TOSHINOBU ISHIDA

TURBO PASCAL

Para Programadores BASIC

VERSÃO 3.0

- Conceitos básicos
- Comandos
- Programas-exemplo
- Arquivos em disco

2ª EDIÇÃO

atlas

EDITORA ATLAS S.A.
Rua Conselheiro Nébias, 1384 (Campos Elísios)
Caixa Postal 7186 — Tel.: (011) 221-9144 (PABX)
01203 São Paulo (SP)

TOSHINOBU ISHIDA

TURBO PASCAL

Para Programadores BASIC – Versão 3.0

- Conceitos básicos
- Comandos
- Programas-exemplo
- Arquivos em disco

2ª Edição

SÃO PAULO
EDITORA ATLAS S.A. – 1988

Rua Conselheiro Nébias, 1384 (Campos Elísios)
Caixa Postal 7186 – Tel.: (011) 221-9144 (PABX)
01203 São Paulo (SP)

1. ed. 1987; 2. ed. 1988

ISBN 85-224-0288-4

Impresso no Brasil/**Printed in Brazil**

Depósito legal na Biblioteca Nacional conforme Decreto nº 1.825, de 20 de dezembro de 1907.

Capa
Paulo Ferreira Leite

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

Ishida, Toshinobu, 1960 -
I77t Turbo Pascal : para programadores BASIC, versão 3.0 /
2.ed. Toshinobu Ishida . -- São Paulo : Atlas, 1988.

Bibliografia.
ISBN 85-224-0288-4

1. BASIC (Linguagem de programação para computadores) 2. Turbo Pascal (Programa de computador) I. Título.

87-1642 CDD-001.6425
-001.6424

Índices para catálogo sistemático:

1. BASIC : Linguagem de programação : Computadores : Processamento de dados 001.6424
2. Turbo Pascal : Computadores : Programas : Processamento de dados 001.6425

SUMÁRIO

Prefácio à segunda edição, 7
Prefácio à primeira edição, 9

1 **TURBO PASCAL 3.0,** 11

Introdução, 11
Preliminares, 12

2 **UTILIZAÇÃO DO TURBO PASCAL,** 15

Chamada do Turbo, 15
Comandos de edição, 17

3 **CONCEITOS BÁSICOS,** 20

Variáveis numéricas, 20
Outras variáveis, 20
Operadores aritméticos, 21
Operadores lógicos, 21
Declarações, 21
Expressões de atribuição, 22
Ponto e vírgula, 23
Comentários, 23
Convenção, 24

4 **COMANDOS DE ENTRADA E SAÍDA,** 26

Saída, 26
Entrada, 28

5 **MATRIZES, FUNÇÕES E PROCEDIMENTOS,** 31

Matrizes, 31
Funções aritméticas, 33
Funções e procedimentos de manipulação de *strings,* 33
Funções de transformação, 35
Outras funções, 35

6 **COMANDO FOR,** 37

For, 37
For e *for,* 39
Uma pequena pausa, 40

7 **COMANDO IF,** 42

IF, 42
Tabela verdade, 46

8 **UMA PEQUENA PAUSA,** 49

Refrescando, 49
Compatibilidade, 49
Potenciação, 49
Matrizes inteiras, 50
Interrupção, 51
Caracteres de controle, 51
Inicialização de variáveis, 51
Comando Q, 52

9 **COMANDO REPEAT E WHILE,** 53

Repeat, 53
While, 55
Resumo, 56

10 **COMANDOS CASE E GOTO,** 57

Case, 57
GOTO, 60

11 **PROCEDIMENTOS,** 62

O que é procedimento, 62
Variáveis locais, 64
Variáveis globais, 64
Parâmetros, 65

12 **FUNÇÕES,** 71

O que são funções, 71
Observação, 73

13 **COMPLEMENTO SOBRE PROCEDIMENTOS,** 75

Tela, 75
Outras utilidades, 75
Observação, 77

14 **DECLARAÇÃO DE TIPOS,** 78

Introdução, 78
Scalar, 78
Subrange, 81
Observação, 82
Conjunto, 82

15 **ARQUIVOS EM DISCO,** 84

Sua utilidade, 84
Tipos de arquivos, 84
Registro, 84
Buffer, 85
Declarações, 85
Comandos, 87

16 **COMPILAÇÃO DE PROGRAMAS,** 92

Compilação, 92
Opções de compilação, 93
Modo COM, 93
Modo CHN, 94
Cuidados especiais, 94
Diretivas de compilação, 94
Para todos os sistemas, 94
Somente para CP/M-80, 95
Somente para MS/DOS e CP/M-86, 95

17 **EDIÇÃO DE DADOS,** 96

Recursos de edição, 96
TurboBCD, 98
Campos alfanuméricos, 100

18 **RECURSIVIDADE,** 101

O que é recursividade, 101
Observações, 104

19 **ARQUIVOS DE TEXTO,**

O que são arquivos de texto, 105
Utilidade, 105
Declarações, 105
Comandos, 106

20 **IMPRESSORAS,** 108

Compatibilidade, 108
Como enviar os comandos, 108
Tabela de comandos, 105

Apêndice 1 – **Mensagens de erro,** 111
Apêndice 2 – **Conversões,** 118
Apêndice 3 – **Tabelas ASCII do PC,** 119

Bibliografia, 124

PREFÁCIO À SEGUNDA EDIÇÃO

A proposta inicial deste livro parece ter sido muito bem aceita, surpreendendo o autor o esgotamento da primeira edição, pois, ao ser lançado, o Turbo Pascal não tinha um espectro de usuários muito grande. Hoje, tornou-se uma linguagem de aprendizado obrigatório nos cursos de graduação da Universidade de São Paulo, que dispõe de aproximadamente uma centena de microcomputadores do tipo PC, para serem utilizados em elaboração de exercícios-programas em Turbo Pascal.

Atendendo a algumas sugestões dos leitores, adicionou-se um capítulo referente à utilização de arquivos-textos, que podem ser muito úteis para processamento em *batch* e passagens de informações entre programas e um capítulo dedicado a uma melhor utilização dos *efeitos especiais* das impressoras matriciais.

Para atender a razoável parcela de leitores que utilizam o Turbo Pascal em aplicações científicas, foi colocada uma tabela de equivalências matemáticas e trigonométricas no apêndice. E para complementar, o próprio autor sentiu a falta de uma tabela ASCII com caracteres especiais do PC, ao utilizar o livro nas suas aulas.

Este mês completa o quarto aniversário da estréia do autor em algum tipo de publicação. Nesta oportunidade, gostaria de agradecer aos amigos (amizade não se perde no tempo) que participavam da revista *Microhobby* e que deram muito apoio a mim e ao Tomaz para que fizéssemos um artigo para a revista. São eles: Álvaro A. L. Domingues, Ana Lúcia de Alcântara, Caio Marques Bulhões, Marcos Lorenzi e Márcia Regina Dominiquini. Vocês lembram que para carregar o programa "Bolonha e Milano" se demorava quase 20 minutos no TK?

Setembro/1988
Toshinobu Ishida

PREFÁCIO À PRIMEIRA EDIÇÃO

O meu primeiro contato com uma linguagem de programação foi através do curso Introdução à Ciência da Computação na Escola Politécnica da USP. A linguagem utilizada para o ensino era LEAL (parecida com uma tradução para português do ALGOL) e aprendíamos a traduzir para FORTRAN a fim de que pudéssemos processar os programas num Burroughs B-6700.

Como FORTRAN não era uma linguagem dirigida à programação estruturada, utilizava-se de artifícios para deixá-lo com sintaxe acadêmica.

Com o surgimento e a popularização dos microcomputadores, a linguagem mais utilizada passou a ser BASIC, que era oferecida juntamente com o equipamento. A facilidade de aprendizado inicial, aliada a poderosos comandos, fizeram com que esta linguagem fosse utilizada do Pocket-Computer até Sistemas Multiusuários, sem grandes problemas.

Assim, nasceu uma geração de programadores profissionais e *hobbystas* capazes de fazer programas complexos na área comercial ou tecnológica sem nunca ter aprendido COBOL, que era considerada uma linguagem de aprendizagem obrigatória para programadores profissionais.

À medida que os anos passaram, as linguagens modernas, como Pascal, C e dBASE, aposentaram as linguagens mais antigas, pelo menos a nível de microcomputadores. Isto significa que a cada dia que passa essas linguagens são aperfeiçoadas de tal forma que as linguagens modernas ficaram mais rápidas, poderosas e fartas em biblioteca ou ferramentas de trabalho.

Este livro visa levar o leitor ao universo do Pascal através de uma das suas mais famosas implementações, que é o Turbo Pascal. Isto, no entanto, foi feito não para principiantes em programação, mas para aqueles que já têm alguma experiência em programar utilizando a linguagem BASIC.

Para quem conhece um pouco de programação torna-se cansativo aprender novamente como funciona um computador, os comandos de atribuição, entrada e saída de dados, controle de repetição, desvios condicionais etc. Portanto, neste livro faz-se um desenvolvimento do Turbo Pascal baseado em conceitos que um programador BASIC já teria formado através de um aprendizado anterior.

Para poder acompanhar a programação estruturada do Pascal, os exemplos em BASIC podem parecer um pouco estranhos à primeira vista, principalmente na utilização do IF..THEN..ELSE, mas uma leitura mais atenta indicará que não compromete a compreensão dos programas.

No capítulo referente a arquivos em disco foi utilizado um abuso de nomenclatura (principalmente no item *buffer*), mas isto foi proposital, para facilitar o aprendizado dentro do espírito proposto neste livro.

Espero que a passagem do BASIC para Pascal abra novos horizontes no seu eterno aprendizado, tal como tem sido para mim. Alerto que este livro é apenas introdutório no assunto. O Turbo Pascal possui tantos recursos sofisticados que um livro como este não consegue cobrir nem um terço.

Agradeço a todos que me apoiaram direta ou indiretamente, com destaque para Tomaz Tauscher (IT Systems Informática), João Lúcio Neto (ADP Systems) e Lívia Tomoko Haiachiguti.

Gostaria de registrar um agradecimento ao Eduardo Takeo Uehara pelo apoio que recebi em diversas oportunidades da minha vida profissional.

TOSHINOBU ISHIDA

1

TURBO PASCAL 3.0

INTRODUÇÃO

O Pascal é uma linguagem que nasceu a partir de um projeto do professor Niklaus Wirth, da Escola Politécnica de Zurique. A descrição formal desta linguagem foi publicada em 1971.

O Pascal nasceu dentro do conceito de programação estruturada, muito discutido na Europa durante a década de 60.

No Brasil, a Universidade de São Paulo adota-o atualmente como a linguagem de programação para os seus alunos da área de tecnologia, como engenharia, matemática, computação e física.

Existem no mercado diversas implementações da linguagem Pascal para microcomputadores. Eis aqui as três versões mais difundidas.

- **UCSD** - Versão desenvolvida com a colaboração da Universidade da Califórnia, Campus de San Diego.
- **MS-Pascal** - Versão desenvolvida pela Software House Microsoft, especialista em desenvolvimento de compiladores e interpretadores de linguagens.
- **Turbo Pascal** - Versão produzida pela Borland International, que, sem dúvida, é a mais difundida e tornou-se padrão para utilização em microcomputadores.

Neste livro, em virtude de três razões, será desenvolvida a última implementação exposta, o Turbo Pascal 3.0:

- velocidade;
- facilidade;
- recursos.

Vamos fazer uma ligeira análise das três vantagens do Turbo Pascal:

1. Velocidade

O Turbo Pascal é extremamente rápido no seu processo de compilação e, pelo menos, 20 vezes mais veloz do que seus concorrentes, que necessitam de diversos passos para compilar.

A velocidade de execução é razoavelmente equilibrada em relação aos outros compiladores, melhorando consideravelmente a sua *performance* se houver a possibilidade de utilizar a versão que opera com o co-processador aritmético 8087.

2. Facilidade

Durante os testes de funcionamento dos programas, um compilador normal obriga-nos a chamar o editor de texto e a corrigir o programa sempre que for detectado o erro. O Turbo Pascal já possui editor de textos próprio (semelhante, mas muito mais rápido do que WordStar) que, à simples constatação de erro durante a compilação, é reativado e aponta com o cursor o local em que ocorreu o erro.

3. Recursos

Turbo Pascal permite a existência simultânea na memória do programa objeto, fonte, compilador e editor de textos. Além disso, devido à política de marketing da Borland, há diversos pacotes de sub-rotinas prontas para executar funções de banco de dados, *sort* (ordenação de arquivos), gráficos em alta resolução e editores de texto. Este suporte da Borland, aliado à disponibilidade do compilador para diversos sistemas operacionais, como MS-DOS (PC-DOS), CP/M-80, CP/M-86 e, recentemente, para equipamento Macintosh da Apple Computers, fez do Turbo Pascal o padrão para microcomputadores.

PRELIMINARES

Antes de começar a usar o seu disco com Turbo Pascal, não se esqueça de tirar uma cópia de segurança (*back-up*). Criar um *back-up* é um procedimento muito importante, mas freqüentemente esquecido até entre os que se consideram experientes em trabalhar com microcomputadores.

Formate um disco, gere o sistema operacional e copie os arquivos necessários para trabalhar, com Turbo Pascal, o disco recém-formatado.

Se o seu microcomputador utilizar sistema operacional compatível com MS-DOS e possuir dois *drives*, siga este roteiro:

1. Coloque o disco-mestre do sistema operacional no *drive* A.
2. Coloque, no *drive* B, o disco a ser futuramente utilizado para armazenar o seu *back-up* do Turbo Pascal.
3. Digite FORMAT B: /S.
4. Após o término da formatação, coloque, no *drive* A, o disco a ser copiado.
5. Digite COPY A:TURBO*.* B:

Assim, o disco preparado para o seu uso deve conter pelo menos os arquivos:

```
TURBO.COM
TURBO.MSG
```

Se o leitor estiver querendo utilizar a versão do Turbo Pascal que utiliza o co-processador 8087, deve possuir TURBO-87.COM. Se desejar operar com BCD (*Binary Coded Decimals*), que apresenta maior precisão numérica, deve utilizar a versão TURBOBCD.COM.

As principais características das três versões do Turbo Pascal são as seguintes:

TURBO - Os números reais são representados através de 11 algarismos no máximo. Esta versão é padrão e pode ser utilizada em qualquer equipamento para o qual está disponível.

TURBO-87 - Esta versão roda somente nos equipamentos de 16 *bits* que possuam o *chip* 8087, ou compatível, e permite a representação real em 16 dígitos numéricos. É extremamente rápido para trabalhar com valores numéricos.

TURBOBCD - Utilizando 10 *bytes* para representação dos números reais, esta versão permite o uso de 18 algarismos para representação real. Tem a vantagem de praticamente não haver imprecisão e possuir recursos adicionais de formatação para a impressão de dados, sendo extremamente útil em aplicações comerciais. Não executa certas funções matemáticas e trigonométricas e somente é disponível para equipamentos de 16 *bits*.

Supondo um teste de desempenho num equipamento de 16 *bits* com 8087, a versão TURBO-87 seria a mais veloz de todas, seguida pelo TURBO e depois pelo TURBOBCD que não reconhecem a existência do co-processador aritmético.

Qualquer referência feita à linguagem BASIC neste texto será baseada na sintaxe padronizada pelo BASIC da Microsoft, que é, sem dúvida, a versão mais popular do mundo. Microsoft BASIC é conhecido comercial-

mente como MBASIC, GBASIC, GW-BASIC, BASICA e outros, equipando microcomputadores da família IBM-PC/XT/AT, TRS-80, CP/M, MSX-1/2, Macintosh e muitos outros que saem da fábrica como linguagem residente.

Turbo Pascal é um compilador da tradicional linguagem Pascal, mas dirigido a microcomputadores. O Pascal possui pequeno número de comandos e não é suficientemente completo para tirar todas as vantagens que um microcomputador pode oferecer. Para cobrir esta deficiência, o Turbo Pascal implementa diversos novos comandos, sem, no entanto, afastar-se demasiadamente das definições e do espírito que o professor Wirth utilizou na sua criação.

Ao se comparar o Microsoft BASIC e o Turbo Pascal, vemos que muitos comandos que o BASIC possui podem ser criados em Pascal com a utilização de procedimentos e funções. É aqui que o Pascal leva enorme vantagem. O Turbo Pascal não possui nenhuma mensagem de erro equivalente a *Syntax error* do Microsoft BASIC porque, definindo-se funções e procedimentos, podemos enriquecer o seu vocabulário. A mensagem de erro mais próxima do BASIC que o Turbo Pascal acusa é *Unknown identifier or syntax error*, que significa, simplificadamente, que esta palavra não foi definida anteriormente ou é erro de sintaxe.

Estes recursos do Pascal permitem que o seu usuário enriqueça a linguagem através da ampliação do acervo da sua biblioteca particular de rotinas. Isto não seria impossível fazer em BASIC, mas tornaria extremamente trabalhoso devido à necessidade de numeração de linhas e desvios do tipo GOSUB e RETURN. Para pequenos programas, as duas linguagens se equivalem quando somados os seus pontos positivos e negativos. Para programas extensos e complexos, sem dúvida nenhuma o Pascal é a melhor solução. Muito melhor ainda se for Turbo Pascal.

