que estiver posicionada logo após a instrução **else**. Caso seja necessário considerar mais de uma instrução para as condições *Verdadeira* ou *Falsa,* utilizar-se-á o conceito de blocos. Então, esta instrução possui a seguinte estrutura:

```
if <(<condição>)> then
  <instruções para condição verdadeira>
else
  <instruções para condição falsa>;
```

Caso venha a existir mais de uma instrução verdadeira ou falsa para uma determinada condição, elas devem estar inseridas em um bloco, como indicado em seguida:

```
if <(condição)> then
  begin
    <instrução1 para condição verdadeira>;
    <instruçãoN para condição verdadeira>;
  end
else
  begin
    <instrução1 para condição falsa>;
    <instruçãoN para condição falsa>;
  end;
```

Observe que nos dois casos abordados anteriormente, qualquer instrução que precede **else** está escrita sem o (;) ponto-e-vírgula. Isto ocorre pelo fato de a instrução **else** ser uma extensão da **if...then**, e sendo assim, o final da condição somente ocorre após o processamento da instrução **else**.

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, ele deve ser apresentado somando a ele mais 5; caso o valor somado não seja maior ou igual a 10, ele deve ser apresentado subtraindo 7".

Algoritmo

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

Programa em Pascal

```
program ADICIONA_NUMEROS_V2;
var
  X, A, B : integer;
begin
  write('Informe um valor para a variavel A: '); readln(A);
  write('Informe um valor para a variavel B: '); readln(B);
  writeln;
  X := A + B;
  write('O resultado equivale a: ');
  if (X >= 10) then
    writeln(X + 5)
  else
    writeln(X - 7);
end.
```

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. É apresentada então a mensagem **O resultado equivale a:**, que não posicionará o cursor na próxima linha, uma vez que está sendo utilizada a instrução **write.** Desta forma, qualquer que seja o resultado avaliado pela condição, ele será apresentado do lado direito da mensagem. Em seguida a esta linha, é questionado no programa a condição que permitirá imprimir o resultado da soma adicionado de 5, caso esta seja maior ou igual a 10. Não sendo, o programa apresentará o resultado subtraindo 7. Atente para a instrução **writeln(X+5)** sem o ponto-e-vírgula. Note que ela está posicionada antes da instrução **else**. Grave este programa com o nome ADIC_NR2.

4.4 - OPERADORES LÓGICOS

Existem ocasiões em que é necessário trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo na mesma instrução **if...then**, efetuando deste forma testes múltiplos. Para estes casos é necessário trabalhar com a utilização dos operadores lógicos, também conhecidos como operadores booleanos. Os operadores relacionais são três: **and**, **or** e **not**. Em alguns casos, o uso de operadores lógicos evita a utilização de muitas instruções **if...then** encadeadas.

4.4.1 - Operador Lógico: AND

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

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

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

```
program TESTA_LOGICA_AND;
var
  NUMERO : integer;
begin
  write('Informe um numero: '); readln(NUMERO);
  writeln;
  if (NUMERO >= 20) and (NUMERO <= 90) then
    writeln('O numero esta na faixa de 20 a 90')
  else
    writeln('O numero esta fora da faixa de 20 a 90');
end.
```

O exemplo acima mostra, por meio da utilização do operador **and**, que somente será apresentada a mensagem: **O número está na faixa de 20 a 90**, caso o valor fornecido para a variável *NUMERO* esteja entre 20 e 90. Qualquer valor fornecido fora da faixa definida apresentará a mensagem: **O número não está na faixa de 20 a 90**. Grave este programa com o nome OP_AND.

4.4.2 - Operador Lógico: OR

O operador do tipo **or** 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-verdade para este tipo de operador:

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

O operador **or** faz com que seja executada uma determinada operação se pelo menos uma das condições mencionadas gerar um resultado lógico verdadeiro. Veja o exemplo em seguida:

```
program TESTA_LOGICA_OR;
var
  SEXO : string;
begin
  write('Entre com o seu sexo: '); readln(SEXO);
  writeln;
  if (SEXO = 'masculino') or (SEXO = 'feminino') then
    writeln('O seu sexo e valido')
  else
    writeln('O seu sexo e invalido');
end.
```

O exemplo acima mostra, por meio da utilização do operador **or**, que somente será apresentada a mensagem "**O seu sexo é válido**", caso o valor fornecido para a variável *SEXO* seja um valor **masculino** ou **feminino** (em caracteres minúsculos). Qualquer outro valor fornecido apresentará a mensagem "**O seu sexo é inválido**". Grave este programa com o nome OP_OR.

Observação: *Quando em um programa são trabalhados dados do tipo string (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.*

4.4.3 - Operador Lógico: NOT

O operador do tipo **not** é utilizado quando se necessita estabelecer que uma determinada condição deve não ser verdadeira ou deve não ser falsa. O operador **not** se caracteriza por inverter o estado lógico de uma condição. Abaixo é apresentada a tabela-verdade para este tipo de operador:

Condição	Resultado
Verdadeira	Falso
Falso	Verdadeira

O operador **not** faz com que seja executada uma determinada operação, invertendo o resultado lógico da condição. Veja o exemplo em seguida:

```
program TESTA_LOGICA_NOT;
var
  A, B, C, X : integer;
begin
  write('Entre um valor para a variavel A: '); readln(A);
  write('Entre um valor para a variavel B: '); readln(B);
  write('Entre um valor para a variavel X: '); readln(X);
  if not (X > 5) then
    C := (A + B) * X
  else
    C := (A - B) * X;
  writeln('O resultado da variavel C corresponde a: ', C);
end.
```

O exemplo acima mostra, por meio da utilização do operador **not**, que somente será efetuado o cálculo de **C := (A + B) * X**, se o valor da variável *X* não for maior que 5. Qualquer valor de 5 para baixo efetuará o cálculo **C := (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* = 12, pois o valor 2 da variável *X* é controlado pela instrução **if not** (*X* > 5) **then**, 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* = 24, como sendo falso, sendo assim, os valores 5 e 1 são subtraídos resultando 4 e multiplicados por 6 resultando 24. Grave este programa com o nome OP_NOT.

Para demonstrar a utilização de operadores lógicos em um exemplo um pouco maior, considere o seguinte problema: "Ler três valores para os lados de um triângulo, considerando lados como: A, B e C. Verificar se os lados fornecidos formam realmente um triângulo. Se for esta condição verdadeira, deve ser indicado qual tipo de triângulo foi formado: isósceles, escaleno ou equilátero".

Algoritmo

Para 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 deve ser considerada. É um triângulo quando A<B+C, quando B<A+C e quando C<A+B.

Tendo certeza de que os valores informados para os três lados formam um triângulo, 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 será escaleno;
3. Caso os lados fornecidos não caracterizem um triângulo, avisar a ocorrência.

Programa em Pascal

```
program TRIANGULO;
var
   A, B, C : real;
begin
  write('Informe o 1o. lado de triangulo: '); readln(A);
  write('Informe o 2o. lado de triangulo: '); readln(B);
  write('Informe o 3o. lado de triangulo: '); readln(C);
  writeln;
  if (A < B + C) and (B < A + C) and (C < A + B) then
    if (A = B) and (B = C) then
      writeln('Triangulo Equilatero')
    else
      if (A = B) or (A = C) or (C = B) then
        writeln('Triangulo Isosceles')
      else
        writeln('Triangulo Escaleno')
  else
    writeln('Os valores fornecidos nao formam um triangulo');
end.
```

Este exemplo demonstra, por meio da utilização dos operadores lógicos, a capacidade de um programa definir se determinados valores fornecidos formam realmente um triângulo. Se a condição for verdadeira, este programa indica o tipo de triângulo formado. Se forem fornecidos, respectivamente, os valores 8, 2 e 3 para os lados A, B e C, será apresentada a mensagem: "**Os valores fornecidos nao formam um triangulo**", pois a primeira condição da instrução **if** ($A < B + C$) **and** ($B < A + C$) **and** ($C < A + B$) **then** resulta o valor lógico falso, uma vez que 8 é maior que a soma de 2 com 3.

Caso sejam fornecidos os valores 8, 8 e 2, a primeira instrução **if** será verdadeira, indicando que os lados fornecidos formam um triângulo. Apesar de o lado A ser igual ao lado B, mas este não ser igual ao lado C, o triângulo formado não é do tipo equilátero. Desta forma, a segunda instrução **if** tem seu resultado lógico como falso, sobrando a verificação da condição para a terceira instrução **if** que indica o triângulo como sendo do tipo isósceles, pois o lado A é igual ao lado B. Grave este programa com o nome OPERADOR.

4.5 - EXERCÍCIOS

1. Com base nas tabelas-verdade, 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 = 7, C = 8 e D = 6.

   a. `not (X > 3)`

   Verdadeiro ( ) Falso ( )

   b. `(X < 1) and not (B > D)`

   Verdadeiro ( ) Falso ( )

   c. `not (D < 0) and (C > 5)`

   Verdadeiro ( ) Falso ( )

   d. `not (X > 3) or (C < A)`

   Verdadeiro ( ) Falso ( )

   e. `(A > B) or (C > B)`

   Verdadeiro ( ) Falso ( )

   f. `(X >= 2) or (X < 7)`

   Verdadeiro ( ) Falso ( )

   g. `(X < 1) and (B >= D)`

   Verdadeiro ( ) Falso ( )

   h. `(D < 0) or (C > 5)`

   Verdadeiro ( ) Falso ( )

   i. `not (D > 3) or not (B < 7)`

   Verdadeiro ( ) Falso ( )

   j. `(A > B) or not (C > B)`

   Verdadeiro ( ) Falso ( )

2. Transcreva apenas a fórmula que será executada de acordo com o resultado lógico de cada condição. Para as saídas considere os seguintes valores: A=3, B=2, C=-5 e D=7.

a. Resposta:_____

```
if not (D > 5) then
   X := (A + B) * D
else
   X := (A - B) / C;
```

b. Resposta: _____

```
if (A > 2) and (B < 7) then
  X := (A + 2) * (B - 2)
else
  X := (A + B) / D * (C + D);
```

c. Resposta: _____

```
if (A = 2) or (B < 7) then
  X := (A + 2) * (B - 2)
else
  X := (A + B) / D * (C + D);
```

d. Resposta: _____

```
if (A > 2) or not (B < 7) then
  X := A + B - 2
else
  X := A - B;
```

e. Resposta: _____

```
if not (A > 2) or not (B < 7) then
  X := A + B
else
  X := A / B;
```

f. Resposta: _____

```
if not (A > 3) and not (B < 5) then
  X := A + D
else
  X := D / B;
```

g. Resposta: _____

```
if (C >= 2) and (B <= 7) then
  X := (A + D) / (B + D)
else
  X := D * (C - (B + D));
```

h. Resposta: _____

```
if (A >= 2) or (C <= 1) then
  X := (A + D) / 2
else
  X := D * C;
```

3. Desenvolva a codificação em Pascal dos seguintes problemas:

a. Efetuar a leitura de três valores de qualquer tipo (*integer*, *real*, *string*), representados pelas variáveis A, B e C, e apresentá-los dispostos em ordem crescente. A ordem de entrada dos valores nas variáveis é A, B e C. Assim sendo, a ordem de saída das variáveis também será A, B e C. Nenhuma outra combinação de saída deve ser utilizada.

b. Efetuar a leitura de três valores (variáveis A, B e C) e efetuar o cálculo da equação de segundo grau, apresentando as duas raízes, se para os valores informados for passível efetuar o referido cálculo. Lembre-se de que se o valor de delta for menor que zero, não existirá o resultado de nenhuma raiz, pois as duas raízes somente poderão ser calculadas caso delta seja maior que zero. Se o valor de delta for igual a zero, existirá apenas uma raiz como resultado final.

c. Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir em vídeo uma mensagem informando que o aluno foi aprovado, se o valor da média escolar for maior ou igual a 5. Se o aluno não foi aprovado, apresentar uma mensagem informando a condição de reprovação. Apresentar junto das mensagens o valor da média obtida pelo aluno.

d. Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir em vídeo uma mensagem informando que o aluno foi aprovado, se o valor da média escolar for maior ou igual a 7. Se o valor da média for menor que 7, solicitar que seja informada a nota de exame, somar com o valor da média e obter nova média. Se a nova média for maior ou igual a 5, apresentar uma mensagem informando que o aluno foi aprovado em exame. Se o aluno não foi aprovado em nenhuma das circunstâncias anteriores, apresentar uma mensagem informando sua reprovação. Apresentar junto das mensagens o valor da média obtida pelo aluno.

e. Ler dois valores numéricos inteiros e apresentar a diferença do maior valor em relação ao menor valor informados.

Capítulo 5

Laços, Loopings ou Malhas de Repetição

Existem situações em que é necessário repetir um determinado trecho de um programa um número de vezes. Isto pode ser conseguido de duas formas: na primeira será escrito o mesmo trecho tantas vezes quanto necessário, um tanto trabalhoso, e na segunda forma pode ser utilizado o conceito de looping. Os loopings são conhecidos também por laços ou malhas.

Suponha que você tem um programa que deve executar um determinado trecho de instruções cinco vezes. Com o conhecimento adquirido até este momento, você, com toda certeza, vai optar pela primeira forma, escrevendo-o tantas vezes quanto necessário, no caso cinco vezes. Por exemplo, imagine um programa que peça a leitura de dois valores para as variáveis A e B, respectivamente, efetue a adição de um com o outro, implicando o resultado na variável de resposta R e em seguida apresente o valor do resultado obtido, repetindo esta seqüência por cinco vezes. A seguir, é apresentado um programa que exemplifica esta situação. Digite-o e grave-o com o nome CALCULO:

```
program CALCULO;
var
  A, B, R : integer;
begin
  write('Entre um valor para A: '); readln(A);
  write('Entre um valor para B: '); readln(B);
  writeln;
  R := A + B;
  writeln('O resultado corresponde a: ', R);
  writeln;
  write('Entre um valor para A: '); readln(A);
  write('Entre um valor para B: '); readln(B);
  writeln;
  R := A + B;
  writeln('O resultado corresponde a: ', R);
  writeln;
  write('Entre um valor para A: '); readln(A);
  write('Entre um valor para B: '); readln(B);
  writeln;
```

```
  R := A + B;
  writeln('O resultado corresponde a: ', R);
  writeln;
  write('Entre um valor para A: '); readln(A);
  write('Entre um valor para B: '); readln(B);
  writeln;
  R := A + B;
  writeln('O resultado corresponde a: ', R);
  writeln;
  write('Entre um valor para A: '); readln(A);
  write('Entre um valor para B: '); readln(B);
  writeln;
  R := A + B;
  writeln('O resultado corresponde a: ', R);
  writeln;
end.
```

A principal vantagem em utilizar o conceito de looping é que o programa passa a ter um tamanho menor, podendo ser aumentada sua amplitude de processamento sem alterar o tamanho do código de programação. Desta forma, podem-se determinar repetições com número variado de vezes.

5.1 - Repetição com Teste Lógico no Início do Looping

Esta estrutura de looping caracteriza-se por efetuar um teste lógico no início de um looping, verificando se é permitido executar o trecho de instruções subordinado a este. A utilização de teste lógico no início do looping é conseguida com a instrução **while...do.**

A estrutura **while...do** tem o seu funcionamento controlado por condição. Desta forma, pode executar um determinado conjunto de instruções, enquanto a condição verificada permanecer verdadeira. No momento em que essa condição se torna falsa, o processamento da rotina é desviado para fora do looping. Sendo a condição falsa logo no início do looping, as instruções contidas nele são ignoradas. Caso seja necessário executar mais de uma instrução para uma condição verdadeira dentro de um looping, elas devem estar mencionadas em um bloco definido com as instruções **begin** e **end**. Desta forma a instrução **while...do** deve ser escrita:

```
while <(condição)> do
  begin
    <instruções para condição verdadeira>;
  end;
```

1º Exemplo

Como exemplo, considere o problema proposto, em que é necessário executar cinco vezes a solicitação dos dois números para o cálculo da adição. Neste caso, é necessário definir um contador de vezes para controlar a execução do programa, e a cada vez que for

executado o trecho desejado do programa, esse contador deve ser incrementado de mais 1. Observe em seguida os detalhes para a solução deste problema:

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 os valores;
4. Efetuar o cálculo, implicando o resultado em R;
5. Apresentar o valor calculado contido na variável R;
6. Acrescentar ao contador mais 1;
7. Quando o contador for maior que 5, encerrar o processamento.

Programa em Pascal

```
program LOOPING_1A;
var
  A, B, R, I : integer;
begin
  I := 1;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I + 1;
    end;
end.
```

Além da utilização das variáveis *A*, *B* e *R*, foi necessário criar uma terceira variável, no caso *I*, para controlar a contagem do número de vezes que o trecho de programa deve ser executado.

Assim que o programa é executado, a variável contador é atribuída com o valor 1 (*I* := 1). Em seguida, a instrução **while** (*I* <= 5) **do** efetua a checagem da condição estabelecida, verificando que a condição é verdadeira, pois o valor da variável *I* que neste momento é 1, é realmente menor que 5 e enquanto for, deve processar o looping.

Desta forma, é iniciada a execução da rotina de instruções contidas no looping delimitado com a instrução **while**. Depois de efetuar a solicitação dos valores, ter processado a operação de adição e exibido o resultado, o programa encontra a linha com a instrução *I* := *I* + 1, indicando o acréscimo de 1 à variável contador. Observe que a variável *I* possui neste

momento o valor 1 e acrescentando mais 1, ela passa a ter o valor 2, ou seja, *I* := *I* + 1, sendo assim *I* := 1 + 1 que resultará *I* = 2.

Estando a variável *I* com o valor 2, o processamento do programa volta para a instrução **while** (*I* <= 5) **do**, que verifica a condição da variável. Sendo essa condição *Verdadeira,* será executada novamente a mesma rotina de instruções, e quando o processamento do programa chegar na instrução *I* := *I* + 1, fará com que a variável *I* passe a possuir o valor 3. Desta forma, o programa processa novamente a rotina de instruções, passando o valor de *I* para 4, que será verificado, e sendo menor que 5, será executada mais uma vez a mesma rotina de instruções. Neste ponto, a variável *I* passa a ter o valor 5. Perceba que a instrução **while** (*I* <= 5) **do** vai efetuar a checagem do valor da variável *I* que é 5 com a condição *I* <= 5. Veja que 5 não é menor que 5, mas é igual. Sendo esta condição verdadeira, a rotina deve ser executada mais uma vez. Em seguida, o valor da variável *I* passa a ser 6, que resultará para a instrução **while** uma condição falsa. E por conseguinte, desviará o processamento para a primeira instrução encontrada após o bloco definido entre as instruções **begin** e **end**, no caso para o fim do programa definido pela instrução (**end.**). Grave o programa com o nome LOOP_1A.

No exemplo apresentado, a rotina de cálculo é executada durante cinco vezes. Caso queira executar essa rotina um número maior ou menor de vezes, basta trocar o valor 5 da instrução **while** (*I* <= 5) **do** pelo valor desejado. Por exemplo, executar a mesma rotina por 15 vezes, a instrução definida deve ficar assim: **while** (*I* <= 15) **do**.

Imagine ainda uma outra situação, em que o usuário deseja executar a rotina do programa várias vezes, mas ele não sabe, ao certo, quantas vezes deve executar o trecho de programa. Neste caso, não é conveniente manter um contador para controlar o looping. Seria melhor que o programa fizesse ao usuário uma pergunta, solicitando se ele deseja ou não continuar executando o programa. 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 os valores;
4. Efetuar o cálculo, 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.

Programa em Pascal

```
program LOOPING_1B;
var
  A, B, R : integer;
  RESP : string;
begin
```

```
  RESP := 'SIM';
  while (RESP = 'SIM') or (RESP = 'S') do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      write('Deseja continuar? '); readln(RESP);
    end;

end.
```

No programa anterior, o contador foi substituído pela variável *RESP*, que enquanto tiver o seu valor igual a 'SIM' ou igual a 'S' executará a rotina existente entre as instruções **begin** e **end;**. Neste caso, o número de vezes que a rotina vai se repetir será controlado pelo usuário e dependerá da informação fornecida para a variável *RESP*. Grave o programa com o nome LOOP_1B.

2º Exemplo

Considere como exemplo um programa que efetue o cálculo da fatorial de um número qualquer. Supondo que esse número seja 5, o programa deve apresentar o resultado 5! (fatorial de 5). Desta forma, temos que 5! = 5 . 4 . 3 . 2 . 1 ou 5! = 1 . 2 . 3 . 4 . 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 deve executar as multiplicações sucessivamente e acumulá-las a fim de possuir o valor 120 após cinco passos. O número de passos deve ser controlado por um contador. Veja a seguir:

Algoritmo

1. Inicializar as variáveis FATORIAL e CONTADOR com 1;
2. Solicitar o valor de um número para calcular a sua fatorial;
3. Multiplicar sucessivamente a variável FATORIAL pela variável CONTADOR;
4. Incrementar 1 à variável CONTADOR, efetuando o controle até o limite definido no passo 2;
5. Apresentar ao final o valor obtido.

Pelo fato de ter que efetuar o cálculo de uma fatorial de um número qualquer (N!), isto implica que o contador deve variar de 1 a N, e por este motivo, deve ser a variável *CONTA-*

DOR inicializada com valor 1. Pelo fato de a variável *FATORIAL* possuir ao final o resultado do cálculo da fatorial pretendida (por meio de uma multiplicação sucessiva), ela deve ser iniciali`zada com valor 1. Se ela for inicializada com zero, não existirá resultado final, pois qualquer valor multiplicado por zero resulta zero.

Observe dentro do looping a indicação de dois contadores. O primeiro funcionando como um acumulador, pois é este que 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 programa, as variáveis *CONTADOR* e *FATORIAL* são igualadas ao valor 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 de mais 1, tendo agora o valor 2. Como 2 é menor ou igual a cinco, ocorre um novo cálculo, e desta vez a variável *FATORIAL* que possui o valor 1 é multiplicada pela variável *CONTADOR* que possui o valor 2, resultando 2 para *FATORIAL*. Então, 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 da fatorial do número definido. Veja -em seguid a tabela com os valores das variáveis antes e durante a execução do looping, considerando obter o valor da 5!:

Contador	Fatorial	Fatorial ← 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*. Você pode fazer este acompanhamento usando o processo de debug do Turbo Pascal (veja o *tópico* 5.2 deste capítulo).

