

PROGRAMAÇÃO ESTRUTURADA DE COMPUTADORES

- Algoritmos Estruturados
- Pascal Estruturado
- Basic Estruturado
- Fortran Estruturado

ISBN 978-85-216-3380-6



9 788521 611806

ALGORITMOS ESTRUTURADOS

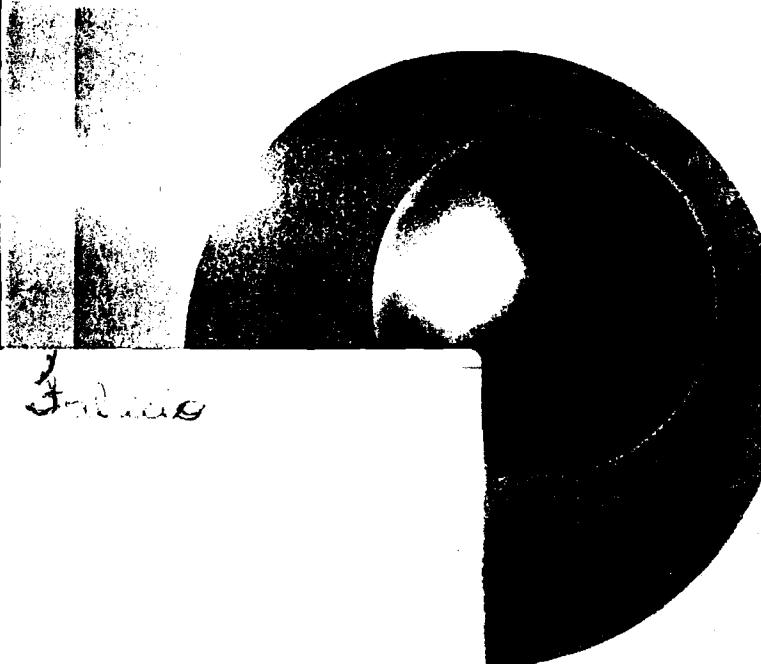
FARRER • B.
MATOS • S.

DC

HARRY FARRER
CHRISTIANO GONCALVES BECKER
EDUARDO CHAVES FARIA
HELTON FABIO DE MATOS
MARCOS AUGUSTO DOS SANTOS
MIRIAM LOURENÇO MAIA

ESTRUTURADA
DE COMPUTADORES

ALGORITMOS ESTRUTURADOS



DC

3.a
Edição

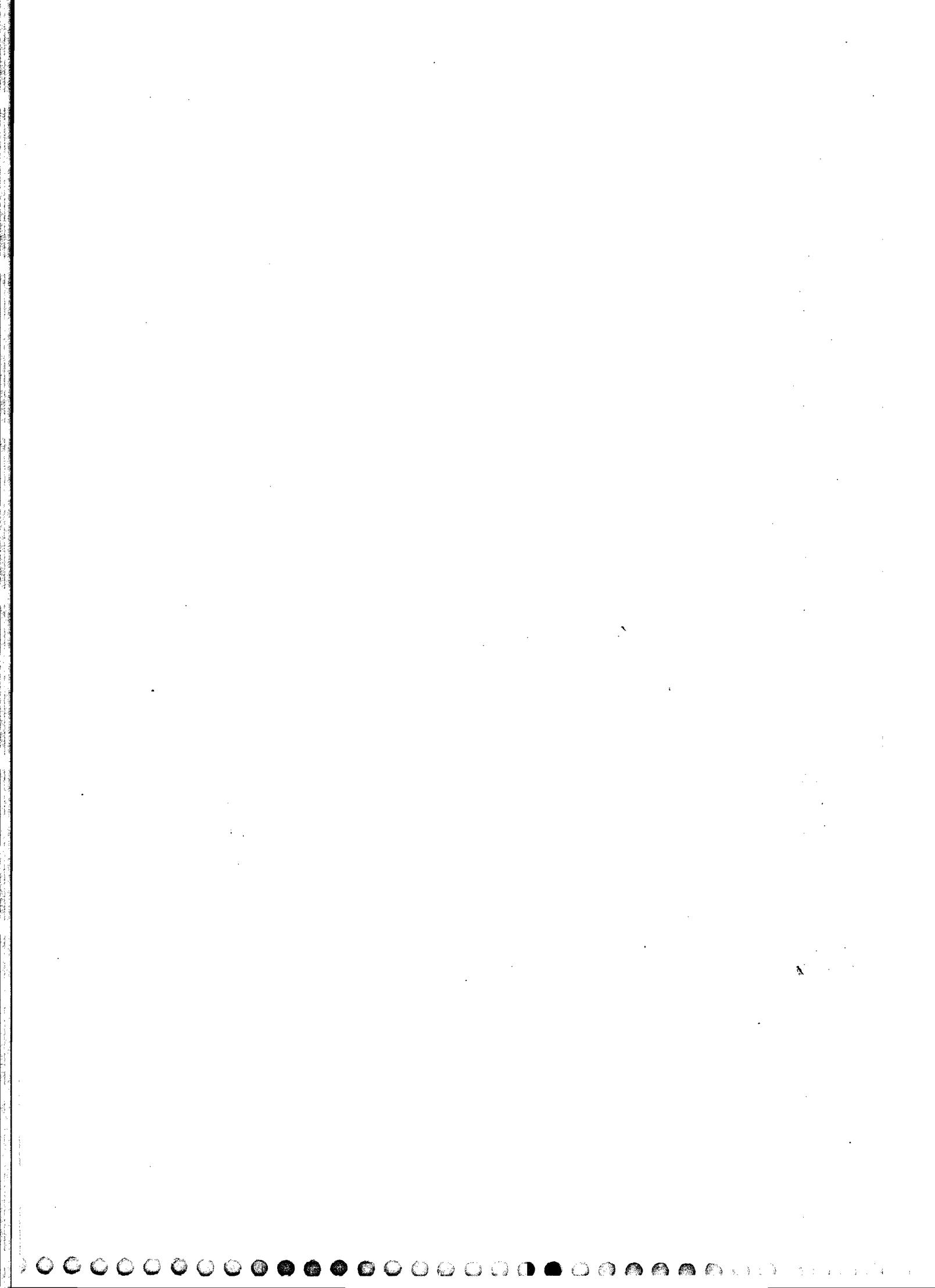


Programação Estruturada
de Computadores

Algoritmos Estruturados

A. Souza





Programação Estruturada
de Computadores

Algoritmos Estruturados

Harry Farrer
Christiano Gonçalves Becker
Eduardo Chaves Faria
Helton Fábio de Matos
Marcos Augusto dos Santos
Miriam Lourenço Maia

Professores do Departamento de Ciência da Computação da
Universidade Federal de Minas Gerais

UFMG — ICEx — DCC
Av. Antônio Carlos, 6627
31270-010 — Belo Horizonte — MG
tel: (31) 499-5860
fax: (31) 499-5858
e-mail: echaves@dcc.ufmg.br
www: <http://www.dcc.ufmg.br/~echaves>

Terceira edição



No interesse de difusão da cultura e do conhecimento, os autores e a editora emvidaram o máximo esforço para localizar os detentores dos direitos autorais de qualquer material utilizado, dispondo-se a possíveis acertos posteriores caso, inadvertidamente, a identificação de algum deles tenha sido omitida.

Escola N. S. de Fátima
B. Rui Barbosa
Nº 021529 Data 02-04-2007

Direitos exclusivos para a língua portuguesa
Copyright © 1999 by
LTC — Livros Técnicos e Científicos Editora S.A.
Travessa do Ouvidor, 11
Rio de Janeiro, RJ — CEP 20040-040
Tel.: 21-3970-9480
Fax: 21-2221-3202
ltc@ltceditora.com.br
www.ltceditora.com.br

Reservados todos os direitos. É proibida a duplicação ou reprodução deste volume, no todo ou em parte, sob quaisquer formas ou por quaisquer meios (eletrônico, mecânico, gravação, fotocópia, distribuição na Web ou outros), sem permissão expressa da Editora.

Prefácio da Terceira Edição

Hoje em dia, mais do que em qualquer outra época, a Informática sofre profundas transformações em decorrência das novas tecnologias de hardware e das novas técnicas e paradigmas de desenvolvimento de software. Contudo, apesar de toda e qualquer inovação, a porta de entrada para esta área ainda continua sendo o aprendizado da construção de algoritmos, ou, como querem outros, do aprendizado da lógica de programação ou, ainda, da expressão do raciocínio lógico. Também, quanto à forma desse aprendizado, nada mudou. Aprender a lógica de programação é aprender a desenvolver algoritmos de forma organizada e sistemática. Esse aprendizado depende muito mais do aluno do que de qualquer outro auxílio. Cabe ao professor introduzir os conceitos básicos envolvidos e guiar o aluno na prática de construir algoritmos gradativamente maiores e mais complexos. Para isso, é indispensável um material didático que ofereça o máximo de exemplos e exercícios de forma a viabilizar essa prática dentro da correta metodologia.

Além de eventuais correções e pequenas alterações no texto já existente, esta terceira edição de *Algoritmos Estruturados* inclui um apêndice com soluções para vários Exercícios Propostos no livro. Foi usado o critério de selecionar um exercício em cada classe de problemas semelhantes. Recomendamos que o leitor tente resolver um Exercício Proposto e somente depois consulte a solução no final do livro. Se houver diferença entre as soluções, o leitor deve usar o bom senso para determinar se cometeu algum erro ou se ambos os algoritmos são alternativas possíveis para o problema.

Esta terceira edição de *Algoritmos Estruturados* permanece compatível com os demais volumes desta série dedicados ao estudo das linguagens de programação. Nesse caso, se já existir a terceira edição de um desses volumes, esta seguirá a mesma ordem de assuntos, manterá a mesma numeração para os capítulos, parágrafos, exemplos e exercícios, e conterá um apêndice com programas para os mesmos Exercícios Propostos resolvidos neste volume inicial.

201.4a.1
F 245 p
3 ed.
3.6x

Tradicionalmente, o ensino da programação de computadores tem-se confundido com o ensino de uma linguagem de programação, de tal maneira que, há alguns anos, ensinar COBOL ou FORTRAN era a mesma coisa que ensinar a programar.

Embora tais linguagens sejam instrumentos possíveis para a expressão do raciocínio algorítmico e para a sua execução automática por um computador (Dijkstra, 1976), sua utilização geralmente é uma condição limitante para os que se iniciam na arte da programação. As primeiras linguagens de uso generalizado (FORTRAN, COBOL, BASIC etc.) foram projetadas com o principal objetivo de viabilizarem a expressão, em um nível mais elevado, dos programas a serem executados em computador. Isto conduziu a linguagens que devem obedecer a um conjunto amplo de regras extremamente rígidas. Mesmo as linguagens projetadas segundo os conceitos da Programação Estruturada (ALGOL, PASCAL, C, MODULA-2 etc.) ainda limitam muito a forma de expressão do raciocínio lógico.

A arte de programar consiste na arte de organizar e dominar a complexidade (Dijkstra, 1972). Para alcançar esses objetivos, o programador deve servir-se de sucessivas abstrações. A utilização das linguagens de programação para expressar as várias etapas do raciocínio algorítmico força o iniciante a se preocupar excessivamente com aspectos pouco significativos para o desenvolvimento do raciocínio (as regras de sintaxe, por exemplo). Acrescente-se o fato de que o raciocínio apoiado estritamente em uma linguagem de programação não percebe a importância da regra áurea de Dijkstra — “dividir para reinar” —, pois as linguagens disponíveis não facilitam o uso de refinamentos sucessivos.

A tarefa de programação, em última análise, resume-se no desenvolvimento de um raciocínio lógico e, como tal, exige do programador uma boa dose de criatividade. A liberdade de expressão é essencial ao desenvolvimento da criatividade. Segundo Wirth, a linguagem exerce forte e inegável influência nos processos do pensamento e, efetivamente, define, limita e deforma o espaço no qual são formulados (Wirth, 1974).

Tudo isso leva a concluir que uma ferramenta mais adequada para o desenvolvimento de algoritmos deve ser livre e desprovida dos detalhes pouco significantes para a expressão do raciocínio lógico. Como este não é o caso das atuais linguagens de programação, a solução é utilizar uma notação especial que sirva como instrumento intermediário no processo da elaboração de programas. Obviamente, o modelo lógico obtido através dessa notação deve ser transportado com facilidade para qualquer das linguagens de programação disponíveis, de maneira que sua utilização não seja mais um inconveniente para o programador. Assim, outra característica de uma boa notação algorítmica é a facilidade de codificação posterior. Uma vantagem adicional no uso de notação adequada ao desenvolvimento de algoritmos é conduzir o programador a expressar o seu raciocínio lógico independente da linguagem de programação.

Resumindo, simplicidade, poder de expressão e coerência são importantes aspectos de uma notação cujo principal objetivo é tornar-se um instrumento para o desenvolvimento e expressão do raciocínio lógico. A quantidade de estruturas de controle deve ser restrita a um mínimo que seja compatível com a potencialidade da notação. Esse conjunto mínimo deve

ser, na medida do possível, implementado sem esforço nas linguagens de programação existentes. Por fim, a notação deve incorporar os conceitos já consagrados da *programação estruturada*.

A equipe de professores que definiu e sedimentou a notação algorítmica apresentada neste livro acredita que tenha atingido os objetivos aqui propostos. De qualquer forma, um indício de que realmente se trata de uma ferramenta adequada foi o resultado obtido na sua aplicação a milhares de alunos nestes últimos anos, comparado com o observado anteriormente.

Os Autores

Programação Estruturada de Computadores é o resultado da experiência adquirida pelos seus autores no ensino e na prática do uso acadêmico e profissional dos computadores eletrônicos. Destina-se não apenas aos estudantes que iniciam o curso universitário, mas a todas as pessoas que desejam ou precisam adquirir o conhecimento destas máquinas e de sua utilização, através de uma ou mais linguagens de programação.

Programação Estruturada de Computadores estará sendo apresentado em diversos volumes. O volume inicial, *Algoritmos Estruturados*, que serve de base para todos os demais, faz uma introdução à Ciência da Computação, com um histórico e uma descrição dos computadores, oferecendo em seguida uma metodologia simples, gradativa e sistemática, destinada ao desenvolvimento de algoritmos. Esta metodologia visa atingir os objetivos da denominada Programação Estruturada, com um produto final claro, de fácil elaboração e de fácil entendimento, partindo de uma abordagem global do problema e obtendo módulos de uma forma natural, através de refinamentos sucessivos. Procurou-se expressar os algoritmos com a utilização de poucas formas, que fossem claras, concisas, gerais, simples e espontâneas.

Os volumes seguintes tratam (cada um) de uma linguagem de programação, seguindo a mesma ordem de assuntos deste volume. É obedecida a mesma numeração para os capítulos, parágrafos, exemplos etc., evitando-se a repetição de conceitos já estudados, facilitando-se a sua consulta e, sobretudo, permitindo um aprendizado unificado das diversas linguagens. Para isto, na sua maioria, os exemplos e exercícios são os mesmos nos diversos volumes, tendo sido escolhidos cuidadosamente para ilustrar os itens estudados e, ao mesmo tempo, mostrar a variedade de problemas que podem ser levados a um computador.

Espera-se que o leitor desta obra adquira uma disciplina de pensamento e de raciocínio que lhe seja útil, não apenas no seu contato com os computadores, mas também para enfrentar os difíceis problemas que a vida profissional e pessoal lhe venha a apresentar.