A Borland oferece, para acompanhar o Turbo Pascal, muitos procedimentos e programas prontos para uso, bastando apenas incluí-los na compilação ou, simplesmente, compilá-los. Entre eles, os dois mais interessantes são:

DATABASE TOOLBOX - É uma biblioteca que contém procedimentos que gerenciam os arquivos através de uma técnica chamada *B+Tree* e um *sort* extremamente eficiente, que utiliza o algoritmo *Quicksort*.

GRAPHIX TOOLBOX - Permite o pronto uso de gráficos de alta resolução, janelas, gravação de telas, transferência de telas para outras partes da memória e movimentos espetaculares de janelas pela tela.

Além dos dois pacotes acima, ainda encontramos:

GAMEWORKS - Jogos tradicionais de salão, como xadrez, *bridge* e *gomoku*, em Pascal, para estudar, alterar, compilar e jogar.

EDITOR TOOLBOX - Contém procedimentos que permitem construir o seu próprio editor de textos.

2

UTILIZAÇÃO DO TURBO PASCAL

CHAMADA DO TURBO

Ao carregar o seu microcomputador com o Turbo Pascal, aparece na tela a seguinte mensagem:

```
TURBO Pascal system                    Version 3.01A
                                                MS-DOS

Copyright (C) 1983,84,85          BORLAND Inc.
-------------------------------------------------------

Terminal: Toshiba T-3100 AT


Include error messages (Y/N)?
```

O cursor permanece piscando até que se digite *Y* ou *N* para responder à mensagem. A resposta normalmente digitada deve ser *Y*. Caso haja necessidade de poupar memória, pode-se responder *N*. Em geral, digita-se *N* quando:

- O texto a ser editado é muito extenso.
- A memória disponível é muito pequena.
- Já se acostumou às mensagens mais corriqueiras.

Mesmo que responda *N*, o Turbo Pascal continuará a emitir mensagens de erro em forma de código numérico.

A próxima tela é a seguinte:

```
Logged drive: A
Active directory: \

Work file:
Main file:

Edit        Compile   Run     Save

Dir         Quit      compiler Options

Text:          0 bytes
Free:  62104 bytes
```

O significado do *menu* principal é dado a seguir:

Logged drive - indica a letra do *drive* de uso corrente.

Active directory - mostra o nome do diretório corrente.

Work file - indica o nome do arquivo (texto) que está sendo editado.

Main file - indica o nome do arquivo principal a ser editado. Sua utilização será explicada adiante, juntamente com a inclusão de arquivos externos durante a compilação (diretiva de compilação $I).

Edit - coloca em ação o editor de texto para um arquivo definido no *Work file.*

Compile - ativa a compilação do texto editado.

Run - executa o programa compilado em memória. Se o programa ainda não estiver compilado, executará primeiro a compilação.

Save - grava o texto atualmente em memória.

Dir - fornece o diretório do *drive* de uso corrente. Podem-se colocar máscaras (b:, c:, * e ?) para obter um diretório filtrado. Fornece, também, o número de *bytes* remanescentes no disco.

Quit - encerra a utilização do Turbo Pascal e retorna ao sistema operacional.

Compiler Options - entra num *menu* de opção de compilação. Este assunto será abordado posteriormente. Por enquanto só interessa a compilação em memória.

Para ativar esses comandos basta pressionar as letras maiúsculas correspondentes a cada comando. Se o seu equipamento reconhecer a va-

riação de tonalidade no monitor de vídeo, as letras estarão com brilho mais intenso ou mais fraco em relação às demais.

Para iniciar a edição de um texto (programa) digite a tecla *W*. Aparecerá a seguir:

```
Work file name:
```

Digite o nome do programa. Se a terminação for omitida, o Turbo Pascal assumirá ".PAS" para este arquivo. Se o arquivo não existir no disco, a mensagem apresentada será a seguinte:

```
Loading A:\TESTE.PAS
New File
```

Caso exista, este será imediatamente carregado para a memória, aparecendo apenas a primeira linha acima.

Para iniciar a edição basta pressionar *E*.

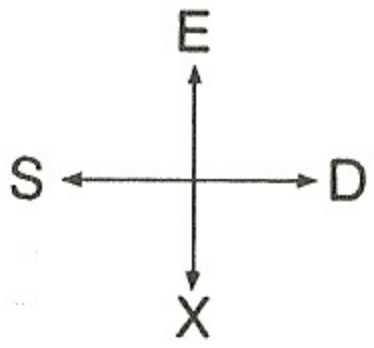
COMANDOS DE EDIÇÃO

O editor de texto do Turbo Pascal é muito parecido com o famoso WordStar da MicroPro. Os comandos de edição são os mesmos e até mais rápidos e aperfeiçoados em relação ao WordStar. A única limitação é que o texto não pode exceder a área livre da memória.

Para a finalidade deste livro são suficientes, por enquanto, os seguintes comandos:

COMANDO	AÇÃO
CTRL - E	Movimenta o cursor uma linha para cima.
CTRL - S	Movimenta o cursor uma coluna para a esquerda.
CTRL - X	Movimenta o cursor uma linha para baixo.
CTRL - D	Movimenta o cursor uma coluna para a direita.

Observe que as teclas acima formam uma figura geométrica que lembra os movimentos nas suas respectivas direções.



COMANDO	AÇÃO
CTRL - R	Movimenta o cursor uma página acima.
CTRL - C	Movimenta o cursor uma página abaixo.
CTRL - V	Liga ou desliga o modo de inserção.
CTRL - N	Insere uma linha.
CTRL - Y	Deleta uma linha.
CTRL - G	Deleta um caractere.
CTRL - QR	Mostra o início do texto.
CTRL - QC	Mostra o fim do texto.
CTRL - KD	Temina o modo de edição de texto.

Para os equipamentos de 16 *bits* há ainda o recurso de utilizar as funções das teclas numéricas em vez de seqüências com Control. Teste-os para verificar os comandos equivalentes, pois são passíveis de alterações através da instalação do sistema (TINST). Maiores informações são obtidas no Manual de Referência, Apêndice *L* sobre instalação, que utiliza o programa TINST.COM.

O restante dos comandos de edição será apresentado no decorrer do texto ou deverá ser visto no Manual de Referência do Turbo Pascal ou WordStar.

APENAS UM TESTE

Estando no *menu* principal, façamos um pequeno teste:

1. Digite um pequeno texto e veja como funciona. Para isto é preciso dar o nome ao texto. Digite *W* e logo a seguir TESTE.
2. Após a mensagem *New file*, digite *E* para entrar no modo de edição.
3. A tela vai ser limpa, sobrando apenas os indicadores de linhas, colunas, inserção e nome do arquivo texto.
4. Digite o nome e dê CTRL -KD. Você verá que o *menu* principal voltou.
5. Isto não significa que tudo foi perdido. Para ter o texto de volta basta digitar *E*, que é o comando de edição de texto.
6. Estando no *menu* principal, digite *S* e o texto será gravado em disco. Desta maneira, poderá ser recarregado do disco quando for novamente necessário.

Somente para certificar que o texto que contém o seu nome ficou gravado, podemos ativar o comando *D* e observar o diretório. Aí deverá estar o arquivo TESTE.PAS.

Se desejar encerrar os testes, digite *Q* (no *menu* principal) para voltar ao sistema operacional.

Neste instante, aconselho aos leitores que não estão ainda familiarizados com os comandos de edição do Turbo Pascal ou do WordStar prosseguir os testes, experimentando todos eles. Afinal, nada é melhor do que aprender praticando no próprio equipamento.

3

CONCEITOS BÁSICOS

VARIÁVEIS NUMÉRICAS

O Turbo Pascal trabalha com números que são do tipo:

- **Integer :** números inteiros entre -32768 e 32767.
- **Real :** números reais de até 11 algarismos na mantissa (exceto nas versões TURBOBCD e TURBO-87).
- **Byte :** números inteiros entre 0 e 255.

Comparando com o BASIC, há a coincidência dos valores dos inteiros, mas o real é intermediário entre simples e dupla precisão. No BASIC não existe números do tipo *byte.*

A precisão oferecida pelos números reais é mais do que suficiente para aplicações normais, mas, caso haja necessidade de maior precisão, pode-se usar a versão TURBOBCD, que suporta até 18 algarismos.

Para processar os três tipos de números dados, o Turbo Pascal possui, respectivamente, variáveis numéricas do tipo *integer*, *real* e *byte*.

OUTRAS VARIÁVEIS

O Turbo Pascal possui outros tipos de variáveis que são:

- **String** : é igual ao *string* do BASIC.
- **Char** : é um *string* de apenas um caractere.
- **Boolean** : assume valores *booleanos* que são True (verdadeiro) e False (falso).

Os seis tipos de dados mencionados são chamados *predefined data types,* ou seja, para usá-los não há necessidade de defini-los.

Este conceito de definir o tipo de dados não existe no BASIC, mas é extremamente importante no Turbo Pascal. Analisaremos este conceito adiante, à medida que formos avançando no nosso estudo.

OPERADORES ARITMÉTICOS

As quatro operações básicas utilizam os mesmos caracteres do BASIC

+ adição.
- subtração.
* multiplicação.
/ divisão.

OPERADORES LÓGICOS

São iguais ao do BASIC

<	menor que
>	maior que
<=	menor ou igual que
>=	maior ou igual que
=	igual a
< >	diferente de
and	e
or	ou
not	negação
xor	ou exclusivo

DECLARAÇÕES

Declarar as variáveis é um ato a que os programadores em BASIC não estão acostumados. Na verdade, a não-necessidade de declarar as variáveis é o ponto forte do BASIC, pois permite construções simplificadas de programas. Mas, para se aventurar em campos de grandes, complexos e legíveis programas, não podemos fugir destas declarações. No Turbo Pascal, a declaração destas variáveis é obrigatória, sem o que o compilador irá acusar erro de compilação.

No BASIC existem comandos bem parecidos que declaram o tipo de variáveis, como DEFINT, DEFSNG, DEFDBL e DEFSTR.

Suponha que precise usar a variável inteira para determinado programa. A declaração deve possuir o seguinte aspecto:

```
var
    I : integer;
```

Desejando usar a variável inteira I, real J e *booleana* S, fica assim:

```
var
    I : integer;
    J : real;
    S : boolean;
```

Para declarar mais de uma variável do mesmo tipo pode-se proceder das seguintes maneiras:

```
var
    I,X,Y : integer;
```

```
var
    I : integer;
    X : integer;
    Y : integer;
```

As variáveis *strings* no Turbo Pascal devem ter a sua máxima extensão declarada. Esta declaração reserva uma área fixa na memória para o processamento. A inexistência deste conceito no BASIC faz com que ele seja extremamente lento no processamento alfanumérico, sendo interrompido de vez em quando pela rotina de Garbage Collection. Esta rotina é a famosa paradinha que o BASIC dá, quando nem o *break* consegue interrompê-lo.

A declaração de um *string* S para 16 caracteres seria:

```
var
    S : string[16];
```

Do ponto de vista sintático, a declaração acima não está errada. Mas há um modo mais elegante e conciso que iremos abordar adiante.

EXPRESSÕES DE ATRIBUIÇÃO

Veja a seguinte expressão em BASIC:

```
A=B+C
```

Isto significa que a expressão B + C vai ser atribuída à variável *A*, ou seja, não deve ser lida como *A* igual a B + C. Este abuso de notação em BASIC é permitido, pois originalmente todas as expressões de atribuição eram precedidas pelo comando LET. Mas com o desenvolvimento de no-

vas versões, o comando LET do BASIC acabou tornando-se opcional e, portanto, caiu em desuso.

No Turbo Pascal não existe comando LET nem o abuso de notação. A mesma expressão seria representada assim:

```
A:=B+C;                    (A <= B+C)
```

O símbolo := equivale a uma flecha, indicando que é uma atribuição.

PONTO-E-VÍRGULA

O ponto-e-vírgula no Turbo Pascal indica o fim do comando. A sua função é parecida com a dos dois-pontos (:) do BASIC, mas a sua presença é obrigatória. Aprender a usar corretamente o ponto-e-vírgula é o primeiro desafio para passar do BASIC para o Pascal.

Uma regra importante:

Todos os programas em Turbo Pascal devem começar e terminar desse modo.

```
program Teste;
begin
        .
        .
        .
end.
```

O *program* indica o início do programa; *teste* é um nome qualquer dado ao programa; *begin* indica o início; e *end.* o fim do processamento.

COMENTÁRIOS

Os comentários em BASIC são precedidos por REM ou apóstrofo ('). No Turbo Pascal, os comentários devem estar entre { ... } ou (*...*). Não há necessidade de fechar o comentário com ponto-e-vírgula.

```
{ Isto e um comentario }

(* Isto tambem e um comentario *)
```

Vejamos um pequeno programa em que existam apenas comentários em BASIC:

```
BASIC

1 REM PROGRAMA TESTE1
2 REM COMO UTILIZAR COMENTARIOS
3 REM PODE-SE USAR ESTE COMANDO OU
4 ' ESTE PARA INDICAR O COMENTARIO
5 END
```

Em Turbo Pascal teríamos esta apresentação:

```
Turbo Pascal

begin
    { Como utilizar comentarios }
    { Pode-se usar este comando ou }
    (* este para indicar o comentario *)
end.
```

Em BASIC, cada linha deve ser indicada caso seja apenas comentário. Em Turbo Pascal foi apresentado que o comentário deve estar entre os caracteres utilizados acima. Poderia ser então:

```
Turbo Pascal


program Teste1;
begin
    (* Como utilizar comentarios.
       Pode-se usar este comando ou
       este para indicar o comentario *)
end.
```

O Turbo Pascal não depende da numeração das linhas. Esse fator permite inserir linhas a mais ou deslocar colunas livremente dentro de um programa, sem afetar a lógica de programação. Este recurso é excelente para aumentar a legibilidade dos programas.

CONVENÇÃO

No BASIC, mesmo que se digite um comando em letras minúsculas, elas tornam-se maiúsculas na listagem da tela ou impressora. No Turbo

Pascal, o programa permanece como foi digitado pelo editor de textos. A leitura de programas escritos em letras minúsculas é muito mais confortável do que com as letras maiúsculas, principalmente na tela do computador. Por isso vamos estabelecer algumas convenções utilizadas neste texto:

1. Os comandos pertencentes ao Turbo Pascal serão escritos em letras minúsculas.
2. Qualquer palavra criada pelo usuário terá a primeira letra maiúscula e o restante em minúscula.

As duas regras acima fazem com que os comandos *begin*, *end*, *program* etc. sejam escritos desta maneira, enquanto variáveis, funções, procedimentos etc. sejam escritos com a letra inicial em maiúscula. Por ora, ainda não conhecemos as funções e os procedimentos.

4

COMANDOS DE ENTRADA E SAÍDA

SAÍDA

Vamos abordar simplificadamente os comandos de saída de dados equivalentes ao PRINT e LPRINT do BASIC.

O Turbo Pascal admite duas formas de impressão de dados. As equivalências em BASIC são dadas a seguir:

BASIC	Turbo Pascal
PRINT	writeln
PRINT;	write

O comando *writeln* envia o sinal de *line feed* (mudança de linha), enquanto o *write* manda apenas imprimir.

No Turbo Pascal, aquilo que se deseja imprimir deve vir entre parênteses. Suponha uma variável denominada KONT:

BASIC	Turbo Pascal
PRINT KONT	writeln(Kont);

Caso se deseje imprimir um texto, este deve vir entre apóstrofos ('), enquanto no BASIC são utilizadas aspas (").

BASIC	Turbo Pascal
PRINT"Confirma ?"	writeln('Confirma ?');

Vamos exemplificar com um pequeno programa:

```
BASIC

1 ' Programa Teste2
2 '
3   DEFINT A,L,P
4   A=10
5   L=2
6   P=2
7   PRINT "Resposta :  " A*L*P " cm3"
8   END
```

Suponha que exista uma caixa retangular cuja altura seja 10 cm, largura e profundidade 2 cm. O volume ocupado pela caixa é de 10 x 2 x 2 = 40 cm^3. Este cálculo é efetuado pelo programa BASIC acima. Convertido para Turbo Pascal, ficaria assim:

```
Turbo Pascal

program Teste2;

var
    A,L,P : integer;

begin
    A:=10;
    L:=2;
    P:=2;
    writeln('Resposta : ',A*L*P,' cm3');
end.
```

O Turbo Pascal não possui um comando diretamente equivalente ao LPRINT do BASIC. Os comandos que acionam a impressora são os mesmos do vídeo, ou seja, *write* e *writeln*, acrescidos de um parâmetro extra.

BASIC	Turbo Pascal
LPRINT	writeln(Lst)
LPRINT;	write(Lst)

Usando o mesmo exemplo anterior, ficariam assim as linhas que executam a impressão:

```
BASIC

7  LPRINT "Resposta : " A*L*P " cm3"


Turbo Pascal

writeln(Lst,'Resposta : ',A*L*P,' cm3');
```

Saliente-se que o *line feed* é dado após a impressão de dados. Isto significa que teríamos os mesmos formatos de impressão se a linha acima fosse desmembrada assim:

```
Turbo Pascal

write(Lst,'Resposta : ');
write(Lst,A*L*P);
writeln(Lst,' cm3');
```

ENTRADA

O comando INPUT do BASIC tem como seu equivalente em Turbo Pascal o *readln* e o *read*.