Programa em Pascal

```
program FATORIAL_A;
var
  CONTADOR, N : integer;
  FATORIAL : longint;
begin
  FATORIAL := 1;
  CONTADOR := 1;
  writeln('Programa Fatorial');
  writeln;
```

```
  write('Fatorial de que numero: '); readln(N);
  writeln;
  while (CONTADOR <= N) do
    begin
      FATORIAL := FATORIAL * CONTADOR;
      CONTADOR := CONTADOR + 1;
    end;
  writeln('Fatorial de ', N, ' equivale a ', FATORIAL);
  writeln;
  write('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar este programa, é adequado fechar a janela ativa. Para tanto, execute o comando **Window/Close**. Em seguida abra uma nova janela com o comando **File/New** e digite o programa em questão, gravando-o com o nome FATOR_A.

No código apresentado, estão sendo exibidas pela instrução **writeln**('Fatorial de ', *N*, 'equivale a', *FATORIAL*) quatro informações, sendo duas mensagens e duas variáveis. Note que isto é feito utilizando uma vírgula para separar os elementos a serem apresentados.

Após esta linha é apresentada a instrução **write**('Tecle <ENTER> para encerrar: '), seguida da instrução **readln** com ponto-e-vírgula, que fica no aguardo do pressionamento da tecla **<ENTER>** para então encerrar o programa. Esta alternativa permite que seja visualizado o resultado do programa antes de voltar ao editor. Desta forma, não é necessário utilizar as teclas **<ALT> + <F5>** para visualizar a tela de saída.

5.2 - O Modo Debug

Os ambientes Turbo Pascal e Free Pascal possuem um recurso bastante útil ao programador, que possibilita visualizar o conteúdo e o comportamento de variáveis e de expressões lógicas durante a execução de um programa. Desta forma, é possível identificar erros de lógica e de definição de valores. Este recurso é denominado "debug", ou seja, o processo para tirar erros de um programa de computador. O termo "bug" (na terminologia para informática) em inglês é utilizado como sinônimo de erro em programas.

Para fazer uso deste recurso, é ideal estar com um programa carregado na memória do seu computador. Mas antes de fazer isto, execute o comando **Window/Close all**, para fechar todas as janelas que eventualmente estejam abertas, tanto no Turbo Pascal como no Free Pascal.

5.2.1 - Debug no Turbo Pascal

No ambiente Turbo Pascal execute o comando **File/Open**, ou acione a tecla de função **<F3>**. Será apresentada a caixa de diálogo **Open a File**. Neste momento, selecione na área **Files** da caixa de diálogo **Open a File** o programa **LOOP_1A.PAS**.

Após carregar o programa **LOOP_1A.PAS** na memória, é necessário abrir a tela do modo debug que fará a vigilância das variáveis e expressões lógicas. Selecione neste momento o comando **Debug/Watch** no menu, como indicado na figura 5.1. Será apresentada a janela **Watches**, como indicado na figura 5.2, sendo colocada sobre a janela em que está o programa a ser verificado.

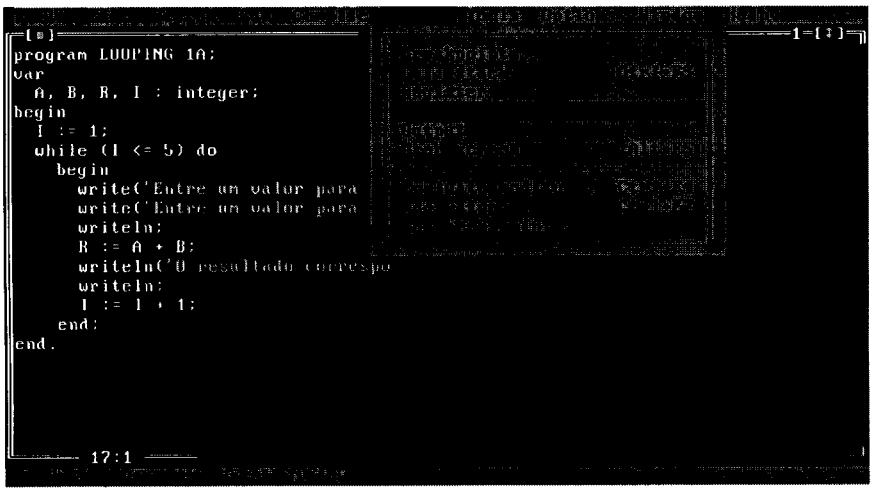


Figura 5.1 - Execução do comando Debug/Watch.

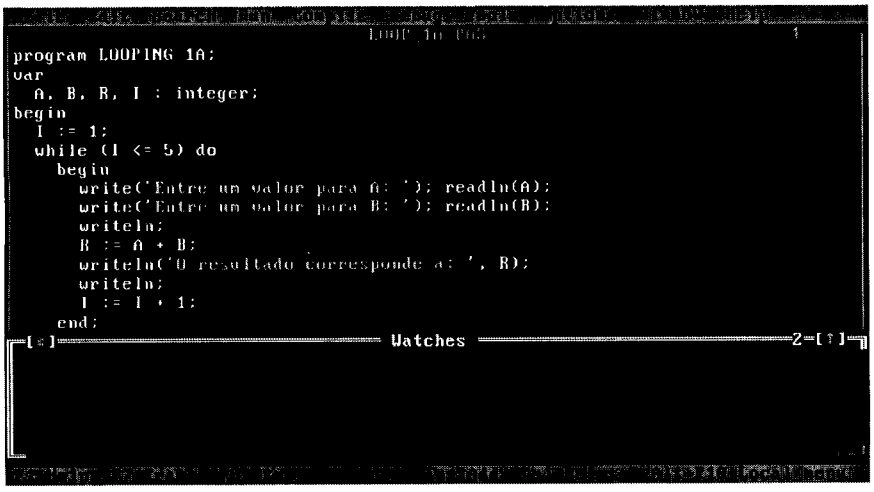


Figura 5.2 - Apresentação da janela Watches.

Neste momento, é necessário informar na janela **Watches**, as variáveis que devem ser vigiadas durante a execução do programa. Neste caso, é interessante informar todas as variáveis e também a expressão lógica que controla o looping da instrução **while**. Para tanto, existem três maneiras de fazer esta operação: a primeira executando o comando **Debug/Add watch** (esse comando pode ser executado independentemente da execução do comando **Debug/Watch**, pois caso a janela **Watches** não esteja aberta, esta o será), a segunda

pressionando as teclas **<CTRL> + <F7>** e a terceira maneira acionando a tecla **<INS>**. Após a execução de uma das formas descritas, será apresentada a caixa de diálogo **Add Watch**, como indicado na figura 5.3.

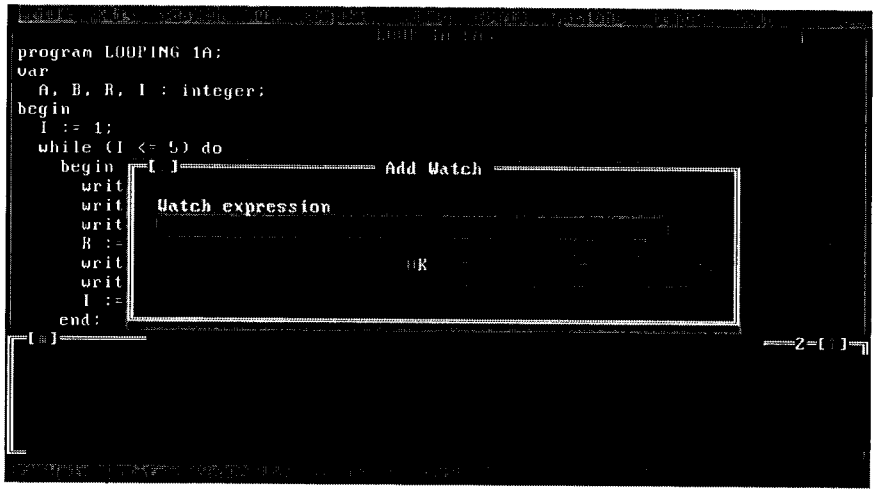


Figura 5.3 - Apresentação da caixa de diálogo Add Watch.

No campo **Watch expression**, digite a expressão lógica **I <= 5** e tecle **<ENTER>** ou clique no botão **OK**. Será então inserida na janela **Watches** a referida expressão com a indicação **Unknown identifier**, avisando que a expressão não é conhecida. A figura 5.4 mostra esta ocorrência.

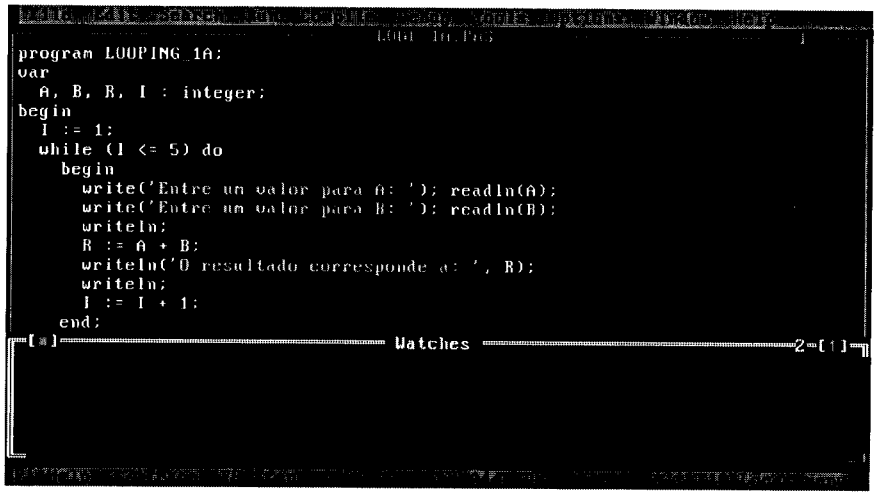


Figura 5.4 - Apresentação da janela Watches com a expressão lógica inserida.

Da mesma forma, insira também na janela **Watches** as variáveis **A**, **B**, **R** e **I**, de forma que fique semelhante ao exemplo da figura 5.5.

```
program LOOPING_10;
var
  A, B, R, I : integer;
begin
  I := 1;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I + 1;
    end;
[=]═══════ Watches ═══════2═[↑]═
 I <= 5: Unknown identifier
```

Figura 5.5 - Apresentação da janela Watches com a expressão lógica e as variáveis inseridas.

Agora basta executar o programa para acompanhar a evolução dos valores das variáveis e da expressão lógica, mas a execução não será com o comando **Run/Run**, e sim com o pressionamento sucessivo da tecla de função **<F8>**, denominada **Step**, que fará a execução do programa passo a passo.

Ao ser pressionada a tecla de função **<F8>** pela primeira vez, os valores das variáveis são alterados, a expressão lógica resulta como resposta **True** (Verdadeiro) e uma barra fica posicionada sobre a instrução **begin**.

Em seguida, pressione pela segunda vez a tecla **<F8>** e a barra é posicionada sobre a instrução **I := 1**. Observe que neste momento o valor da condição **I <= 5** passa a ser **True** (verdadeiro).

Ao executar a tecla de função **<F8>** pela terceira vez, a barra é posicionada sobre a instrução **while (I <= 5) do** e a variável **I** é inicializada com valor 1. A figura 5.6 mostra esta ocorrência.

```
program LOOPING_10;
var
  A, B, R, I : integer;
begin
  I := 1;

    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I + 1;
    end;
[=]═══════ Watches ═══════2═[↑]═
 I <= 5: True
```

Figura 5.6 - Apresentação da terceira execução.

Na quarta vez, a barra é posicionada sobre a instrução **write**('Entre um valor para A: '); **readln**(*A*). Pressione pela quinta vez a tecla de função **<F8>** e será apresentada a mensagem solicitando a digitação de um valor para a variável **A.** No caso, digite o valor **3** e tecle **<ENTER>**. Neste momento, é apresentada a barra posicionada na instrução **write**('Entre um valor para B: '); **readln**(*B*), e na janela **Watches** a variável **A** com o valor digitado, no caso **3.**

Pressione pela sexta vez a tecla de função **<F8>**. Será apresentada a mensagem solicitando a digitação de um valor para a variável **B.** No caso digite **2** e tecle **<ENTER>**. Neste momento, a barra de selecionamento é posicionada sobre a instrução **writeln** que pula uma linha em branco. Pressione pela sétima vez a tecla de função **<F8>** e será posicionada a barra de selecionamento sobre a linha com a fórmula **R := A + B;** como mostra a figura 5.7.

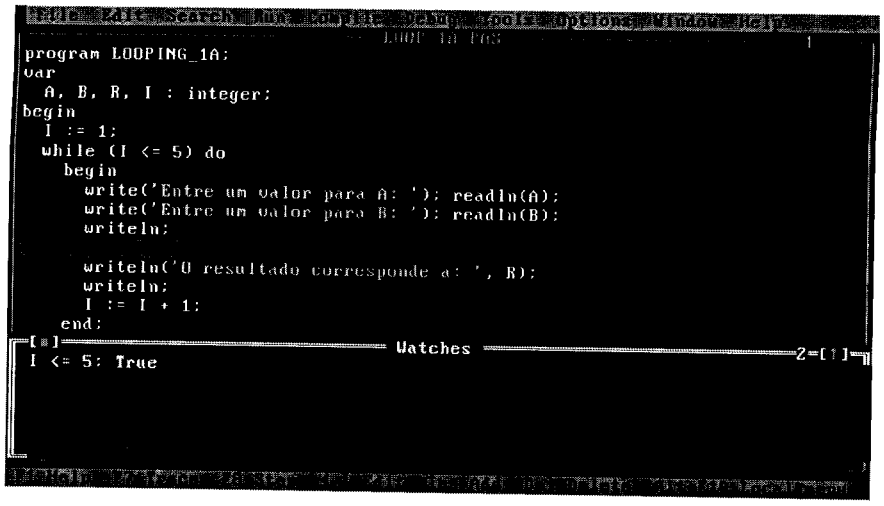


Figura 5.7 - Apresentação da sétima execução.

Ao ser pressionada a tecla de função pela oitava vez, o valor da adição dos dois valores é apresentado ao lado da variável **R**, no caso **5**. Na janela **Watches** a barra é posicionada sobre a mensagem **writeln**('O resultado corresponde a: ', *R*).

Em seguida, acione três vezes a tecla de função **<F8>** e será então incrementado mais um à variável **I**, que passa a ter o valor **2**, como mostra a figura 5.8. Na décima segunda execução, a barra é posicionada sobre a instrução **while (I <= 5) do** novamente, e a partir deste ponto será executada novamente a rotina descrita anteriormente por mais quatro vezes.

```
var
  A, B, R, I : integer;
begin
  I := 1;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I + 1;

end.
[ ]═══ Watches ═══2=[↑]
 I <= 5: True
```

Figura 5.8 - Apresentação da variável I com valor 2 após a décima primeira execução.

A figura 5.9 mostra a variável **I** com valor **5** após várias execuções da tecla de função **<F8>**. Note que nesta figura, a barra está posicionada sobre a instrução **while (I <= 5) do** cuja condição ainda é verdadeira. Continue acionando a tecla de função **<F8>**, fornecendo cuidadosamente mais dois valores. Quando a variável **I** é incrementada de mais um, passando seu valor para **6**, a condição da expressão **I <= 5** se torna **False** (Falsa) e o laço será encerrado.

```
var
  A, B, R, I : integer;
begin
  I := 1;

    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I + 1;
    end;
end.
[ ]═══ Watches ═══2=[↑]
 I <= 5: True
```

Figura 5.9 - Apresentação da variável I com valor 5 e a condição I <= 5 com valor True.

Acione a tecla de função **<F8>** que posicionará a barra sobre a instrução **while (I <= 5) do**, que neste momento é falsa. Acionando mais uma vez a tecla de função **<F8>**, neste caso, pela última vez, posicionará a barra na instrução **end** com o ponto final.

5.2.2 - Debug no Free Pascal

No ambiente Turbo Pascal execute o comando **File/Open**, ou acione a tecla de função **<F3>**. Será apresentada a caixa de diálogo **Open a File**. Neste momento, selecione na área **Files** da caixa de diálogo **Open a File** o programa **LOOP_1A.PAS**.

Após carregar o programa **LOOP_1A.PAS** na memória, é necessário abrir a tela do modo debug que fará a vigilância das variáveis e expressões lógicas. Selecione neste momento o comando **Debug/Watches** no menu, como indicado na figura 5.10. Será apresentada a janela **Watches**, como indicado na figura 5.11, sendo colocada sobre a janela em que está o programa a ser verificado.

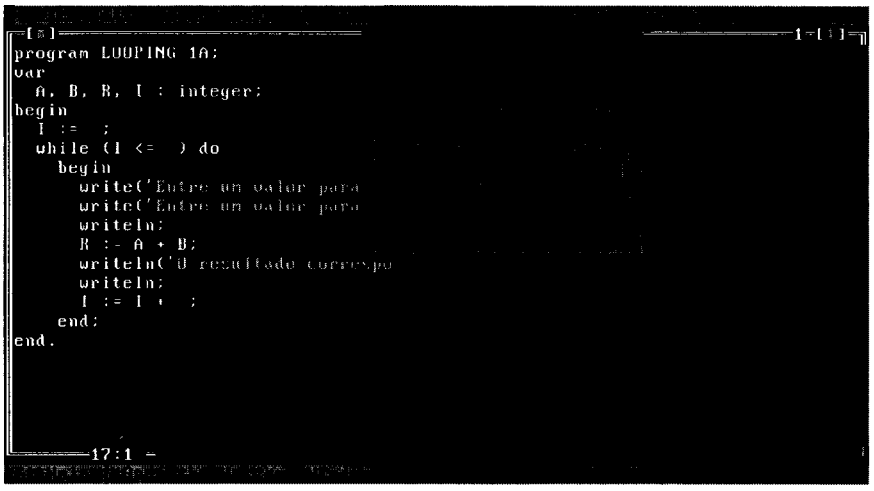


Figura 5.10 - Execução do comando Debug/Watches.

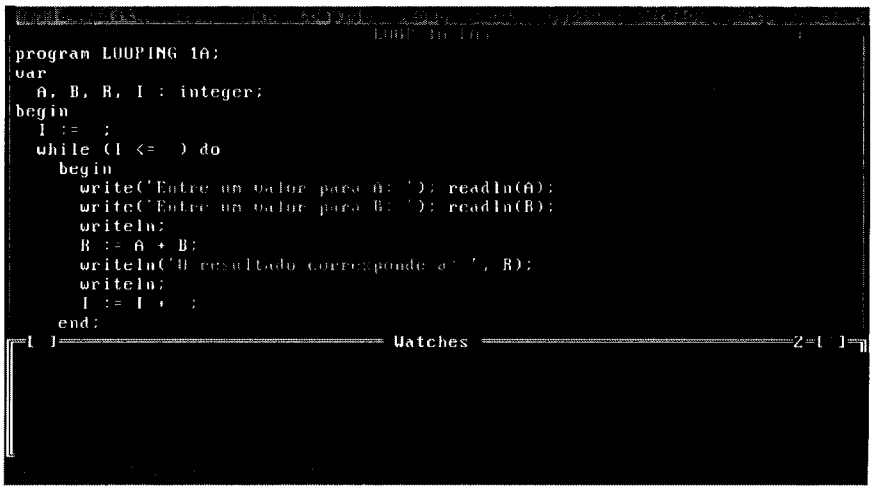


Figura 5.11 - Apresentação da janela Watches.

Neste momento, é necessário informar na janela **Watches** as variáveis que devem ser vigiadas durante a execução do programa. Neste caso, é interessante informar todas as variáveis e também a expressão lógica que controla o looping da instrução **while**. Para tanto, existem três maneiras de fazer esta operação: a primeira executando o comando **Debug/Add watch** (esse comando pode ser executado independentemente da execução do comando **Debug/Watch**, pois caso a janela **Watches** não esteja aberta, esta o será), a segunda pressionando as teclas **<CTRL> + <F7>** e a terceira maneira acionando a tecla **<INS>**. Após a execução de uma das formas descritas, será apresentada a caixa de diálogo **Edit Watch**, como indicado na figura 5.12.

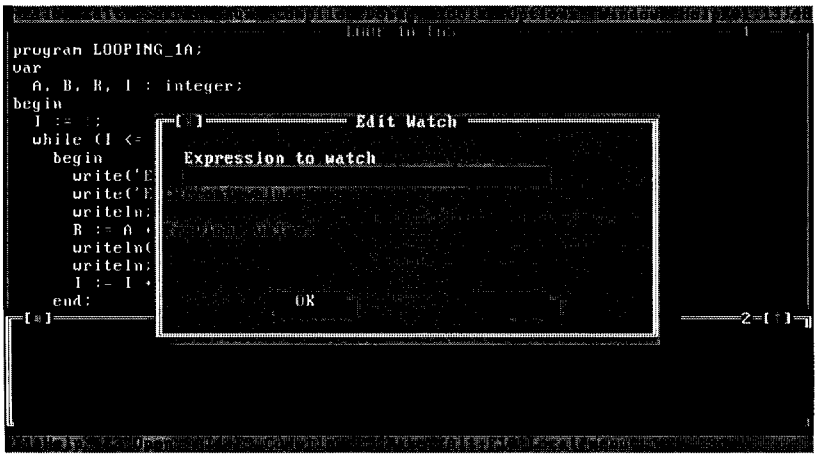


Figura 5.12 - Apresentação da caixa de diálogo Edit Watch.

No campo **Expression to watch**, digite a expressão lógica **I <= 5** e tecle **<ENTER>** ou clique no botão **OK**. Será então inserida na janela **Watches** a referida expressão com a indicação **Unknown value**, avisando que a expressão não é conhecida. A figura 5.13 mostra esta ocorrência.

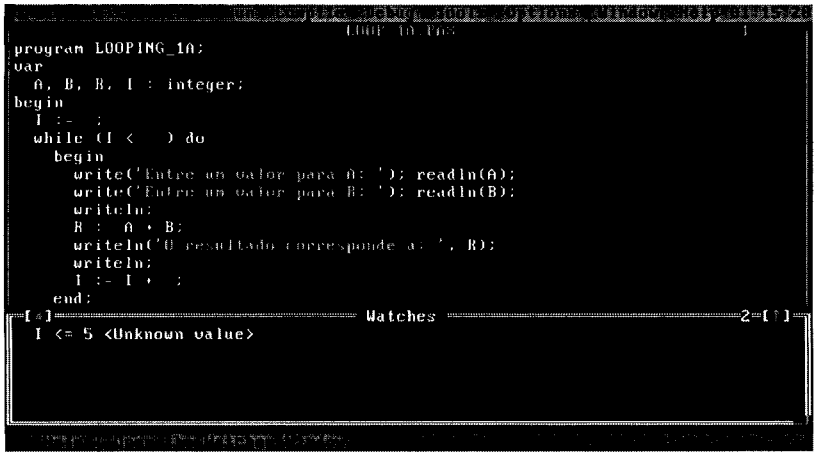


Figura 5.13 - Apresentação da janela Watches com a expressão lógica inserida.

Da mesma forma, insira também na janela **Watches** as variáveis **A**, **B**, **R** e **I**, de forma que fique semelhante ao exemplo da figura 5.14.

```
program LOOPING_1A;
var
  A, B, R, I : integer;
begin
  I := 1;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I + 1;
    end;
[1]═══ Watches ═══2=[1]
I <Unknown value>
```

Figura 5.14 - Apresentação da janela Watches com a expressão lógica e as variáveis inseridas.

Agora basta executar o programa para acompanhar a evolução dos valores das variáveis e da expressão lógica, mas a execução não será com o comando **Run/Run**, e sim com o pressionamento sucessivo da tecla de função **<F8>**, denominada **Step**, que fará a execução do programa passo a passo.

Ao ser pressionada a tecla de função **<F8>** pela primeira vez, os valores das variáveis são alterados, a expressão lógica resulta como resposta **true** (Verdadeiro) e uma barra fica posicionada sobre a instrução **begin**.

Em seguida, pressione pela segunda vez a tecla **<F8>** e a barra é posicionada sobre a instrução **I := 1**. Observe que neste momento o valor da condição **I <= 5** passa a ser **true** (verdadeiro).

Ao executar a tecla de função **<F8>** pela terceira vez, a barra é posicionada sobre a instrução **while (I <= 5) do**, e a variável **I** é inicializada com valor 1. A figura 5.15 mostra esta ocorrência.

Na quarta vez, a barra é posicionada sobre a instrução **write('**Entre um valor para A: '**); readln(***A***)**. Pressione pela quinta vez a tecla de função **<F8>** e será apresentada a mensagem solicitando a digitação de um valor para a variável **A.** No caso, digite o valor **3** e tecle **<ENTER>**. Neste momento, é apresentada a barra posicionada na instrução **write('**Entre um valor para B: '**); readln(***B***)**, e na janela **Watches** a variável **A** com o valor digitado, no caso **3**.

Pressione pela sexta vez a tecla de função **<F8>**. Será apresentada a mensagem solicitando a digitação de um valor para a variável **B.** No caso digite **2** e tecle **<ENTER>**. Neste momento, a barra de selecionamento é posicionada sobre a instrução **writeln** que pula uma linha em branco. Pressione pela sétima vez a tecla de função **<F8>** e será posicionada a barra de selecionamento sobre a linha com a fórmula **R := A + B;** como mostra a figura 5.16.

```
program LOOPING 1A;
var
  A, B, R, I : integer;
begin
  I :=  ;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I +  ;
    end;
[#]═══════════════ Watches ═══════════════2=[ ]═

 !I!= 1
```

Figura 5.15 - Apresentação da terceira execução.

```
program LOOPING 1A;
var
  A, B, R, I : integer;
begin
  I :=  ;
  while (I <=  ) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I +  ;
    end;
[#]═══════════════ Watches ═══════════════2=[↑]═

  I = 1
```

Figura 5.16 - Apresentação da sétima execução.

Ao ser pressionada a tecla de função pela oitava vez, o valor da adição dos dois valores é apresentado ao lado da variável **R**, no caso **5**. Na janela **Watches** a barra é posicionada sobre a mensagem **writeln('**O resultado corresponde a: ', *R*).

Em seguida, acione três vezes a tecla de função **<F8>** e será então incrementado mais um à variável **I**, que passa a ter o valor **2**, como mostra a figura 5.17. Na décima segunda execução, a barra é posicionada sobre a instrução **while (I <= 5) do** novamente, e a partir deste ponto será executada novamente a rotina descrita anteriormente por mais quatro vezes.

```
program LOOPING_1A;
var
  A, B, R, I : integer;
begin
  I :=  ;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I +  ;
    end;
[=]═══════════ Watches ═══════════2=[ ]═
!I!= 2
```

Figura 5.17 - Apresentação da variável I com valor 2 após a décima primeira execução.

A figura 5.18 mostra a variável **I** com valor **5** após várias execuções da tecla de função **<F8>**. Note que nesta figura, a barra está posicionada sobre a instrução **while (I <= 5) do** cuja condição ainda é verdadeira. Continue acionando a tecla de função **<F8>**, fornecendo cuidadosamente mais dois valores. Quando a variável **I** é incrementada de mais um, passando seu valor para **6**, a condição da expressão **I <= 5** se torna **false** (Falsa) e o laço será encerrado.

