

*ASSOCIAÇÃO PARAIBANA DE ENSINO RENOVADO
FACULDADE PARAIBANA DE PROCESSAMENTO DE DADOS*

LINGUAGENS E TÉCNICAS DE PROGRAMAÇÃO I



***CURSO DE TECNOLOGIA EM
PROCESSAMENTO DE DADOS***

1998

Copyright © 1998 de Cândido J. R. Egypto

Este material foi elaborado para ser utilizado pelos alunos da disciplina Linguagens e Técnicas de Programação I, do Curso de Tecnologia em Processamento de Dados, da Faculdade Paraibana de Processamento de Dados.

Nenhuma parte deste material pode ser reproduzida ou transmitida de qualquer modo ou por qualquer meio, sem prévia autorização do autor e sem lhe ser dado o devido crédito.

SUMÁRIO

Capítulo 1 - ALGORITMOS	5
1.1. CONCEITO	5
1.2. POR QUE PRECISAMOS DE ALGORITMOS ?	5
1.3. CARACTERÍSTICAS	5
1.4. FORMAS DE REPRESENTAÇÃO	6
1.4.1. DESCRIÇÃO NARRATIVA	6
1.4.2. FLUXOGRAMA	6
1.4.3. LINGUAGEM ALGORÍTMICA	7
1.5. UM AMBIENTE PARA ESCREVER ALGORITMOS	7
1.5.1. FUNCIONAMENTO DO NOSSO COMPUTADOR	8
1.6. ESTRUTURAS CHAVES DA CONSTRUÇÃO DE ALGORITMOS	9
1.6.1. SEQUENCIAÇÃO	9
1.6.2. DECISÃO OU SELEÇÃO	9
1.6.3. REPETIÇÃO OU ITERAÇÃO	9
1.7. REFINAMENTOS SUCESSIVOS	10
Capítulo 2 - LINGUAGEM ALGORÍTMICA	12
2.1. CONCEITO DE VARIÁVEL	12
2.2. OPERAÇÃO DE ATRIBUIÇÃO	12
2.3. OPERAÇÕES DE ENTRADA E SAÍDA	12
2.4. ESTRUTURA SEQUENCIAL	13
2.5. ESTRUTURA CONDICIONAL	14
2.5.1. ESTRUTURA CONDICIONAL SIMPLES	14
2.5.2. ESTRUTURA CONDICIONAL COMPOSTA	15
2.6. ESTRUTURA DE REPETIÇÃO	16
Capítulo 3 - LINGUAGEM DE PROGRAMAÇÃO PASCAL	19
3.1. INTRODUÇÃO	19
3.1.1. LINGUAGENS DE PROGRAMAÇÃO	19
3.1.2. TRADUTORES	19
3.1.3. A LINGUAGEM PASCAL	20
3.2. ELEMENTOS BÁSICOS	20
3.2.1. IDENTIFICADORES	20
3.2.2. PALAVRAS RESERVADAS	20
3.3. TIPOS DE DADOS	21
3.3.1. SIMPLES	21
3.3.2. ESTRUTURADOS	21
3.3.3. DEFINIDOS PELO USUÁRIO	21
3.4. EXPRESSÕES ARITMÉTICAS	23
3.4.1. OPERADORES ARITMÉTICOS	23
3.4.2. PRIORIDADE	23
3.4.3. FUNÇÕES E PROCEDIMENTOS NUMÉRICOS PREDEFINIDOS	23
3.5. EXPRESSÕES LÓGICAS	24
3.5.1. OPERADORES RELACIONAIS	24
3.5.2. OPERADORES LÓGICOS	24
3.5.3. PRIORIDADE	25
3.6. FORMATO DE UM PROGRAMA PASCAL	26
3.6.1. DECLARAÇÃO DE USO DE UNIDADES	26
3.6.2. DECLARAÇÃO DE CONSTANTES	27
3.6.3. DECLARAÇÃO DE TIPOS	27
3.6.4. DECLARAÇÃO DE VARIÁVEIS	27
3.6.5. DECLARAÇÃO DE PROCEDIMENTOS E FUNÇÕES	27
3.6.6. ÁREA DE COMANDOS	27
3.7. COMENTÁRIOS	28

Capítulo 4 - COMANDOS BÁSICOS DA LINGUAGEM PASCAL 29

4.1. ATRIBUIÇÃO.....29
4.2. ENTRADA29
4.3. SAÍDA.....30
4.4. COMANDOS DE DECISÃO31
4.4.1. DECISÃO SIMPLES (IF-THEN).....31
4.4.2. DECISÃO COMPOSTA (IF-THEN-ELSE)32
4.4.3. DECISÃO MÚLTIPLA (CASE-OF)33
4.5. COMANDOS DE REPETIÇÃO.....36
4.5.1. REPETIÇÃO COM TESTE NO INÍCIO (WHILE-DO).....36
4.5.2. REPETIÇÃO COM TESTE NO FINAL (REPEAT-UNTIL)37
4.5.3. REPETIÇÃO AUTOMÁTICA (FOR).....38

Capítulo 5 - ARRAYS 41

5.1. VETORES41
5.2. MATRIZES45
5.3. ARRAYS MULTIDIMENSIONAIS.....47

Capítulo 6 - MODULARIZAÇÃO 50

6.1. PROCEDIMENTO.....50
6.2. FUNÇÃO.....51
6.3. VARIÁVEIS GLOBAIS E VARIÁVEIS LOCAIS52
6.4. PARÂMETROS.....53
6.5. UTILIZANDO ARRAYS COMO PARÂMETROS57
6.6. RECURSIVIDADE.....59
6.7. CRIAÇÃO DE UNITS64
6.7.1. ESTRUTURA DE UMA UNIT.....64
6.7.2. UTILIZAÇÃO DE UNITS.....65

Capítulo 7 - MANIPULAÇÃO DE STRINGS..... 66

7.1. O TIPO DE DADO STRING.....66
7.2. USANDO STRINGS COMO PARÂMETROS EM SUBROTINAS.....66
7.3. FUNÇÕES E PROCEDIMENTO PREDEFINIDOS66
7.4. CONTROLE DO VÍDEO E DO TECLADO.....71

Capítulo 8 - ARQUIVOS E REGISTROS 75

8.1. REGISTROS.....75
8.1.1. DECLARAÇÃO.....75
8.1.2. REFERÊNCIA75
8.1.3. CONJUNTO DE REGISTROS76
8.1.4. O COMANDO WITH.....77
8.2. ARQUIVOS.....79
8.2.1. DECLARAÇÃO DE ARQUIVOS79
8.2.2. UTILIZAÇÃO DE ARQUIVOS80

Capítulo 9 - CLASSIFICAÇÃO E PESQUISA 90

9.1. CLASSIFICAÇÃO.....90
9.1.1. MÉTODO DA BOLHA90
9.1.2. CLASSIFICAÇÃO POR INSERÇÃO.....91
9.1.3. CLASSIFICAÇÃO POR SELEÇÃO.....92
9.2. COMPARAÇÃO DOS MÉTODOS DE CLASSIFICAÇÃO APRESENTADOS.....93
9.3. PESQUISA95
9.3.1. PESQUISA SEQUENCIAL.....95
9.3.2. PESQUISA SEQUENCIAL ORDENADA.....95
9.3.3. PESQUISA BINÁRIA.....96
9.4. COMPARAÇÃO DOS MÉTODOS DE BUSCA APRESENTADOS.....97

Apêndice - GLOSSÁRIO 98

BIBLIOGRAFIA..... 101

Capítulo 1

ALGORITMOS

1.1. CONCEITO

A palavra **algoritmo**, à primeira vista, parece-nos estranha. Embora possua designação desconhecida, fazemos uso constantemente de algoritmos em nosso cotidiano: a maneira como uma pessoa toma banho é um algoritmo. Outros algoritmos freqüentemente encontrados são:

- instruções para se utilizar um aparelho eletrodoméstico;
- uma receita para preparo de algum prato;
- guia de preenchimento para declaração do imposto de renda;
- a regra para determinação de máximos e mínimos de funções por derivadas sucessivas;
- a maneira como as contas de água, luz e telefone são calculadas mensalmente; etc.

São vários os conceitos para algoritmo. Escolhemos alguns para serem apresentados aqui:

“Um conjunto finito de regras que provê uma seqüência de operações para resolver um tipo de problema específico”
[KNUTH]

“Seqüência ordenada, e não ambígua, de passos que levam à solução de um dado problema”
[TREMBLAY]

“Processo de cálculo, ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, as regras formais para a obtenção do resultado ou da solução do problema”
[AURÉLIO]

1.2. POR QUE PRECISAMOS DE ALGORITMOS ?

Vejamos o que algumas pessoas importantes, para a Ciência da Computação, disseram a respeito de algoritmo:

“A noção de algoritmo é básica para toda a programação de computadores”.
[KNUTH - Professor da Universidade de Stanford, autor da coleção “The art of computer programming”]

“O conceito central da programação e da ciência da computação é o conceito de algoritmo”.
[WIRTH - Professor da Universidade de Zurique, autor de diversos livros na área e responsável pela criação de linguagens de programação como ALGOL, PASCAL e MODULA-2]

A importância do algoritmo está no fato de termos que especificar uma seqüência de passos lógicos para que o computador possa executar uma tarefa qualquer, pois o mesmo por si só não tem vontade própria, faz apenas o que mandamos. Com uma ferramenta algorítmica, podemos conceber uma solução para um dado problema, independentemente de uma linguagem específica e até mesmo do próprio computador.

1.3. CARACTERÍSTICAS

Todo algoritmo deve apresentar algumas características básicas:

- ter fim;
- não dar margem à dupla interpretação (não ambíguo);
- capacidade de receber dado(s) de entrada do mundo exterior;
- poder gerar informações de saída para o mundo externo ao do ambiente do algoritmo;
- ser efetivo (todas as etapas especificadas no algoritmo devem ser alcançáveis em um tempo finito).

1.4. FORMAS DE REPRESENTAÇÃO

Algoritmos podem ser representados, dentre outras maneiras, por:

1.4.1. DESCRIÇÃO NARRATIVA

Faz-se uso do português para descrever algoritmos.

EXEMPLO: Receita de Bolo:

Providencie manteiga, ovos, 2 Kg de massa, etc.
 Misture os ingredientes
 Despeje a mistura na fôrma de bolo
 Leve a fôrma ao forno
 Espere 20 minutos
 Retire a fôrma do forno
 Deixe esfriar
 Prove

VANTAGENS:

- o português é bastante conhecido por nós;

DESVANTAGENS:

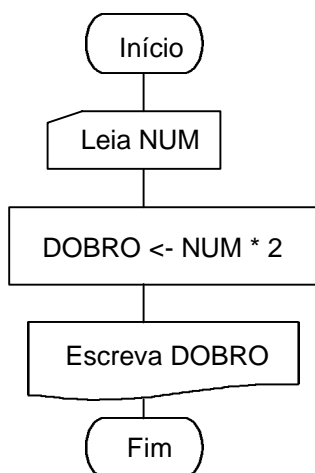
- imprecisão;
- pouca confiabilidade (a imprecisão acarreta a desconfiança);
- extensão (normalmente, escreve-se muito para dizer pouca coisa).

1.4.2. FLUXOGRAMA

Utilização de símbolos gráficos para representar algoritmos. No fluxograma existem símbolos padronizados para início, entrada de dados, cálculos, saída de dados, fim, etc.



EXEMPLO



EXPLICAÇÃO

Início do algoritmo
 Entrada do número
 Cálculo do dobro do número
 Apresentação do resultado
 Fim do algoritmo

VANTAGENS:

- Uma das ferramentas mais conhecidas;
- Figuras dizem muito mais que palavras;
- Padrão mundial

DESVANTAGENS:

- Faz com que a solução do problema já esteja amarrada a dispositivos físicos;
- Pouca atenção aos dados, não oferecendo recursos para descrevê-los ou representá-los;
- Complica-se à medida que o algoritmo cresce.

1.4.3. LINGUAGEM ALGORÍTMICA

Consiste na definição de uma pseudolinguagem de programação, cujos comandos são em português, para representar algoritmos.

EXEMPLO: Algoritmo CALCULA_DOBRO
 início
 Leia NUM
 DOBRO \leftarrow 2 * NUM
 Escreva DOBRO
 fim

VANTAGENS:

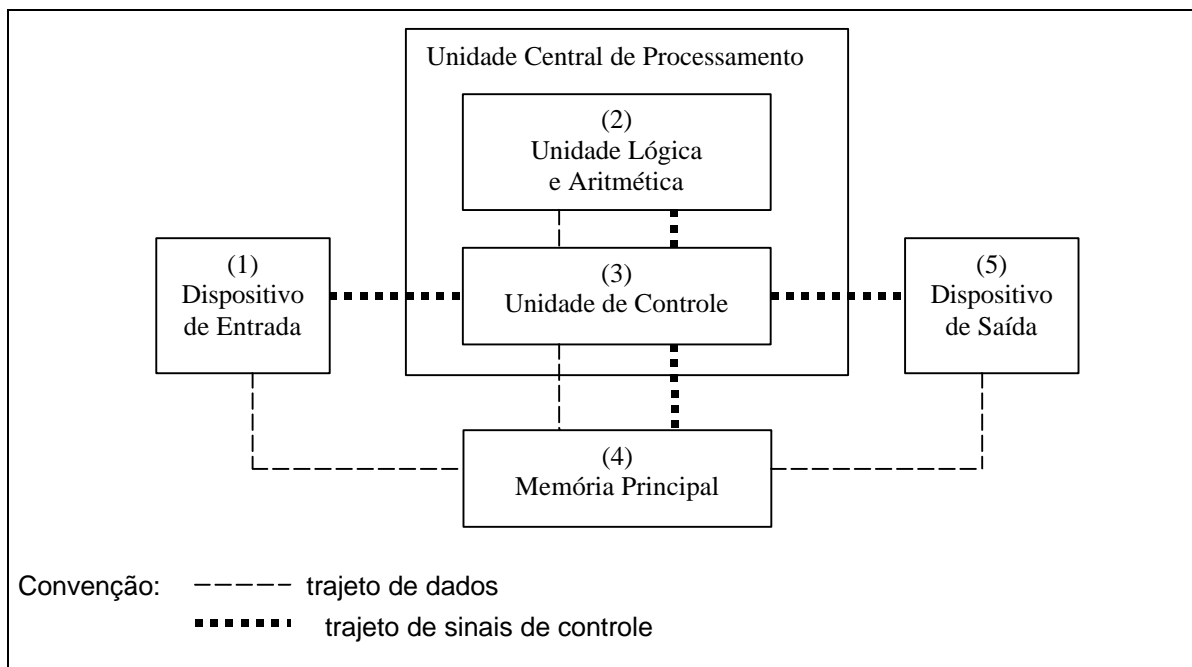
- Independência física da solução (solução lógica apenas);
- Usa o português como base;
- Pode-se definir quais e como os dados vão estar estruturados;
- Passagem quase imediata do algoritmo para uma linguagem de programação qualquer.

DESVANTAGENS:

- Exige a definição de uma linguagem não real para trabalho;
- Não padronizado.

1.5. UM AMBIENTE PARA ESCREVER ALGORITMOS

Descreveremos uma máquina hipotética para a qual escreveremos nossos algoritmos. O nosso computador hipotético apresentará a seguinte organização:



Cada uma das partes constituintes da figura acima tem os seguintes significados:

- (1) Dispositivo de entrada (o teclado):
É o meio pelo qual os dados que serão trabalhados pelo algoritmo vão ser introduzidos em nosso computador hipotético;
- (2) Unidade Lógica e Aritmética (ULA):
Parte responsável pelas operações matemáticas e avaliações lógicas;
- (3) Unidade de Controle:
Exerce controle sobre as demais partes do nosso computador. É uma verdadeira gerente que distribui tarefas às outras unidades;
- (4) Memória:
Guarda o algoritmo a ser executado e os dados a serem utilizados pelo mesmo. Todo dado fornecido ao computador e o resultado de suas operações ficam guardados na memória;
- (5) Dispositivo de Saída (vídeo e impressora):
É o meio que se dispõe para apresentação dos resultados obtidos.

1.5.1. FUNCIONAMENTO DO NOSSO COMPUTADOR

Todos os computadores, independentemente dos seus tamanhos, são conceitualmente semelhantes ao esquema da figura anterior (há algumas diferenças, mas não trataremos aqui dos casos especiais).

Resumidamente, podemos afirmar que existem 4 (quatro) operações básicas que qualquer computador pode executar:

- a) **operações de entrada e saída:** ler dados do teclado e escrever dados na tela são exemplos destas operações. Elas servem para introduzir dados na memória do nosso computador e exibir dados que já estejam lá armazenados;
- b) **operações aritméticas:** são utilizadas na realização de operações matemáticas (adição, subtração, multiplicação e divisão);
- c) **operações lógicas e relacionais:** têm aplicabilidade em comparações, testes de condições lógicas ($2 > 6$? $X = Y$?);
- d) **movimentação de dados entre os vários componentes:** as operações aritméticas são executadas na Unidade Lógica e Aritmética, necessitando da transferência dos dados para essa unidade e da volta do resultado final para ser guardado na memória.

1.5.2. RESOLVENDO UM PROBLEMA

Suponha que queiramos resolver o seguinte problema: a partir de dois números que serão informados, calcular a adição dos mesmos. Se você fosse encarregado de efetuar essa tarefa, seria bem provável que utilizasse os passos a seguir:

- a) saber quais são os números;
- b) calcular a soma dos números;
- c) responder à questão com o valor do resultado.

Vejamos como seria resolvido esse mesmo problema em termos das operações básicas citadas anteriormente:

- a) operação de entrada de dados dos números ;
- b1) movimento do valor dos números entre a memória e a ULA;
- b2) operação aritmética de somar os 2 números;
- b3) movimentação do resultado da ULA para guardar na memória;
- c) operação de saída do resultado, que está guardado na memória, para o dispositivo de saída desejado.

Deve-se salientar que os passos b1 e b3, normalmente, ficam embutidos na operação matemática, não sendo explicitados.

Em resumo, pode-se dizer que escrever algoritmos ou, em última análise, programar, consiste em dividir qualquer problema em muitos pequenos **passos**, usando uma ou mais das quatro operações básicas citadas.

Esses passos que compõem o algoritmo são denominados de **comandos**. Os comandos de uma linguagem de programação podem estar mais próximos da máquina (linguagens de baixo nível) ou serem mais facilmente entendidos pelo homem (linguagens de alto nível). A seqüência de operações básicas, dada anteriormente, para resolver o problema de adicionar dois números, está em uma linguagem de baixo nível para o nosso computador hipotético. Em uma linguagem de alto nível teríamos um resultado assim:

```
Leia X,Y
SOMA ← X + Y
Escreva SOMA
```

1.6. ESTRUTURAS CHAVES DA CONSTRUÇÃO DE ALGORITMOS

Existem 3 estruturas básicas de controle nas quais se baseiam os algoritmos: sequenciação, decisão e repetição. Detalharemos cada uma delas:

1.6.1. SEQUENCIAÇÃO

Os comandos do algoritmo fazem parte de uma seqüência, onde é relevante a ordem na qual se encontram os mesmos, pois serão executados um de cada vez, estritamente, de acordo com essa ordem. De uma forma genérica, poderíamos expressar uma seqüência da seguinte maneira:

```
Comando-1
Comando-2
Comando-3
:
Comando-n
```

Tem-se uma sequenciação de n comandos na qual os comandos serão executados na ordem em que aparecem, isto é, o comando de ordem $i+1$ só será executado após a execução do de ordem i (o 3º só será executado após o 2º).

Todo algoritmo é uma seqüência. A sequenciação é aplicada quando a solução do problema pode ser decomposta em passos individuais.

1.6.2. DECISÃO OU SELEÇÃO

Essa estrutura também é conhecida por estrutura condicional. Há a subordinação da execução de um ou mais comandos à veracidade de uma condição. Vejamos o funcionamento:

```
Se <condição> Então
  <comando-1>
Senão
  <comando-2>
```

Se a <condição> for verdadeira será executado o <comando-1> e, em caso contrário, teremos a execução de <comando-2>.

A decisão deve ser sempre usada quando há a necessidade de testar alguma condição e em função da mesma tomar uma atitude. Em nosso dia-a-dia, estamos sempre tomando decisões, vejamos um exemplo:

Se tiver dinheiro suficiente, **então** vou almoçar em um bom restaurante.
Caso contrário (**senão**), vou comer um sanduíche na lanchonete da esquina.

1.6.3. REPETIÇÃO OU ITERAÇÃO

Essa estrutura também é conhecida por "looping" ou laço. A repetição permite que tarefas individuais sejam repetidas um número determinado de vezes ou tantas vezes quantas uma condição lógica permita. Vejamos alguns exemplos:

- a) vou atirar pedras na vidraça até quebrá-la;
- b) baterei cinco pênaltis;
- c) enquanto tiver saúde e dinheiro, vou desfrutar a vida.

No exemplo (a), vai-se repetir a ação de atirar pedras na janela até que seja satisfeita a condição de quebrar a janela.

No exemplo (b), haverá a repetição da atitude de bater um pênalti um número determinado de vezes (cinco).

No exemplo (c), a condição que me permitirá continuar desfrutando a vida é ter dinheiro e saúde.

A utilização combinada dessas 3 estruturas descritas vai permitir expressar, usando qualquer que seja a ferramenta, a solução para uma gama muito grande de problemas. Todas as linguagens de programação oferecem representantes dessas estruturas.

1.7. REFINAMENTOS SUCESSIVOS

Um algoritmo é considerado completo se os seus comandos forem do entendimento do seu destinatário.

Num algoritmos, um comando que não for do entendimento do destinatário terá que ser desdobrado em novos comandos, que constituirão um **refinamento** do comando inicial, e assim sucessivamente, até que os comandos sejam entendidos pelo destinatário.

Por exemplo, o algoritmo para calcular a média aritmética de dois números pode ser escrito da seguinte forma:

```
Algoritmo CALCULA_MÉDIA
Início
  Receba os dois números
  Calcule a média dos dois números
  Exiba o resultado
Fim
```

Podemos desdobrar o comando “Calcule a média dos dois números” em:

```
Soma os dois números
Divida o resultado por 2
```

Após esse refinamento, o algoritmo pode ser considerado completo, a menos que o destinatário não saiba fazer as operações de adição e divisão, ou não seja capaz de entender diretamente algum comando.

O algoritmo estando completo, podemos reescrevê-lo, inserindo o refinamento na posição do comando que foi refinado. Assim sendo, obtém-se:

```
Algoritmo CALCULA_MÉDIA
Início
  Receba os dois números
  Soma os dois números
  Divida o resultado por 2
  Exiba o resultado
Fim
```

Reescrever um algoritmo completo, com os refinamentos sucessivos inseridos nos seus devidos lugares, permite ter uma visão global de como o algoritmo deve ser executado.

À medida que o algoritmo passa a ser maior e mais complexo, esta visão global torna-se menos clara e, neste caso, um algoritmo apresentado com os refinamentos sucessivos separados oferece uma melhor abordagem para quem precisar entendê-lo.

EXERCÍCIOS PROPOSTOS

- P1.01. Defina, com suas palavras, o que é algoritmo.
- P1.02. Cite alguns algoritmos que podemos encontrar na vida cotidiana.
- P1.03. De acordo com seu entendimento, qual é a característica mais importante em um algoritmo? Justifique a sua resposta.
- P1.04. Um algoritmo não pode conter um comando como "Escreva todos os números inteiros positivos". Por quê?
- P1.05. Suponha que temos um robô a nossa disposição. Esse robô chama-se MANNY e precisa ser ensinado a fazer determinadas tarefas. Para ensinar o MANNY, vamos fazer uso do português para passar-lhe as instruções necessárias à execução de cada atividade. Escreva os passos necessários para o nosso robô executar:
- a) encher uma bacia com água;
 - b) trocar uma lâmpada no teto de sua casa;
 - c) trocar o pneu de um carro;
 - d) calcular a sua idade daqui a 20 anos;
 - e) calcular a média de um aluno com 3 notas.
- P1.06. Cite as formas básicas para se representar algoritmos, definindo-as.
- P1.07. Em sua opinião, qual a melhor forma de se representar algoritmos? Justifique sua resposta.
- P1.08. Descreva, com suas próprias palavras, o funcionamento do nosso computador hipotético.
- P1.09. Especifique soluções, em termos das operações básicas do nosso computador, para os itens (d) e (e) do exercício P1.05.
- P1.10. Quais as estruturas básicas de controle dos algoritmos? Explique cada uma delas.
- P1.11. Identifique nas respostas do exercício P1.05 a utilização das estruturas básicas de controle de fluxo.
- P1.12. Escreva o algoritmo solução para o problema de multiplicar dois números (a solução deve ser expressa em alto nível).
- P1.13. Resolva o P1.09 em termos de uma linguagem de alto nível.
- P1.14. Em que consiste a técnica de "refinamentos sucessivos" ?
- P1.15. É comum ouvirmos programadores experientados afirmarem:
"algoritmos ... aprendi e nunca usei na prática ... não vejo necessidade...".
Discuta esse tipo de afirmativa.

Capítulo 2

LINGUAGEM ALGORÍTMICA

Para uma melhor padronização de nossos estudos, vamos agora definir uma linguagem para a construção de nossos algoritmos. Antes de começarmos a definir as operações básicas de nossa linguagem algorítmica, é importante que conheçamos o conceito de *variável*.

2.1. CONCEITO DE VARIÁVEL

Sabe-se da Matemática que uma variável é a representação simbólica dos elementos de um certo conjunto.

Nos algoritmos destinados a resolver um problema no computador, **a cada variável corresponde uma posição de memória, cujo conteúdo pode variar ao longo do tempo durante a execução de um algoritmo.** Embora a variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

Toda variável é identificada por um nome ou identificador. Assim, por exemplo, num algoritmo para calcular a área de um triângulo retângulo pelo teorema de pitágoras ($a^2 = b^2 + c^2$), os identificadores A, B e C podem representar as posições de memória que armazenam o valor da hipotenusa e dos catetos. É importante que nunca usemos uma palavra reservada, isto é, que faça parte da linguagem algorítmica, como um identificador, pois poderá causar ambigüidade no entendimento do algoritmo.

Na nossa linguagem algorítmica, vamos admitir que nossas variáveis poderão armazenar valores numéricos ou alfanuméricos (strings).

Agora que sabemos o que é uma variável, vamos às definições das operações básicas de nossa linguagem algorítmica.

2.2. OPERAÇÃO DE ATRIBUIÇÃO

A operação de atribuição permite que se forneça um valor a uma certa variável. Se for atribuído uma expressão à variável, será armazenado o resultado daquela expressão. Se for atribuído uma outra variável, será armazenado o conteúdo daquela variável. Para a operação de atribuição, utilizaremos a seguinte sintaxe:

variável ← expressão

Exemplos:

A ← 2
B ← A
NOTA ← 10

NOME ← 'João'
SENHA ← 'X3Y9'
C ← 1 / 3

A ← B + C
NOTA ← NOTA - 1
X ← 2.5

2.3. OPERAÇÕES DE ENTRADA E SAÍDA

Os cálculos do computador são de pouco valor a não ser que, primeiro, possamos fornecer os dados sobre os quais estes cálculos serão efetuados e, segundo, ver os resultados destes cálculos.