BASIC	Turbo Pascal
INPUT	readln
	read

Como no comando *write* e *writeln*, as duas variações do comando de leitura diferenciam-se pela inclusão do *line feed* após a operação.

As variáveis que irão receber os valores devem estar dentro de parênteses.

BASIC	Turbo Pascal
INPUT IDADE	readln(Idade);

Atenção neste detalhe. Ao se pedir mais de uma entrada para *read* ou *readln*, os dados a serem digitados não devem ser separados por vírgulas (,), mas por espaço.

```
Turbo Pascal

readln(Idade, Altura, Peso);
```

Quando se deseja introduzir dados do tipo *string*, fica mais complicado. Como Turbo Pascal é uma linguagem que prima pela facilidade de leitura, é bom hábito pedir apenas uma entrada de dados para cada comando *read* ou *readln*.

O programa Teste2 em BASIC que utiliza entrada de dados ficaria assim:

```
BASIC

1' Programa Teste2
2'
3   DEFINT A,L,P
4   INPUT A
5   INPUT L
6   INPUT P
7   PRINT "Resposta :  " A*L*P " cm3"
8   END

Turbo Pascal

program Teste2;

var
    A,L,P : integer;

begin
    readln(A);
    readln(L);
    readln(P);
    writeln('Resposta : ',A*L*P,' cm3');
end.
```

No Turbo Pascal não existe um comando equivalente ao INPUT "Mensagem" do BASIC. Para contornar este problema, o comando deve ser desmembrado em dois. Vejamos um exemplo:

BASIC

```
1' Programa Teste2
2'
3   DEFINT A,L,P
4   INPUT "Altura : ";A
5   INPUT "Largura : ";L
6   INPUT " Profundidade : ";P
7   PRINT "Resposta : " A*L*P " cm3"
8   END
```

Turbo Pascal

```
program Teste2;

var
    A,L,P : integer;

begin
    write('Altura : ');
    readln(A);
    write('Largura : ');
    readln(L);
    write('Profundidade : ');
    readln(P);
    writeln('Resposta : ',A*L*P,' cm3');
end.
```

Que tal digitar o programa acima e exercitar um pouco para reforçar os conhecimentos sobre a utilização do editor de texto e comandos de compilação ?

Os comandos *write* e *writeln* possuem diversos parâmetros que permitem editar (formatar) os dados a serem apresentados.

Esta capacidade não é tão poderosa como o comando PRINT USING do BASIC, mas já ajuda bastante na apresentação de dados, principalmente em programas científicos e comerciais. Este assunto será desenvolvido posteriormente, salientando-se que o TURBOBCD possui comandos extras de formatação tão ou mais poderosos do que o BASIC.

Para encerrar este capítulo, o comando *writeln*, sem variáveis a serem impressas, equivale a um comando PRINT nas mesmas condições em BASIC. Envia apenas o *line feed* para a tela ou impressora (writeln(Lst)).

5

MATRIZES, FUNÇÕES E PROCEDIMENTOS

MATRIZES

Uma matriz no Turbo Pascal funciona da mesma forma que no BASIC. Apenas a declaração é obrigatória e um pouco mais completa.

Vamos supor uma matriz chamada MAT de uma única dimensão (vetor) composta de 50 números inteiros. Em BASIC, a declaração é conhecida como dimensionamento e é feita assim:

```
BASIC

DIM MAT%(50)
```

Em Turbo Pascal, a apresentação é mais completa:

```
Turbo Pascal

var
   Mat : array [1..50] of integer;
```

No caso de uma matriz bidimensional de 50 linhas e 100 colunas composta de *string* de dez caracteres, teríamos:

```
BASIC

DIM MAT$(50,100)

Turbo Pascal

var
   Mat : array [1..50,1..100] of string[10];
```

Os elementos que formam as matrizes são conhecidos como variáveis indexadas e a sua utilização é semelhante ao BASIC.

BASIC

```
1' Programa Teste3
2'
3   DIM MAT%(3)
4   MAT%(1)=4
5   MAT%(2)=3
6   MAT%(3)=2
7   TOTAL%=MAT%(1)+MAT%(2)+MAT%(3)
8   PRINT TOTAL%
9   END
```

O programa acima soma os elementos da matriz inteira MAT colocando o resultado na variável inteira TOTAL para imprimi-lo. Em Turbo Pascal, o programa ficaria assim:

Turbo Pascal

```
program Teste3;

var
    Mat : array [1..3] of integer;
    Total : integer;

begin
    Mat[1]:=4;
    Mat[2]:=3;
    Mat[3]:=2;
    Total:=Mat[1]+Mat[2]+Mat[3];
    writeln(Total);
end.
```

As matrizes podem ser de todos os tipos até aqui apresentados:

Integer
Real
Char
String
Boolean

FUNÇÕES ARITMÉTICAS

As funções aritméticas do Turbo Pascal são extremamente semelhantes às do BASIC. São elas:

Abs(X): Retorna o valor absoluto do X.
Arctan(X): Retorna o arco tangente de X em radianos.
Cos(X): Retorna o cosseno de X. X em radianos.
Exp(X): Retorna o valor de *e* elevado a X.
Frac(X): Retorna a parte decimal (fracionária) de X.
Int(X): Retorna o valor inteiro de X.
Ln(X): Retorna o logaritmo natural de X.
Sin(X): Retorna o seno de X. X em radianos.
Sqr(X): Retorna o quadrado de X.
Sqrt(X): Retorna a raiz quadrada de X.

FUNÇÕES E PROCEDIMENTOS DE MANIPULAÇÃO DE STRINGS

Concat(X,Y,...): concatena os *strings* X,Y,... Incluiu-se no Turbo Pascal apenas para manter relativa compatibilidade com os outros compiladores Pascal. No Turbo Pascal pode-se usar a soma (+) de *strings*, como é feito em BASIC.

Exemplo:

```
X:='Tele';
Y:='fone';
```

concat(X,Y) retorna Telefone

Copy(X,Y,Z): extrai *Z* caracteres a partir do *Y*-ésimo caractere do *string X*. É semelhante ao comando MID$ do BASIC.

Exemplo:

```
X:='Telefone';
```

copy(X,5,4) retorna fone.

Lenght(X): retorna o comprimento do *string X*. É semelhante à função LEN do BASIC.

Exemplo:

```
X:='Telefone';
```

length(X) retorna 8.

Pos(X,Y): retorna a posição do *string X* dentro do *string Y*. Se não for encontrado, vale zero. É semelhante à função INSTR do BASIC.

Exemplo:

```
X:='C';
Y:='Micro';
```

pos(X,Y) retorna 3.

PROCEDIMENTOS DE MANIPULAÇÃO DE STRINGS

Delete (X,Y,Z): remove *Z* caracteres a partir da posição *Y* do *string X*.

Exemplo:

```
X:='Pancadaria';
delete(X,3,3);
```

X contém Padaria.

Insert(X,Y,Z): insere X no *string Y* a partir da posição Z.

Exemplo:

```
X:='nca';
Y:='Padaria';
insert(X,Y,Z);
```

Y contém Pancadaria.

Str(X,Y): exerce a mesma função do STR$ do BASIC. Converte o valor numérico *X* num *string Y*.

Exemplo:

```
X:=12;
str(X,Y);
```

X contém *string* 12.

Val(X,Y,Z): tem função semelhante ao VAL do BASIC. Converte *string X* em número e atribui a variável *Y*. *Z* deve ser uma variável inteira e indica o sucesso ou não da conversão. Em caso de sucesso, *Z* contém zero; caso contrário, contém a posição do caractere que ocasionou o problema.

Exemplos:

```
X := '1234';
val(X,Y,Z);
```

Y vale 1234 e *Z* vale zero.

```
X := '1234';
val(X,Y,Z);
```

Y é um valor indefinido e *Z* vale 4.

FUNÇÕES DE TRANSFORMAÇÃO

Chr(X): é semelhante ao CHR$ do BASIC. Retorna o caractere cujo código ASCII é X.

Round(X): provoca arredondamento na primeira casa decimal.

Exemplo:

```
X := 15.5;
```

round(X) retorna 16.

```
X := 15.4;
```

round(X) retorna 15.

Trunc(X): retorna o maior inteiro menor ou igual ao X.

OUTRAS FUNÇÕES

Keypressed: é uma função *booleana* um pouco parecida com INKEY$ do BASIC. Devolve o valor *True* se alguma tecla estiver sendo pressionada e *False*, caso contrário.

Random: retorna um número real randômico entre 0 e 1.

Random(X): retorna um número real randômico entre 0 e *X*.

Upcase(X): se o caractere *X* do tipo *char* for minúsculo, converte-o para maiúsculo. Caso contrário, não há alteração.

As funções e os procedimentos descritos neste capítulo são uma pequena parte dos chamados predefinidos ou predeclarados.

Mais adiante estudaremos como se utilizam as funções e os procedimentos declarados pelo usuário. Esta forma de uso é uma das mais poderosas ferramentas das linguagens estruturadas, como o Turbo Pascal.

6

COMANDO FOR

FOR

Vamos continuar a desenvolver o Turbo Pascal através de um comando cuja característica é bem parecida com o seu homônimo em BASIC. Apesar disso, há pequenas características muito importantes em Turbo Pascal que não devem ser esquecidas.

1. As linhas do programa a serem controladas pelo comando *for* devem vir entre um *begin* e um *end;*
2. A variável de controle pode ser somente do tipo *integer*.
3. Não há possibilidade de especificar o passo (STEP) como no BASIC. O passo é sempre +1 ou -1

BASIC

```
1' Programa Teste4
2'
3  DEFINT I
4  FOR I=1 TO 10
5  PRINT I
6  NEXT I
7  END
```

O programa em BASIC imprime valores numéricos de 1 a 10. Em Turbo Pascal seria assim:

```
Turbo Pascal

program Teste4;

var
    I : integer;

begin
    for I:=1 to 10 do
        begin
            writeln(I);
        end;
    end.
```

Desejando decrementar a variável de controle, teremos a seguinte configuração:

```
BASIC

1 ' Programa Teste4;
2 '
3   DEFINT I
4   FOR I=10 TO 1 STEP -1
5   PRINT I
6   NEXT I
7   END
```

```
Turbo Pascal

program Teste4;

var
    I : integer;

begin
    for I:=10 downto 1 do
        begin
            writeln(I);
        end;
end.
```

Será que o leitor já começou a perceber que a leitura de um programa em Turbo Pascal é mais fácil? Pelos menos parece ser mais limpo e menos carregado do que o BASIC.

FOR E FOR

Desejando-se colocar um *for* dentro de outro *for* para controlar, por exemplo, uma matriz bidimensional, pode-se proceder assim:

BASIC

```
1 ' Programa Teste5
2 '
3  DEFINT I,J,M
4  DIM MAT(10,10)
5  FOR I=1 TO 10
6  FOR J=1 TO 10
7  INPUT MAT(I,J)
8  NEXT J
9  NEXT I
10 END
```

O programa acima carrega todos os elementos da matriz MAT com os dados a serem digitados na linha 7 do programa.

O seu equivalente em Turbo Pascal é:

Turbo Pascal

```
program Teste5;

var
    I,J : integer;
    Mat : array[1..10,1..10] of integer;

begin
    for I:=1 to 10 do
        begin
            for J:=1 to 10 do
                begin
                    readln(Mat[I,J]);
                end;
        end;
end.
```

Quando começa a complicar um pouco é bom conferir se há um *end* para cada *begin* antes de tentar rodar o programa. Em caso de erro, o compilador aponta o local ou a região suspeita, mas nem sempre é muito fácil notar este tipo de erro.

Para evitá-lo podemos deslocar os comandos para a direita e esquerda sempre que abrirmos ou fecharmos um bloco com *begin* e *end*. Este expediente facilita a visualização e, na medida do possível, devemos traçar linhas enquanto o programa estiver no rascunho.

```
begin
¦   for I:=1 to 10 do
¦       begin
¦       ¦   for j:=1 to 10 do
¦       ¦       begin
¦       ¦       ¦   readln(Mat[I,J]);
¦       ¦       end;
¦       end;
end.
```

Procedendo como no exemplo acima, evitam-se pequenos aborrecimentos durante a programação, principalmente se a sua estrutura for complexa.

Os deslocamentos das linhas para a direita e esquerda são geralmente de duas ou três colunas em relação à linha anterior.

UMA PEQUENA PAUSA

O comando repetitivo *for* deve ser utilizado nos locais em que a quantidade de vezes a ser repetida já está determinada antes da sua execução. Isto significa que não devemos fazer atribuições à variável de controle do *for* dentro da própria estrutura. Esta variável pode ser utilizada para ser impressa ou para cálculos, mas jamais deve receber algum valor.

Para controlar a repetição internamente, o Turbo Pascal oferece outros comandos que analisaremos adiante.

Esta regra torna-se necessária, pois algumas versões do BASIC admitem este tipo de técnica de programação sem que apresente erros. Pode haver diferença de execução entre um programa BASIC interpretado e compilado nestas condições, dependendo das versões e marcas.

BASIC

```
1'  Programa Errado1
2'
3   DEFINT I
4   FOR I=1 TO 10
5   I=I+1
6   NEXT I
7   END
```

Turbo Pascal

```
program Errado1;

var
    I : integer;

begin
    for I=1 to 10 do
        begin
            I:=I+1;
        end;
end.
```

Evite programar como está exposto. Se necessitar controlar internamente a repetição, utilize outros comandos.

7

COMANDO IF

IF

O comando de decisão *if* é bastante semelhante ao *if* do BASIC. A única diferença é devida a características de programação estruturada do Turbo Pascal.

Utilizando o *if* normalmente, como é feito pelos programadores BASIC, teremos alguns problemas na hora de converter as idéias e os exemplos.

Apesar de o BASIC não ser uma linguagem dirigida para a estruturação, podemos forçá-lo a ser, desde que respeitadas algumas regras e estilos de programação. Vejamos alguns programas BASIC e Turbo Pascal, mas antes vamos parar de declarar variáveis nos programas BASIC daqui para a frente.

BASIC

```
1' Programa Teste6
2'
3   PRINT "Entre com um valor : ";
4   INPUT VALOR
5   IF VALOR > 10 THEN 6 ELSE 8
6   PRINT "O valor e' maior que 10"
7   GOTO 9
8   PRINT "O valor e' menor que 10"
9   END
```

O leitor deve estar estranhando o programa acima quanto à colocação das conseqüências do *if*, mas em programação estruturada em Turbo Pascal tal fato é natural.

Turbo Pascal

```
program Teste6;

var
    Valor : real;

begin
    write('Entre com um valor : ');
    readln(Valor);
    if Valor > 10 then
        begin
            writeln('O valor e maior que 10');
        end
    else
        begin
            writeln('O valor e menor que 10');
        end;
end.
```

Podemos notar que o primeiro par *begin..end* após o *if* somente é executado em caso afirmativo (*then*) e o segundo em caso negativo (*else*).

É necessário tomar alguns cuidados neste ponto. O comando *end* que precede o *else* não deve levar ponto-e-vírgula (;). O ponto-e-vírgula aqui indicaria que já chegou ao fim da atuação do comando *if* anterior. Portanto, somente o segundo e último *end* deve receber o ponto-e-vírgula.

Mesmo com a inexistência de uma operação, caso a sentença do *if* seja falsa (opção para entrar no *else*), somos obrigados a mencionar se o BASIC estiver dentro da filosofia de programação estruturada.

BASIC

```
1' Programa Teste7
2'
3  PRINT "Entre com um valor negativo : ";
4  INPUT VALOR
5  IF VALOR > 0 THEN 6 ELSE 8
6  PRINT "Voce digitou valor positivo !"
7  GOTO 8
8  END
```

```
Turbo Pascal

program Teste7;

var
   Valor : real;

begin
   write('Entre com um valor negativo');
   readln(Valor);
   if  Valor  >  0  then  writeln('Voce  digitou  valor positivo !');
end.
```

Observe o *end* com ponto-e-vírgula após o *if*.

O programa BASIC retromencionado ainda deve continuar estranho, pois, na verdade, bastaria ser assim:

```
BASIC

1' Programa Teste7;
2'
3  INPUT "Entre com um valor negativo"; VALOR
4 IF VALOR > 0 THEN PRINT "Voce digitou valor positivo"
5  END
```

Este programa funciona tal como o anterior. Mas devemos fazer uma ligeira lavagem cerebral se quisermos ir adiante com a programação estruturada e compreender a necessidade de alterar os vícios de programação, mesmo em BASIC.