```
program LOOPING_1A;
var
  A, B, R, I : integer;
begin
  I :=  ;
  while (I <= 5) do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
      I := I +  ;
    end;
[=]═══════════ Watches ═══════════2=[ ]═
!I!= 5
```

Figura 5.18 - Apresentação da variável I com valor 5 e a condição I <= 5 com valor True.

Acione a tecla de função **<F8>** que posicionará a barra sobre a instrução **while (I <= 5) do**, que neste momento é falsa. Acionando mais uma vez a tecla de função **<F8>**, neste caso, pela última vez, posicionará a barra na instrução **end** com o ponto final.

Com o recurso apresentado neste tópico, você aprendeu a fazer uma verificação dos valores de seu programa, podendo identificar erros lógicos de uma forma mais fácil.

5.3 - Repetição do Tipo com Teste Lógico no Fim do Looping

Esta estrutura caracteriza-se por efetuar um teste lógico no final de um looping, sendo parecida com a **while**. A estrutura em questão é denominada de **repeat**, sendo utilizada com o conjunto de instruções **repeat...until**.

Seu funcionamento é controlado por decisão, porém este tipo de looping efetua a execução de um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida. Diferente da estrutura **while**, que executa somente um conjunto de instruções enquanto a condição é verdadeira.

Desta forma **repeat** tem seu funcionamento em sentido contrário a **while**, pois sempre processa um conjunto de instruções, no mínimo uma vez, até que a condição se torne *Verdadeira*. Para a estrutura **repeat** um conjunto de instruções é executado enquanto a condição se mantém *Falsa* e até que ela seja *Verdadeira*. Desta forma, a instrução **repeat...until** deve ser escrita:

```
repeat
  <instrução1 até que condição seja verdadeira>;
  <instrução2 até que condição seja verdadeira>;
  <instrução3 até que condição seja verdadeira>;
  <instruçãoN até que condição seja verdadeira>;
until <(condição)>;
```

1º Exemplo

Para exemplificar a utilização deste tipo de estrutura de looping, será considerado o exemplo anterior: "o programa deve pedir a leitura de dois valores para as variáveis *A* e *B*, efetuar a adição dos dois valores e implicar o resultado na variável de resposta *R* e em seguida apresentar o valor do resultado obtido, repetindo esta seqüência por cinco vezes".

Algoritmo

1. Criar uma variável para servir como contador com valor inicial 1;
2. Ler os valores;
3. Efetuar o cálculo, implicando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Acrescentar ao contador mais 1;
6. Repetir os passos 2, 3, 4 e 5 até que o contador seja maior que 5.

Programa em Pascal

```
program LOOPING_2A;
var
  A, B, R, I : integer;
begin
  I := 1;
  repeat
    write('Entre um valor para A: '); readln(A);
    write('Entre um valor para B: '); readln(B);
    writeln;
    R := A + B;
    writeln('O resultado corresponde a: ', R);
    writeln;
    I := I + 1;
  until (I > 5);
end.
```

Antes de digitar este programa, é adequado fechar as janelas que estejam ativas. Para tanto, execute o comando **Window/Close all**. Em seguida abra uma nova janela com o comando **File/New** e digite o programa em questão, gravando-o com o nome LOOP_2A.

Assim que o programa é executado, a variável contador é inicializada com o valor 1 (*I* := 1). Em seguida, a instrução **repeat** indica que todo trecho de instruções situado até a instrução **until** deve ter o seu processamento repetido até que a sua condição se torne satisfeita, ou seja, se torne verdadeira.

Sendo assim, tem início a execução da rotina de instruções contidas entre as instruções **repeat...until**. Depois de efetuar a primeira execução das instruções, o programa encontra a linha *I* := *I* + 1. Neste momento é acrescentado mais 1 à variável *I*, passando a ter o valor 2. Em seguida é encontrada a linha com a instrução **until** (*I* > 5), que efetuará o retorno à instrução **repeat** para que seja reprocessada a seqüência de comandos, até que o valor da variável *I* 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 do número de vezes de uma determinada rotina em uma estrutura de repetição. Considere que o usuário encerrará o processamento segundo a sua vontade.

Algoritmo

1. Criar uma variável para ser utilizada como resposta;
2. Ler os valores;
3. Efetuar o cálculo, implicando o resultado em R;
4. Apresentar o valor calculado contido na variável R;
5. Perguntar ao usuário se deseja continuar executando o programa;
6. Repetir os passos 2, 3, 4 e 5 até que a resposta do usuário seja não.

Programa em Pascal

```
program LOOPING_2B;
var
  A, B, R : integer;
  RESP : string;
begin
  RESP := 'SIM';
  repeat
    write('Entre um valor para A: '); readln(A);
    write('Entre um valor para B: '); readln(B);
    writeln;
    R := A + B;
    writeln('O resultado corresponde a: ', R);
    write('Deseja continuar? SIM/NAO: '); readln(RESP);
    writeln;
  until (RESP <> 'SIM') and (RESP <> 'sim');
end.
```

Antes de digitar este programa, é adequado fechar a janela ativa. Para tanto, execute o comando **Window/Close**. Em seguida abra uma nova janela com o comando **File/New** e digite o programa em questão, gravando-o com o nome LOOP_2B.

Assim que o programa é executado, a variável *RESP* é inicializada com o valor aSIM'. Em seguida, a instrução **repeat** indica que todo trecho de instruções situado até a instrução **until** deve ter o seu processamento repetido até que a sua condição se torne satisfeita, ou seja, torne-se verdadeira quando o usuário der como resposta **NÃO**.

2º Exemplo

Utilizando a instrução **repeat**, é apresentada em seguida uma segunda versão do programa para cálculo da fatorial de um número qualquer, o qual deve ser gravado com o nome FATOR_B. Antes de digitar o programa proposto, execute o comando **Window/Close** para fechar a janela ativa. Em seguida abra uma nova janela com o comando **File/New**.

Programa em Pascal

```
program FATORIAL_B;
var
  CONTADOR, N : integer;
  FATORIAL : longint;
begin
  FATORIAL := 1;
  CONTADOR := 1;
  writeln('Programa Fatorial');
  writeln;
  write('Fatorial de que numero: '); readln(N);
  writeln;
```

```
  repeat
    FATORIAL := FATORIAL * CONTADOR;
    CONTADOR := CONTADOR + 1;
  until (CONTADOR > N);
  writeln('Fatorial de ', N, ' equivale a ', FATORIAL);
  writeln;
  write('Tecle <ENTER> para encerrar: '); readln;
end.
```

5.4 - REPETIÇÃO COM VARIÁVEL DE CONTROLE

Nos tópicos 5.1 e 5.3, foram apresentadas duas formas de elaborar loopings. Uma usando o conceito **while** e a outra usando o conceito **repeat**. Com estas duas técnicas é possível elaborar rotinas que efetuam a execução de um looping um determinado número de vezes com a utilização de um contador (variáveis que executam um determinado número de vezes) ou mesmo com uma variável que aguarde a resposta do usuário. Independentemente da forma de tratamento, essa variável é denominada variável de controle.

Existe uma outra forma que visa facilitar o uso de contadores finitos, sem fazer uso das estruturas anteriores. Desta forma, os loopings com **while** e **repeat** passam a ser utilizados em loopings em que não se conhece de antemão o número de vezes que uma determinada seqüência de instruções deve ser executada.

Os loopings que possuem um número finito de execuções podem ser processados pelo looping do tipo: **for**, que tem seu funcionamento controlado por variável de controle do tipo contador, podendo ser crescente ou decrescente, tendo como sintaxe para looping crescente:

```
for <variável> := <inicio> to <fim> do
  <instruções>
```

ou, tendo como sintaxe para looping decrescente:

```
for <variável> := <inicio> downto <fim> do
  <instruções>;
```

Caso venha a existir mais de uma instrução para ser executada dentro do looping, elas devem estar inseridas em um bloco de instruções **begin** e **end**:

```
for <variável> := <inicio> to <fim> do
  begin
    <instruções1>;
    <instruções2>;
    <instruçõesN>;
  end;
```

ou:

```
for <variável> := <inicio> downto <fim> do
  begin
```

```
    <instruções1>;
    <instruções2>;
    <instruçõesN>;
  end;
```

1º Exemplo

Para exemplificar a utilização deste tipo de estrutura de looping, será considerado o exemplo anterior: "o programa deve pedir a leitura de dois valores para as variáveis *A* e *B*, efetuar a adição dos dois valores e implicar o resultado na variável de resposta *R* e em seguida apresentar o valor do resultado obtido, repetindo esta seqüência por cinco vezes".

Algoritmo

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

Programa em Pascal

```
program LOOPING_3A;
var
  A, B, R, I : integer;
begin
  for I := 1 to 5 do
    begin
      write('Entre um valor para A: '); readln(A);
      write('Entre um valor para B: '); readln(B);
      writeln;
      R := A + B;
      writeln('O resultado corresponde a: ', R);
      writeln;
    end;
end.
```

Antes de digitar este programa, é adequado fechar a janela ativa. Para tanto, execute o comando **Window/Close**. Em seguida abra uma nova janela com o comando **File/New** e digite o programa em questão, gravando-o com o nome LOOP_3A.

Quando executado o programa, o conjunto de instruções situado abaixo da instrução **for** será executado durante cinco vezes, pois a variável I (variável de controle) inicializada com valor 1 é incrementada de mais 1 a cada vez que o processamento passa pela linha da instrução **for**. Este tipo de estrutura de repetição pode ser utilizado todas as vezes que for

necessário repetir trechos finitos, ou seja, quando se conhecem o valor inicial e o valor final de uma variável tipo contador.

2º Exemplo

Utilizando a instrução **for**, é apresentada em seguida uma terceira versão do programa para cálculo da fatorial de um número qualquer, o qual deve ser gravado com o nome FATOR_C. Antes de digitar o programa proposto, execute o comando **Window/Close** para fechar a janela ativa. Em seguida, abra uma nova janela com o comando **File/New**.

Programa em Pascal

```
program FATORIAL_C;
var
  CONTADOR, N : integer;
  FATORIAL : longint;
begin
  FATORIAL := 1;
  writeln('Programa Fatorial');
  writeln;
  write('Fatorial de que numero: '); readln(N);
  writeln;
  for CONTADOR := 1 to N do
    begin
      FATORIAL := FATORIAL * CONTADOR;
    end;
  writeln('Fatorial de ', N, ' equivale a ', FATORIAL);
  writeln;
  write('Tecle <ENTER> para encerrar: '); readln;
end.
```

5.5 - Consideração entre as Estruturas de Looping

No decorrer deste capítulo, foram apresentadas três estruturas de controle em nível de repetição: **while**, **repeat** e **for**, cada qual com sua característica de processamento. Dentro deste aspecto, você deve ter notado que as estruturas mais versáteis são **while** e **repeat**, pois podem ser substituídas uma pela outra, além de poderem substituir perfeitamente a estrutura **for**. Porém, a estrutura **for** possibilita um código de programa menor e mais rápido que as outras duas, somente não podendo ser usado em condições que envolvam respostas lógicas do tipo **SIM** ou **NÃO**.

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

5.6 - EXERCÍCIO

Desenvolva os respectivos programas para os problemas indicados em seguida. Usar na resolução dos problemas as estruturas de loopings **while**, **repeat** e **for** (pode ocorrer de algum exercício não poder ser resolvido com **for**; esteja atento):

a. 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 de repetição a verificação lógica desta condição com a instrução **if**, perguntando se o número em foco é ímpar. Sendo, mostre-o; não sendo, passe para o próximo passo, até chegar ao final do laço.

b. Apresentar o total da soma obtida dos cem primeiros números inteiros (1+2+3+4+5+6+7+...97+98+99+100).

c. Apresentar os resultados de uma tabuada para um número qualquer.

d. Ler um número N qualquer maior ou igual a 1 e menor ou igual a 50 e apresentar o valor obtido da multiplicação sucessiva de N por 2, enquanto o produto for menor que 250 (N*2; N*2*2; N*2*2*2; etc.). O valor N deve ser checado quanto a sua validade, caso o usuário informe um valor fora da faixa. O programa deve informar por meio de uma mensagem que o valor digitado não é válido.

e. Apresentar todos os números divisíveis por 4 que sejam menores que 200.

f. Apresentar os quadrados dos números inteiros de 15 a 200.

g. Apresentar as potências de 3, variando de 0 a 15. Deve ser considerado que qualquer número elevado a zero é 1, e elevado a 1 é ele próprio. Este problema deve ser solucionado com o uso de laço para calcular as potências. Em hipótese alguma utilize a forma exp(ln(base)*expoente).

h. 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 subseqüente.

Parte 3

Tabelas em Memórias

```
procedure CALC_ADICAO(A, B :
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Capítulo 6

Estruturas de Dados Homogêneas I

Neste capítulo, será apresentada uma técnica de programação que permite trabalhar com o agrupamento de várias informações dentro de uma mesma variável. Vale salientar que esse agrupamento ocorre obedecendo sempre ao mesmo tipo de dado, e é por esta razão que recebe a denominação de estrutura de dados homogênea.

O tipo de dado homogêneo recebe diversos outros nomes, tais como: variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, vetores, matrizes, tabelas em memória ou arrays. Neste livro, este tipo de estrutura será tratado com a denominação de matriz.

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 matriz unidimensional, sendo que 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 em variáveis simples.

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

Aluno	Nota1	Nota2	Nota3	Nota4	Media
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 oito médias de cada aluno. Para representar a média do primeiro aluno, será utilizada a variável MD1, para o segundo MD2 e assim por diante. Sendo assim, 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:

```
program MEDIA_TURMA;
var
  MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8 : real;
  SOMA, MEDIA : real;
begin
  SOMA := 0;
  readln(MD1, MD2, MD3, MD4, MD5, MD6, MD7, MD8);
  SOMA := MD1 + MD2 + MD3 + MD4 + MD5 + MD6 + MD7 + MD8;
  MEDIA := SOMA / 8;
  writeln(MEDIA:6:2);
end.
```

Perceba que para receber a média, foram utilizadas oito variáveis. Com a técnica de matrizes pode ser utilizada apenas uma variável com a capacidade de armazenar oito valores.

Antes de digitar o referido programa, feche as janelas que estejam ativas. Para tanto, execute o comando **Window/Close all**. Em seguida abra uma nova janela com o comando **File/New** e digite o programa, gravando-o com o nome MEDIA_1.

6.2 - OPERAÇÕES COM MATRIZES DO TIPO VETOR

Uma matriz de uma dimensão ou vetor é representada por seu nome, tamanho (dimensão) entre colchetes e seu tipo, tendo assim, a seguinte sintaxe:

```
<matriz> : array[<dimensão>] of <tipo de dado>;
```

em que:

`<matriz>`	-	o nome atribuído à matriz;
`<dimensão>`	-	o tamanho da matriz em número de elementos;
`<tipo de dado>`	-	o tipo de elemento armazenado (inteiro, real, etc.).

Uma variável somente pode conter um valor por vez. No caso das matrizes, elas podem armazenar mais de um valor por vez, pois são dimensionadas exatamente para este fim. Lembrando ainda que a manipulação dos elementos de uma matriz ocorre 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 oito alunos, ter-se-ia então uma única variável indexada (a matriz) contendo todos os valores das oito notas. Isto é 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, ou seja, a nota do aluno.

Tanto a entrada como a saída de dados manipuladas com uma matriz são processadas passo a passo, um elemento por vez. A instrução de leitura **read/readln** ou de escrita **write/writeln** deve ser seguida da variável mais o índice. Estes processos são executados com o auxílio de um looping.

1º Exemplo

Considerando a proposta anterior, segue o programa que fará em primeiro lugar a leitura da média de cada aluno, o cálculo da média da sala e em seguida a apresentação da média geral.

```
program MEDIA_TURMA;
var
  MD : array[1..8] of real;
  SOMA, MEDIA : real;
  I : integer;
begin
  SOMA := 0;
  writeln('Calculo de media escolar');
  writeln;
  for I := 1 to 8 do
    begin
      write('Informe a ', I, 'a. nota: '); readln(MD[I]);
      SOMA := SOMA + MD[I];
    end;
  MEDIA := SOMA / 8;
  writeln;
  writeln('A media do grupo equivale a: ', MEDIA:6:2);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome MEDIA_2.

O programa em questão faz referência na área de declaração de variáveis à matriz *MD* : **array**[1..8] **of real**, em que é definido o número de elementos, no caso oito elementos, iniciando com índice 1 e estendendo até o índice 8.

2º Exemplo

Desenvolver um programa que efetue a leitura de dez elementos de uma matriz A tipo vetor. Construir uma matriz B de mesmo tipo, acompanhando a seguinte lei de formação: se o valor do *índice* for par, o valor deve ser multiplicado por 5; sendo ímpar, deve ser somado com 5. Ao final, mostrar os conteúdos das duas matrizes. Este exemplo demonstra como fazer o tratamento da condição do índice.

Algoritmo

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, multiplicar por 5, se não, somar 5. Criar a matriz B;
4. Apresentar o conteúdo das duas matrizes.

Programa em Pascal

```
program CHECK_INDICE;
var
  A, B : array[1..10] of integer;
  I, R : integer;
begin
  writeln('Calculo com checagem do indice da matriz');
  writeln;
  { *** Entrada dos dados *** }
  for I := 1 to 10 do
    begin
      write('Informe o ', I:2, 'o. valor: ');
      readln(A[I]);
    end;
  { *** Processamento par ou impar *** }
  for I := 1 to 10 do
    begin
      R := I mod 2;
      if (R = 0) then
         B[I] := A[I] * 5
      else
         B[I] := A[I] + 5;
    end;
  writeln;
  { *** Apresentacao das matrizes *** }
  for I := 1 to 10 do
    writeln('A[',I:2,'] = ',A[I]:2,'    ','B[',I:2,'] = ',B[I]:2);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome CHKIND.

No programa anterior, são utilizados três looping do tipo **for**: o primeiro looping controla a entrada dos dados, o segundo looping verifica se cada elemento da matriz A é par ou ímpar e faz as operações, implicando os elementos calculados na matriz B, o terceiro looping é utilizado para apresentar as duas matrizes.

No looping de processamento, é utilizada a instrução **if** (*R* = 0) **then**, sendo a variável R carregada com o resultado do operador aritmético **MOD** da linguagem Pascal que possibilita extrair o resto de uma divisão de números inteiros. Qualquer valor dividido por 2 que resultar zero caracteriza-se como par; se o resto for diferente de zero, o valor é ímpar.

Um detalhe utilizado neste exemplo são os comentários colocados entre as chaves. Comentários destes tipos servem para documentar o programa, facilitando a interpretação de um determinado trecho, sendo que eles não são compilados pelo Turbo Pascal.

3º 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. 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 se analise a condição do *elemento* e não do índice. Já foi alertado anteriormente para tomar cuidado e não confundir elemento com índice. Veja a solução.

Algoritmo

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 de todos os elementos ímpares da matriz.

Programa em Pascal

```
program ELEMENTO_IMPAR;
var
  A          : array[1..5] of integer;
  I, SOMA, R : integer;
begin
  SOMA := 0;
  writeln('Somatorio de elementos impares');
  writeln;
  { *** Entrada dos dados *** }
  for I := 1 to 5 do
    begin
      write('Informe o ', I:2, 'o. valor: ');
      readln(A[I]);
    end;
  { *** Processamento par ou impar *** }
  for I := 1 to 5 do
    begin
      R := A[I] mod 2;
      if (R <> 0) then
        SOMA := SOMA + A[I];
    end;
  writeln;
  { *** Apresentacao do somatorio *** }
  writeln('A soma dos elementos impares equivale a: ', SOMA:2);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome CHKELE.

No looping de processamento, é utilizada a instrução **if** (*R* <> 0) **then**, para verificar se o elemento informado pelo teclado é um valor ímpar; sendo, ele é acumulado na variável *SOMA* que ao final apresenta o somatório de todos os elementos ímpares digitados durante a execução do programa.

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

6.3 - Aplicações Práticas

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 um programa que necessite ler e apresentar os nomes de dez pessoas. O programa em questão fará a leitura e em seguida a escrita dos dez nomes.

Algoritmo

1. Definir a variável I do tipo inteira para controlar a malha de repetição;
2. Definir a matriz NOME do tipo caractere para 10 elementos;
3. Iniciar o programa, fazendo a leitura dos 10 nomes;
4. Apresentar, após a leitura, os 10 nomes.

Programa em Pascal

```
program LISTA_NOME;
var
  NOME : array[1..10] of string;
  I : integer;
begin
  writeln('Listagem de nomes');
  writeln;
  { *** Entrada dos dados *** }
  for I := 1 to 10 do
    begin
      write('Digite o ', I:2 ,'o. nome: '); readln(NOME[I]);
    end;
  writeln;
  { *** Apresentacao dos nomes *** }
  for I := 1 to 10 do
    writeln('Nome: ' ,I:2, ' --> ', NOME[I]);
  writeln;
```

```
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome LISTNOME.

Observe que ao término da execução do programa, ocorre a apresentação dos nomes na mesma ordem em que foram informados.

6.3.1 - Classificação de Elementos

Tendo construído o programa de entrada e saída dos dez 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 a localização de algum nome, quando for efetuada uma pesquisa visual.

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 no tópico anterior.

Algoritmo

1. Definir a variável I do tipo inteira para controlar a malha de repetição;
2. Definir a matriz NOME do tipo caractere para 10 elementos;
3. Colocar em ordem crescente os elementos da matriz;
4. Iniciar o programa, fazendo a leitura dos 10 nomes;
5. Apresentar, após a leitura, os 10 nomes.

É importante que se estabeleça adequadamente o terceiro passo do algoritmo, ou seja, como fazer para colocar os elementos em ordem crescente. Imagine um problema um pouco menor:

Colocar em ordem crescente cinco valores numéricos armazenados numa matriz A e apresentá-los. Considere para esta explicação os valores apresentados na tabela seguinte:

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 devem 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, e sendo assim, o elemento que estiver em A[1] deve 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] deve ser comparado com o próximo valor após a sua última comparação. Sendo assim, ele deve 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 devem 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]. São estes 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 necessita mais ser comparado. Assim sendo, deve 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] deve 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:

```
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 subseqüentes, 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, elas terão valores diferentes.

Cada caractere é guardado na memória de um computador segundo o valor de um código que recebe o nome de ASCII (ver Apêndice B). E é com base nessa tabela que o processo de ordenação trabalha, pois cada caractere tem um peso, um valor previamente determinado, segundo este padrão.

Observe a seguir, a codificação da ordenação no programa. Atente para o detalhe das instruções **for** encadeadas. Vale salientar 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 em Pascal

```
program LISTA_NOME_ORDENADA;
var
  NOME : array[1..10] of string;
  I, J : integer;
  X    : string;
  {*** Rotina de entrada de dados ***}
begin
  writeln('Listagem de nomes');
  writeln;
  for I := 1 to 10 do
    begin
      write('Digite o ', I:2 ,'o. nome: '); readln(NOME[I]);
    end;
  {*** Rotina de processamento de ordenacao ***}
  for I := 1 to 9 do
    for J := I + 1 to 10 do
      if (NOME[I] > NOME[J]) then
         begin
           X := NOME[I];
           NOME[I] := NOME[J];
           NOME[J] := X;
  end;
  {*** Rotina de saida com dados ordenados ***}
  writeln;
  for I := 1 to 10 do
    writeln('Nome: ' ,I:2, ' --> ', NOME[I]);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome SORTNOME. Atente para dois detalhes no programa.

O primeiro detalhe a ser observado é a utilização de uma segunda variável no processo de ordenação, no caso, a variável **J**. Observe que somente quando a variável **J** atinge o valor 10 é que esse looping se encerra, retornando ao looping da variável **I**, acrescentando mais um a **I** até que **I** atinja o seu limite e ambos os loopings sejam encerrados. O quadro seguinte mostra o comportamento dos valores das variáveis **I** e **J** durante o processo de ordenação:

Quando I for	J será
1	2, 3, 4, 5, 6, 7, 8, 9, 10
2	3, 4, 5, 6, 7, 8, 9, 10
3	4, 5, 6, 7, 8, 9, 10
4	5, 6, 7, 8, 9, 10
5	6, 7, 8, 9, 10

Quando I for	J será
6	7, 8, 9, 10
7	8, 9, 10
8	9, 10
9	10

Quando a variável I for 1, a variável J será 2 e contará até 10. Ao final deste ciclo, é acrescentado mais 1 à variável I, tornando-se 2. Assim sendo, a variável J passa a ser 3. Quando a variável J voltar a ser 10 novamente, a variável I passa a ser 3 e a variável J passa a ser 4. Este ciclo será executado até que por fim a variável I seja 9 e a variável J seja 10, sendo comparado o penúltimo elemento com o seu elemento subseqüente, no caso, o último.

O segundo ponto a ser observado é o fato da utilização do algoritmo de troca, utilizado junto da instrução **if** (*NOME[I]* > *NOME[J]*) **then**. Após a verificação desta condição, sendo o primeiro nome maior que o segundo, efetua-se então a sua troca com a seqüência:

```
      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] deve estar com "ALBERTO" e NOME[J] deve estar com "CARLOS". Para conseguir este efeito, é necessária a utilização de uma variável de apoio, a qual será chamada X.

Para que o vetor NOME[I] fique livre para receber o valor do vetor NOME[J], X deve 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".

6.3.2 - Métodos de Pesquisa

A utilização de matrizes pode gerar grandes tabelas, tornando-se difícil 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.

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, esta é

apresentada. Este método de pesquisa é lento, porém eficiente nos casos em que uma matriz encontra-se com seus elementos desordenados.

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, 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	Sílvia
6	Sílvio
7	Waldir

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

O processo binário consiste em pegar o número de registro e dividir por 2. Sendo assim, 7/2 é igual a 3,5, e o que nos interessa é somente o valor inteiro, no caso 3. Então, a tabela fica dividida em duas partes, como a seguir:

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	Sílvio
7	Waldir

Estando a tabela dividida em duas partes, deve 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 quatro elementos dividido por 2, resultam duas tabelas, cada uma com dois elementos, conforme apresentado em seguida:

Primeira Parte Após Segunda Divisão

Índice	Nomes
4	Golias
5	Sílvia

Segunda Parte Após Segunda Divisão

Índice	Nomes
6	Sílvio
7	Waldir

Após esta segunda divisão, verifica-se em qual das partes Waldir está situado. Constatando 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	Sílvio

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, ele não será apresentado por não existir.

1º Exemplo

Considerando a necessidade de trabalhar com uma matriz com dez nomes, segue o programa para efetuar uma pesquisa seqüencial na referida matriz.

Algoritmo

O algoritmo seguinte estabelece a entrada de dez 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, deve ser solicitada a informação a ser pesquisada;

   Essa informação deve 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, deve mostrá-lo;
3. Encerrar a pesquisa quando desejado.

Programa em Pascal

```
program PESQUISA_SEQUENCIAL;
var
  NOME : array[1..10] of string;
  I    : integer;
  PESQ : string;
  RESP : string;
  ACHA : boolean;
begin
  writeln('Pesquisa sequencial de nomes');
  writeln;
  for I := 1 to 10 do
    begin
      write('Digite o ', I:2 ,'o. nome: '); readln(NOME[I]);
    end;
  RESP := 'SIM';
  while (RESP = 'SIM') or (RESP = 'sim') do
    begin
      {*** Rotina de pesquisa ***}
      writeln;
      write('Entre o nome a ser pesquisado: '); readln(PESQ);
```

```
      I := 1;
      ACHA := false;
      while (I <= 10) and (ACHA = false) do
        if (PESQ = NOME[I]) then
          ACHA := true
        else
          I := I + 1;
      if (ACHA = true) then
        writeln(PESQ, ' foi localizado na posicao ', I:2)
      else
        writeln(PESQ, ' nao foi localizado');
      {*** Fim da rotina de pesquisa ***}
      writeln;
      write('Deseja continuar? SIM/NAO: ');
      readln(RESP);
    end;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome PESQSEQ.