Definimos aqui dois novos comandos da nossa linguagem algorítmica para manusear entrada e saída. O comando **leia** nos permite ler valores dados atribuindo-os à variáveis indicadas; o comando **escreva** nos permite mostrar os resultados. A entrada pode vir do teclado ou de qualquer outro dispositivo de entrada. A saída pode aparecer na tela do monitor de vídeo ou ser impressa em papel. Não nos preocuparemos com detalhes destes dispositivos. A sintaxe destes comandos são:

leia variável-1, variável-2, ..., variável-n

escreva expressão-1, expressão-2, ..., expressão-n

Exemplos:

leia NOTA
leia A,B,C
leia NOME

escreva 15
escreva NOTA+2
escreva A,B

EXERCÍCIOS PROPOSTOS

- P2.01. Dê o conceito de variável.
- P2.02. Exemplifique o uso da operação de Atribuição.
- P2.03. Qual a finalidade de uma operação de entrada ? Dê exemplos.
- P2.04. Qual a finalidade de uma operação de saída? Dê exemplos.
- P2.05. Escreva os comandos necessários para:
 - a) ler o nome de uma pessoa
 - b) ler as 3 notas de um aluno
 - c) ler o peso e altura de uma pessoa

2.4. ESTRUTURA SEQÜENCIAL

Num algoritmo, os comandos deverão ser executados numa seqüência linear, seguindo-se o texto em que estão escritos, de cima para baixo, se não houver indicação em contrário.

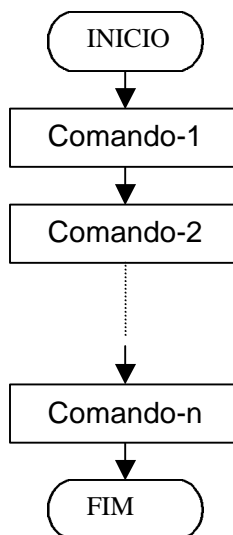
Nesta apostila, os algoritmos são iniciados com a palavra **início** e finalizados com a palavra **fim**.

Linguagem Algorítmica:

```

início
  comando-1
  comando-2
  ...
  comando-n
fim
    
```

Fluxograma:



Exemplo:

```

início
  leia A,B
  SOMA ← A + B
  escreva SOMA
fim
    
```

EXERCÍCIOS PROPOSTOS

- P2.06. Em que consiste a estrutura seqüencial?
- P2.07. Determine os valores finais de A, B e C após a execução do trecho do algoritmo abaixo:

```

A ← 0
B ← 1
C ← A + B
A ← A + 1
B ← A + B + C
    
```

A	B	C

- P2.08. A ordem das atribuições é importante? $A \leftarrow B$ e $C \leftarrow A$ tem o mesmo efeito de $C \leftarrow A$ e $A \leftarrow B$?
- P2.09. Em quais dos seguintes pares é importante a ordem dos comandos ?

- a) $X \leftarrow Y$
 $Y \leftarrow X$
- b) $X \leftarrow Y$
 $Z \leftarrow X$
- c) $X \leftarrow Z$
 $X \leftarrow Y$
- d) $Z \leftarrow Y$
 $X \leftarrow Y$

P2.10. Escreva um algoritmo que leia duas variáveis inteiras e troque o conteúdo entre elas.

P2.11. Escreva um algoritmo que leia um número inteiro positivo e exiba o dobro do mesmo.

P2.12. Escreva um algoritmo para calcular e exibir a média ponderada de 2 notas dadas.
(nota1= peso 6 e nota2= peso 4)

P2.13. Escreva um algoritmo para calcular e exibir o comprimento de uma circunferência, sendo dada o valor de seu raio.

$$C = 2\pi R$$

P2.14. Escreva um algoritmo para ler uma temperatura dada na escala Fahrenheit e exibir o equivalente em Celsius.

$$C = \frac{5}{9}(F - 32)$$

P2.15. Escreva um algoritmo para calcular a área de um triângulo, sendo dados a sua base e a sua altura.

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

2.5. ESTRUTURA CONDICIONAL

A estrutura condicional permite a escolha do grupo de ações e estruturas a ser executado quando determinadas condições (expressões lógicas) são ou não satisfeitas.

Esta estrutura pode se apresentar de duas formas.

2.5.1. ESTRUTURA CONDICIONAL SIMPLES

Linguagem Algorítmica:

```

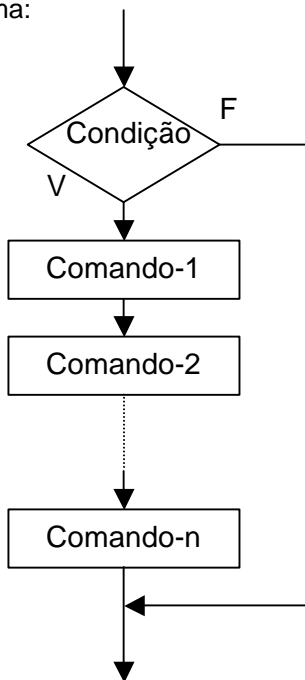
se condição então
  comando-1
  comando-2
  ...
  comando-n
    
```

Exemplo:

```

...
se MÉDIA ≥ 7 então
  SITUAÇÃO ← 'Aprovado'
  Escreva SITUAÇÃO
...
    
```

Fluxograma:



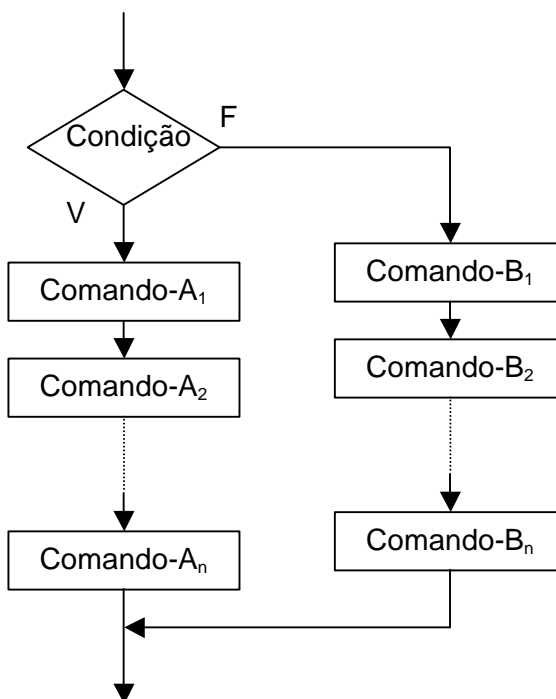
2.5.2. ESTRUTURA CONDICIONAL COMPOSTA

Linguagem Algorítmica:

```

se condição então
    comando-A1
    comando-A2
    ...
    comando-An
senão
    comando-B1
    comando-B2
    ...
    comando-Bn
    
```

Fluxograma:



Exemplo:

```

...
se PESO > 80 então
    escreva 'Você está obeso'
    escreva 'Faça atividades físicas'
senão
    escreva 'Você está no peso certo'
    escreva 'Procure manter sua forma'
...
    
```

EXERCÍCIOS PROPOSTOS

- P2.16. Qual a utilidade da estrutura condicional ?
- P2.17. Qual a diferença entre a estrutura condicional simples e a composta ?
- P2.18. Escreva um algoritmo para ler um número e determinar se ele é maior, igual ou menor que zero.
- P2.19. Escreva um algoritmo que leia dois números e exiba-os em ordem crescente.
- P2.20. Escreva um algoritmo que leia dois números e exiba o maior deles.
- P2.21. Deseja-se calcular a conta de consumo de energia elétrica de um consumidor. Para isto, escreva um algoritmo que leia o código do consumidor, o preço do Kw e a quantidade de Kw consumido, e exiba o código do consumidor e o total a pagar.
 - total a pagar = preço x quantidade
 - total a pagar mínimo = R\$ 11,20
- P2.22. Escreva um algoritmo que determine o grau de obesidade de uma pessoa, sendo fornecido o peso e a altura da pessoa. O grau de obesidade é determinado pelo índice da massa corpórea ($Massa = \frac{Peso}{Altura^2}$) através da tabela abaixo:

MASSA CORPÓ-REA	GRAU DE OBESIDADE
< 26	Normal
≥ 26 e < 30	Obeso
≥ 30	Obeso Mórbido

- P2.23. Faça um algoritmo que, dado as três notas de um aluno, determine e exiba a sua média final e o seu conceito, sabendo-se que:
 - a média final é calculada pela média aritmética das 3 notas;
 - o conceito é determinado de com base na tabela abaixo:

MÉDIA FINAL	CONCEITO
≥ 8,0	A
≥ 5,0 e < 8,0	B
< 5,0	C

P2.24. O Botafogo Futebol Clube deseja aumentar o salário de seus jogadores. O reajuste deve obedecer a seguinte tabela:

SALÁRIO ATUAL (R\$)	AUMENTO
0,00 a 1.000,00	20%
1.000,01 a 5.000,00	10%
acima de 5.000,00	0%

Escrever um algoritmo que leia o nome e o salário atual de um jogador, e exiba o nome, o salário atual e o salário reajustado.

P2.25. Faça um algoritmo para calcular a conta final de um hóspede de um hotel fictício, considerando que:

- a) serão lidos o nome do hóspede, o tipo do apartamento utilizado (A, B, C ou D), o número de diárias utilizadas pelo hóspede e o valor do consumo interno do hóspede;
- b) o valor da diária é determinado pela seguinte tabela:

TIPO DO APTO.	VALOR DA DIÁRIA (R\$)
A	150,00
B	100,00
C	75,00
D	50,00

- c) o valor total das diárias é calculado pela multiplicação do número de diárias utilizadas pelo valor da diária;
- d) o subtotal é calculado pela soma do valor total das diárias e o valor do consumo interno;
- e) o valor da taxa de serviço equivale a 10% do subtotal;
- f) a total geral resulta da soma do subtotal com a taxa de serviço.
- g) escreva a conta final contendo: o nome do hóspede, o tipo do apartamento, o número de diárias utilizadas, o valor unitário da diária, o valor total das diárias, o valor do consumo interno, o subtotal, o valor da taxa de serviço e o total geral.

2.6. ESTRUTURA DE REPETIÇÃO

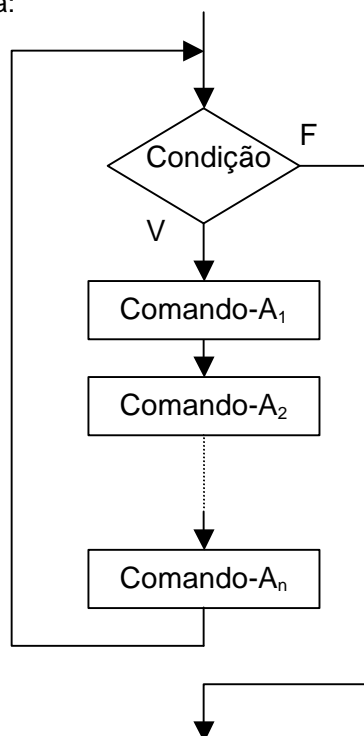
A estrutura de repetição permite que uma seqüência de comandos seja executada repetidamente até que uma determinada condição não seja satisfeita. Utilizaremos o comando **enquanto** para representar esta estrutura. Sua sintaxe é:

Linguagem Algorítmica:

```

enquanto condição
  comando-A1
  comando-A2
  ...
  comando-An
    
```

Fluxograma:



Exemplo:

```

...
leia SENHA
enquanto SENHA <> 'ASPER'
  escreva 'Senha inválida'
  escreva 'Digite a senha novamente'
  leia SENHA
...
    
```

EXERCÍCIOS PROPOSTOS

P2.26. Qual a utilidade da estrutura de repetição?

P2.27. Faça o acompanhamento da execução do trecho de algoritmo abaixo e preencha a Tabela de Variáveis:

TRECHO DE ALGORITMO	TABELA DE VARIÁVEIS			
	N	L	N ≠ 6	Saída
N ← 0 L ← 1 enquanto N ≠ 6 L ← L . (-1) N ← N + 1 se L > 0 então escreva N				

P2.28. Faça um algoritmo que mostre todos os números inteiros de 1 a 50.

P2.29. Faça um algoritmo que leia um número N, some todos os números inteiros de 1 a N, e mostre o resultado obtido.

P2.30. Faça um algoritmo que leia uma lista de números inteiros positivos terminada pelo número 0 (zero). Ao final, o algoritmo deve mostrar a média aritmética de todos os números lidos (excluindo o zero).

P2.31. Faça um algoritmo que leia N e uma lista de N números e mostre a soma de todos os números da lista.

P2.32. Escreva um algoritmo que leia um conjunto de 100 números inteiros positivos e determine o maior deles.

P2.33. Escreva um algoritmo que leia um número N e uma lista de N números inteiros positivos e determine o maior número da lista.

P2.34. Escreva um algoritmo que leia um conjunto de números inteiros positivos e determine o maior deles. A leitura do valor 0 (zero) indica o fim dos dados (flag).

P2.35. Faça um algoritmo que gere a seguinte série: 10, 20, 30, 40, ..., 990, 1000.

P2.36. Sendo
$$H = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{N}$$
, faça um algoritmo p/ calcular H. O número N é lido.

P2.37. Faça um algoritmo que leia um número N, calcule e mostre os N primeiros termos da seqüência de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, ...). O valor lido para N sempre será maior ou igual a 2.

P2.38. Escreva um algoritmo que calcule o fatorial de um número inteiro lido, sabendo-se que:

$$N! = 1 \times 2 \times 3 \times \dots \times N-1 \times N$$

$$0! = 1$$

P2.39. O Botafogo Futebol Clube deseja aumentar o salário de seus 22 jogadores. O reajuste deve obedecer a seguinte tabela:

SALÁRIO ATUAL (R\$)	AUMENTO
0,00 a 1.000,00	20%
1.000,01 a 5.000,00	10%
acima de 5.000,00	0%

Escrever um algoritmo que:

- leia o nome e o salário atual de cada jogador;
- exiba o nome, o salário atual e o salário reajustado de cada jogador;
- exiba o total da folha de salários do clube, antes do reajuste.
- exiba o total da folha de salários do clube, após o reajuste.
- exiba o percentual de reajuste sobre o total da folha de salários.

P2.40. O cardápio de uma casa de lanches, especializada em sanduíches, é dado abaixo. Escreva um algoritmo que leia o código e a quantidade de cada item comprado por um freguês, calcule e exiba o total a pagar. Obs: A leitura do código 'X' indica o fim dos itens.

CÓDIGO	PRODUTO	PREÇO (R\$)
H	Hamburger	1,50
C	Cheeseburger	1,80
M	Misto Quente	1,20
A	Americano	2,00
Q	Queijo Prato	1,00

P2.41. Num frigorífico existem 90 bois. Cada boi traz preso em seu pescoço um cartão contendo seu número de identificação e seu peso. Faça um algoritmo que escreva o número e o peso do boi mais gordo e do boi mais magro (supondo que não haja empates).

P2.42. Faça um algoritmo que leia a altura de um grupo de 20 pessoas, calcule e exiba:

- a maior altura do grupo;
- a altura média;
- o número de pessoas com altura superior a 2 metros.

P2.43. Faça um algoritmo que, para um número indeterminado de pessoas:

- leia a idade de cada pessoa, sendo que a leitura da idade 0 (zero) indica o fim dos dados (flag) e não deve ser considerada;
- calcule e escreva o número de pessoas;
- calcule e escreva a idade média do grupo;
- calcule e escreva a menor idade e a maior idade.

P2.44. Faça um algoritmo que leia uma lista de letras terminada pela letra Z. Ao final, o algoritmo deve mostrar a quantidade lida de cada vogal.

P2.45. Uma certa firma fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um novo produto lançado no mercado. Para isto, forneceu o sexo do entrevistado (M-masculino ou F-feminino) e sua resposta (S-sim ou N-não). Sabendo-se que foram entrevistadas 2.000 pessoas, fazer um algoritmo que calcule e escreva:

- número de pessoas que responderam sim (S);
- número de pessoas que responderam não (N);
- a porcentagem de pessoas do sexo feminino (F);
- a porcentagem de pessoas do sexo masculino (M);
- a porcentagem de pessoas do sexo feminino (F) que responderam sim (S);
- a porcentagem de pessoas do sexo masculino (M) que responderam não (N).

P2.46. Foi feita uma pesquisa de audiência de canal de TV em várias casas de uma certa cidade, num determinado dia. Para cada casa visitada, é fornecido o número do canal (5, 7, 10 ou 12) e o número de pessoas que o estavam assistindo naquela casa. Fazer um algoritmo que:

- leia um número indeterminado de dados, sendo que o flag corresponde ao número de canal igual a 0 (zero);
- calcule e escreva a porcentagem de audiência de cada emissora.

P2.47. Escreva um algoritmo que leia o número de andares de um prédio e, a seguir, para cada andar do prédio, leia o número de pessoas que entraram e saíram do elevador.

Considere que o elevador está vazio e está subindo, os dados se referem a apenas uma subida do elevador e que o número de pessoas dentro do elevador será sempre maior ou igual a zero.

Se o número de pessoas, após a entrada e saída, for maior que 15, deve ser mostrada a mensagem "Excesso de passageiros. Devem sair X", sendo X o número de pessoas que devem sair do elevador, de modo que seja obedecido o limite de 15 passageiros.

Após a entrada e saída no último andar, o algoritmo deve mostrar quantas pessoas permaneceram no elevador para descer.

P2.48. Faça um algoritmo que leia vários códigos do jogador (1 ou 2) que ganhou o ponto em uma partida de pingue-pongue, e responda quem ganha a partida.

A partida chega ao final se um dos jogadores chega a 21 pontos e a diferença de pontos entre os jogadores é maior ou igual a dois. Caso contrário, ganha aquele que, com mais de 21 pontos, consiga colocar uma vantagem de dois pontos sobre o adversário.

Capítulo 3

LINGUAGEM DE PROGRAMAÇÃO PASCAL

3.1. INTRODUÇÃO

Para armazenar um algoritmo na memória de um computador e para que ele possa, em seguida, comandar as operações a serem executadas, é necessário que ele seja **programado**, isto é, que seja transcrito para uma **linguagem** que o computador possa entender, direta ou indiretamente.

3.1.1. LINGUAGENS DE PROGRAMAÇÃO

Linguagem é uma maneira de comunicação que segue uma forma e uma estrutura com significado interpretável. Portanto, **linguagem de programação** é um conjunto finito de palavras, comandos e instruções, escritos com o objetivo de orientar a realização de uma tarefa pelo computador.

Logicamente, a linguagem que nós utilizamos em nosso cotidiano é diferente da linguagem utilizada pela máquina. A máquina trabalha somente com códigos numéricos (linguagem de máquina), baseados nos números 0 e 1 (sistema binário), que representam impulsos elétricos, ausente e presente.

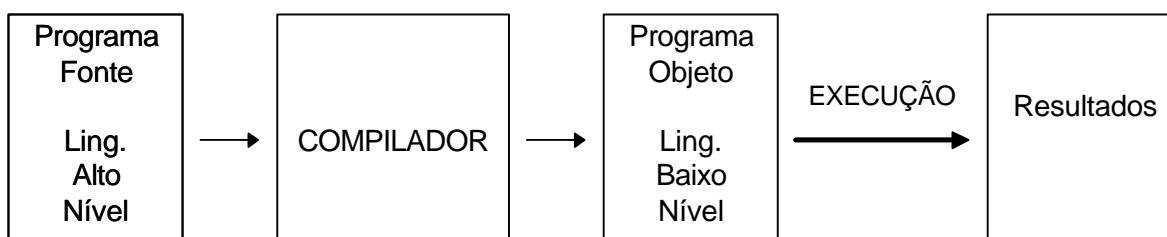
Assim, qualquer linguagem de programação deve estar situada entre dois extremos: o da linguagem natural do homem (muito clara, porém lenta) e o da linguagem de máquina (muito rápida, porém complexa).

Este é o conceito de nível de linguagem: alto nível para as mais próximas da linguagem humana; baixo nível para as mais semelhantes à linguagem de máquina.

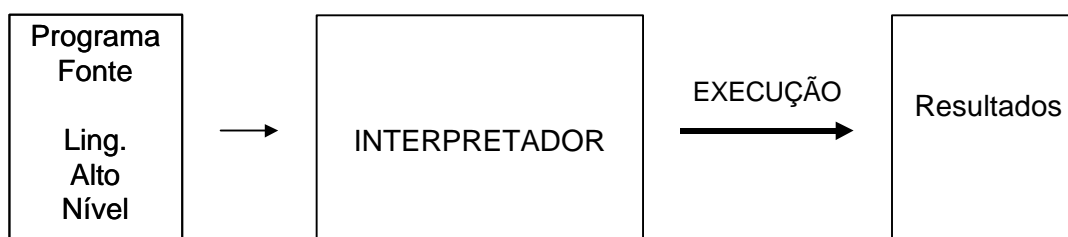
3.1.2. TRADUTORES

Para que um computador possa "entender" um programa escrito em uma linguagem de alto nível, torna-se necessário um meio de tradução entre a linguagem utilizada no programa e a linguagem de máquina. Este meio pode ser de dois tipos: **compilador** e **interpretador**.

COMPILADOR - traduz o programa escrito em linguagem de alto nível (programa-fonte) para um programa equivalente escrito em linguagem de máquina (programa-objeto).



INTERPRETADOR - traduz e envia para execução, instrução por instrução e o programa permanece na forma fonte.



Exemplos de linguagens de programação: PASCAL, C, CLIPPER, BASIC, COBOL, etc.

3.1.3. A LINGUAGEM PASCAL

A linguagem de programação PASCAL foi criada para ser uma ferramenta educacional, isto no início da década de 70 pelo Prof. Niklaus Wirth da Universidade de Zurique. Foi batizada pelo seu idealizador em homenagem ao grande matemático Blaise Pascal, inventor de uma das primeiras máquinas lógicas conhecidas. Foi baseada em algumas linguagens estruturadas existentes na época, ALGOL e PLI.

Apesar de seu propósito inicial, o PASCAL começou a ser utilizado por programadores de outras linguagens, tornando-se, para surpresa do próprio Niklaus, um produto comercial. Contudo, somente ao final de 1983 foi que a empresa americana Borland International lançou o TURBO PASCAL.

O **TURBO PASCAL** é um programa que contém, além do compilador PASCAL, um ambiente completo de programação, com editor de programa, depurador de erros, sistema de ajuda, etc.

Estudaremos, a seguir, os itens fundamentais que compõe a linguagem PASCAL.

3.2. ELEMENTOS BÁSICOS

3.2.1. IDENTIFICADORES

São nomes escolhidos para representar constantes, variáveis, tipos, funções, procedimentos, unidades, programas e campos de um registro. Para definirmos um identificador, devemos observar o seguinte:

- pode ter qualquer comprimento, mas apenas os sessenta e três primeiros caracteres são significativos;
- deve ter como primeiro caracter uma letra;
- após a primeira letra só pode conter letras, dígitos ou sublinha (_);
- não pode conter espaços;
- letras maiúsculas e minúsculas são indiferentes;
- não podem haver identificadores repetidos;
- não pode ser uma palavra reservada.

3.2.2. PALAVRAS RESERVADAS

São palavras que têm um sentido predeterminado na linguagem e não podem ser usadas como identificadores.

ABSOLUTE	END	INLINE	PROCEDURE	TYPE
AND	EXTERNAL	INTERFACE	PROGRAM	UNIT
ARRAY	FILE	INTERRUPT	RECORD	UNTIL
BEGIN	FOR	LABEL	REPEAT	USES
CASE	FORWARD	MOD	SET	VAR
CONST	FUNCTION	NIL	SHL	WHILE
DIV	GOTO	NOT	SHR	WITH
DO	IF	OF	STRING	XOR
DOWNTON	IMPLEMENTATION	OR	THEN	
ELSE	IN	PACKED	TO	

3.3. TIPOS DE DADOS

3.3.1. SIMPLES

INTEGER - Envolve os números inteiros. Na versão 5.0 do Turbo Pascal, existem também outros tipos de números inteiros: SHORTINT, BYTE, WORD e LONGINT.

Tipo	Valor mínimo	Valor máximo	Bytes ocupados
SHORTINT	-128	127	1
BYTE	0	255	1
INTEGER	-32768	32767	2
WORD	0	65535	2
LONGINT	-2147483648	2147483647	4

Exemplos: -45, 1, 138, 0, -2

REAL - abrange os números reais. Na versão 5.0, existem também outros tipos de números reais: SINGLE, DOUBLE, EXTENDED e COMP.

Tipo	Valor mínimo	Valor máximo	Bytes ocupados	Dígitos Significativos
REAL	2.9×10^{-39}	1.7×10^{38}	6	11-12
SINGLE	1.5×10^{-45}	3.4×10^{38}	4	7-8
DOUBLE	5.0×10^{-324}	1.7×10^{308}	8	15-16
EXTENDED	3.4×10^{-4932}	1.1×10^{4932}	10	19-20
COMP	$-2^{63} + 1$	$2^{63} - 1$	8	19-20

Exemplos: 4.5, -32.0, .5, 7.8E3, 21E+3, -315E-3

CHAR - representa um único carácter, escrito entre apóstrofos ('). A maioria dos computadores utilizam a tabela de códigos ASCII para representar todos os caracteres disponíveis. Exemplos:

'A', 'B', 'a', '1', '@', ' '

BOOLEAN - representa um valor lógico. Utiliza apenas duas constantes lógicas: TRUE (verdadeiro) e FALSE (falso).

3.3.2. ESTRUTURADOS

STRING - formado por um conjunto de elementos do tipo CHAR. O tamanho máximo é de 255 caracteres. Exemplos:

'ASPER', 'Processamento de Dados', '123'

Discutiremos com mais detalhes o tipo string em um capítulo especialmente dedicado a este fim.

Os outros tipos de dados estruturados são: **ARRAY**, **RECORD**, **FILE**, **SET** e **TEXT** e serão apresentados no decorrer da apostila.

3.3.3. DEFINIDOS PELO USUÁRIO

A linguagem Pascal permite que os programadores, além de usar os tipos predefinidos na linguagem, também possam criar novos tipos de dados. Isto torna-se bastante útil quando lidamos com estruturas de dados complexas, como também ajuda a tornar o programa mais legível. Este assunto, pela sua complexidade, será abordado mais adiante.

3.4. EXPRESSÕES ARITMÉTICAS

São expressões onde utilizamos os números inteiros ou reais como operandos e os operadores aritméticos, dando sempre como resultado valores numéricos.

3.4.1. OPERADORES ARITMÉTICOS

Os operadores aritméticos representam as operações mais comuns da matemática. São eles:

Operador	Operação	Operandos	Resultado
+	Adição	Inteiro, Real	Inteiro, Real
-	Subtração	Inteiro, Real	Inteiro, Real
*	Multiplicação	Inteiro, Real	Inteiro, Real
/	Divisão Real	Inteiro, Real	Real
DIV	Divisão Inteira	Inteiro	Inteiro
MOD	Resto da Divisão	Inteiro	Inteiro

EXEMPLOS:

Expressão	Resultado
1 + 2	3
5.0 - 1	4.0
2 * 1.5	3.0
5 / 2	2.5
5 DIV 2	2
5 MOD 2	1

3.4.2. PRIORIDADE

Em uma expressão aritmética, a ordem de avaliação dos operadores obedece a tabela abaixo:

Prioridade	Operadores
1ª	* / DIV MOD
2ª	+ -

OBSERVAÇÕES:

- Quando existe em uma expressão operadores com a mesma prioridade, a execução é da esquerda para direita.
- Caso seja necessário alterar a ordem de prioridade, deve-se utilizar parênteses. A expressão entre parênteses terá prioridade máxima. Caso haja parênteses aninhados, a ordem de execução será do mais interno para o mais externo.