O Turbo Pascal também admite abusos do tipo:

```
Turbo Pascal

program Teste7;

var
   Valor : real;

begin
   write('Entre com um valor negativo : ');
   readln(Valor);
   if Valor > 0 then
      begin
         writeln('Voce digitou valor positivo !');
      end;
end.
```

Vamos evitar fazer isso e sempre envolver as linhas que se seguem ao *if* com *begin..end* e, se necessário, *else begin..end*, mesmo que seja executado apenas um comando. Em caso de mais de um comando tornam-se obrigatórios os seus usos.

```
BASIC

1' Programa Teste8
2'
3  PRINT "Digite um valor nao negativo";
4  INPUT VALOR
5  IF VALOR < 0 THEN 6 ELSE 9
6  PRINT "Voce digitou : " VALOR
7  PRINT "Este valor e' negativo"
8  GOTO 12
9  PRINT "Voce digitou certo : " VALOR
10  PRINT "Este valor nao e' negativo"
11  GOTO 12
12  END
```

```
Turbo Pascal

program Teste8;

var
    Valor : real;

begin
    write('Digite um valor nao negativo');
    readln(Valor);
    if Valor < 0 then
        begin
            writeln('Voce digitou : ',Valor);
            writeln('Este valor e negativo');
        end
    else
        begin
            writeln('Voce digitou certo : ',
            Valor);
            writeln('Este valor nao e
            negativo');
        end;
end.
```

TABELA VERDADE

Ao compor duas ou mais condições num mesmo *if*, precisamos lembrar-nos da tabela verdade. Por ser uma regra da lógica matemática, a sua utilização é igual à do BASIC.

Sendo *p* e *q* duas condições que podem ser verdadeiras (V) ou falsas (F) e *r* o resultado, temos:

p and q		
p	q	r
V	V	V
V	F	F
F	V	F
F	F	F

Tabela *p* e *q*

p or q		
p	q	r
V	V	V
V	F	V
F	V	V
F	F	F

Tabela *p* ou *q*

p xor q		
p	q	r
V	V	F
V	F	V
F	V	V
F	F	F

Tabela *p* ou *q*

("ou forte" ou "ou exclusivo")

not p	
p	r
V	F
F	V

Tabela não *p*

(negação)

```
BASIC

1' Programa Teste9
2'
3  PRINT "Digite um valor positivo menor que 100";
4  INPUT VALOR
5  IF VALOR > 0 AND VALOR < 100 THEN 6 ELSE 8
6  PRINT "Voce digitou corretamente"
7  GOTO 10
8  PRINT "Voce digitou errado"
9  GOTO 10
10  END
```

Turbo Pascal

```
program Teste9;

var
   Valor : real;

begin
   write('Digite um valor positivo menor que 100');
   readln(Valor);
   if (Valor > 0) and (Valor < 100) then
      begin
         writeln('Voce digitou corretamente');
      end
   else
      begin
         writeln('Voce digitou errado');
      end;
end.
```

Ainda bem que a lógica matemática está acima de qualquer definição de linguagem, concorda?

8

UMA PEQUENA PAUSA

REFRESCANDO

Usando os comandos do Turbo Pascal até aqui apresentados, já podemos fazer e rodar muitos programas simples. Para sedimentar os conhecimentos, o leitor precisa fazer os seus próprios programas e testá-los no microcomputador. Se lembrar bem, não foi através da prática que o leitor aprendeu a programar em BASIC?

COMPATIBILIDADE

A versão do Turbo Pascal para equipamentos de 16 *bits* possui muito mais recurso do que para os de 8 *bits*. Mas os comandos básicos foram mantidos de tal forma que um programa feito em micros de 8 *bits* roda perfeitamente nos de 16 *bits*, desde que não haja chamadas para sub-rotinas em linguagem de máquina.

O transplante em direção oposta é também possível, desde que não tenham sido usados comandos específicos (em geral, controle de telas e gráficos) dos equipamentos de 16 *bits*.

POTENCIAÇÃO

É um comando que não faz parte do Pascal nem do Turbo Pascal. Em BASIC, a potenciação seria:

```
BASIC

A=B^C
```

A expressão acima indica que *A* recebe o valor do cálculo *B* elevado a *C*.

Em linguagens mais antigas bastava colocar dois sinais de multiplicação:

```
A=B**C
```

Para contornar esta situação basta ter um pequeno conhecimento adicional de álgebra e chegar à conclusão de que a expressão a seguir é equivalente.

```
Turbo Pascal

A:=exp(C*ln(B))
```

MATRIZES INTEIRAS

Para transferir todos os elementos de uma matriz para outra em BASIC era necessário utilizar técnicas que acessassem todos os elementos da matriz usando geralmente o comando FOR.

```
BASIC

FOR I=1 TO 10
COP(I)=MAT(I)
NEXT I
END
```

Se a matriz MAT estivesse anteriormente carregada, precisaria colocar num laço FOR..NEXT para criar uma matriz igual chamada COP.

Em Turbo Pascal isto não é necessário. Desde que Mat e Cop tenham sido corretamente declarados como matrizes, pode-se fazer a operação simplesmente indicando:

```
Turbo Pascal

Cop:=Mat;
```

A atribuição acima pode ser feita com qualquer matriz, independendo da sua dimensão.

INTERRUPÇÃO

Quando se compila um programa, a pressão em qualquer tecla faz interromper a operação e o Turbo Pascal pergunta se deseja abortar ou não. Após esta pergunta, pode-se responder sim (Y) ou não (N).

Suponha que inicie um processo de compilação indesejado. Em vez de interromper com uma tecla e responder *Y* para abortar, pode-se pressionar diretamente *Y* que a compilação será suspensa sem perguntas.

CARACTERES DE CONTROLE

Os primeiros caracteres do código ASCII são conhecidos como caracteres de controle. Em BASIC, para enviar estes caracteres era necessário executar o seguinte:

```
BASIC

PRINT CHR$(7)
```

O exemplo acima aciona o *beep* do seu microcomputador. No Turbo Pascal, o envio pode ser feito assim:

```
Turbo Pascal

write(^G);
```

Sendo o símbolo ^G (control G) definido como *beep*, o seu microcomputador irá executar esta função ao encontrá-lo. ^G não se obtém apertando simultaneamente a tecla control e G. Neste exemplo são caracteres distintos ^ e G.

INICIALIZAÇÃO DE VARIÁVEIS

O BASIC é uma linguagem "boazinha", pois qualquer variável antes de receber uma atribuição vale zero ou é nula.

Isto não acontece com Turbo Pascal. Se aparecerem valores absurdos ou de repente surgir um valor próximo ao esperado pelo programador, eles se tornarão fontes de *bugs* (erros) para o seu programa.

Portanto, inicialize todas as variáveis que irá utilizar. Neste caso, o BASIC é o mocinho e o Turbo Pascal é o bandido.

COMANDO Q

O *menu* principal possui o comando Q (*Quit*) para sair do Turbo Pascal e voltar para o sistema operacional. O Turbo Pascal tem rotinas que checam se o usuário salvou em disco ou não o programa em disco antes de efetuar operações que acarretam perda do programa atualmente em memória.

Apesar destas qualidades do Turbo Pascal, não espere que envie a mensagem. Antecipe e fique tranqüilo, pois mesmo programadores experientes podem perder o programa por bobagens. Atenção redobrada se o leitor é usuário do dBASE, pois o seu editor de textos salva o programa automaticamente ao sair da edição.

9

COMANDOS REPEAT E WHILE

REPEAT

É um comando com o qual não existe algo parecido em BASIC. Para exercer uma função parecida, o BASIC necessita compor uma sentença do tipo IF..THEN..ELSE.

O Comando *repeat* controla repetições de trecho de programas tal como o *for*. Mas há duas diferenças fundamentais que são estas:

1. O número de vezes que as linhas de programa situadas dentro do *repeat..until* não precisa estar predefinido. Os próprios comandos situados no *repeat..until* podem determinar a saída ou não do laço de repetição.
2. A verificação da suficiência ou não da condição de repetição é feita no fim do laço.

Vejamos uma pequena comparação do uso deste comando:

BASIC

```
1' Programa Teste10
2'
3  VALOR=0
4  PRINT "Digite um número negativo";
5  INPUT VALOR
6  IF VALOR < 0 THEN 7 ELSE 4
7  END
```

O programa retroapresentado só termina se for digitado um número menor que zero. Observe que a verificação do dado digitado é feita no fim do trecho a ser repetido.

```
Turbo Pascal
program Teste10;

var
    Valor : real;

begin
    Valor:=0;
    repeat
        write('Digite um numero negativo');
        readln(Valor);
    until Valor < 0;
end.
```

Mais duas observações importantes:

1. Os comandos a serem repetidos não necessitam ser limitados por *begin* e *end*.
2. Os comandos internos ao *repeat..until* são executados pelo menos uma única vez.

O programa BASIC a seguir imprime números de zero a 100.

```
BASIC
1' Programa Teste11
2'
3   I=0
4   PRINT I
5   I=I+1
6   IF I>100 THEN 7 ELSE 4
7   END
```

```
Turbo Pascal

program Teste11;
var
    I : real;
begin
    I:=0;
    repeat
        writeln(I);
        I:=I+1;
    until I>100;
end.
```

A elegância da apresentação e a legibilidade de programas em Turbo Pascal já começa a aparecer por aqui. Vai melhorar ainda mais após o aprendizado do próximo comando.

WHILE

O BASIC possui comando WHILE...WEND equivalente ao do Turbo Pascal. Mas este comando é pouquíssimo lembrado pelos programadores BASIC.

Os comandos a serem repetidos devem estar limitados por *begin* e *end*. A condição de repetição é testada antes e não depois, como é feito pelo *repeat..until*. Portanto, os comandos envolvidos no *while* podem não ser executados nenhuma vez.

Usando *while*, o programa para imprimir números de zero a 100 ficaria assim:

```
BASIC

1 ' Programa Teste12
2 '
3   I=0
4   WHILE  I<=100
5   PRINT  I
6   I=I+1
7   WEND
8   END
```

Como o WHILE...WEND é pouco utilizado e ausente em versões mais primitivas do BASIC, aqui está o mesmo programa com a utilização de IF..THEN..ELSE:

```
BASIC

1 ' Programa Teste12
2 '
3   I=0
4   IF  I<=100  THEN 5 ELSE 8
5   PRINT  I
6   I=I+1
7   GOTO 4
8   END
```

Agora a tradução das versões BASIC para Turbo Pascal com a utilização do comando *while*:

```
Turbo Pascal

program Teste12;

var
    I : real;

begin
    I:=0;
    while I<=100 do
        begin
            writeln(I);
            I:=I+1;
        end;
end.
```

As condições que acompanham o *repeat* e o *while* podem ser compostas através de *or*, *and*, *xor* e *not,* tal como estudamos no comando *if.*

RESUMO

Os três tipos de comandos de controle de repetição devem ser utilizados visando às seguintes características:

Comando	Características
for	Utiliza-se no caso de saber o número de repetições antecipadamente.
repeat	Controla a condição de repetição após a sua execução.
while	Controla a condição de repetição antes da sua execução.

Pode-se acrescentar à tabela acima que:

1. *Repeat* é sempre executado pelo menos uma vez.
2. *While* pode nunca ser executado.
3. Não se deve alterar o valor da variável de controle numa sentença *for*.
4. No *repeat* e *while*, o incremento ou decremento da variável de controle deve ser providenciado internamente pelo programador.
5. O passo do incremento ou decremento automático no Turbo Pascal é 1, enquanto no BASIC é livre.

10

COMANDOS CASE E GOTO

CASE

O comando *case* é extremamente importante para estruturação de um programa que possua diversas variantes de execução, tornando-o bem legível, pois evita o uso repetido do *if*.

Não há um comando equivalente em BASIC, mas os comandos ON..GOSUB e ON..GOTO possuem características parecidas. Mas como estes comandos não são de aplicação geral, iremos utilizar *if* nos exemplos em BASIC.

```
BASIC

1' Programa Teste 13
2'
3' PRINT "Digite um numero (0-2)"
4  INPUT VALOR
5  IF VALOR = 0 THEN PRINT "numero digitado = zero"
6  IF VALOR = 1 THEN PRINT "numero digitado = um"
7  IF VALOR = 2 THEN PRINT "numero digitado = dois"
8  END
```

```
Turbo Pascal

program Teste13;

var
   Valor : integer;

begin
   write('Digite um numero (0-2) ');
   readln(Valor);
   case Valor of

      0 : begin
            writeln('numero digitado = zero');
          end;

      1 : begin
            writeln('numero digitado = um');
          end;

      2 : begin
            writeln('numero digitado = dois');
          end;

  end;   { do case }
end.
```

O comando *case* oferece uma opção de *else* (se não). Veja o exemplo de utilização deste recurso.

```
Turbo Pascal

program Teste14;

var
   Valor : integer;

begin
   writeln('Digite um numero (0-2) ');
   readln(Valor);
   case Valor of
```

```
0 : begin
        writeln('numero digitado = zero');
    end;

1 : begin
        writeln('numero digitado = um');
    end;

2 : begin
        writeln('numero digitado = dois');
    end;

else
        begin
            writeln('Voce digitou um numero');
            writeln('fora da faixa (0-2)');
        end;

    end;
end.
```

Observe a colocação dos *end*. O penúltimo *end* está fechando o comando *case*.

A variável usada no *case* pode ser de qualquer tipo até agora estudado, tomando cuidado para não misturar os elementos. Veja um exemplo de *case* e de variável de seleção do tipo *char:*

```
BASIC

1' Programa Teste15
2'
3  PRINT "Digite S ou N";
4  INPUT RESP$
5  IF RESP$ = "S" THEN PRINT "Voce digitou S"
6  IF RESP$ = "N" THEN PRINT "Voce digitou N"
7  END
```

```
Turbo Pascal

program Teste15;

var
    Resp : char;

begin
    write('Digite S ou N');
    readln(Resp);
    case Resp of

        'S' : begin
                  writeln('Voce digitou S');
              end;

        'N' : begin
                  writeln('Voce digitou N');
              end;

    end;
end.
```

Observe a presença dos apóstrofos limitando as letras *S* e *N* para indicar que a comparação deve ser feita em relação a um *string*.

GOTO

Um dos pontos fortes do Pascal e outras linguagens modernas é a eliminação do comando de desvio incondicional do tipo GOTO do BASIC que dificulta enormemente a leitura de um programa. Mas esta posição não deve ser assumida radicalmente. Há momentos em que a existência de um comando de desvio, mesmo passando por cima de regras da programação estruturada, é extremamente útil e até melhora a legibilidade. Para estes casos, o Turbo Pascal possui o comando *goto* que desvia o processamento para outra linha do programa.

Algum tipo de identificador deve ser utilizado para indicar o destino do desvio, pois Turbo Pascal não possui numeração de linhas. Esta sinalização é conhecida como *label* (rótulo) e também necessita ser declarada. Vejamos um exemplo:

BASIC

```
1' programa Teste16
2'
3   DIM MAT(10)
4   FOR I=1 TO 10
5   INPUT MAT(I)
6   IF MAT(I) = 0 THEN 8
7   NEXT I
8   END
```

O programa acima permite a entrada de dez números no máximo ou até digitar zero. A construção acima não é recomendada mesmo em BASIC, pois o *for* deve ser utilizado quando o número de repetições está predeterminado.

Turbo Pascal

```
program Teste16;

var
    Mat : array [1..10] of real;
    I : integer;

label
    Fim;

begin
    for I:=1 to 10 do
        begin
            readln(Mat[I]);
            if Mat[I] = 0 then
                begin
                    goto Fim;
                end;
        end;
    Fim:
end.
```

Existe uma importante regra na utilização do *goto*. Apesar de ainda não termos apresentado o que é procedimento e função, o comando de desvio somente pode ser utilizado dentro de um dos blocos do programa retromencionado. Isto significa que não se pode desviar de um procedimento para outro, ou seja, de um subprograma para outro.

O conceito de subprograma será desenvolvido a seguir e deverá esclarecer as eventuais dúvidas.

11

PROCEDIMENTOS

O QUE É PROCEDIMENTO

Uma prática bastante recomendada aos programadores em BASIC é a utilização de sub-rotinas. As sub-rotinas permitem certo desmembramento do programa em programa principal e subprogramas (sub-rotinas). As pessoas que possuem maior vivência em BASIC já devem ter tentado construir uma biblioteca de sub-rotinas que possam ser margeadas (incluídas no programa principal.

Para trabalhar com sub-rotinas em BASIC era necessário tomar cuidado para que não houvesse problemas de interferência ou colisão da numeração de linhas. Duas sub-rotinas diferentes precisariam ter o número de suas linhas diferentes.

Há mais uma dificuldade em BASIC. Era necessária atenção para que no programa principal não fosse utilizada nenhuma variável que integrasse a sub-rotina, pois poderia haver problemas de alteração do seu conteúdo.

O Turbo Pascal possui grandes vantagens para trabalhar com sub-rotinas. Doravante, chamaremos as sub-rotinas procedimentos e a sua declaração em Turbo Pascal, *procedure*.

As vantagens dos procedimentos do Turbo Pascal em relação às sub-rotinas do BASIC podem ser resumidas no seguinte:

1. Utilização de variáveis locais.
2. Execução do procedimento apenas fazendo referência ao nome dele (sem GOSUB).
3. Não há preocupação com numeração das linhas.

Eis um pequeno exemplo de aplicação de sub-rotinas:

```
BASIC

1' Programa Teste17
2'
3   GOSUB 100
4   END
98'
99' Subrotina Imprimem
100   PRINT "Eu sou Turbo Pascal"
101   RETURN
```

O programa acima imprime a frase "Eu sou Turbo Pascal" devido a um GOSUB efetuado na linha 3. Se houvesse outro GOSUB 100, a mensagem seria novamente impressa.