Os Autores

Conteúdo

0 Introdução à Ciência da Computação, 1

0.1. Evolução Tecnológica dos Computadores, 1

0.2. O Computador, 1

 0.2.1. *A Estrutura de um Computador Digital*, 2

 0.2.2. *Memória*, 3

 0.2.3. *Unidade Central de Processamento (UCP)*, 6

 0.2.4. *Periféricos*, 7

0.3. Algoritmo, 14

 0.3.1. *Conceituação*, 14

 0.3.2. *Exercícios de Fixação*, 15

 0.3.3. *Refinamentos Sucessivos*, 15

 0.3.4. *Exercícios de Fixação*, 17

0.4. Algoritmos Estruturados, 19

0.5. Linguagens de Programação, 22

1 Itens Fundamentais, 29

1.1. Constantes, 29

 1.1.1. *Constante Numérica*, 29

 1.1.2. *Constante Lógica*, 30

 1.1.3. *Constante Literal*, 30

 1.1.4. *Exercício de Fixação*, 30

1.2. Variáveis, 30

 1.2.1. *Formação dos Identificadores*, 31

 1.2.2. *Declaração de Variáveis*, 31

 1.2.3. *Exercícios de Fixação*, 32

1.3. Comentários, 32

1.4. Expressões Aritméticas, 32

 1.4.1. *Funções*, 33

 1.4.2. *Exercício de Fixação*, 35

1.5. Expressões Lógicas, 35	2.4.2. <i>Organização de Arquivos</i> , 149
1.5.1. <i>Relações</i> , 35	2.4.3. <i>Declaração</i> , 149
1.5.2. <i>Exercício de Fixação</i> , 36	2.4.3.1. <i>Exercícios de Fixação</i> , 150
1.5.3. <i>Operadores Lógicos</i> , 36	2.4.4. <i>Abertura de Arquivo</i> , 150
1.5.4. <i>Prioridade</i> , 38	2.4.4.1. <i>Exercício de Fixação</i> , 151
1.5.5. <i>Exercícios de Fixação</i> , 39	2.4.5. <i>Fechamento de Arquivo</i> , 151
1.6. Expressões Literais, 39	2.4.6. <i>Organização Seqüencial</i> , 152
1.6.1. <i>Exercício de Fixação</i> , 40	2.4.6.1. <i>Comando de Entrada</i> , 152
1.7. Comando de Atribuição, 40	2.4.6.2. <i>Comando de Saída</i> , 152
1.7.1. <i>Exercícios de Fixação</i> , 41	2.4.6.3. <i>Exercícios de Fixação</i> , 159
1.8. Comandos de Entrada e Saída, 41	2.4.7. <i>Organização Direta</i> , 160
1.8.1. <i>Exercício de Fixação</i> , 44	2.4.7.1. <i>Comando de Entrada</i> , 161
1.9. Estrutura Seqüencial, 44	2.4.7.2. <i>Comando de Saída</i> , 162
1.10. Estrutura Condicional, 45	2.5. Exercícios Propostos, 164
1.10.1. <i>Estrutura Condicional Simples</i> , 45	2.5.1. <i>Variáveis Compostas Unidimensionais</i> , 164
1.10.2. <i>Estrutura Condicional Composta</i> , 46	2.5.2. <i>Variáveis Compostas Multidimensionais</i> , 166
1.10.3. <i>Exercícios de Fixação</i> , 51	2.5.3. <i>Registros</i> , 170
1.11. Estrutura de Repetição, 53	2.5.4. <i>Arquivos</i> , 174
1.11.1. <i>Exercícios de Fixação</i> , 56	3 Modularização, 175
1.11.2. <i>Exercício de Fixação</i> , 62	3.1. Introdução, 175
1.12. Exercícios Propostos, 75	3.2. Ferramentas para Modularização, 177
2 Estruturas de Dados, 87	3.2.1. <i>Sub-rotina</i> , 179
2.1. Introdução, 87	3.2.2. <i>Função</i> , 183
2.2. Variáveis Compostas Homogêneas, 90	3.3. Considerações sobre a Modularização de Programas, 204
2.2.1. <i>Variáveis Compostas Unidimensionais</i> , 93	3.4. Exercícios Propostos, 204
2.2.1.1. <i>Declaração</i> , 93	4 Conclusões, 209
2.2.1.2. <i>Exercícios de Fixação</i> , 108	4.1. Definição do Problema, 209
2.2.2. <i>Variáveis Compostas Multidimensionais</i> , 110	4.2. Desenvolvimento de um Algoritmo, 209
2.2.2.1. <i>Declaração</i> , 111	4.3. Codificação, 209
2.2.2.2. <i>Exercícios de Fixação</i> , 131	4.4. Digitação, 209
2.3. Variáveis Compostas Heterogêneas, 131	4.5. Processamento do Programa, 210
2.3.1. <i>Registros</i> , 131	4.6. Análise dos Resultados, 210
2.3.1.1. <i>Declaração</i> , 133	Bibliografia, 282
2.3.2. <i>Conjunto de Registros</i> , 134	Índice Alfabético, 283
2.3.2.1. <i>Declaração</i> , 135	
2.4. Arquivos, 147	
2.4.1. <i>Conceito de Arquivo</i> , 148	

Programação Estruturada
de Computadores

Algoritmos Estruturados

Introdução à Ciência da Computação

A tarefa de processamento de dados consiste em tomar certa informação, processá-la e obter o resultado desejado.

Desde que o homem começou a processar dados, ele tentou construir máquinas para ajudá-lo em seu trabalho. O computador eletrônico é o resultado dessas tentativas que vêm sendo realizadas através dos séculos. Ele é, sem dúvida alguma, um dos principais produtos da ciência do século XX.

0.1. EVOLUÇÃO TECNOLÓGICA DOS COMPUTADORES

Os primeiros computadores utilizavam circuitos eletromecânicos e válvulas. O aparecimento do transistor (Bell Telephone Laboratories, 1948) trouxe a redução do tamanho e da potência consumida em relação às válvulas, além de serem dispositivos mais robustos e confiáveis. Os computadores utilizando esta tecnologia são classificados como de segunda geração. O domínio da tecnologia da física do estado sólido permitiu a integração de vários transistores em uma única embalagem com aproximadamente as mesmas dimensões de um único transistor. Surgiram então os circuitos integrados que foram responsáveis pelo aparecimento dos computadores de terceira geração. Estes computadores tinham maior potência de cálculo, eram mais rápidos, mais confiáveis e menores fisicamente do que seus antecessores da segunda geração.

Atualmente, o processo de integração tem praticamente o mesmo custo para se integrar dezenas, centenas ou milhares de transistores em uma única pastilha. Pode-se falar então na quarta geração de computadores pela utilização da integração em altíssima escala (VLSI).

Ao mesmo tempo, as telecomunicações se desenvolveram enormemente pelo uso da mesma tecnologia. Isto viabilizou a utilização de recursos de telecomunicações aplicados à computação, e vice-versa. O efeito imediato foi a possibilidade de interligação de sistemas de computação, do uso à distância de um computador por um ou vários usuários.

0.2. O COMPUTADOR

Existem dois tipos de computadores: os **analógicos** (do grego *analogos*, que significa proporcionado) e os **digitais** (do latim *digitus*, que significa dedo).

Um termômetro é um exemplo de funcionamento por analogia, onde a dilatação de mercúrio é analoga à mudança de temperatura.

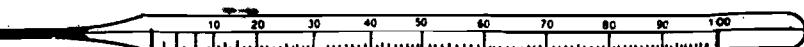


Fig. 0.1 Termômetro.

O princípio de analogia é utilizado também numa régua de cálculo, que pode ser considerada como um computador analógico, onde as operações de multiplicação e divisão são executadas simplesmente somando-se ou subtraindo-se a distância medida na régua externa, àquela marcada na régua interna deslizante.

Um dos primeiros "computadores" analógicos de grande porte foi construído por Lord Kelvin, em 1872, para prever a altura das marés nos portos ingleses. Polias e pesos simulavam os efeitos do Sol, da Lua e dos ventos nas marés. Os resultados eram mostrados em gráficos que indicavam a variação das marés.

Os computadores analógicos de hoje são capazes, por exemplo, de simular as forças que atuam numa represa ou as forças que atuam nas asas de um avião a partir da voltagem elétrica.

O que caracteriza o computador analógico é o fato de lidar com grandezas contínuas. Nele, as variáveis do problema são representadas por tensões, que são quantidades físicas contínuas.

Do ponto de vista de utilização, a característica principal do computador analógico está na rapidez com que as informações são processadas.

Ao contrário dos computadores analógicos, que trabalham com grandezas físicas, os computadores digitais são capazes de somar, subtrair, multiplicar, dividir e comparar através de pulsações elétricas que, em última análise, representam os dígitos 0 (ausência de corrente) e 1 (presença de corrente).

Os computadores digitais são os mais utilizados e é deles que este livro tratará.

0.2.1. A estrutura de um computador digital

O esquema da estrutura de um computador digital é dado pela figura 0.2.

Unidade de entrada. Esta unidade traduz informação de uma grande variedade de dispositivos em um código que a unidade central de processamento é capaz de entender. Em outras palavras, ela é capaz de traduzir letras, números, imagens, marcas ou tinta magnética em padrões de pulsos elétricos que são compreensíveis ao computador.

Memória. A memória é capaz de armazenar não só os dados, mas também o programa que irá "manipular" estes dados.

Unidade lógica e aritmética. Nesta unidade são feitos todos os cálculos aritméticos e qualquer manipulação de dados, sejam eles numéricos ou não.

Unidade de controle. É a unidade responsável pelo "trânsito" dos dados. Ela obtém dados armazenados na memória e interpreta-os. Controla a transferência de dados da memória para a unidade lógica e aritmética, da entrada para a memória e da memória para a saída.

Unidade de saída. Os dados processados são convertidos, por esta unidade, de impulsos elétricos em palavras ou números que podem ser "escritos" em impressoras ou "mostrados" em vídeos ou numa série de outros dispositivos.

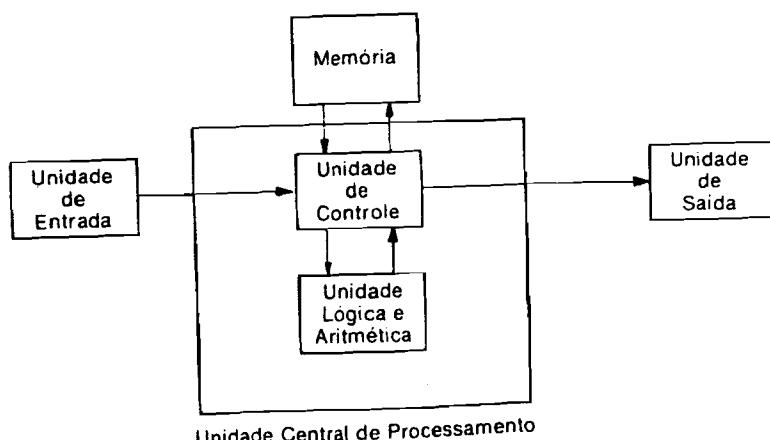


Fig. 0.2 Unidade Central de Processamento.

Ao conjunto de circuitos eletrônicos e dispositivos mecânicos dá-se o nome de *hardware*, que significa ferramenta e é usado como referência à máquina, propriamente dita, e não à sua utilização. Será descrito a seguir cada um destes componentes e seu respectivo funcionamento.

0.2.2. Memória

A Unidade Central de Processamento de um computador geralmente utiliza uma memória de alta velocidade para armazenar, temporariamente, dados e programas que estão sendo processados, já que qualquer programa, para ser executado, tem de estar nesta memória; além disso, as unidades de controle e aritmética e lógica se comunicam com cada um de seus bytes. Esta memória costuma ser denominada "memória temporária", "memória de trabalho" ou "memória principal".

Ao menor item de informação binária dá-se o nome de BIT. A palavra *bit* é uma contração de *binary digit* (dígito binário) significando um dígito que pode assumir um dos dois valores ou estados diferentes 0 ou 1, tal como um dígito decimal pode assumir um dos dez valores 0, 1, 2... ou 9.

Os bits são agrupados de modo a possibilitar ao usuário representar os dados e programas que deseje. Denomina-se BYTE a um conjunto de bits e PALAVRA a um conjunto de bytes. O número de bits que formam um byte, bem como o número de bytes que formam uma palavra, não é fixo e depende exclusivamente da máquina considerada. No caso de bytes, a situação mais usual é de encontrá-los constituídos de 8 bits.

A memória do computador é constituída por um conjunto de bits, sendo que o byte, ou palavra, recebe um endereço a fim de que os programas possam fazer referência aos seus dados.

A capacidade de um computador normalmente é medida pelo tamanho de sua memória. Usualmente, a unidade utilizada para fazer tal medida é o "byte".

Cada 1024 bytes representam 1 Kbyte (kilobyte). Cada 1024 Kbytes representam 1 Mbyte (megabyte). Cada 1024 Mbytes representam 1 Gbyte (gigabyte). Os computadores atuais podem variar, em termos de memória, desde algumas dezenas de Kbytes a vários Gbytes.

Os dados e programas que necessitam ser armazenados para utilização futura ou periódica são armazenados em outros tipos de memória. Estas, por sua vez, são denominadas "memórias de armazenamento secundário" ou "memórias auxiliares". Neste caso, deve ser levado em consideração o fato de que os dados e programas não podem ser perdidos ao ser desligado o sistema.

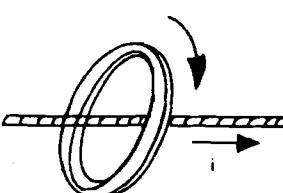
Serão apresentados a seguir alguns tipos de memória quanto às suas principais características e princípios de funcionamento.

Memórias magnéticas. A tecnologia para memória principal que predominou entre os computadores mais antigos foi a memória de núcleos ou memória magnética.

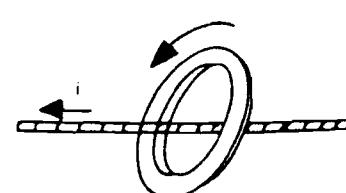
Uma memória deste tipo é constituída de um conjunto de minúsculos anéis de ferrite magnetizáveis, tal qual é mostrado na figura 0.3.

Sabe-se da Física que a corrente elétrica ao passar por um fio cria um campo magnético em torno do fio, no sentido horário ou anti-horário, dependendo do sentido da corrente. Portanto, este campo magnético estará presente no núcleo de ferrite que envolve o fio e pode-se convencionar os dois sentidos de magnetização como sendo "0" e "1".

Um grande número de núcleos de ferrite é montado na forma de matriz com fios de endereçamento passando por cada núcleo (figura 0.4). A memória é formada por um grupo destas matrizes interligadas



a) Magnetização horária



b) Magnetização anti-horária

Fig. 0.3 O núcleo de ferrite.

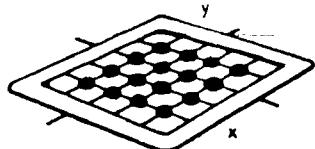


Fig. 0.4 Matriz de núcleos de ferrete.

adequadamente, onde cada núcleo de ferrete representa um bit. Assim, a partir de impulsos elétricos, é possível mudar a magnetização de cada um dos bits e, portanto, armazenar dados.

É claro que os dados não podem ser armazenados diretamente na memória, eles precisam antes ser codificados. O código utilizado é, em última análise, uma combinação de dígitos binários. Como exemplo, é mostrada na figura 0.5 a representação da letra R, na codificação EBCDIC, em uma porção de memória magnética.

O bit de paridade é um bit adicional colocado em cada byte de modo que o número de 1's seja sempre ímpar e o número de 0's seja sempre par, isto para o caso de sistema com paridade ímpar.

Assim, no exemplo representado na figura 0.5, tem-se:

1 1 0 1 1 0 0 1 0

Um outro exemplo é o caractere especial "=", que, representado em EBCDIC, seria:

0 1 1 1 1 1 0 1

Com este mecanismo é possível ao computador verificar eletronicamente se a configuração está correta. Qualquer anormalidade é acusada como erro de paridade.

Memórias semicondutoras. Com a evolução da eletrônica, os computadores, que inicialmente eram equipamentos grandes e lentos, tornaram-se compactos e rápidos graças ao desenvolvimento dos circuitos integrados que, nos dias de hoje, concentram milhares de componentes. Esta evolução também se faz presente na tecnologia de fabricação das memórias.

Os computadores mais recentes utilizam as denominadas memórias semicondutoras, que se apresentam na forma de um circuito integrado.

A memória semicondutora, como todo circuito integrado, possui um certo número de pinos através dos quais é feita a comunicação com o mundo exterior. De uma forma geral, os pinos das memórias podem ser classificados em quatro grupos (figura 0.6), a saber:

- pinos de entrada de dados;
- pinos de saída de dados;
- pinos de endereçamento (indicam a qual posição de memória está-se tendo acesso);
- pinos de comando e controle.

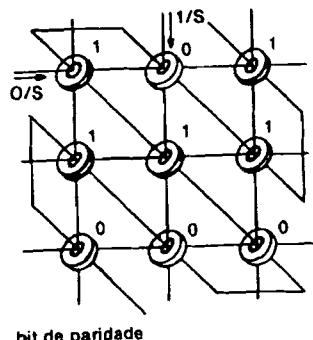


Fig. 0.5 Representação da letra R em EBCDIC.

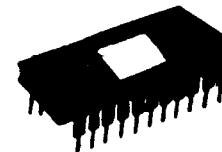


Fig. 0.6.a Circuito integrado.