A seguir, é apresentado o trecho responsável pela execução da pesquisa seqüencial definida no programa anterior.

```
 1 - write('Entre o nome a ser pesquisado: '); readln(PESQ);
 2 - I := 1;
 3 - ACHA := false;
 4 - while (I <= 10) and (ACHA = false) do
 5 -   if (PESQ = NOME[I]) then
 6 -     ACHA := true
 7 -   else
 8 -     I := I + 1;
 9 - if (ACHA = true) then
10 -   writeln(PESQ, ' foi localizado na posicao ', I:2)
11 - else
12 -   writeln(PESQ, ' nao foi localizado');
13 - writeln;
14 - write('Deseja continuar? SIM/NAO: ');
15 - readln(RESP);
```

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 **while** indicando que enquanto o valor da variável I for menor ou igual a 10 e simultaneamente o valor da variável *ACHA* for falso, deve ser processado o conjunto de instruções situado nas linhas: 5, 6, 7, 8 e 9.

Neste ponto, a instrução **if** 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 (linha 6), forçando a execução da linha 10. Se o valor da linha 9 é verdadeiro, é apresentada a mensagem da linha 10.

Caso na linha 5 seja verificado que o valor de *PESQ* não é igual a *NOME*[1], será então incrementado 1 à variável I. Será executada a próxima verificação de *PESQ* com *NOME*[2], e assim por diante. Caso o processamento chegue até o final e não seja encontrado nada, a variável *ACHA* permanece com valor falso. Quando analisada pela linha 9, será então falsa e apresentará a mensagem da linha 12.

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, ao começar a rotina, a bandeira está "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 dez nomes, segue o programa para efetuar uma pesquisa binária na referida matriz.

Algoritmo

O algoritmo abaixo indicado estabelece a entrada de dez 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 looping que efetue a pesquisa enquanto o usuário assim o desejar;

   Durante a fase de pesquisa, deve ser solicitada a informação a ser pesquisada;

   Essa informação deve ser comparada, utilizando 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, deve mostrá-lo;
3. Encerrar a pesquisa quando desejado.

Programa em Pascal

```
program PESQUISA_BINARIA;
var
  NOME : array[1..10] of string;
  I, J : integer;
  COMECO, FINAL, MEIO : integer;
  PESQ, RESP, X : string;
  ACHA : boolean;
begin
  writeln('Pesquisa binaria de nomes');
```

```
  writeln;
  for I := 1 to 10 do
    begin
      write('Digite o ', I:2 ,'o. nome: '); readln(NOME[I]);
    end;
  {*** Ordenacao ***}
  for I := 1 to 9 do
    for J := I + 1 to 10 do
      if (NOME[I] > NOME[J]) then
         begin
           X := NOME[I];
           NOME[I] := NOME[J];
           NOME[J] := X;
  end;
  {*** Trecho de pesquisa ***}
  RESP := 'SIM';
  while (RESP = 'SIM') or (RESP = 'sim') do
    begin
      writeln;
      write('Entre o nome a ser pesquisado: '); readln(PESQ);
      COMECO:= 1;
      FINAL := 10;
      ACHA := false;
      while (COMECO <= FINAL) and (ACHA = false) do
        begin
          MEIO := (COMECO + FINAL) div 2;
          if (PESQ = NOME[MEIO]) then
            ACHA := true
          else
            if (PESQ < NOME[MEIO]) then
                 FINAL := MEIO - 1
            else
                 COMECO := MEIO + 1;
        end;
      if (ACHA = true) then
        writeln(PESQ, ' foi localizado na posicao ', MEIO:2)
      else
        writeln(PESQ, ' nao foi localizado');
      writeln;
      write('Deseja continuar? SIM/NAO: ');
      readln(RESP);
    end;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome PESQBIN.

A seguir, é apresentado o trecho responsável pela execução da pesquisa binária definida no programa anterior.

```
1 -  write('Entre o nome a ser pesquisado: '); readln(PESQ);
2 -  COMECO:= 1;
```

```
 3 -  FINAL := 10;
 4 -  ACHA := false;
 5 -  while (COMECO <= FINAL) and (ACHA = false) do
 6 -    begin
 7 -      MEIO := (COMECO + FINAL) div 2;
 8 -      if (PESQ = NOME[MEIO]) then
 9 -        ACHA := true
10 -      else
11 -        if (PESQ < NOME[MEIO]) then
12 -           FINAL := MEIO - 1
13 -        else
14 -           COMECO := MEIO + 1;
15 -    end;
16 -  if (ACHA = true) then
17 -    writeln(PESQ, ' foi localizado na posicao ', MEIO:2)
18 -  else
19 -    writeln(PESQ, ' nao foi localizado');
```

Na linha 1, é solicitado que seja informado o dado a ser pesquisado. As linhas 2 e 3 inicializam as variáveis de controle *COMECO* com 1 e *FINAL* com 10. A linha 4 inicializa o flag *ACHA*. A linha 5 apresenta a instrução que manterá a pesquisa em execução enquanto o *COMECO* for menor ou igual ao *FINAL* e, simultaneamente, o flag *ACHA* seja falso. Assim sendo, o processamento dividirá a tabela ao meio, conforme instrução na linha 7, sendo que 1 é o começo da tabela e é somado com 10 que é o fim da tabela, resultando no total 11 que dividido por 2 resulta 5,5. Observe a utilização da função DIV, que tem por finalidade considerar a parte inteira do valor da divisão, no caso 5, que é o meio da tabela.

Neste ponto, a tabela está dividida em duas partes. A instrução da linha 8 verifica se o valor fornecido para *PESQ* é igual ao valor armazenado na posição *NOME*[5]. Se for, o flag é setado como verdadeiro, sendo em seguida apresentada a mensagem da linha 17. Se a condição de busca não for igual, pode ocorrer uma de duas situações.

Na primeira situação, 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 11). Neste caso, a variável *FINAL* deve ser implicada pelo valor da variável *MEIO* subtraído de 1 (linha 12), ficando a variável *FINAL* com valor 4. Se for esta a situação ocorrida, será processada a linha 5 que efetuará novamente o looping, pelo fato de o valor 1 da variável *COMECO* ser menor ou igual ao valor 4 da variável *FINAL*. A instrução da linha 7 divide a primeira parte da tabela em mais duas partes. Desta forma, é somado com 4, resultando 5, e dividido por 2 resulta 2 (sendo considerada a parte inteira do resultado da divisão), que é o meio da primeira parte da tabela.

A segunda situação pode 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 11). Neste caso, a variável *COMECO* deve ser implicada pelo valor da variável *MEIO* somado com 1 (linha 14), ficando a variável *COMECO* com valor 6. Se for esta a situação ocorrida, será processada a linha 5 que efetuará novamente o looping, pelo fato de o valor 6 da variável COMECO ser menor ou igual ao valor 10 da variável *FINAL*. A instrução da linha 7 divide a

segunda parte da tabela em mais duas partes. Desta forma, 6 é somado com 10, resultando 16, que dividido por 2 resulta 8, que é o meio da segunda parte da tabela.

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

6.4 - EXERCÍCIO

Desenvolva os programas dos seguintes problemas:

a. Ler dez elementos de uma matriz tipo vetor.

b. Ler oito elementos de uma matriz A tipo vetor. Construir uma matriz B de mesma dimensão com os seus elementos multiplicados por 3. Apresentar a matriz B.

c. Ler uma matriz A do tipo vetor com quinze 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 a matriz B.

d. Ler duas matrizes A e B do tipo vetor com vinte elementos. Construir uma matriz C, sendo que cada elemento é a subtração do elemento correspondente de A com B. Apresentar a matriz C.

e. Ler duas matrizes A e B do tipo vetor com quinze elementos cada. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deve ter o dobro de elementos de A e B. Apresentar a matriz C.

f. Ler duas matrizes do tipo vetor, sendo A com vinte elementos e B com trinta elementos. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deve ter a capacidade de armazenar cinqüenta elementos. Apresentar a matriz C.

g. Ler quinze elementos de uma matriz A do tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: todo elemento da matriz B deve ser o quadrado do elemento de A correspondente. Apresentar a matriz B.

h. Ler vinte elementos de uma matriz A tipo vetor e construir uma matriz B de mesma dimensão com os mesmos elementos de A, sendo que estes devem 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 duas matrizes.

i. Ler doze elementos de uma matriz tipo vetor, colocá-los em ordem decrescente e apresentar os elementos ordenados.

j. Ler oito elementos de uma matriz A tipo vetor. Construir uma matriz B de mesma dimensão com os elementos da matriz multiplicados por 5. Apresentar a matriz B na ordem crescente. Montar uma rotina de pesquisa binária, para pesquisar os elementos armazenados na matriz B.

k. Ler uma matriz A do tipo vetor com quinze 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.

l. Ler uma matriz A com doze elementos. Após a leitura, colocar os seus elementos em ordem crescente. Depois ler uma matriz B também com doze elementos e colocar seus elementos em ordem crescente. Construir uma matriz C, sendo cada elemento de C a soma do elemento correspondente de A com B. Colocar em ordem crescente a matriz C e apresentar os seus valores.

m. Ler duas matrizes do tipo vetor, sendo A com vinte elementos e B com trinta elementos. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deve ter a capacidade de armazenar cinqüenta elementos. Apresentar os elementos da matriz C em ordem decrescente.

n. Ler trinta 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 deve ser o cubo do elemento de A correspondente. Montar uma rotina de pesquisa seqüencial, para pesquisar os elementos armazenados na matriz B.

o. Ler vinte elementos de uma matriz A tipo vetor e construir uma matriz B de mesma dimensão com os mesmos elementos de A com acréscimo 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.

Capítulo 7

Estruturas de Dados Homogêneas II

No capítulo anterior, 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 estarmos ainda utilizando uma estrutura de dados homogênea, todos os elementos de uma matriz devem ser do mesmo tipo.

7.1 - Matrizes com mais de Uma Dimensão

Anteriormente, você teve 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 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 se considerar que o trabalho é grande, uma vez que é necessário manter um controle de cada índice em cada matriz para um mesmo aluno.

Para facilitar o trabalho com estruturas deste porte, serão utilizadas matrizes com mais dimensões. A mais comum é a matriz de duas dimensões por relacionar-se diretamente com a utilização de tabelas. Matrizes com mais de duas dimensões são utilizadas com menos freqüência, mas podem ocorrer momentos em que é preciso trabalhar com um número maior de dimensões, porém estas são fáceis de ser utilizadas se você dominar 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. No caso de matrizes com mais dimensões,

deve ser utilizado o número de loopings relativo ao tamanho de sua dimensão. Desta forma, uma matriz de duas dimensões deve ser controlada com dois loopings, sendo que de três dimensões deve ser controlada por três loopings e assim por diante.

Em matrizes de duas dimensões, 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.

7.2 - 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], sendo que 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 quarenta elementos. A figura 7.1 apresenta a matriz TABELA com a indicação dos endereços (posições) que podem ser utilizados para armazenamento de seus elementos.

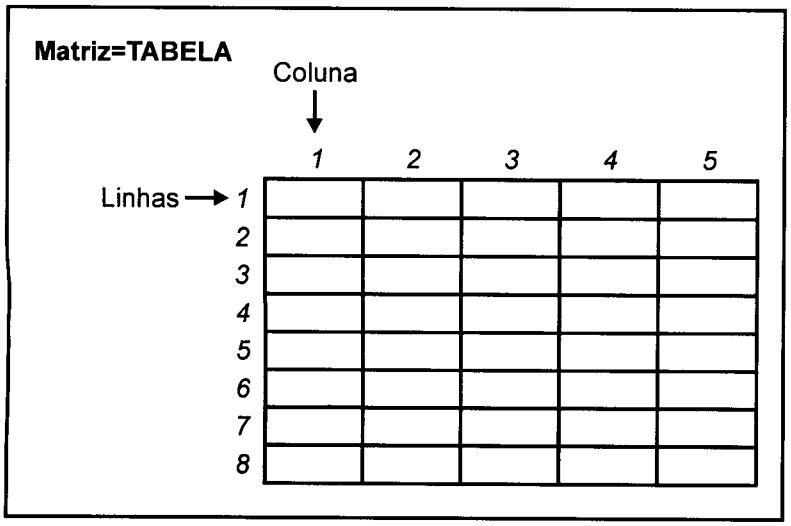


Figura 7.1 - Exemplo da matriz TABELA com suas posições.

Uma matriz de duas dimensões é atribuída de forma semelhante a uma matriz de uma dimensão, sendo representada pelo seu nome, tamanho (dimensão de linhas e colunas entre colchetes) e seu tipo, tendo assim, a seguinte sintaxe:

```
<matriz> : array[<dimensão linha>,<dimensão coluna>] of <tipo de dado>;
```

em que:

`<matriz>`	-	o nome atribuído à matriz;
`<dimensão linha>`	-	o tamanho da matriz em número de linhas;
`<dimensão coluna>`	-	o tamanho da matriz em número de colunas;
`<tipo de dado>`	-	o tipo de elemento armazenado (inteiro, real, etc.).

1º Exemplo

Para exemplificar a utilização de matrizes deste tipo, considere o programa de entrada e saída das notas escolares. A leitura e a escrita de uma matriz de duas dimensões, assim como das matrizes de uma dimensão, são processadas passo a passo.

Considerando a manipulação de quatro notas de oito alunos, a tabela em questão armazena trinta e dois elementos. Um detalhe a ser considerado é a utilização de duas variáveis para controlar os dois índices de posicionamento de dados na tabela.

```
program NOTA_ALUNO;
var
  NOTAS : array[1..8,1..4] of real;
  I, J  : integer;
begin
  writeln('Leitura e apresentacao de notas');
  writeln;
  for I := 1 to 8 do
    begin
      writeln;
      writeln('Entre as notas do ', I:2,'o. aluno');
      for J := 1 to 4 do
        begin
          write('Nota ', J:2,': ');
          readln(NOTAS[I,J]);
        end;
    end;
  writeln;
  for I := 1 to 8 do
    begin
      write('As notas do aluno ', I:2,' sao: ');
      for J := 1 to 4 do

        write(NOTAS[I,J]:5:2, ' ');
      writeln;
    end;
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome NOTAS.

Em um exemplo anterior, foi utilizada a variável *I* para controlar as posições dos elementos dentro da matriz, ou seja, a posição em nível de linha. Neste exemplo, a variável *I* continua tendo o mesmo efeito e a segunda variável, a *J*, está controlando a posição da coluna.

Analisando o programa, 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 digitada 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 digitada 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 assim, 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. Em seguida é apresentada a relação dos alunos com as respectivas notas.

2º Exemplo

Desenvolver um programa de agenda que cadastre o nome, endereço, cep, bairro e telefone de dez pessoas. Ao final, o programa deve apresentar os seus elementos dispostos em ordem alfabética, independentemente da forma em que foram digitados.

Algoritmo

Para resolver este problema, você deve possuir uma tabela com dez linhas (pessoas) e cinco 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 cinco colunas, uma para cada informação pessoal e o número de linha, totalizando um conjunto de dez informações.

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

Depois de cadastrar todos os elementos, será 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, eles devem ser trocados, mas os elementos relacionados ao nome também devem ser trocados no mesmo nível de verificação, ficando para o final o trecho de apresentação de todos os elementos.

Programa em Pascal

```
program AGENDA;
var
  DADO : array[1..10,1..5] of string;
  I, J, ATUAL, PROXIMO : integer;
  X    : string;
begin
  {*** Rotina de entrada ***}
  writeln('Programa agenda');
  writeln;
  for I := 1 to 10 do
    begin
      write('Nome .....: '); readln(DADO[I,1]);
      write('Endereco .: '); readln(DADO[I,2]);
      write('CEP ......: '); readln(DADO[I,3]);
      write('Bairro ...: '); readln(DADO[I,4]);
      write('Telefone .: '); readln(DADO[I,5]);
```

```
      writeln;
    end;
  {*** Rotina de ordenacao e troca de elementos ***}
  for ATUAL := 1 to 9 do
    for PROXIMO := ATUAL + 1 to 10 do
      if (DADO[ATUAL,1] > DADO[PROXIMO,1]) then
         begin
           {Troca Nome}
           X := DADO[ATUAL,1];
           DADO[ATUAL,1] := DADO[PROXIMO,1];
           DADO[PROXIMO,1] := X;
           {Troca Endereço}
           X := DADO[ATUAL,2];
           DADO[ATUAL,2] := DADO[PROXIMO,2];
           DADO[PROXIMO,2] := X;
           {Troca CEP}
           X := DADO[ATUAL,3];
           DADO[ATUAL,3] := DADO[PROXIMO,3];
           DADO[PROXIMO,3] := X;
           {Troca Bairro}
           X := DADO[ATUAL,4];
           DADO[ATUAL,4] := DADO[PROXIMO,4];
           DADO[PROXIMO,4] := X;
           {Troca Telefone}
           X := DADO[ATUAL,5];
           DADO[ATUAL,5] := DADO[PROXIMO,5];
           DADO[PROXIMO,5] := X;
         end;
  {*** Rotina de saida ***}
  for I := 1 to 10 do
    for J := 1 to 5 do
      writeln(DADO[I,J]);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome AGENDA.

No programa anterior, não estão sendo utilizados para a entrada de dados dois loopings. Note que as referências feitas ao endereço das colunas são citadas como constantes, durante a variação do valor da variável *I* que representa o controle da linha. Isto foi aplicado, uma vez que está sendo definida uma tela de entrada com mensagens, indicando o que deve ser digitado.

Com relação ao trecho de ordenação de elementos de uma matriz de duas dimensões, o processo é o mesmo utilizado para ordenar matrizes de uma dimensão. Observe que a troca de posição de todos os elementos ocorre quando os nomes comparados 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.

3º Exemplo

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

Algoritmo

Neste exemplo, é apresentado um problema cuja solução é 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, é 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:

	Nomes
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 deve 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 a respectiva média, bem como, a média do grupo.

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

Depois a instrução **for** *I* := 1 **to** 8 **do** 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. Essa variável guarda 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 vai 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 das suas respectivas médias. Ao final, a variável MEDIA_GP determina o cálculo da média do grupo (média das médias) por meio do valor armazenado na variável SOMA_MD dividido por 8 (total de alunos).

Programa em Pascal

```
program CALC_MEDIA;
var
  X : string;
  I, J, ATUAL, PROXIMO : integer;
  Y, SOMA_NT, SOMA_MD, MEDIA_GP : real;
  NOTA : array[1..8,1..4] of real;
  MEDIA : array[1..8] of real;
  NOMES : array[1..8] of string;
  {*** Rotina de entrada de dados ***}
begin
  SOMA_MD := 0;
  for I := 1 to 8 do
    begin
      SOMA_NT := 0;
      write('Digite o nome do ', I:2,'o. aluno: '); readln(NOMES[I]);
      for J := 1 to 4 do
        begin
          write(J:2,'a. nota: '); readln(NOTA[I,J]);
          SOMA_NT := SOMA_NT + NOTA[I,J];
        end;
      MEDIA[I] := SOMA_NT / 4;
      SOMA_MD := SOMA_MD + MEDIA[I];
      writeln;
  end;
  {*** Rotina de ordenacao e troca de elementos ***}
  for ATUAL :=1 to 7 do
    for PROXIMO := ATUAL + 1 to 8 do
      if (NOMES[ATUAL] > NOMES[PROXIMO]) then
         begin
           X := NOMES[ATUAL];
           NOMES[ATUAL] := NOMES[PROXIMO];
           NOMES[PROXIMO] := X;
           Y := MEDIA[ATUAL];
           MEDIA[ATUAL] := MEDIA[PROXIMO];
           MEDIA[PROXIMO] := Y;
        end;
  {*** Rotina de apresentacao ***}
  MEDIA_GP := SOMA_MD / 8;
  for I := 1 to 8 do
```

```
    begin
      writeln('Aluno .: ', NOMES[I]);
      writeln('Media .: ', MEDIA[I]:5:2);
      writeln;
    end;
  writeln('Media Geral : ', MEDIA_GP:5:2);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Antes de digitar o referido programa, feche as janelas que estejam ativas. Em seguida abra uma nova janela e digite o programa, gravando-o com o nome ESCOLAR.

7.3 - EXERCÍCIO

Desenvolva os programas dos seguintes exercícios:

a. Ler duas matrizes A e B, cada uma com duas dimensões com cinco linhas e três colunas. Construir uma matriz C de mesma dimensão, sendo que C é formada pela soma dos elementos da matriz A com os elementos da matriz B. Apresentar os elementos da matriz C.

b. Ler duas matrizes A e B, cada uma com uma dimensão para sete elementos. Construir uma matriz C de duas dimensões, sendo que a primeira coluna deve ser formada pelos elementos da matriz A e a segunda coluna, pelos elementos da matriz B. Apresentar os elementos da matriz C.

c. Ler vinte elementos para uma matriz qualquer, considerando que essa matriz tenha o tamanho de quatro linhas por cinco colunas.

d. Ler uma matriz A de uma dimensão com dez elementos. Construir uma matriz B de duas dimensões com três colunas, sendo que a primeira coluna da matriz B 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 deve ser formada pelos quadrados dos elementos correspondentes da matriz A. Apresentar os elementos da matriz B.

e. Ler duas matrizes A e B, cada uma com uma dimensão para doze elementos. Construir uma matriz C de duas dimensões, sendo que a primeira coluna da matriz C deve ser formada pelos elementos da matriz A multiplicados por 2, e a segunda coluna deve ser formada pelos elementos da matriz B subtraídos de 5. Apresentar os elementos da matriz C.

f. Ler duas matrizes A e B de duas dimensões, sendo a matriz A de 5 x 4 (cinco linhas por quatro colunas) e a matriz B de 4 x 3 (quatro linhas e três colunas). Apresentar a matriz C como resultante da multiplicação das matrizes A e B. Lembre-se de que existe uma lei matemática que rege a multiplicação de matrizes. Essa lei estabelece que uma multiplicação entre matrizes pode ocorrer

quando o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz, e o valor resultante da terceira matriz é o somatório da multiplicação de cada elemento da coluna da primeira matriz por cada elemento de linha da segunda matriz. Para determinar a matriz C que será uma matriz de 5 x 3, o programa deve então efetuar o cálculo da multiplicação das matrizes, obedecendo à lei matemática de multiplicação de matrizes. Assim sendo:

C[1,1] será o resultado de A[1,1]*B[1,1]+A[1,2]*B[2,1]+A[1,3]*B[3,1]+A[1,4]*B[4,1]

C[1,2] será o resultado de A[1,1]*B[1,2]+A[1,2]*B[2,2]+A[1,3]*B[3,2]+A[1,4]*B[4,2]

C[1,3] será o resultado de A[1,1]*B[1,3]+A[1,2]*B[2,3]+A[1,3]*B[3,3]+A[1,4]*B[4,3]

Acima, foi indicada a primeira linha da matriz resultante C. Sua função é fazer com que o programa faça o restante até obter o resultado do último elemento da matriz C.

Capítulo 8

Estruturas de Dados Heterogêneas

Anteriormente, você teve contato com a utilização de matrizes e notou que somente foi possível trabalhar com um tipo de dado por matriz. No momento em que foi preciso trabalhar com dois tipos de dados diferentes, foi necessária a utilização de duas matrizes, sendo uma de cada tipo.

8.1 - Estrutura de Dados com o Uso do Tipo Registro

Neste capítulo, você terá contato com a utilização da principal estrutura de dados, o registro, que consiste em trabalhar com vários dados de tipos diferentes (os campos) em uma mesma estrutura. Por esta razão, este tipo de dado é considerado heterogêneo.

Para tanto, considere que sejam informados o nome de um aluno e suas quatro notas bimestrais que devem ser agrupados em uma mesma estrutura. A figura 8.1 mostra um exemplo do layout de um registro, o qual é o conjunto de campos. Note que o registro está formado pelos campos: Nome, Primeira Nota, Segunda Nota, Terceira Nota e Quarta Nota, podendo ser denominado um registro de aluno.

Layout do formato de um registro com seus campos

Cadastro de Notas Escolares

Nome . . . . . : _____

Primeira Nota. . . . . .: _____
Segunda Nota . . . . . .: _____
Terceira Nota. . . . . .: _____
Quarta Nota . . . . . .: _____

Figura 8.1 - Exemplo da estrutura de um registro com seus campos.

Na estrutura da linguagem Pascal, os tipos registro devem ser declarados ou atribuídos antes das definições das variáveis, pois é muito comum ocorrer a necessidade de declarar uma variável com o tipo de registro atribuído. Um tipo registro é declarado em Pascal com a instrução **type** em conjunto com a instrução **record**, conforme a seguinte sintaxe:

```
type
  <identificador> = record
                      <lista dos campos e seus tipos>;
                    end;
var
  <variável> : <identificador>;
```

Em que *identificador* é o nome do tipo registro, que neste livro será escrito em caracteres maiúsculos, em itálico, seguindo as mesmas regras de definição das variáveis, e *lista dos campos e seus tipos* é a relação de variáveis que serão usadas como campos, bem como o seu tipo, podendo ser: real, integer, boolean, string, entre outros existentes no Turbo Pascal.