A sua tradução para Turbo Pascal ficaria assim:

```
Turbo Pascal

program Teste17;

procedure Imprimem;
begin
    writeln('Eu sou Turbo Pascal');
end;

(* programa principal *)

begin
    Imprimem;
end.
```

Um hábito muito difundido entre programadores BASIC é deixar para o final do programa as sub-rotinas (geralmente com altos números de linhas), mas o Turbo Pascal exige o contrário.

Os procedimentos devem vir no início do programa, pois a chamada efetuada pelo programa principal utiliza o próprio nome do procedimento. A causa desta ordem das coisas é a seguinte: À medida que o compilador vai avançando o seu trabalho, ele confere as palavras e as sentenças para verificar se não há erro de sintaxe. Caso o compilador encontre uma pala-

vra desconhecida, torna-se necessário analisar se foi erro do programador ou uma palavra nova criada por ele. Se a declaração do procedimento não viesse antes da sua utilização, o compilador não teria como distinguir entre um erro e a chamada de um procedimento.

É neste ponto que o Turbo Pascal se torna uma poderosa linguagem. O Pascal em si possui poucos comandos. O Turbo Pascal recebeu muitos implementos pré-declarados dentro de si mesmo. O usuário pode criar a sua biblioteca de procedimentos e enriquecer o vocabulário do Turbo Pascal. A força desta linguagem passa a depender da pesquisa e da experiência do seu usuário.

VARIÁVEIS LOCAIS

A força das variáveis locais evidencia-se ao se imaginar um grande programa escrito simultaneamente por diversos programadores. Cada programador não precisa preocupar-se com o nome das variáveis que outro esteja utilizando para desenvolver o seu procedimento. Cada variável só vale dentro do seu respectivo procedimento e deve ser declarada cada vez que se fizer presente. Os valores contidos em variáveis locais perdem-se após a execução do procedimento.

VARIÁVEIS GLOBAIS

Estas variáveis são semelhantes às do BASIC. Valem para qualquer parte do programa, exceto no caso de ser novamente declarada como variável local dentro de um procedimento. Toda declaração de variáveis é feita no início do programa.

Veja um exemplo demonstrativo de variáveis locais e globais:

```
Turbo Pascal

program Teste18;

(* declaracao de variaveis globais *)
var
    I : integer;

(*-----------------------------------------------------------------------*)
```

```
procedure Teste;
var
    I : integer;

begin
    I:=5;
end;

(*------------- programa principal ----------------------*)

begin
    I:=1;
    Teste;
    writeln(I);
end.
```

Ao dar *run* ao programa acima, o valor impresso na tela deverá ser 1.0000000000E+00 que é 1 e não 5, pois a variável *I* foi também declarada dentro do procedimento. Esta declaração tornou-a uma variável local e, portanto, distinta da *I* global. Se retirássemos a declaração da variável *I* que está dentro do procedimento, esta seria tratada como variável global e o resultado a ser impresso seria 5.000000000E+00, que é 5.

PARÂMETROS

Um procedimento pode trabalhar com valores, mesmo que não sejam usadas variáveis globais. Para isto temos um recurso chamado passagens de parâmetros. O grande poder dos parâmetros está nesta correta passagem de valores entre o programa principal ou procedimento e o procedimento chamado.

Turbo Pascal

```
program Teste19;

procedure Mult(X,Y : real);
begin
    writeln(X*Y);
end;

(*------------------- programa principal ----------*)

begin
    Mult(2,3);
end.
```

O programa anterior consiste em um procedimento que imprime o valor do produto de dois números recebidos pela passagem de parâmetros e um programa principal que manda executar o procedimento Mult enviando os valores numéricos 2 e 3 como parâmetros.

Observe que:

1. Não se utilizou nenhuma variável global.
2. As variáveis *X* e *Y* do procedimento não precisam ser declaradas, pois são variáveis de passagem. Foram declaradas no início do procedimento ao lado da palavra *procedure*.
3. O programa principal não utiliza nenhuma variável.
4. As variáveis de passagem recebem os valores na mesma ordem em que são declaradas como parâmetros. Isto quer dizer que, para efeito de cálculo dentro do procedimento Mult, o *X* assumiu o valor 2 e *Y* assumiu o valor 3.

Como a perfeita compreensão do funcionamento de um procedimento é extremamente importante, vamos apresentar mais alguns exemplos.

```
Turbo Pascal

program Teste20;

procedure Mult(X,Y : real);
var
    Z : real;      (* variavel local *)

begin
    Z:=X*Y;
    writeln(Z);
end;

(*------------------- programa principal --------------------*)

begin
    Mult(2,3);
end.
```

Neste programa foi incluída a variável *Z*, que é local, para armazenamento temporário do resultado da multiplicação. É também um bom hábito separar cada um dos módulos do programa de forma bem visível para facilitar o acompanhamento, principalmente se os programas forem ficando extensos.

O resultado do processamento do programa anterior é 6.

Utilizando somente variáveis globais, não haveria necessidade de passar parâmetros para procedimentos. O programa ficaria assim:

```
Turbo Pascal

program Teste21;

var
    X,Y,Z : real;       (* variaveis globais *)

(*-------------------------------------------------------------------------*)

procedure Mult;
begin
    Z:=X*Y;
    writeln(Z);
end;

(*------------ programa principal  ----------------------*)

begin
    X:=2;
    Y:=3;
    Mult;
end.
```

E o resultado também seria 6. Neste caso, *writeln* poderia estar no programa principal que iria imprimir do mesmo modo o mesmo valor desde que fosse após a chamada do procedimento.

Há mais um modo de trabalhar com procedimentos. Até agora, os valores eram passados a partir de uma chamada para o procedimento. O processo inverso, ou seja, passar valores de um procedimento para o programa principal ou procedimento que o chamou pode ser efetuado mediante pequena alteração na declaração, acrescentando a palavra *var*. Vejamos um exemplo:

```
Turbo Pascal

program Teste22;

var
    A,B,C : real;

(*-------------------------------------------------------------------------*)

procedure Mult(X,Y : real;  var Z : real);
begin
    Z:=X*Y;
end;

(*---------------- programa principal ----------------------*)

begin
    A:=2;
    B:=3;
    Mult(A,B,C);
    writeln(C);
end.
```

Este exemplo diferencia-se dos demais devido aos seguintes fatores:

1. Inclusão de mais um parâmetro na chamada do procedimento.
2. O valor calculado no procedimento é impresso no programa principal.

A novidade em relação aos exemplos anteriores está no item 3. Quando se usa este tipo de declaração do procedimento, permite-se que um valor seja retornado de um procedimento para utilização fora dele. No exemplo, a variável global *Z* passou o valor 6 de volta para a variável global *C* quando o processamento retornou do procedimento para o programa principal.

Numa comparação entre os dois tipos de procedimentos temos:

```
Turbo Pascal

program Teste23;

var
    A,B : real;
```



```
(*-------------------------------------------------------------------------*)
```

```
procedure Soma(X,Y : real);
begin
    X:=X+Y;
    writeln(X);
end;

(*------------------- programa principal ---------------------*)

begin
    A:=2;
    B:=10;
    Soma(A,B);
    writeln(A);
end.
```

Os resultados obtidos ao rodar o programa acima são, respectivamente, 12 e 2. Observe que a variável global *A* permaneceu com o mesmo valor, pois houve apenas a passagem do parâmetro no sentido de ida.

Turbo Pascal

```
program Teste24;

var
    A,B : real;

(*-----------------------------------------------------------*)

procedure Soma(var X,Y : real);
begin
    X:=X+Y;
    writeln(X);
end;

(*------------------- programa principal ---------------------*)

begin
    A:=2;
    B:=10;
    Soma(A,B);
    writeln(A);
end.
```

Desta vez, os resultados impressos são respectivamente 12 e 12. Devido à presença do *var* na declaração, o valor da variável *X* foi passado de volta para a variável global *A* ao retornar para o programa principal.

Neste capítulo não apareceu nenhuma comparação entre um programa BASIC e Turbo Pascal, pois o procedimento é um recurso incomparavelmente mais poderoso do que as sub-rotinas em BASIC. Mas devido, exatamente, a esta característica, recomenda-se prosseguir os estudos somente após um perfeito entendimento deste capítulo e, portanto, do funcionamento de procedimentos.

12

FUNÇÕES

O QUE SÃO FUNÇÕES

As funções do Turbo Pascal são utilizadas (chamadas) de forma semelhante às do BASIC. Mas não há limitação para complexidade de uma função. No BASIC, uma função pode apenas possuir expressões de cálculo ou no máximo uma seqüência de decisões lógicas. No Turbo Pascal, uma função pode ser tão bem elaborada quanto um programa qualquer, não havendo nenhuma limitação neste sentido.

A estrutura de uma função é muito parecida com um procedimento. Pode-se imaginar que uma função é um procedimento com características especiais quanto ao retorno de valores.

Turbo Pascal

```
program Teste25;

var
A,B,C : real;

(*-------------------------------------------------------------------------*)

function Prod(X,Y : real) : real;
begin
    Prod:=X*Y;
end;

(*----------------- programa principal ----------------------*)
```

```
begin
    A:=2;
    B:=10;
    C:=Prod(A,B);
    write(C);
end.
```

O resultado impresso será 20.

Na declaração da função, além do que está dentro dos parênteses, há mais uma definição, indicando que a função é do tipo real, ou seja, retorna ao local onde foi chamada, assumindo a condição de uma variável real. Neste caso, a função só assume valores coerentes com essa declaração.

Uma característica que distingue uma função de um procedimento é que a primeira pode ser impressa, atribuída ou participar de cálculos como uma variável qualquer. Isto quer dizer que, em vez de utilizar a variável *C*, podemos imprimir diretamente a função desta maneira:

```
writeln(Prod(A,B))
```

E teremos o mesmo resultado.

Para funções simples como as deste exemplo é possível visualizar um programa equivalente em BASIC. Vamos tentar, então:

```
BASIC

1 ' Programa Teste25;
2 '
3   DEF FNP(X,Y)=X*Y
4 '
5   A=2
6   B=10
7   C=FNP(A,B)
8   PRINT C
9   END
```

Para imprimir diretamente sem utilizar a variável C basta eliminar a linha 7 e substituir a 8 por:

```
8   PRINT FNP(A,B);
```

Vejamos um exemplo de uma função que soma dois números inteiros digitados no teclado.

BASIC

```
1'  Programa Teste26
2'
3   DEF FNS%(A%,B%)=A%+B%
4'
5   INPUT X%
6   INPUT Y%
7   PRINT FNS(X%,Y%)
8   END
```

Se forem digitados respectivamente 150 e 40, o programa imprimirá a soma 190.

Turbo Pascal

```
program Teste26;

var
   X,Y : integer;

(*-------------------------------------------------------------------------*)

function Soma(A,B : integer) : integer;
begin
   Soma:=A+B;
end;

(*------------------ programa principal ------------------*)

begin
   readln(X);
   readln(Y);
   writeln(Soma(X,Y));
end.
```

Observação:

Na declaração da função podemos acrescentar a partícula *var* e obter os mesmos recursos adicionais do procedimento feito desta forma.

```
program Teste27;

var
    A,B,C : real;

(*-------------------------------------------------------------------------*)

function Prod(X : real; var Y : real) : real;
begin
    Prod:=X*Y;
    X:=1;
    Y:=1;
end;

(*------------------- programa principal ------------------*)

begin
    A:=2;
    B:=10;
    writeln(Prod(A,B));
end.
```

Após a chamada da função, as variáveis *X* e *Y* terminam valendo 1. Mas na declaração da função somente a variável *Y* recebeu *var*. Isto implica que o valor final do *Y* é passado de volta ao local onde foi solicitada a função, fazendo com que *B* passe a tornar-se 1 em vez do valor original 10. Isto tudo não ocorreu com a variável *A*, que continua valendo 2 tal como antes.

No caso de uma função ser do tipo *string*, não se deve definir:

```
function Exemplo(                    ) : string[10];
```

Deve-se definir primeiro o tipo de dado e depois colocar o nome deste tipo na declaração da função, conforme apresentado no Capítulo 14 – Declaração de tipos.

```
type
    Str10 : string[10];

function Exemplo(                    ) : Str10;
```

13

COMPLEMENTO SOBRE PROCEDIMENTOS

TELA

O Turbo Pascal já possui funções e procedimentos pré-declarados para controle da tela. Algumas são muito poderosas quando comparadas aos comandos de controle de tela do BASIC.

Clrscr – Equivale ao CLS do BASIC. Em algumas implementações do BASIC, o comando para limpar a tela pode ser HOME ou PRINT seguido de um caractere de controle (CHR$(12) ou CHR$(26)).

Clreol – Este comando apaga todos os caracteres da linha que estão à direita do cursor.

Delline – Apaga a linha onde está o cursor e causa *scroll* nas linhas seguintes de tal modo que preencha a linha deletada.

Insline – Este comando funciona de modo oposto ao *delline*. Insere uma linha vazia na posição onde está o cursor. Provoca *scroll* nas linhas seguintes.

GotoXY(X,Y) – Movimenta o cursor para linha *X*, coluna *Y* da tela. É igual ao comando LOCATE do BASIC. A contagem das linhas e colunas começa no 1, ou seja, o canto superior esquerdo possui coordenadas (1,1).

Lowvideo – Diminui a luminosidade dos caracteres na tela.

Highvideo – Aumenta a luminosidade dos caracteres na tela.

Normvideo – Retorna à luminosidade normal.

OUTRAS UTILIDADES

Delay(X) – Provoca uma pausa no processamento equivalente a *X* milissegundos. *X* deve ser definido como número inteiro.

Exit – Abandona o bloco corrente. Estando numa sub–rotina, retorna ao bloco onde foi feita a sua chamada. Se usado no programa principal, termina a execução do mesmo.

Halt – Interrompe o processamento do programa e retorna ao nível de sistema operacional.

Randomize – Inicializa o gerador de números aleatórios com um valor aleatório.

Eis um exemplo de programas equivalentes nas duas linguagens:

BASIC

```
1 ' Programa Teste28
2 '
3   CLS
4   FOR I=1 TO 20
5   LOCATE I,10
6   PRINT "Turbo Pascal"
7   NEXT
8   CLS
9   END
```

Turbo Pascal

```
program Teste28;

var
    I : integer;

begin
    clrscr;
    for I:=1 to 20 do
        begin
            gotoXY(10,I);
            writeln('Turbo Pascal');
        end;
    clrscr;
end.
```

Observação:

Alguns equipamentos não conseguem utilizar os comandos de controle de intensidade de vídeo devido à limitação de *hardware* (equipamento em si ou monitor de vídeo).

BASIC

```
1 ' Programa Teste29
2 '
3   CLS
4   LOCATE 6,25:PRINT "<0> Retorno ao MS-DOS"
5   LOCATE 8,25:PRINT "<1> Inclusao"
6   LOCATE 10,25:PRINT "<2> Alteracao"
7   LOCATE 12,25:PRINT "<3> Exclusao"
8   LOCATE 20,25:PRINT "Faca a sua opcao : "
9   END
```

Este programa poderia fazer parte de um cadastro de clientes e a sua transcrição para Turbo Pascal ficaria do seguinte modo:

Turbo Pascal

```
program teste29;

begin
    clrscr;
    gotoXY(25,6); write('<0> Retorno ao MS-DOS');
    gotoXY(25,8); write('<1> Inclusao');
    gotoXY(25,10); write('<2> Alteracao');
    gotoXY(25,12); write('<3> Exclusao');
    gotoXY(25,20); write('Faca a opcao : ');
end.
```

14

DECLARAÇÃO DE TIPOS

INTRODUÇÃO

Apesar de a declaração de tipos que usa o comando *type* ser algo básico para a estrutura da linguagem Pascal, vínhamos adiando este estudo, pois para grande parte dos programas não era necessário o domínio sobre este assunto.

Até o momento estivemos trabalhando com os dados do tipo:

- integer
- real
- byte
- string
- char
- boolean

Em condições normais, estes tipos de dados são suficientes para resolver os problemas. O Turbo Pascal, no entanto, oferece mais alguns recursos de definição, como:

- **scalar** (escalar);
- **subrange** (subintervalo);
- **set** (conjunto);
- **record** (registro).

SCALAR

Scalar type é um tipo de dado que se caracteriza pela semelhança a uma matriz unidimensional com os seus elementos ordenados. Isto inclui tipos como *integer, real, byte, char* e *boolean.* A grande vantagem aqui é que o próprio usuário pode definir os seus tipos de dados.

Já sabemos que os dados do tipo *byte* são formados de números entre 0 e 255. Podemos criar dados do tipo dias da semana, como domingo, segunda, terça, quarta, quinta, sexta e sábado. Isto quer dizer que vou de-

clarar um tipo de dados conforme o apresentado, e a variável associada a este tipo somente pode assumir os dados que são os dias da semana.

Esta é uma estrutura que não possui o seu equivalente em BASIC e, por isso, iremos direto ao Turbo Pascal.

Turbo Pascal

```
program Teste30;

type
   Semana = (domingo,segunda,terca,quarta,quinta,sexta,sabado);

var
   Dia : Semana;

begin
   for Dia:=domingo to sabado do
      begin

      end;
end.
```

O programa anterior tem as seguintes características:

1. Uso do *type* para declaração (definição) do tipo.
2. O tipo de dado chamado Semana é composto pelos valores definidos no programa que são os dias da semana (de domingo a sábado).
3. A variável Dia pode assumir valores que são os dias da semana.
4. A variável Dia é utilizada como controlador do comando *for*.
5. A repetição será executada sete vezes, e serão atribuídas à variável Dia todas as sete possibilidades definidas. Um *escalar* definido pelo usuário não pode ser impresso através de um comando *write* ou *writeln*. Deve ser detectado de modo indireto como no próximo exemplo.