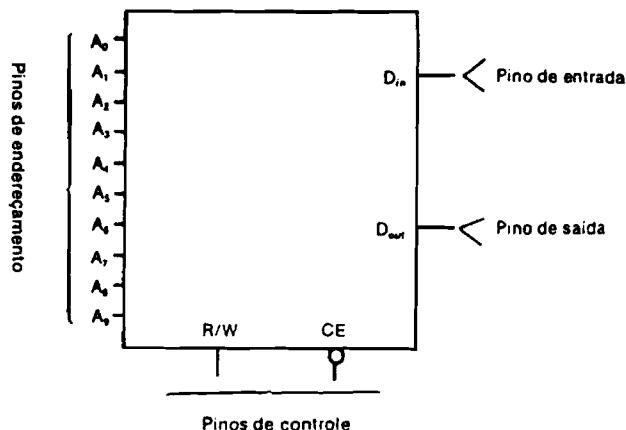


Fig. 0.6.b Símbolo lógico de uma memória de 1.024 palavras de 1 bit cada. (1 K x 1).

Na operação de armazenamento, o dado é colocado nos pinos de entrada (na forma de níveis lógicos "0" e "1") e pelos pinos de endereçamento é informada qual a posição interna da memória em que o dado será gravado. Em seguida, pelos pinos de comando e controle, é indicado que a operação a ser efetuada é de escrita (armazenamento) e que esta operação deve ser realizada.

O processo de leitura de dados é bastante similar ao descrito para armazenamento. O endereço da posição de memória a ser lida é informado pelos pinos de endereçamento e um sinal indicando a condição de leitura de dados é aplicado aos pinos de comando e leitura. Feito isto, o dado estará presente nos pinos de saída da memória. A leitura não é destrutiva, isto é, a posição de memória selecionada continua armazenando o dado que está sendo mostrado nos pinos de saída.

Conforme suas características, as memórias semicondutoras podem ser classificadas em vários tipos usualmente identificados por meio de siglas, a saber: RAM, ROM, PROM, EPROM e EAROM.

RAM (Random Access Memory). A memória RAM é um tipo de memória na qual pode-se "ler" ou "escrever" em qualquer de suas posições (*Read/Write Memory*). O acesso a uma determinada posição de memória é feito aleatoriamente, isto é, pode-se ter acesso direto a qualquer uma das posições.

Este tipo de memória é "volátil", ou seja, os dados armazenados se perdem caso o sistema de alimentação seja interrompido.

As memórias semicondutoras RAM são divididas nas categorias estática e dinâmica. A RAM estática é formada por grupos de transistores dispostos dois a dois na configuração de *flip-flop*, enquanto a RAM dinâmica armazena os dados em capacitores que perdem sua carga com o tempo, sendo necessário reescrever os dados periodicamente (*refresh*).

As vantagens das memórias dinâmicas sobre as estáticas são que as primeiras consomem menos potência, são mais simples de ser construídas e, consequentemente, têm um custo menor. Já as memórias estáticas são mais rápidas e mais simples de usar do que as dinâmicas.

ROM (Read Only Memory). A memória semicondutora do tipo ROM caracteriza-se basicamente por ser apenas de leitura, ou seja, não é possível escrever dados durante a operação normal do dispositivo. O conteúdo de uma ROM é gravado durante o seu processo de fabricação, de acordo com a vontade do usuá-

rio. Uma vez que o usuário decidiu quais os dados que devem ser armazenados na ROM, ele os transmite ao fabricante da memória. Só a gravação da ROM, o seu conteúdo não mais poderá ser alterado.

Este tipo de memória é não-volátil, sendo utilizado para armazenar, de forma permanente, tabelas e programas de finalidades especiais em computadores.

PROM (Programmable Read Only Memory). A memória semicondutora do tipo PROM é basicamente uma ROM na qual a gravação de seu conteúdo é feita pelo usuário, e não mais pelo fabricante. A PROM possui circuitos internos que permitem a sua gravação, por meio de processos especiais, fora da fábrica em que foi produzida.

Uma vez feita a gravação da PROM, esta não poderá ser alterada.

EPROM (Erasable Programmable Read Only Memory). Este tipo de memória é uma PROM que permite a reprogramação através de processos especiais, ou seja, o usuário pode apagar o seu conteúdo e gravar novamente conforme a sua vontade.

O conteúdo da EPROM pode ser apagado fazendo-se incidir luz ultravioleta sobre o *chip* através de uma janela transparente existente na parte superior do dispositivo. O processo de apagar não é seletivo, resultando no apagamento de todas as posições da memória.

Apesar de seu custo mais elevado a EPROM é bastante útil nos casos de programas ou dados que estão sempre sendo modificados pelos fabricantes dos equipamentos nos quais ela é utilizada. Por exemplo, na atualização do *software*, contido em EPROM, de um certo computador, o custo seria mínimo se comparado com a troca do componente, caso as memórias dos tipos ROM ou PROM fossem utilizadas.

EAROM (Electrically Alterable Read Only Memory). A memória do tipo EAROM é similar à EPROM, porém, ao invés do uso de luz ultravioleta, o seu conteúdo pode ser apagado aplicando-se um certo valor de voltagem aos pinos de programação.

Uma vantagem no uso da EAROM é que seu apagamento pode ser feito no próprio circuito em que se encontra, dispensando lâmpadas especiais e simplificando o processo. Outra grande vantagem é que a operação de apagamento é seletiva, isto é, cada posição da memória pode ser apagada sem que se altere o conteúdo das demais.

0.2.3. Unidade Central de Processamento (UCP)

A Unidade Central de Processamento de um computador, devido à sua complexidade, é normalmente dividida, para fins de estudo e projeto, em duas partes:

- a Unidade de Controle, onde os códigos, que representam as operações a serem realizadas, são identificados e através da qual os dados são obtidos da memória;
- a Unidade Lógica e Aritmética, onde as operações são efetivamente executadas.

A Unidade de Controle. Códigos especiais (instruções) são usados para indicar ao computador as operações que ele deve realizar e os dados a que elas se referem. Um conjunto de circuitos lógicos se encarrega de interpretar o significado desses códigos, trazer da memória os dados referidos e executar as ações requisitadas.

Umas tantas convenções são adotadas quando se projeta um computador, entre as quais está a maneira pela qual uma operação que se deseja realizar deve ser informada à Unidade de Controle.

Assim, sempre que esta unidade entrar em contato com uma informação que obedece às convenções especificadas no seu projeto, ela está apta a interpretá-la convenientemente e, com isso, executar a operação intencionada.

Um conjunto de formas-padrão é adotado para informar à Unidade de Controle qual a operação a ser feita e onde está o dado na memória a que esta operação se refere.

A manifestação explícita dessas formas chama-se INSTRUÇÃO.

Em um computador, onde os dados são binários e os circuitos são lógicos, instrução é uma configuração de dígitos binários.

Em um conjunto de bits que representa uma instrução, podem-se distinguir, normalmente, dois elementos:

- Código de Operação: subconjunto de dígitos da instrução que identifica uma entre as várias operações que o computador é capaz de executar;
- Campo de Operandos: subconjunto de dígitos da instrução que especifica onde estão os dados necessários à execução da operação.

A Unidade Lógica e Aritmética (ULA). Nesta unidade são executadas as operações aritméticas de adição, subtração, multiplicação e divisão e operações lógicas, tais como complemento de um, conjunção, disjunção, ou exclusivo, e outras. Atualmente, existem disponíveis comercialmente unidades lógicas e aritméticas integradas em uma pastilha.

As operações na ULA são efetuadas em velocidades bastante elevadas, demorando cerca de dezenas de nanosegundos (10^{-9} s) para somar duas palavras de 16 bits.

Os Microprocessadores. Graças ao desenvolvimento da microeletrônica é possível construir toda uma Unidade Central de Processamento em uma única pastilha de silício. Esta pastilha, ou *chip*, denomina-se microprocessador, sendo conhecido pelo nome de seu fabricante seguido de determinado número. São comercializados, por exemplo, INTEL 8080, INTEL 8088, Z80, MOTOROLA 6800, INTEL 80286 etc.

Os microprocessadores são classificados pelo tamanho da palavra — ou comprimento, em bits, da unidade de informação — que são capazes de processar de uma só vez. Os primeiros microprocessadores foram os de 8 bits, seguidos pelos de 16 bits e, mais recentemente, pelos de 32 bits.

O microprocessador é, portanto, a Unidade Central de Processamento de um microcomputador.

0.2.4. Periféricos

As unidades de entrada, que servem para introduzir programas ou dados no computador, e as unidades de saída, que servem para receber programas ou dados do computador, são denominadas periféricos.

Teclado. O teclado é uma unidade de entrada semelhante a uma máquina de escrever, com algumas teclas adicionais de controle do computador. Às vezes, apresenta-se com um teclado numérico adicional, com disposição semelhante à das calculadoras.

Geralmente, as informações são introduzidas através do teclado por meio de linhas: digita-se uma linha de texto, corrige-se o que for necessário e, quando a linha estiver perfeita, ela é introduzida na Unidade Central de Processamento, apertando-se uma certa tecla de controle (ENTER, às vezes RETURN ou XMIT nos terminais).

Normalmente, o que está sendo digitado vai aparecendo no vídeo, que é uma unidade de saída. A posição que irá receber o próximo caractere a ser digitado é indicada no vídeo com um cursor. Em computadores mais antigos, o que é digitado vai sendo escrito em folha de papel por uma impressora de caracteres semelhante ao dispositivo de impacto das máquinas de escrever.

O teclado oferece a vantagem da introdução direta das informações que podem estar sendo criadas naquele momento pelo operador. Mas tem o inconveniente de prender o computador enquanto é utilizado. O teclado é a principal unidade de entrada apenas nos microcomputadores. Em computadores maiores prefere-se preparar os dados em máquinas fora de linha, isto é, não ligadas ao computador, para depois levá-los ao computador através de outro periférico, usando-se o teclado apenas para poucas informações que devem ser introduzidas durante o seu funcionamento.

Vídeo. O vídeo é uma unidade de saída dos computadores semelhante ao vídeo de um aparelho de televisão (às vezes é mesmo um aparelho de televisão que, desligado do computador, pode receber novelas, comerciais, filmes antigos e tudo mais que a televisão costuma oferecer). O vídeo, geralmente, repro-

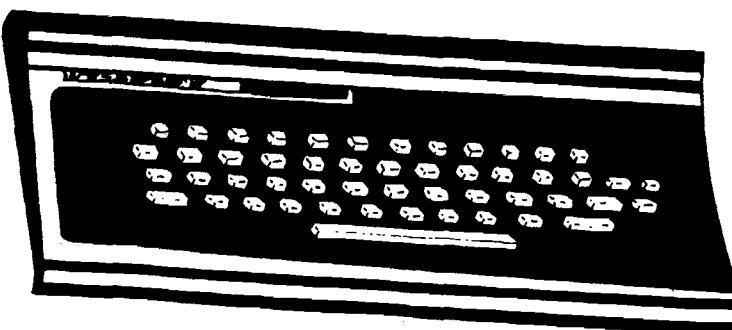


Fig. 0.7 Teclado.

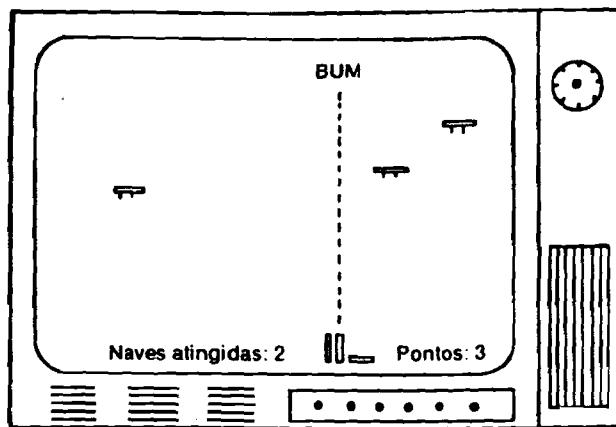


Fig. 0.8 Vídeo.

duz o que está sendo digitado no teclado, mas pode mostrar também resultados de cálculos ou mensagens, sob controle dos programas. Geralmente, os vídeos podem funcionar sob dois modos, dependendo do controle do programa: no modo caractere (que pode mostrar cerca de 16 a 24 linhas de 48 a 80 caracteres) ou no modo gráfico (quando usa uma resolução mais alta, isto é, mais pontos luminosos para formar desenhos).

O vídeo oferece a grande vantagem de permitir uma interação instantânea entre o operador e o computador, indispensável nos jogos eletrônicos, por exemplo. Mas tem a desvantagem de não deixar registrado os resultados obtidos, além de também prender o computador durante a sua utilização. Como o teclado, o vídeo é a principal unidade de saída nos microcomputadores, sendo utilizado nos computadores maiores apenas para mensagens curtas que o computador envia para o operador através do programa que o esteja controlando.

Impressora. É uma unidade de saída em que as informações provenientes do computador são registradas em forma impressa numa folha de papel.

A velocidade das impressoras mais comuns varia de 100 a 400 cps (caracteres por segundo), quando ligadas a microcomputadores ou a terminais, e de 600 a 1.000 lps (linhas por segundo), quando ligadas a computadores maiores. As impressoras geralmente permitem imprimir, no máximo, 80 ou 132 cpi (caracteres por linha) com a densidade de 10 cpi (caracteres por polegada = inch) na horizontal e 6 ou 8 lpi (linhas por polegada) na vertical. Algumas impressoras permitem variar, por programa, a densidade horizontal, isto é, imprimir caracteres mais longos ou mais estreitos.

A impressão pode ser feita com tipos sólidos, como nas máquinas de escrever, ou pela seleção de pontos dispostos numa matriz, geralmente de 9×7 pontos, nas impressoras matriciais.

Quanto ao modo de produzir a impressão, as impressoras se dividem entre as que usam o impacto mecânico contra uma fita de carbono e as que não são de impacto. As impressoras de impacto podem usar uma cabeça de nove agulhas (movimentadas por programa através de pequenos eletroímãs), uma corrente de caracteres ou uma série de rodas de caracteres (que recebem uma martelada no momento em que passam pela posição em que devem ser impressas), um cilindro ou esfera de caracteres ou uma margarida.

Impressoras que não usam impacto podem ser térmicas (que marcam papel especial por meio de aquecimento), eletrostáticas (que usam o princípio das copiadoras), injeção de tinta (que esguicham finos jatos de tinta), a raios laser etc.

Unidades de discos magnéticos e disquetes. São unidades de entrada e de saída do computador. Informações provenientes do computador podem ser registradas de forma magnetizada em discos ou disquetes e, posteriormente, podem ser introduzidas de novo no computador, através de unidades leitoras/gravadoras de discos ou disquetes.

O meio utilizado é um disco com a superfície revestida por uma camada de óxido de ferro, que pode ser localmente magnetizada num sentido ou no outro por um cabeçote, através de um pequeno eletroímã.

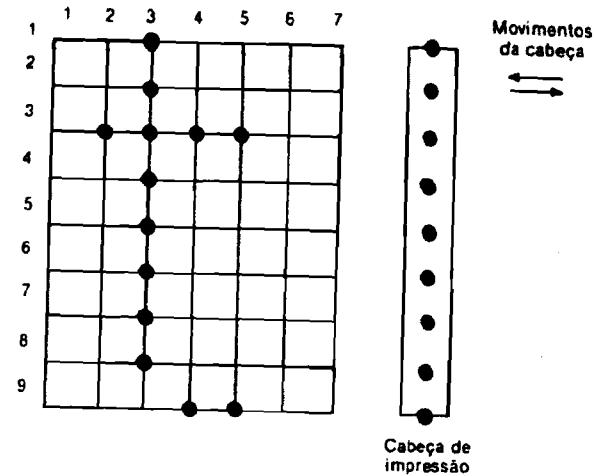


Fig. 0.9 Impressora matricial.

O mesmo cabeçote permite "ler" magnetizações feitas anteriormente, sem apagá-las, pela produção de uma pequena corrente induzida no seu eletroímã ao passar sobre regiões em que houve mudança no sentido da magnetização. O disco tem um movimento de rotação em torno de seu eixo, por exemplo, de 300 rpm (rotações por minuto) enquanto o cabeçote pode ser movimentado radialmente em cada posição. O cabeçote, ele escreve ou lê informações ao longo de uma circunferência, que tem o nome de trilha. Para conseguir maior economia no mecanismo de acionamento dos discos e dos cabeçotes de leitura e escrita,

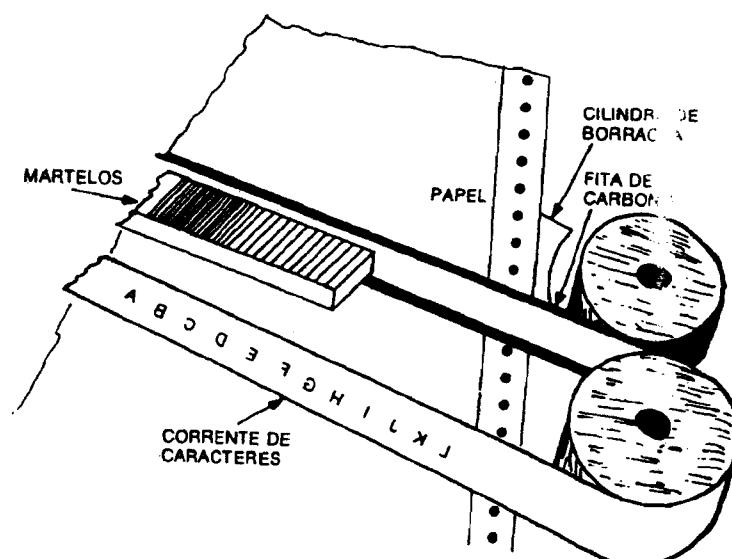


Fig. 0.10 Corrente de caracteres.