EXEMPLOS:

$$2 + 3 / 2 = 2 + 1.5 = 3.5$$

$$(2 + 3) / 2 = 5 / 2 = 2.5$$

3.4.3. FUNÇÕES E PROCEDIMENTOS NUMÉRICOS PREDEFINIDOS

São subprogramas já prontos à disposição dos usuários, para o cálculo das funções matemáticas mais comuns.

Função	Finalidade	Tipo do argumento	Tipo do resultado
ABS (X)	Valor Absoluto	Inteiro, Real	o mesmo do argumento
FRAC (X)	Parte Fracionária	Real	Real
TRUNC (X)	Parte Inteira	Real	Inteiro
ROUND (X)	Valor Arredondado	Real	Inteiro
SQR (X)	Eleva ao quadrado	Inteiro, Real	o mesmo do argumento
SQRT (X)	Raiz quadrada	Inteiro, Real	Real
LN (X)	Logaritmo Natural	Real	Real
EXP (X)	Exponencial	Real	Real

Como não existe em Pascal um operador nem uma função específica para a operação de Potenciação, podemos conseguí-la utilizando as funções LN(X) e EXP(X). Para calcular o valor de X^N é suficiente usar:

`EXP (LN (X) * N)`

EXEMPLOS:

Expressão	Resultado
<code>ABS (-2.5)</code>	2.5
<code>ABS (8)</code>	8
<code>FRAC (5.234)</code>	0.234
<code>TRUNC (2.78)</code>	2
<code>ROUND (2.78)</code>	3
<code>SQR (2)</code>	4
<code>SQR (1.5)</code>	2.25
<code>SQRT (4)</code>	2.0
<code>SQRT (2.25)</code>	1.5
<code>EXP (LN (2) * 3)</code>	8

3.5. EXPRESSÕES LÓGICAS

As operações lógicas podem ser consideradas afirmações que serão testadas pelo computador, tendo como resultado, um valor **verdadeiro** ou **falso**. São utilizadas com os operadores **relacionais** e **lógicos**.

3.5.1. OPERADORES RELACIONAIS

São usados na comparação de duas expressões de qualquer tipo, retornando um valor lógico (**TRUE** ou **FALSE**) como resultado da operação.

Operador	Operação
<code>=</code>	igual
<code>></code>	maior
<code><</code>	menor
<code>>=</code>	maior ou igual
<code><=</code>	menor ou igual
<code><></code>	diferente

Obs: as operações lógicas só podem ser efetuadas com relação a valores do mesmo tipo.

EXEMPLOS:

Expressão	Resultado
<code>1 = 2</code>	FALSE
<code>'A' = 'a'</code>	FALSE
<code>5 > 2</code>	TRUE
<code>3 <= 3</code>	TRUE
<code>TRUE < FALSE</code>	FALSE
<code>'JOAO' > 'JOSE'</code>	FALSE
<code>2 + 3 <> 5</code>	FALSE
<code>'comp' <> 'COMP'</code>	TRUE
<code>'11' < '4'</code>	TRUE

3.5.2. OPERADORES LÓGICOS

São usados para combinar expressões lógicas.

Operador	Operação
<code>not</code>	não (negação)
<code>and</code>	e (conjunção)
<code>or</code>	ou (disjunção)

A tabela verdade (abaixo) apresenta o resultado de cada operador lógico, com os valores dados para as expressões lógicas A e B:

A	B	A and B	A or B	not A	not B
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

3.5.3. PRIORIDADE

Em uma expressão lógica, a ordem de avaliação dos operadores segue a tabela abaixo:

Prioridade	Operadores
1 ^a	NOT
2 ^a	AND
3 ^a	OR
4 ^a	= > < >= <= <>

Como a ordem de precedência dos operadores lógicos é maior que a dos operadores relacionais, devem sempre ser usados parênteses quando se escrever uma expressão lógica complexa. Por exemplo:

(A > B) OR (B = C)

EXERCÍCIOS PROPOSTOS

P3.13. Qual o resultado das expressões aritméticas abaixo, sabendo-se que os valores de X, Y e Z são, respectivamente, 1, 2 e 5 ?

- Z mod Y div Y
- X + Y + Z / 3
- FRAC(X / Z) + ROUND(Z / Y) * TRUNC(Z / Y)
- SQRT(Z div Y + X * Y)
- Z - ABS(X - SQR(Y))

P3.14. O que são Funções Predefinidas ?

P3.15. Escreva o resultado das seguintes funções:

- ABS(-4)
- ABS(5.2)
- FRAC(23.0)
- FRAC(-3.1)
- TRUNC(1.8)
- TRUNC(2.2)
- ROUND(1.8)
- ROUND(2.2)
- SQR(1.0)
- SQR(10)
- SQRT(25)
- SQRT(9.0)

P3.16. Escreva a expressão para se calcular o valor de 2^5 .

P3.17. Preencha a Tabela Verdade abaixo:

A	B	A and B	A or B	not A	not B
TRUE	TRUE				
TRUE	FALSE				
FALSE	TRUE				
FALSE	FALSE				

P3.18. Escreva o resultado das seguintes comparações:

- a) `3 = 3.0`
- b) `'a' <= 'A'`
- c) `' ' = ' ' '`
- d) `'CASA' <> 'casa'`
- e) `FALSE = FALSE`
- f) `'JOAQUIM' < 'JOSE'`
- g) `'22' > '200'`

P3.19. Qual o resultado das expressões lógicas abaixo, sabendo-se que os valores de A e B são, respectivamente, TRUE e FALSE ?

- a) `not A and B or A and not B`
- b) `not (not (A or B) and (A or B))`
- c) `A or B and not A or not B`
- d) `(A or B) and (not A or not B)`

3.6. FORMATO DE UM PROGRAMA PASCAL

Pascal é uma linguagem altamente *estruturada* que possui uma rigidez definida, embora sua estrutura de programa seja flexível. Cada seção ou parte de um programa em Pascal deve aparecer numa seqüência apropriada e ser sistematicamente correta, senão ocorrerá um erro.

Por outro lado, no Pascal não há regras específicas para o uso de espaço, linhas quebradas, requisições e assim os comandos podem ser escritos no *formato livre* em quase todos os estilos em que o programador deseja utilizar.

Um programa escrito em Pascal tem o seguinte formato:

```
PROGRAM <identificador>;
<bloco>.
```

O <bloco>, por sua vez, está dividido em seis áreas, onde somente a última é obrigatória e devem obedecer a seqüência abaixo. São elas:

- Área de declaração de uso de unidades
- Área de declaração de constantes
- Área de declaração de tipos
- Área de declaração de variáveis
- Área de declaração de procedimentos e funções
- Área de comandos

Observação: no Turbo Pascal, a cláusula PROGRAM, bem como a seqüência correta das declarações, não são obrigatórios.

3.6.1. DECLARAÇÃO DE USO DE UNIDADES

Um programa Pascal pode fazer uso de algumas unidades padrão que estão disponíveis no Sistema Turbo, tais como: CRT, DOS, PRINTER, GRAPH, etc.

A área de declaração de uso de unidades possui o seguinte formato:

```
USES <unidade> , ... , <unidade> ;
```

EXEMPLO:

```
USES CRT, PRINTER;
```

3.6.2. DECLARAÇÃO DE CONSTANTES

Serve para associarmos nomes às constantes utilizadas no programa. Possui o seguinte formato:

```
CONST <identificador>=<valor>;...;<identificador>=<valor>;
```

EXEMPLO:

```
CONST BRANCO = ' ' ; PI = 3.1416 ; MAX = 10 ; OK = TRUE;
```

3.6.3. DECLARAÇÃO DE TIPOS

Serve para definirmos novos tipos e estruturas de dados. Não detalharemos esse tópico agora por ser assunto dos próximos capítulos..

3.6.4. DECLARAÇÃO DE VARIÁVEIS

Serve para associarmos tipos às variáveis utilizadas no programa. Possui o seguinte formato:

```
VAR
  <lista-de-identificadores> : <tipo>;
  ...
  <lista-de-identificadores> : <tipo>;
```

Onde:

<lista-de-identificadores> é uma lista de nomes de variáveis de um mesmo tipo, separadas por vírgula.
 <tipo> é o nome de um dos tipos predefinidos na linguagem ou criado pelo programador.

EXEMPLO:

```
VAR  X,Y,Z : REAL;
      I,J : INTEGER;
      ACHOU : BOOLEAN;
      LETRA : CHAR;
      PALAVRA,FRASE : STRING;
```

3.6.5. DECLARAÇÃO DE PROCEDIMENTOS E FUNÇÕES

Nesta área são definidos os procedimentos e funções utilizados pelo programa. Também é um assunto que será detalhado mais adiante.

3.6.6. ÁREA DE COMANDOS

É nesta área onde é inserido o algoritmo do programa. Os comandos são separados entre si pelo delimitador ponto-e-vírgula. A forma geral é:

```
BEGIN
  <comando> ;
  ... ;
  <comando>
END
```

3.7. COMENTÁRIOS

Um comentário é usado para aumentar a clareza de um programa, embora não seja analisado pelo computador. Deve ser escrito entre chaves:

```
{ comentário }
```

EXEMPLO:

```
Program REAJUSTE_SALARIO;
  {Finalidade: Calc. o reajuste de um salario em 20%}

Uses
  Crt;

Const
  IND = 0.20; {indice do reajuste}

Var
  SAL_ATUAL {salario atual},
  SAL_NOVO {novo salario},
  AUMENTO {valor do aumento} : Real;

Begin

  clrscr; {limpa a tela}

  {leitura do salario atual}
  write('Digite o salario atual: ');
  readln(SAL_ATUAL);

  {calculo do reajuste}
  AUMENTO := SAL_ATUAL * IND;
  SAL_NOVO := SAL_ATUAL + AUMENTO;

  {exibicao do resultado}
  writeln('Novo Salario = ',SAL_NOVO:10:2);

  readkey;

End.
```

EXERCÍCIOS PROPOSTOS

P3.20. Qual o formato básico de um programa Pascal ?

P3.21. Quais as áreas que podem existir em um bloco do Pascal? Qual delas é obrigatória?

P3.22. Qual a finalidade das seguintes áreas:

- a) Declaração de Uso de Unidades
- b) Declaração de Constantes
- c) Declaração de Tipos
- d) Declaração de Variáveis
- e) Declaração de Procedimentos e Funções
- f) Área de Comandos

P3.23. Dê um exemplo para cada uma das áreas abaixo:

- a) Declaração de Uso de Unidades
- b) Declaração de Constantes
- c) Declaração de Variáveis

P3.24. Qual a finalidade de um comentário dentro de um programa? Como deve ser escrito?

Capítulo 4

COMANDOS BÁSICOS DA LINGUAGEM PASCAL

4.1. ATRIBUIÇÃO

O comando de atribuição tem a forma:

```
<identificador> := <expressão>
```

No comando de atribuição, a variável e a expressão devem ser do mesmo tipo, exceto nos seguintes casos:

- a) a variável sendo *real*, a expressão pode ser *integer*
- b) a variável sendo *string*, a expressão pode ser *char*

EXEMPLOS:

```
Var
  I : Integer;
  R : Real;
  S : String;
  C : Char;
Begin
  I := 5;
  R := I;
  C := 'A';
  S := C
End.
```

4.2. ENTRADA

Um comando de entrada serve para que o programa solicite dados no momento em que o mesmo está sendo executado. Esses dados fornecidos serão armazenados em variáveis na memória. Em geral a unidade de entrada é o teclado, podendo também ser uma memória auxiliar como o winchester.

Considerando a unidade de entrada padrão, o teclado, o comando seria:

```
READ (<identificador-1>,...<identificador-n>)
OU
READLN (<identificador-1>,...,<identificador-n>)
```

Com *READ* o cursor permanece na mesma linha após a execução do comando; com o *READLN* o cursor muda para a próxima linha.

Observação: No Turbo Pascal, o comando *READ* só deve ser utilizado para a leitura de arquivos. Portanto, para a leitura de variáveis, devemos sempre utilizar o comando *READLN*.

EXEMPLOS:

1) Se o programa deve solicitar as três notas de um aluno, teríamos:

```
readln (NOTA1,NOTA2,NOTA3); ...
```

No momento da execução do comando acima, o programa mostra a tela do usuário e o cursor aparece esperando a digitação dos três valores que devem ser separada por, pelo menos, um espaço em branco.

EXERCÍCIOS PROPOSTOS

- P4.01. Exemplifique o uso do comando de Atribuição.
- P4.02. Qual a finalidade de um comando de entrada ? Dê exemplos.
- P4.03. Qual a diferença entre os comandos READ e READLN ?
- P4.04. Qual a finalidade de um comando de saída? Dê exemplos.
- P4.05. Qual a diferença entre os comandos WRITE e WRITELN ?
- P4.06. Como podemos direcionar a saída de um programa para a impressora? Dê exemplos.
- P4.07. Como podemos orientar o usuário na digitação dos dados? Exemplifique.
- P4.08. Escreva os comandos necessários para:
- a) ler o nome de uma pessoa
 - b) ler as 3 notas de um aluno
 - c) ler o peso e altura de uma pessoa
- P4.09. Escreva um programa em Pascal para calcular e exibir o valor de x^y , sendo dados a base (x) e o expoente (y).
- P4.10. Escreva um programa para ler o nome e o sobrenome de uma pessoa e escrevê-los na seguinte forma: sobrenome seguido por uma vírgula e pelo nome.
- Exemplo:
- ```
entrada: "Antonio", "Soares"
saída: Soares, Antonio
```
- P4.11. Converta para a linguagem Pascal o algoritmo construído na questão P2.11.
- P4.12. Idem para a questão P2.12.
- P4.13. Idem para a questão P2.13.
- P4.14. Idem para a questão P2.14.
- P4.15. Idem para a questão P2.15.

## 4.4. COMANDOS DE DECISÃO

As estruturas de decisão (condicionais) são utilizadas para tomar uma decisão baseada no resultado da avaliação de uma condição de controle e seleciona uma ou mais ações possíveis (comandos) para serem executados pelo computador.

No Pascal, existe três tipos de estrutura de decisão: O comando IF, que pode ser utilizado de duas formas: simples ou composto; e o comando CASE, que é utilizado para uma decisão seletiva.

### 4.4.1. DECISÃO SIMPLES ( IF-THEN )

Utilizado quando se deseja executar uma ação (um comando ou uma seqüência de comandos) caso uma determinada condição seja verdadeira. A estrutura de decisão simples do Pascal é o **IF**, e deve ser utilizada da seguinte forma:

```
IF <condição> THEN <comando>
```

Neste caso, o <comando> só será executado se a <condição> resultar no valor TRUE.

A <condição> deve ser uma expressão lógica. O <comando> pode ser um comando simples ou um comando composto. Um comando composto é formado por dois ou mais comandos, separados por ponto-e-vírgula e delimitados por BEGIN e END.

EXEMPLO:

```
Program EXEMPLO_DE_DECISAO_SIMPLES;
 {Ler um número inteiro e exibí-lo se for positivo}

Var
 N : integer;

Begin
 readln(N);
 if N > 0 then
 writeln(N)
 End.
```

No exemplo acima, o comando WRITE só será executado se a condição  $N > 0$  for verdadeira. Caso contrário, nenhuma ação será executada.

#### 4.4.2. DECISÃO COMPOSTA ( IF-THEN-ELSE )

Utilizada quando se deseja executar um entre dois comandos (ou uma entre duas seqüências de comandos) dependendo do resultado de uma condição.

A estrutura de decisão composta do Pascal também é o **IF**, mas executado com a seguinte sintaxe:

```
IF <condição> THEN <comando1> ELSE <comando2>
```

Neste caso, se a <condição> resultar no valor TRUE, será executado o <comando1>; caso contrário, será executado o <comando2>. Também aqui, a <condição> deve ser uma expressão lógica, e <comando1> e <comando2> devem ser um comando simples ou um comando composto.

EXEMPLO:

```
Program EXEMPLO_DE_DECISAO_COMPOSTA;
 {Lê um número e determinar se é maior que zero ou não}

Var
 N : integer;

Begin
 readln(N);
 if N > 0 then
 writeln (N, ' é maior que zero')
 else
 writeln (N, ' não é maior que zero')
 End.
```

Neste exemplo, a mensagem que será exibida dependerá do resultado da expressão lógica  $N > 0$ . Se for verdadeira, será executado o comando WRITE que sucede a palavra THEN. Caso contrário, será executado o WRITE que sucede a palavra ELSE. Em nenhuma hipótese serão executados os dois comandos.

Em algoritmos mais complexos, é comum a utilização de IF's aninhados, ou seja, uma estrutura IF possuindo como <comando> uma outra estrutura IF.

## EXEMPLO:

```

Program EXEMPLO_DE_IFS_ANINHADOS;
 {Determinar se um número é maior, menor ou igual a zero}
Var
 N : integer;

Begin
 readln(N);
 if N > 0 then
 writeln (N, ' é maior que zero')
 else
 if N < 0 then
 writeln (N, ' é menor que zero')
 else
 writeln (N, ' é igual a zero')
 End.

```

Pode-se observar que diversas linhas deste programa terminaram sem o ponto-e-vírgula, isto porque o ponto-e-vírgula só é utilizado para separar comandos e/ou estruturas.

Deve-se tomar cuidado quando da utilização de IF's aninhados, pois a cláusula ELSE é sempre relacionada ao último IF. Se, dentro de algum programa, precisarmos contornar este fato, podemos fazê-lo com os delimitadores BEGIN e END.

## EXEMPLO:

| LINGUAGEM ALGORITMICA                                          | LINGUAGEM PASCAL                                                                          |
|----------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| se COND1 então<br>Se COND2 então<br>Comando1<br>senão Comando2 | if COND1 then<br><b>begin</b><br>if COND2 then<br>comando1<br><b>end</b><br>else Comando2 |

## 4.4.3. DECISÃO MÚLTIPLA ( CASE-OF )

Utilizada quando se deseja executar um entre vários comandos (ou uma entre várias seqüências de comandos) dependendo do resultado de uma expressão.

A estrutura de seleção (decisão múltipla) do Pascal é o CASE, e obedece a seguinte sintaxe:

```

CASE <expressão> OF
 <lista-de-constantes-1> : <comando-1>;
 <lista-de-constantes-2> : <comando-2>;
 ...
 [ELSE <comando-n>]
END

```

A <expressão> deve resultar um tipo escalar (outros tipos que não o REAL e o STRING). A <lista-de-constantes-x> devem conter uma ou mais constantes (separadas por vírgula), e devem ser do mesmo tipo da <expressão>. O <comando-x> pode ser um comando simples ou composto.

O resultado de <expressão> é comparado com cada constante da <lista-de-constante> para verificar igualdade. Caso a igualdade seja verificada, o <comando> correspondente é executado e a estrutura finalizada. Caso nenhuma igualdade seja verificada, o <comando> correspondente ao ELSE (optativo) será executado.

## EXEMPLO:

```

Program EXEMPLO_DE_DECISAO_MÚLTIPLA;
 {Simulador de uma calculadora básica de números inteiros}
Uses
 CRT;
Var
 X,Y : integer;
 OP : char;
Begin
 clrscr;
 write('Digite os operandos: ');
 readln(X,Y);
 write('Digite o operador: ');
 readln(OP);
 case OP of
 '+' : writeln(X + Y);
 '-' : writeln(X - Y);
 '*', 'x', 'X' : writeln(X * Y);
 '/' : writeln(X div Y);
 else writeln('operação inválida');
 end {case};
 readkey;
End.

```

Neste exemplo, a mensagem que será exibida dependerá do conteúdo da variável OP. Se for igual a uma das constantes especificadas, será executado o comando WRITELN correspondente. Se nenhuma constante for igual ao conteúdo de OP, será executado o WRITELN do ELSE.

Podemos também escrever o mesmo programa acima sem utilizar a estrutura CASE, apenas utilizando IF's aninhados.

## EXEMPLO:

```

Program EXEMPLO_DE_DECISAO_MÚLTIPLA_2;
 {Simulador de uma calculadora básica de números inteiros}
Uses
 CRT;
Var
 X,Y : integer;
 OP : char;
Begin
 clrscr;
 write('Digite os operandos: ');
 readln(X,Y);
 write('Digite o operador: ');
 readln(OP);
 if OP='+' then
 writeln(X + Y)
 else
 if OP='-' then
 writeln(X - Y)
 else
 if (OP='*') or (OP='x') or (OP='X') then
 writeln(X * Y)
 else
 if OP='/' then
 writeln(X div Y)
 else
 writeln('op.inválida');
 end;
 end;
 end;
 end;
 readkey;
End.

```

**EXERCÍCIOS PROPOSTOS**

P4.16. Quais são as estruturas de decisão existentes no Pascal ?

P4.17. Em que situações é mais indicado o uso da estrutura CASE-OF ?

P4.18. Em que situações não podemos utilizar a estrutura CASE-OF ?

P4.19. Os comandos (i) e (ii) são equivalentes? Explique sua resposta.

```
(i) A := B = C (ii) if B = C then
 A := TRUE
 Else
 A := FALSE
```

P4.20. Observe o programa abaixo:

```
Program NÓ_no_juizo;
Var L1,L2,L3 : char;
Begin
 readln(L1,L2,L3); {deve ser digitado apenas as letras V ou F}
 if L1='V' then
 write('A')
 else
 if L2='V' then
 if L3='V' then
 write('B')
 else
 begin
 write('C');
 write('D')
 end;
 write('E');
End.
```

Agora, responda as seguintes questões:

- Se forem lidos V, V e F, o que será escrito pelo programa?
- Se forem lidos F, V e F, o que será escrito pelo programa?
- Se forem lidos F, V e V, o que será escrito pelo programa?
- Que valores deveriam ser lidos para que fosse escrito apenas 'E'?

P4.21. Escreva um programa para ler um número inteiro positivo e determinar se ele é par ou ímpar.

P4.22. Converta o algoritmo construído na questão P2.17 para a linguagem Pascal.

P4.23. Idem para a questão P2.18.

P4.24. Idem para a questão P2.19.

P4.25. Idem para a questão P2.20.

P4.26. Idem para a questão P2.21.

P4.27. Idem para a questão P2.22.

P4.28. Idem para a questão P2.23.

P4.29. Idem para a questão P2.24.

P4.30. Idem para a questão P2.25, utilizando o comando CASE.

P4.31. Deseja-se calcular o imposto de renda de um contribuinte. Para isto, escreva um programa que:

- leia os seguintes dados do contribuinte: CPF, nome, rendimento anual, imposto retido na fonte, contribuição previdenciária, despesas médicas, número de dependentes;
- é deduzido o valor de R\$ 1.080,00 por cada dependente;
- cálculo do valor total das deduções: contribuição previdenciária + despesas médicas + dedução dos dependentes;
- cálculo da base de cálculo: rendimento anual – total das deduções;
- com base na tabela abaixo:

| Base de Cálculo            | Alíquota | Parcela a Deduzir |
|----------------------------|----------|-------------------|
| até 10.800,00              | Isento   | -                 |
| De 10.800,01 até 21.600,00 | 15%      | 1.620,00          |
| acima de 21.600,00         | 25%      | 3.780,00          |

Cálculo do imposto devido: (base de cálculo x alíquota) – parcela a deduzir

- haverá imposto a pagar se a diferença entre o imposto devido e o imposto retido na fonte for positiva; caso contrário, haverá imposto a restituir.
- exiba todos os dados lidos e calculados.

## 4.5. COMANDOS DE REPETIÇÃO

Além de permitir a mudança da seqüência de execução de um conjunto de comandos de um programa, as estruturas de controle de uma linguagem, dispõem de recursos para repetir a execução de um conjunto de comandos.

No Pascal, existe três tipos de estrutura de repetição: com teste no início (WHILE), com teste no final (REPEAT) e automática (FOR).

### 4.5.1. REPETIÇÃO COM TESTE NO INÍCIO ( WHILE-DO )

A estrutura de controle WHILE permite que um comando simples ou composto seja executado repetidamente, enquanto uma condição de controle seja VERDADEIRA. A forma geral do WHILE é:

```
WHILE <condição> DO <comando>
```

A <condição> deve ser uma expressão lógica. O <comando> pode ser um comando simples ou um comando composto.

Como o teste da <condição> é realizado no início do laço, o <comando> será executado zero ou mais vezes, dependendo da avaliação da <condição>.

EXEMPLO:

```
Program EXEMPLO_DE_WHILE; {escrever os números inteiros de 1 a 100}
Var
 N : integer;
Begin
 N := 1;
 while N <= 100 do
 begin
 writeln(N);
 N := N + 1
 end
End.
```

Neste exemplo, o comando WRITELN será executado repetidas vezes enquanto a variável N possuir um valor igual ou inferior a 100. O programa terá como saída a seqüência dos números inteiros de 1 a 100.

#### 4.5.2. REPETIÇÃO COM TESTE NO FINAL ( REPEAT-UNTIL )

A estrutura de controle REPEAT permite que um comando simples ou composto seja executado repetidamente até que uma condição de controle seja FALSA. A forma geral do REPEAT é:

```
REPEAT <comando> UNTIL <condição>
```

A <condição> deve ser uma expressão lógica. O <comando> pode ser um comando simples ou um comando composto. Não há a necessidade dos delimitadores BEGIN e END no comando composto em um REPEAT.

Como o teste da <condição> é realizado no final do laço, o <comando> será executado uma ou mais vezes, dependendo da avaliação da <condição>.

EXEMPLO:

```
Program EXEMPLO_DE_REPEAT; {escrever os núm. inteiros de 1 a 100}

Var
 N : Integer;

Begin
 N := 1;
 repeat
 writeln(N);
 N := N + 1
 until N > 100
End.
```

O exemplo anterior é equivalente ao exemplo do WHILE, onde o comando WRITELN será executado repetidas vezes até que a variável N possua um valor superior a 100.

A estrutura REPEAT também é bastante utilizada para repetirmos um programa diversas vezes, até que o usuário deseje sair do mesmo.

EXEMPLO:

```
Program EXEMPLO_DE_REPEAT_2;
 {calcula a média de 2 números dados repetidas vezes}

Uses
 CRT;

Var
 N1,N2,MEDIA : real;
 RESP : char;

Begin
 clrScr;
 repeat
 write('Digite os dois números: ');
 readln (N1,N2);
 MEDIA := (N1+N2)/2;
 writeln (MEDIA);
 write ('Deseja repetir o programa (s/n)? ');
 RESP := readkey;
 until (RESP='N') or (RESP='n')
End.
```

### 4.5.3. REPETIÇÃO AUTOMÁTICA ( FOR )

A estrutura de controle FOR permite que um comando simples ou composto seja repetido um número específico de vezes. A sua forma geral é:

```
FOR <var> := <vi> TO <vf> DO <comando>
```

Onde <var> é uma variável de controle, do tipo inteira, que assumirá inicialmente o valor inicial <vi> e será **incrementada** do valor 1 após cada repetição do laço. A repetição será finalizada quando o conteúdo de <var> for superior ao valor final <vf>. O <comando> também pode ser simples ou composto.

Uma outra forma da estrutura FOR é a seguinte:

```
FOR <var> := <vi> DOWNTO <vf> DO <comando>
```

Neste caso, a variável de controle <var> será **decrementada** do valor 1 após cada repetição do laço e a repetição será finalizada quando o conteúdo de <var> for inferior ao valor final <vf>.