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

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

```
type
  CAD_ALUNO = record
                NOME  : string;
                NOTA1 : real;
                NOTA2 : real;
                NOTA3 : real;
                NOTA4 : real;
              end;
var
  ALUNO : cad_aluno;
```

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

Tanto a leitura quanto a escrita de um registro são efetuadas, respectivamente, com as instruções **read/readln** e **write/writeln** seguidas do nome da variável do tipo registro e de seu campo correspondente separado por um caractere "." (ponto), como exemplificado em seguida.

```
program LEITURA_ESCRITA;
type
  CAD_ALUNO = record
                NOME  : string;
                NOTA1 : real;
                NOTA2 : real;
```

```
            NOTA3 : real;
          NOTA4 : real;
          end;
var
  ALUNO : cad_aluno;
begin
  writeln('Cadastro de aluno');
  writeln;
  write('Informe o nome ...........: '); readln(ALUNO.NOME);
  write('Informe a primeira nota ..: '); readln(ALUNO.NOTA1);
  write('Informe a segunda nota ...: '); readln(ALUNO.NOTA2);
  write('Informe a terceira nota ..: '); readln(ALUNO.NOTA3);
  write('Informe a quarta nota ....: '); readln(ALUNO.NOTA4);
  writeln;
  writeln('Nome ....: ', ALUNO.NOME);
  writeln('Nota 1 ..: ', ALUNO.NOTA1:5:2);
  writeln('Nota 2 ..: ', ALUNO.NOTA2:5:2);
  writeln('Nota 3 ..: ', ALUNO.NOTA3:5:2);
  writeln('Nota 4 ..: ', ALUNO.NOTA4:5:2);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Feche as janelas que estejam abertas, em seguida abra uma nova janela e digite o programa anterior, gravando-o com o nome ALUNO1. Perceba que no registro definido existem quatro variáveis do mesmo tipo, definidas para armazenar quatro notas (NOTA1, NOTA2, NOTA3 e NOTA4). Desta forma, este registro pode ser definido usando uma matriz do tipo vetor para armazenar as notas. A figura 8.2 apresenta uma estrutura para esta proposta.

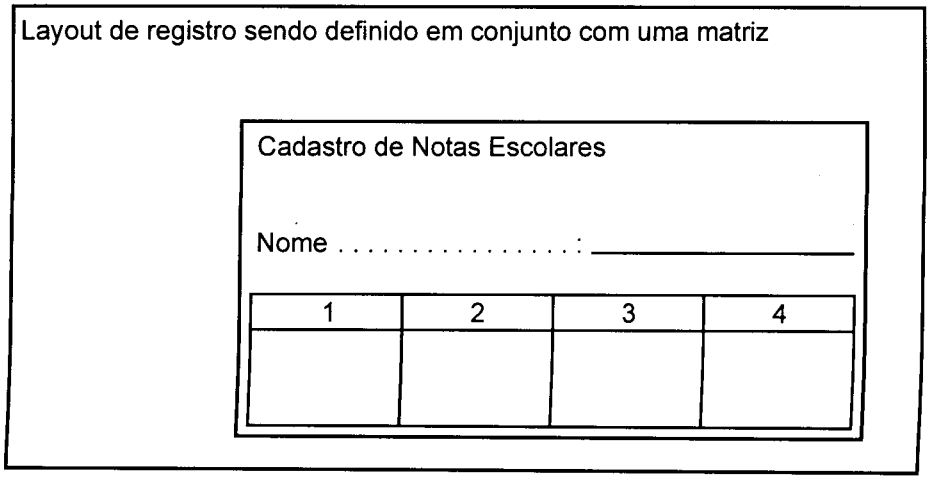


Figura 8.2 - Exemplo da estrutura de um registro usando em conjunto uma matriz.

Tomando por base a proposta de criar um registro denominado ALUNO, cujas notas serão informadas em uma matriz do tipo vetor, ele seria assim declarado:

```
type
  BIMESTRE  = array[1..4] of real;
  CAD_ALUNO = record
                NOME : string;
                NOTA : bimestre;
              end;
var
  ALUNO : cad_aluno;
```

Ao ser especificado o registro CAD_ALUNO, existe nele um campo chamado NOTA do tipo bimestre, sendo que bimestre é a especificação de um tipo de conjunto matricial de uma única dimensão com capacidade para quatro elementos. Veja que o tipo bimestre foi definido acima do tipo CAD_ALUNO. Manter uma ordem na definição de tipos que são dependentes de outros tipos definidos é imprescindível, pois se o tipo BIMESTRE fosse definido abaixo do tipo CAD_ALUNO, geraria um erro na execução do programa, uma vez que o compilador do Turbo Pascal consideraria como sendo tipo desconhecido. A seguir, é apresentado o código-fonte para a leitura e escrita dos dados usando o registro de matriz.

```
program LEITURA_ESCRITA;
type
  BIMESTRE  = array[1..4] of real;
  CAD_ALUNO = record
                NOME : string;
                NOTA : bimestre;
              end;
var
  ALUNO : cad_aluno;
  I : byte;
begin
  writeln('Cadastro de aluno');
  writeln;
  write('Informe o nome .......: '); readln(ALUNO.NOME);
  writeln;
  for I := 1 to 4 do
    begin
      write('Informe a ', I:2, 'a. nota ..: ');
      readln(ALUNO.NOTA[I]);
    end;
  writeln;
  writeln;
  writeln('Nome ....: ', ALUNO.NOME);
  writeln;
  for I := 1 to 4 do
    writeln('Nota ', I, ' ..: ', ALUNO.NOTA[I]:5:2);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Feche as janelas que estejam abertas, em seguida abra uma nova janela e digite o programa anterior, gravando-o com o nome ALUNO2.

Com as técnicas de programação expostas até este momento, passou-se a ter uma mobilidade bastante grande, com a possibilidade de 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, seria pertinente construir um programa que permitisse trabalhar com vários registros de alunos. Para exemplificar, considere que você deve elaborar um programa que faça a entrada e a saída de nome e notas de oito alunos. Isto já é familiar. Veja, a seguir, a definição do tipo registro e também a definição da matriz de registros para os oito alunos:

```
program LEITURA_ESCRITA;
type
  BIMESTRE  = array[1..4] of real;
  CAD_ALUNO = record
                NOME : string;
                NOTA : bimestre;
              end;
var
  ALUNO : array[1..8] of cad_aluno;
```

Observe que após a instrução **var**, é indicada a variável de registro ALUNO, sendo um conjunto de oito 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. A seguir, é apresentado um programa que fará a entrada e saída dos dados de oito alunos.

```
program LEITURA_ESCRITA;
type
  BIMESTRE  = array[1..4] of real;
  CAD_ALUNO = record
                NOME : string;
                NOTA : bimestre;
              end;
var
  ALUNO : array[1..8] of cad_aluno;
  I, J  : byte;
begin
  writeln('Cadastro de aluno');
  writeln;
  for J := 1 to 8 do
    begin
      write('Informe nome do ', J:2,'o. aluno ...: ');
      readln(ALUNO[J].NOME);
      writeln;
      for I := 1 to 4 do
        begin
          write('Informe a ', I:2, 'a. nota ..........: ');
```

```
          readln(ALUNO[J].NOTA[I]);
        end;
      writeln;
    end;
  writeln;
  writeln;
  for J := 1 to 8 do
    begin
      writeln('Nome aluno: ', J:2,' ...: ', ALUNO[J].NOME);
      writeln;
      for I := 1 to 4 do
       writeln('Nota ', I, ' ...........: ', ALUNO[J].NOTA[I]:5:2);
      writeln;
    end;
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Perceba que o looping da variável *J* controla o número de alunos da turma, no caso oito, e o looping da variável *I* controla o número de notas, até quatro por aluno. Para cada movimentação de mais um na variável *J*, existem quatro movimentações na variável *I*. Em seguida feche as janelas que estejam abertas, abra uma nova janela e digite o programa anterior, gravando-o com o nome CLASSE.

8.2 - Aplicação Prática do Tipo Registro

Para demonstrar a utilização de programas com tabelas de dados heterogêneos, considere um programa que efetue a leitura das quatro notas bimestrais de oito alunos, apresentando no final os dados dos alunos classificados por nome.

Por se tratar de uma ordenação, é necessário estabelecer mais duas variáveis, as quais podem ser *ATUAL* e *PRÓXIMO*. Deve ser também estabelecida uma variável de auxílio à troca, a qual pode ser a variável *X*, porém deve ser do tipo registro. A ordenação será efetuada com base no nome de cada aluno e quando estiver fora da ordem, os dados devem ser trocados de posição.

```
program LEITURA_ORDENACAO_ESCRITA;
type
  BIMESTRE  = array[1..4] of real;
  CAD_ALUNO = record
                NOME : string;
                NOTA : bimestre;
              end;
var
  ALUNO : array[1..8] of cad_aluno;
  I, J, ATUAL, PROXIMO : byte;
  X : cad_aluno;
begin
```

```
  {*** Rotina de entrada ***}
  writeln('Cadastro de aluno');
  writeln;
  for J := 1 to 8 do
    begin
      write('Informe nome do ', J:2,'o. aluno ...: ');
      readln(ALUNO[J].NOME);
      writeln;
      for I := 1 to 4 do
        begin
          write('Informe a ', I:2, 'a. nota ..........: ');
          readln(ALUNO[J].NOTA[I]);
        end;
      writeln;
    end;
  writeln;
  {*** Rotina de ordenacao ***}
  for ATUAL := 1 to 7 do
    for PROXIMO := ATUAL + 1 to 8 do
      if (ALUNO[ATUAL].NOME > ALUNO[PROXIMO].NOME) then
        begin
          X := ALUNO[ATUAL];
          ALUNO[ATUAL] := ALUNO[PROXIMO];
          ALUNO[PROXIMO] := X;
        end;
  {*** Rotina de saída ***}
  writeln;
  for J := 1 to 8 do
begin
writeln('Nome aluno: ', J:2,' ...: ', ALUNO[J].NOME);
writeln;
for I := 1 to 4 do
writeln('Nota ', I, ' ...........: ', ALUNO[J].NOTA[I]:5:2);
writeln;
writeln('Tecle <ENTER> para ver o proximo: '); readln;
  end;
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

O programa anterior apresenta a comparação efetuada entre os nomes dos alunos. Sendo o nome do aluno atual maior que o nome do próximo aluno, 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 da tabela ALUNO, no caso CAD_ALUNO. Em seguida feche as janelas que estejam abertas, abra uma nova janela e digite o programa anterior, gravando-o com o nome SORTCLAS.

8.3 - EXERCÍCIO

Tome como base de trabalho o desenvolvimento de um programa de cadastro para uma pequena agenda de endereços, nomes e telefones, defina a estrutura de registro apropriada e construa um programa que, por meio de um menu de seleção, esteja capacitado a efetuar:

- ✓ o cadastramento das informações e sua classificação;
- ✓ a pesquisa dos nomes pelo método binário;
- ✓ alteração de registro cadastrado com erro (pesquisar por nome - método seqüencial);
- ✓ remoção de um determinado registro (pesquisar por nome - método seqüencial).

Cada uma destas operações deve ser controlada dentro de um (**if...then**). Não se esqueça de deixar uma opção reservada para o usuário sair do sistema.

Parte 4

Programação Estruturada ou Modular

```
procedure CALC_ADITAO(A
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Utilização de Sub-rotinas

A partir deste capítulo será estudada a aplicação de sub-rotinas. Você terá contato com o conceito da criação estruturada de programas (modularização) utilizando a linguagem Pascal. Esta técnica de programação é das mais utilizadas, considerada como vantajosa no desenvolvimento de grandes programas.

9.1 - As Sub-rotinas

Em 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 esse trecho menor é chamado de sub-rotina. Quando uma sub-rotina é chamada por um programa principal, 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 sua chamada.

Ao trabalhar com esta técnica, pode haver 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.*

9.2 - Tipos de Sub-rotinas

A estrutura da linguagem Pascal permite a utilização de dois tipos de rotinas definidos pelo programador, sendo **Procedure** (procedimento) e **Function** (função). Uma **Procedure** ou **Function** é, na verdade, um bloco de programa, contendo início e fim, identificado por um nome por meio do qual será referenciado em qualquer parte do programa principal, ou do programa chamador da rotina. A diferença entre os dois tipos de sub-rotina está no fato de uma **Procedure** poder ou não retornar um valor após seu processamento, enquanto uma **Function** sempre vai retornar um valor após seu processamento. São exatamente estes dois tipos de sub-rotinas que serão estudados nos próximos capítulos.

Além das rotinas definidas pelo programador, no ambiente de programação Turbo Pascal existe um conjunto de rotinas embutidas, fornecidas pela Borland (detentora do ambiente de programação Turbo Pascal e não da linguagem Pascal propriamente dita). Este tipo de rotina embutida é conhecido pelo nome de **unidade**, em inglês **unit**.

9.3 - Utilização de Rotinas Embutidas: As Units

O objetivo deste tópico é apresentar de forma básica o conceito de utilização das **units** embutidas no Turbo Pascal. As **Units** são um conjunto de rotinas prontas para serem usadas pelo programador. O conceito de unidades foi incorporado ao Turbo Pascal a partir da versão 4, objetivando agrupar diversas rotinas básicas.

Uma **Unit** é uma biblioteca de funções, procedimentos e constantes. Abaixo, são relacionadas as unidades do Turbo Pascal e sua rápida descrição:

Unidade	Descrição
CRT	Esta unidade é a mais utilizada no ambiente Turbo Pascal. Por esta razão, ela possui a maior parte das rotinas e variáveis de geração de som, controle de vídeo e teclado.
DOS	Esta unidade possui as rotinas que envolvem a utilização de recursos do sistema operacional, na maior parte das vezes permitindo controles de baixo nível.
GRAPH	Esta unidade possui rotinas destinadas à manipulação da capacidade gráfica de um PC.
OVERLAY	Esta unidade possibilita gerenciar as atividades de um programa, desta forma é possível aproveitar uma mesma área de memória para rodar várias rotinas diferentes, economizando memória.
PRINTER	Esta unidade permite que a impressora seja usada como saída de informação. Internamente esta unidade declara o conteúdo a ser escrito como um arquivo tipo texto de nome LST e o associa ao dispositivo de impressora.
SYSTEM	Esta unidade possui a maior parte das rotinas padrão da linguagem Pascal, não necessitando ser citada para ser usada, pois o ambiente Turbo Pascal já a executa de forma automática.

Para fazer uso deste recurso, é necessário utilizar a instrução **uses** situada antes da declaração da instrução **var**. Desta forma, sua sintaxe corresponde a:

```
uses <unidade>
```

Sendo que *unidade* é uma das unidades citadas acima, com exceção da unidade SYSTEM.

Para exemplificar o uso de uma unidade, considere como exemplo a necessidade de limpar a tela do computador após a execução de um programa, ou mesmo posicionar as mensagens em um determinado ponto da tela. Você deve ter percebido que todos os programas criados até este momento apresentam as mensagens uma embaixo da outra e não fazem a limpeza da tela.

Para usar o recurso de limpar tela e posicionar mensagens, será usada a unidade do ambiente Turbo Pascal, denominada: **Crt**. Para tanto, feche todas as janelas abertas e carregue para a memória o programa SORTNOME, criado durante o estudo do capítulo 6, como apresentado em seguida:

```
program LISTA_NOME_ORDENADA;
var
  NOME : array[1..10] of string;
  I, J : integer;
  X    : string;
  {*** Rotina de entrada de dados ***}
begin
  writeln('Listagem de nomes');
  writeln;
  for I := 1 to 10 do
    begin
      write('Digite o ', I:2 ,'o. nome: '); readln(NOME[I]);
    end;
  {*** Rotina de processamento de ordenacao ***}
  for I := 1 to 9 do
    for J := I + 1 to 10 do
      if (NOME[I] > NOME[J]) then
         begin
           X := NOME[I];
           NOME[I] := NOME[J];
           NOME[J] := X;
         end;
  {*** Rotina de saida com dados ordenados ***}
  writeln;
  for I := 1 to 10 do
    writeln('Nome: ' ,I:2, ' --> ', NOME[I]);
  writeln;
  writeln('Tecle <ENTER> para encerrar: '); readln;
end.
```

Para utilizar uma unidade do ambiente Turbo Pascal, será definida uma regra. Após a citação da instrução **uses**, a unidade em uso será escrita neste livro com o primeiro caractere em maiúsculo e os demais em minúsculo, ou seja, **uses** Crt seguida de um ponto-e-vírgula. No caso de ser utilizada mais de uma unidade, elas devem ser separadas uma das outras com uma vírgula, da mesma maneira que as variáveis são definidas numa mesma linha.

A unidade **Crt** possibilita, neste momento, fazer uso de três recursos diferentes de controle para o programa SORTNOME, sendo: limpeza de tela (**clrscr**), posicionamento de mensagem (**gotoxy**) e interrupção do processamento de um programa, aguardando que seja pressionada qualquer tecla para o programa continuar (**readkey**). Assim sendo, observe as novas mudanças no programa seguinte:

```
program LISTA_NOME_ORDENADA;
uses
  Crt;
var
  NOME  : array[1..10] of string;
  I, J  : integer;
  X     : string;
  TECLA : char;
  {*** Rotina de entrada de dados ***}
begin
  clrscr;
  gotoxy(31, 2); writeln('Listagem de nomes');
  writeln;
  for I := 1 to 10 do
    begin
      write('Digite o ', I:2 ,'o. nome: '); readln(NOME[I]);
    end;
  gotoxy(23,24); writeln('Tecle algo para ver lista ordenada');
  TECLA := readkey;
  {*** Rotina de processamento de ordenacao ***}
  for I := 1 to 9 do
    for J := I + 1 to 10 do
      if (NOME[I] > NOME[J]) then
         begin
           X := NOME[I];
           NOME[I] := NOME[J];
           NOME[J] := X;
         end;
  {*** Rotina de saída com dados ordenados ***}
  clrscr;
  gotoxy(31, 2); writeln('Listagem ordenada');
  writeln;
  for I := 1 to 10 do
    writeln('Nome: ' ,I:2, ' --> ', NOME[I]);
  gotoxy(22,24); writeln('Tecle algo para encerrar o programa');
  TECLA := readkey;
end.
```

Após a digitação do programa, efetue a sua gravação com o nome SORTNOM2. Para tanto, execute o comando **File/Save as**.

Depois execute o programa e observe que primeiro a tela é limpa. Em seguida, a mensagem **'Listagem de nomes'** é apresentada e posicionada no meio da tela, e é solicitada a digitação dos nomes. Ao fim de dez nomes, é apresentada no meio da tela na parte inferior, a mensagem **'Tecle algo para ver lista ordenada'**. Ao ser pressionada qualquer tecla, é efetuada uma nova limpeza de tela, a mensagem **'Listagem ordenada'** é apresentada com todos os nomes listados e na parte inferior da tela ao centro, a mensagem **'Tecle algo para encerrar o programa'**. Observe também a inclusão de uma nova variável TECLA do tipo **char**. Este tipo é similar ao tipo **string**, porém aceita apenas um caractere.

Na alteração do programa, foram usadas três rotinas da unidade **Crt**, sendo que as três somente serão executadas se antes da instrução **var** estiver citada a instrução **uses** com a referida unidade. Caso não seja indicada a unidade a ser usada, qualquer rotina a ela pertencente gerará um erro na execução do programa.

No exemplo apresentado, foram utilizados dois procedimentos (clrscr e gotoxy) e uma função (readkey) descritos a seguir:

Rotina	Tipo	Descrição
clrscr	Procedimento	Este procedimento efetua a limpeza da tela, retirando qualquer caractere anteriormente escrito, posicionando o cursor no canto esquerdo superior. O termo clrscr é a contração de duas palavras em inglês CLeaR SCReen, ou seja, limpar tela.
gotoxy(coluna,linha)	Procedimento	Este procedimento permite posicionar o cursor em um determinado ponto da tela. Para utilizá-lo é necessário informar dois parâmetros: o primeiro representa o valor da coluna, que deve ser um valor numérico inteiro positivo entre 1 e 80, o segundo parâmetro representa o número da linha, que deve ser um valor numérico inteiro positivo entre 1 e 25.
readkey	Função	Esta é uma função que permite retornar o valor da tecla acionada, sendo esse valor do tipo caractere. Quando utilizada em um programa, esta função faz a leitura de apenas um caractere. Por esta razão não necessita ser usada a tecla <ENTER> para confirmar a entrada da informação.

Um pouco mais adiante, será visto o uso de alguns outros recursos baseados em unidades. Será demonstrado o uso de saída em impressora (unidade **printer**) e obtenção de data e hora do sistema (unidade **dos**), além de apresentar as instruções para a criação de bibliotecas particulares e pessoais.

Capítulo 10

Utilização de Procedures

Neste capítulo, você terá contato com a sub-rotina do tipo **Procedure**, que será tratada com os mesmos cuidados de programação já estudados e aplicados. A sintaxe em Pascal é definida como:

```
procedure <nome> [(parâmetros)];
var
  <variáveis>;
begin
  <instruções>;
end;
```

em que:

`<nome>`	-	o nome atribuído ao procedimento;
`<parâmetro>`	-	uma informação opcional (será estudada mais à frente).

Os demais detalhes (var, begin e end) de uma sub-rotina do tipo **Procedure** já são conhecidos e estudados. A melhor maneira de entender como trabalhar com este conceito é fazer a sua aplicação em um problema mais complexo.

10.1 - Aplicação Prática de Procedure

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

Algoritmo

Note que este programa deve 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, efetuarão o pedido de leitura de dois valores, farão a operação e apresentarão o resultado obtido.

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.

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 deve ser executada;
3. Ao escolher o valor 5, o programa deve 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 X;
3. Apresentar o valor da variável X;
4. Voltar ao programa principal.

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 X;
3. Apresentar o valor da variável X;
4. Voltar ao programa principal.

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 X;
3. Apresentar o valor da variável X;
4. Voltar ao programa principal.

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 X;
3. Apresentar o valor da variável X;
4. Voltar ao programa principal.

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

Programa em Pascal

A linguagem Pascal exige que as sub-rotinas sejam definidas à frente do programa principal ou rotina chamadora. No tocante à definição do nome de um procedimento, será adotado o mesmo critério para a definição do nome de um programa. Quanto à chamada da rotina, será sempre escrita com o primeiro caractere em maiúsculo e os demais em minúsculo. Caso o nome seja uma contração de duas palavras, podem ser utilizados caracteres maiúsculos no meio do nome. Por exemplo: ROT_ADICAO pode ser indicado como Rot_Adicao. Digite o programa seguinte, gravando-o com o nome CALCULA1.

```
program CALCULADORA;
uses
  Crt;
var
  OPCAO : char;
  {*** Sub-rotinas de calculos ***}
  procedure ROT_ADICAO;
  var
    X, A, B : real;
    TECLA   : char;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Adicao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A + B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:6:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
  procedure ROT_SUBTRACAO;
  var
    X, A, B : real;
    TECLA   : char;
  begin
   clrscr;
```

```
    gotoxy(30, 1); write('Rotina de Subtracao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A - B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:6:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
  procedure ROT_MULTIPLICACAO;
  var
    X, A, B : real;
    TECLA   : char;
  begin
    clrscr;
    gotoxy(28, 1); write('Rotina de Multiplicacao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A * B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:6:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
  procedure ROT_DIVISAO;
  var
    X, A, B : real;
    TECLA   : char;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Divisao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A / B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:6:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
{*** Programa Principal ***}
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      gotoxy(33, 1); write('Menu Principal');
      gotoxy(28, 6); write('1 ................ Soma');
      gotoxy(28, 8); write('2 ........... Subtracao');
      gotoxy(28,10); write('3 ........ Multiplicacao');
      gotoxy(28,12); write('4 ............ Divisao');
      gotoxy(28,14); write('5 ...... Fim de Programa');
      gotoxy(28,18); write('Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO = '1') then
```

```
        Rot_Adicao;
      if (OPCAO = '2') then
        Rot_Subtracao;
      if (OPCAO = '3') then
        Rot_Multiplicacao;
      if (OPCAO = '4') then
        Rot_Divisao;
    end;
end.
```

10.2 - Estrutura de Controle com Múltipla Escolha

O programa anterior apresenta o selecionamento das sub-rotinas feito com instruções **if...then**. Perceba que se o programa possuir um menu com quinze opções, devem ser definidas quinze instruções do tipo **if...then** para verificar a escolha do operador. Apesar de a forma apresentada ser válida, ela pode ser simplificada com a utilização da instrução **case...of**, que possui a sintaxe:

```
case <variável> of
  <opção1> : <operação1>;
  <opção2> : <operação2>;
  <opçãoN> : <operaçãoN>;
else
  <operação>;
end;
```

ou

```
case <variável> of
  <opção1> : begin
               <operação1.1>;
               <operação1.2>;
               <operação1.N>;
             end;
  <opção2> : begin
               <operação2.1>;
               <operação2.2>;
               <operação2.N>;
             end;
  <opçãoN> : begin
               <operaçãoN.1>
            <operaçãoN.2>
               <operaçãoN.N>
             end;
else
  <operação_else>
end;
```

em que:

<variável>	-	o nome da variável a ser controlada na decisão;
<opção>	-	o conteúdo da variável a ser verificado;
<operação>	-	a execução de uma instrução Pascal ou de uma rotina.

Observação: *A instrução **else** de **case...of** e qualquer operação a ela subordinada são opcionais, não caracterizando uma obrigatoriedade do seu uso.*

A rotina principal do programa calculadora pode ser escrita fazendo uso da instrução **case...of** no selecionamento de uma opção escolhida pelo operador. Perceba como o programa principal ficou menor em número de linhas. A seguir, é apresentado em Pascal apenas o programa principal alterado.

```
{*** Programa Principal ***}
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      gotoxy(33, 1); write('Menu Principal');
      gotoxy(28, 6); write('1 ................ Soma');
      gotoxy(28, 8); write('2 ........... Subtracao');
      gotoxy(28,10); write('3 ........ Multiplicacao');
      gotoxy(28,12); write('4 ............. Divisao');
      gotoxy(28,14); write('5 ...... Fim de Programa');
      gotoxy(28,18); write('Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO <> '5') then
        case OPCAO of
           '1' : Rot_Adicao;
           '2' : Rot_Subtracao;
           '3' : Rot_Multiplicacao;
           '4' : Rot_Divisao;
        else
           gotoxy(27,24); writeln('Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

Observe que antes da utilização da instrução **case...of**, está sendo verificado com a instrução **if...then** 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 **else** de **case...of** executa a apresentação da mensagem **'Opcao invalida - Tecle algo'**, informando que a tentativa foi inválida. Faça a alteração proposta e grave o programa com o nome CALCULA2. Não se esqueça de utilizar o comando de menu **File/Save as**.