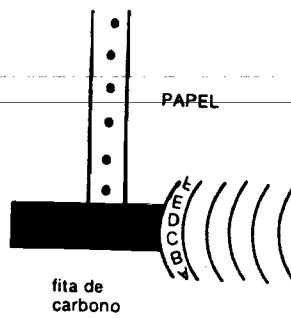


Fig. 0.11 Rodas de caracteres.

constroem-se unidades que usam as duas faces do disco ou que utilizam uma pilha de discos solidários no mesmo eixo, acessíveis por um pente de cabeçotes movimentados por um único mecanismo (*disk pack*). A capacidade de um disco magnético, hoje, atinge cerca de dezenas de megabytes.

Nos discos magnéticos, o cabeçote de leitura e escrita "voa" sobre as superfícies dos discos a uma distância da ordem de 1 micron (1 milésimo de milímetro), sem tocá-las.

Os disquetes ou discos flexíveis têm tamanho menor (geralmente 5,25 ou 8 polegadas), capacidade de 2 ou 5 centenas de Kbytes por face, dependendo da densidade de gravação ser simples ou dupla. Os disquetes são embalados em um envelope revestido internamente de um tecido especial que protege e limpa o disquete. Através de janelas feitas no envelope, a unidade de disquete o prende e o põe em movimento, com o cabeçote tocando a sua superfície.

Os discos magnéticos são usados nos grandes computadores e os disquetes, às vezes, são usados como entrada de dados, sendo a sua gravação feita fora de linha em máquinas gravadoras de disquetes ou em microcomputadores.

Unidades de fitas magnéticas e cassetes. As unidades de fitas magnéticas permitem escrever e ler informações registradas magneticamente em fitas, por um processo semelhante ao usado nos discos magnéticos. Cada uma destas unidades possui dois carretéis. Por comandos de um programa, a fita se move de um carretel para o outro, passando por um cabeçote de leitura/gravação.

As fitas magnéticas têm um custo por byte armazenado muito inferior aos discos magnéticos, aliado a uma grande capacidade de armazenamento. Por exemplo, uma fita de 1.200 pés de comprimento pode armazenar mais de 20 megabytes. A densidade de gravação mais usual é de 1.600 bpi (*bytes per inch*).

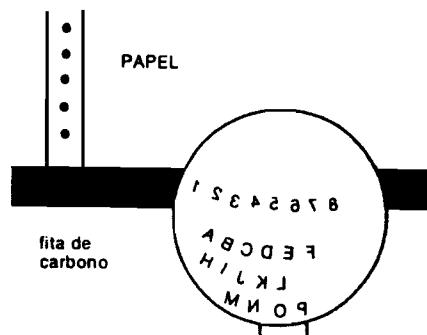


Fig. 0.12 Esfera de caracteres.

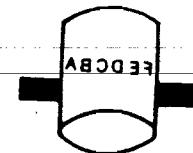


Fig. 0.13 Cilindro.

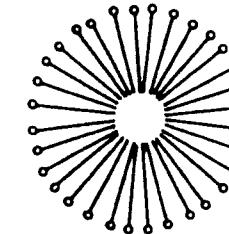


Fig. 0.14 Margarida.

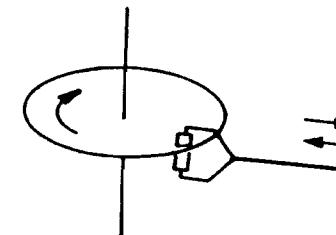


Fig. 0.15 Disco magnético de dupla face.

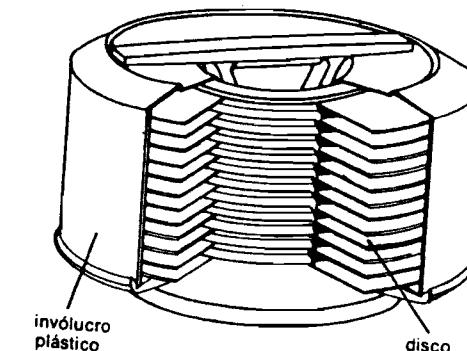


Fig. 0.16 Pilha de discos.

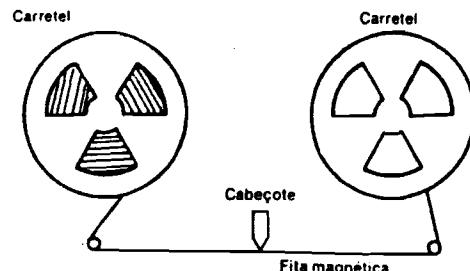


Fig. 0.17 Unidade de fita magnética

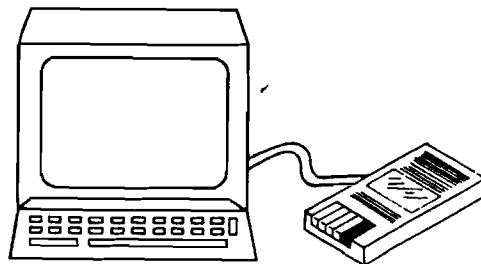


Fig. 0.18 Gravador de fita cassette ligado a um microcomputador.

A grande desvantagem das fitas em relação aos discos magnéticos é a de só permitirem acesso sequencial às informações: uma informação escrita na fita só se torna acessível após a leitura de todas as informações que a antecedem, enquanto que num disco magnético, saltando-se para a trilha em que está a informação desejada, obtém-se um acesso mais direto.

As fitas magnéticas são usadas em computadores de maior porte. O seu correspondente nos micro-computadores são as fitas cassetes, as mesmas utilizadas nos aparelhos de som. Muitos microcomputadores permitem que se leiam ou escrevam em fitas cassetes através de um gravador comum de som, oferecendo assim uma solução mais econômica, embora mais incômoda e menos eficiente que a dos disquetes.

Leitoras/perfuradoras de cartão. O cartão perfurado já foi o principal meio de entrada de dados no computador e, às vezes, também, de saída.

O cartão perfurado tem cerca de 18,7 cm por 8,2 cm e permite que se faça perfuração em 12 alturas diferentes de 80 colunas. Diferentes combinações destas 12 alturas de uma coluna permitem representar letras, dígitos e outros caracteres especiais. Os cartões são preparados numa máquina perfuradora de cartões, totalmente separada do computador. Esta máquina possui um teclado semelhante ao de uma máquina de escrever e, à medida que vai perfurando os cartões, interpreta estas perfurações na margem superior do cartão, com auxílio de uma impressora de agulhas.

No computador, a leitora de cartões lê uma fila de cartões, fazendo-os passar entre uma fonte de luz e uma série de células fotoelétricas, cuja sensibilização é traduzida em impulsos elétricos recebidos e interpretados pelo computador.

Às vezes, o próprio computador controla a perfuração dos resultados em cartões (com a finalidade de serem posteriormente realimentados no computador). Há periféricos que fazem o cartão passar sucessivamente por uma estação de leitura e uma estação de perfuração, permitindo que sejam feitos cálculos com os dados lidos num cartão e os resultados sejam perfurados no mesmo cartão.

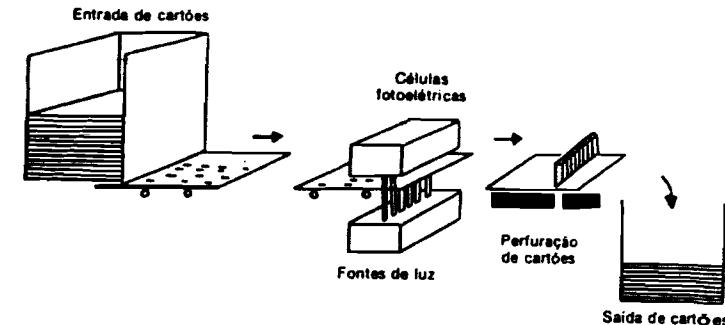


Fig. 0.19 Esquema de uma leitora/perfuradora de cartões.

Existem periféricos que permitem a leitura e perfuração de cartões menores, com 96 posições de perfurações circulares e outros que lêem e perfuram fita de papel.

Uma grande facilidade oferecida pelos cartões perfurados é o seu manuseio direto, permitindo a inserção, supressão ou substituição manual dos cartões de uma massa. A alteração de informações gravadas em discos ou fitas magnéticas, por outro lado, só pode ser feita através de um computador.

O inconveniente do cartão perfurado, porém, está no seu custo elevado, na maior lentidão de leitura ou perfuração por parte dos periféricos e no fato de ele não poder ser reaproveitado para novas informações.

Embora com menor freqüência, diversos outros periféricos podem eventualmente ser encontrados nos computadores.

Traçador de gráficos. É um aparelho em que o movimento de uma caneta esferográfica em duas direções perpendiculares, permite fazer desenhos controlados por um programa no computador.

Leitora de marcas ópticas. É uma unidade de entrada do computador que lhe permite, através de células fotoelétricas, ler marcas feitas em posições prefixadas de uma folha de papel. É muito usada na avaliação de provas de múltipla escolha de exames vestibulares.

Leitora de caracteres ópticos. É um periférico, um pouco mais complexo que a leitora de marcas, que permite ler caracteres de tamanho e formato predeterminados, geralmente impressos pelo próprio computador. É usada para fazer o controle do pagamento de contas de água, luz e telefone, lendo o cahnoto da conta, devolvida pelo banco, após o seu pagamento. Algumas leitoras de caracteres ópticos lêem caracteres comuns, porém outras só lêem caracteres com um desenho especial.

Leitora de caracteres magnéticos. Este periférico permite ler caracteres especiais impressos com tinta contendo partículas magnetizáveis. Geralmente usada para o processamento de cheques bancários.

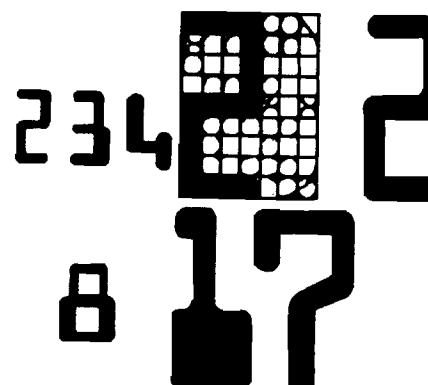


Fig. 0.20 Caracteres especiais para leitoras ópticas.

Emissor de som. É um dispositivo que, sob controle do computador, permite emitir um som com duração e frequência desejáveis. É geralmente usado para emitir sinais programados de advertência, mas pode inclusive produzir música.

Sintetizador de voz. É um aparelho capaz de emitir um certo número de fonemas que, codificados e combinados por um programa de computador, permitem-lhe produzir respostas audíveis.

Sensores e controladores. São dispositivos que transmitem ao computador sinais provenientes de aparelhos de medição (temperatura, pressão, ruído, peso, volume, tráfego etc.) ou que recebem do computador sinais destinados a controlar aparelhos (ligar, desligar, aumentar ou diminuir a intensidade de seu funcionamento etc.). São usados em computadores destinados a controle de processos.

0.3. ALGORITMO

Nesta seção serão abordados os conceitos associados à utilização do computador.

O conjunto destes conceitos, técnicas, metodologias e ferramentas, que viabilizam a programação dos computadores, denomina-se *software*.

0.3.1. Conceituação

Para estabelecer o conceito de algoritmo, que é da maior importância em Ciência da Computação, será, antes fixado o conceito de ação.

Ação é um acontecimento que, a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e bem definido.

Para se ter alguns exemplos, foi pedido a três co-autores deste livro que escolhessem um valor numérico L e escrevessem os termos da seqüência de Fibonacci inferiores a L. A seqüência de Fibonacci se define como tendo os dois primeiros termos iguais a 1 e cada termo seguinte é igual à soma dos dois termos imediatamente anteriores.

A Miriam escolheu L = 50 e escreveu.

1 1 2 3 5 8 13 21 34

O Eduardo, que gosta de brincar com superstições, escolheu L = 13 e escreveu:

1 2 3 5 8

O Harry, que prefere ser diferente dos outros, escolheu L = 1 e nada escreveu.

Apenas parte do que a Miriam, o Eduardo e o Harry fizeram pode ser considerado como ação. A escolha dos valores L = 50, 13 e 1 é totalmente imprevisível e portanto não é uma ação. Porém, ao escrever os termos da seqüência de Fibonacci inferiores a L, cada uma destas pessoas realizou uma ação: partiu de um **estado inicial** (o valor escolhido para L) e, após um período de tempo limitado (alguns segundos), produziu um **estado final** (o que foi escrito ou não foi), previsível e bem definido.

Estas três ações são distintas, foram executadas por pessoas diferentes, ocorreram em instantes diferentes, partiram de valores iniciais diferentes e produziram resultados finais diferentes. Mas, apesar disso, mesmo que o Harry queira ser diferente dos outros, pode-se reconhecer nas três ações distintas um mesmo **padrão de comportamento**, a subordinação a uma mesma **norma de execução**. É como se as três ações tivessem sido executadas em obediência ao comando:

Escreva os termos de Fibonacci inferiores a L.

Adota-se a seguinte definição de algoritmo:

Algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de ações.

Geralmente, um algoritmo se destina a resolver um problema: fixa um padrão de comportamento a ser seguido, uma norma de execução a ser trilhada, para se atingir, como resultado final, a solução de um problema.

Pode-se dizer que, após a escolha do valor de L, tanto a Miriam, o Eduardo como o Harry executaram ações indicadas pelo algoritmo seguinte:

Algoritmo

Escreva os termos de Fibonacci inferiores a L.
fim algoritmo.

Neste livro, os algoritmos sempre serão iniciados com a palavra Algoritmo e terminados com a expressão fim algoritmo.

0.3.2. Exercícios de fixação

○ 0.3.2.1. Um algoritmo não pode conter um comando como "Escreva todos os termos da seqüência de Fibonacci". Por quê?

○ 0.3.2.2. Escrever um algoritmo que, partindo de diferentes estados iniciais, produza os valores:

- a) 2 4 6 8 10 12 14
b) 1 3 5 7 9 11 13

○ 0.3.2.3. Escrever um segundo algoritmo que produza os mesmos valores que o exercício 0.3.2.2.

0.3.3. Refinamentos sucessivos

Na vida quotidiana, os algoritmos são encontrados freqüentemente: instruções para se utilizar um aparelho eletrodoméstico, uma receita para preparo de algum prato, o guia de preenchimento da 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.

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

Num algoritmo, um comando que não for de entendimento do destinatário terá de ser desdobrado em novos comandos, que constituirão um **refinamento** do comando inicial.

Se um algoritmo é formado não apenas por um comando, mas por vários, isto significa que na sua execução não se consideram apenas o **estado inicial** e o **final** de uma ação dele resultante, mas que se consideram também **estados intermediários** que delimitam as ações decorrentes de cada comando.

Por exemplo, o algoritmo para escrever os termos de Fibonacci inferiores a L poderia ser desdobrado em:

Ref. Escreva os termos de Fibonacci inferiores a L
Receba o valor L
Processe os 2 primeiros termos
Processe os termos restantes
fim ref.

Neste livro, um refinamento será sempre iniciado com a palavra Ref., seguida do comando a ser refinado e terminará com a expressão fim ref.

Um algoritmo e os seus refinamentos são formados por **comandos**, que determinam as ações a serem executadas pelo seu destinatário e por **estruturas de controle** que determinam a ordem em que os comandos deve ser executados, se devem ser executados ou não e quando devem ser repetidos.

No refinamento acima vigora a mais simples das estruturas de controle: a **estrutura seqüencial**, segundo a qual os comandos devem ser executados um após o outro, na mesma ordem em que aparecem escritos.