EXEMPLO:

```
Program EXEMPLO_DE_FOR;
 {escreve os números inteiros de 1 a 100}
Var
 N : integer;
Begin
 for N := 1 to 100 do
 writeln(N)
 End.
```

Observe, no exemplo acima, que não foi necessário utilizar um comando para atribuir um valor inicial a variável N, nem também um outro comando para incrementá-la com o valor 1. Isto é feito automaticamente pela estrutura FOR.

A estrutura de repetição FOR é especialmente indicada para quando o número de repetições é previamente conhecido. Caso contrário, devemos utilizar o WHILE ou o REPEAT, dependendo do caso.

## EXERCÍCIOS RESOLVIDOS

R4.01. Escreva um programa que leia um conjunto 100 números inteiros e exiba o valor médio dos mesmos.

```
Program R4_01;

Var
 N,SOMA,CONT : integer;

Begin

 SOMA := 0;

 for CONT := 1 to 100 do
 begin
 readln(N)
 SOMA := SOMA + N;
 end;

 writeln(SOMA);

End.
```

R4.02. Escreva um programa que leia um conjunto de números inteiros e exiba o valor médio dos mesmos.

Obs: A condição de saída do laço será a leitura do valor 0 (flag).

```
Program MEDIA_NUMEROS;

Var
 N,CONT,SOMA,MEDIA : integer;

Begin

 SOMA := 0;
 CONT := 0;

 readln(N);

 while N <> 0 do
 begin
 SOMA := SOMA + N;
 CONT := CONT + 1;
 readln(N)
 end;

 MEDIA := SOMA div CONT;

 writeln(MEDIA);

End.
```

## EXERCÍCIOS PROPOSTOS

P4.32. Quais são as estruturas de repetição existentes no Pascal ?

P4.33. Qual a principal diferença entre o WHILE-DO e o REPEAT-UNTIL ?

P4.34. Em que situações é mais indicado o uso da estrutura FOR ?

P4.35. Em que situações não podemos utilizar a estrutura FOR ?

P4.36. Escreva um programa Pascal que leia um conjunto de 100 números inteiros positivos e determine a quantidade de números pares e números ímpares contidos no mesmo.

P4.37. Dado o trecho de programa abaixo:

```
readln(N)
R := 1;
I := 2;
while I <= N-1 do
 begin
 R := R * 2;
 I := I + 1;
 end;
write(R);
```

Reescreva-o utilizando:

- a) o comando FOR
- b) o comando REPEAT

P4.38. Converta para a linguagem Pascal o algoritmo construído na questão P2.28.

P4.39. Idem para a questão P2.29.

P4.40. Idem para a questão P2.30.

P4.41. Idem para a questão P2.31.

P4.42. Idem para a questão P2.32.

P4.43. Idem para a questão P2.33.

P4.44. Idem para a questão P2.34.

P4.45. Idem para a questão P2.35.

P4.46. Idem para a questão P2.36.

P4.47. Idem para a questão P2.37.

P4.48. Idem para a questão P2.38.

P4.49. Idem para a questão P2.39.

P4.50. Idem para a questão P2.40.

P4.51. Idem para a questão P2.41.

P4.52. Idem para a questão P2.42.

P4.53. Idem para a questão P2.43.

P4.54. Idem para a questão P2.44.

P4.55. Idem para a questão P2.45.

P4.56. Idem para a questão P2.46.

P4.57. Idem para a questão P2.47.

P4.58. Idem para a questão P2.48.

P4.59. Escreva um programa Pascal que apresente o menu de opções abaixo:

|                                                                     |
|---------------------------------------------------------------------|
| OPÇÕES:<br>1 - SAUDAÇÃO<br>2 - BRONCA<br>3 - FELICITAÇÃO<br>0 - FIM |
|---------------------------------------------------------------------|

O programa deve ler a opção do usuário e exibir, para cada opção, a respectiva mensagem:

|                                                                                              |
|----------------------------------------------------------------------------------------------|
| 1 - Olá. Como vai ?<br>2 - Vamos estudar mais.<br>3 - Meus Parabéns !<br>0 - Fim de serviço. |
|----------------------------------------------------------------------------------------------|

Enquanto a opção for diferente de 0 (zero) deve-se continuar apresentando as opções.

Obs: use como estrutura de repetição o comando REPEAT e como estrutura condicional o comando CASE.

P4.60. Faça um programa que leia 3 valores inteiros (N, X, Y) e mostre todos os números múltiplos de N entre X e Y.

P4.61. Um número é, por definição, primo se ele não tem divisores, exceto 1 e ele próprio. Escreva um programa que leia um número e determine se ele é ou não primo.

P4.62. Faça um programa que leia dois valores inteiros (X e Y) e mostre todos os números primos entre X e Y.

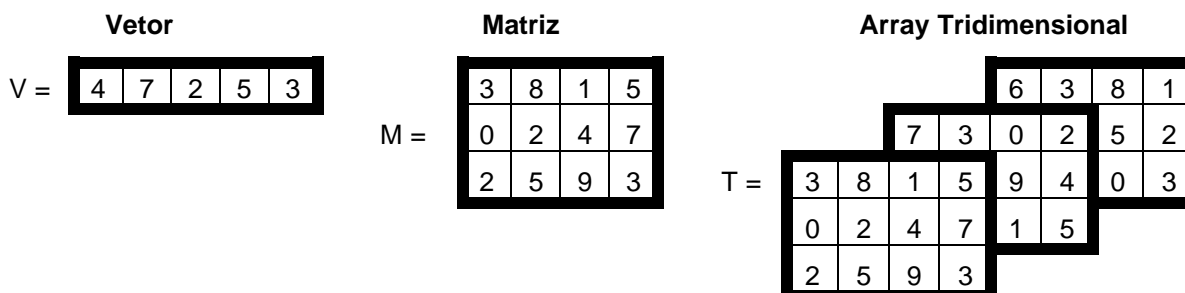
## Capítulo 5

# ARRAYS

As *variáveis compostas homogêneas*, mais conhecidas como **arrays**, correspondem a conjuntos de elementos de um mesmo tipo, representados por um único nome.

Os arrays podem variar quanto a sua dimensão, isto é, a quantidade de índices necessária para a individualização de cada elemento do conjunto. O array unidimensional também é conhecido por **vetor**, enquanto o array bidimensional é denominado de **matriz**.

EXEMPLOS:



Cada elemento dos arrays podem ser referenciados através de índices. Exemplos:

|            |              |                |
|------------|--------------|----------------|
| $V[1] = 4$ | $M[1,1] = 3$ | $T[1,1,1] = 3$ |
| $V[2] = 7$ | $M[2,3] = 4$ | $T[2,3,2] = 9$ |
| $V[5] = 3$ | $M[3,1] = 2$ | $T[1,2,3] = 3$ |

### 5.1. VETORES

Vetores são arrays que necessitam de apenas um índice para individualizar um elemento do conjunto.

**Declaração:**

Para definirmos uma variável do tipo vetor, utilizamos a seguinte sintaxe:

```
lista-de-identificadores : ARRAY[índice-inicial..índice-final] OF tipo
```

onde:

|                                 |                                                         |
|---------------------------------|---------------------------------------------------------|
| <i>lista-de-identificadores</i> | são os nomes das variáveis que se deseja declarar;      |
| <i>índice-inicial</i>           | é o limite inferior do intervalo de variação do índice; |
| <i>índice-final</i>             | é o limite superior do intervalo de variação do índice; |
| <i>tipo</i>                     | é o tipo dos componentes da variável                    |

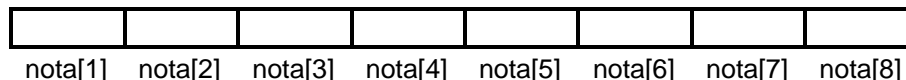
O **índice-inicial** e o **índice-final** devem ser do mesmo tipo escalar (inteiro, caracter ou booleano).

EXEMPLO:

Declarar uma variável composta de 8 elementos numéricos de nome NOTA.

```
var NOTA : array[1..8] of real;
```

fará com que passe a existir um conjunto de 8 elementos do tipo real, individualizáveis pelos índices 1, 2, 3, ..., 8.



Outros exemplos de declarações de vetores:

```
var IDADE : array[1..20] of integer;
 NOME : array[1..30] of string;
 QUANT : array['A'..'Z'] of integer;
 RECEITA, DESPESA : array[90..96] of real;
 VETOR : array[-5..5] of char;
```

Para processarmos individualmente todos os componentes de um vetor, é aconselhável o uso da estrutura FOR, para que possamos variar o índice do vetor.

## EXERCÍCIOS RESOLVIDOS

R5.01. Dado um vetor A definido com `A : array[1..100] of integer;`

a) preenchê-lo com o valor 30;

```
for I:=1 to 100 do
 A[I]:=30;
```

b) preenchê-lo com os números inteiros 1,2,3,...,100.

```
for I:=1 to 100 do
 A[I]:=I;
```

R5.02. Fazer um programa que leia um vetor A contendo 30 números inteiros, calcule e exiba:

a) o maior elemento;

b) a posição (índice) do maior elemento.

```
Program R5_02;

const
 N = 30; {número de elementos do vetor}

var
 A : array[1..N] of integer;
 I, POS, MAIOR : integer;

begin
 {leitura do vetor}
 for I:=1 to N do
 readln(A[I]);
 {inicialização das variáveis}
 MAIOR := A[1];
 POS := 1;
 {comparação dos elementos com a variável MAIOR}
 for I:=2 to N do
 if A[I] > MAIOR then
 begin
 MAIOR := A[I];
 POS := I;
 end;
 {exibição dos resultados}
 writeln(MAIOR, POS);
end.
```

R5.03. Fazer um programa para ler 20 números, calcular a média dos mesmos e exibir os números que estiverem acima da média.

```
Program R5_03;

const
 N = 20; {número de elementos do vetor}

var
 NUM : array[1..N] of real;
 I,SOMA : integer;
 MEDIA : real;

begin
 {inicialização da variável SOMA}
 SOMA := 0;
 {leitura e soma dos números}
 for I:=1 to N do
 begin
 readln(NUM[I]);
 SOMA := SOMA + NUM[I];
 end;
 {cálculo da média}
 MEDIA := SOMA / N;
 {exibição dos números maiores que a média}
 for I:=1 to N do
 if NUM[I] > MEDIA then
 writeln(NUM[I]);
 end.
```

R5.04. Fazer um programa que:

- a) leia um vetor VET de 100 números inteiros;
- b) leia um valor inteiro NUM;
- c) determine e exiba a posição de NUM dentro de VET. Caso NUM não seja encontrado dentro de VET, exiba o valor 0 (zero).

```
Program R5_04;

const
 N = 100; {número de elementos do vetor}

var
 VET : array[1..N] of integer;
 NUM,I,POS : integer;

begin
 {leitura dos dados}
 for I:=1 to N do
 readln(VET[I]);
 readln(NUM);
 {determinação da posição}
 POS := 0;
 for I:=1 to N do
 if VET[I] = NUM then
 POS := I;
 {exibição do resultado}
 writeln(POS);
end.
```

**EXERCÍCIOS PROPOSTOS**

P5.01. Defina com suas palavras e exemplifique:

- Array
- Vetor
- Matriz
- Índice
- Elemento ou Componente

P5.02. Dar o número de elementos de cada um dos vetores dados abaixo:

- VET : array[-5..5] of integer;
- NOME: array[0..20] of string;
- CONT: array['A'..'Z'] of integer;
- NOTA: array[1..50] of real;

P5.03. Dado o seguinte vetor:

|       |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|
|       | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| VET = | 7 | 4 | 1 | 5 | 8 | 2 | 3 | 6 |

qual será o seu conteúdo após a execução dos seguintes comandos:

```
for I:= 8 downto 5 do
 begin
 AUX := VET[I];
 VET[I] := VET[8-I+1];
 VET[8-I+1] := AUX;
 end;
```

P5.04. Dado um vetor A contendo 100 elementos inteiros, gerar e exibir um vetor B cujos elementos estão na ordem inversa de A.

Exemplo:

|     |    |    |     |    |     |
|-----|----|----|-----|----|-----|
|     | 1  | 2  |     | 99 | 100 |
| A = | 23 | 37 | ... | 20 | 26  |
| B = | 26 | 20 | ... | 37 | 23  |

P5.05. Dado dois vetores A e B contendo 20 elementos inteiros cada, gerar e exibir um vetor C do mesmo tamanho cujos elementos sejam a soma dos respectivos elementos de A e B.

Exemplo:

|     |    |    |    |     |    |    |
|-----|----|----|----|-----|----|----|
|     | 1  | 2  | 3  |     | 19 | 20 |
| A = | 23 | 37 | 30 | ... | 45 | 35 |
| B = | 30 | 32 | 46 | ... | 33 | 42 |
| C = | 53 | 69 | 76 | ... | 88 | 77 |

P5.06. Dado dois vetores A e B contendo 25 elementos inteiros cada, gerar e exibir um vetor C de 50 elementos, cujos elementos sejam a intercalação dos elementos de A e B.

Exemplo:

|     |    |    |    |     |    |    |     |    |    |    |    |
|-----|----|----|----|-----|----|----|-----|----|----|----|----|
|     | 1  | 2  | 3  |     | 24 | 25 |     |    |    |    |    |
| A = | 23 | 37 | 30 | ... | 38 | 55 |     |    |    |    |    |
| B = | 30 | 32 | 46 | ... | 43 | 49 |     |    |    |    |    |
| C = | 23 | 30 | 37 | 32  | 30 | 46 | ... | 38 | 43 | 55 | 49 |

P5.07. Um time de basquete possui 12 jogadores. Deseja-se um programa que, dado o nome e a altura dos jogadores, determine:

- a) o nome e a altura do jogador mais alto;
- b) a média de altura do time;
- c) a quantidade de jogadores com altura superior a média, listando o nome e a altura de cada um.

P5.08. Fazer um programa em Pascal para corrigir provas de múltipla escolha. Cada prova tem 10 questões e cada questão vale 1 ponto. O primeiro conjunto de dados a ser lido será o gabarito para a correção da prova. Os outros dados serão os números dos alunos e suas respectivas respostas, e o último número, do aluno fictício, será 0 (zero). O programa deverá calcular e imprimir:

- a) para cada aluno, o seu número e sua nota;
- b) o percentual de aprovação, sabendo-se que a nota mínima de aprovação é 6.
- c) a nota que teve maior frequência absoluta, ou seja, a nota que apareceu maior número de vezes (supondo a inexistência de empates).

A estrutura de dados para este programa de ser a seguinte:

|                      |                      |                      |
|----------------------|----------------------|----------------------|
| GABARITO             | NUMERO               | NOTA                 |
| <input type="text"/> | <input type="text"/> | <input type="text"/> |
| RESPOSTAS            | APROVADOS            | TOTAL                |
| <input type="text"/> | <input type="text"/> | <input type="text"/> |
| FREQUENCIA           | MAIOR                | PORCENT              |
| <input type="text"/> | <input type="text"/> | <input type="text"/> |

## 5.2. MATRIZES

Matrizes são arrays que necessitam de dois índices para individualizar um elemento do conjunto. O primeiro índice representa as linhas e o segundo as colunas.

### Declaração:

Para definirmos uma variável do tipo matriz, utilizamos a seguinte sintaxe:

```
lista-de-identificadores : ARRAY [índice1-inicial..índice1-final,
 índice2-inicial..índice2-final]
 OF tipo
```

onde:

- lista-de-identificadores* são os nomes das variáveis que se deseja declarar;
- índice1-inicial* é o limite inferior do intervalo de variação do primeiro índice;
- índice1-final* é o limite superior do intervalo de variação do primeiro índice;
- índice2-inicial* é o limite inferior do intervalo de variação do segundo índice;
- índice2-final* é o limite superior do intervalo de variação do segundo índice;
- tipo* é o tipo dos componentes da variável

o *índice1-inicial* e o *índice1-final* devem ser do mesmo tipo escalar (inteiro, caracter ou booleano). O *índice2-inicial* também deve ser do mesmo tipo escalar do *índice2-final*.

### EXEMPLO:

Declarar uma matriz M, de 4 linhas por 3 colunas, constituída de elementos numéricos inteiros.

```
var M : array[1..4,1..3] of integer;
```

fará com que passe a existir uma estrutura de dados agrupada denominada M, com  $4 \times 3 = 12$  elementos inteiros, endereçáveis por um par de índices, com o primeiro indicando a linha e o outro, a coluna.

$$M = \begin{array}{|c|c|c|} \hline m_{11} & m_{12} & m_{13} \\ \hline m_{21} & m_{22} & m_{23} \\ \hline m_{31} & m_{32} & m_{33} \\ \hline m_{41} & m_{42} & m_{43} \\ \hline \end{array}$$

Outros exemplos de declarações de matrizes:

```
var
 M1 : array[1..4,80..90] of real;
 M2 : array['A'..'E',0..10] of string;
 M3 : array[-3..3,1..3] of char;
```

## EXERCÍCIOS RESOLVIDOS

R5.05. Fazer um programa para ler uma matriz 3 x 5 de números inteiros e escrevê-la após ter multiplicado cada elemento por 2.

```
Program R5_05;

const
 NL = 3; {número de linhas}
 NC = 5; {número de colunas}
 K = 2; {fator para multiplicação}

var
 M : array[1..NL,1..NC] of integer;
 I,J : integer;

begin
 {leitura da matriz}
 for I:=1 to NL do
 for J:=1 to NC do
 begin
 write('Elemento da linha ',I,' coluna ',J,' : ');
 readln(M[I,J]);
 end;
 end;
 end;

 {cálculo da multiplicação}
 for I:=1 to NL do
 for J:=1 to NC do
 M[I,J] := M[I,J] * K;
 end;
 end;

 {exibição da matriz resultante}
 writeln('Resultado:');
 for I:=1 to NL do
 begin
 for J:=1 to NC do
 write(M[I,J], ' ');
 end;
 writeln;
 end;
 end;

end.
```

R5.06. Dada uma matriz de 4 x 5 elementos inteiros, calcular a soma de cada linha, de cada coluna e de todos os seus elementos.

Obs: utilize um vetor para armazenar o resultado da soma de cada linha e outro para a soma de cada coluna.

Program R5\_06;

```

const
 NL = 4; {número de linhas}
 NC = 5; {número de colunas}

var
 M : array[1..NL,1..NC] of integer;
 L : array[1..NL] of integer;
 C : array[1..NC] of integer;
 I,J,SOMA : integer;

begin
 {leitura da matriz}
 for I:=1 to NL do
 for J:=1 to NC do
 begin
 write('Elemento da linha ',I,' coluna ',J,' : ');
 readln(M[I,J]);
 end;
 {cálculo da soma dos elementos de cada linha}
 for I:=1 to NL do
 begin
 L[I] := 0;
 for J:=1 to NC do
 L[I] := L[I] + M[I,J];
 end;
 {cálculo da soma dos elementos de cada coluna}
 for J:=1 to NC do
 begin
 C[J] := 0;
 for I:=1 to NL do
 C[J] := C[J] + M[I,J];
 end;
 {cálculo da soma de todos os elementos da matriz}
 SOMA := 0;
 for I:=1 to NL do
 for J:=1 to NC do
 SOMA := SOMA + M[I,J];
 {exibição dos resultados}
 for I:=1 to NL do
 writeln('Soma da Linha ',I,' : ',L[I]);
 for J:=1 to NC do
 writeln('Soma da Coluna ',J,' : ',C[J]);
 writeln('Soma da Matriz: ',SOMA);
end.

```

### 5.3. ARRAYS MULTIDIMENSIONAIS

O Pascal permite a criação de arrays multidimensionais, que necessitam de vários índices para serem manipulados. A maneira de utilizarmos este tipo de array segue a mesma lógica das matrizes, diferenciando apenas no número de índices.

**EXERCÍCIOS PROPOSTOS**

P5.09. Dar o número de elementos de cada uma das matrizes abaixo dados abaixo:

- MAT : array[1..3,1..4] of integer;
- CONJ: array[0..2,1..3] of string;
- TAB : array['A'..'E',-1..1] of integer;
- NOTA: array[90..98,0..1] of real;

P5.10. Dado as matrizes M e R abaixo:

|     |                                                                                                                                                                                                                                                                                                                   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M = | <table border="1" style="display: inline-table; border-collapse: collapse; text-align: center;"> <tr><td>O</td><td>Q</td><td>*</td><td>I</td></tr> <tr><td>E</td><td>*</td><td>E</td><td>S</td></tr> <tr><td>R</td><td>E</td><td>U</td><td>T</td></tr> <tr><td>A</td><td>*</td><td>*</td><td>S</td></tr> </table> | O | Q | * | I | E | * | E | S | R | E | U | T | A | * | * | S |
| O   | Q                                                                                                                                                                                                                                                                                                                 | * | I |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| E   | *                                                                                                                                                                                                                                                                                                                 | E | S |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| R   | E                                                                                                                                                                                                                                                                                                                 | U | T |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| A   | *                                                                                                                                                                                                                                                                                                                 | * | S |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

|     |                                                                                                                                                                                                                                                                                                                          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| R = | <table border="1" style="display: inline-table; border-collapse: collapse; width: 80px; height: 40px;"> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> </table> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|     |                                                                                                                                                                                                                                                                                                                          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|     |                                                                                                                                                                                                                                                                                                                          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|     |                                                                                                                                                                                                                                                                                                                          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|     |                                                                                                                                                                                                                                                                                                                          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

qual será o conteúdo de R depois de executado os comandos:

```
for I:= 1 to 4 do
 for J:=1 to 4 do
 R[J,I] := M[I,J];
AUX := R[1,1];
R[1,1] := R[4,4];
R[4,4] := AUX;
AUX := R[2,2];
R[2,2] := R[3,3];
R[3,3] := AUX;
```

P5.11. Dado duas matrizes A e B, com 2 x 3 elementos inteiros cada, gerar e exibir uma matriz C do mesmo tamanho que resultará da soma da matriz A com a matriz B.

P5.12. Faça um programa que leia uma matriz de ordem 3 x 5 de elementos inteiros, calcular e exibir:

- o maior elemento da matriz;
- a soma dos elementos da matriz;
- a média dos elementos da matriz;

P5.13. Dado uma matriz quadrada de ordem N, de elementos inteiros, exibir os elementos da diagonal principal, isto é, os elementos onde  $i = j$ . Obs: N será lido ( $N \leq 10$ ).

P5.14. Dado uma matriz A com 3 x 4 elementos inteiros, gerar e exibir uma matriz B que será a matriz transposta de A.

P5.15. Faça um programa que leia o nome e as 3 notas dos 50 alunos de uma turma e:

- calcule:
    - a média aritmética de cada aluno;
    - a situação de cada aluno; (aprovado se média superior ou igual a 7.0)
    - o número de alunos aprovados;
    - a média geral da turma;
  - exiba:
    - o nome e a situação de cada aluno;
    - o número de alunos aprovados;
    - a média geral da turma;
    - o nome e a média dos alunos com média superior ou igual à média geral da turma.
- ⇒ Use vetores para armazenar nome, média e situação, e uma matriz para armazenar as notas.

P5.16. A tabela abaixo demonstra a quantidade de vendas dos fabricantes de veículos durante o período de 1993 a 1998, em mil unidades.

| Fabricante / Ano | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 |
|------------------|------|------|------|------|------|------|
| Fiat             | 204  | 223  | 230  | 257  | 290  | 322  |
| Ford             | 195  | 192  | 198  | 203  | 208  | 228  |
| GM               | 220  | 222  | 217  | 231  | 245  | 280  |
| Wolkswagen       | 254  | 262  | 270  | 284  | 296  | 330  |

Faça um programa que:

- lea os dados da tabela;
- determine e exiba o fabricante que mais vendeu em 1996;
- determine e exiba o ano de maior volume geral de vendas.
- determine e exiba a média anual de vendas de cada fabricante durante o período.

P5.17. Faça um programa que monte um array tridimensional 5 x 7 x 3, onde o conteúdo de cada elemento é igual a soma de seus índices.

P5.18. Faça um programa que leia e armazene em um array tridimensional contendo os valores do faturamento anual de uma empresa, especificados mês a mês e também por filial. Veja a estrutura do array abaixo:

|              |          |          |          |       | ANO DE 1997 |  |       |
|--------------|----------|----------|----------|-------|-------------|--|-------|
|              |          |          |          |       | ANO DE 1996 |  | TOTAL |
|              |          |          |          |       | ANO DE 1995 |  | TOTAL |
| ANO DE 1994  |          |          |          |       | TOTAL       |  |       |
| MESES        | FILIAL 1 | FILIAL 2 | FILIAL 3 | TOTAL |             |  |       |
| Janeiro      |          |          |          |       |             |  |       |
| Fevereiro    |          |          |          |       |             |  |       |
| Março        |          |          |          |       |             |  |       |
| Abril        |          |          |          |       |             |  |       |
| Mai          |          |          |          |       |             |  |       |
| Junho        |          |          |          |       |             |  |       |
| Julho        |          |          |          |       |             |  |       |
| Agosto       |          |          |          |       |             |  |       |
| Setembro     |          |          |          |       |             |  |       |
| Outubro      |          |          |          |       |             |  |       |
| Novembro     |          |          |          |       |             |  |       |
| Dezembro     |          |          |          |       |             |  |       |
| <b>TOTAL</b> |          |          |          |       |             |  |       |

Após a leitura dos dados faça o seguinte:

- calcule os totais das linhas e das colunas em cada ano;
- crie uma nova página contendo a consolidação dos dados, isto é, a soma de todos os anos (mês a mês e por filial);
- exiba todos os dados lidos e calculados.

## Capítulo 6

# MODULARIZAÇÃO

A modularização consiste num método utilizado para facilitar a construção de grandes programas, através de sua divisão em pequenas etapas, que são os **módulos** ou **subprogramas**. A primeira delas, por onde começa a execução do trabalho, recebe o nome de **programa principal**, e as outras são os **subprogramas** propriamente ditos, que são executados sempre que ocorre uma chamada dos mesmos, o que é feito através da especificação de seus nomes.

### Vantagens da utilização de subprogramas:

- Economia de código: escreve-se menos;
- Desenvolvimento modularizado: pensa-se no algoritmo por partes;
- Facilidade de depuração (correção/acompanhamento): é mais fácil corrigir/detectar um erro apenas uma vez do que dez vezes;
- Facilidade de alteração do código: se é preciso alterar, altera-se apenas uma vez;
- Generalidade de código com o uso de parâmetros: escreve-se algoritmos para situações genéricas.

Há duas espécies de subprogramas: **PROCEDIMENTO** e **FUNÇÃO**.

### 6.1. PROCEDIMENTO

Um subprograma do tipo PROCEDIMENTO é, na realidade, um programa com vida própria, mas que, para ser processado, tem que ser solicitado pelo programa principal que o contém, ou por outro subprograma, ou por ele mesmo.

#### Declaração:

```
PROCEDURE nome;
 declaração dos objetos locais ao Procedimento
BEGIN
 comandos do Procedimento
END;
```

onde: **nome** é o identificador associado ao procedimento.

#### EXEMPLO:

O programa abaixo calcula a média aritmética entre 2 notas, sem o uso de procedimentos.

```
Program CALCULA_MÉDIA; {sem o uso de procedimentos}

var
 NOTA1,NOTA2,MEDIA : real;

begin
 {lê as notas}
 write('Digite a primeira nota: ');
 readln(NOTA1);
 write('Digite a segunda nota: ');
 readln(NOTA2);
 {calcula a media}
 MEDIA := (NOTA1 + NOTA2) / 2;
 {escreve o resultado}
 writeln('Media = ',MEDIA,4:1)
end.
```

Mostraremos agora o mesmo programa, utilizando um procedimento.

```

Program CALCULA_MÉDIA; {usando procedimento}
var
 NOTA1,NOTA2,MEDIA : real;

{declaração do procedimento}
procedure LER_NOTAS;
begin
 write('Digite a primeira nota: ');
 readln(NOTA1);
 write('Digite a segunda nota: ');
 readln(NOTA2);
end;

{Programa Principal}
begin
 LER_NOTAS; {ativação do procedimento LER_NOTAS}
 MEDIA := (NOTA1 + NOTA2) / 2; {calcula a media}
 Writeln('Media = ',MEDIA,4:1) {escreve o resultado}
end.