Vejamos mais um exemplo de definição de escalares utilizando planetas do Sistema Solar. Supondo que uma variável seja utilizada somente para representar planetas interiores, como Mercúrio, Vênus, Terra e Marte, qualquer tentativa de atribuir um valor que não seja um dos quatro citados anteriormente poderá provocar o erro.

Turbo Pascal

```
program Teste31;

type
   Planint = (Mercurio,Venus,Terra,Marte);

var
   Nome : Planint;

begin
   for Nome:=Marte downto Mercurio do
      begin
case Nome of
   Mercurio : begin
                 writeln('Menor em tamanho');
              end;

   Venus : begin
              writeln('Estrela Dalva');
           end;

   Terra : begin
              writeln('Cheio de agua');
           end;

   Marte : begin
              writeln('O planeta vermelho');
           end;
      end;
   end;
end.
```

A saída deste programa é:

```
O planeta vermelho
Cheio de agua
Estrela Dalva
Menor em tamanho
```

Observe que no comando *for* se utilizou a opção *downto*, o que inverteu a ordem de impressão dos dados do tipo *Planint*.

Para o próximo item a ser estudado é extremamente importante lembrar que os tipos *integer*, *real*, *byte*, *char* e *boolean* também fazem parte dos escalares. A diferença é que eles já estão predefinidos.

SUBRANGE

Todos os dados do tipo *escalar* apresentam intervalos em que a sua existência é válida. A palavra *subintervalo* indica um tipo de definição de dados conhecido como *subrange*. A característica principal destes dados é a utilização de apenas trechos (subintervalos) de algo maior e mais completo.

```
Turbo Pascal

program Teste32;

type
    Nchapa = 0..9999;

var
    Chapa : Nchapa;

begin
    readln(Chapa);
    writeln(Chapa);
end.
```

Supondo que o número de algarismos que compõem a placa de identificação de um veículo seja quatro, surge a limitação do seu valor entre 0 e 9999. Portanto, não há possibilidade de a variável encarregada de assumir este número ultrapassar o limite inferior e superior. Devido a isso, declarou-se que o tipo de dados *Nchapa* está na faixa 0..9999 e que a variável *Chapa* é do tipo *Nchapa*. O tipo de dado *Nchapa* é subintervalo de um *escalar* predefinido.

```
    type
        Letmin = 'a'..'z';
        Planeta = (Mercurio,Venus,Terra,Marte,
Jupiter,Saturno,Urano,Plutao);
        Planint = Mercurio..Marte;
        Planext = Jupiter..Plutao;
```

1. O tipo de dados *Letmin* é um subintervalo do *escalar char* (alfabeto em letras minúsculas).
2. O tipo de dados *Planint* (planetas interiores) e *Planext* (planetas exteriores) são subintervalos de um tipo de dados definidos como Planeta (planetas do Sistema Solar).

Observação:

O Turbo Pascal checa a validade dos subintervalos somente quando a diretiva de compilação *R* estiver ativada. Veja melhor no Capítulo 16, que trata de diretivas de compilação.

CONJUNTO

Os dados do tipo conjunto são reuniões de diversos elementos do mesmo tipo. Para se tornar um elemento do conjunto, este precisa ser do tipo *escalar*, exceto os números reais. Vejamos o formato desta declaração através de um programa-exemplo:

```
Turbo Pascal

program Teste33;

type
    Num = set of byte;

var
    Primo,Impar,Par,Result1,Result2 : Num

begin
    Primo:=[1,2,3,5,7];
    Impar:=[1,3,5,7,9];
    Par  :=[2,4,6,8];
    Result1:=Primo+Impar;
    Result2:=Primo*Par;
end.
```

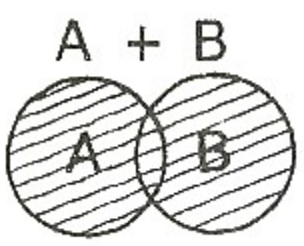
O programa acima possui as seguintes características:

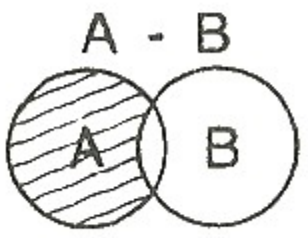
1. Define-se *Num* como tipo de dado *set of* formado por números do tipo *byte*.
2. Definem-se as variáveis de conjunto Primo, Ímpar, Par, *Result1* e *Result2* como do tipo *Num*.
3. Atribuem-se os respectivos números (maior que 0 e menor que 10).
4. Executa-se a operação de união (+) e intersecção (*) entre os conjuntos.

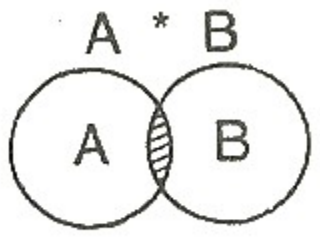
5. As variáveis de conjunto *Result1* e *Result2* irão conter os seguintes elementos:

Result1 {1,2,3,5,7,9}
Result2 {2

A operação dos conjuntos obedece às seguintes notações:







Decisões lógicos
A = B: A igual a B
A < > B: A diferente de B
A >= B: A contém B
A <= B: A está contido em B
A in B: A pertence a B

O uso do *in* é muito útil para a programação. Vejamos um exemplo:

Turbo Pascal

```
program Teste34;

var
   Ch : char;

begin
   write('Digite uma letra minuscula entre a e h');
   readln(Ch);
   if Ch in ['a'..'h'] then
      begin
         writeln('OK!');
      end
   else
      begin
         writeln('Erro!');
      end;
end.
```

O programa acima imprime OK! se a letra digitada estiver conforme a mensagem. Se não, o comando *if* desvia o processamento e imprime Erro! Uma questão importante: o *set* (conjunto) foi definido na própria sentença do *if*. Poderia ser declarado do jeito normal, utilizando *type* e *var*.

15

ARQUIVOS EM DISCO

SUA UTILIDADE

A utilização de uma unidade de armazenamento em disco possibilita a preservação de dados mesmo após o desligamento do computador. As informações guardadas na memória RAM perdem-se ao desligar-se a corrente elétrica, enquanto no disco permanecem magneticamente preservadas. Além disso, a memória RAM não é suficientemente grande para a maior parte das aplicações sérias.

TIPOS DE ARQUIVOS

O Turbo Pascal possui três tipos de arquivos: seqüencial, randômico e de texto. Neste livro serão desenvolvidos apenas o arquivo randômico, que é o mais prático para utilização em grandes cadastros, e de texto.

O arquivo randômico, também conhecido como aleatório ou direto (para micros), possui as mesmas características nas duas linguagens, não havendo diferenças na sua funcionalidade.

Este capítulo será desenvolvido pelo ponto de vista prático de quem conhece BASIC, em vez do acadêmico utilizado geralmente em textos sobre Pascal.

REGISTRO

O registro é um conjunto de dados que pode ser formado por diversos tipos deles. Os dados que compõem o registro são conhecidos como campos.

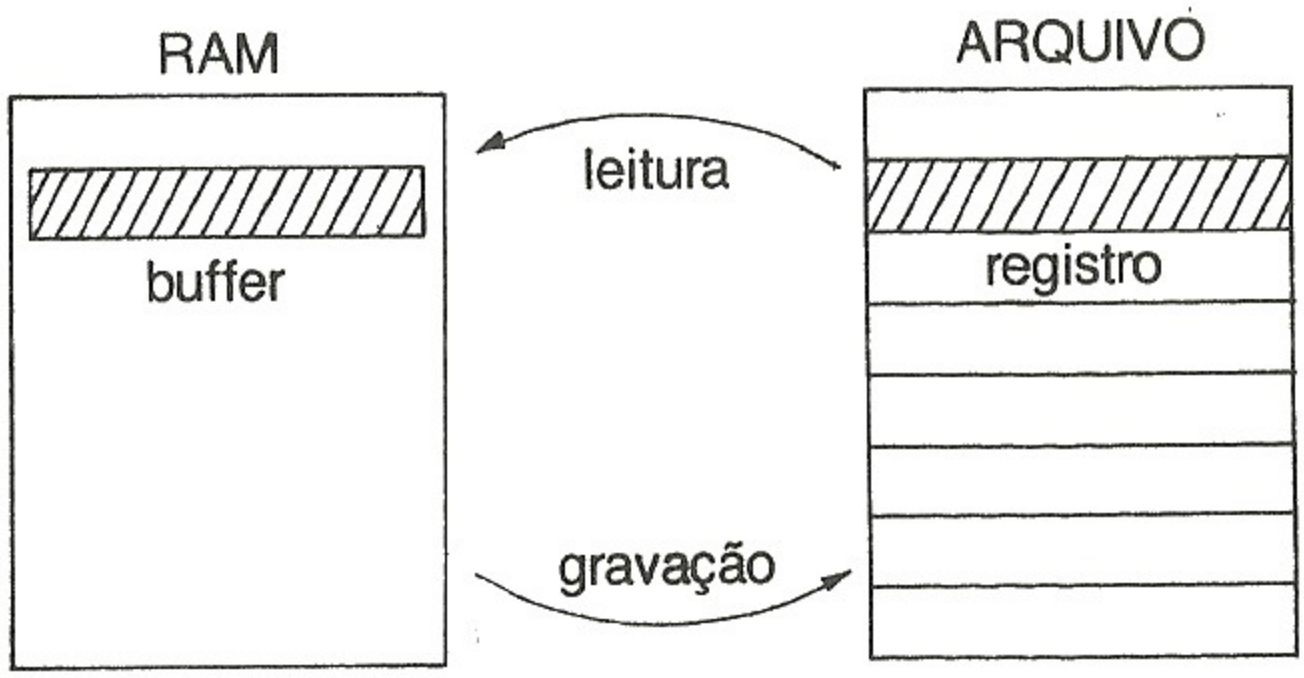
Um arquivo é formado por diversos registros. Cada registro é completamente independente do outro, mas a sua formação interna de campos é igual.

BUFFER

Buffer é uma região de memória que tem a mesma extensão e divisão em campos do registro associado a ele. Esta região serve de molde para se encaixar o registro.

O *buffer* serve de intermediário para que possamos utilizar mais eficientemente os arquivos em disco. Observe que:

1. Ao ler um registro, os dados serão trazidos para o *buffer*, tornando-se disponíveis para o processamento.
2. Para gravar um registro, devemos prepará-lo no *buffer* e depois mandar gravá-lo. Isto faz com que os dados que estavam no *buffer* sejam transferidos para o arquivo.



DECLARAÇÕES

Considere um arquivo de um cadastro de automóveis usados que estão numa agência. Este cadastro possui campos que indicam modelo, cor, ano de fabricação e quilometragem rodada.

Um registro utilizado no arquivo randômico deve ter a sua extensão física fixa, ou seja, cada campo deve possuir um número fixo de *bytes*. Devido a esta razão, precisamos definir a máxima extensão dos campos que irão compor este registro.

Modelo	-> 10 bytes
Cor	-> 10 bytes
Ano	-> 04 bytes
Quilometragem	-> 06 bytes

Em BASIC, um registro com estas características seria declarado assim:

```
FIELD #1,10 AS MODELO$,10 AS COR$,2
AS ANO$,4 AS QUILO$
```

Obserque que:

1. Para o campo ANO$ utilizou-se a compactação MKI$/CVI e para QUILO$, MKS$/CVS.
2. A definição #1 indica o canal e *buffer* a ser utilizado para leitura e gravação.

No Turbo Pascal a declaração seria:

```
type
    Carro = record
                Modelo : string[10];
                Cor    : string[10];
                Ano    : integer;
                Quilo  : real;
            end;

var
    Buffercarro : Carro;
```

Interpretação da declaração:

1. Os campos modelo, cor, ano e quilometragem formam um registro do tipo Carro.
2. Foi necessário criar e identificar um *buffer* para leitura e gravação. Este *buffer* chama-se Buffercarro e sua formatação (molde) é igual à do registro Carro.

Atenção: as declarações acima ainda não permitem o uso de arquivos. Apenas criou-se um *buffer* que pode ser utilizado normalmente como variáveis.

Vamos trabalhar com os campos de um registro sem, no entanto, gravar ou ler os dados. Uma variável de campo não pode ser diretamente referenciada. Para fazer uma referência precisamos colocar o nome do *buffer* seguido de um ponto (.) e o nome do campo.

```
Turbo Pascal

program Teste35;

type
    Carro = record
                Modelo : string[10];
                Cor    : string[10];
                Ano    : integer;
                Quilo  : real;
            end;
```

```
var
    Buffercarro : Carro;

begin
    write('Escreva um modelo de automovel ');
    readln(Buffercarro.Modelo);
    writeln(Buffercarro.Modelo);
end.
```

Vamos entender o programa:

1. O campo Buffercarro.Modelo foi utilizado para armazenar, mesmo não existindo operações com disco, o dado coletado no *readln*.
2. Mandou-se imprimir o conteúdo do campo Buffercarro.Modelo.

Esta necessidade de referenciar o nome do *buffer* e depois o campo propriamente dito provoca as seguintes conseqüências:

1. O Turbo Pascal admite mais de um campo com o mesmo nome desde que sejam de *buffers* diferentes.
2. Para não haver problemas de diferenciação, tornou-se necessária a colocação do nome do *buffer* correspondente antes do nome do campo.

COMANDOS

Um programa BASIC exige os seguintes comandos para utilizar os arquivos randômicos:

OPEN – Abre o arquivo e o canal.
FIELD – Define os campos e o *buffer*.
GET – Lê um registro.
PUT – Grava um registro.
LSET – Monta *buffer* alinhando à esqueda.
RSET – Monta *buffer* alinhando à direita.

O Turbo Pascal utiliza os seguintes comandos:

Assign – Associa o nome do arquivo utilizado no programa com o nome do arquivo a ser utilizado em disco.

assign(var. arquivo, arq. disco);

Rewrite – Cria um novo arquivo e deixa-o aberto. Deleta um arquivo que esteja eventualmente no mesmo disco e que possua o mesmo nome.

rewrite(var. arquivo);

Reset – Abre um arquivo já existente. Erro de I/O em caso de inexistência.

reset(var. arquivo);

Write – Grava o *buffer* no disco, no endereço apontado pelo *seek*.

write(var. arquivo, buffer);

Read – Lê o registro apontado pelo *seek* e o traz para o *buffer*.

read(var. arquivo, buffer);

Close – Esvazia o *buffer* e fecha o arquivo.

close(var. arquivo);

Seek – Procura determinado registro num arquivo.

seek(var. arquivo, número do registro);

Flush – Esvazia o *buffer*.

flush(var. arquivo);

Um arquivo deve ser declarado assim:

```
var
    [var. arquivo] : file of [var. de registro];
```

Vejamos um exemplo prático:

```
Turbo Pascal

program Teste36;

type
    Carro = record
                Modelo : string[10];
                Cor    : string[10];
                Ano    : integer;
                Quilo  : real;
            end;

var
    Buffercarro : Carro;
    Arqcarro    : file of Carro;
```

```
begin
    assign(Arqcarro,'CADASTRO.ARQ');
    rewrite(Arqcarro);
    seek(Arqcarro,0);
    Buffercarro.Modelo:='Escort';
    Buffercarro.Cor:='Azul';
    Buffercarro.Ano:=1987;
    Buffercarro.Guilo:=11000;
    write(Arqcarro,Buffercarro);
    close(Arqcarro);
end.
```

Após a definição de campos, registros, *buffers* e arquivos, utiliza-se o comando *assign* para associar o nome do arquivo usado internamente no programa (Arqcarro) ao nome do arquivo em disco (CADASTRO.ARQ). O programa anterior tem as seguintes características:

1. Cria e mantém aberto o arquivo (Arqcarro) através do comando *rewrite*.
2. Coloca o registro 0 sob a mira do apontador através do comando *seek*.
3. Faz as atribuições aos campos do *buffer* Buffercarro.
4. Grava no registro 0 do arquivo Arqcarro o conteúdo do Buffercarro.

O Turbo Pascal permite a gravação a partir do registro zero. Um registro não pode ser acessado no caso da inexistência do anterior, ou seja, o registro 2 somente pode ser acessado se já existir o registro 1. A única exceção é para o registro zero. O BASIC permite a gravação de qualquer registro.

O processo inverso de leitura é efetuado apontando o registro desejado através do *seek* e um comando *read*. Veja a utilização do *read* nos próximos exemplos.