Se um comando de um refinamento for um tanto vago, ele poderá, por sua vez, ser desdobrado em novos comandos, produzindo-se o refinamento de um refinamento, e assim sucessivamente. Portanto, o comando "Processe os 2 primeiros termos" poderia ser desdobrado em:

Ref. Processe os 2 primeiros termos

```
Atribua o valor 1 ao primeiro termo  
se ele for menor que L  
| então escreva-o  
fim se  
Atribua o valor 1 ao segundo termo  
se ele for menor que L  
| então escreva-o  
fim se  
fim ref.
```

Neste refinamento, aparece uma segunda estrutura de controle: a **estrutura condicional**

```
se condição  
| então comandos  
fim se
```

O comando "escreva-o" só será executado se a condição "ele for menor que L" for verdadeira. O comando "escreva-o" não será executado se a condição for falsa, isto é, se ele for maior ou igual a L.

Vê-se, portanto, que um algoritmo, através de estruturas condicionais, pode provocar ou não a realização de uma ação.

Uma terceira estrutura de controle, a **estrutura de repetição**, será necessária ao se desdobrar o comando "Processe os termos restantes", através do novo refinamento:

Ref. Processe os termos restantes

```
repita  
| Calcule novo termo somando os 2 anteriores  
| se novo termo for maior ou igual a L  
| | então interrompa  
| fim se  
| Escreva novo termo  
fim repita  
fim ref.
```

Na estrutura de repetição, os comandos e as estruturas de controle abrangidos devem ser executados repetidamente até que se verifique uma condição (no caso, o novo termo ser maior ou igual a L) para que se interrompa a repetição.

No caso do Eduardo, que fixou $L = 13$, a execução do refinamento acima produziria as seguintes ações:

Calcula novo termo, como sendo	$1 + 1 = 2$
Como $2 < 13$, escreve o valor	2
Calcula novo termo, como sendo	$1 + 2 = 3$
Como $3 < 13$, escreve o valor	3
Calcula novo termo, como sendo	$2 + 3 = 5$
Como $5 < 13$, escreve o valor	5
Calcula novo termo, como sendo	$3 + 5 = 8$
Como $8 < 13$, escreve o valor	8
Calcula novo termo, como sendo	$8 + 5 = 13$
Como 13 não é menor que 13, interrompe	

A forma de descrição, usada nos algoritmos, com auxílio das estruturas de controle, além de mais elegante, é mais sintética do que a simples descrição das ações realizadas e é mais geral, pois abrange também as ações diferentes, realizadas pela Miriam e pelo Harry.

Após esses refinamentos sucessivos, o algoritmo pode ser considerado completo, a menos que o destinatário não saiba fazer a adição de dois termos ou não seja capaz de entender diretamente algum comando.

O algoritmo estando completo, pode-se reescrivê-lo, inserindo os refinamentos nas posições dos comandos que foram refinados e usando estes como comentários. Assim sendo, obtém-se:

Algoritmo (Escrita dos termos de Fibonacci inferiores a L)

```
Receba o valor L  
{Processamento dos 2 primeiros termos}  
Atribua o valor 1 ao primeiro termo  
se ele for menor que L  
| então escreva-o  
fim se  
Atribua o valor 1 ao segundo termo  
se ele for menor que L  
| então escreva-o  
fim se  
{Processamento dos termos restantes}  
repita  
| Calcule novo termo somando os 2 anteriores  
| se novo termo for maior ou igual a L  
| | então interrompa  
| fim se  
| Escreva novo termo  
fim repita  
fim algoritmo
```

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.

A 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. Então este entendimento poderá se fazer também por refinamentos sucessivos, repetindo-se o processo usado na sua criação.

0.3.4. Exercícios de fixação

0.3.4.1. Se você já tiver estudado a determinação de máximos e mínimos de uma função, por derivações sucessivas, então escreva um algoritmo correspondente.

Algoritmo**repita**

Invente um problema
Escreva um algoritmo para sua resolução
se estiver cansado
então interrompa

fim se**fim repita**

Mostre os algoritmos feitos a um colega

fim algoritmo.

0.3.4.3. Se você executar os algoritmos escritos nos exercícios 0.3.4.1 e 0.3.4.2, qual é o menor número de algoritmos que poderão ser escritos em cada exercício?

Para terminar a Seção 0.3, será visto a seguir um curioso padrão de comportamento encontrado na natureza.

Ao se tomarem medidas do corpo humano, encontram-se as seguintes relações:

- a razão entre a altura total H e a altura do coração h é aproximadamente 1,6;
- o comprimento do braço b dividido pelo comprimento do antebraço a é aproximadamente 1,6;
- a razão entre o comprimento do antebraço a e o da mão m também é aproximadamente 1,6;
- ao se envolver a orelha num retângulo de altura h e largura ℓ , $h/\ell \approx 1,6$.

Nos animais encontram-se as mesmas relações:

- e) na estrela do mar, a razão entre o lado do pentágono estrelado t e o do pentágono regular circunscrito p é aproximadamente 1,6.

Também nas plantas se encontram estas relações:

- f) na flor do girassol ou no miolo da margarida, o número de espirais num sentido dividido pelo número de espirais no sentido contrário se aproxima de 1,6;
- g) a razão entre os ângulos α e β formados pelos ramos de uma planta é aproximadamente 1,6.

Na própria Matemática, esta relação é freqüente:

- h) a relação $\frac{a}{b} = \frac{b}{a+b}$ é satisfeita para $a \approx 1,6b$, ou, mais precisamente, $a \approx 1.618b$;

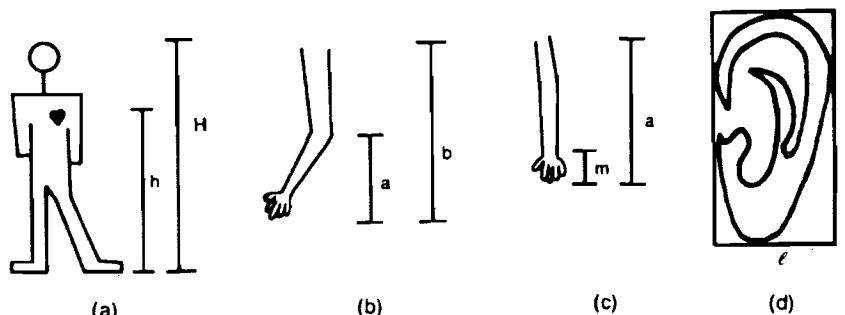
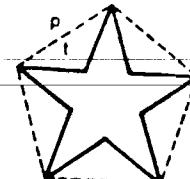
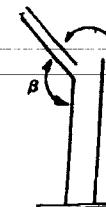


Fig. 0.21 Razão entre medidas do corpo humano.



(e)



(g)

Fig. 0.22 Relação existente na estrela do mar e nos ramos de uma planta.

a proporção acima é conhecida como média e extrema razão ou proporção áurea e 1,618 é uma aproximação do número irracional chamado **número áureo** e denotado por Φ .

$$i) \Phi = \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \dots}}}}$$

$$j) \Phi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}$$

$$k) \Phi = \frac{\sqrt{5} + 1}{2}$$

$$l) \Phi \approx \frac{89}{55}$$

$$m) \Phi \approx \frac{144}{89}$$

- n) a razão entre dois termos consecutivos da seqüência de Fibonacci tende muito rapidamente para o limite que é $\Phi = 1,618 \dots$

Fibonacci é sobrenome de Leonardo de Pisa, que foi o primeiro a observar esta seqüência, em 1202, quando determinava o crescimento do número de coelhos de uma criação (...).

0.4. ALGORITMOS ESTRUTURADOS

Os computadores são máquinas destinadas a resolver problemas com grande rapidez: uma operação aritmética pode ser efetuada num tempo da ordem de 1 microsegundo, isto é, 1 milionésimo de segundo; computadores mais potentes estão atingindo o tempo de 1 nanosegundo, isto é, 1 milésimo de microsegundo. O aproveitamento desta grande rapidez exige que as operações sejam efetuadas automaticamente, sem a interferência humana. Qualquer decisão humana demanda dezenas de segundos (às vezes, horas ou dias). Pouco adiantaria a máquina fazer uma operação em 1/1.000.000 de segundo, se em seguida ela tivesse de esperar dezenas de segundos (ou horas, ou dias) para efetuar a próxima operação. Em casos como este seria preferível usar uma calculadora, em vez de um computador.

Por outro lado, máquinas dotadas desta rapidez e deste automatismo são muito caras. Investimentos nestas máquinas só se justificam se elas puderem ser usadas para resolver problemas da mais diversa natureza: se elas forem flexíveis.

A flexibilidade exige que, para cada problema a ser levado ao computador, sejam planejadas as operações correspondentes. O automatismo, por outro lado, exige que o planejamento destas operações seja feito previamente, antes de se utilizar o computador.

Então, a utilização de um computador para resolver problemas exige, antes de mais nada, que se desenvolva um algoritmo, isto é, que se faça a descrição de um conjunto de comandos que, obedecidos, provocarão uma sucessão finita de ações que resultarão na resolução do problema proposto. Este algoritmo

tem de ser transmitido ao computador e armazenado na sua memória, para, em seguida, ser posto em execução e conduzir o computador para a solução desejada. O algoritmo deve, portanto, prever antecipadamente todas as situações que possam ocorrer quando for posto em execução.

Se um problema for simples e de pequeno vulto, não há maiores dificuldades em se desenvolver um algoritmo adequado para resolvê-lo no computador, nem há necessidade de aplicação de técnicas especiais para desenvolvê-lo. Na década de 50, quando os computadores começaram a ser fabricados em escala industrial, eles ainda eram máquinas bastante lentas e com pouca capacidade de memória. O porte dos problemas que podiam ser processados nestes computadores era muito limitado. Mas, com o desenvolvimento da tecnologia na área eletrônica, a rapidez, a memória e as potencialidades dos computadores cresceram de maneira dramática. Já se tem feito a comparação de que se a indústria automobilística tivesse evoluído de forma equivalente aos computadores, qualquer automóvel nacional custaria hoje menos de um dólar, faria um milhão de quilômetros com um litro de gasolina e poderia desenvolver a velocidade de cem milhões de quilômetros por hora.

Com esta evolução, os problemas levados aos computadores são cada vez de maior porte e maior complexidade. Os algoritmos para resolvê-los ainda devem ser desenvolvidos pelo ser humano, mas podem ultrapassar os limites de sua compreensão. Por esta razão, nas últimas décadas surgiram técnicas que permitem sistematizar e ajudar o desenvolvimento de algoritmos para a resolução de grandes e complexos problemas nos computadores: são as técnicas de **desenvolvimento estruturado** de algoritmos.

Os objetivos destas técnicas são:

- facilitar o desenvolvimento dos algoritmos;
- facilitar o seu entendimento pelos humanos;
- antecipar a comprovação da sua correção;
- facilitar a sua manutenção e a sua modificação;
- permitir que o seu desenvolvimento possa ser empreendido simultaneamente por uma equipe de pessoas.

Para atingir estes objetivos, o desenvolvimento estruturado preconiza que:

- a) os algoritmos sejam desenvolvidos por **refinamentos sucessivos** partindo de uma descrição geral e, gradativa e sucessivamente, atacando as minúcias e particularidades. Este desenvolvimento também se denomina "construção hierárquica de algoritmos" e "desenvolvimento de cima para baixo" (em inglês, *top-down*);
- b) os sucessivos refinamentos são módulos, que delimitam poucas funções e são o mais independente possível, isto é, conservam poucos vínculos com outros módulos;
- c) nos módulos deve ser usado um número **limitado** de diferentes comandos e de diferentes estruturas de controle.

A técnica do desenvolvimento estruturado de algoritmos, por si só, não atinge automaticamente os objetivos visados. Ela apenas preconiza uma maneira sistemática que ajuda a atingir estes objetivos. É necessário ainda, por parte de quem a emprega, esforço e disciplina na busca incessante da simplicidade e da clareza.

O refinamento sucessivo dos algoritmos permite uma abordagem mais segura e objetiva do problema. A integração dos módulos é mais perfeita, porque cada módulo é atacado quando já se estudou claramente o ambiente em que ele deve atuar. A alteração dos módulos já desenvolvidos é teoricamente desnecessária. Na prática, consegue-se diminuir muito estas alterações pela atenção e disciplina. Os módulos de mais alto nível podem ser testados antes de se desenvolverem os de níveis mais baixos, permitindo o desenvolvimento mais seguro dos algoritmos.

A divisão em módulos funcionais permite contornar a limitação humana para compreender a complexidade. Cada módulo pode ser desenvolvido ou analisado de forma quase independente, dados os poucos vínculos que deve manter com os outros módulos do algoritmo. Cada módulo pode, portanto, ser desenvolvido por uma pessoa, após acertar poucos acordos com o resto da equipe.

O uso ilimitado de diferentes comandos e diferentes estruturas de controle pode restringir bastante o raciocínio, que fica obrigado a se enquadrar em poucas formas. Mas os benefícios em termos de simplicidade e clareza do produto final são imensos.

Clareza e simplicidade de um algoritmo são atributos inestimáveis quando se faz a sua manutenção (melhora, correção e aperfeiçoamento de algum algoritmo) e a sua modificação (para atender a novas aplicações). O desenvolvimento estruturado dos algoritmos, aliado à busca da clareza e da simplicidade, facilita a sua manutenção e a sua modificação.

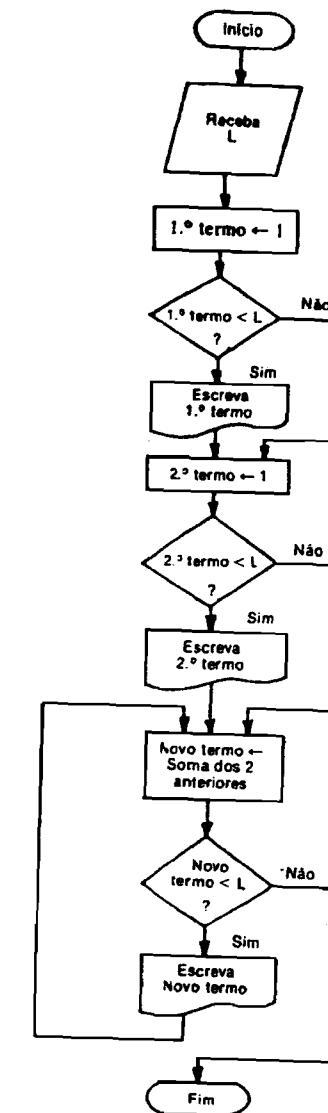


Fig. 0.23 Diagrama de blocos.

Para se obter uma melhor clareza dos algoritmos, costuma-se desenvolvê-los e ilustrá-los com o auxílio de um diagrama de blocos: algumas figuras geométricas e dizeres são usados para representar as diversas operações do computador, sendo ligados por setas, para indicar a seqüência da sua execução. Por exemplo, o algoritmo visto no parágrafo 0.3, para escrever os termos de Fibonacci inferiores a L, poderia ser ilustrado com o diagrama de blocos da figura 0.23. Este diagrama permite visualizar o fluxo de comandos que constituem o algoritmo, de modo que se possa ver com facilidade como se processará a sua execução. Mas ele tem um grande inconveniente: permite uma liberdade ampla demais do raciocínio, o que pode redundar na utilização de outras estruturas de controle, além das preconizadas no parágrafo 0.3,

Escreva termos de Fibonacci inferiores a L

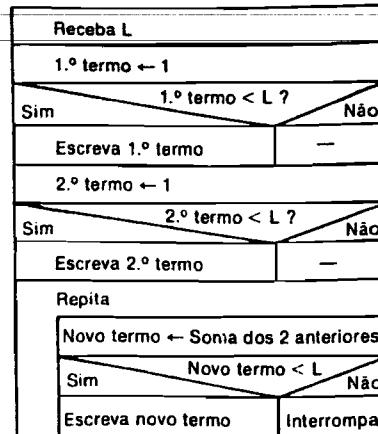


Fig. 0.24 Diagrama de Chapin.

com transferências livres de um comando para qualquer outro, pela simples indicação de uma seta. Isto pode ocasionar algoritmos extremamente complicados e de difícil compreensão, de perigosa manutenção e modificação.

Este inconveniente, por si só, desaconselha a utilização dos diagramas de blocos. Usa-se também, visualizar os algoritmos através do diagrama de Chapin (ou Nassi-Sneider), que só permite representar as estruturas de controle preconizadas, evitando as livres transferências. O mesmo exemplo de algoritmo foi representado na figura 0.24 através do diagrama de Chapin.

Neste livro, os algoritmos dados como exemplos são geralmente simples. Apesar disto, todos eles são desenvolvidos de forma estruturada, como se eles fossem mais complexos do que realmente são. Após algum treino e experiência, a técnica do desenvolvimento estruturado do algoritmo, nos casos mais simples, pode ser usada mentalmente.