```

## 6.2. FUNÇÃO

As funções, embora bastante semelhantes aos procedimentos, têm a característica especial de retornar ao programa que as chamou um valor associado ao nome da função. Esta característica permite uma analogia com o conceito de função da Matemática.

### Declaração:

```

FUNCTION nome : tipo;
 declaração dos objetos locais à Função
BEGIN
 Comandos da Função
END;

```

onde: **nome** é o identificador associado à função.  
**tipo** é o tipo da função, ou seja, o tipo do valor de retorno.

### EXEMPLO:

O programa abaixo calcula a média dos elementos de um vetor, sem uso de Procedimentos ou Funções.

```

Program SOMA_VETOR; {sem o uso de procedimentos ou funções}
const N = 30;
var VETOR : array[1..N] of integer;
 I,SOMA,MEDIA : integer;
begin
 {lê os valores do vetor}
 for I:=1 to N do
 readln(VETOR[I]);
 {calcula a media}
 SOMA := 0;
 for I:=1 to N do
 SOMA := SOMA + VETOR[I];
 MEDIA := SOMA div N;
 {escreve o resultado}
 writeln(MEDIA)
end.

```

Mostraremos agora o mesmo programa, utilizando um procedimento para ler os valores do vetor e uma função para efetuar o cálculo da média.

```

Program SOMA_VETOR; {usando uma função e um procedimento}

const
 N = 30;

var
 VETOR : array[1..N] of integer;

{declaração do procedimento}
procedure LER_DADOS;
 var I : integer;
 begin
 for I:=1 to N do
 readln(VETOR[I]);
 end;

{declaração da função}
function MEDIA : integer;
 var I,SOMA : integer;
 begin
 SOMA := 0;
 for I:=1 to N do
 SOMA := SOMA + VETOR[I];
 MEDIA := SOMA div N;
 end;

{Programa Principal}
begin
 {ativa o procedimento LER_DADOS}
 LER_DADOS;
 {escreve o resultado, chamando a função MEDIA}
 writeln(MEDIA)
end.

```

### 6.3. VARIÁVEIS GLOBAIS E VARIÁVEIS LOCAIS

Observe que, no exemplo anterior, declaramos uma variável no programa principal e outras nos subprogramas. Podemos dizer que a variável VETOR, que foi declarada no programa principal é uma **variável global** aos subprogramas, enquanto que a variável I é dita **variável local** ao procedimento LER\_DADOS e as variáveis I e SOMA são locais à função MEDIA. É importante ressaltar que a variável I do procedimento LER\_DADOS é diferente da variável I da função MEDIA, embora possuam o mesmo identificador.

O uso de *variáveis globais* dentro de procedimentos e funções serve para implementar um mecanismo de transmissão de informações de um nível mais externo para um mais interno.

As *variáveis locais* dos procedimentos e funções são criadas e alocadas quando da sua ativação e automaticamente liberadas quando do seu término.

A utilização de *variáveis globais* não constitui, no entanto, uma boa prática de programação. Assim, todos subprogramas devem apenas utilizar as *variáveis locais*, conhecidas dentro dos mesmos, e a transmissão de informações para dentro e fora dos subprogramas deve ser feita através dos **parâmetros** de transmissão, que serão apresentados a seguir.

## 6.4. PARÂMETROS

Quando se deseja escrever um subprograma que seja o mais genérico possível, deve-se usar a passagem de parâmetros.

A passagem de parâmetros formaliza a comunicação entre os módulos. Além disto, permite que um módulo seja utilizado com operandos diferentes, dependendo do que se deseja do mesmo.

Dá-se a designação de **parâmetro real** ou **de chamada** ao objeto utilizado na unidade chamadora/ativadora e de **parâmetro formal** ou **de definição** ao recebido como parâmetro no subprograma.

Dentre os modos de passagem de parâmetros, podemos destacar a **passagem por valor** e a **passagem por referência**.

Na passagem de parâmetros **por valor**, as alterações feitas nos parâmetros formais, dentro do subprograma, não se refletem nos parâmetros reais. O valor do parâmetro real é copiado no parâmetro formal, na chamada do subprograma. Assim, quando a passagem é por valor, isto significa que o parâmetro é de **entrada**.

Na passagem de parâmetros **por referência**, a toda alteração feita num parâmetro formal corresponde a mesma alteração feita no seu parâmetro real associado. Assim, quando a passagem é por referência, isto significa que o parâmetro é de **entrada-saída**.

Na linguagem Pascal, a declaração dos procedimentos e funções com parâmetros se diferencia da forma já apresentada apenas pela inclusão da lista de parâmetros formais no cabeçalho. Esta deve vir entre parênteses e cada parâmetro deve ter o seu tipo especificado. A forma geral é:

```
PROCEDURE nome (lista de parâmetros formais)

FUNCTION nome (lista de parâmetros formais) : tipo
```

A **lista de parâmetros formais** tem a seguinte forma:

```
parâmetro1 : tipo; parâmetro2 : tipo; ...; parâmetro n : tipo
```

Exemplos da lista de parâmetros:

```
procedure PROC (X,Y,Z:integer; K:real)

function FUNC (A,B:real; C:string) : integer
```

Na forma apresentada, a passagem dos parâmetros será por valor. Para se utilizar a passagem por referência, deve-se acrescentar a palavra **VAR** antes do nome do parâmetro.

EXEMPLO:

```
Procedure PROC(A:integer; var B,C:integer)
```

Na chamada de procedimentos ou funções utilizando parâmetros, devemos acrescentar após o nome do procedimento ou função uma lista de parâmetros reais (de chamada), os quais devem ser do mesmo tipo e quantidade dos parâmetros formais declarados.

O exemplo a seguir demonstra a diferença entre a passagem de parâmetros por referência e a passagem de parâmetros por valor:

```
Program EXEMPLO_PASSAGEM_PARÂMETROS;

var N1,N2 : integer;

Procedure PROC(X:integer; var Y:integer);
begin
 X:=1;
 Y:=1;
end;

begin
 N1:=0; N2:=0;
 PROC(N1,N2);
 writeln(N1); {será exibido o valor 0}
 writeln(N2); {será exibido o valor 1}
end.
```

## EXERCÍCIOS RESOLVIDOS

R6.01. Escrever uma função chamada MAIOR que receba dois números inteiros e retorne o maior deles. Escrever um programa para ler dois números inteiros e, utilizando a função MAIOR, calcular e exibir o maior valor entre os números lidos.

```
Program CALCULA_MAIOR;

var X,Y,M : integer;

function MAIOR (NUM1,NUM2:integer) : integer;
begin
 If NUM1 > NUM2 then
 MAIOR := NUM1
 else
 MAIOR := NUM2;
end;

begin
 readln(X,Y);
 M := MAIOR(X,Y);
 writeln(M);
end.
```

R6.02. Escrever um procedimento chamado DOBRA que multiplique um número inteiro (recebido como parâmetro) por 2. Escrever um programa para ler um valor inteiro e, utilizando o procedimento DOBRA, calcular e exibir o dobro do mesmo.

```
Program CALCULA_DOBRO;

var X : integer;

procedure DOBRA (var NUM:integer);
begin
 NUM := NUM * 2
end;

begin
 readln(X);
 DOBRA(X);
 writeln(X);
end.
```

## EXERCÍCIOS PROPOSTOS

- P6.01. Defina modularização.
- P6.02. Cite as principais vantagens da utilização de subprogramas.
- P6.03. Conceitue procedimento e função. Em que eles são semelhantes e em que eles são diferentes?
- P6.04. Que tipo de informação deve ser incluído na declaração de um procedimento? E na declaração de uma função? Se houver diferenças, explique o motivo.
- P6.05. Qual a diferença entre variável global e variável local?
- P6.06. Como deve ser feita a transmissão de informações entre um subprograma e o programa principal?
- P6.07. Qual a diferença entre parâmetro real e parâmetro formal?
- P6.08. Cite os modos de passagem de parâmetros, explicando como funciona cada um deles.
- P6.09. Escreva um procedimento que limpe a tela do micro e exiba o seu nome.
- P6.10. Escreva um procedimento que receba um valor string S e um valor inteiro positivo N e exiba o string S por N vezes seguidas na tela.
- P6.11. Escreva uma função chamada CUBO que receba um valor do tipo real e retorne a potência elevado a 3 do mesmo.
- P6.12. Escreva um procedimento chamado TROCA que receba 2 variáveis inteiras (X e Y) e troque o conteúdo entre elas;
- P6.13. Escreva um procedimento chamado SINAL que receba como parâmetro um valor N inteiro e escreva a palavra POSITIVO se N for um número maior que zero, NEGATIVO se N for menor que zero, ou ZERO se N for igual a zero.
- Escreva um programa que leia um número inteiro e, usando o procedimento SINAL, mostre se ele é maior, menor ou igual a zero.
- P6.14. Escreva um procedimento chamado METADE que divida um valor do tipo real (passado como parâmetro) pela metade.
- Escreva um programa que leia um vetor A de 30 elementos reais e, usando o procedimento METADE, divida todos seus elementos pela metade.
- P6.15. Escreva uma função chamada MEDIA que retorne a média de 3 valores reais (X, Y e Z) passados como parâmetros.
- Escreva um programa que, para um número indeterminado de alunos, faça para cada uma deles:
- ⇒ ler o nome e as 3 notas do aluno (a leitura do nome FIM indica o fim dos dados - flag);
  - ⇒ calcule a média do aluno (usando a função MEDIA);
  - ⇒ exiba o nome e a média do aluno.
- P6.16. Escreva um procedimento chamado AUMENTO que receba dois valores reais X e Y como parâmetros e aumente o valor de X em Y%.
- Escreva um programa que leia uma variável K do tipo real e, para um número indeterminado de funcionários de uma empresa, faça para cada uma delas:
- ⇒ ler a matrícula, o nome e o salário (a leitura da matrícula 0 (zero) indica o fim dos dados - flag);
  - ⇒ aumente o salário em K% (usando o procedimento AUMENTO) e exiba o salário aumentado.

P6.17. Escreva um programa Pascal que leia as 3 notas e o número de faltas de um aluno, calcule a sua média e determine e exiba a sua situação. Caso a aluno tenha mais de 10 faltas, ele está REPROVADO POR FALTA. Caso contrário, estará REPROVADO se sua média for menor que 5.0 ou APROVADO se sua média for superior a 5.0.

Observações:

- a) utilize uma função para calcular a média e um procedimento para determinar e exibir a situação do aluno;
- b) não utilize variáveis globais.

P6.18. Escreva um programa em Pascal que calcule o valor do coseno de X através de 20 termos da série abaixo:

$$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

Observações:

- a) O valor de x será lido;
- b) Deve ser implementado em funções independentes o cálculo do fatorial e o cálculo das potências.
- c) Utilize, como resultado do fatorial e da potência, o tipo EXTENDED, que é um tipo real, mas que permite valores muito grandes. Porém, inclua no início de seu programa (antes da cláusula USES) a diretiva **{N+}**.

P6.19. Escreva uma função chamada SEG para receber uma medida de tempo expressa em Horas, Minutos e Segundos e retornar esta medida convertida apenas para segundos.

Escreva um procedimento chamado HMS para receber uma medida de tempo expressa apenas em segundos em retornar esta medida convertida para horas, minutos e segundos.

Faça um programa que leia 2 medidas de tempo (expressas em horas, minutos e segundos) e, usando a função SEG e o procedimento HMS, calcule e exiba a diferença (também em horas, minutos e segundos) entre elas.

P6.20. Escreva uma função chamada NOME\_MES que receba um valor inteiro N (de 1 a 12) e retorne um string contendo o nome do mês correspondente a N.

Faça um programa que leia uma data (no formato dia, mês e ano) e, usando a função NOME\_MES, exiba a data lida no formato abaixo:

EXEMPLO:    Entrada: 23 11 1998                    Saída: 23 de novembro de 1998

P6.21. Escreva uma função chamada DIAS\_ANO que receba 3 valores inteiros (DIA, MES, ANO) e retorne o número de dias decorridos no ano até o dia/mês/ano fornecido.

Escreva um função booleana chamada DATA\_VALIDA que receba uma data (DIA, MÊS, ANO) e verifique se a data é válida (considerando os anos bissextos).

Faça um programa que leia 2 datas, no formato dia, mês e ano (as datas devem ter o mesmo ano) verificando se as mesmas são válidas (através da função DATA\_VALIDA), calcule e exiba a diferença de dias entre elas (usando a função DIAS\_ANO).

## 6.5. UTILIZANDO ARRAYS COMO PARÂMETROS

O Turbo Pascal não permite a declaração normal de um array como parâmetro formal em uma função ou procedimento. Porém, podemos utilizar o recurso da criação de novos tipos de dados disponível na linguagem Pascal. Veja o exemplo a seguir:

```
type VETOR = array[1..20] of integer;
```

Essa declaração deve ser colocada na área de declarações do programa, preferencialmente antes da área de declarações de variáveis (var). Após essa declaração, caso você necessite declarar uma variável do tipo array com 20 elementos inteiros, basta fazer o seguinte:

```
var V : VETOR;
```

Desta forma, para utilizar um array como parâmetro formal em uma função ou procedimento, você deve declará-lo na lista de parâmetros com o nome do tipo criado na declaração **type**.

## EXERCÍCIO RESOLVIDO

R6.03. Escrever um programa para ler um vetor de 50 elementos inteiros e determinar o valor médio dos seus elementos (utilizando um procedimento para ler um vetor e uma função para calcular a soma dos elementos do vetor).

```
Program CALCULA_MÉDIA;
const
 N = 50;
type
 VETOR = array[1..N] of integer;
var
 VET : VETOR;
 MEDIA : integer;

procedure LEIA_VETOR (var V:VETOR);
var
 I : integer;
begin
 for I:=1 to N do
 readln(V[I]);
 end;

function SOMA_VETOR (V:VETOR) : integer;
var
 I,S : integer;
begin
 S := 0;
 for I:=1 to N do
 S := S + V[I];
 SOMA_VETOR := S;
end;

begin
 {leitura do vetor}
 LEIA_VETOR(VET);
 {cálculo da média}
 MEDIA := SOMA_VETOR(VET) div N;
 {exibição do resultado}
 writeln(MEDIA);
end.
```

## EXERCÍCIOS PROPOSTOS

- P6.22. Escreva uma função que receba um vetor V de 30 elementos inteiros, e retorne o maior elemento do vetor V.
- P6.23. Escreva uma função que receba um vetor V de 30 elementos inteiros, e retorne a moda do vetor, isto é, o elemento que mais ocorre dentro do vetor V.
- P6.24. Escreva uma função que receba um vetor V de 30 elementos inteiros, e retorne a quantidade de números positivos do vetor V.
- P6.25. Escreva uma função que receba um valor X do tipo inteiro e um vetor V de 80 elementos inteiros, e retorne o número de ocorrências de X dentro do vetor V.
- P6.26. Escreva uma função que receba um valor X do tipo inteiro e um vetor V de 80 elementos inteiros, e retorne o valor lógico TRUE se X existir dentro de V ou o valor lógico FALSE caso contrário.
- P6.27. Escreva um procedimento chamado PAR\_IMPARG que receba um vetor de 100 elementos inteiros e retorne a quantidade de números pares e de números ímpares contidas no mesmo.  
Faça um programa que leia 100 valores inteiros (armazenando em um vetor) e, usando o procedimento PAR\_IMPARG, determine e exiba a quantidade de números pares e de números ímpares.
- P6.28. Escreva um procedimento chamado LEIA que leia um vetor de 50 elementos inteiros (passado como parâmetro).  
Escreva uma função chamada MAIOR que receba um vetor de 50 elementos inteiros e retorne o maior elemento do mesmo.  
Escreva uma função chamada SOMA que receba um vetor de 50 elementos inteiros e retorne a soma dos elementos do vetor.  
Faça um programa que:  
⇒ Usando o procedimento chamado LEIA, leia a idade de 50 pessoas;  
⇒ usando a função MAIOR criada, determine e exiba a maior idade;  
⇒ usando a função SOMA criada, calcule e exiba a idade média das 50 pessoas;
- P6.29. Modifique o programa da questão P5.08, criando os seguintes subprogramas:
- um procedimento que leia o gabarito da prova;
  - um procedimento que leia as respostas de um aluno.
  - Uma função que receba as respostas de um aluno e retorne a sua nota;
  - Uma função que receba a frequência de cada nota e retorne a nota que teve maior frequência absoluta, ou seja, a nota que apareceu maior número de vezes (supondo a inexistência de empates).

## 6.6. RECURSIVIDADE

A recursividade é uma característica que alguns problemas apresentam: a de serem definidos em termos deles mesmos. Todo problema que se comporta assim é dito ser recursivo.

A recursão é uma técnica apropriada se o problema a ser resolvido tem as seguintes características:

- 1) a resolução dos casos maiores do problema envolve a resolução de um ou mais casos menores;
- 2) os menores casos possíveis do problema podem ser resolvidos diretamente;
- 3) a solução iterativa do problema (usando enquanto, para ou repita) é complexa.

### Exemplo:

O problema do fatorial, que foi visto anteriormente, é recursivo por definição:

$$N! = N \times (N - 1) \times (N - 2) \times (N - 3) \dots \times 1 \quad (\text{Equação 1})$$

Existe um caso especial: 0! é igual a 1, por definição.

A partir da equação 1, podemos concluir que o fatorial de N está expresso em termos do fatorial de N-1.

$$N! = N \times \underbrace{(N - 1) \times (N - 2) \times (N - 3) \times \dots \times 1}_{(N - 1)!}$$

Resumindo:

$$N! = N \times (N - 1)! \quad (\text{Equação 2})$$

A equação 2 é válida para todos os números inteiros com exceção do 0 (zero), sendo, portanto, necessário um tratamento especial.

O programa recursivo em Pascal para o cálculo do fatorial de N ficaria assim:

```

Program FATORIAL;

var
 N,F : longint;

{função recursiva que retorna o fatorial de N}

function FAT(N:longint):longint;
begin
 if N = 0 then
 FAT := 1
 else
 FAT := N * FAT (N - 1)
 end;

{programa principal}

begin
 write('Digite um número: ');
 readln(N);
 F := FAT(N);
 writeln('O fatorial de ',N,' é ',F)
end.

```

Note que para cada chamada da função recursiva FAT deve ser criado um novo nível de ativação, guardado em uma pilha, onde para cada um destes níveis de ativação têm-se um parâmetro de chegada e um valor de chamada para uma nova ativação ou um valor de retorno.

Criaremos uma tabela para melhor expor a observação acima e para isso vamos calcular o fatorial de 3:

| Número da ativação | Valor de chegada | Rechamada             | Valor de retorno |
|--------------------|------------------|-----------------------|------------------|
| 1                  | 3                | $3 * \text{FAT}(3-1)$ |                  |
| 2                  | 2                | $2 * \text{FAT}(2-1)$ |                  |
| 3                  | 1                | $1 * \text{FAT}(1-1)$ |                  |
| 4                  | 0                |                       | 1                |
| 3                  |                  |                       | 1                |
| 2                  |                  |                       | 2                |
| 1                  |                  |                       | 6                |

Observe que até ser atendida a condição de retorno ( $N = 0$ ), têm-se apenas reativações de FAT. Quando o valor de chegada é igual a 0 (na ativação 4), começa o processo de retorno, onde cada nível imediatamente superior recebe o resultado do nível inferior. Quando um nível recebe o resultado do seu nível inferior ele o aplica para poder calcular a expressão  $\text{FAT} := N * \text{FAT}(N-1)$  até então indefinida, retornando para o nível de cima o seu valor calculado.

É importante que seja observada a condição de encerramento da recursão para evitar que este processo continue indefinidamente.

## EXERCÍCIO RESOLVIDO

R6.04. Resolva de forma recursiva o jogo das torres de Hanói.

O jogo consiste em trabalhar-se com discos de diferentes tamanhos, onde os mesmos podem ser movidos entre três hastes diferentes que chamaremos de A, B, e C. No princípio do jogo, os discos estarão todos na primeira haste que é a A de tal forma que os maiores devem ficar por baixo dos menores, formando uma pilha do maior para o menor. O objetivo do jogo é mover todos os discos de A para B, usando a haste C como auxiliar e seguindo as regras abaixo:

- 1) apenas um disco pode ser movido por vez;
- 2) um disco maior não pode ficar em cima de um menor;
- 3) todo e qualquer disco pode ser movido de uma haste para outra desde que sejam respeitadas as regras anteriores.

O número de discos a ser usado no jogo fica a critério do jogador. Em nosso caso, utilizaremos 3 (três) discos. A configuração inicial do nosso jogo seria assim:



Figura 1: 3 discos que devem ser movidos da haste A para a haste B, seguindo as regras citadas.

A solução deste problema advém da redução do problema original em problemas menores. Devem ser feitas reduções sucessivas do problema até que se chegue ao problema trivial de mover um disco de uma haste para outra. Explicando: em nosso exemplo temos 3 discos, então vamos reduzi-lo ao problema imediatamente menor que é o de mover 2 discos e assim sucessivamente até que se tenha o problema de mover apenas um disco.

Vamos acompanhar, graficamente, a solução para os nossos 3 discos:

1 - observe que os 3 discos estão, inicialmente, na haste A:

2 - Vejamos como seriam os movimentos para resolver o problema :

Seguindo esse raciocínio, pode-se estabelecer uma estratégia para resolver o problema de mover um número N qualquer de discos da haste A para a haste B:

Se  $N = 1$  então

mover o disco 1 da haste A para B

senão

1. Movemos N-1 discos da haste A para a haste C, usando a haste B para armazenamento temporário;
2. Em seguida, movemos o disco N da haste A para a haste B;
3. Por fim, movemos os N-1 discos da haste C para a haste B, usando a haste A para armazenamento temporário.

O procedimento recursivo para implementar esta estratégia ficaria, na linguagem Pascal, da seguinte forma:

```
procedure HANOI (N:integer; ORIGEM,DESTINO,TEMPORARIO:char);
begin
 if N = 1 then
 writeln('Mover disco 1 da haste ',ORIGEM,' p/ haste ',DESTINO)
 else
 begin
 HANOI(N-1,ORIGEM,TEMPORARIO,DESTINO);
 writeln('Mover disco ',N,' da haste ',ORIGEM,' p/ haste ',DESTINO);
 HANOI(N-1,TEMPORARIO,DESTINO,ORIGEM)
 end
 end;
end;
```

A saída do procedimento deve ser a seguinte:

- 1 - Mover disco 1 da haste A para haste B
- 2 - Mover disco 2 da haste A para haste C
- 3 - Mover disco 1 da haste B para haste C
- 4 - Mover disco 3 da haste A para haste B
- 5 - Mover disco 1 da haste C para haste A
- 6 - Mover disco 2 da haste C para haste B
- 7 - Mover disco 1 da haste A para haste B

CONCLUINDO:

A recursividade é uma técnica elegante e quem a domina, geralmente, demonstra experiência, porém possui um preço: a movimentação de dados na PILHA (controle interno da linguagem para possibilitar ativações recursivas). Essa movimentação de dados de controle, impõe à solução um tempo adicional que pode torná-la ineficiente.

Devido a esse tempo extra utilizado pelas soluções recursivas, deve-se dar preferência às soluções iterativas, deixando a utilização da recursividade para os casos apropriados (atentar para características básicas de um problema recursivo).

**EXERCÍCIOS PROPOSTOS**

P6.30. Dê o conceito de recursividade.

P6.31. Quais as características dos problemas que são apropriados para serem resolvidos por recursão?

P6.32. Escreva uma função recursiva que calcule  $X^N$ , onde  $X$  é um número real e  $N$  é um inteiro não negativo. Essa função pode ser definida da seguinte forma:

$$X^N = \begin{cases} 1.0 & , \text{ se } N = 0 \\ X * X^{N-1} & , \text{ se } N > 0 \end{cases}$$

P6.33. As séries de Fibonacci foram originalmente concebidas no século XIII por Leonardo de Pisa, apelidado de Fibonacci, como um modelo para estudar a criação de coelhos. A aplicação desta série cresceu, não somente na matemática e na ciência da computação, como também em vários fenômenos biológicos.

A série em si é muito simples: 0, 1, 1, 2, 3, 5, 8, 13, ... Os primeiros dois termos são 0 e 1; cada termo subsequente é calculado como a soma dos dois termos anteriores. Em geral,  $t_i = t_{i-1} + t_{i-2}$ .

O problema de encontrar o enésimo número de Fibonacci tem uma formulação recursiva clara, que é:

$$\text{Fibonacci}(N) = \begin{cases} \text{Fibonacci}(N-1) + \text{Fibonacci}(N-2) & , \text{ se } N > 2 \\ 1 & , \text{ se } N = 2 \\ 0 & , \text{ se } N = 1 \end{cases}$$

Escreva uma função recursiva em Pascal que, dado um valor  $N$  inteiro positivo, retorne o enésimo número de Fibonacci.

P6.34. Um algoritmo muito conhecido para determinar o maior divisor comum de dois inteiros é o algoritmo de Euclides. A função maior divisor comum (MDC) é definida como segue:

$$\text{MDC}(X,Y) = \begin{cases} \text{MDC}(Y,X) & , \text{ se } Y > X \\ X & , \text{ se } Y = 0 \\ \text{MDC}(Y, X \bmod Y) & , \text{ se } Y > 0 \end{cases}$$

Construir uma função recursiva em Pascal para resolver este problema.

P6.35. Escreva uma versão não-recursiva da função abaixo:

```
function F (N:integer) : integer;
begin
 if N > 1 then
 F := N + F(N - 1)
 else
 F := 1
 end;
```

P6.36. Escreva uma função recursiva que retorne a soma dos elementos de um vetor de elementos inteiros (máx.de100). O tamanho do vetor também é passado como parâmetro.

P6.37. Escreva uma função recursiva que retorne a quantidade de números positivos de um vetor de elementos inteiros (máx.de100). O tamanho do vetor também é passado como parâmetro.

P6.38. Escreva uma solução iterativa para o problema das torres de Hanoi.

## 6.7. CRIAÇÃO DE UNITS

Uma **Unit** é uma coleção de constantes, tipos de dados, variáveis, procedimentos e funções. Cada Unit é como um programa Pascal separado. Ela é uma biblioteca de declarações que permite dividir seu programa e compilá-lo em partes separadas. Ela pode ter um corpo principal o qual é chamado antes do seu programa ser iniciado para preparar as "inicializações" necessárias.

Todas as declarações em uma Unit estão normalmente relacionadas. Por exemplo, a unit CRT contém todas as declarações de rotinas relativas à tela do computador. O Turbo Pascal possui 8 Units predefinidas: System, Overlay, Graph, Dos, Crt, Printer...

### 6.7.1. ESTRUTURA DE UMA UNIT

Toda unit deve iniciar pela palavra UNIT seguida do identificador da unit (que deve ter obrigatoriamente o mesmo do arquivo que será gravado em disco).