With – As referências feitas aos campos de um registro tornam-se extensas, pois são compostos de [nome do *buffer*] . [nome do campo]. Utilizando o *with*, podemos abreviar e escrever somente o nome do campo. O trecho do programa em que esta abreviação é permitida deve estar limitado pelo:

```
with [nome do buffer] do
    begin
        .
        .
        .
    end;
```

Turbo Pascal

```
program Teste37;

type
    Carro = record
                Modelo : string[10];
                Cor    : string[10];
                Ano    : integer;
                Quilo  : real;
            end;

var
    Buffercarro : Carro;
    Arqcarro    : file of Carro;

begin
    assign(Arqcarro, 'CADASTRO.ARQ');
    reset(Arqcarro);
    seek(Arqcarro,0);
    read(Arqcarro,Buffercarro);
    with Buffercarro do
        begin
            writeln(Modelo);
            writeln(Cor);
            writeln(Ano);
            writeln(Quilo);
        end;
    close(Arqcarro);
end.
```

O programa anterior introduz o uso do *reset*, *with* e *read*:

1. Este programa lê os dados gravados pelo Teste36.
2. Como o arquivo já existia anteriormente, utilizou-se *reset* para abri-lo.
3. Apontou-se para o registro 0 através do *seek*.
4. Leu-se o registro número 0 utilizando *read*.
5. Devido à atuação do *with Buffercarro do*, foi possível a abreviatura das variáveis de campos (usou-se apenas Modelo, em vez de Buffercarro.Modelo).

A utilização de arquivos em disco é aquilo que mais se diferencia em relação ao BASIC. É difícil explicar as declarações de *type* e *var* utilizando conceitos já formados pelo BASIC. Portanto, a atenta leitura dos programas-exemplos deste capítulo torna-se extremamente necessária.

Trabalhar eficientemente com arquivos em disco é um tópico avançado, pois exige conhecimentos de *Sort* (ordenação), pesquisa binária, *hashing,* lista ligada e outros. Mas, mesmo para quem não deseja estudar a arte de programar em profundidade, criar um arquivo gerenciado através de pesquisa seqüencial é bem intuitivo e natural. Se houver a necessidade de utilização de uma estrutura de dados complexa, pode-se utilizar o pacote de programas que formam o DATABASE TOOLBOX. Para utilizar este TOOLBOX não é necessário conhecer nenhuma coisa sobre gerenciamento de arquivos. Basta apenas saber o que se quer e ler atentamente o manual.

16

COMPILAÇÃO DE PROGRAMAS

COMPILAÇÃO

O Turbo Pascal possui os procedimentos *execute* e *chain* que permitem que um programa chame outro. Nos sistemas de 16 *bits* há uma restrição de ser possível chamar apenas outros programas criados pelo Turbo Pascal, enquanto num de 8 *bits* é possível chamar qualquer programa do tipo comando (.COM).

O procedimento *execute* chama para execução qualquer programa compilado no modo COM, enquanto o procedimento *chain* executa programas compilados no formato CHN.

A sintaxe para *execute* e *chain*:

```
execute (var.arq);

chain (var.arq);
```

```
BASIC

1' programa Teste38
2'
3   PRINT "Chamando o programa CADASTRO.COM"
4   RUN "CADASTRO"
5   END
```

```
Turbo Pascal

program Teste38;

Var
    Proxprog : file;

begin
    writeln('Chamando o programa CADASTRO.COM');
    assign(Proxprog,'CADASTRO.COM');
    execute(Proxprog);
end.
```

OPÇÕES DE COMPILAÇÃO

Normalmente, a compilação é efetuada para memória, ou seja, após o término da compilação, o código-objeto permanece na memória pronto para execução. Este código-objeto pode ser salvo em forma de arquivo para posterior execução.

Digitando-se a letra *O* no *menu* principal (referente a *compiler option*), aparecerá o *menu* de opções do compilador.

```
compile -> Memory
           Com-file
           cHn-file

command line Parameters:

Find run-time error   Quit
```

Digitando-se a letra *M*, *C* ou *H*, escolhe-se a opção de compilação.

MODO COM

O código-objeto produzido pelo compilador é gravado no arquivo em disco juntamente com rotinas que dão suporte para execução deste código-objeto. Estas rotinas possuem a extensão aproximada de 10 *kbytes* e são necessárias para rodar um programa compilado a partir do sistema operacional.

MODO CHN

Um código-objeto do tipo CHN não possui dentro de si rotinas de apoio do Turbo Pascal. Estes devem ser chamados sempre através de outro programa Turbo Pascal. Isto garante que as rotinas permanecerão na memória para ser usadas pelo próximo programa.

A vantagem na utilização deste tipo de compilação é bastante simples. Num sistema de programas, apenas o primeiro a ser chamado precisa ser do tipo COM. Isto poupa bastante espaço em disco se levar em consideração que cada código-objeto COM leva consigo quase 10 *kbytes* de rotinas, enquanto o tipo CHN possui apenas o código-objeto referente ao programa.

CUIDADOS ESPECIAIS

O programa COM encadeado com os do tipo CHN deve possuir área de dados e códigos suficientemente grandes para comportá-los.

Sempre que um programa CHN é compilado, aparece uma mensagem indicando o tamanho do código e dos dados. Estes dados devem ser anotados para que na hora da compilação do programa COM sejam definidos parâmetros *minimum code segment size* e *minimum data segment size* com os maiores valores encontrados dentre todos os programas compilados do tipo CHN.

DIRETIVAS DE COMPILAÇÃO

O Turbo Pascal oferece diversas possibilidades de produção do código-objeto compilado de acordo com as necessidades do usuário. Estas instruções são dadas através de diretivas de compilação que devem aparecer no programa entre os caracteres que indicam comentários.

{$ } ou (*$ *)

As diretivas mais importantes são apresentadas a seguir. O valor *default* virá dentro do [ ] e + indica que é ativo e - passivo.

PARA TODOS OS SISTEMAS

C [C +] – Durante a execução do *read* e *write*, permite a atuação do CTRL-C e CTRL-S. Dependendo da versão para CP/M-80, CTRL-C funciona somente para *read*. Deve ser definida apenas uma vez e é válida globalmente.

I [I +] – O programa verifica erros de I/O. Em caso de passivo, a verificação deve ser efetuada através da função *IOresult*, que irá conter o código do erro.

R [R -] – Se ativo, faz verificação das faixas de valores dos índices de uma matriz ou escalar. Na forma passiva há ganhos de velocidade, mas pode-se perder o controle da situação. Utilizar + somente durante o desenvolvimento e - após a conclusão.

U [U-] – Se ativo, torna-se possível interromper a execução do programa em qualquer ponto, bastando apenas CTRL-C. Sacrifica o fator velocidade e deve ser utilizada somente durante o desenvolvimento do programa.

SOMENTE PARA CP/M-80

A [*A* +] – Recursividade não é admitida. No modo passivo é permitida, mas gasta mais memória e sacrifica velocidade. É possível redefinição durante o programa, ou seja, onde for necessário o uso da recursividade deve ser envolto com {$A-} E {$A+}.

W [*W2*] – Especifica a máxima profundidade (entrelaçamento) do *with* (entre 1 e 9).

X [*X* +] – Se ativo, otimiza a utilização de matrizes do ponto de vista da velocidade. Se passivo, otimiza-a do ponto de vista do espaço da memória.

SOMENTE PARA MS-DOS E CP/M-86

Fn [F16] – Número de arquivos possíveis de serem abertos simultaneamente é igual ao número *n*. Verificar também o arquivo CONFIG.SYS do sistema operacional.

17

EDIÇÃO DE DADOS

RECURSOS DE EDIÇÃO

A linguagem BASIC cónsegue formatar (editar) os dados a serem impressos na tela ou na impressora através do comando PRINT USING ou LPRINT USING.

O Turbo Pascal, através dos parâmetros dos procedimentos *write* e *writeln,* consegue formatar os seus dados, sendo que a versão TurboBCD possui um comando adicional com características avançadas de edição.

Char – Para imprimir variáveis ou constantes do tipo *char*, há duas maneiras de fazê-lo. Suponha uma variável do tipo *char*, *Ch*, sendo *n* um número inteiro:

```
write(Ch);

write(Ch:n);
```

A presença do parâmetro *n* faz com que o caractere seja impresso no lado direito de um campo de extensão *n*. Em outras palavras, o caractere será precedido de *n*-1 espaço em branco (aqui representado por _).

```
exemplo                    saida

write('T');                T
write('T':4)               ___T
```

String – A formatação para *strings* é muito parecida com a do tipo *char*. Suporta uma variável do tipo *string,* St, sendo *n* um número inteiro:

```
write(St);

write(St:n);
```

A presença do parâmetro *n* faz com que o *string* seja impresso no lado direito de um campo com a extensão *n*. Em outras palavras, o *string* será precedido de *n-lenght*(St) espaços.

exemplos	saídas
`write('Pascal');`	Pascal
`write('Pascal':10)`	.....Pascal

Boolean – A formatação com variáveis *booleanas* segue a mesma regra dos *strings*. Há apenas uma restrição quanto ao conteúdo que só pode ser FALSE ou TRUE.

Integer – A formatação para números inteiros pode ser de dois tipos. Suponha uma variável inteira *I* e um número inteiro *n*.

```
write(I);

write(I:n);
```

A presença do parâmetro *n* faz com que o número apresentado decimalmente seja impresso no lado direito do campo de extensão *n*. Se *n* for menor que a extensão do número, este será impresso normalmente, sendo, portanto, diferente do PRINT USING do BASIC que alerta o usuário.

exemplo	saída
`write(1234);`	1234
`write(1234:6);`	..1234

Real – A formatação de um número real pode ser de três tipos. Suponha uma variável *R* real, *n* inteira e *m* inteira 0 e 24.

```
write(R);

write(R:n);

write(R:n:m);
```

Sem parâmetros, o valor real será impresso em notação exponencial. A presença do valor *n* maior do que 18 faz com que o número real seja impresso no lado direito de um campo com a extensão *n*. Se *n* estiver entre 8 e 18, os algarismos à direita do ponto decimal serão omitidos para que formem um número de *n* algarismos. O parâmetro *m* indica a impressão do número real na sua forma decimal com *m* algarismos após o ponto.

exemplo	saída
write(123.456);	1.23456000000E+02
write(123.456:9);	1.235E+02
write(123.4567:6:2);	____123.46

TURBOBCD

O compilador TurboBCD possui uma função adicional chamada *form* para formatar as saídas. A sua utilização é praticamente igual ao PRINT USING e PRINT EDT$ presente em algumas versões do BASIC.

Utilizando o conceito de máscaras de edição, deve-se indicar o formato de impressão desejado.

Campo numérico deve ser editado usando os seguintes caracteres: '#', '@', '*', '---', '+', , , , "", "".

A função *form* segue as três regras abaixo:

1. O número é posicionado à direita do campo em caso de campo maior do que o número.
2. Os valores à direita do ponto decimal são arredondados se maiores do que especificado.
3. Se o número for maior do que especificado, são impressos somente asteriscos.

'#' – representa a posição do número e este caractere é igual ao do BASIC. Preenche com zeros à direita. Se negativo, retorna com o sinal logo à esquerda.

exemplo	saída
write(form('###') ,345.6) ;	346
write(form('#####.#',345.67) ;	_345.7
write(form('##.###',3.45) ;	_3.450
write(form('##.##',345.678) ;	**.**

'@' – mesmo funcionamento do #. A única diferença é que os campos à esquerda são preenchidos com zeros em vez de brancos. A presença de um único caractere @ misturado com # já produz efeitos da sua presença. Não imprime sinal.

exemplo	saída
write(form('@@@@.@',345.67)) ;	0345.7
write(form('@###.#',345.67)) ;	0345.7

'*' – mesmo funcionamento do @. A única diferença é que os campos à esquerda são preenchidos com asteriscos em vez de zeros.

exemplo	saída
write(form('****.*',345.67)) ;	*345.7
write(form('*###.#',345.67)) ;	*345.7

'$' – igual ao funcionamento do #. A única diferença é o aparecimento do caractere $ à esquerda do número. Basta uma ocorrência do caractere para fazer atuar.

exemplo	saída
write(form('$######.##',345.67)) ;	___$345.67

'–' – indica a posição para o sinal. Aparece somente quando negativo.

exemplo	saída
write(form('-###.##',-34.56)) ;	-_34.56

'+' – indica a posição para o sinal. Aparece para valores positivos e negativos.

```
write(form('+####.##',34.56));          +_34.50
```

'.' e ',' – utilizados para separar milhares ou decimais. São ótimos para formatar de acordo com o padrão de representação no Brasil, segundo o qual o ponto separa os milhares e a vírgula os decimais. O caractere que for utilizado por último (mais à direita) será considerado como o separador de decimais.

```
exemplos                                    saidas

write(form('#,###,###.##',4567.89));    ____4,567.89

write(form('##.###,#',4567.89));          __4.567,9
```

CAMPOS ALFANUMÉRICOS

Um campo alfanumérico deve ser editado com o uso de máscaras compostas de caracteres @ e #.

'#' – se o campo for maior que a extensão daquilo que se quer editar, virá alinhado ao lado esquerdo.

```
exemplo                                     saida

write(form('########','Turbo));          Turbo___
```

'@' – se o campo for maior que a extensão daquilo que se quer imprimir, virá alinhado ao lado direito. Basta a presença de apenas um caractere @.

```
exemplo                                     saida

write('form('@@@@@@@@','Turbo))          __Turbo

write(form('@########','Turbo))          ___Turbo
```

18

RECURSIVIDADE

O QUE É RECURSIVIDADE

Uma das principais características da linguagem Pascal é a possibilidade do utilizar recursividade nos seus programas. A recursividade consiste no seguinte: um procedimento pode chamar a si mesmo para execução.

Como não poderia deixar de ser, o Turbo Pascal incorpora esta elegante solução a diversos problemas em técnicas de programação. Um exemplo de aplicação prática desta técnica é encontrado para resolver as Torres de Hanói.

Deixando de lado as complexidades, vamos para uma aplicação simples e ilustrativa da recursividade.

Fatorial

O fatorial é uma operação matemática representada por (!) e é calculado assim:

```
5! = 5 x 4 x 3 x 2 x 1 = 120
3! = 3 x 2 x 1 = 6
7! = 7 x 6 x 5 x 4 x 3 x 2 x 1 = 5040
```

Usando fórmulas, o fatorial é representado de tal forma que:

```
n! = n x (n-1) x (n-2) x...x 2 x 1
```

em que *n* é um valor inteiro maior do que zero.

Define-se o fatorial de zero como sendo 1 (0! = 1).

A fórmula anterior ganha outro aspecto se apresentada no modo recursivo:

```
n!  =  1                    (para n=0)        (I)

n!  =  n  x  (n-1)!  (para n>=0)       (II)
```

Vamos acompanhar um exemplo de cálculo do fatorial utilizando o método recursivo:

Calcule 4!

1. Sendo n = 4, aplicar (II).
   4! = 4 x (4 – 1)! = 4 x 3!

2. Como não se conhece o valor de 3!, aplicar novamente (II).
   4! = 4 x 3 x (3 – 1)! = 12 x 2!

3. Como não se conhece o valor de 2!, aplicar novamente (II).
   4! = 12 x 2 x (2 – 1)! = 24 x 1!

4. Como não se conhece o valor de 1!, aplicar novamente (II).
   4! = 24 x 1 x (1 – 1)! = 24 x 0!

5. O fatorial de zero é igual a 1, pela definição (I); portanto:
   4! = 24

As cinco etapas anteriores demonstraram que a aplicação recursiva da fórmula *n! = n* x *(n – 1)!* permitiu encontrar o valor de 4! partindo da definição 0! = 1.

O cálculo do fatorial utilizando a linguagem BASIC não pode adotar a fórmula recursiva. Veja o exemplo:

BASIC

```
1 ' programa Teste39
2 '
3   INPUT "Entre com um numero";NUM
4   FAT=1
5   FOR J=NUM TO 1 STEP -1
6   FAT=FAT*J
7   NEXT J
8   PRINT "Fatorial =";FAT
9   END
```

A tradução direta para Turbo Pascal seria:

```
Turbo Pascal

program Teste39;

var
    J,Fat,Num : integer;

begin
    readln(Num);
    Fat:=1;
    for J:=Num downto 1 do
        begin
            Fat:=Fat*J;
        end;
    writeln('Fatorial = ',Fat);
end.
```

A versão em Turbo Pascal utilizando a fórmula recursiva seria:

```
program Teste40;

var
    J : integer;

(*-------------------------------------------------------------------------*)

function Fatorial(n:integer) : integer;

begin
    if n=0 then
        begin
            Fatorial:=1;
        end
    else
        begin
            Fatorial:=n*Fatorial(n-1);
        end;
end;

(*-----------------------programa principal-------------------------------*)
```

```
begin
    write('Entre com um número');
    readln(J);
    write(Fatorial(J));
end.
```

A utilização dos procedimentos recursivos não é tão complexa como à primeira vista, mas exige cuidados do ponto de vista da lógica de programação.

Observações

A versão do Turbo Pascal para oito *bits* exige a inclusão da diretiva de compilação {$A+} que libera o uso deste recurso. A diretiva pode estar restrita ao bloco em que é exigida a recursividade, ou seja, o procedimento pode ser envolvido por {$A–} E {$A+}.

Nos sistemas de oito *bits*, uma função ou procedimento não deve ser referenciada diretamente dentro de outro procedimento do tipo *write*, tal como foi feito no exemplo.

19

ARQUIVOS DE TEXTO

O QUE SÃO ARQUIVOS DE TEXTO

Um arquivo de texto consiste em um conjunto de caracteres ASCII que podem ser lidos diretamente, sem que haja algum processo de conversão. Em outras palavras, basta executar o comando *type* do MS-DOS no arquivo para poder ler o conteúdo diretamente no display. Como exemplo, podem-se citar os programas escritos pelo editor do Turbo Pascal, um texto qualquer do WordStar criado no modo não documento ou algo editado pelo EDLIN que integra o MS-DOS.