10.3 - VARIÁVEIS GLOBAIS E LOCAIS

Na confecção do programa anterior, foram utilizadas algumas variáveis dentro das sub--rotinas, no caso as variáveis *A*, *B* e *X*, e também fora das sub-rotinas, no caso a variável *OPCAO*, sem que houvesse a preocupação de agrupar as variáveis de uma forma coerente segundo o seu tipo. São dois os tipos de variáveis utilizadas em um programa: as variáveis *Globais* e as variáveis *Locais*.

Uma variável é considerada *Global* quando é declarada no início de um programa escrito em Pascal, podendo ser utilizada por qualquer sub-rotina subordinada. Assim sendo, este tipo de variável passa a ser visível a todas as sub-rotinas. No programa anterior, *OPCAO* é uma variável global, apesar de não estar sendo utilizada nas sub-rotinas de cálculo.

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 podem fazer uso daquelas variáveis como *Global*, pois não visualizam a existência delas. No programa anterior, as variáveis *A*, *B* e *X* 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 os dois exemplos seguintes, sendo o primeiro sem uso de sub-rotina e o segundo com sub-rotina, utilizando o conceito de variável global e local:

1º Exemplo

```
program TROCA1;
var
  X : integer;
  A : integer;
  B : integer;
begin
  readln(A);
  readln(B);
  X := A;
  A := B;
  B := X;
  writeln(A);
  writeln(B);
end.
```

2º Exemplo

```
program TROCA2;
var
  A : integer;
  B : integer;
```

```
procedure TROCA;
var
  X : integer;
begin
  X := A;
  A := B;
  B := X;
end;
begin
  readln(A);
  readln(B);
  Troca;
  writeln(A);
  writeln(B);
end.
```

Perceba que, apesar de o segundo exemplo ser um pouco maior que o primeiro em número de linhas, 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*.

Digite os programas em separado, gravando o primeiro exemplo com o nome TROCA1 e o segundo exemplo com o nome TROCA2.

10.3.1 - Escopo de Variáveis

O escopo de uma variável ou sua abrangência está vinculada 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 pode ser considerada global para todas as sub-rotinas inferiores a uma rotina principal, e dentro de uma dessas sub-rotinas a mesma variável pode estar sendo utilizada como local. Observe o gráfico apresentado na figura 10.1.

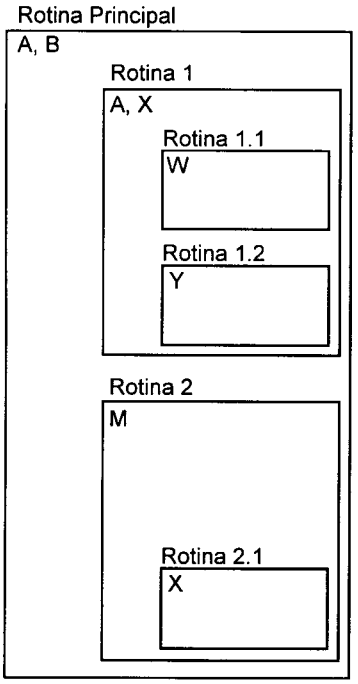


Figura 10.1 - Representação do escopo de variáveis.

Com relação à figura 10.1, 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 essa sub-rotina (é como se estivesse utilizando uma nova variável, no caso *A'*). A variável A considerada como *A'* na rotina 1 torna-se global para as sub-rotinas 1.1 e 1.2, que também terão como global a variável *X*. As variáveis *W* e *Y* que, respectivamente, pertencem às sub-rotinas 1.1 e 1.2 são variáveis 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 essa 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 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 de troca, seriam locais, conforme é demonstrado em seguida:

```
program TROCA3;
{*** Rotina com erro de declaracao de variaveis ***}
procedure TROCA;
var
  X : integer;
begin
  X := A;
  A := B;
  B := X;
end;
var
  A : integer;
  B : integer;
begin
  readln(A);
  readln(B);
  Troca;
  writeln(A);
  writeln(B);
end.
```

Digite o programa e grave-o com o nome TROCA3. Ao ser executado, será apresentada a mensagem de erro **Error 3: Unknown Identifier.**, com o cursor posicionado sobre a primeira ocorrência da variável **A**. Isto ocorre pelo fato de as variáveis *A* e *B* para a sub-rotina TROCA não serem válidas, uma vez que não estão definidas para essa sub-rotina. Assim sendo, essas variáveis devem ser declaradas dentro da sub-rotina para eliminar o erro encontrado.

10.4 - 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 nas quatro sub-rotinas de cálculo, existem 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 *X* são definidas quatro vezes, uma em cada sub-rotina.

A solução é definir as variáveis *A*, *B* e *X* como globais e construir mais duas sub-rotinas, sendo 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. Desta forma, as variáveis *A*, *B* e *X* serão definidas como globais. Serão implementadas duas novas rotinas, sendo uma para efetuar a entrada dos valores e outra para efetuar a saída do resultado da operação. A seguir, é apresentada a nova versão do programa calculadora.

```
program CALCULADORA;
uses
  Crt;
var
  OPCAO : char;
  X, A, B : real;
  {*** Rotinas de entrada e saida ***}
  procedure ENTRADA;
  begin
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
  end;
  procedure SAIDA;
  var
    TECLA    : char;
  begin
    gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
  {*** Rotinas de calculo ***}
  procedure ROT_ADICAO;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Adicao');
    Entrada;
    X := A + B;
    Saida;
  end;
  procedure ROT_SUBTRACAO;
```

```
  begin
    clrscr;
    gotoxy(30, 1); write('Rotina de Subtracao');
    Entrada;
    X := A - B;
Saida;
  end;
  procedure ROT_MULTIPLICACAO;
  begin
    clrscr;
    gotoxy(28, 1); write('Rotina de Multiplicacao');
Entrada;
    X := A * B;
    Saida;
  end;
  procedure ROT_DIVISAO;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Divisao');
    Entrada;
    X := A / B;
    Saida;
  end;
{*** Programa Principal ***}
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      gotoxy(33, 1); write('Menu Principal');
      gotoxy(28, 6); write('1 ................ Soma');
      gotoxy(28, 8); write('2 ........... Subtracao');
      gotoxy(28,10); write('3 ........ Multiplicacao');
      gotoxy(28,12); write('4 ............. Divisao');
      gotoxy(28,14); write('5 ...... Fim de Programa');
      gotoxy(28,18); write('Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO <> '5') then
        case OPCAO of
           '1' : Rot_Adicao;
           '2' : Rot_Subtracao;
           '3' : Rot_Multiplicacao;
           '4' : Rot_Divisao;
        else
           gotoxy(27,24); writeln('Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

Perceba que nas rotinas de cálculos, não está sendo declarada nenhuma das variáveis utilizadas no cálculo, uma vez que elas estão definidas como variáveis globais. A variável *TECLA* está definida como local na rotina de saída. Digite esta nova versão do programa, gravando-o com o nome CALCULA3.

Ao executar o programa, você notará que em relação às versões anteriores, ele executa as mesmas funções, não existindo nenhuma diferença do ponto de vista operacional. A diferença está na construção lógica do programa.

10.5 - EXERCÍCIO

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

a. Com base nas técnicas de programação apresentadas até este ponto, desenvolva um programa de cadastro de uma pequena agenda de endereços, nomes e telefones, crie um programa utilizando sub-rotinas e menu que efetue:

- ✓ cadastramento de elementos;
- ✓ pesquisa com método binário (necessita de classificação);
- ✓ alteração de elementos que tenham sido cadastrados com erro (necessita pesquisar);
- ✓ remoção de elementos cadastrados (necessita pesquisar).

b. Desenvolver um programa que controle as notas bimestrais de dezoito alunos de uma sala de aula. O programa deve efetuar o cálculo da média de cada aluno, sendo que será considerado aprovado o aluno que obtiver média igual ou superior a 7. O programa em questão deve executar as seguintes rotinas:

- ✓ cadastrar os dezoito alunos e suas notas, calcular média e colocar em ordem alfabética por nome;
- ✓ alterar os dados dos alunos cadastrados com erro (com pesquisa seqüencial);
- ✓ pesquisar alunos por nome (com pesquisa binária);
- ✓ listar alunos aprovados;
- ✓ listar alunos reprovados.

Utilização de Parâmetros

Os parâmetros têm por finalidade servir como um ponto de comunicação bidirecional entre uma sub-rotina e o programa principal, ou com uma outra 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, com a utilização de parâmetros que poderão ser *formais* ou *reais*.

11.1 - 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 apresentado em seguida:

```
procedure CALC_ADICAO(A, B : integer);
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Observe que *Z* é uma variável 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 estes 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 do programa principal que faz uso da sub-rotina CALCSOMA, apresentado em seguida:

```
begin
  readln(X);
  readln(Y);
  Calc_Adicao(X, Y);
  readln(W);
  readln(T);
  Calc_Adicao(W, T);
  Calc_Adicao(6, 4);
end.
```

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 são fornecidos seus valores pela instrução **readln**. São também considerados parâmetros reais os valores 6 e 4. A seguir, é apresentado o código completo do programa que deve ser digitado e gravado com o nome CALCADIC.

```
program CALCADIC;
procedure CALC_ADICAO(A, B : integer);
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
var
  X, Y, W, T : integer;
begin
  readln(X);
  readln(Y);
  Calc_Adicao(X, Y);
  readln(W);
  readln(T);
  Calc_Adicao(W, T);
  Calc_Adicao(6, 4);
end.
```

11.2 - 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 serão passados por variáveis de duas formas: por valor e por referência.

11.2.1 - 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 afeta 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. A seguir, é apresentado um exemplo utilizando passagem de parâmetro por valor:

```
program FATORIAL1;
uses
  Crt;
procedure FATORIAL(N : integer);
var
  I, FAT : integer;
begin
  FAT := 1;
  for I := 1 to N do
    FAT := FAT * I;
  writeln('A fatorial de ', N,' equivale a: ', FAT);
end;
var
  LIMITE : integer;
begin
  clrscr;
  writeln('Programa Fatorial');
  writeln;
  write('Informe um valor inteiro: '); readln(LIMITE);
  Fatorial(LIMITE);
  writeln('Tecle <ENTER> para encerrar o programa'); readln;
end.
```

Digite o programa e grave-o com o nome FATOR_1. Observe que é indicado no programa 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 vai efetuar um looping, tendo ao final o valor da fatorial informado para o parâmetro *N*. Ao término do looping, o valor calculado é apresentado por meio da variável *FAT*, que somente é válida dentro da sub-rotina, e por esta razão o valor dessa variável fica "preso" na rotina. A passagem de parâmetro por valor é utilizada somente para a entrada de um determinado valor em uma sub-rotina.

11.2.2 - Por Referência

A passagem de parâmetro por referência caracteriza-se pela ocorrência da 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 em seguida, e observe a instrução **var** sendo utilizada junto da declaração do parâmetro em português estruturado:

```
program FATORIAL2;
uses
  Crt;
procedure FATORIAL(N : integer; var FAT : integer);
var
 I : integer;
begin
  FAT := 1;
  for I := 1 to N do
    FAT := FAT * I;
end;
var
  LIMITE, RETORNO : integer;
begin
  clrscr;
  writeln('Programa Fatorial');
  writeln;
  write('Informe um valor inteiro: '); readln(LIMITE);
  Fatorial(LIMITE, RETORNO);
  writeln('A fatorial de ', LIMITE,' equivale a: ', RETORNO);
  writeln('Tecle <ENTER> para encerrar o programa'); readln;
end.
```

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 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 possuirá no final o valor acumulado do cálculo da fatorial. Ao término do looping, o valor da variável *FAT* é transferido para fora da rotina, ou seja, é transferido para a variável *RETORNO* do programa principal, a qual apresenta o valor recebido de dentro da sub-rotina por meio da variável *FAT*. A passagem de parâmetro por referência é utilizada para que se tenha a saída de um determinado valor de dentro de uma sub-rotina. Digite o programa anterior e grave-o com o nome FATOR_2.

11.3 - Passagem de Parâmetro em uma Sub-rotina

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

O processo de ordenação já é conhecido. A proposta é que se crie uma sub-rotina que faça a ordenação dos valores. Você deve lembrar que uma ordenação ocorre com a troca de valores entre duas variáveis, ou a troca de elementos entre dois índices de uma matriz. E que 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. Desta forma, este problema pode ser resolvido de duas formas básicas, sendo a criação de duas sub-rotinas: 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 devem acompanhar os seguintes passos:

1. Criar duas sub-rotinas, 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.

1ª Solução

O trecho de programa seguinte especifica apenas as sub-rotinas de troca e de verificação da condição para a troca de valores. Como os parâmetros a serem utilizados são por referência, eles estão sendo declarados com a instrução **var**.

```
procedure TROCA(var A, B : integer);
var
 X : integer;
begin
  X := A;
  A := B;
  B := X;
end;
procedure ORDENA(var A, B : integer);
begin
  if (A > B) then
    Troca(A, B);
end;
```

2ª Solução

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

```
procedure ORDENA(var A, B : integer);
var
 X : integer;
begin
  if (A > B) then
    begin
```

```
      X := A;
      A := B;
      B := X;
    end;
end;
```

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

```
program ORDENA_NUMEROS;
uses
  Crt;
var
  VETOR : array[1..10] of integer;
  I, J  : integer;
  TECLA : char;
{*** Rotina de ordenacao ***}
procedure ORDENA(var A, B : integer);
var
 X : integer;
begin
  if (A > B) then
    begin
      X := A;
      A := B;
      B := X;
    end;
end;
begin
  {*** Entrada dos dados ***}
  clrscr;
  gotoxy(23, 1); write('Programa para ordenacao de valores');
  for I := 1 to 10 do
    begin
      gotoxy( 5,I+2);
      write('Entre um valor para VETOR[',I:2,']: ');
      readln(VETOR[I]);
    end;
  gotoxy(25,24); writeln('Tecle algo para continuar ...');
  TECLA := readkey;
  {*** Ordenacao dos dados ***}
  for I := 1 to 9 do
    for J := I + 1 to 10 do
      Ordena(VETOR[I], VETOR[J]);
  {*** Saida dos dados ***}
  for I := 1 to 10 do
    begin
      gotoxy(46,I+2);
      write('Agora VETOR[',I:2,'] vale: ', VETOR[I]:2);
```

```
    end;
  gotoxy(22,24); writeln('Tecle algo para encerrar o programa');
  TECLA := readkey;
end.
```

Digite e grave o programa com o nome SORTNR. Perceba que a criação de sub-rotina de ordenação deixou o programa "mais limpo". Uma sub-rotina, se bem projetada, pode ser de grande valia. O segundo parâmetro de **gotoxy** definido como *I*+2 indica que para cada alteração da variável *I* será adicionado +2 para manipular o número de linha.

Imagine ainda uma sub-rotina de ordenação em que você possa informar um parâmetro para que a ordenação seja crescente ou decrescente. 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 a seguir, como ficará este exemplo.

```
procedure ORDENA(var A, B : integer; TIPO : char);
var
  X : integer;
begin
  if (TIPO = '>') then
     begin
       if (A > B) then
         begin
           X := A;
           A := B;
           B := X;
         end;
     end
  else
     begin
       if (A < B) then
         begin
           X := A;
           A := B;
           B := X;
         end;
     end;
end;
```

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 linguagem Pascal ficará:

```
procedure TROCA(var A, B : integer);
var
 X : integer;
begin
  X := A;
  A := B;
  B := X;
end;
procedure ORDENA(var A, B : integer; TIPO : char);
var
  X : integer;
begin
  if (TIPO = '>') then
    begin
      if (A > B) then
        begin
          Troca(A, B);
        end;
    end
  else
    begin
      if (A < B) then
        begin
          Troca(A, B);
        end;
    end;
end;
```

A seguir, é apresentado o programa de ordenação de matrizes com a implementação da nova rotina. Digite o programa, gravando-o com o nome SORTNR2.

```
program ORDENA_NUMEROS;
uses
  Crt;
var
  VETOR : array[1..10] of integer;
  I, J  : integer;
  TECLA : char;
  OPCAO : char;
{*** Rotina de ordenacao ***}
procedure TROCA(var A, B : integer);
var
 X : integer;
begin
  X := A;
  A := B;
  B := X;
end;
procedure ORDENA(var A, B : integer; TIPO : char);
var
  X : integer;
```

```
begin
  if (TIPO = '>') then
    begin
      if (A > B) then
        begin
          Troca(A, B);
        end;
    end
  else
    begin
      if (A < B) then
        begin
          Troca(A, B);
        end;
    end;
end;
begin
  {*** Entrada dos dados ***}
  clrscr;
  gotoxy(23, 1); write('Programa para ordenacao de valores');
  gotoxy( 5, 3); write('Selecione um operador:');
  gotoxy( 5, 5); write('[>] ........ Crescente');
  gotoxy( 5, 6); write('[<] ...... Decrescente');
  gotoxy( 5, 8); write('==> '); readln(OPCAO);
  for I := 1 to 10 do
    begin
      gotoxy( 5,I+12);
      write('Entre um valor para VETOR[',I:2,']: ');
      readln(VETOR[I]);
    end;
  gotoxy(25,24); writeln('Tecle algo para continuar ...');
  TECLA := readkey;
  {*** Ordenacao dos dados ***}
  for I := 1 to 9 do
    for J := I + 1 to 10 do
      Ordena(VETOR[I], VETOR[J], OPCAO);
  {*** Saida dos dados ***}
  for I := 1 to 10 do
    begin
      gotoxy(46,I+12);
      write('Agora VETOR[',I:2,'] vale: ', VETOR[I]:2);
    end;
  gotoxy(22,24); writeln('Tecle algo para encerrar o programa');
  TECLA := readkey;
end.
```

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.

11.4 - EXERCÍCIO

Desenvolva os programas dos problemas indicados em seguida e suas sub-rotinas do tipo *procedure*. Cada problema deve ser analisado, pois você deve detectar se será usado passagem de parâmetro por valor ou por referência.

a. Elaborar um programa que possua uma sub-rotina que efetue e permita apresentar o somatório dos N primeiros números inteiros, definidos por um operador (1+2+3+4+5+6+7+...+N).

b. 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.

c. Ler duas matrizes A e B do tipo vetor com trinta elementos cada. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deve ter o dobro de elementos de A e B. Neste exercício, deve ser criada uma sub-rotina para cada tarefa do programa, ou seja, serão quatro sub-rotinas, sendo duas para leitura das matrizes, uma para a junção e a última para apresentação dos dados. No final, o programa principal deve chamar as rotinas definidas.

d. Criar um algoritmo que efetue o cálculo de uma prestação em atraso. Para tanto, utilize a fórmula PREST = VALOR + (VALOR * (TAXA/100) * TEMPO).

e. Desenvolva um algoritmo de programa que crie uma sub-rotina para calcular o valor de uma potência de um número qualquer, ou seja, ao informar para a sub--rotina o valor da base e de seu expoente, deve ser apresentado o resultado da sua potência. Por exemplo, se for mencionado no programa principal a sub-rotina POTENCIA(2,3), deve ser apresentado o valor 8. Para calcular, não utilize a forma baseada em exp(ln(base)*expoente), utilize apenas laços para efetuar o cálculo.

f. Ler uma matriz A com doze elementos. Após a leitura, colocar os seus elementos em ordem crescente. Depois ler uma matriz B também com doze elementos, colocar os elementos de B em ordem crescente. Construir uma matriz C, sendo que cada elemento de C é a soma do elemento correspondente de A com B. Colocar em ordem crescente a matriz C e apresentar os seus valores.

Utilização de Function

Uma **Function**, assim como uma *Procedure,* é um bloco de programa, ao qual são válidas todas as regras já estudadas tanto de programação como de definição de uma sub--rotina, seus parâmetros e também a utilização do conceito de variáveis globais e locais. Sua diferença em relação a uma *Procedure* está no fato de retornar sempre um valor. O valor de uma função é retornado no próprio nome da função. Quando se diz valor, devem ser levados em consideração os valores numéricos, lógicos ou literais (string ou caracteres). Uma **Function** possui a seguinte sintaxe:

```
function <nome> [(parâmetros)] : <tipo>;
var
  <variáveis>
begin
  <instruções>
end;
```

em que:

`<nome>`	-	o nome atribuído à função;
`<parâmetro>`	-	uma informação opcional;
`<tipo>`	-	o tipo de dado a ser retornado pela função.

Os demais detalhes (var, begin e end) de uma sub-rotina do tipo **Function** já são conhecidos e estudados.

12.1 - Aplicação Prática de Function

As sub-rotinas do tipo *Procedure* por meio da passagem de parâmetro por referência permitem que sejam retornados valores à rotina chamadora. Mas esta tarefa pode ser simplificada com a utilização de sub-rotinas do tipo **Function**.

Para exemplificar, considere a *Procedure* seguinte, usada para calcular a fatorial de um número qualquer. Observe que é necessário fornecer dois valores para os parâmetros *N* e

FAT, sendo *N* a entrada do valor a ser calculado (passagem de parâmetro por valor) e *FAT* o valor resultado obtido como retorno (passagem de parâmetro por referência).

```
procedure FATORIAL(N : integer; var FAT : integer);
var
 I : integer;
begin
  FAT := 1;
  for I := 1 to N do
    FAT := FAT * I;
end;
```

Se a sub-rotina para o cálculo da fatorial de um número qualquer for escrita como uma função, é necessária apenas a informação de um parâmetro, no caso o valor do parâmetro *N*, do qual deve ser calculada a fatorial, pois o resultado do cálculo será atribuído ao próprio nome da função. Veja, a seguir, como isto é feito:

```
function FATORIAL(N : integer) : integer;
var
 I, FAT : inteiro
begin
  FAT := 1;
  for I := 1 to N do
    FAT := FAT * I;
  FATORIAL := FAT;
end;
```

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 fatorial de zero. Desta forma, uma função retorna um valor pelo seu próprio nome, pois esse nome é usado no corpo da função para a recepção do valor calculado.

Outro detalhe a ser considerado numa função, além de seu nome ser usado como variável de retorno, é o fato da definição do seu tipo, no caso, **function** FATORIAL(*N* : **integer**) : **integer;**, 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 pode ser entrado um parâmetro de um tipo e retorná-lo como um tipo diferente do entrado.

12.2 - Considerações a Respeito de Functions

Você pode estar achando que em nível de número de linhas a 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 utilizada.

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

```
Fatorial(5, RETORNO);
writeln(RETORNO);
```

Caso se deseje obter o resultado da fatorial 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 é necessário duas linhas de programa.

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

```
writeln(Fatorial(5));
```

Desta 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 pela 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 e não em uma variável, como foi o caso da sub-rotina de procedimento equivalente.

Vale salientar que uma função também pode trabalhar com passagem de parâmetros por valor ou por 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. Mas, pode ocorrer em alguns casos, além de obter o retorno da função, obter o valor alterado do parâmetro. Neste caso, a passagem do parâmetro deve ser feita por referência.

Para demonstrar a utilização de programas com sub-rotinas do tipo **Function**, considere os seguintes exemplos:

1º Exemplo

Desenvolver um programa que faça uso de uma sub-rotina de função que retorne o valor da adição de dois valores fornecidos como parâmetros.

Algoritmo

O problema proposto é bem simples, pois a função deve receber dois valores e retornar o resultado da adição 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 Adição

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

Programa em Pascal

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 adição dos dois valores e o resultado é implicado na variável ADICAO (nome da função) que efetua o retorno à função, a qual também é do tipo real.

```
program CALC_ADICAO;
uses
  Crt;
function ADICAO(A, B : real) : real;
begin
  ADICAO := A + B;
end;
var
  NUM1, NUM2 : real;
begin
  clrscr;
  write('Informe o 1o. valor: '); readln(NUM1);
  write('Informe o 2o. valor: '); readln(NUM2);
  writeln;
  writeln('O resultado da adicao = ', Adicao(NUM1, NUM2):5:2);
  writeln('Tecle <ENTER> para encerrar o programa'); readln;
end.
```

Feche as janelas que estejam abertas, digite o programa anterior e grave-o com o nome ADICAO2.

2º Exemplo

Para este exemplo deve ser criado um programa que faça uso de uma sub-rotina de função que possibilite comparar dois valores fornecidos, retornando se os valores fornecidos são iguais ou diferentes.

Algoritmo

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 pede os valores e chama 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.

Programa em Pascal

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 implicada à função o retorno da condição, desta forma o retorno desta função deve ser lógico.

```
program COMPARACAO;
uses
  Crt;
function COMPARA(A, B : real) : boolean;
begin
  COMPARA := A = B;
end;
var
  NUM1, NUM2 : real;
begin
```

```
  clrscr;
  write('Informe o 1o. valor: '); readln(NUM1);
  write('Informe o 2o. valor: '); readln(NUM2);
  writeln;
  if (Compara(NUM1, NUM2)) then
     writeln('Os valores fornecidos sao iguais')
  else
     writeln('Os valores fornecidos sao diferentes');
  writeln('Tecle <ENTER> para encerrar o programa'); readln;
end.
```

Feche as janelas que estejam abertas, digite o programa anterior e grave-o com o nome COMPARA.

3º Exemplo

Anteriormente, foi estudado o uso de sub-rotinas do tipo procedimento no programa calculadora. Neste programa, as sub-rotinas de cálculo foram simplificadas com o uso das sub--rotinas de entrada e saída que eram operações genéricas de todas as rotinas de cálculo.