Em seguida aparece a seção INTERFACE, onde deve ser colocado tudo que será tornado público, isto é, tudo que os programas ou as outras units que usarem esta terão disponíveis.

A outra seção é a IMPLEMENTATION, onde deve ser colocado a implementação das rotinas que foram declaradas na INTERFACE e também tudo que for privado, isto é, que será local a esta unit, não estando disponíveis aos outros programas ou units que usarem esta.

Veja o formato geral:

```
UNIT <identificador>; {identificador deve ser o mesmo nome do arquivo}

INTERFACE
 uses <lista de units> {opcional}
 <declarações públicas> {só cabeçalho}

IMPLEMENTATION
 uses <lista de units> {opcional}
 <declarações privadas>

 <implementação de proc. e funções>
 {corpo das funções e proc.}

End.
```

### EXEMPLO:

```
Unit ROTINAS; {salva em disco com o nome ROTINAS.PAS}

Interface

 Function COMP_CIRC (R:real) : real;
 Function AREA_CIRC (R:real) : real;

Implementation

 Const PI=3.1416;

 Function COMP_CIRC (R:real) : real;
 {retorna o comprimento de uma circunferência}
 Begin
 COMP_CIRC := 2 * PI * R;
 End;

 Function AREA_CIRC (R:real) : real;
 {retorna a área de uma circunferência}
 Begin
 AREA_CIRC := PI * SQR(R);
 End;

End. (final da Unit)
```

## 6.7.2. UTILIZAÇÃO DE UNITS

O programa abaixo, localizado em outro arquivo, utiliza esta unit.

```
Program TESTA_UNIT;

Uses
 CRT, ROTINAS;

Var
 RAIO, COMP, AREA : integer;

Begin
 clrscr;
 write('Digite o raio da circunferência: ');
 readln(RAIO);
 COMP := COMP_CIRC(RAIO);
 AREA := AREA_CIRC(RAIO);
 writeln('Comprimento = ', COMP:0:2);
 writeln('Área = ', AREA:0:2);
 readkey;
End.
```

Assim que este programa for executado, o Turbo Pascal irá gerar o arquivo ROTINAS.TPU, que nada mais é do que o arquivo objeto que contém a unit ROTINAS compilada.

**Observação:** problemas poderão ocorrer em virtude da localização do arquivo da unit. O Turbo Pascal permite definirmos o diretório (pasta) onde estarão armazenadas todas as units necessárias.

## EXERCÍCIO PROPOSTO

P6.39. Criar uma unit chamada BIBLIOT contendo os seguintes subprogramas:

- uma procedimento chamado LEIA que receba um string S e uma variável inteira V, exiba o string S e, em seguida, leia a variável V;
- um procedimento chamado PAUSA que exiba a mensagem "Pressione qualquer tecla para continuar", seguida de um comando de espera por uma tecla digitada;
- uma função chamada MENOR que receba 3 valores inteiros e retorne o menor deles;
- uma função chamada MEDIA que receba 2 valores inteiros e retorne a média (inteira) deles.

P6.40. Faça um programa que leia as 3 notas (do tipo inteiro) dos 50 alunos de uma turma, elimine a nota mais baixa e calcule e exiba a média das duas notas restantes. Obs: faça uso dos subprogramas da unit BIBLIOT que você achar possível.

P6.41. Acrescente a unit BIBLIOT um procedimento chamado FINAL que exiba a mensagem "Pressione qualquer tecla para retornar ao Turbo Pascal", seguida de um comando de espera por uma tecla digitada. Substitua no seu programa o procedimento PAUSA pelo procedimento FINAL.

## Capítulo 7

# MANIPULAÇÃO DE STRINGS

### 7.1. O TIPO DE DADO STRING

Os strings, como trechos de texto, são os tipos de dados mais familiares aos seres humanos. O Pascal padrão não fornecia tipos de dados de String; tínhamos que utilizar apenas o tipo Char e, para trabalharmos com textos, devíamos utilizar um array de Char.

O Turbo Pascal, felizmente, oferece para os usuários o tipo de dado String. Embora este tipo seja considerado um array de Char, podemos ignorar este fato e utilizá-lo normalmente. Quando necessário, podemos utilizá-lo como um array (por exemplo, usando os colchetes com um índice para individualizar cada caracter do string). Observe o exemplo abaixo:

```
S := 'ASPER';
write(s[4]); {será exibida a letra E, correspondente a 4ª letra da string S}
```

O tamanho de um string pode variar entre 0 e 255 caracteres. Se na declaração de uma variável string não especificarmos o tamanho máximo do string, ele assumirá 255. Por exemplo:

```
var
 S1 : string;
 S2 : string[10];
```

No exemplo acima, a variável S1 pode conter até 255 caracteres, enquanto a variável S2 pode conter no máximo 10 caracteres. Esse tamanho máximo nós denominamos de comprimento físico do string, que é o que determina o espaço reservado para a variável.

Se na variável S2 for armazenado um string de 4 caracteres, por exemplo, o comprimento físico continua sendo de 10 caracteres, enquanto que o espaço ocupado, no caso 4 caracteres, é o que denominamos comprimento lógico do string. O comprimento lógico de um string pode variar conforme o valor recebido pela variável durante o programa.

Resumindo, temos então que um string pode ter o seu comprimento físico variando de 1 a 255 caracteres, e o seu comprimento lógico variando de 0 até o valor do comprimento físico.

### 7.2. USANDO STRINGS COMO PARÂMETROS EM SUBROTINAS

O Turbo Pascal aceita a declaração do tipo **string** (sem definição de tamanho) como parâmetro de um procedimento ou função, porém não aceita a declaração do tipo **string[n]**, nos obrigando da utilização da declaração **type** para a criação de um novo tipo de dado do tipo string[n], assim como acontece com os arrays,. Veja o exemplo:

```
procedure PROC (s1:string; s2:string[10]);
```

Essa declaração causaria um erro de compilação por causa do tipo string[10]. Teríamos então que fazer o seguinte:

```
type STRING10 = string[10];

procedure PROC (s1:string; s2:STRING10);
```

### 7.3. FUNÇÕES E PROCEDIMENTO PREDEFINIDOS

O Turbo Pascal dispõe de algumas funções e procedimentos que visam em essência, à otimização do trabalho do programador na parte que se refere à utilização de strings:

- |          |          |       |
|----------|----------|-------|
| – LENGTH | – COPY   | – STR |
| – UPCASE | – DELETE | – CHR |
| – CONCAT | – INSERT | – ORD |
| – POS    | – VAL    |       |

**LENGTH** – Função que retorna o número de caracteres de uma string. Sua sintaxe é:

```
LENGTH (str : string) : byte;
```

Exemplo:

```
tam := length('TURBO PASCAL');
writeln (tam); {será exibido o valor 12}
```

**UPCASE** – Função que retorna o caractere contido no parâmetro em maiúsculo. Sua sintaxe é:

```
UPCASE (ch : char) : char;
```

Exemplo:

```
letra := 'a';
maiusc := upcase (letra);
writeln (maiusc); {será exibida a letra 'A' (maiúscula) }
```

**CONCAT** – Função que retorna a união de duas ou mais strings passadas como parâmetros. Sua sintaxe é:

```
CONCAT (str1 , str2 , ... , strn : string) : string;
```

Exemplo:

```
pal1 := 'TURBO';
pal2 := 'PASCAL';
uniao := concat (pal1,' ',pal2);
writeln (uniao); {será exibido o string 'TURBO PASCAL'}
```

A função CONCAT tem efeito semelhante ao operador + (operador de concatenação).

Exemplo:

```
pal1 := 'TURBO';
pal2 := 'PASCAL';
uniao := pal1 + ' ' + pal2;
writeln (uniao); {será exibido o string 'TURBO PASCAL'}
```

**POS** – Função que retorna a posição que uma substring ocupa dentro de uma string passadas como parâmetro. Sua sintaxe é:

```
POS (substr , str : string) : byte;
```

Exemplo:

```
frase := 'VAMOS ESTUDAR MAIS';
pesq := 'ESTU';
posicao := pos (pesq,frase);
writeln (posicao); {será exibido o valor 7}
```

**COPY** – Função que retorna uma substring de uma string passadas como parâmetro, de acordo com sua posição e quantidade de caracteres especificados. Sua sintaxe é:

```
COPY (str:string; pos:byte; quant:byte) : string;
```

Exemplo:

```
frase := 'VAMOS ESTUDAR MAIS';
pedaco := copy(frase,7,4);
writeln (pedaco); {será exibido o string 'ESTU'}
```

**DELETE** – Procedimento que exclui um pedaço de uma string passada como parâmetro, de acordo com uma posição e quantidade de caracteres especificados. Sua sintaxe é:

```
DELETE (var str:string; pos:byte; quant:byte);
```

Exemplo:

```
frase := 'TURBO PASCAL 7.0';
delete (frase,7,7);
writeln (frase); {será exibido o string 'TURBO 7.0'}
```

**INSERT** – Procedimento que permite inserir uma substring dentro de uma string, em uma posição especificada. Sua sintaxe é:

```
INSERT (substr:string; var str:string; pos:byte);
```

Exemplo:

```
frase := 'CURSO DE INFORMATICA';
insert ('MICRO',frase,10);
writeln (frase); {será exibido o string 'CURSO DE MICROINFORMATICA'}
```

**VAL** – Procedimento que converte uma string passada como parâmetro para valor numérico. Caso o conteúdo da string não seja possível de ser convertido, o fato será informado em uma variável de retorno de erro. Se o retorno de erro for diferente de 0 (zero), implica que houve um erro de conversão, e este valor de retorno é a posição onde ocorreu o primeiro erro. Sua sintaxe é:

```
VAL (str:string; var num:integer|real; var erro:integer);
```

Exemplo 1:

```
codigo := '017348';
val (codigo,numero,erro);
writeln (numero); {será exibido o valor 17348}
writeln (erro); {será exibido o valor 0}
```

Exemplo 2:

```
codigo := '12X345';
val (codigo,numero,erro);
writeln (erro) {será exibido o valor 3}
```

**STR** – Procedimento que converte uma variável numérica em um string, determinando o tamanho do string e a quantidade de casas decimais. Sua sintaxe é:

```
STR (num [:tam [:dec]]; var str:string);
```

Exemplo:

```
numero := 12.3;
str (numero:6:2,conv);
writeln (conv); {será exibido o string ' 12.30'}
```

**CHR** – Função que retorna o caracter correspondente ao valor ASCII especificado. Sua sintaxe é:

```
CHR (codigo:byte) : char;
```

Exemplo:

```
codigo := 65;
caracter := CHR(codigo);
writeln (caracter); {será exibido o caracter 'A'}
```

**ORD** – Função que retorna o valor ASCII correspondente ao caracter especificado. Sua sintaxe é:

```
ORD (caracter:char) : byte;
```

Exemplo:

```
caracter := 'A';
codigo := ORD(caracter);
writeln (codigo); {será exibido 65}
```

**EXERCÍCIOS RESOLVIDOS**

R7.01. Escreva um procedimento que receba um string S e converta o mesmo para letras maiúsculas.

```
procedure MAIUSC (var S:string);
var
 I,TAM : byte;
begin
 TAM := length(S);
 for I:= 1 to TAM do
 S[I] := upcase(S[I]);
 end;
```

R7.02. Escreva uma função que retorne o número de ocorrências de um substring SUB dentro de um string S, passados como parâmetros.

```
function OCORRENCIAS (SUB,S:string) : byte;
var
 I,CONT,TSUB,TS : byte;
begin
 CONT := 0;
 TSUB := length(SUB)
 TS := length(S);
 for I:= 1 to (TS-TSUB+1) do
 if copy(S,I,TSUB) = SUB then
 CONT := CONT+1;
 OCORRENCIAS := CONT;
end;
```

R7.03. Escreva um procedimento que receba um string S como parâmetro e retire todos os brancos contidos no mesmo.

```
procedure TIRABRANCOS (var S:string);
var
 I,TAM : byte;
begin
 TAM := length(S);
 I := 1;
 while I<=TAM do
 if S[I]=' ' then
 begin
 delete(S,I,1);
 TAM := TAM-1;
 end
 else
 I := I+1;
 end;
```

R7.04. Escreva uma função que receba um número real e retorne um string correspondente ao número recebido, com o mesmo convertido para string com tamanho mínimo e 2 casas decimais, e com uma vírgula no lugar do ponto decimal.

```
function CONVERSAO (X:real) : string;
var
 P : byte;
 S : string;
begin
 str(X:0:2,S);
 P := pos('.',S);
 S[P] := ',';
 CONVERSAO := S;
end;
```

**EXERCÍCIOS PROPOSTOS**

- P7.01. O que é um string?
- P7.02. Qual o tamanho que um string pode assumir?
- P7.03. Qual a diferença entre comprimento físico e comprimento lógico? Qual o tamanho mínimo e máximo de cada um deles?
- P7.04. Qual a restrição do Turbo Pascal com relação ao uso de strings como parâmetros em subrotinas?
- P7.05. Qual a unit do Turbo Pascal onde estão localizados os comandos de controle do vídeo e do teclado?
- P7.06. Escreva a finalidade e dê um exemplo de cada um dos comandos abaixo:
- |           |           |        |
|-----------|-----------|--------|
| a) LENGTH | e) COPY   | i) STR |
| b) UPCASE | f) DELETE | j) CHR |
| c) CONCAT | g) INSERT | k) ORD |
| d) POS    | h) VAL    |        |
- P7.07. Escreva uma função que receba uma string S e retorne o número de brancos existentes na mesma.
- P7.08. Escreva uma função que retorna uma string preenchida com vários brancos. A quantidade de brancos deverá ser fornecida como parâmetro.
- P7.09. Escreva uma função que receba um string S e um valor inteiro N e retorne os N primeiros caracteres do string S.
- P7.10. Escreva uma função que receba um string S e um valor inteiro N e retorne os N últimos caracteres do string S.
- P7.11. Escreva um procedimento que receba uma string S e elimine os brancos à esquerda da mesma.
- P7.12. Escreva uma função para substituir uma seqüência de caracteres por outra dentro de uma frase. O retorno da função será a frase modificada. Obs: deve-se considerar apenas a primeira ocorrência da seqüência a ser substituída, caso exista mais de uma ocorrência.
- P7.13. Supondo que no Turbo Pascal não existisse a função UPCASE, escreva uma função que simule a mesma.
- P7.14. Idem para a função POS.
- P7.15. Idem para a função COPY.
- P7.16. Idem para o procedimento DELETE.
- P7.17. Idem para o procedimento INSERT.
- P7.18. Escreva um programa que lê uma frase pelo teclado e diz quantos caracteres foram digitados na mesma.
- P7.19. Escreva um programa que receba uma frase pelo teclado e determina a quantidade de cada vogal contida na mesma.
- P7.20. Escreva um programa que receba uma string do teclado e a exiba invertida.
- |          |         |              |
|----------|---------|--------------|
| Exemplo: | Entrada | MARIA BONITA |
|          | Saída   | ATINOB AIRAM |
- P7.21. Escreva um programa que leia o nome de uma pessoa e exiba-o conforme o exemplo abaixo. Obs: Suponha que o nome lido não possua nenhuma preposição, artigo, etc.
- |          |         |                         |
|----------|---------|-------------------------|
| Exemplo: | Entrada | FLAVIO RIBEIRO COUTINHO |
|          | Saída   | COUTINHO, F. R.         |

P7.22. Escreva um programa que leia uma frase de no máximo 20 caracteres e o exiba de acordo com o exemplo seguinte.

|          |         |       |       |                                           |
|----------|---------|-------|-------|-------------------------------------------|
| Exemplo: | Entrada | ASPER | Saída | ASPER<br>SPERA<br>PERAS<br>ERASP<br>RASPE |
|----------|---------|-------|-------|-------------------------------------------|

P7.23. Escreva um programa que, a partir da digitação do infinitivo de um verbo regular, faça a conjugação do mesmo no presente do indicativo para as pessoas do singular e plural .

|          |         |        |       |                                                                                  |
|----------|---------|--------|-------|----------------------------------------------------------------------------------|
| Exemplo: | Entrada | CANTAR | Saída | Eu canto<br>Tu cantas<br>Ele canta<br>Nós cantamos<br>Vós cantais<br>Eles cantam |
|----------|---------|--------|-------|----------------------------------------------------------------------------------|

## 7.4. CONTROLE DO VÍDEO E DO TECLADO

Os procedimentos e funções mais usados que controlam o vídeo e o teclado no Turbo Pascal são:

|             |         |                 |
|-------------|---------|-----------------|
| -KEYPRESSED | -CLRSCR | -CLREOL         |
| -READKEY    | -GOTOXY | -TEXTCOLOR      |
| -DELAY      | -WHEREX | -TEXTBACKGROUND |
| -WINDOW     | -WHEREY |                 |

Todos esses comandos estão localizados na Unit CRT, sendo necessário para usá-los a declaração:

```
uses CRT
```

Detalharemos, a seguir, cada um destes comandos.

**KEYPRESSED** – Função que retorna o valor lógico TRUE caso tenha sido pressionada alguma tecla. Sua sintaxe é:

```
KEYPRESSED : boolean
```

**READKEY** – Função que retorna o valor (do tipo caracter) de uma tecla pressionada. Bastante utilizada quando queremos receber um caracter pelo teclado sem que o usuário precise teclar ENTER. Sua sintaxe é:

```
READKEY : char
```

**Observação:** As teclas de função (F1, F2, ...) nos retornam dois códigos, o primeiro sendo o caracter zero da tabela ASCII, e o segundo o da própria tecla de função.

**DELAY** – Procedimento que provoca uma pausa num determinado intervalo de tempo antes de ser executado o próximo comando. O intervalo de tempo especificado é sempre em milissegundo. Sua sintaxe é:

```
DELAY (tempo : word)
```

**WINDOW** – Procedimento que nos permite definir o tamanho útil da tela. Quando definimos uma window, as coordenadas de referência de linha e coluna ficam relativas à nova window e sempre o canto superior esquerdo da tela é a posição (1,1), estando os procedimentos de vídeo também vinculados a esta nova janela. Sua sintaxe é:

```
WINDOW (x1, y1, x2, y2 : byte)
```

onde:

x1, y1 – são as coordenadas do canto superior esquerdo da janela

x2, y2 – são as coordenadas do canto inferior direito da janela

**CLRSCR** – Procedimento que limpa a tela e automaticamente coloca o cursor no canto superior esquerdo da mesma. Sua sintaxe é:

```
CLRSCR
```

**GOTOXY** – Este procedimento posiciona o cursor em um ponto determinado da tela, referenciado pelos eixos X e Y, ou seja, coluna e linha. Sua sintaxe é:

```
GOTOXY (x, y : byte)
```

**WHEREX** – Função que retorna a coluna em que se encontra o cursor (em relação à window atual). Sua sintaxe é:

```
WHEREX : byte
```

**WHEREY** – Função que retorna a linha em que se encontra o cursor (em relação à window atual). Sua sintaxe é:

```
WHEREY : byte
```

**CLREOL** – Procedimento que apaga todos os caracteres de uma linha do vídeo, que se encontram à direita do cursor. Sua sintaxe é:

```
CLREOL
```

**TEXTCOLOR** – Procedimento que determina a cor do texto que aparecerá no vídeo. Sua sintaxe é:

```
TEXTCOLOR (cor : byte)
```

As cores são representadas pelos valores inteiros de 0 a 15, que corresponde as seguintes cores:

|   |       |   |             |    |              |    |                |
|---|-------|---|-------------|----|--------------|----|----------------|
| 0 | Preto | 4 | Vermelho    | 8  | Cinza escuro | 12 | Vermelho claro |
| 1 | Azul  | 5 | Magenta     | 9  | Azul claro   | 13 | Magenta claro  |
| 2 | Verde | 6 | Marrom      | 10 | Verde claro  | 14 | Amarelo        |
| 3 | Ciano | 7 | Cinza claro | 11 | Ciano claro  | 15 | Branco         |

Além destas 16 cores disponíveis, podemos somar a qualquer uma delas 128 para que o texto fique piscante.

**TEXTBACKGROUND** – Procedimento que permite selecionar a cor de fundo da tela. Sua sintaxe é:

```
TEXTBACKGROUND (cor : byte)
```

As cores são representadas pelos valores inteiros de 0 a 7, que corresponde as seguintes cores:

|   |       |   |             |
|---|-------|---|-------------|
| 0 | Preto | 4 | Vermelho    |
| 1 | Azul  | 5 | Magenta     |
| 2 | Verde | 6 | Marrom      |
| 3 | Ciano | 7 | Cinza claro |

**EXERCÍCIOS RESOLVIDOS**

- R7.05. Escreva um procedimento que receba um string S e dois valores inteiros X e Y, e exiba o string S na coluna X e linha Y da tela.

```
procedure EXIBA (S:string; X,Y:byte);
begin
 gotoxy(X,Y);
 write(S);
end;
```

- R7.06. Escreva um procedimento que receba um string S e um valor inteiro LIN e exiba o string S centralizado na linha L da tela. Obs: suponha que a "window" ativa está ocupando toda a tela.

```
procedure CENTRA (S:string; LIN:byte);
var
 TAM,COL : byte;
begin
 TAM := length(S);
 COL := ((80-TAM) div 2) + 1;
 gotoxy(COL,LIN);
 write(S);
end;
```

- R7.07. Escreva um programa que leia uma frase (máximo de 30 caracteres) e faça a mesma "passar" pela tela do computador, isto é, faça a frase movimentar-se horizontalmente coluna a coluna, iniciando na coluna 1 e linha 1, e quando chegar à última coluna de cada linha, passar para a linha seguinte, até a última linha da tela. Crie também uma alternativa do programa parar ao se pressionar qualquer tecla.

```
Program PASSEIO;
uses
 CRT;
const
 TEMPO = 100;
var
 FRASE : string[30];
 TAM,COL,LIN : byte;
begin
 clrscr;
 write('Frase: ');
 readln(FRASE);
 TAM := length(FRASE);
 COL := 1;
 LIN := 1;
 repeat
 clrscr;
 gotoxy(COL,LIN);
 write(FRASE);
 delay(TEMPO);
 COL := COL+1;
 if COL >= (80-TAM+1) then
 begin
 LIN := LIN+1;
 COL := 1;
 end;
 until (LIN > 25) or keypressed;
end.
```

## EXERCÍCIOS PROPOSTOS

P7.24. Escreva a finalidade e dê um exemplo de cada um dos comandos abaixo:

- a) KEYPRESSED
- b) READKEY
- c) DELAY
- d) WINDOW
- e) CLRSCR
- f) GOTOXY
- g) WHEREX
- h) WHEREY
- i) CLREOL
- j) TEXTCOLOR
- k) TEXTBACKGROUND

P7.25. Escreva um procedimento que receba um string S e uma valor inteiro C, e exiba o string S com a cor da fonte C.

P7.26. Escreva um procedimento que limpe uma área retangular da tela, sendo passados como parâmetros as coordenadas (coluna esquerda, linha superior, coluna direita, linha inferior).

P7.27. Escreva um procedimento que exiba uma janela na tela, sendo passados como parâmetros as coordenadas da janela (coluna esquerda, linha superior, coluna direita, linha inferior), e a cor de preenchimento da janela.

P7.28. Escreva um procedimento que exiba uma janela centralizada na tela, sendo passados com parâmetro o número de colunas e o número de linhas da janela, e a cor de preenchimento da mesma.

P7.29. Escreva um procedimento para exibir uma janela com sombra, isto é, uma janela de uma cor sobreposta a outra janela de cor diferente, com um pequeno deslocamento entre as duas. São passados como parâmetro as coordenadas da janela, a cor de preenchimento da janela e a cor da sombra.

P7.30. Escreva um programa para transformar a tela do computador em um verdadeiro arco-íris, ou seja, colocar em cada linha uma cor de fundo diferente.

P7.30. Escreva um programa que exiba uma tabela de conversão de graus Celsius para Fahrenheit, no intervalo de 1 a 100, variando de 1 em 1, dispostos em 5 colunas na tela.

## Capítulo 8

# ARQUIVOS E REGISTROS

### 8.1. REGISTROS

São conjuntos de dados logicamente relacionados, mas de tipos diferentes (inteiro, real, string, etc.).

Exemplo: Numa dada aplicação, podem-se ter os seguintes dados sobre funcionários de uma empresa:

- ⇒ Nome
- ⇒ Endereço
- ⇒ Idade
- ⇒ Salário

Cada conjunto de informações do funcionário pode ser referenciado por um mesmo nome, por exemplo, FICHA. Tais estruturas são conhecidas como registros, e aos elementos do registros dá-se o nome de campos.

O conceito de registro visa facilitar o agrupamento de variáveis que não são do mesmo tipo, mas que guardam estreita relação lógica.

#### 8.1.1. DECLARAÇÃO

Criam-se estruturas de dados agrupados na forma de registro através da seguinte declaração:

```
lista-de-identificadores : RECORD
 campos
 END;
```

onde:

- lista-de-identificadores** são os nomes que estão sendo associados aos registros que se deseja declarar;
- campos** são declarações de variáveis, separadas por ponto-e-vírgula.

Exemplo: Declarar o registro FICHA especificado anteriormente.

```
Var FICHA : record
 NOME : string[30];
 ENDERECO : string[40];
 IDADE : byte;
 SALARIO : real;
end;
```

#### 8.1.2. REFERÊNCIA

A referência ao conteúdo de um dado campo do registro será indicada pela notação:

```
identificador-do-registro . identificador-do-campo
```

Exemplo: Sabendo-se que o registro FICHA, em um dado instante, contivesse os valores a seguir:

**NOME** : Antônio Ajuizado  
**ENDEREÇO**: Rua das Virtudes, s/n  
**IDADE**: 20 anos  
**SALÁRIO**: R\$ 150,00

**FICHA.IDADE** estaria fazendo referência ao conteúdo do campo IDADE do registro FICHA, isto é, 20.

### 8.1.3. CONJUNTO DE REGISTROS

Podem-se ter conjunto de registros referenciáveis por um mesmo nome e individualizados por índices, através da utilização de um array de registros.