0.5. LINGUAGENS DE PROGRAMAÇÃO

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 transrito para uma linguagem que o computador possa "entender", direta ou indiretamente.

Os computadores só podem executar diretamente os algoritmos expressos em **linguagem de máquina**, que é um conjunto de instruções capazes de ativar diretamente os dispositivos eletrônicos do computador. Esta linguagem tem vários inconvenientes para os humanos. É diferente para cada tipo de computador, pois depende de sua arquitetura. Além disto, é extremamente rudimentar e exige que, mesmo as operações mais simples ainda sejam refinadas, para expressá-las em termos de registros, acumuladores e outros dispositivos da máquina. É totalmente expressa em forma numérica (binária ou hexadecimal), que a torna pouco expressiva para os humanos. Exige um cuidado extremo para se estabelecer o posicionamento dos dados e das instruções na memória.

Apesar de tudo isto, nos primeiros computadores, a linguagem de máquina era a única em que se podia fazer a programação. Mas logo, surgiu a idéia de se escreverem programas em **Linguagem simbólica**, mais conhecida por **Linguagem Assembler** ou **Linguagem montadora**, em que a linguagem de máquina é expressa não apenas por números, mas também por letras e símbolos mais significativos para os humanos. O posicionamento dos dados e instruções na memória é também feito de forma simbólica. Um programa em linguagem Assembler, para controlar o computador, deve primeiro ser transformado em linguagem de máquina. Como cada comando da linguagem Assembler corresponde a um comando em

C ESCREVA TERMOS DE FIBONACCI INFERIORES A L
INTEGER L,A,B,C

```

    RECEBA L
    READ(5,10) L
    10 FORMAT (1I0)
    PROCESSE OS 2 PRIMEIROS TERMOS
    A = 1
    IF (A.LT.L) WRITE (3,10) A
    B = 1
    IF (B.LT.L) WRITE (3,10) B
    PROCESSE TERMOS RESTANTES
    20 CONTINUE
    C = A + B
    IF (C.GE.L) GO TO 30
    WRITE (3,10) C
    A = B
    B = C
    GO TO 20
    30 CONTINUE
    PARE
    STOP
    END

```

Fig. 0.25 Programa escrito em linguagem FORTRAN.

linguagem de máquina, esta transformação pode ser feita facilmente pelo próprio computador, através de um programa chamado montador ou Assembler, escrito em linguagem de máquina e geralmente distribuído pelo próprio fabricante do computador. Posteriormente, quase todas as linguagens Assembler passaram também a ser dotadas de alguns comandos que correspondem a várias instruções em linguagem de máquina.

O sucesso da linguagem Assembler logo animou os primeiros pesquisadores a criar linguagens em que a programação era feita através de uma notação matemática e de algumas palavras da língua inglesa, deixando ao próprio computador a tarefa (não muito simples) de traduzir este programa para a linguagem de máquina, através de um programa chamado **compilador**.

A primeira destas linguagens, que teve ampla aceitação, surgiu em 1957 e é ainda hoje utilizada. Trata-se da linguagem FORTRAN, nome formado com partes das palavras "Formula Translation". A figura 0.25 mostra como se programaria em FORTRAN o algoritmo para se escrever os termos de Fibonacci inferiores a L. Observando que .LT. significa menor que (*less than*) e .GE. significa maior que ou igual (*greater than or equal to*), o entendimento do programa não é difícil.

Além da grande facilidade, uma imensa vantagem de se escrever os programas em linguagens de alto nível, como passaram a ser chamadas a linguagem FORTRAN e outras que se seguiram, é a sua quase total independência da máquina a ser usada. O programador não precisa se deter em particularidades do computador que irá usar. Um programa escrito em linguagem de alto nível, geralmente com pouquíssimas alterações, é aceito por qualquer computador.

Como o FORTRAN se mostrou mais adequado à programação de natureza técnica e científica, logo surgiu a idéia de se criar uma linguagem mais voltada para problemas de natureza comercial e administrativa. Assim, surgiu em 1959 o COBOL ("Common Business Oriented Language"). A linguagem COBOL permite que um programa seja escrito com uma forma mais próxima das linguagens naturais (*no caso, o inglês*), facilitando o seu entendimento pelos humanos e permitindo uma documentação mais clara. O seu sucesso foi tanto que até hoje, anos depois da sua criação, o COBOL ainda é a linguagem mais usada nos computadores, em todo o mundo. Porém, um programa escrito em COBOL fica muito mais longo do que se fosse usada outra linguagem. Entenda-se, também, que a maior utilização do COBOL se explica não porque ele seja uma linguagem melhor que as outras, mas sim porque os problemas de natureza comercial e administrativa para os quais ela é mais adequada, são os que mais ocupam os computadores. A figura 0.26 mostra como seria programado em COBOL o algoritmo dos termos de Fibonacci inferiores a L.

```

IDENTIFICATION DIVISION.
PROGRAM ID. FIBONACCI.
REMARKS. DETERMINA OS TERMOS DA SEQUENCIA DE FIBONACCI MENORES
QUE UM VALOR L PREDETERMINADO.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT ENTRADA ASSIGN TO READER.
  SELECT SAIDA ASSIGN TO PRINTER.

DATA DIVISION.
FILE SECTION.
FD ENTRADA, RECORDING MODE IS F,
LABEL RECORDS ARE OMITTED,
DATA RECORD IS CARTAO.
01 CARTAO.
  02 L PIC 9(10).

FD SAIDA, RECORDING MODE IS F,
LABEL RECORDS ARE OMITTED,
DATA RECORD IS TERMO.
01 TERMO PICTURE X(133).

WORKING STORAGE SECTION.
  77 A      PIC 9(10).
  77 B      PIC 9(10).
  77 C      PIC 9(10).

PROCEDURE DIVISION.
  OPEN INPUT ENTRADA, OUTPUT SAIDA.
  READ ENTRADA.
  COMPUTE A = 1.
  IF A IS LESS THAN L, WRITE SAIDA FROM A.
  COMPUTE B = 1.
  IF B IS LESS THAN L, WRITE SAIDA FROM B.
  TERMOS-RESTANTES:
    COMPUTE C = A + B.
    IF C IS NOT LESS THAN L, GO TO FINAL.
    WRITE SAIDA FROM C.
    COMPUTE A = B.
    COMPUTE B = C.
    GO TO TERMOS-RESTANTES.
  FINAL:
    CLOSE ENTRADA, SAIDA.
  STOP RUN.

```

Fig. 0.26 Programa em linguagem COBOL.

Com o FORTRAN na área técnica e científica e o COBOL na área comercial e administrativa, surgiu por volta de 1963 a idéia de se cirar uma linguagem única que fosse igualmente apropriada para todas as áreas de aplicação — apareceu assim a linguagem PL/I ("Programming Language One"). O entusiasmo provocado pelas linguagens de alto nível precedentes animou os criadores da PL/I a elaborar uma linguagem extremamente vasta, com numerosíssimos recursos. Embora ainda bastante usada, a linguagem PL/I exige um compilador muito complexo e o seu aprendizado completo é longo e trabalhoso. A figura 0.27 mostra o mesmo algoritmo programado em PL/I.

Com objetivo oposto ao da PL/I, foi criada por volta de 1964 a linguagem BASIC ("Beginer's All-Purpose Symbolic Instruction Code") que inicialmente pretendia ser apenas uma linguagem mais simples que o FORTRAN, para permitir aos não especialistas a utilização dos computadores. A simplicidade do BASIC foi possivelmente o principal motivo de sua ampla aceitação, anos mais tarde, com o aparecimento dos microcomputadores. A sua evolução, porém, foi um tanto desordenada, não tendo ainda sido

```

FIBONACCI: PROCEDURE OPTIONS (MAIN);
/* ESCRVE TERMOS DE FIBONACCI INFERIORES A L */
DECLARE (L,A,B,C) FIXED BINARY (S1);
GET LIST L;           /* RECEBE L */
A = 1;                /* PROCESSE OS 2 PRIMEIROS TERMOS */
IF A < L THEN PUT SKIP LIST (A);
B = 1;
IF B < L THEN PUT SKIP LIST (B);
REPITA:               /* PROCESSE OS TERMOS RESTANTES */
  C = A + B;
  IF C >= L THEN GO TO FINAL;
  PUT SKIP LIST (C);
  A = B;
  B = C;
  GO TO REPITA;
FINAL:                 /* PARE */
END FIBONACCI;

```

Fig. 0.27 Programa em linguagem PL/I.

padronizada. Falta-lhe uma maior coerência e uniformidade, além de se ressentir de construções mais adequadas que só foram efetivamente compreendidas após o aparecimento das técnicas de desenvolvimento estruturado de algoritmos, que resultam em **programas estruturados**. Algunas pessoas, porém, atribuem a estas falhas da linguagem BASIC a causa da sua aceitação, pois esta linguagem pode assim acompanhar a evolução dos mais inesperados recursos que vão surgindo nos computadores. Por exemplo, hoje, o BASIC possui um comando para emitir no computador um som com a duração e freqüência desejada (comando BEEP), para fazer desenhos no vídeo (PLOT, DRAWN etc.), para usar cores no vídeo etc. A figura 0.28 dá a versão BASIC do mesmo algoritmo.

Para evitar muitas limitações do FORTRAN e para permitir uma melhor expressão dos algoritmos, foi criada em 1960 a linguagem ALGOL ("Algorithmic Language"). A sua lógica perfeita tem sido o motivo da sua utilização, muito difundida sobretudo na Europa. Infelizmente, a linguagem ALGOL é muito ampla e o programa compilador correspondente só está desenvolvido para poucos computadores. Mas há computadores cuja arquitetura foi criada sobre esta linguagem, como o Burroughs 6700, em que a própria linguagem de máquina é uma variante do ALGOL.

```

10 REM TERMOS DE FIBONACCI MENORES QUE L
20 REM RECEBA L
30 INPUT L
40 REM PROCESSE OS 2 PRIMEIROS TERMOS
50 LET A = 1
60 IF A < L THEN PRINT A
70 LET B = 1
80 IF B < L THEN PRINT B
90 REM PROCESSE TERMOS RESTANTES
100 LET C = A + B
110 IF C >= L THEN 160
120 PRINT C'
130 LET A = B
140 LET B = C
150 GO TO 100
160 END

```

Fig. 0.28 Programa em linguagem BASIC.

```

BEGIN % TERMOS DE FIBONACCI MENORES QUE L
FILE CARD (KIND=READER), SAIDA(KIND=PRINTER);
INTEGER L,A,B,C;
% RECEBA L
READ (CARD,/,L);
% PROCESSE OS 2 PRIMEIROS TERMOS
A := 1;
IF A < L THEN WRITE (SAIDA,/,A);
B := 1;
IF B < L THEN WRITE (SAIDA,/,B);
% PROCESSE OS TERMOS RESTANTES
C := A + B;
WHILE C < L DO
BEGIN
  WRITE(SAIDA,/,C);
  A := B;
  B := C;
  C := A + B;
END;
% TERMINE
END.

```

Fig. 0.29 Programa em linguagem ALGOL.

A figura 0.29 mostra o algoritmo dos termos de Fibonacci inferiores a L escrito em ALGOL.

A partir de 1968, N. Wirth, em Zurique, na Suíça, desenvolveu uma nova linguagem que foi denominada PASCAL, em homenagem ao matemático francês que, em 1642, foi o primeiro a planejar e a construir uma máquina de calcular. Esta linguagem, sob muitos aspectos semelhante à ALGOL, foi criada para facilitar o ensino da Informática, mas a sua proposital simplicidade, aliada a uma adequada perfeição lógica, logo a tornaram bastante difundida, estando a sua preferência em franca ascensão. O algoritmo da seqüência de Fibonacci está programado na linguagem PASCAL na figura 0.30.

Além destas, diversas outras linguagens de alto nível foram criadas e podem ser encontradas em uso, como: RPG ("Report Program Generator"), para problemas comerciais; FORTH, para programação de

```

PROGRAM FIBONACCI (INPUT,OUTPUT);
(* CALCULA TERMOS DE FIBONACCI MENORES QUE L.*)
VAR L,A,B,C: INTEGER;
BEGIN
  READ (L); (*RECEBA O VALOR DE L*)
  A := 1; (*PROCESSE OS 2 PRIMEIROS TERMOS*)
  IF A < L THEN WRITELN(A);
  B := 1;
  IF B < L THEN WRITELN(B);
  C := A + B; (*PROCESSE TERMOS RESTANTES*)
  WHILE C < L DO
    BEGIN
      WRITELN(C);
      A := B;
      B := C;
      C := A + B
    END;
END.

```

Fig. 0.30 Programa em linguagem PASCAL.

jogos em micros; a linguagem C, para programação de sistemas operacionais; APL ("A Programming Language"), para programação interativa; ADA (do nome da Ada Lovelace, que se tornou a primeira programadora da história ao fazer programas para a máquina de Babbage), para utilização geral; LOGO (palavra grega que significa conhecimento), para estudo de problemas ciberneticos etc.

A escolha da linguagem de programação para se usar um computador depende antes de tudo da existência de um programa compilador (que traduz o algoritmo escrito na linguagem escolhida para a linguagem de máquina) ou de um programa interpretador (que interprete cada comando do programa e execute uma série de instruções que a ele correspondem). Existindo compiladores ou interpretadores para diversas linguagens, a escolha pode ser pela linguagem preferida ou mais familiar para o programador, por aquela melhor implementada no computador, por aquela mais adequada para o tipo de aplicação que se deseja fazer etc.

Mas, para se resolver um problema num computador, mais importante que a escolha da linguagem de programação é o desenvolvimento de um algoritmo adequado. Este algoritmo deve ser desenvolvido objetivando sobretudo a clareza, permitindo que os erros cometidos sejam detectados o quanto antes, evitando excessivas revisões e visando facilitar futuras modificações. Para se conseguir isto de uma forma natural e eficiente, devem-se adotar técnicas de desenvolvimento estruturado dos algoritmos. Depois disto, a programação consistirá quase que só numa transcrição do algoritmo obtido.

Itens Fundamentais

Uma vez conhecida a definição de algoritmo, a partir deste capítulo será introduzido um conjunto particular de regras e convenções para o seu desenvolvimento.

As normas apresentadas neste livro não são únicas, nem tampouco universais, mas foram estabelecidas a partir da experiência adquirida pelos autores no ensino de Programação de Computadores.

O leitor deve dedicar atenção especial a este capítulo, pois ele contém os fundamentos necessários para a compreensão dos demais.

1.1. CONSTANTES

Uma constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa.

Uma constante pode ser um número (como se conhece na Matemática), um valor lógico ou uma seqüência de caracteres quaisquer com algum significado para o problema em estudo. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica ou literal.

1.1.1. Constante numérica

A representação de uma constante numérica nos algoritmos é feita no sistema decimal, podendo ser um número com ou sem parte fracionária.

Exemplo 1.1

- a) 25;
- b) 3,14.

Na Matemática é comum a existência de constantes com uma parte exponencial, isto é, um fator 10 elevado a um expoente inteiro. Neste caso, é usada a notação já conhecida.

Exemplo 1.2

$$7.8 \times 10^3$$

A constante numérica pode ser positiva ou negativa, de acordo com o sinal que precede os algarismos formadores do número. Caso não exista um sinal, a constante é considerada positiva. Além disso, também o expoente da parte exponencial possui um sinal indicando deslocamento da vírgula para a direita ou para a esquerda, conforme o sinal seja positivo ou negativo, respectivamente.

A seguir, são apresentados alguns exemplos de constantes numéricas.

□ Exemplo 1.3

- | | | | |
|------------|------------------------|-----------------------------|-----------------|
| a) 15; | d) $0,342$; | g) $-2,5 \times 10^3$; | j) 10^4 ; |
| b) $+15$; | e) $-2,726$; | h) $2,5 \times 10^{-3}$; | l) -10^6 ; |
| c) -15 ; | f) $9,7 \times 10^6$; | i) $-0,91 \times 10^{-9}$; | m) -10^{-9} . |

1.1.2. Constante lógica

É um valor lógico, isto é, que só pode ser falso ou verdadeiro, usado em proposições lógicas, conforme será visto mais adiante.

Só existem duas constantes deste tipo, sendo representadas pelas palavras falso e verdadeiro.

1.1.3. Constante literal

Uma constante deste tipo pode ser qualquer seqüência de caracteres (letras, dígitos ou símbolos especiais) que forme um literal com algum significado para o problema em estudo.

Toda constante literal que aparece no algoritmo será colocada entre aspas para que não seja confundida com outro item qualquer.