O programa calculadora pode ainda possuir uma sub-rotina 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 deve receber três parâmetros: os dois números mais o operador para cálculo.

1. Se o operador for "+", faz-se a adição 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, a seguir, a estrutura de controle com múltipla escolha **case...of**. Note que está sendo omitida a instrução **else**. Isto é possível e pode ser utilizado quando não se deseja estabelecer uma operação ou execução para alguma opção inválida. Perceba que, neste exemplo, não é necessário, uma vez que o terceiro parâmetro será indicado automaticamente dentro de cada sub-rotina de cálculo.

```
function CALCULO(R, T : real; OPERADOR : char) : real;
begin
  case OPERADOR of
    '+' : CALCULO := R + T;
    '-' : CALCULO := R - T;
    '*' : CALCULO := R * T;
    '/' : CALCULO := R / T;
  end;
end;
```

A seguir, é apresentado o código completo do programa calculadora, fazendo uso da nova função. Digite este programa ou altere o existente, gravando-o com o nome CALCULA4.

```
program CALCULADORA;
uses
  Crt;
var
  OPCAO : char;
  X, A, B : real;
  {*** Rotinas de entrada ***}
  procedure ENTRADA;
  begin
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
  end;
  {*** Funcao para o calculo das operacoes ***}
  function CALCULO(R, T : real; OPERADOR : char) : real;
  begin
    case OPERADOR of
      '+' : CALCULO := R + T;
      '-' : CALCULO := R - T;
      '*' : CALCULO := R * T;
      '/' : CALCULO := R / T;
    end;
  end;
  {*** Rotinas de saida ***}
  procedure SAIDA;
  var
    TECLA   : char;
  begin
    gotoxy( 5,10); write('O resultado equivale a: ', X:5:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
  {*** Rotinas de calculo ***}
  procedure ROT_ADICAO;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Adicao');
    Entrada;
    X := Calculo(A, B, '+');
    Saida;
  end;
  procedure ROT_SUBTRACAO;
  begin
    clrscr;
    gotoxy(30, 1); write('Rotina de Subtracao');
```

```
    Entrada;
    X := Calculo(A, B, '-');
    Saida;
  end;
  procedure ROT_MULTIPLICACAO;
  begin
    clrscr;
    gotoxy(28, 1); write('Rotina de Multiplicacao');
    Entrada;
    X := Calculo(A, B, '*');
    Saida;
  end;
  procedure ROT_DIVISAO;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Divisao');
    Entrada;
    X := Calculo(A, B, '/');
    Saida;
  end;
{*** Programa Principal ***}
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      gotoxy(33, 1); write('Menu Principal');
      gotoxy(28, 6); write('1 ................ Soma');
      gotoxy(28, 8); write('2 ........... Subtracao');
      gotoxy(28,10); write('3 ........ Multiplicacao');
      gotoxy(28,12); write('4 ............. Divisao');
      gotoxy(28,14); write('5 ...... Fim de Programa');
      gotoxy(28,18); write('Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO <> '5') then
        case OPCAO of
           '1' : Rot_Adicao;
           '2' : Rot_Subtracao;
           '3' : Rot_Multiplicacao;
           '4' : Rot_Divisao;
        else
           gotoxy(27,24); writeln('Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

12.3 - EXERCÍCIOS

1. Desenvolva os seguintes programas, usando o conceito de sub-rotina do tipo **Function**.

   a. 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+...+N).

   b. Criar um algoritmo que efetue o cálculo de uma prestação em atraso. Para tanto, utilize a fórmula PREST = VALOR + (VALOR * (TAXA/100) * TEMPO).

   c. Desenvolva um algoritmo de programa que crie uma sub-rotina para calcular o valor de uma potência de um número qualquer, ou seja, ao informar para a sub-rotina o valor da base e de seu expoente, deve ser apresentado o resultado da sua potência. Por exemplo, se for mencionado no programa principal a sub-rotina POTENCIA(2,3), deve ser apresentado o valor 8. Para calcular, não utilize a forma baseada em exp(ln(base)*expoente), utilize apenas laços para efetuar o cálculo.

2. Desenvolver um programa que efetue o gerenciamento de notas escolares de uma escola com 30 salas de aula. Cada sala de aula pode conter até 50 alunos. Assim sendo, a escola pode ter até 1500 alunos. Não está sendo levado em consideração a série à qual o aluno pertence, mas somente a sala em que ele está alocado. O programa em questão deve executar as seguintes rotinas:

   ✓ cadastrar os alunos e suas notas, segundo a sala à qual pertencem;

   ✓ classificar os alunos cadastrados;

   ✓ corrigir alunos e/ou notas que tenham sido cadastrados com erro;

   ✓ pesquisar alunos por nome, fazer uma função que execute a pesquisa seqüencial;

   ✓ pesquisar alunos por sala, fazer uma função que execute a pesquisa binária;

   ✓ listar alunos aprovados, fazer uma função que identifique esta condição.

Construir a Biblioteca de Sub-rotinas do Programador

Uma facilidade encontrada com a utilização de sub-rotinas em Pascal é o fato de criar rotinas que venham facilitar o trabalho do programador. Por exemplo, podem ser criadas procedures que efetuem a centralização de um texto na tela, que escrevam um texto em uma determinada posição da tela sem usar **gotoxy** e **write**, e diversas outras aplicações, sendo que essas sub-rotinas podem ser utilizadas em qualquer programa, pois ao longo de um tempo, esse conjunto de sub-rotinas será a biblioteca de recursos do programador.

Existem duas formas de utilizar as sub-rotinas que sejam genéricas: a primeira definindo cada sub-rotina no início do programa, como visto até aqui, ou seja, dentro do programa, e a segunda forma criando uma biblioteca (unidade padrão) que será utilizada por meio da instrução **uses** (sendo esta compilada e estando fora do programa).

13.1 - Sub-rotinas Dentro do Programa

Este tipo de sub-rotina é mais utilizado para **procedures** e **functions** que sejam específicas do programa que está sendo desenvolvido, mas nada impede de serem utilizadas em qualquer outro programa. Um exemplo deste tipo de rotina são aquelas utilizadas nos três capítulos anteriores.

No caso de uma mesma sub-rotina ser usada em um outro programa, ela deve ser copiada para o novo programa, o que muitas vezes torna-se muito trabalhoso. A seguir, serão apresentados alguns exemplos de sub-rotinas que podem ser utilizadas em um programa, facilitando alguns recursos de programação. As sub-rotinas apresentadas em seguida, serão primeiramente definidas dentro do programa. No próximo tópico, você aprenderá a fazer uso das mesmas rotinas, mas só que fora do programa.

1º Exemplo

No capítulo 10, existe um exemplo do programa calculadora que apresenta a string "Menu Principal" centralizada na tela, utilizando as instruções **gotoxy** (33, 1); **write** ('Menu Principal'), sendo que a instrução **gotoxy** coloca o cursor na linha 1 a partir da coluna 33 e escreve a string. Se você contar o número de caracteres que possui a string **Menu Principal**, verá que ela mede 14, incluindo o espaço em branco. Sabendo que a tela mede 80 colunas, basta subtrair o valor do tamanho da string. Desta forma, 80-14 resulta o valor 66; dividi-se este valor por 2 e ter-se-á 33. Se o valor fosse um número ímpar, seria considerada apenas a parte inteira do resultado.

A procedure seguinte tem por finalidade apresentar um texto centralizado na posição da tela em que o cursor estiver colocado.

```
program CENTRALIZA_TEXTO;
procedure CENTER(MENSAGEM : string);
var
  TAMANHO : integer;
begin
  TAMANHO := 40 + length(MENSAGEM) div 2;
  writeln(MENSAGEM:TAMANHO);
end;
begin
  Center('Centralizacao de uma mensagem');
  readln;
end.
```

Após ter digitado o programa anterior, grave-o com o nome SUBROT01. Não se esqueça de utilizar o comando **File/Save**.

Note que o exemplo anterior apresenta a **procedure** recebendo um parâmetro chamado *MENSAGEM* que é do tipo string. Depois é definida a variável TAMANHO do tipo inteiro. Observe que a variável TAMANHO é implicada em uma operação de cálculo, 40 + **length**(MENSAGEM) **div** 2, em que a instrução **length** faz a contagem, ou seja, retorna o valor do número de caracteres que contém a variável *MENSAGEM*, depois efetua a divisão por 2 por meio da função **div**, que tem por finalidade retornar o valor do resultado sempre inteiro da divisão do tamanho da string por 2, e adicionar esse valor à constante 40.

Por exemplo, considere a variável MENSAGEM contendo a string 'bolha' que possui tamanho 5. Desta forma, a instrução **length**(MENSAGEM) retorna o valor 5, divide 5 por 2, resultando 2 (resultado sempre inteiro). O valor 2 é somado à constante 40 que representa o valor da metade da tela. Desta forma tem-se como resultado o valor 42 que é implicado na variável TAMANHO.

Depois de calculado o valor do tamanho, a instrução **write** (MENSAGEM:TAMANHO) escreve a string contida na variável MENSAGEM da direita para a esquerda, tomando por base o valor da variável TAMANHO. Esta forma de escrita é muito utilizada para apresentar valores numéricos. Tomando por base a string 'bolha', que seria escrita da direita para a esquerda a

partir da 42ª posição, que na prática seria idêntico a utilizar o comando **gotoxy**(38, 1); **write**('bolha'). Perceba que, no caso, está sendo escrita a string da esquerda para a direita, a partir da 38ª posição.

Depois de apresentada a mensagem no meio da tela, o processamento é interrompido devido ao uso da instrução **readln.** Neste caso tecle **<ENTER>** para encerrar o programa.

2º Exemplo

Outra facilidade que pode ser desenvolvida é uma sub-rotina que, além de posicionar o cursor em um determinado ponto da tela, também faça a escrita de uma mensagem. Anteriormente, para conseguir este efeito, foi necessário fazer uso de duas instruções: **gotoxy** e **write**. Neste exemplo será criada a instrução **writexy**, que vai posicionar o cursor e também escrever o texto na posição indicada.

A procedure seguinte tem por finalidade escrever um texto em uma determinada posição da tela.

```
program ESCREVE_POSICIONADO;
uses
  Crt;
procedure WRITEXY(X, Y : byte; MENSAGEM : string);
begin
  gotoxy(X, Y); write(MENSAGEM);
end;
begin
  clrscr;
  WriteXy(4,6,'Texto posicionado na coluna 4 linha 6');
  readln;
end.
```

Após ter digitado o programa acima, grave-o com o nome SUBROT02. Não se esqueça de utilizar o comando **File/Save**.

Observe que foi criada a instrução **writexy** que, na realidade, quando é chamada, executa as instruções **gotoxy** e **write**. Porém, com a definição deste recurso, fica mais fácil escrever um programa que faça uso dele.

3º Exemplo

A seguir, é apresentado um exemplo, bastante simples, de como conseguir uma linha tracejada na tela. O nosso tracejado é conseguido com a utilização do símbolo "-", que representa o sinal matemático para operações de subtração.

A procedure seguinte tem por finalidade apresentar uma linha tracejada, que vai cobrir a extensão de uma linha nas 80 colunas do vídeo.

```
program TRACA_LINHA;
procedure LINE;
var
  I : byte;
begin
  for I := 1 to 80 do
    write('-');
end;
begin
  Line;
  writeln('Apresentacao de linha tracejada');
  Line;
  readln;
end.
```

Após ter digitado o programa acima, grave-o com o nome SUBROT03. Não se esqueça de utilizar o comando **File/Save**.

Observe que a instrução **Line** apresenta uma linha tracejada em que o cursor estiver posicionado. O tracejado pode ser substituído por qualquer caractere do teclado, mas existem alguns caracteres que não são gerados pelo teclado. Por exemplo, a string '-' pode ser substituída por uma linha dupla contínua. Isto pode ser conseguido com a utilização do código 205. Para tanto, troque o símbolo de subtração por #205. Neste caso, a linha da **procedure** que está indicada **write('-')** passa a ser escrita como **write(#205)**.

4º Exemplo

Nos três exemplos anteriores, você teve um contato com uma técnica bastante útil, que agora será aplicada no programa calculadora, já existente. Você pode carregá-lo para a memória e fazer as alterações indicadas. Neste caso, não se esqueça de carregar a quarta versão do programa: CALCULA4. A seguir, é apresentado o programa completo. Depois de efetuar as alterações, utilizando o comando **File/Save as**, grave o programa com o nome CALCULA5.

```
program CALCULADORA;
uses
  Crt;
var
  OPCAO : char;
  X, A, B : real;
  {*** Rotinas definidas pelo usuario ***}
  procedure CENTER(MENSAGEM : string);
  var
    TAMANHO : integer;
  begin
    TAMANHO := 40 + length(MENSAGEM) div 2;
    writeln(MENSAGEM:TAMANHO);
  end;
  procedure WRITEXY(X, Y : byte; MENSAGEM : string);
```

```
begin
  gotoxy(X, Y); write(MENSAGEM);
end;
procedure LINE;
var
  I : byte;
begin
  for I := 1 to 80 do
    write('-');
end;
{*** Rotinas de entrada ***}
procedure ENTRADA;
begin
  WriteXy( 5, 6,'Entre um valor para A: '); readln(A);
  WriteXy( 5, 7,'Entre um valor para B: '); readln(B);
end;
{*** Funcao para o calculo das operacoes ***}
function CALCULO(R, T : real; OPERADOR : char) : real;
begin
  case OPERADOR of
    '+' : CALCULO := R + T;
    '-' : CALCULO := R - T;
    '*' : CALCULO := R * T;
    '/' : CALCULO := R / T;
  end;
end;
{*** Rotinas de saida ***}
procedure SAIDA;
var
  TECLA    : char;
begin
  gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
  WriteXy(25,24,'Tecle algo para voltar ao menu');
  TECLA := readkey;
end;
{*** Rotinas de calculo ***}
procedure ROT_ADICAO;
begin
  clrscr;
  Line;
  Center('Rotina de Adicao');
  Line;
  Entrada;
  X := Calculo(A, B, '+');
  Saida;
end;
procedure ROT_SUBTRACAO;
begin
  clrscr;
```

```
    Line;
    Center('Rotina de Subtracao');
    Line;
    Entrada;
    X := Calculo(A, B, '-');
    Saida;
  end;
  procedure ROT_MULTIPLICACAO;
  begin
    clrscr;
 Line;
    Center('Rotina de Multiplicacao');
    Line;
    Entrada;
    X := Calculo(A, B, '*');
    Saida;
  end;
  procedure ROT_DIVISAO;
  begin
    clrscr;
    Line;
    Center('Rotina de Divisao');
    Line;
    Entrada;
    X := Calculo(A, B, '/');
    Saida;
  end;
{*** Programa Principal ***}
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      Line;
      Center('Menu Principal');
      Line;
      WriteXy(28, 6,'1 ................. Soma');
      WriteXy(28, 8,'2 ............ Subtracao');
      WriteXy(28,10,'3 ........ Multiplicacao');
      WriteXy(28,12,'4 .............. Divisao');
      WriteXy(28,14,'5 ...... Fim de Programa');
      WriteXy(28,18,'Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO <> '5') then
        case OPCAO of
           '1' : Rot_Adicao;
           '2' : Rot_Subtracao;
           '3' : Rot_Multiplicacao;
           '4' : Rot_Divisao;
```

```
        else
           WriteXy(27,24,'Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

13.2 - SUB-ROTINAS FORA DO PROGRAMA

Este tipo de sub-rotina é mais utilizado para **procedures** e **functions** que sejam genéricas, e desta forma podem ser utilizadas em qualquer programa, uma vez que serão compiladas e utilizadas por meio da instrução **uses,** como é feito com a unidade **Crt**. No capítulo 9, você teve uma rápida abordagem das unidades padrão do Turbo Pascal.

Neste tópico, você aprenderá a construir uma unidade padrão definida pelo programador, que pode ser utilizada a qualquer momento e em qualquer programa. Assim sendo, as sub-rotinas (**procedures** e/ou **functions**) serão definidas fora do programa principal.

Para definir um programa como sendo uma unidade padrão (biblioteca de sub-rotinas externa), substitui-se a instrução **program** por **unit**. O nome do programa a ser gravado deve ser o mesmo nome mencionado após a instrução **unit**.

Em seguida o programa é dividido em duas seções, sendo a primeira denominada **interface** e a segunda denominada **implementation**.

Interface

Essa parte do programa apresenta uma relação de todos os elementos públicos existentes na unidade. Nessa seção podem ser definidas constantes, variáveis, tipos, *procedures* e *functions* que estarão disponíveis para uso de outros programas. No caso de *procedures* e *functions*, ficam relacionadas em uma lista que contém os cabeçalhos de todas as *functions* e *procedures* que se deseja deixar disponíveis para uso externo.

Implementation

Essa parte do programa contém o código das rotinas externas, que podem ser públicas (quando mencionadas na seção *interface*, e assim estarem disponíveis para uso por outros programas) ou privadas (rotinas que somente serão usadas pela própria unidade, e não estarão disponíveis para uso externo). Nessa seção podem ser definidas constantes, variáveis, tipos, procedures, functions e códigos-fonte.

Uma **unit** é finalizada por meio da instrução **end** com ponto final. Para criação de uma UNIT, deve-se alterar o destino da compilação para DISCO. Para tanto, execute o comando de menu **Compile** e verifique se em **Destination** está indicado **Disk**. Caso esteja

Memory, selecione o comando **Destination** para ativar o disco e assim a unidade ser gravada.

Após a construção do programa que será usado como unidade, ele não será executado por meio do comando **Run/Run**. Apenas será compilado com o comando **Compile/Compile** ou **<Alt> + <F9>**. Um programa convencional, quando é compilado, gera um arquivo com a extensão **.EXE** e no caso de uma unidade padrão, será gerado um arquivo compilado com a extensão **.TPU**[6], a qual deve estar presente no diretório em que o programa for compilado, ou no diretório padrão do Turbo Pascal 7.0, **TP\UNITS**.

Se a biblioteca a ser gerada for muito grande, é ideal que o programador crie um manual, contendo o nome da rotina, descrição, função e até exemplos de uso dela. Em seguida, é apresentado um exemplo das sub-rotinas desenvolvidas no tópico anterior, como uma unidade padrão. O programa a ser gerado será denominado UDP (Unidade Definida pelo Programador), que deve ser a biblioteca mencionada no programa principal.

```
unit UDP;
  interface
    procedure CENTER(MENSAGEM : string);
    procedure WRITEXY(X, Y : byte; MENSAGEM : string);
    procedure LINE;
    function CALCULO(R, T : real; OPERADOR : char) : real;
  implementation
    uses Crt;
    procedure CENTER(MENSAGEM : string);
    var
      TAMANHO : integer;
    begin
      TAMANHO := 40 + length(MENSAGEM) div 2;
      writeln(MENSAGEM:TAMANHO);
    end;
    procedure WRITEXY(X, Y : byte; MENSAGEM : string);
      begin
      gotoxy(X, Y); write(MENSAGEM);
    end;
    procedure LINE;
    var
      I : byte;
    begin
      for I := 1 to 80 do
        write('-');
    end;
```

6 No caso de unidades compiladas com o **Free Pascal Compiler**, elas terão a extensão **.PPU**.

```
    function CALCULO(R, T : real; OPERADOR : char) : real;
    begin
      case OPERADOR of
        '+' : CALCULO := R + T;
        '-' : CALCULO := R - T;
        '*' : CALCULO := R * T;
        '/' : CALCULO := R / T;
      end;
    end;
end.
```

Após ter digitado este programa, grave-o com o nome UDP (mesmo nome atribuído após a instrução **unit**), com o comando **File/Save**, em seguida execute o comando **Run/Compile** para ser compilada e gerada a nova unidade.

Carregue para a memória o programa CALCULA5, retirando dele as sub-rotinas CENTER, LINE, WRITEXY e CALCULO, gravando em seguida o programa com o nome CALCULA6. Veja, a seguir, como deve ficar a nova versão do programa e observe que a nova unidade é mencionada ao lado da unidade **Crt**.

```
program CALCULADORA;
uses
  Crt, Udp;
var
  OPCAO : char;
  X, A, B : real;
  {*** Rotinas de entrada ***}
  procedure ENTRADA;
  begin
    WriteXy( 5, 6,'Entre um valor para A: '); readln(A);
    WriteXy( 5, 7,'Entre um valor para B: '); readln(B);
  end;
  {*** Rotinas de saida ***}
  procedure SAIDA;
  var
    TECLA    : char;
  begin
    gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
    WriteXy(25,24,'Tecle algo para voltar ao menu');
    TECLA := readkey;
  end;
  {*** Rotinas de calculo ***}
  procedure ROT_ADICAO;
  begin
    clrscr;
    Line;
    Center('Rotina de Adicao');
    Line;
    Entrada;
```

```
    X := Calculo(A, B, '+');
    Saida;
  end;
  procedure ROT_SUBTRACAO;
  begin
    clrscr;
    Line;
    Center('Rotina de Subtracao');
    Line;
    Entrada;
    X := Calculo(A, B, '-');
    Saida;
  end;
  procedure ROT_MULTIPLICACAO;
  begin
    clrscr;
    Line;
    Center('Rotina de Multiplicacao');
    Line;
    Entrada;
    X := Calculo(A, B, '*');
    Saida;
  end;
  procedure ROT_DIVISAO;
  begin
    clrscr;
    Line;
    Center('Rotina de Divisao');
    Line;
    Entrada;
    X := Calculo(A, B, '/');
    Saida;
  end;
{*** Programa Principal ***}
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      Line;
      Center('Menu Principal');
      Line;
      WriteXy(28, 6,'1 ................. Soma');
      WriteXy(28, 8,'2 ............ Subtracao');
      WriteXy(28,10,'3 ........ Multiplicacao');
      WriteXy(28,12,'4 ............... Divisao');
      WriteXy(28,14,'5 ...... Fim de Programa');
      WriteXy(28,18,'Escolha uma opcao ....: ');
      readln(OPCAO);
```

```
      if (OPCAO <> '5') then
        case OPCAO of
           '1' : Rot_Adicao;
           '2' : Rot_Subtracao;
           '3' : Rot_Multiplicacao;
           '4' : Rot_Divisao;
        else
           WriteXy(27,24,'Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

Observação: *Para maiores detalhes a respeito deste recurso, aconselha-se um estudo mais detalhado dos manuais do ambiente de programação Turbo Pascal 7.0.*

Parte 5

Programação com Arquivos

```
procedure CALC_ADICAO(A, B :
var
  Z : integer;
begin
  Z := A + B;
  writeln(Z);
end;
```

Capítulo 14 Arquivos

Nos capítulos 6 e 7, foram estudados os conceitos de tabelas em memória por meio da utilização de matrizes. Depois, no capítulo 8, foi estudado o conceito de utilização do tipo definido: registro. Esses três capítulos formaram a base para a utilização de um arquivo que é, na verdade, uma tabela de informações gravada em um meio físico (disquetes, winchesters, fitas magnéticas, entre outros), enquanto as matrizes são tabelas manipuladas na memória RAM[7]. As matrizes são manipuladas por intermédio de um índice de controle, enquanto os arquivos são manipulados por um ponteiro de registro[8].

A principal vantagem na utilização de um arquivo está no fato de as informações armazenadas poderem ser utilizadas a qualquer momento. Outra vantagem encontrada na utilização de arquivos é o fato de poder armazenar um número maior de registros do que em uma tabela em memória, estando apenas limitado ao tamanho do meio físico utilizado para a sua gravação.

14.1 - Definição de Um Arquivo

A manipulação de um arquivo em Pascal ocorre com a definição do tipo **file**, o qual se caracteriza por ser uma estrutura formada por elementos do mesmo tipo, dispostos de forma seqüencial, tendo como objetivo fazer a comunicação entre a memória principal (RAM) e a memória secundária (meios magnéticos). Assim sendo, este tipo deve ser definido com a utilização da seguinte sintaxe em Pascal:

---

7 RAM - Random Access Memory (Memória de Acesso Randômico). Este tipo de memória mantém as informações armazenadas enquanto o computador se mantém ligado.

8 Um arquivo é um conjunto de registros (que pode ser apenas um registro) que, por sua vez, é um conjunto de campos, sendo cada campo o conjunto de informações nele contido segundo seu tipo (caractere ou numérico). As informações NOME, ENDEREÇO, TELEFONE, IDADE são exemplos de campos. Uma ficha que contenha os campos para preenchimento é o registro.