Exemplo:

```
Var
 TAB : array[1..50] of record
 MATR : integer;
 NOME : string[30];
 MEDIA : real;
 end;
```

### EXERCÍCIO RESOLVIDO

R8.01. Considerando o registro de uma mercadoria de uma loja contendo as seguintes informações: código, nome, preço e estoque, fazer um programa que, dado o registro de 50 mercadorias, leia um código exiba o nome, preço e estoque da respectiva mercadoria.

```
Program MERCADORIAS;

Uses Crt;

Const N = 50;

Var TAB : array[1..N] of record
 COD : string[6];
 NOME : string[15];
 PRECO: real;
 EST : integer;
end;

 I : integer;
 K : string[6];
 RESP : char;

Begin
 clrscr;
 {Leitura dos dados}
 for I:=1 to N do
 begin
 write('Código: '); readln(TAB[I].COD);
 write('Nome: '); readln(TAB[I].NOME);
 write('Preço: '); readln(TAB[I].PRECO);
 write('Estoque: '); readln(TAB[I].EST);
 end;
 repeat
 {leitura da chave de pesquisa}
 write('entre com o código desejado: ');
 Readln(K);
 {testa em cada registro se o código é igual a chave pesquisada}
 for I:=1 to N do
 if K = TAB[I].COD then
 writeln(TAB[I].NOME, TAB[I].PRECO, TAB[I].EST);
 {verifica se o usuário deseja pesquisar outro código}
 write('Repetir(S/N)?');
 RESP := readkey;
 until upcase(RESP) = 'N';
 End.
```

### 8.1.4. O COMANDO WITH

Este comando permite que os campos de um registro sejam denotados unicamente por seus identificadores, sem a necessidade de serem precedidos pelo identificador do registro. Forma geral:

```
WITH identificador-do-registro DO
 comandos
```

## EXERCÍCIO RESOLVIDO

R8.02. Reescrever o programa MERCADORIAS (R8.01), utilizando o comando WITH.

```
Program MERCADORIAS_WITH;

Uses Crt;

Const N = 50;

Var TAB : array[1..N] of record
 COD : string[6];
 NOME : string[15];
 PRECO: real;
 EST : integer;
end;

I : integer;
K : string[6];
RESP : char;

Begin
 clrscr;
 {Leitura dos dados}
 for I:=1 to N do
 with TAB[I] do
 begin
 write('Código: '); readln(COD);
 write('Nome: '); readln(NOME);
 write('Preço: '); readln(PRECO);
 write('Estoque: '); readln(EST);
 end;
 end;
 repeat
 {leitura da chave de pesquisa}
 write('entre com o código desejado: '); Readln(K);
 {testa em cada registro se o código é igual a chave pesquisada}
 for I:=1 to N do
 with TAB[I] do
 if K = COD then
 writeln(NOME, PRECO, EST);
 {verifica se o usuário deseja pesquisar outro código}
 write('Repetir(S/N)?');
 RESP := readkey;
 end;
 until upcase(RESP) = 'N';
 End.
```

## EXERCÍCIOS PROPOSTOS

P8.01. Escreva, na linguagem Pascal, as declarações que definem os seguintes registros:

- a) NOME, ENDEREÇO, CPF, SEXO;
- b) MATRÍCULA, NOTA1, NOTA2, NOTA3, MEDIA.

P8.02. Uma indústria faz a folha mensal de pagamentos de seus 80 empregados baseada no seguinte:

- ⇒ Existe uma tabela com os dados de cada funcionário (matrícula, nome e salário bruto);
- ⇒ Escreva um programa que leia e processe a tabela e emita, para cada funcionário, seu contracheque, cujo formato é dado a seguir:

|                                                                            |
|----------------------------------------------------------------------------|
| Matrícula:<br>Nome:<br>Salário Bruto:<br>Dedução INSS:<br>Salário Líquido: |
|----------------------------------------------------------------------------|

- ⇒ O desconto do INSS é de 12% do salário bruto.
- ⇒ O salário líquido é a diferença entre o salário bruto e a dedução do INSS.

P8.03. Em certo município, vários proprietários de imóveis estão em atraso com o pagamento do IPTU. Escrever um programa que calcule e escreva o valor da multa a ser paga por estes proprietários, considerando que:

- ⇒ os dados de cada imóvel (identificação, valor do imposto e número de meses em atraso) estão à disposição para leitura;
- ⇒ as multas devem ser calculadas no valor de 1% por mês de atraso.
- ⇒ o último registro lido, que não deve ser considerado, contém identificação do imóvel igual a XXX;
- ⇒ o programa deve exibir: a identificação do imóvel, valor do imposto, número de meses em atraso e multa a ser paga.

P8.04. Escreva um programa que armazene um cadastro de 50 pessoas com os seguintes dados: nome, telefone e data de nascimento (dia, mês, ano) e realize consultas da seguinte forma:

- ⇒ Leia o número de um determinado mês (1 a 12). Obs: a leitura do mês 0 encerra as consultas.
- ⇒ Exiba o nome, o telefone e o dia do aniversário das pessoas daquele respectivo mês.

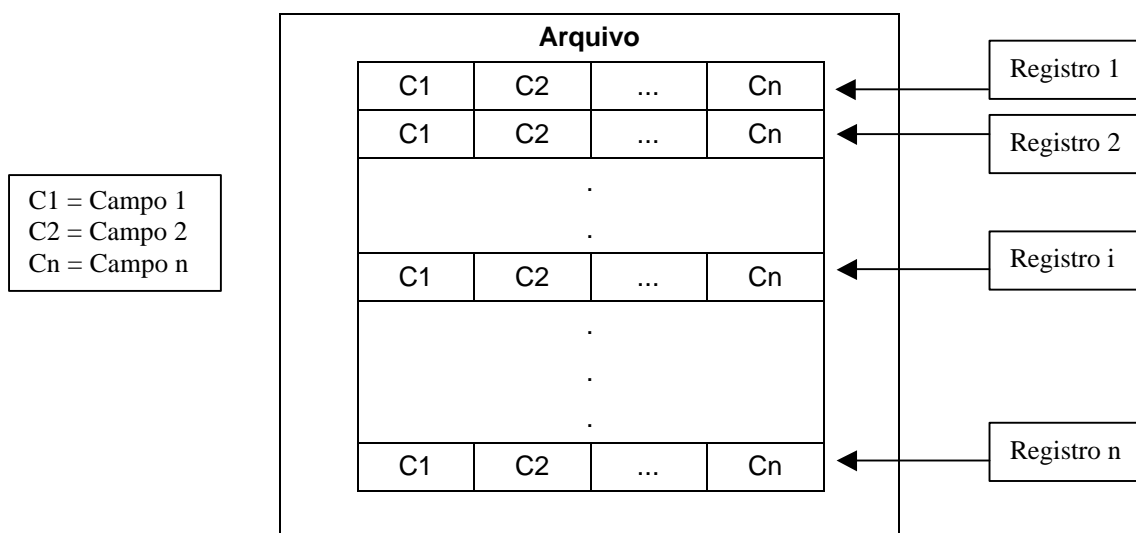
## 8.2. ARQUIVOS

Entende-se por arquivo uma coleção de dados, onde os mesmos possuem alguma relação entre si. Por exemplo, o conjunto de dados sobre os alunos da ASPER. Dentre estes dados sobre os alunos, temos:

- MATRÍCULA (string);
- NOME (string);
- ENDEREÇO (string);
- MEDIA ANUAL (real).

Cada conjunto de dados sobre um determinado aluno recebe o nome de registro. Note-se que um registro é composto por tipos de dados diferentes. Cada dado que compõe o registro (MATRÍCULA, NOME etc) é dito ser um campo.

A figura a seguir, ilustra a relação existente entre campo, registro e arquivo:



Um arquivo necessita de um meio de armazenamento que sirva de depósito (suporte) para seus dados. Os suportes mais comuns para os arquivos são os discos flexíveis, os discos rígidos (winchesters) e as fitas magnéticas.

Esses arquivos podem ser de 3 tipos: seqüencial, indexado e relativo. Essa diferenciação em tipos, está associada a algumas restrições quanto ao acesso aos dados guardados nesses arquivos. Em nosso caso, trataremos apenas com arquivos de organização seqüencial. O arquivo seqüencial é aquele cujo acesso aos seus dados só pode ser feito de forma seqüencial, isto é, um registro de ordem N está localizado após um outro de ordem N-1 e antecede um de ordem N+1.

### 8.2.1. DECLARAÇÃO DE ARQUIVOS

Um arquivo antes de ser usado precisa ser declarado. Essa declaração cria uma variável que poderá ser associada aos dados gravados no dispositivo físico (disco, fita etc). Para fazermos a declaração de um arquivo usamos o seguinte formato geral:

```
lista-de-identificadores : FILE OF tipo;
```

onde:

- lista-de-identificadores** são os nomes das variáveis que serão associadas aos arquivos que se deseja declarar;
- tipo** representa o tipo da variável correspondente aos registros do arquivo.

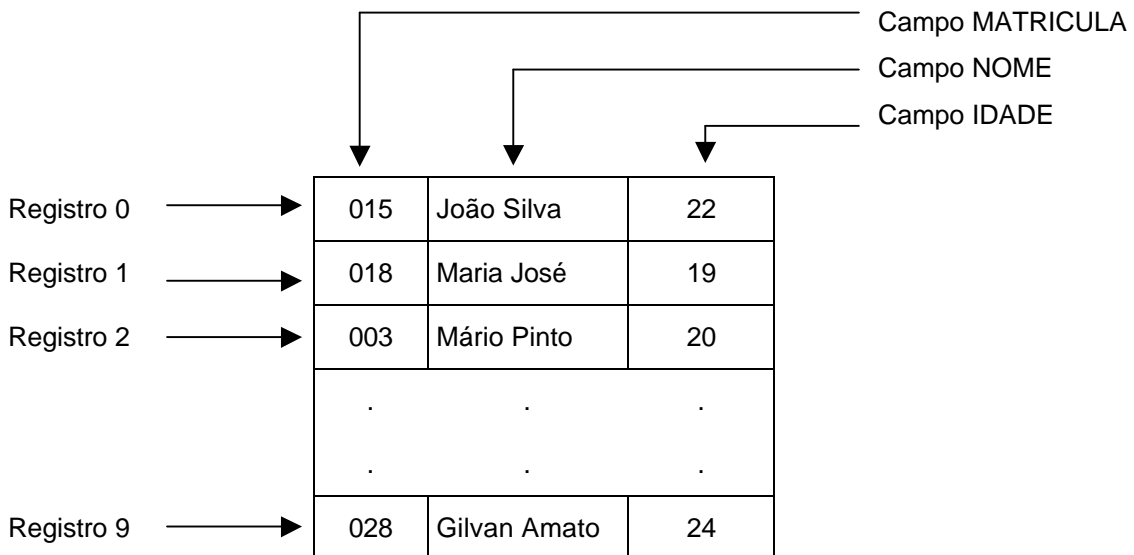
Exemplo:

```

Type
 REG = record
 MATRICULA : string[8];
 NOME : string[30];
 IDADE : byte;
 end;
Var
 ALUNOS : file of REG;

```

Estrutura do arquivo ALUNOS, após a sua declaração e utilização, supondo que existam 10 (dez) registros:



No exemplo anterior, podemos observar que o Pascal numera os registros começando pelo número 0 (zero), portanto o último registro terá sempre o número N-1 (sendo N a quantidade de registros do arquivo).

### 8.2.2. UTILIZAÇÃO DE ARQUIVOS

Para ser utilizado, um arquivo precisa, além de ser declarado, de uma série de tratamentos específicos, que são realizados pelos comandos :

**ASSIGN** - Este procedimento permite que associemos o nome externo de um arquivo a uma variável do tipo FILE. O nome externo é aquele utilizado pelo sistema operacional, portanto deve ser válido para o mesmo. São possíveis todas as formas usadas no PATH, e quando a unidade ou subdiretórios forem omitidos, estes assumiram o default. Após o uso do ASSIGN, este continuará valendo até que seja dado um novo ASSIGN. O tamanho máximo do nome do arquivo é de 79 caracteres. Este procedimento nunca deve ser usado em um arquivo já aberto. Sua sintaxe:

```
ASSIGN (VAR nome-pascal:FILE; nome-externo:STRING);
```

Exemplo:

```
ASSIGN(CADASTRO, 'A:\CADASTRO.ARQ');
```

**RESET** - Este procedimento permite-nos abrir um arquivo já existente. No caso do uso deste, para a tentativa de abertura de um arquivo não existente, haverá um erro de execução. Para que o procedimento tenha sucesso, é necessário que antes de executá-lo tenhamos utilizado o procedimento ASSIGN. Após o uso do RESET, o ponteiro de registros do arquivo será posicionado no registro 0 (primeiro registro). Sua sintaxe:

```
RESET (VAR arquivo:FILE);
```

Exemplo:

```
RESET(CADASTRO);
```

**REWRITE** - Este procedimento permite criar e abrir um novo arquivo. Caso o arquivo já exista, terá seu conteúdo eliminado e será gerado um novo arquivo. Antes de executarmos este procedimento, devemos usar o ASSIGN. Sua sintaxe:

```
REWRITE (VAR arquivo:FILE);
```

Exemplo:

```
REWRITE(CADASTRO);
```

**CLOSE** - Este procedimento permite que se feche um arquivo anteriormente aberto, só sendo permitido o fechamento de um arquivo por vez. Sua sintaxe:

```
CLOSE (VAR arquivo:FILE);
```

Exemplo:

```
CLOSE(CADASTRO);
```

Nos comandos apresentados, vemos uma deficiência: enquanto o RESET apenas nos permite a abertura de arquivos já existentes, o REWRITE apenas abre arquivos novos ou destrói um possível conteúdo anterior. Como poderemos então resolver essa situação? Bem, existe um modo relativamente simples, que consiste no uso de uma diretiva de compilação para checagem de erros de entrada e/ou saída, {\$!}. Esta diretiva retorna um código de erro em uma função do Turbo chamada IORESULT. Vejamos um exemplo prático da utilização da diretiva \$!:

```
Program TESTA_ABERTURA;

Uses
 crt;

Type
 REG = record
 COD : string[6];
 DESC : string[20];
 end;

Var
 ARQ : file of REG;
 NOMEARQ : string;

Begin

 clrscr;
 write('Entre com o nome do arquivo');
 readln(NOMEARQ);
 assign(ARQ,NOMEARQ);
 {$I-}
 reset(ARQ);
 {$I+}
 if ioresult = 0 then
 writeln('Arquivo aberto sem problemas')
 else
 begin
 rewrite(ARQ);
 writeln('Arquivo inexistente. Novo arquivo criado');
 end;
 close(ARQ);
 writeln('Arquivo fechado');
 readkey;

End.
```

Porém, para manipularmos um arquivo não basta apenas abri-lo e fechá-lo. Temos, na maioria das vezes, que ler uma informação contida nele, outras vezes registrar informações novas, ou ainda fazer alterações nas informações já existentes. Vejamos então os comandos que nos permitem tais tarefas:

**WRITE** - Este procedimento, além de ser usado para exibir dados no vídeo, pode ser usado para gravar informações em um arquivo. Após a execução deste procedimento, o ponteiro de registros do arquivo será deslocado um registro para a frente. Sua sintaxe:

```
WRITE (arquivo:FILE; registro:RECORD);
```

Exemplo:

```
WRITE(CADASTRO,REGISTRO);
```

## EXERCÍCIO RESOLVIDO

R8.03. Escreva um programa crie um novo arquivo chamado AGENDA.DAT contendo os campos NOME e FONE, e grave no mesmo 10 registros lidos pelo teclado.

```
Program CRIA_AGENDA;

Uses
 crt;

Type
 REGISTRO = record
 NOME : string[20];
 FONE : string[8];
 end;

Const
 N = 10;

Var
 AGENDA : file of REGISTRO;
 REG : REGISTRO;
 I : integer;

Begin
 clrscr;
 assign(AGENDA, 'AGENDA.DAT');
 rewrite(AGENDA);
 for i:=1 to N do
 begin
 with REG do
 begin
 write('Nome: ');
 readln(NOME);
 write('Fone: ');
 readln(FONE);
 end;
 write(AGENDA,REG);
 end;
 close(AGENDA);
 readkey;

End.
```

**READ** - Como já vimos anteriormente, este procedimento permite que atribuamos a uma variável um valor obtido por um dispositivo de entrada. Esse dispositivo pode ser também um arquivo. Após a execução deste procedimento, o ponteiro de registros do arquivo será deslocado um registro para a frente. Sua sintaxe:

```
READ (arquivo:FILE; registro:RECORD);
```

Exemplo:

```
READ(CADASTRO,REGISTRO);
```

## EXERCÍCIO RESOLVIDO

R8.04. Escreva um programa que abra o arquivo AGENDA.DAT criado no exercício anterior e exiba os 10 registros na tela do computador.

```
Program EXIBE_10;

Uses
 crt;

Type
 REGISTRO = record
 NOME : string[20];
 FONE : string[8];
 end;

Const
 N = 10;

Var
 AGENDA : file of REGISTRO;
 REG : REGISTRO;
 I : integer;

Begin

 clrscr;
 assign(AGENDA, 'AGENDA.DAT');
 reset(AGENDA);
 for i:=1 to N do
 begin
 read(AGENDA,REG);
 with REG do
 writeln(NOME, ' ', FONE);
 end;
 close(AGENDA);
 readkey;

End.
```

No exercício anterior, não tivemos problema algum ao executar a leitura no arquivo, pois este havia sido gravado por nós mesmos e com uma quantidade de registros predefinidos. Porém, na maioria das vezes, não temos quantidade de registros contidos num arquivo e, para esta situação, devemos saber quando chegarmos ao final de um arquivo. O Turbo Pascal nos fornece tal função:

**EOF** - Esta função nos retorna o valor TRUE quando for encontrada a marca de fim de arquivo. Sua sintaxe:

```
EOF (VAR arquivo:FILE) : boolean;
```

Exemplo:

```
FIM := EOF(CADASTRO);
```

## EXERCÍCIO RESOLVIDO

R8.05. Escreva um programa que abra o arquivo AGENDA.DAT criado anteriormente e exiba todos os seus registros na tela do computador (supondo não ser conhecida a quantidade de registros).

```

Program EXIBE_TODOS;
Uses
 crt;
Type
 REGISTRO = record
 NOME : string[20];
 FONE : string[8];
 end;
Var
 AGENDA : file of REGISTRO;
 REG : REGISTRO;

Begin
 clrscr;
 assign(AGENDA, 'AGENDA.DAT');
 reset(AGENDA);
 while not eof(AGENDA) do
 begin
 read(AGENDA, REG);
 with REG do
 writeln(NOME, ' ', FONE);
 end;
 close(AGENDA);
 readkey;
 End.

```

Porém, para a manutenção de alguns registros de um determinado arquivo, temos que saber qual a posição deste registro no arquivo e também qual a posição do último registro.

Quando estamos manipulando um arquivo, existe a todo momento um ponteiro que fica deslocando-se de acordo com as leituras e gravações, ou seja, a cada leitura ou gravação este ponteiro avança uma posição. Mas existem algumas funções e procedimentos que permitem a manipulação deste ponteiro. São elas:

**SEEK** - Este procedimento permite que movamos o ponteiro do arquivo para uma posição preestabelecida. Só pode ser usado em arquivos previamente abertos. Sua sintaxe:

```
SEEK (VAR arquivo:FILE; posição:LONGINT);
```

Exemplo:

```
SEEK(CADASTRO, 5); {posiciona o ponteiro no sexto registro do arquivo}
```

**FILEPOS** - Esta função nos retorna a posição atual do ponteiro do arquivo. Sua sintaxe:

```
FILEPOS (VAR arquivo:FILE) : LONGINT;
```

Exemplo:

```
PONTEIRO := FILEPOS(CADASTRO);
```

**FILESIZE** - Esta função nos retorna a quantidade de registros do arquivo. Sua sintaxe:

```
FILESIZE (VAR arquivo:FILE) : LONGINT;
```

Exemplo:

```
TAMANHO := FILESIZE(CADASTRO);
```

**EXERCÍCIO RESOLVIDO**

R8.06. Escreva um programa que faça a inclusão de registros em um arquivo chamado ALUNOS. Se o arquivo não existir, o programa deve criar um arquivo vazio e proceder a inclusão. A leitura da matrícula 0 (zero) indica fim dos dados.

“Lay-out” do arquivo: MATRICULA  
NOME  
MEDIA

Explicação da solução:

Deve-se mover o ponteiro para a posição posterior ao último registro e então efetuar a gravação dos dados lidos pelo teclado.

```
Program INCLUSAO;

Uses

 crt;

Type

 REGISTRO = record
 MATR : integer;
 NOME : string[35];
 MEDIA: real;
 end;

Var

 ARQ : file of REGISTRO;
 REG : REGISTRO;

Begin

 clrscr;
 assign(ARQ, 'ALUNOS');
 {$I-}
 reset(ARQ);
 {$I+}
 if ioresult > 0 then
 rewrite(ARQ);
 seek(ARQ, filesize(ARQ));

 write('Matricula: ');
 readln(REG.MATR);

 while REG.MATR <> 0 do
 begin
 write('Nome: ');
 readln(REG.NOME);
 write('Média: ');
 readln(REG.MEDIA);
 write(ARQ, REG);
 write('Matrícula: ');
 readln(REG.MATR);
 end;

 close(ARQ);

End.
```

R8.07. Escreva um programa que altera a média de um aluno no arquivo ALUNOS.

Explicação da solução:

Deve-se solicitar a digitação da matrícula do aluno que se deseja alterar e percorrer todo o arquivo comparando a matrícula de cada aluno com a matrícula digitada. Ao encontrar, exibe-se os dados atuais e solicita-se a digitação da nova média.

```
Program ALTERACAO;

Uses

 crt;

Type

 REGISTRO = record
 MATR : integer;
 NOME : string[35];
 MEDIA: real;
 end;

Var

 ARQ : file of REGISTRO;
 REG : REGISTRO;
 K : integer;
 RESP : char;

Begin

 assign(ARQ, 'ALUNOS');
 reset(ARQ);

 repeat

 clrscr;
 write('Matricula: ');
 readln(K);
 seek(ARQ,0);

 while not eof(ARQ) do
 begin
 read(ARQ,REG);
 with REG do
 if MATR = K then
 begin
 writeln('Nome: ',NOME);
 writeln('Média Atual: ',MEDIA:1:1);
 write('Nova Média: ');
 readln(MEDIA);
 seek(ARQ,filepos(ARQ)-1);
 write(ARQ,REG);
 end;
 end;

 write('Repetir(S/N)? ');
 RESP :=readkey;

 until upcase(RESP) = 'N'

 close(ARQ);

 End.
```

R8.08. Escreva um programa que abre o arquivo ALUNOS e exibe todos os alunos com média superior ou igual a 7.0:

Explicação da solução:

Deve-se acessar todos os registros e selecionar aqueles cuja média seja maior ou igual a 7.0.

```
Program EXIBICAO;

Uses
 crt;

Type
 REGISTRO = record
 MATR : integer;
 NOME : string[35];
 MEDIA: real;
 end;

Var
 ARQ : file of REGISTRO;
 REG : REGISTRO;

Begin
 clrscr;
 assign(ARQ, 'ALUNOS');
 reset(ARQ);

 while not eof(ARQ) do
 begin
 read(ARQ,REG);
 with REG do
 if MEDIA >= 7.0 then
 writeln(MATR, ' ', NOME, ' ', MEDIA:1:1);
 end;
 close(ARQ);
 readkey;
 end;

End.
```

R8.09. Escreva um programa que nos permita excluir registros do arquivo ALUNOS.

Explicação da solução:

Para que num arquivo se possa excluir registros é necessário que o mesmo possua um campo que funcione como um "flag", ou seja, um indicador de exclusão. Nesse campo, que normalmente é do tipo lógico (boolean), todos os registros estarão inicialmente com valor FALSE, indicando que nenhum registro deverá ser deletado. Para efetuarmos a exclusão, passaremos por duas etapas:

**1ª Etapa: EXCLUSÃO LÓGICA** – Ao especificarmos que um determinado registro deva ser deletado, devemos alterar o conteúdo do campo indicador de exclusão para o valor TRUE. Se quisermos excluir mais registros, faremos a mesma operação.

**2ª Etapa: EXCLUSÃO FÍSICA** – Após definirmos todos os registros que deverão ser deletados, devemos copiar todos os registros não marcados para deleção para um outro arquivo temporário. Feito isto, apagamos o arquivo original e renomeamos o arquivo temporário com o nome do arquivo original.

```
Program EXCLUSAO;

Uses
 crt;

Const
 NOMEARQ = 'ALUNOS';

Type
 REGISTRO = record
 MATR : integer;
 NOME : string[35];
 MEDIA: real;
 DEL : boolean; {indicador de exclusão}
 end;

Var
 ARQ,TMP : file of REGISTRO;
 REG : REGISTRO;
 K : integer;
 EXC,MAIS : char;

Begin

 {EXCLUSÃO LÓGICA}
 assign(ARQ,NOMEARQ);
 reset(ARQ);
 repeat
 clrscr;
 write('Matrícula: ');
 readln(K);
 seek(ARQ,0);
 while not eof(ARQ) do
 begin
 read(ARQ,REG);
 with REG do
 if MATR = K then
 begin
 writeln('Nome: ',NOME);
 write('Excluir (S/N)? ');
 EXC := upcase(readkey);
 writeln;
 if EXC = 'S' then
 DEL := true;
 seek(ARQ,FILEPOS(ARQ)-1);
 write(ARQ,REG);
 end;
 end;
 write('Excluir outro (S/N)? ');
 MAIS := upcase(readkey);
 until MAIS = 'N';

 {EXCLUSÃO FÍSICA}
 assign(TMP,'TMP');
 rewrite(TMP);
 seek(TMP,0);
 while not eof(ARQ) do
 begin
 read(ARQ,REG);
 if not REG.DEL then
 write(TMP,REG);
 end;
 close(ARQ);
 close(TMP);
 erase(ARQ); {deleta o arquivo original}
 rename(TMP,NOMEARQ); {renomeia o arquivo temporário}

End.
```

**EXERCÍCIOS PROPOSTOS**

P8.05. Elabore um programa que copie o arquivo ALUNOS (veja exemplos anteriores) em um outro chamado ALUNOS2.

P8.06. Elabore um programa que concatena (junta) dois arquivos com o mesmo "lay-out" em um terceiro.

P8.07. Escreva um programa que, a partir de um arquivo CAD já existente cujo "lay-out" é dado abaixo, determina quantas pessoas são do sexo feminino e quantas são do sexo masculino.

"Lay-out" do arquivo: MATRICULA  
NOME  
ENDERECO  
SEXO  
SALARIO

P8.08. Construa um programa que seleciona de um arquivo, gravando em outro, todos os registros cujo campo salário é maior que R\$ 500,00.

"Lay-out" do arquivo: MATRICULA  
NOME  
SALARIO

P8.09. Escreva um programa que procura por uma matrícula, informada pelo teclado, dentro do arquivo definido no problema anterior. Se for encontrada a matrícula desejada, devem ser exibidos o nome e o salário correspondentes à mesma.

P8.10. Escreva um procedimento que, a partir do início de um arquivo, apresente um determinado número de registros.

Arquivo MÓVEIS – Lay-out: CODIGO – string[2]  
DESCRICAO – string[20]

| Exemplo de Conteúdo: | CODIGO | DESCRICAO          |
|----------------------|--------|--------------------|
|                      | CG     | Cadeira giratória  |
|                      | MR     | Mesa Redonda       |
|                      | CP     | Cadeira presidente |
|                      | BB     | Bebedouro          |
|                      | EA     | Estante de aço     |
|                      | CM     | Cadeira de madeira |

Ativação do Procedimento: COMECO(MOVEIS,2)

Saída: CG Cadeira giratória  
MR Mesa Redonda

P8.11. Escreva um procedimento semelhante ao do problema anterior, que faça a exibição dos registros do arquivo MÓVEIS a partir de um determinado registro do mesmo, apresentando uma quantidade solicitada.

Ativação: EXTRAIR(MOVEIS,4,2)

Saída: BB Bebedouro  
EA Estante de aço

P8.12. Produza um procedimento que exiba os registros de um arquivo com exceção dos pertencentes a um intervalo dado.

Ativação: OMITIR(MOVEIS,2,4)

Saída: CG Cadeira giratória  
EA Estante de aço  
CM Cadeira de madeira

P8.13. Modifique o procedimento OMITIR do exercício anterior para gerar um arquivo de saída sem os registros a serem removidos.

## Capítulo 9

# CLASSIFICAÇÃO E PESQUISA

### 9.1. CLASSIFICAÇÃO

Uma das operações mais comuns em computadores como também na vida cotidiana, é a de classificação. Nesse tipo de operação tenta-se estabelecer uma ordem entre os elementos de uma lista.