□ Exemplo 1.4

- | | |
|---------------------|----------------|
| a) "JOSÉ DA SILVA"; | d) "X1Y2W3"; |
| b) "MENSAGEM"; | e) "*A!B?:"; |
| c) "12345"; | f) "23/09/55". |

Note que um numeral entre aspas é considerado como uma seqüência de dígitos (literal), e não como uma constante numérica.

Também não se deve confundir uma constante lógica (por exemplo, falso) com uma seqüência de caracteres (literal), que apareça entre aspas (por exemplo "FALSO").

1.1.4. Exercício de fixação

Identificar o tipo de cada uma das constantes abaixo:

- a) 21;
- b) "BOLA";
- c) "VERDADEIRO";
- d) $0,21 \times 10^3$;
- e) falso.

1.2. VARIÁVEIS

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 programa. Embora uma 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 cálculo das raízes de uma equação de 2.º grau ($ax^2 + bx + c = 0$), os identificadores A, B e C podem representar as posições de memória que armazenam os coeficientes da equação, fazendo, neste caso, o papel das variáveis na Matemática.

1.2.1. Formação dos identificadores

Um identificador é formado por um ou mais caracteres, sendo que o primeiro caractere deve, obrigatoriamente, ser uma letra e os caracteres seguintes, letras ou dígitos, não sendo permitido o uso de símbolos especiais.

□ Exemplo 1.5

a) Identificadores permitidos

A	X5
NOTA	A32B
MATRÍCULA	F1G3H5

b) Identificadores não permitidos

5B	X - Y
E(13)	NOTA [1]
A:B	B * D

É recomendável que os nomes das variáveis sejam os mais significativos possíveis, isto é, que reflitam, da melhor maneira, a natureza dos valores que nelas estão sendo armazenados. Isto ajuda muito no entendimento do algoritmo.

A título de exemplo: se a variável vai armazenar o salário de um empregado, por que não escolher o identificador SALARIO para representá-la?

1.2.2. Declaração de variáveis

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo **numéricas**, **lógicas** e **literais**.

Como saber, então, qual o tipo da variável, ou seja, que conjunto de valores ela pode armazenar?

Para indicar o tipo de uma ou mais variáveis é usada a **declaração de variáveis**. Além disso, no momento em que se declara uma variável, é feita a associação do nome escolhido, ou identificador, com a respectiva posição de memória que o mesmo passa a simbolizar.

Uma vez declarada a variável, qualquer referência que se faça ao seu identificador implica a referência ao conteúdo do local da memória representado pelo mesmo.

Toda declaração de variáveis tem a seguinte forma:

declare lista-de-identificadores nome-do-tipo

onde:

declare	é uma palavra-chave do algoritmo;
lista-de-identificadores	são os nomes escolhidos para as variáveis, que devem estar separados por vírgula;
nome-do-tipo	é uma das três palavras-chaves, numérico, lógico ou literal, que indicam o tipo associado às variáveis.

□ Exemplo 1.6

- a) **declare NOTA,CÓDIGO,X5 numérico;**
- b) **declare TESTE,SIM lógico;**
- c) **declare NOME, END1,END2,F1F2literal.**

Neste exemplo, os identificadores NOTA, CÓDIGO e X5 são declarados variáveis numéricas e, portanto, simbolizam posições na memória do computador capazes de armazenar apenas valores numéricos. Qualquer tentativa de atribuição de valores de outro tipo nestas variáveis é considerada como erro.

Tendo em vista facilitar a compreensão do algoritmo, os identificadores são escritos sempre com letras maiúsculas, enquanto que as palavras-chaves são escritas com minúsculas e grifadas. Como se verá mais adiante existem muitas outras palavras-chaves que aparecem no algoritmo, prevalecendo para todas elas a mesma regra de legibilidade.

Denomina-se **palavra-chave** aquela que tem um significado próprio, independente do algoritmo em que esteja inserida. Em vista disto, as palavras-chaves não podem ser usadas como identificadores.

1.2.3. Exercícios de fixação

○ 1.2.3.1. Assinalar com um X os identificadores válidos:

- | | | |
|------------|---------------------|-------------|
| () VALOR | () SALÁRIO-LÍQUIDO | () B248 |
| () X2 | () NOTA*DO*ALUNO | () A1B2C3 |
| () 3 × 4 | () MARIA | () KM/H |
| () XYZ | () NOMEDAEMPRESA | () SALA215 |
| () "NOTA" | () AH! | () M(A) |

○ 1.2.3.2. Supondo-se que as variáveis NOM, PROF, ID e SALÁRIO serão utilizadas para armazenar o nome, profissão, idade e salário de uma pessoa, escrever o conjunto de declarações necessárias para criar essas variáveis e associar às mesmas os respectivos tipos básicos.

1.3. COMENTÁRIOS

A esta altura o leitor já percebeu a preocupação existente com a clareza do algoritmo, ou seja, o grau de facilidade que as pessoas terão em compreender o que nele está descrito.

Um instrumento de grande valia usado para esta finalidade denomina-se **comentário**. Ele é um texto, ou simplesmente uma frase, que aparece sempre delimitado por chaves (`{comentário}`).

Os comentários podem ser colocados em qualquer ponto do algoritmo onde se façam necessários.

No exemplo a seguir é mostrado um conjunto de declarações onde foram introduzidos comentários com o intuito de explicar o significado de cada uma das variáveis.

□ Exemplo 1.7

```
declare MAT, {número de matrícula do aluno}
NOTA, {total de pontos obtidos no semestre letivo}
COD {código do curso}
    numérico
declare NOME, {nome completo do aluno}
END, {endereço do aluno}
C {conceito final}
    literal
```

Importante. Todo algoritmo deve conter comentários, a fim de que as pessoas possam entendê-los mais facilmente.

1.4. EXPRESSÕES ARITMÉTICAS

Denomina-se expressão aritmética aquela cujos operadores são aritméticos e cujos operandos são constantes e/ou variáveis do tipo numérico.

O conjunto de operações básicas adotado é o que se conhece da Matemática, a saber:

adição	subtração
multiplicação	divisão
potenciação	radiciação

A seguir, são apresentadas algumas expressões aritméticas, onde aparecem as operações mencionadas acima.

□ Exemplo 1.8

- | | | |
|---------------------|----------------|---------------------------|
| a) $X + Y;$ | d) TOTAL/N; | g) $\sqrt{FI + G^2} - H;$ |
| b) $X - Y;$ | e) $\sqrt{P};$ | h) $A \times B + C;$ |
| c) $2 \times NOTA;$ | f) $SOMA^2;$ | i) $TOT/M + K^Y.$ |

A notação utilizada para expressões aritméticas nos algoritmos é, basicamente, a mesma da Matemática, a menos das seguintes restrições:

- não é permitido omitir o operador de multiplicação, o que é comum nas expressões matemáticas. Isto evita confusão quanto aos nomes de variáveis, pois numa expressão da forma $AB + C$, como saber se AB é o nome de uma variável ou a multiplicação entre os conteúdos de duas variáveis, cujos nomes são A e B ? Pelo mesmo motivo, o operador da multiplicação deve sempre ser escrito com letra minúscula (x);
- nas expressões aritméticas, as operações guardam entre si uma relação de prioridade, tal como na Matemática (tabela 1.1).

Tabela 1.1 Prioridade das operações

Prioridade	Operação
1. ^o	potenciação, radiciação
2. ^o	multiplicação, divisão
3. ^o	adição, subtração

Para se obter uma sequência de cálculo diferente, vários níveis de parênteses podem ser usados para quebrar as prioridades definidas. Não é permitido o uso de colchetes e chaves, uma vez que estes símbolos são utilizados nos algoritmos para outras finalidades.

□ Exemplo 1.9

- $\sqrt{P \times (P - A) \times (P - B) \times (P - C)};$
- $A - B \times (C + D/(E - 1) - F) + G;$

Por questão de uniformidade, adota-se um único símbolo para cada um dos operadores. Assim sendo, não é permitido o uso do símbolo ' \div ' e do 'ponto' para indicar a divisão e a multiplicação, respectivamente.

1.4.1. Funções

Além das operações básicas, anteriormente citadas, podem-se usar nas expressões aritméticas algumas funções muito comuns na Matemática.

Na tabela 1.2, encontram-se algumas das principais funções existentes e o resultado fornecido por cada uma delas. Também é permitido o uso de todas as funções trigonométricas já conhecidas da Matemática.

Tabela 1.2 Funções de EA, EAx, EAy — expressões aritméticas

Nome	Resultado
LOG (EA)	logaritmo na base 10 de EA
LN (EA)	logaritmo neperiano de EA
EXP (EA)	o número e (base dos logaritmos neperianos) elevado a EA
ABS (EA)	valor absoluto de EA
TRUNCA (EA)	a parte inteira de um número fracionário
ARREDONDA (EA)	transforma, por arredondamento, um número fracionário em inteiro
SINAL (EA)	fornecer o valor -1, +1 ou zero conforme o valor de EA seja negativo, positivo ou igual a zero
QUOCIENTE (EAx, EAy)	quociente inteiro da divisão de EAx por EAy
RESTO (EAx, EAy)	resto da divisão de EAx por EAy

) A função atua sobre um argumento numérico, que é o resultado obtido após a avaliação da expressão aritmética entre parênteses.

+ As palavras que designam as funções são escritas com letras maiúsculas.

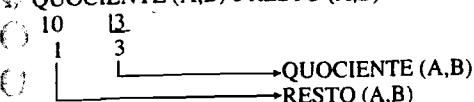
Exemplo 1.10

- X + SEN (A + B + C);
- QUOCIENTE (NOTA,2) × 100 + T;
- X + LN(Y) - ABS (A - B);
- H² - G × F × SINAL (C + D).

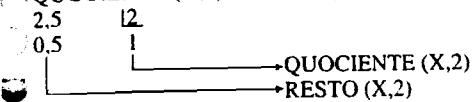
Exemplo 1.11

Sejam A,B,X,Y variáveis do tipo numérico, quais os resultados fornecidos por cada uma das seguintes funções, onde A = 10, B = 3, X = 2,5 e Y = 1,2.

QUOCIENTE (A,B) e RESTO (A,B)



QUOCIENTE (X,2) E RESTO (X,2)



SINAL (X + Y - A), SINAL (A - B² + Y) e SINAL (A - 4 × X)
SINAL (X + Y - A) = SINAL (2,5 + 1,2 - 10) = SINAL (-6,3) = -1
SINAL (A - B² + Y) = SINAL (10 - 9 + 1,2) = SINAL (2,2) = 1
SINAL (A - 4 × X) = SINAL (10 - 4 × 2,5) = SINAL (0) = 0

ARREDONDA (A - X), ARREDONDA (B + Y) e ARREDONDA (Y - X)

ARREDONDA (A - X) = ARREDONDA (10 - 2,5) =

ARREDONDA (7,5) = 8

ARREDONDA (B + Y) = ARREDONDA (3 + 1,2) =

ARREDONDA (4,2) = 4

ARREDONDA (Y - X) = ARREDONDA (1,2 - 2,5) =

ARREDONDA (-1,3) = -1

TRUNCA (B² + X), TRUNCA (A/3 + 1) e TRUNCA (X - 3,2)

TRUNCA (B² + X) = TRUNCA (9 + 2,5) = TRUNCA (11,5) = 11

TRUNCA (A/3 + 1) = TRUNCA (3,333 + 1) = TRUNCA (4,333) = 4

TRUNCA (X - 3,2) = TRUNCA (2,5 - 3,2) = TRUNCA (-0,7) = 0

ABS (A - B³) e ABS (A - B)

ABS (A - B³) = ABS (10 - 27) = ABS (-17) = 17

ABS (A - B) = ABS (10 - 3) = ABS (7) = 7

EXP (Y × (B + 2) - 6) e EXP (sqrt(A + 2 × B) - B)

EXP (Y × (B + 2) - 6) = EXP (1,2 × (3 + 2) - 6) = EXP (0) = e⁰ = 1

EXP (sqrt(A + 2 × B) - B) = EXP (sqrt(10 + 2 × 3) - 3) = EXP (1) =

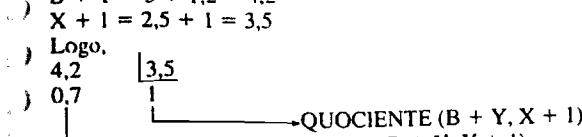
e¹ = 2,72

QUOCIENTE (B + Y, X + 1) e RESTO (B + Y, X + 1)

B + Y = 3 + 1,2 = 4,2

X + 1 = 2,5 + 1 = 3,5

Logo,



1.4.2. Exercício de fixação

O Sendo P, Q, R e S variáveis do tipo numérico, cujos conteúdos são iguais a 2,3, 12 e 4,5, respectivamente, quais os valores fornecidos por cada uma das expressões aritméticas abaixo?

- a) 100 × QUOCIENTE (Q,P) + R
- b) P × RESTO (R,S) - $\frac{Q}{2}$
- c) SINAL (S - R) + EXP $\left(Q^2 - \frac{R}{4} \times P - 3\right)$

- d) $\sqrt{R + P^2} + \text{ARREDONDA}(S)$
- e) RESTO (S, P + 1) - Q × R
- f) $1 + \sqrt[3]{P^3 + 2 \times R} - \text{TRUNCA}(S - 1)$
- g) $1 + \text{QUOCIENTE}(R + S, Q^2) \times \text{SINAL}(2 \times P \times Q - R)$
- h) P + ARREDONDA (2,9 + TRUNCA (0,3 + S) × 2)

1.5. EXPRESSÕES LÓGICAS

É comum nos algoritmos surgirem situações em que a execução de uma ação, ou sequência de subações, está sujeita a uma certa condição. Esta condição é representada no texto do algoritmo por meio de uma expressão lógica.

Denomina-se expressão lógica a expressão cujos operadores são lógicos e cujos operandos são relações, constantes e/ou variáveis do tipo lógico.

1.5.1. Relações

Uma expressão relacional, ou simplesmente relação, é uma comparação realizada entre dois valores de mesmo tipo básico. Estes valores são representados na relação através de constantes, variáveis ou expressões aritméticas, estas últimas para o caso de valores numéricos.

Os operadores relacionais, que indicam a comparação a ser realizada entre os termos da relação, são os conhecidos da Matemática, a saber:

= igual a	< menor que
≠ diferente de	≥ maior ou igual a
> maior que	≤ menor ou igual a

O resultado obtido de uma relação é sempre um valor lógico.

Exemplo 1.12

Analisando a relação numérica X + Y = Z, o resultado será verdadeiro ou falso conforme o valor da expressão aritmética X + Y seja igual ou diferente do conteúdo da variável Z, respectivamente.

Exemplo 1.13

Eis algumas relações

- a) A ≠ B;
- b) NOME = "JOÃO";
- c) B² - 4 × A × C < 0;
- d) X = 1.

Exemplo 1.14

Dadas as variáveis numéricas X, Y, Z e as variáveis literais NOME e COR, observar os resultados obtidos para as relações a partir dos valores atribuídos a estas variáveis.

VARIÁVEIS					RELACIONES		
X	Y	Z	COR	NOME	$X^2 + Y > Z$	COR = "AZUL"	$NOME \neq "JOSÉ"$
1	2	5	"AZUL"	"PAULO"	falso	verdadeiro	verdadeiro
4	3	1	"VERDE"	"JOSÉ"	verdadeiro	falso	falso
1	1	2	"BRANCO"	"PEDRO"	falso	falso	verdadeiro
1	2	1	"AZUL"	"JOSÉ"	verdadeiro	verdadeiro	falso

1.5.2. Exercício de fixação

○ Dadas as variáveis numéricas A e B, as variáveis literais NOME e PROFISSÃO, completar o quadro a seguir, preenchendo os espaços em branco com os resultados lógicos (falso ou verdadeiro) obtidos como resultado das relações, tendo em vista os valores atribuídos a estas variáveis:

VARIÁVEIS			RELACIONES		
A	B	NOME	PROFISSÃO	$A + 1 \geq \sqrt{B}$	NOME \neq "ANA" PROFISSÃO = "MÉDICO"
3	16	"MIRIAM"	"ADVOGADO"		
5	64	"PEDRO"	"MÉDICO"		
2,5	9	"ANA"	"PROFESSOR"		

1.5.3. Operadores lógicos

A Álgebra das Proposições define três conectivos usados na formação de novas proposições a partir de outras já conhecidas. Estes conectivos são os operadores nas expressões lógicas, a saber:

- e — para a conjunção
- ou — para a disjunção
- não — para a negação

Neste contexto considera-se uma proposição como sendo uma variável lógica, uma relação ou uma expressão lógica composta.

Duas proposições podem ser combinadas pelo conectivo e para formar uma proposição chamada conjunção das proposições originais. A conjunção das proposições p e q representa-se por:

$p \wedge q$ lê-se p e q

A conjunção de duas proposições é verdadeira se e somente se ambas as proposições são verdadeiras.