UTILIDADE

Um arquivo de texto pode ser utilizado para guardar certas informações cujo acesso não é prioritário. Ele exige leitura seqüencial e pode não ser muito interessante para guardar grande volume de dados.

Através deste tipo de arquivo, pode-se criar, por exemplo, um *batch* para ser utilizado pelo sistema operacional, um simples texto no modo seqüencial ou até mesmo comunicação com outro programa.

DECLARAÇÕES

Tal como na utilização de arquivos randômicos, precisa-se declarar uma variável de arquivo:

```
[var] arquivo : text;
```

O tipo *text* já está predefinido no Turbo Pascal, facilitando o manuseio pelo usuário.

COMANDOS

Os comandos para abrir e fechar os arquivos do tipo texto são os mesmos do arquivo randômico:

```
assign          flush          reset
rewrite         close
```

A utilização dos comandos deve ser vista no Capítulo 15.

Para leitura e gravação de dados, há diversas combinações possíveis utilizando *write, writeln, read e readln.* Neste livro, interessa desenvolver apenas a utilização do *readln e writeln.*

Todo *writeln* executado, tendo como primeiro parâmetro a variável de arquivo definida como *text,* será direcionado para o disco:

```
writeln(var. arquivo, ...
```

Estes dados relacionados no lugar do (. . .) poderão ser lidos mais tarde utilizando o comando de leitura *readln.*

```
readln(var, arquivo, ...
```

É uma boa política utilizar o *writeln e readln* com os mesmos parâmetros e variáveis para evitar confusões.

Vejamos um pequeno exemplo:

```
Turbo Pascal

program Teste41;

var
   Arqamigo : text;
   Nome : string[30];

begin
   clrscr;
   assign(Arqamigo,'AMIGO.ARQ');
   rewrte(Arqamigo);
   writeln('Digite o nome de um amigo');
   readln(Nome);
   writeln(Arqamigo,Nome);
   close(Arqamigo);
end.
```

O programa anterior grava o texto que foi digitado num arquivo em disco denominado AMIGO.ARQ. Para recuperar o texto gravado, veja o próximo exemplo:

```
Turbo Pascal

program Teste42;

var
   Arqamigo : text;
   Nome : string[30];

begin
   clrscr;
   assign(Arqamigo,'AMIGO.ARQ');
   reset(Arqamigo);
   writeln('O nome lido é : ');
   readln(Arqamigo,Nome);
   writeln(Nome);
   close(Arqamigo);
end.
```

O leitor pode tentar ler outros arquivos, inclusive aqueles gerados por diferentes linguagens e aplicativos. Tenho certeza de que irá encontrar surpresas agradáveis.

20

IMPRESSORAS

COMPATIBILIDADE

Começa-se este capítulo com uma má notícia. Antigamente, cada fabricante de impressora criava o seu próprio sistema de codificação das funções especiais possíveis de serem obtidas, tais como caracteres em negrito, comprimidos ou expandidos. De comum entre os modelos disponíveis eram apenas os caracteres padrões ASCII entre 32 e 128. Alguns anos atrás, um acordo entre algumas das mais importantes empresas do ramo determinou que os seus produtos iriam passar a ser compatíveis com a codificação em uso pela EPSON (o mais bem-sucedido entre as empresas).

Após este acordo, as incompatibilidades diminuíram sensivelmente, mas de vez em quando depara-se com pequena parcela de impressoras que estão fora deste padrão. Estas impressoras são geralmente de tecnologia procedente da Europa.

Por todos estes problemas, os comandos aqui abordados podem funcionar parcialmente ou totalmente na impressora usada pelo leitor. Como a incompatibilidade aumenta proporcionalmente a complexidade dos comandos, serão tratados apenas aqueles que são mais simples. Os demais devem ser analisados no próprio manual do usuário da impressora.

COMO ENVIAR OS COMANDOS

Para enviar os códigos especiais que vão comandar a impressora, basta utilizar *write* com o parâmetro *Lst*.

Qualidade carta

```
write(Lst,chr(27),chr(120),chr(1));     Ativa
write(Lst,chr(27),chr(120),chr(0));     Desativa
```

Caractere comprimido

```
write(Lst,chr(27),chr(15));             Ativa
write(Lst,chr(27),chr(18));             Desativa
```

Caractere expandido

```
write(Lst,chr(27),chr(87),chr(1));      Ativa
write(Lst,chr(27),chr(87),chr(0));      Desativa
```

Modo enfatizado

```
write(Lst,chr(27),chr(69));             Ativa
write(Lst,chr(27),chr(70));             Desativa
```

Modo duplo impacto

```
write(Lst,chr(27),chr(71));             Ativa
write(Lst,chr(27),chr(72));             Desativa
```

Caractere itálico

```
write(Lst,chr(27),chr(52));             Ativa
write(Lst,chr(27),chr(53));             Desativa
```

Reset geral (anula todos os outros códigos)

```
write(Lst,chr(27),chr(64));
```

Pula para o início da próxima folha

```
write(Lst,chr(12));
```

Retrocede uma coluna

```
write(Lst,chr(8));
```

Sensor de papel (inclusive o *beep*)

```
write(Lst,chr(27),chr(56));                    Desativa
write(Lst,chr(27),chr(57));                    Ativa
```

Pula uma linha

```
write(Lst,chr(10));  ou
write(Lst,chr(13),chr(10));
```

Beep na impressora

```
write(Lst,chr(7));
```

Modo sublinhado

```
write(Lst,chr(27),chr(45),chr(0));         Desativa
write(Lst,chr(27),chr(45),chr(1));         Ativa
```

Impressão unidirecional

```
write(Lst,chr(27),chr(85),chr(0));         Desativa
write(Lst,chr(27),chr(85),chr(1));         Ativa
```

Impressão a baixa velocidade

```
write(Lst,chr(27),chr(115),chr(0));        Desativa
write(Lst,chr(27),chr(115),chr(1));        Ativa
```

Espaçamento entre linhas de 1/8 de polegada

```
write(Lst,chr(27),chr(48));
```

Espaçamento entre linhas de 1/6 de polegada (default)

```
write(Lst,chr(27),chr(50));
```

Salta o picote do formulário

```
write(Lst,chr(27),chr(79));                    Desativa
write(Lst,chr(27),chr(78));                    Ativa
```

Observação: A grande maioria dos comandos expostos aqui pode ser utilizada simultaneamente sem problemas de sobreposição. Por exemplo: duplo impacto + itálico + salto do picote. O que não é possível fazer é algo do tipo 1/6 polegada + 1/8 polegada.

Código	Explicação
31	Simple expression expected Falta expressão simples
32	String constant expected Falta constante string
33	String expression expected Falta expressão string
34	String variable expected Falta variável string
35	Textfile expected Falta textfile
36	Type identifier expected Falta identificador de type
37	Untyped file expected Falta arquivo sem tipo
40	Undefined label Rótulo não definido. A sentença faz referência a um rótulo não definido.
41	Unknown identifier or syntax error Identificador desconhecido ou erro de sintaxe
42	Undefined pointer type in preceding type definitions Na definição de tipos há ocorrência de pointer sem definição.
43	Duplicate identifier or label Dupla utilização de rótulo ou identificador
44	Type mismatch Incompatibilidade de tipos
45	Constant out of range Constante fora do intervalo permitido
46	Constant and CASE selector type does not match Incompatibilidade de tipo entre o seletor do CASE e da constante
47	Operand type(s) does not match operator Incompatibilidade entre operador e operando
48	Invalid result type Tipo do retorno inválido
49	Invalid string length Comprimento do string inválido

Código	Explicação
50	String constant length does not match type Incompatibilidade de tamanhos entre os strings
51	Invalid subrange base type Intervalo do tipo básico é inválido
52	Lower bound > upper bound Limite superior é menor do que o limite inferior
53	Reserved word Palavra reservada. Utilização da palavra reservada
54	Illegal assignment Atribuição ilegal
55	String constant exceeds line Constante string excede a linha
56	Error in integer constant Erro numa constante inteira. Valor fora do intervalo permitido. Os números reais devem ser usados distintamente em relação aos inteiros, isto é, 1234.0 para representar 1234
57	Error in real constant Erro numa constante real
58	Illegal character in identifier Caractere ilegal dentro do identificador
60	Constant not allowed here Constantes não são permitidas aqui
61	Files and pointers are not allowed here Arquivos e apontadores não são permitidos aqui
62	Structed variables are not allowed here Variáveis estruturadas não são permitidas aqui
63	Textfiles are not allowed here Arquivos-texto não são permitidos aqui
64	Textfiles and untyped files are not allowed here Arquivos-texto e arquivos sem tipo não são permitidos aqui
65	Untyped files are not allowed here Arquivos sem tipo não são permitidos aqui
66	I/O not allowed here I/O não é permitido aqui
67	Files must be var parameters Arquivos devem ser do tipo parâmetro var

Código	Explicação
68	File components may not be files Componentes de arquivos não podem ser arquivos
69	Invalid ordering of fields Ordem dos campos inválida
70	Set base type out of range Tipo básico deve ter no máximo 256 valores possíveis
71	Invalid goto Goto inválido
72	Label not within current block O rótulo não está dentro do bloco corrente
73	Undefined FORWARD procedures Inexistência do bloco declarado através do FORWARD
74	Inline error Erro no inline
75	Illegal use of ABSOLUTE Uso ilegal do ABSOLUTE
76	Overlays can not be forwarded FORWARD não deve ser usado com overlays
77	Overlays not allowed in direct mode Overlays não são permitidos no modo direto
90	File not found Arquivo não encontrado
91	Unexpected end of source Fim inesperado do programa
92	Unable to create overlay file Impossível criar arquivo overlay
93	Invalid compile directive Diretiva de compilação inválida
97	Too many nested withs Excesso no nível de withs
98	Memory overflow Estourou a memória
99	Compiler overflow Estourou a memória durante compilação

ERROS DE I/O

Código	Explicação
01	File does not exist Arquivo inexistente
02	File not open for input Arquivo para entrada fechado
03	File not open for output Arquivo para saída fechado
04	File not open Arquivo fechado
10	Error in numeric format Formato numérico impróprio
20	Operation not allowed on a logical device Operação não permitida no dispositivo lógico
21	Not allowed in direct mode Não permitido no modo direto
22	Assign to std files not allowed Não é permitido usar ASSIGN para arquivos-padrão
90	Record length mismatch Incompatibilidade de tamanho do registro
91	Seek beyond end-of-file Tentativa de dar seek após o fim de arquivo
99	Unexpected end-of-file Fim de arquivo não esperado
F0	Disk write error Disco cheio
F1	Directory is full Diretório cheio
F2	File size overflow Tentativa de usar além do registro 65535
F3	Too many files opens Excesso de arquivos abertos
FF	File disappeared Tentativa de dar close num arquivo inexistente no disco

Erros de execução

Código	Explicação
01	Floating point overflow Estorou o valor em ponto flutuante
02	Division by zero attempeted Tentativa de divisão por zero
03	Sqrt argument error Tentativa de extrair raiz quadrada de um número negativo
04	Ln argument error Ln com argumento zero ou negativo
10	String length error String com mais de 255 caracteres Conversão de string com mais de um caractere para char
11	Invalid string index Índice referente a operações com string fora do 1..255
90	Index out of range Índice da matriz fora do intervalo determinado
91	Scalar or subrange out of range Escalar ou subintervalo fora do intervalo determinado
92	Out of integer range Número inteiro fora do intervalo entre -32768 e 32767
10	Overlay file not found O arquivo overlay não foi encontrado
11	Heap/stack collision Colisão entre heap e stack

APÊNDICE – 2
CONVERSÕES

Utilizando as funções já existentes no Turbo Pascal, podemos facilmente obter outras que serão muito úteis em aplicações técnicas. As funções trigonométricas estão em inglês para manter coerência entre a função e o seu equivalente em Turbo Pascal.

Função	**Equivalente**
Tan	Sin(x)/Cos(x)
Sec	1/Cos(x)
Cosec	1/Sin(x)
Cot	Cos(x)/Sin(x)
ArcSin	ArcTan(x/Sqrt(-x*x+1))
ArcCos	-ArcTan(x/Sqrt(-x*x+1))+Pi/2
ArcCot	-ArcTan(x)+Pi/2
Sinh	(Exp(x)-Exp(-x))/2
Cosh	(Exp(x)+Exp(-x))/2
Tanh	-Exp(-x)/(Exp(x)+Exp(-x))*2+1
Sech	2/(Exp(x)+Exp(-x))
Cosech	2/(Exp(x)-Exp(-x))
Coth	Exp(-x)/(Exp(x)-Exp(-x))*2+1
ArcSinh	Ln(x+Sqrt(x*x+1))
ArcCosh	Ln(x+Sqrt(x*x-1))
ArcTanh	Ln((1+x)/(1-x))/2
ArcCoth	Ln((x+1)/(x-1))/2
$Log_a{}^b$	Ln(b)/Ln(a)
a^b	Exp(b*Ln(x))

APÊNDICE – 3
TABELA ASCII DO PC

Decimal	Hex	Caractere	ASCII
0	00	**NUL**	null
1	01	**SOH**	start of heading
2	02	**STX**	start of text
3	03	**ETX**	end of text
4	04	**EOT**	end of transmission
5	05	**ENQ**	enquiry
6	06	**ACK**	acknowledge
7	07	**BEL**	bell
8	08	**BS**	backspace
9	09	**HT**	tab horizontally
10	0A	**LF**	line feed
11	0B	**VT**	tab vertically
12	0C	**FF**	form feed
13	0D	**CR**	carriage return
14	0E	**SO**	shift out
15	0F	**SI**	shift in
16	10	**DLE**	data link escape
17	11	**DC1**	device control 1
18	12	**DC2**	device control 2
19	13	**DC3**	device control 3
20	14	**DC4**	device control 4
21	15	**NAK**	negative acknowledge
22	16	**SYN**	synchronous idle
23	17	**ETB**	end of transmitted block
24	18	**CAN**	cancel line
25	19	**EM**	end of medium
26	1A	**SUB**	substitute
27	1B	**ESC**	escape
28	1C	**FS**	file separator
29	1D	**GS**	group separator
30	1E	**RS**	record separator
31	1F	**US**	unit separator

(Continuação)

Dec	Hex	Caractere
32	20	SP
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Caractere
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

(Continuação)

Dec	Hex	Caractere
96	60	
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	¦
125	7D	}
126	7E	~
127	7F	**DEL**

Dec	Hex	Caractere
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

(Continuação)

Dec	Hex	Caractere
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Caractere
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	▉
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

(Continuação)

Dec	Hex	Caractere
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	ø
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

BIBLIOGRAFIA

BORLAND. *Turbo Pascal 3.0 reference manual*. Borland International.

BORLAND. *Turbo tutor manual*. Borland International.

ISHIDA, Toshinobu & TAUSCHER, Tomaz. *Técnicas avançadas de computação*. São Paulo, Ícone, 1985 e 1986. V. I e II.

ITO. *Understanding Turbo Pascal*. SST, 1986.

KNUTH, D. E. *The art of computer programming*. Cambridge, Addison & Wesley.

KOBAYASHI. *Turbo Pascal quick manual*. JICC Editor, 1986.

MICROSOFT. *BASIC-80 reference manual*. Microsoft.

ROSENFELDER, Lewis. *BASIC faster and better*. Radio Shack, 1983.

WOOD, Steve. *Using Turbo Pascal*. New York, McGraw-Hill, 1986.

TURBO PASCAL

Para programadores BASIC

A literatura disponível sobre Turbo Pascal é ainda escassa no Brasil. Entre os livros existentes, há os manuais do usuário (apresentação de comandos) e aqueles que ensinam a programação com a utilização de Pascal.

Este livro destina-se àqueles que já possuem alguma noção sobre programação BASIC e desejam iniciar-se em Pascal. Objetiva desenvolver o Turbo Pascal, fazer comparações com o BASIC, evitando, assim, ser apenas mais um dicionário de comandos ou insistir no funcionamento de comandos básicos, que, para quem tem noção sobre programação, acaba tornando-se cansativo.

Os assuntos abordados são exemplificados através de programas-exemplo em BASIC e Pascal, principalmente nos primeiros capítulos, que mostram que a semelhança entre as duas linguagens é grande.

O BASIC utilizado para desenvolver este livro é o padrão Microsoft (IBM-PC/XT/AT, MSX, CP/M-80, TRS-80 I/III/Color).

O Turbo Pascal é disponível para MS-DOS, CP/M-80/86 e Mac.

NOTA SOBRE O AUTOR

TOSHINOBU ISHIDA está concluindo curso de graduação em Engenharia de Produção da Escola Politécnica da USP. Foi professor das disciplinas Linguagens de Programação e Técnicas de Processamento de Dados. Participou de diversos sistemas dirigidos à área comercial, industrial e econômica utilizando BASIC e Pascal. Atualmente é sócio da IT Systems Informática.

APLICAÇÃO

Indicado como introdução ao Turbo Pascal para os que têm noções do BASIC, permitindo aprendizado rápido através de paralelos desenvolvidos entre as duas linguagens.

publicação atlas

ISBN 85-224-0288-4