Vejamos alguns exemplos:

#### Exemplo 1:

Coloque na ordem alfabética a relação de nomes a seguir:

MARIA, ANTONIO, JOSÉ, FRANCISCO

#### Exemplo 2:

Ponha os salários abaixo na ordem crescente:

1.700,00    1.000,00    1.200,00

Ao pedirmos para um grupo de pessoas resolverem os exemplos 1 e 2, vão, inevitavelmente, aparecer maneiras diferentes de resolver os referidos problemas. Essas maneiras diferentes de classificarmos listas de dados, recebem a designação de métodos de classificação. Nós vamos analisar alguns desses métodos para classificar dados:

**Observação:** para todos os métodos tratados estaremos considerando como objeto de classificação um vetor  $V$  de  $N$  componentes.

#### 9.1.1. MÉTODO DA BOLHA

Para  $I$  de 1 até  $N-1$ , compare  $V_i$  com  $V_{i+1}$ , trocando-os de posição se  $V_i > V_{i+1}$ . Isso faz com que o maior componente dentro do vetor vá para a extremidade mais à direita do mesmo (sobe como bolhas de óleo na superfície da água). Em seguida, passe outra vez pelo vetor, começando novamente com  $I$  igual a 1. Continua-se repetindo esse processo até que o vetor esteja classificado (não haja nenhuma troca). Vejamos o algoritmo:

```

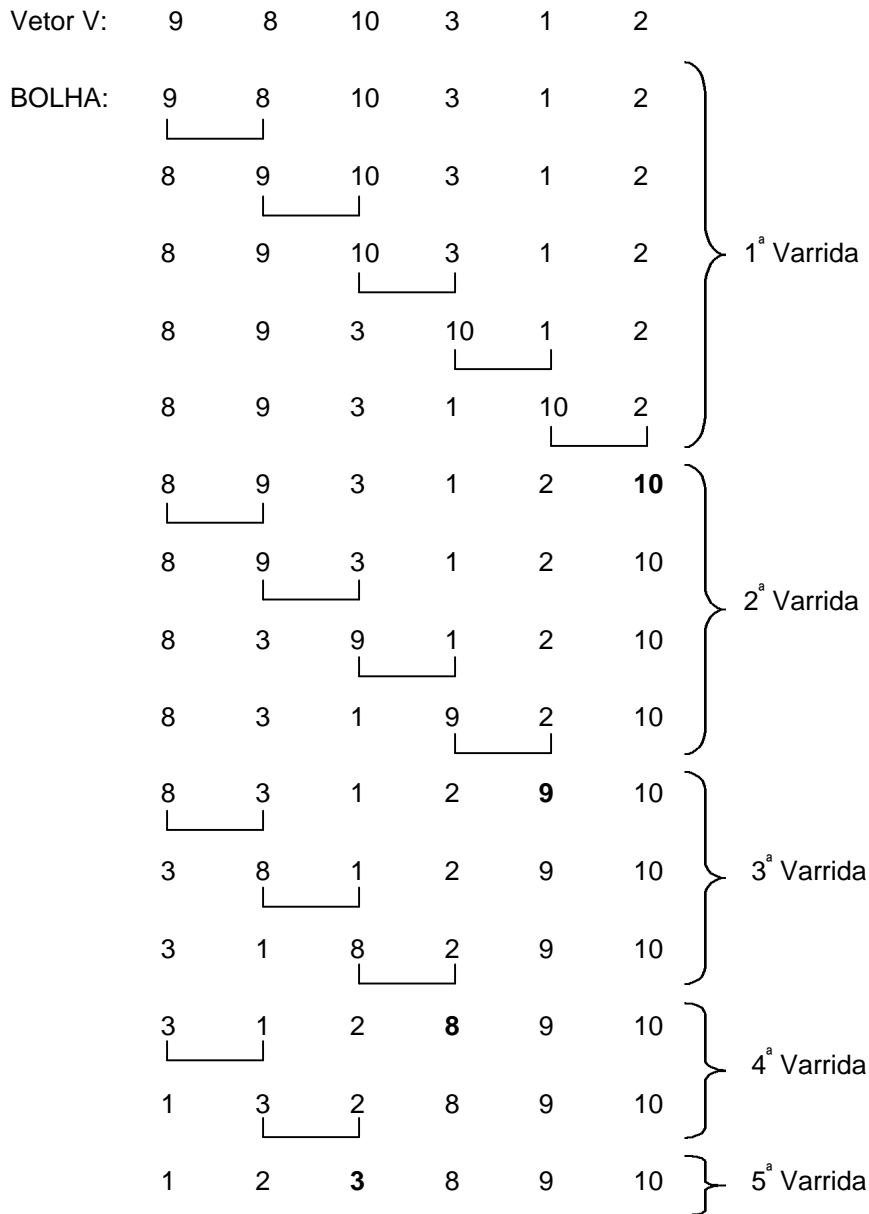
procedure BOLHA (V:vetor; N:integer);

var I, AUX, FIM : integer;
 TROCOU : boolean;

begin
 FIM := N;
 repeat
 TROCOU := false;
 for I:=1 to FIM-1 do
 if V[I] > V[I+1] then
 begin
 AUX := V[I];
 V[I] := V[I+1];
 V[I+1] := AUX;
 TROCOU := true;
 end;
 FIM := FIM-1;
 until not TROCOU;
 end;

```

Vamos analisar o comportamento do método sobre o vetor V:



O algoritmo detecta na 5ª varrida que não pode efetuar mais trocas.

### 9.1.2. CLASSIFICAÇÃO POR INSERÇÃO

Nesse método, classificam-se os sub-vetores sucessivos  $\{V_1\}$ ,  $\{V_1, V_2\}$ ,  $\{V_1, V_2, V_3\}$ , ... ,  $\{V_1, V_2, V_3, \dots, V_n\}$ . Em cada passagem, após a primeira, temos um sub-vetor classificado no qual queremos inserir um elemento. Assim, para  $I$  variando de 2 até  $N$ , começamos com  $\{V_1, \dots, V_{i-1}\}$  já classificado. A seguir, separamos o elemento  $V_i$ , inserindo-o na sua posição correta no sub-vetor  $\{V_1, \dots, V_{i-1}\}$ , obtendo o sub-vetor  $\{V_1, \dots, V_i\}$  classificado. Vejamos o algoritmo:

```

procedure INSERCAO (V:vetor; N:integer);
var I,J,X : integer;
begin
 for I:=2 to N do
 begin
 X := V[I];
 J := I-1;
 while (X < VET[J]) and (J >= 1) do
 begin
 VET[J+1] := VET[J];
 J:=J-1;
 end;
 VET[J+1] := X;
 end;
end;

```

Vamos analisar o comportamento do método sobre o vetor V:

```

Vetor V: 9 8 10 3 1 2

INSERÇÃO: 9 8 10 3 1 2
 └───┬───┘
 8 9 10 3 1 2
 └───┬───┘
 8 9 3 10 1 2
 └───┬───┘
 8 3 9 10 1 2
 └───┬───┘
 3 8 9 10 1 2
 └───┬───┘
 3 8 9 1 10 2
 └───┬───┘
 3 8 1 9 10 2
 └───┬───┘
 3 1 8 9 10 2
 └───┬───┘
 1 3 8 9 10 2
 └───┬───┘
 1 3 8 9 2 10
 └───┬───┘
 1 3 8 2 9 10
 └───┬───┘
 1 3 2 8 9 10
 └───┬───┘
 1 2 3 8 9 10

```

### 9.1.3. CLASSIFICAÇÃO POR SELEÇÃO

Essa técnica consiste em achar o menor dos N elementos da lista a ser classificada e trocá-lo pelo elemento número 1. Na segunda passagem, procura-se o segundo menor elemento no sub-vetor {V2, V3, ..., Vn} e o mesmo é trocado pelo elemento de ordem 2 e assim sucessivamente. Vejamos o algoritmo:

```

procedure SELECAO (V:vetor; N:integer);

var I,J,AUX,MENOR:integer;

begin
 for I:=1 to N-1 do
 begin
 MENOR := I;
 for J:=I+1 to N do
 if V[J] < V[MENOR] then
 MENOR := J;
 AUX := V[MENOR];
 V[MENOR] := V[I];
 V[I] := AUX;
 end;
 end;
end;

```

Vamos analisar o comportamento do método sobre o vetor V:

```

Vetor V: 9 8 10 3 1 2

SELEÇÃO: 9 8 10 3 1 2
 └──────────────────┘
 1 8 10 3 9 2
 └──────────────────┘
 1 2 10 3 9 8
 └──────────┘
 1 2 3 10 9 8
 └──────────┘
 1 2 3 8 9 10

```

## 9.2. COMPARAÇÃO DOS MÉTODOS DE CLASSIFICAÇÃO APRESENTADOS

Na comparação de métodos de classificação, existem alguns critérios que nos permitem avaliar a eficiência de um dado algoritmo:

espaço de memória utilizado - é um dos critérios menos importantes porque a maioria dos algoritmos de classificação exige, aproximadamente, a mesma quantidade de memória;

número médio de comparações (Ex: se  $V[i] > V[i+1]$ ) - calcula-se o número médio de comparações sobre todas as possíveis disposições iniciais dos elementos a serem classificados;

número médio de trocas efetuadas - determina-se a média de trocas de posições entre os elementos da lista a ser classificada;

comportamento no melhor e pior casos - há algoritmos que prevêm o aparecimento da lista já classificada (melhor caso) e aproveitam isso para ganhar tempo; o pior caso ocorre quando o objeto a ser classificado está na ordem inversa à desejada.

A eficiência de um algoritmo de classificação é influenciada, fortemente, pela quantidade de dados a classificar. Para uma quantidade pequena de elementos, praticamente, não há diferença de desempenho entre os métodos.

De uma maneira geral, os algoritmos menos complexos e que requerem uma quantidade menor de memória são menos eficientes que outros que utilizam estes recursos em maior quantidade.

A tabela a seguir compara os três métodos de classificação apresentados anteriormente, segundo os critérios discutidos nesta seção (a variável **N**, que aparece nas fórmulas, refere-se à quantidade de itens a classificar):

| Critério Analisado          | BOLHA                          | INSERÇÃO                       | SELEÇÃO                        |
|-----------------------------|--------------------------------|--------------------------------|--------------------------------|
| Memória                     | Depende da quantidade de itens | Depende da quantidade de itens | Depende da quantidade de itens |
| Número Médio de Comparações | $(N^2 - N) / 2$                | $(N^2 - N) / 4$                | $(N^2 - N) / 2$                |
| Melhor Caso                 | $N - 1$                        | $N - 1$                        | $(N^2 - N) / 2$                |
| Número Médio de Trocas      | $(N^2 - N) / 4$                | $(N^2 - N) / 4$                | $N - 1$                        |

A conclusão imediata que se chega, ao analisar os dados da tabela anterior, é a de que os três métodos discutidos possuem desempenho semelhantes, devendo ser usados para classificar conjuntos pequenos de dados porque o tempo de computação cresce muito para valores grandes de  $N$  (vide tabela a seguir).

| <b>N</b> | <b><math>(N^2 - N) / 2</math></b> |
|----------|-----------------------------------|
| 1        | 0                                 |
| 2        | 1                                 |
| 5        | 10                                |
| 10       | 45                                |
| 20       | 190                               |
| 100      | 4.950                             |
| 1000     | 499.500                           |

## EXERCÍCIOS PROPOSTOS

P9.01. Classifique a seguinte lista de valores usando o método da BOLHA: 85, 67, 3 e 12.

P9.02. Classifique a lista de nomes que segue, usando a classificação por INSERÇÃO:

MARIA, ANTÔNIO, JOÃO e RICARDO.

P9.03. Classifique os mesmos valores do item 01 usando a classificação por SELEÇÃO.

P9.04. Faça uma análise comparativa, do ponto de vista de desempenho, dos 3 métodos de classificação apresentados.

P9.05. Reescreva os 3 algoritmos de classificação de tal forma que passem a classificar na ordem decrescente, isto é, do maior elemento para o menor.

P9.06. Pesquise, descreva e apresente os algoritmos de, pelo menos, mais dois outros métodos de classificação diferentes dos apresentados.

### 9.3. PESQUISA

Uma outra operação muito utilizada em processamento de dados é a de pesquisa ou busca. A operação de busca consiste na procura por um dado elemento dentro de um universo conhecido.

Quem já não procurou alguma coisa ? Vejamos exemplos:

- 1) procurou seu nome na relação dos aprovados para Computação;
- 2) procurou por um número de casa em uma rua;
- 3) pesquisou nas páginas amarelas por uma livraria técnica.

Como no caso dos métodos de ordenação, aqui também existem maneiras diferentes para procurar ou buscar alguma coisa. Para apresentar os métodos de busca, utilizaremos um vetor de N posições.

#### 9.3.1. PESQUISA SEQUENCIAL

Consiste na procura de um dado elemento em um vetor não classificado. A pesquisa sequencial termina quando o elemento é encontrado ou quando o fim do vetor ocorre. Eis o algoritmo (em Pascal):

```
function SEQUENCIAL (V:vetor; N,K:integer):integer;

var
 ACHOU:boolean
 I,POS:integer;

begin
 ACHOU := false;
 POS := 0;
 I := 1;

 repeat

 if V[I] = K then
 begin
 POS := I;
 ACHOU := true;
 end
 else
 I := I+1;

 until ACHOU or (I > N);

 SEQUENCIAL := POS;

end;
```

#### 9.3.2. PESQUISA SEQUENCIAL ORDENADA

A diferença desse método para o anterior é que o vetor no qual se fará a pesquisa deve estar previamente classificado, permitindo que se abandone a busca quando se passar do ponto em que deveria estar o elemento procurado. A idéia básica aqui é a mesma usada quando se procura por um nome em uma relação classificada por ordem alfabética: se procuramos JOÃO SILVA, chegando, por exemplo, em MARIA RITA, devemos concluir que o nome procurado não está na relação.

Vejamos o algoritmo (em Pascal):

```
function SEQORD (V:vetor; N,K:integer) : integer;

var
 ACHOU,PASSOU : boolean;
 I,POS : integer;

begin

 ACHOU := false;
 PASSOU := false;
 POS := 0;
 I := 1;

 repeat
 if V[I] < K then
 I := I+1
 else
 if V[I] > K then
 PASSOU := true
 else
 begin
 POS := I;
 ACHOU := true;
 end
 until ACHOU or PASSOU or (I > N);

 SEQORD := POS;

end;
```

### 9.3.3. PESQUISA BINÁRIA

É um método de pesquisa que tem como pré-requisito a classificação do vetor que será objeto da busca. O método consiste em divisões sucessivas do espaço de pesquisa ao meio, conseguindo uma redução do tempo de busca.

```
function BINARIA (V:vetor; N,K:integer) : integer;

var
 ACHOU : boolean;
 INICIO,MEIO,FIM,POS : integer;

begin

 INICIO := 1;
 FIM := N;
 ACHOU := false;
 POS := 0;

 repeat
 MEIO := (INICIO + FIM) div 2;
 if V[MEIO] > K then
 FIM := MEIO - 1
 else
 if V[MEIO] < K then
 INICIO := MEIO + 1
 else
 begin
 POS := MEIO;
 ACHOU := true;
 end;
 until ACHOU or (INICIO > FIM);

 BINARIA := POS;

end;
```

## 9.4. COMPARAÇÃO DOS MÉTODOS DE BUSCA APRESENTADOS

O critério mais importante na comparação dos métodos de busca é o número médio de comparações.

Na determinação do número médio de comparações (se  $V[i] = K$ ), calcula-se o número médio de comparações para encontrar cada elemento do vetor a ser pesquisado.

A tabela a seguir compara os três métodos de busca discutidos anteriormente (a variável  $N$ , que aparece nas fórmulas, refere-se à quantidade de itens do vetor pesquisado).

| Critério Analisado                                     | SEQUENCIAL        | SEQ. ORDENADA | BINÁRIA                |
|--------------------------------------------------------|-------------------|---------------|------------------------|
| Número Médio de Comparações para Pesquisa Bem Sucedida | $N / 2$           | $(N / 2) + 1$ | $(\text{LOG}_2 N) + 1$ |
| Número Médio de Comparações para Pesquisa Incerta      | $N - (N * P / 2)$ | $(N / 2) + 2$ | $\text{LOG}_2 N + 1$   |

P Probabilidade da pesquisa ser bem sucedida

### CONCLUSÕES:

a busca binária possui desempenho melhor que os outros dois métodos;

para a pesquisa bem sucedida, os métodos seqüencial e seqüencial ordenado possuem, praticamente, o mesmo desempenho;

a busca binária e a seqüencial ordenada requerem um esforço adicional para classificar o vetor de pesquisa;

para vetores pequenos, os três métodos assemelham-se; à medida que  $N$  cresce, as diferenças no número médio de comparações cresce assustadoramente (veja tabela a seguir).

| N    | $\text{LOG}_2 N + 1$ | $N / 2$ | $(N+1) / 2$ |
|------|----------------------|---------|-------------|
| 2    | 2                    | 1       | 2           |
| 4    | 3                    | 2       | 3           |
| 8    | 4                    | 4       | 5           |
| 1024 | 11                   | 512     | 513         |

### EXERCÍCIOS PROPOSTOS

P9.07. Faça um programa que gere, aleatoriamente, um vetor  $V$  de 1000 elementos inteiros, leia um valor inteiro  $N$  e, usando a pesquisa seqüencial, verifique se existe algum elemento de  $V$  que seja igual a  $N$ .

P9.08. Faça um programa que gere, aleatoriamente, um vetor  $V$  de 1000 elementos inteiros, leia um valor inteiro  $N$  e:

- usando o método da bolha, classifique  $V$  (em ordem crescente);
- usando a pesquisa seqüencial ordenada, verifique se existe algum elemento de  $V$  que seja igual a  $N$ .

P9.10. Faça um programa que gere, aleatoriamente, um vetor  $V$  de 1000 elementos inteiros, leia um valor inteiro  $N$  e:

- usando o método da seleção ou inserção, classifique  $V$  (em ordem crescente);
- usando a pesquisa seqüencial binária, verifique se existe algum elemento de  $V$  que seja igual a  $N$ .

P9.11. Faça uma análise comparativa, do ponto de vista de desempenho, dos 3 métodos de pesquisa apresentados.

## APÊNDICE

# GLOSSÁRIO

**Algoritmo** - Conjunto ordenado de passos, não ambíguos e com a finalidade de resolver um determinado problema;

**Arquivo** - Conjunto de dados que possuem uma relação entre si. Por exemplo arquivo dos professores da escola X;

**Array** - Variável composta homogênea (de um mesmo tipo), podendo ser vetor ou matriz;

**Atribuição** - Operação pela qual se faz com que uma variável receba um valor constante ou de outra variável;

**Bloco** - Trecho de um algoritmo que possui características comuns;

**C** - Linguagem de programação muito utilizada para desenvolvimento de "software" básico. Geralmente é compilada;

**Cadeia** - Sinônimo para cadeia de caracteres, tendo por significado conjunto de caracteres;

**Campo** - Sub-divisão de um registro. Um campo pode conter um nome de uma pessoa, um salário etc;

**Caracter** - Uma letra, um dígito ou um símbolo qualquer que pertença ao conjunto de caracteres da linguagem;

**Clipper** - Inicialmente era apenas um compilador para a linguagem de programação do Dbase, tornando-se depois a linguagem de programação mais utilizada no Brasil. Apenas oferecida na forma compilada;

**Comando** - Um comando é uma ordem que deve ser dada para um computador executar. Toda linguagem de programação possui comandos.

**Comentário** - Explicações dadas no corpo do algoritmo para facilitar a compreensão de um trecho ou do algoritmo por inteiro;

**Compilador** - Programa tradutor cuja finalidade é converter um programa em uma linguagem de programação qualquer para uma forma que o computador possa entender. Essa tradução é gravada em disco, podendo ser utilizada outras vezes;

**Concatenação** - Operação de juntar dois ou mais conjuntos de caracteres. Por exemplo, a concatenação de "BON" com "ITA" forma a palavra "BONITA";

**Condição Lógica** - Condição que deve ser avaliada durante a execução do algoritmo para decidir o que deve ser feito;

**Constante** - Valor imutável dentro do algoritmo. Valores como  $PI = 3.1416$  não tem por que ser mudado durante a execução de algoritmo que calcula a área de um círculo;

**Contador** - Variável utilizada para contar alguma coisa ou ocorrência de um fato. Um contador é incrementado à medida em que a grandeza a que se destina contar cresce. Por exemplo, todas às vezes em que se acerta uma questão de uma prova, o contador de acertos aumenta de uma unidade;

**Dbase** - Gerenciador de bancos de dados para micros que foi, por muito tempo, um dos mais populares do mundo;

**Depuração** - Ato de seguir um algoritmo durante a sua execução para eliminação de erros;

**Indentação** - Conjunto de alinhamentos e recuos das linhas de um algoritmo que permitem melhor visualização e compreensão das relações hierárquicas entre as mesmas;

**Escopo** - Está associado ao conceito de bloco e diz respeito ao espaço de existência de uma variável dentro de um algoritmo;

**Expressão** - São construções algorítmicas especificadas por regras para calcular valores ou ter como resposta um valor lógico. Uma expressão pode ser lógica ou aritmética;

**Finalização** - Área de um algoritmo onde são feitos os preparativos para encerramento da execução do mesmo;

**Flag** - Significa bandeira ou estandarte em inglês. É uma variável usada para indicar a ocorrência de algum evento durante a execução do algoritmo, por exemplo, quando for atingido o fim do arquivo de funcionários faça com que a variável FIM\_ARQUIVO receba valor lógico VERDADEIRO;

**Fluxograma** - Representação gráfica de um algoritmo;

**Função** - Subalgoritmo que além de resolver uma dada tarefa, tem por obrigação mandar de volta para quem o chamou um valor que pode ser um número, uma cadeia de caracteres ou um valor lógico;

**Identificador** - Nome dado a variáveis ou a partes do algoritmo para efeito de identificação;

**Inicialização** - Atitude de atribuir a uma variável um valor inicial para começar o algoritmo. Pode-se falar, no corpo de um algoritmo, em uma área de inicialização que é aquela onde são inicializadas todas as variáveis a serem utilizadas;

**Interpretador** - Programa tradutor de um programa em uma linguagem de programação para uma forma que o computador possa entender. Essa tradução não é registrada sendo feita à medida em que o comando vai sendo executado. Tem maior aplicação para programadores iniciantes;

**Laço** - Sinônimo da palavra inglesa LOOP (laçada, fazer curvas). Significa a execução repetida de um conjunto de comandos por um número determinado de vezes.

**Matriz** - Array multidimensional (ver "array");

**Menu** - Significa cardápio de opções. É através de um menu que se oferecem opções de trabalho. Exibe-se as opções e lê-se a resposta do usuário para determinar o que deve ser feito;

**Módulo** - Subdivisão de um "software" que possui denominação e corpo próprios;

**Operador** - É um dos constituintes de uma expressão, podendo ser aritméticos, lógicos ou relacionais. Por exemplo: +, -, >, NOT;

**Otimização de Código** - Ato de reescrever um algoritmo da forma mais eficiente possível;

**Parâmetro** - Valor que é entregue a uma subrotina para guiar a ação da mesma;

**Pascal** - Linguagem de programação muito utilizada nos meios acadêmicos, apresentado-se como ideal para quem está aprendendo a programar;

**Passagem de Parâmetros** - Mecanismo utilizado para estabelecer a ligação entre um algoritmo e um subalgoritmo por ele chamado. A passagem de parâmetros permite que a solução do problema seja feita de forma genérica;

**Procedimento** - Subalgoritmo que visa a realização de uma determinada tarefa;

**Processamento** - Tratamento, transformação de dados. Em um algoritmo, a parte de processamento é quem realmente realiza as tarefas às quais o algoritmo destina-se;

**Produtório** - Variável utilizada para guardar produtos. Por exemplo: o produtório de todas as idades dos alunos da 8ª série deve ser inicializado com 1 (elemento neutro da multiplicação);

**Programa** - É um algoritmo traduzido para uma linguagem de programação;

**Programa Principal** - Parte de um programa que governa a execução do mesmo;

**Sistema** - Conjunto de partes interdependentes; conjunto pessoas, máquinas e programas que visam resolver uma aplicação do usuário; conjunto de programas;

**Software** - Termo da língua inglesa utilizado para designar a parte inteligente (relativa à programação) de um sistema de computação;

**Software Aplicativo** - Todos os programas que visam resolver os problemas relativos ao ramo de atividade fim de um indivíduo ou empresa;

**Software Básico** - Programas que são voltados para os problemas intrínsecos do computador. Não se prestam a resolver um problema fim do usuário. Exemplos: compiladores, sistemas operacionais, etc.;

**Somatório** - Variável utilizada para guardar somas. Por exemplo: o somatório das mesadas de todos os alunos da escola. Toda variável utilizada em somatórios é inicializada com valor 0 (zero) que é o elemento neutro da adição;

**String** - O mesmo que cadeia;

**Subalgoritmo** - Trecho de um algoritmo que recebe um identificador (nome), podendo ser ativado (executado) de qualquer ponto deste algoritmo pela simples menção do nome atribuído. A utilização de subalgoritmos gera, entre outras vantagens, economia no tamanho do algoritmo;

**Registro** - Unidade mínima de leitura/gravação dos dados de um arquivo. Todo registro é composto por campos;

**Recursividade** - Propriedade que possuem alguns problemas de serem definidos em termos deles mesmos. Para compreender melhor, analise o problema do fatorial de um número;

**Variável** - Conjunto de uma ou mais posições da memória de um computador utilizado para guardar valores que podem ser alterados durante a execução do algoritmo. Toda variável deve possuir um identificador (nome) e ter um tipo definido (numérico, carácter etc.);

**Variável de Laço** - Variável utilizada para controlar as repetições em um laço;

**Variável Global** - Variável que existe em todo o corpo do algoritmo. Possui escopo ilimitado;

**Variável Local** - Existe apenas dentro de um bloco, possuindo escopo restrito;

**Vetor** - Variável homogênea unidimensional (que possui um única dimensão);

## BIBLIOGRAFIA

- CARVALHO, Sérgio E. R. – *Introdução à Programação com Pascal* Campus, 1985.
- COLLINS, William J. – *Programação Estruturada com Estudos de Casos em Pascal*. McGraw-Hill, 1988.
- FARRER, Harry et al – *Algoritmos Estruturados*. Guanabara Dois, 1989.
- GRILLO, Maria Célia. A. – *Turbo Pascal 5.0 e 5.5*. LTC, 1991.
- GUIMARÃES & LAJES – *Algoritmos e Estruturas de Dados*. LTC, 1985.
- MANZANO, José Augusto N. G. & YAMATUMI, Wilson Y. – *Programando em Turbo Pascal 7.0*. Érica.
- PAIVA, S. R. – *Algoritmos, Técnicas de Programação e Estruturas de Dados*. Apostila da ASPER, 1995.
- PINTO, Wilson Silva – *Introdução ao Desenvolvimento de Algoritmos e Estruturas de Dados*. Érica, 1990.
- RINALDI, Roberto – *Turbo Pascal 7.0 Comandos e Funções*. Érica, 1993.
- SCHIMTZ, Eber A. & TELES, Antonio A. S. – *Pascal e Técnicas de Programação*. LTC, 1985.
- TREMBLAY, Jean-Paul & BUNT, Richard B. – *Ciência dos computadores - Uma abordagem Algorítmica*. Mcgraw-Hill, 1983.
- WIRTH, Niklaus – *Programação Sistemática em Pascal*. Campus, 1989.
- WORTMAN, Leon A. – *Programando em Turbo Pascal com Aplicações*. Campus, 1988.