```
type
  <arquivo> = [text][file [of <tipo>]];

var
  <variável> : <arquivo>;
```

ou

```
var
  <variável> : [text][file [of <tipo>]];
```

em que:

<arquivo> - o nome de um arquivo com tipo definido;

<tipo> - *o tipo de um arquivo (text, string, real, record, etc);*

<variável> - a variável que será usada para representar o arquivo.

Com relação ao tipo de um arquivo em Pascal, ele pode se caracterizar em três tipos: arquivos de tipo texto, arquivos de tipo definido e arquivos sem tipo definido, que serão estudados em separado, pois assim como existem os tipos de dados, existem também os tipos de arquivo, uma vez que são criados para armazenar os dados trabalhados em um programa.

14.2 - Operações Executadas em um Arquivo

Independentemente do tipo de um arquivo, ele tem a capacidade de executar algumas operações em comum, tais como: abertura, leitura ou escrita e fechamento de um arquivo. Essas operações são conseguidas na linguagem Pascal com a utilização de algumas instruções apropriadas: **assign**, **rewrite**, **reset**, **write** (**writeln**), **read** (**readln**) e **close**, que podem ser utilizadas em qualquer tipo de arquivo:

Instrução	Tipo	Sintaxe	Descrição
assign	Procedure	**assign**(<variável>,<arquivo>)	Esta instrução tem por finalidade associar um nome lógico de arquivo ao arquivo físico. O parâmetro <variável> é a indicação da variável do tipo arquivo e <arquivo> é o nome do arquivo a ser manipulado.
rewrite	Procedure	**rewrite**(<variável>)	Esta instrução tem por finalidade criar um arquivo para uso, utilizando o nome associado ao parâmetro <variável>. Caso o arquivo já exista, esta instrução o apaga para criá-lo novamente.

Instrução	Tipo	Sintaxe	Descrição
Reset	Procedure	**reset**(<variável>)	Esta instrução tem por finalidade abrir um arquivo existente, colocando-o disponível para leitura e escrita, utilizando o nome associado ao parâmetro <variável>.
Write	Procedure	**write**(<variável>,<dado>)	Esta instrução tem por finalidade escrever a informação <dado> no arquivo indicado pelo parâmetro <variável>.
Read	Procedure	**read**(<variável>,<dado>)	Esta instrução tem por finalidade ler a informação <dado> no arquivo indicado pelo parâmetro <variável>.
Close	Procedure	**close**(<variável>)	Esta instrução tem por finalidade fechar um arquivo em uso dentro de um programa. Nenhum programa deve ser encerrado sem antes fechar os arquivos abertos

Os programas de manipulação de arquivos executam sempre uma de duas operações: abertura, escrita e fechamento ou abertura, leitura e fechamento, desde que o arquivo exista.

Existem outras instruções para o processo de manipulação de arquivos, as quais serão vistas em separado para cada tipo de arquivo que será estudado.

14.3 - Formas de Acesso em um Arquivo

Os arquivos criados em meios magnéticos podem ser acessados para leitura e/ou escrita na forma seqüencial, direta ou indexada. Estas formas serão, neste capítulo, apenas conceituadas, ficando sua utilização a ser exemplificada mais adiante.

14.3.1 - Arquivo de Acesso Seqüencial

O acesso seqüencial ocorre quando o processo de gravação e leitura é feito de forma contínua, um arquivo após o outro. Desta forma, para gravar um novo registro, é necessário percorrer todo o arquivo a partir do primeiro registro, registro a registro, até localizar a primeira posição vazia após o último registro. O processo de leitura também ocorre de forma seqüencial. Se o registro a ser lido é o último, primeiro é necessário ler todos os registros que o antecedem.

14.3.2 - Arquivo de Acesso Direto

O acesso direto (conhecido também por acesso randômico) ocorre por meio de um campo *chave*[9] previamente definido. Desta forma, passa-se a possuir um vínculo existente entre um dos campos do registro e sua posição de armazenamento, por intermédio da *chave*. Assim sendo, o acesso a um registro tanto para leitura como para escrita pode ser feito de forma instantânea. Isto implica no fato de que os registros de um arquivo direto possuem um lugar (chave) previamente "reservado" para serem armazenados.

14.3.3 - Arquivo de Acesso Indexado

O acesso indexado ocorre quando se acessa de forma direta um arquivo seqüencial. Na sua maioria, todo arquivo criado armazena os registros de forma seqüencial. A forma seqüencial de acesso se torna inconveniente, pois à medida que o arquivo aumenta de tamanho, aumenta também o tempo de acesso a ele.

Para trabalhar com esta técnica, basta criar um arquivo direto que será o índice de consulta do arquivo seqüencial, passando a ser o arquivo indexado. Assim sendo, existirão dois arquivos: o arquivo índice (arquivo direto) e o indexado (arquivo seqüencial). Os acessos são feitos como em um livro. Primeiro se consulta o arquivo índice, o qual possui a chave de pesquisa, no caso seria o número da página, depois basta se posicionar de forma direta na página identificada no índice, ou seja, no registro do arquivo indexado. Este tipo de arquivo não será abordado nesta obra.

---

9 O termo chave, é utilizado para estabelecer o campo de um registro que será utilizado no processo de localização de todo um registro dentro de um arquivo. Desta forma, é possível acessar um determinado registro diretamente, sem nos preocuparmos com os registros que o antecedem. Por exemplo, em um cadastro de funcionários, pode ser utilizado o campo reservado para a matrícula, como sendo a chave para a manipulação dele.

Capítulo 15 — Arquivos do Tipo Texto

Este capítulo aborda o uso prático do tipo de arquivo mais simples utilizado em Pascal, o tipo texto que permite armazenar registros com tamanhos diferentes (o que não ocorre com os outros tipos de arquivo), sendo que os registros são finalizados com uma seqüência CR/LF, Carriege Return (equivalente ao efeito da tecla <ENTER>) e Line Feed (movimento do ponteiro de registro para a próxima linha). Se você direcionar o prompt do DOS para o diretório **TP**, terá como exemplo de um arquivo texto o **FILELIST.DOC** que pode ser visualizado com o comando DOS: **type filelist.doc**.

Os arquivos do tipo texto estão capacitados a armazenar palavras, frases e também dados numéricos. Os números, entretanto, serão armazenados como um caractere do tipo alfanumérico. Porém, ao serem lidos de um arquivo e passados para a memória, os dados numéricos são convertidos no seu formato original.

15.1 - Criando um Arquivo

Antes de iniciar qualquer operação com arquivo, é necessário criá-lo. Para tanto, digite o programa abaixo e grave-o com o nome ARQTXT01.

```
program CRIA_ARQUIVO_TEXTO;
var
  ARQUIVO_TXT : text;
begin
  assign(ARQUIVO_TXT, 'ARQTXT.XXX');
  rewrite(ARQUIVO_TXT);
  close(ARQUIVO_TXT);
end.
```

Após rodar o programa com o comando **Run/Run**, execute o comando **File/DOS Shell** para visualizar o prompt do DOS. Em seguida execute o comando DOS: **dir arqtxt01.xxx** e você verá a apresentação do arquivo **arqtxt01.xxx** criado com zero bytes. Para voltar ao Turbo Pascal, execute o comando DOS: **exit**.

O programa acima estabelece para a variável *ARQUIVO_TXT* o identificador **text**, que tem por finalidade definir para a variável indicada o tipo de arquivo texto.

Em seguida é utilizada a instrução **assign(***ARQUIVO_TXT*, 'ARQTXT.XXX'**);** que efetua a associação do nome do arquivo (ARQTXT.XXX) com a variável de controle *ARQUIVO_TXT*, a qual será utilizada em todas as operações do programa para fazer referência ao arquivo em uso. Depois é utilizada a instrução **rewrite(***ARQUIVO_TXT***);** que cria o arquivo ARQTXT.XXX, mantendo-o aberto. Por fim é utilizada a instrução **close(***ARQUIVO_TXT***);** que efetua o fechamento do arquivo criado.

15.2 - GRAVANDO INFORMAÇÕES EM UM ARQUIVO

Tendo sido o arquivo criado, ele pode ser agora utilizado para a gravação das informações que vai guardar. Digite o programa abaixo e grave-o com o nome ARQTXT02.

```
program GRAVA_ARQUIVO_TEXTO;
var
  ARQUIVO_TXT : text;
  MENSAGEM : string[50];
begin
  assign(ARQUIVO_TXT, 'ARQTXT.XXX');
  append(ARQUIVO_TXT);
  readln(MENSAGEM);
  writeln(ARQUIVO_TXT, MENSAGEM);
  close(ARQUIVO_TXT);
end.
```

Após rodar o programa, execute novamente o comando **File/DOS Shell** para visualizar o prompt do DOS. Peça o comando DOS: **dir *.xxx** e você verá a apresentação do arquivo **arqtxt01.xxx** com um número de bytes. Não se esqueça de voltar para o Turbo Pascal.

O programa anterior estabelece uma variável *MENSAGEM* do tipo **string** com a capacidade de armazenar até 50 caracteres para a variável *ARQUIVO_TXT* o identificador **text**, que tem por finalidade definir para a variável o tipo de arquivo texto.

Em seguida é utilizada novamente a instrução **assign(***ARQUIVO_TXT*, 'ARQTXT.XXX'**);**. Depois é utilizada a instrução **append(***ARQUIVO_TXT***);**, que abre o arquivo ARQTXT.XXX para a inclusão de um dado, posicionando o ponteiro de controle de registros sempre no final do arquivo, ou seja, se o arquivo já possuir algum conteúdo, o próximo será sempre colocado no final do último cadastrado anteriormente.

A instrução **readln** solicita a entrada de um dado, depois a instrução **writeln(***ARQUIVO_TXT*, *MENSAGEM***);** faz a gravação no arquivo representado pela variável de controle *ARQUIVO_TXT*, do conteúdo armazenado na variável *MENSAGEM*. Observe que a instrução **write** tinha sido usada até este momento para escrever dados na tela, e agora está sendo utilizada para escrever uma informação no arquivo em uso. Ao final o arquivo é fechado.

Observação: *Quando a instrução* ***write*** *ou a* ***writeln*** *é utilizada para escrever um dado em um arquivo, ela não apresenta o dado no vídeo, ficando disponível apenas para a operação de escrita do arquivo.*

15.3 - LENDO INFORMAÇÕES DE UM ARQUIVO

A seguir, é apresentado um programa que vai efetuar a leitura de um arquivo texto mostrando no vídeo a informação armazenada com o programa anterior. Digite o programa abaixo e grave-o com o nome ARQTXT03.

```
program LE_ARQUIVO_TEXTO;
var
  ARQUIVO_TXT : text;
  MENSAGEM : string[50];
begin
  assign(ARQUIVO_TXT, 'ARQTXT.XXX');
  reset(ARQUIVO_TXT);
  readln(ARQUIVO_TXT, MENSAGEM);
  writeln(MENSAGEM);
  close(ARQUIVO_TXT);
end.
```

A instrução **reset(***ARQUIVO_TXT***);** faz apenas a abertura do arquivo. Não confundir com a instrução **append** que faz a abertura do arquivo, posicionando o ponteiro de controle sempre após a última linha existente no arquivo. Depois a instrução **readln(***ARQUIVO_TXT, MENSAGEM***);** faz a leitura do registro no arquivo, transferindo a informação lida para a variável *MENSAGEM,* que será apresentada no vídeo pela instrução **writeln(***MENSAGEM* **);**.

15.4 - UTILIZAÇÃO DE ARQUIVOS DO TIPO TEXTO

Neste tópico você terá contato com dois exemplos de utilização de arquivo texto. Um exemplo é um pequeno programa de agenda e o segundo exemplo é um programa de alteração para o arquivo **autoexec.bat**.

1º Exemplo

O programa de agenda proposto deve solicitar apenas o nome e o telefone de uma pessoa. Esse programa deve ter um menu contendo quatro opções: criar arquivo, cadastrar registro, exibir registros cadastrados e finalizar o programa. Para tanto, observe o algoritmo seguinte:

Programa Principal

1. Definir as variáveis de controle e abertura do arquivo;
2. Apresentar um menu de seleção com quatro opções:
   1. Criar arquivo
   2. Cadastrar registro
   3. Exibir registros
   4. Fim de Programa
3. Ao ser selecionado um valor, a rotina correspondente deve ser executada;
4. Ao escolher o valor 4, encerrar o programa.

Rotina 1 - Criar arquivo

1. Executar o comando de criação de um arquivo;
2. Fechar o arquivo;
3. Voltar ao programa principal.

Rotina 2 - Cadastrar registro

1. Abrir o arquivo para cadastramento, posicionando o ponteiro após o último registro;
2. Ler o nome e o telefone;
3. Escrever os dados no arquivo;
4. Fechar o arquivo;
5. Voltar ao programa principal.

Rotina 3 - Exibir registro

1. Abrir o arquivo para leitura, posicionando o ponteiro no primeiro registro;
2. Apresentar os dados do arquivo enquanto não for encontrado o último registro;
3. Fechar o arquivo;
4. Voltar ao programa principal.

Apesar de simples, o programa dá uma boa visão, pois é necessário trabalhar com alguns novos recursos. Digite o seguinte programa e grave-o com o nome AGENDTXT.

```
program AGENDATXT;
uses
  Crt;
var
  ARQTXT   : text;
  NOME     : string[40];
  TELEFONE : string[8];
  TECLA    : char;
  OPCAO    : char;

{*** Rotinas de Visualizacao ***}

procedure CENTER(MENSAGEM : string);
var
  TAMANHO : integer;
begin
  TAMANHO := 40 + length(MENSAGEM) div 2;
  writeln(MENSAGEM:TAMANHO);
end;

procedure WRITEXY(X, Y : byte; MENSAGEM : string);
begin
  gotoxy(X, Y); write(MENSAGEM);
end;

procedure LINE;
var
  I : byte;
begin
  for I := 1 to 80 do
    write(#205);
end;

{*** Rotinas de Manipulacao de Arquivos ***}

procedure ARQUIVO;
begin
  clrscr;
  line;
  center('Criacao de Arquivo');
  line;
  rewrite(ARQTXT);
  gotoxy( 1,12); center('Arquivo foi criado');
  writexy(25,24,'Tecle algo para voltar ao menu');
  TECLA := readkey;
  close(ARQTXT);
end;

procedure CADASTRA;
begin
  clrscr;
```

```
  line;
  center('Cadastramento de Registro');
  line;
  append(ARQTXT);
  writexy(10, 5, 'Entre com o Nome ..........: '); readln(NOME);
  writexy(10, 6, 'Entre com o Telefone ......: '); readln(TELEFONE);
  writeln(ARQTXT, NOME);
  writeln(ARQTXT, TELEFONE);
  writexy(25,24,'Tecle algo para voltar ao menu');
  TECLA := readkey;
  close(ARQTXT);
end;

procedure EXIBIR;
var
  LINHA : byte;
begin
  clrscr;
  line;
  center('Apresentacao de Registros');
  line;
  LINHA := 5;
  reset(ARQTXT);
  while not eof(ARQTXT) do
    begin
      readln(ARQTXT, NOME);
      readln(ARQTXT, TELEFONE);
      gotoxy( 5,LINHA); write(NOME);
      gotoxy(50,LINHA); write(TELEFONE);
      LINHA := LINHA + 1;
    end;
  writexy(25,24,'Tecle algo para voltar ao menu');
  TECLA := readkey;
  close(ARQTXT);
end;

{*** Programa Principal ***}

begin
  OPCAO := '0';
  assign(ARQTXT, 'AGENDTXT.DAT');
  while (OPCAO <> '4') do
    begin
      clrscr;
      line;
      center('Menu Principal');
      line;
      gotoxy(28, 6); write('1 ........ Criar arquivo');
      gotoxy(28, 8); write('2 ............ Cadastrar');
      gotoxy(28,10); write('3 ..... Exibir registros');
      gotoxy(28,12); write('4 ...... Fim de Programa');
```

```
      gotoxy(28,16); write('Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO <> '4') then
        case OPCAO of
           '1' : Arquivo;
           '2' : Cadastra;
           '3' : Exibir;
        else
           gotoxy(27,24); writeln('Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

Ao fazer uso do programa, procure cadastrar poucos registros (ideal em torno de 15), pois não está sendo previsto exibir uma quantidade muito grande de dados na tela. Perceba que o programa principal associa a variável *ARQTXT* ao arquivo AGENDTXT.DAT, que será usado nas rotinas de manipulação.

Na rotina EXIBIR está sendo usada uma nova instrução, **eof()**, sendo uma função que consegue identificar o final do arquivo (eof - end of file). Perceba que serão apresentados na tela os registros enquanto não for o fim de arquivo. Quando um arquivo é fechado, ele grava na última posição um caractere de controle que pode ser identificado pela função **eof()**. No restante, os demais recursos do programa já são conhecidos. Lembre-se ainda que a instrução **reset()** faz a abertura do arquivo, colocando o ponteiro de registro posicionado na primeira linha de texto.

Procure utilizar o programa acima fazendo primeiro a criação do arquivo, execute alguns cadastramentos e encerre o programa. Sai do Turbo Pascal. Depois volte ao Turbo Pascal, chame o programa e peça para visualizar os dados cadastrados. Cadastre mais alguns e visualize novamente. Preste atenção ainda em um detalhe, o programa utilizado parece manipular uma matriz. Depois de cadastrar alguns registros, se for solicitado efetuar a criação do arquivo novamente, ele será recriado, destruindo os dados já cadastrados.

2º Exemplo

No segundo exemplo será confeccionado um programa que vai escrever uma mensagem dentro do arquivo **autoexec.bat** do seu sistema. Este tipo de operação deve ser feito com muito cuidado. Nosso objetivo com este exemplo é mostrar ao leitor que o recurso de manipulação de arquivo texto é bastante poderoso, apesar de ser considerado a estrutura de arquivo mais simples em Pascal. Digite o programa seguinte e grave-o com o nome ALTREXEC.

```
program ALTERA_AUTOEXEC;
var
  ARQ : text;
begin
  assign(ARQ, '\AUTOEXEC.BAT');
```

```
  append(ARQ);
  write(ARQ, 'REM TESTE DE ALTERACAO DE AUTOEXEC');
  close(ARQ);
end.
```

Após executar o programa acima, saia para o sistema operacional, posicione o prompt na raiz e execute o comando DOS: **type autoexec.bat**. Observe a inserção da mensagem na última linha do seu arquivo, depois volte para o ambiente do Turbo Pascal.

15.5 - EXERCÍCIO

Construa um programa que apresente um menu contendo quatro opções: criar arquivo, cadastrar palavras, exibir palavras e finalizar. Na rotina de cadastro, o programa deve cadastrar as palavras uma após a outra. Quando for digitada a instrução **fim**, a rotina deve ser encerrada e retornar ao menu. As demais rotinas são semelhantes ao exemplo anteriormente mostrado. Tome cuidado toda vez que for digitada a palavra **fim** na rotina de cadastro, para que ela não seja, em hipótese alguma, cadastrada no arquivo, pois está sendo usada apenas no sentido de uma instrução do programa.

Capítulo 16

Arquivos com Tipo Definido

Os arquivos com tipo definido também são denominados, em Pascal, de arquivos tipados, e caracterizam-se por serem do tipo binário, diferentes dos arquivos do tipo texto. Eles permitem armazenar dados dos seguintes tipos: **integer**, **real**, **record**, entre outros.

Um arquivo com tipo definido executa as operações de escrita e leitura mais rápido do que os arquivos textos. Eles estão capacitados a armazenar dados na forma de registro.

16.1 - Criando um Arquivo

Antes de mais nada será criado um arquivo que esteja capacitado a receber dados numéricos inteiros. Para tanto, digite o programa abaixo e grave-o com o nome ARQINT01.

```
program CRIA_ARQUIVO_INTEIRO;
var
  ARQUIVO_INT : file of integer;
begin
  assign(ARQUIVO_INT, 'ARQINT.XXX');
  rewrite(ARQUIVO_INT);
  close(ARQUIVO_INT);
end.
```

Após rodar o programa, o arquivo será criado. Caso queira verificar se o arquivo foi realmente criado, saia temporariamente para o sistema operacional e execute o comando DOS: **dir arqint.xxx**.

O programa acima estabelece para a variável *ARQUIVO_INT* o identificador **file of integer**, que tem por finalidade definir para a variável indicada o tipo de arquivo como sendo inteiro. Todas as outras operações são semelhantes às usadas para criar um arquivo texto.

16.2 - GRAVANDO INFORMAÇÕES EM UM ARQUIVO

Estando o arquivo criado, ele pode ser agora utilizado para a gravação dos dados de tipo inteiro. Digite o programa seguinte e grave-o com o nome ARQINT02.

```
program CADASTRA_ARQUIVO_INTEIRO;
uses
  Crt;
var
  ARQUIVO_INT : file of integer;
  NUMERO : integer;
  RESP   : char;
begin
  assign(ARQUIVO_INT, 'ARQINT.XXX');
  reset(ARQUIVO_INT);
  RESP := 'S';
  while (RESP = 'S') or (RESP = 's') do
    begin
      clrscr;
      writeln('Gravacao de Registros Inteiros');
      writeln;
      seek(ARQUIVO_INT, filesize(ARQUIVO_INT));
      write('Informe um numero inteiro: '); readln(NUMERO);
      write(ARQUIVO_INT, NUMERO);
      writeln;
      write('Deseja continuar? (S/N) '); readln(RESP);
    end;
  close(ARQUIVO_INT);
end.
```

O programa acima utilizada a instrução **assign(***ARQUIVO_INT*, 'ARQINT.XXX'**);** para associar o arquivo à variável de controle. Faz uso da instrução **reset(***ARQUIVO_INT***);** para abrir o arquivo. Depois, um pouco mais abaixo, é apresentada uma linha com a instrução **seek(***ARQUIVO_INT*, **filesize(***ARQUIVO_INT***));**, em que esta linha executa em um arquivo tipado o mesmo efeito que a instrução **append** executa em um arquivo texto, ou seja, se o arquivo já possuir algum registro, o próximo registro será sempre colocado logo após o último cadastrado.

A instrução **seek(***ARQUIVO_INT*, **filesize(***ARQUIVO_INT***));** é formada por duas funções distintas, sendo **FileSize()** a função destinada a retornar o número de registros de um arquivo e a função **Seek()** utilizada para colocar o ponteiro de registro em uma determinada posição do arquivo.

> ***Observação:*** *A função **FileSize()** vai retornar o valor zero, caso o arquivo esteja vazio. Utilizada com arquivos não tipados, esta função considera que os registros desse arquivo possuem o tamanho de 128 caracteres. Se o arquivo em uso tiver um tamanho maior que 32 Kb, você deve usar a função **LongFileSize()**. No caso da função **Seek()**, deve ser utilizada a função **LongSeek()** caso o arquivo seja maior que 32 KB.*

16.3 - LENDO INFORMAÇÕES DE UM ARQUIVO

A seguir, é apresentado um programa que vai efetuar a leitura de um arquivo de números inteiros, mostrando no vídeo a informação armazenada com o programa anterior. Digite o seguinte programa e grave-o com o nome ARQINT03.

```
program LE_ARQUIVO_INTEIRO;
uses
  Crt;
var
  ARQUIVO_INT : file of integer;
  NUMERO : integer;
begin
  clrscr;
  assign(ARQUIVO_INT, 'ARQINT.XXX');
  reset(ARQUIVO_INT);
  while not eof(ARQUIVO_INT) do
    begin
      read(ARQUIVO_INT, NUMERO);
      writeln(NUMERO);
    end;
  close(ARQUIVO_INT);
end.
```

O programa acima é basicamente o mesmo utilizado para apresentar os dados de um arquivo do tipo texto.

Com os programas anteriores você teve um contato com o trabalho executado com arquivos de tipo definido. Foram apresentados apenas arquivos do tipo inteiro, uma vez que para os demais tipos de arquivo as operações serão idênticas. Com relação ao tipo **record**, será tratado no próximo capítulo.

16.4 - UTILIZAÇÃO DE ARQUIVOS COM TIPO DEFINIDO

A seguir, serão apresentadas duas rotinas que exemplificarão o uso de arquivos com matrizes de uma dimensão. A primeira rotina deve solicitar a entrada de dez valores inteiros e armazenar os valores em uma matriz para transferi-los a um arquivo. Depois deve ser escrita uma segunda rotina que leia os dados do arquivo, transferindo-os para uma matriz, para então apresentá-los.

Digite o programa seguinte e grave-o com o nome MATINT01. Observe que será criado um arquivo chamado MATINT.DBP que conterá dez valores inteiros. Perceba que assim que os elementos são fornecidos para a matriz A, em seguida eles são gravados no arquivo com a instrução **write(***ARQ, A[I]***)**.

```
program MATRIZ_E_ARQUIVO;
uses
  Crt;
var
  A   : array[1..10] of integer;
  I   : integer;
  ARQ : file of integer;
begin
  assign(ARQ, 'MATINT.DBP');
  rewrite(ARQ);
  clrscr;
  I := 1;
  while (I <= 10) do
    begin
      write('Digite o elemento - ', I:2, ': '); readln(A[I]);
      write(ARQ, A[I]);
      I := I + 1;
    end;
  close(ARQ);
end.
```

Digite o programa seguinte e grave-o com o nome MATINT02. Observe que o programa fará a abertura e a leitura do arquivo MATINT.DBP que contém dez valores inteiros. Perceba que pelo fato de saber a quantidade de registros armazenada no arquivo, a leitura está sendo feita com a instrução **while** (*I* <= 10) **do**, e não com a instrução **while not eof(*ARQ*) do**, apesar de também poder ser utilizada.

```
program ARQUIVO_E_MATRIZ;
uses
  Crt;
var
  A   : array[1..10] of integer;
  I   : integer;
  ARQ : file of integer;
begin
  assign(ARQ, 'MATINT.DBP');
  reset(ARQ);
  clrscr;
  I := 1;
  while (I <= 10) do
    begin
      read(ARQ, A[I]);
      write('O registro ', I:2, ' contem o elemento: ');
      writeln(A[I]:2);
      I := I + 1;
    end;
  close(ARQ);
end.
```

16.5 - EXERCÍCIOS

a. Elaborar um programa que efetue a leitura de cinco valores numéricos inteiros em uma matriz de uma dimensão. Após a leitura, os dados devem ser armazenados em um arquivo.

b. Elaborar um programa que efetue a leitura de cinco valores numéricos inteiros do arquivo criado no exercício anterior. Depois de os valores estarem armazenados na matriz, o programa deve apresentar a soma dos elementos de índice ímpar.

c. Elaborar um programa que efetue a leitura de dez valores numéricos inteiros para uma matriz A de uma dimensão, construir uma matriz B de uma dimensão, sendo que cada elemento de B é o quadrado correspondente do elemento de A. Ao final, gravar a matriz B em um arquivo.

d. Elaborar um programa que efetue a leitura de duas matrizes A e B de duas dimensões com vinte elementos (4x5), e construir uma matriz C, a qual será a soma dos elementos das matrizes A e B. A matriz C é da mesma dimensão que as matrizes A e B. Gravar a matriz C em um arquivo.