Sejam as seguintes proposições:

- p: OK, onde OK é uma variável lógica cujo conteúdo é verdadeiro;
- q: A = 0, onde o valor de A é 3;
- r: TESTE, onde TESTE é uma variável lógica cujo conteúdo é falso;
- s: B ≠ 1, onde o conteúdo de B é 2.

□ Exemplo 1.15

Qual será o valor lógico (falso, verdadeiro) das conjunções:

- a) p \wedge s;
- b) p \wedge r;
- c) q \wedge s;
- d) q \wedge r.

Ora, considerando que a conjunção exige que as duas proposições sejam verdadeiras para que o seu resultado seja verdadeiro, tem-se:

- a) p \wedge s significa OK e B ≠ 1 e, portanto, é uma proposição verdadeira;
- b) p \wedge r significa OK e TESTE e, portanto, é uma proposição falsa, já que TESTE é falso;
- c) q \wedge s significa A = 0 e B ≠ 1, que do mesmo modo é uma proposição falsa, já que A ≠ 0;
- d) q \wedge r significa A = 0 e TESTE e, portanto, é uma proposição falsa.

Se p é verdadeira e q é verdadeira, então p \wedge q será verdadeira, de outro modo, p \wedge q será falsa. Um meio conveniente de estabelecer esta conclusão é pela tabela:

Tabela 1.3

P	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

onde:
V: verdadeira
F: falsa

Duas proposições quaisquer podem ser combinadas pelo conectivo ou (com sentido de e/ou) para formar uma nova proposição que é chamada disjunção das duas proposições originais. A disjunção de duas proposições p e q é designada por:

$p \vee q$ lê-se p ou q

A disjunção de duas proposições é verdadeira se e somente se, pelo menos, uma delas for verdadeira.

□ Exemplo 1.16

Para as quatro proposições do exemplo anterior qual será o valor lógico das disjunções:

- a) p \vee s;
- b) p \vee r;
- c) q \vee s;
- d) q \vee r.

Ora,

- a) p \vee s é verdadeira;
- b) p \vee r é verdadeira;
- c) q \vee s é verdadeira;
- d) q \vee r é falsa.

Considerando que na disjunção pode-se considerar verdadeira a proposição que contiver, pelo menos, uma subproposição verdadeira, tem-se:

Tabela 1.4

P	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Dada uma proposição p qualquer, uma outra proposição, chamada negação de p, pode ser formada escrevendo-se "É falso que" antes de p ou, se possível, inserindo a palavra "não" em p. Simbolicamente, designa-se a negação de p por:

$\neg p$ lê-se não p

□ Exemplo 1.17

Tendo em vista as proposições p e r anteriores, tem-se:

- a) $\neg p$ significa \neg OK, que é falso;
- b) $\neg r$ significa \neg TESTE, que é verdadeiro.

Logo, pode-se concluir que se p é verdadeira, então $\neg p$ é falsa; se p é falsa, então $\neg p$ é verdadeira.

Pela tabela:

Tabela 1.5

p	$\neg p$
V	F
F	V

Exemplo 1.18

Eis algumas expressões lógicas

- a) $A + B = 0 \text{ e } C \neq 1$;
- b) TESTE ou $A \times C > B$;
- c) não TESTE e COR = "AZUL".

Onde A , B e C são variáveis numéricas, TESTE é uma variável lógica e COR, uma variável literal. O resultado obtido da avaliação de uma expressão lógica é sempre um valor lógico, isto é, um falso ou verdadeiro. Por este motivo, pode-se considerar uma única relação como sendo uma expressão lógica.

Exemplo 1.19

Dadas as variáveis numéricas X , Y e Z , contendo os valores 2, 5 e 9, respectivamente; a variável literal NOME, contendo o literal "MARIA"; e a variável lógica SIM, contendo o valor lógico falso, observar os resultados obtidos das expressões lógicas a seguir.

- a) $X + Y > Z \text{ e } NOME = "MARIA"$
 $2 + 5 > 9 \text{ e } "MARIA" = "MARIA"$
 falso e verdadeiro
 falso
- b) SIM ou $Y \geq X$
 falso ou $5 \geq 2$
 falso ou verdadeiro
 verdadeiro
- c) não SIM e QUOCIENTE (Z, Y) + 1 = X
 não falso e QUOCIENTE (9,5) + 1 = 2
 não falso e verdadeiro
 verdadeiro e verdadeiro
 verdadeiro
- d) NOME = "JORGE" e SIM ou $X^2 < Z + 10$
 "MARIA" = "JORGE" e falso ou $4 < 19$
 falso e falso ou verdadeiro
 falso ou verdadeiro
 verdadeiro

1.5.4. Prioridade

Como mostrado nas letras c e d do exemplo anterior, pode-se ter mais de um operador lógico na mesma expressão. Em alguns casos, conforme os valores envolvidos, a ordem em que são efetuadas as operações lógicas afeta o resultado final. Assim, como acontece entre as operações aritméticas, também existe uma relação de prioridade entre os operadores lógicos. Na tabela 1.6 são apresentadas as prioridades entre todos os operadores conhecidos, visto que podem estar presentes na mesma expressão lógica.

Também nas expressões lógicas vários níveis de parênteses podem ser utilizados com a finalidade de estabelecer uma nova ordem de execução entre os operadores lógicos.

Tabela 1.6 Prioridade das operações

Prioridade	Operador
1. ^a	aritmético
2. ^a	relacional
3. ^a	não
4. ^a	e
5. ^a	ou

Exemplo 1.20

- a) $A = 1 \text{ e } (B + C \neq 0 \text{ ou } K \leq 2)$;
- b) não (TOTAL $\geq 2 \text{ e } A \neq B$) ou TESTE.

1.5.5. Exercícios de fixação

1.5.5.1. Considerando as variáveis e valores do exercício de fixação 1.5.2 e mais a variável lógica TESTE, contendo o valor lógico falso, avaliar as expressões a seguir, para cada uma das três combinações de valores apresentadas:

- a) $A + 1 \geq \sqrt{B} \text{ ou } NOME \neq "ANA"$
- b) $A + 1 \geq \sqrt{B} \text{ e } PROFISSAO = "MÉDICO"$
- c) $NOME \neq "ANA" \text{ ou } PROFISSAO = "MÉDICO" \text{ e } A + 1 \geq \sqrt{B}$
- d) $PROFISSAO = "MÉDICO" \text{ ou } TESTE$
- e) não TESTE e $(A + 1 \geq \sqrt{B} \text{ ou } \text{não } PROFISSAO = "MÉDICO")$
- f) não $(A + 1 \geq \sqrt{B} \text{ e } TESTE)$

1.5.5.2. Considerando A , B e C variáveis numéricas, contendo os valores, 1, 4,5 e 8, respectivamente; NOME e COR variáveis literais contendo as sequências de caracteres "TANIA" e "BRANCO" e TESTE variável lógica contendo o valor verdadeiro, determinar os resultados obtidos da avaliação das seguintes expressões lógicas:

- a) $A = 1 \text{ e } TESTE$
- b) $NOME = "PEDRO" \text{ ou } COR = "BRANCO"$
- c) não TESTE ou RESTO ($B, 2$) = 0,5
- d) $C < 10 \text{ ou } TESTE \text{ e } COR = "PRETO"$
- e) $A^2 + \sqrt[3]{C} = 3 \text{ e } (A - \text{TRUNCA}(B + C) > 13 \text{ ou } NOME = "ANA")$
- f) TESTE e não TESTE

1.6. EXPRESSÕES LITERAIS

Uma expressão literal é aquela formada por operadores literais e operandos que são constantes e/ou variáveis do tipo literal.

Embora estas expressões tenham grande importância no estudo de programação, o objetivo desta seção é apenas o de introduzir o assunto. Isto porque as operações entre valores literais são basicamente diversificadas e dependem das características de cada Linguagem de Programação.

Supondo que A e B são variáveis literais e que o símbolo " $|$ " é um operador de concatenação de literais, a expressão

$A | B$

fornece como resultado um único literal formado pelo conteúdo de A seguido do conteúdo de B .

□ Exemplo 1.21

Se A contém o literal "BOLA" e B contém o literal "PRETA", o valor fornecido pela expressão A | B é o literal "BOLAPRETA".

Além da concatenação de literais, é comum a existência de outras operações desta natureza nas Linguagens de Programação, sendo que normalmente elas são encontradas na forma de funções.

Citam-se dentre as mais comuns aquelas que fornecem como resultado:

- o comprimento do literal (número de caracteres);
- os n primeiros caracteres de um literal, onde n é um número inteiro;
- os n últimos caracteres de um literal, onde n é um número inteiro.

1.6.1. Exercício de fixação

○ Quais seriam os valores obtidos das expressões literais a seguir, supondo A, B e C variáveis literais contendo os valores "BENS", "!", "PARA", respectivamente?

- A | B | C
- A | C | B
- C | A | B

1.7. COMANDO DE ATRIBUIÇÃO

A partir dos conceitos introduzidos no capítulo 0, pode-se definir comando como sendo a descrição de uma ação a ser executada em um dado momento.

Da presente seção ao final deste capítulo, discute-se como descrever as ações básicas contidas em um algoritmo, ou seja, os comandos e as estruturas disponíveis para o desenvolvimento de algoritmos e o conjunto de regras e convenções adotadas para a representação dos mesmos.

O primeiro dos comandos considerado denomina-se comando de atribuição. Este comando permite que se forneça um valor a uma certa variável, onde a natureza deste valor tem de ser compatível com o tipo da variável na qual está sendo armazenado.

O comando de atribuição tem a forma geral apresentada a seguir:

identificador ← expressão

onde:

identificador
←
expressão

é o nome da variável à qual está sendo atribuído o valor;
é o símbolo de atribuição;
pode ser uma expressão aritmética, expressão lógica ou expressão literal de cuja avaliação é obtido o valor a ser atribuído à variável.

□ Exemplo 1.22

- K ← 1;
- COR ← "VERDE";
- TESTE ← falso;
- A ← B;
- MÉDIA ← SOMA/N;
- COD ← N² + 1 ≥ 5;
- SIM ← X = 0 e Y ≠ 2
- TOTAL ← $\sqrt{N} + X^2 + Y$.

Esses comandos atribuem, dinamicamente, às variáveis K, COR, TESTE, A, MÉDIA, COD, SIM e TOTAL os valores fornecidos à direita do símbolo de atribuição.

Nos comandos em que o valor é representado por uma expressão aritmética ou lógica, estas devem ser avaliadas em primeiro lugar para que, então, o resultado obtido seja armazenado na variável.

Ainda sobre o exemplo 1.19, as variáveis K, MÉDIA, SOMA, N, X e Y devem ser do tipo numérico, as variáveis TESTE, COD e SIM, do tipo lógico e a variável COR, do tipo literal.

Não existem restrições quanto ao tipo das variáveis A e B, desde que ambas sejam do mesmo tipo básico.

Finalmente, o leitor pode estranhar o fato de que as constantes e variáveis sejam consideradas expressões, tendo em vista a forma geral do comando de atribuição. Isto é correto na medida em que uma expressão não passa de uma representação simbólica de um certo valor, que pode ser obtido de sua avaliação. Portanto, constantes e variáveis numéricas são consideradas como expressões numéricas na sua forma mais simples. Raciocínio análogo pode ser aplicado para expressões lógicas e expressões literais.

1.7.1. Exercícios de fixação

○ 1.7.1.1. Sendo

SOMA, NUM, X variáveis numéricas,
NOME, COR, DIA variáveis literais, e
TESTE, COD, TUDO variáveis lógicas,
assinalar os comandos de atribuição considerados inválidos:

- NOME ← 5
- SOMA ← NUM + 2 × X
- TESTE ← COD ou $X^2 \neq SOMA$
- TUDO ← SOMA
- COR ← "PRETO" - \sqrt{X}
- X ← X + 1
- NUM ← "*ABC*"
- DIA ← "SEGUNDA"
- SOMA + 2 ← $X^2 - \sqrt{NUM}$
- X ← NOME ≥ COD

○ 1.7.1.2. Quais os valores armazenados em SOMA, NOME e TUDO, supondo-se que NUM, X, COR, DIA, TESTE e COD valem, respectivamente, 5; 2,5; "AZUL"; "TERÇA"; falso e verdadeiro?

- NOME ← DIA
- SOMA ← NUM²/X + ARREDONDA(X + 1)
- TUDO ← não TESTE ou COD e SOMA < X

1.8. COMANDOS DE ENTRADA E SAÍDA

No capítulo 0 foram introduzidos alguns conceitos, com relação a estrutura e funcionamento de um computador, que serão mencionados nesta seção.

Sabe-se que as unidades de entrada e saída são dispositivos que possibilitam a comunicação entre o usuário e o computador.

Por exemplo, através de um teclado, o usuário consegue dar entrada ao programa e aos dados na memória do computador. Por sua vez, o computador pode emitir os resultados e outras mensagens para o usuário através de uma impressora de linhas.

Seja a seguinte situação: início da execução de um programa que se encontra armazenado na memória principal do computador. Como e quem determina o momento da entrada dos dados para o programa e a saída dos resultados obtidos para o usuário?

Isto é tarefa do programador e ele assim o faz quando, no desenvolvimento do algoritmo, descreve as ações a serem executadas pelo computador.

Os comandos de entrada e saída são as ferramentas para esta finalidade.

leia lista-de-identificadores

onde:

leia
lista-de-identificadores

é uma palavra-chave;
são os nomes das variáveis, separados por vírgula,
nas quais serão armazenados os valores provenientes
do meio de entrada.

Exemplo 1.23

Supondo que NOTA e NUM são variáveis do tipo numérico, o comando

leia NOTA, NUM

indica que dois valores numéricos serão lidos de uma unidade de entrada, quando este comando for executado. Os valores serão armazenados nas posições de memória (variáveis) identificadas pelos nomes NOTA e NUM.

Analogamente, um comando de saída tem a forma geral:

escreva lista-de-identificadores e/ou constantes

onde:

escreva
lista-de-identificadores

é uma palavra-chave;
são os nomes das variáveis, cujos conteúdos serão mostrados ao usuário através de um meio de saída.
Além dos conteúdos das variáveis, o valor de uma constante pode ser emitido diretamente.

Exemplo 1.24

O comando

escreva A,X,35

indica que a constante 35 e mais os conteúdos das posições de memória, representados pelos identificadores A e X, serão exibidos em uma unidade de saída.

Exemplo 1.25

- leia** X;
- leia** NOME,N,Y;
- escreva** K,SOMA;
- escreva** 21,"NOME",N;
- escreva** "TABELA DE PREÇOS".

A seguir são apresentados dois exemplos que mostram, com maiores detalhes, o funcionamento dos comandos de entrada e saída.

Exemplo 1.26

Foram digitadas três linhas, onde cada uma delas contém o nome e a nota de um aluno, como se segue:

PAULO, 100
MARIA, 75
JOSÉ, 80

Escrever o(s) comando(s) de entrada que leia(m) estas linhas e armazene(m) os valores na memória principal.

Ora, é preciso que existam seis variáveis (posições de memória) para o armazenamento dos dados contidos nas linhas (dois valores em cada linha).

É necessário, ainda, que haja uma correspondência de número, ordem e tipo entre os valores digitados e as variáveis que aparecem no comando de entrada.

leia X,A,Y,B,Z,C

ao ser executado, armazena nestas variáveis, na ordem em que aparecem, os valores digitados.

A situação da memória, após a execução do comando, é esquematizada, a seguir, pela figura 1.1:

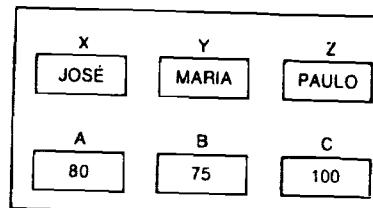


Fig. 1.1 Esquema de memória.

Uma segunda alternativa seria ler as linhas individualmente, ou seja, usar um comando de entrada distinto para cada uma das linhas. Assim, os comandos

leia X,A**leia** Y,B**leia** Z,C

se executados nesta ordem, efetuam a leitura das linhas uma a uma, armazenando em cada par de variáveis o par de valores da respectiva linha.

Pode-se concluir, então, que, a cada execução de um comando de entrada, uma nova linha é colocada à disposição para leitura.

Exemplo 1.27

Seja a unidade de saída uma impressora de linhas. Escrever o(s) comando(s) de saída que imprima(m) os conteúdos das posições de memória A,B,C,X,Y,Z do exemplo anterior.

De maneira análoga à entrada de dados, a cada execução de um comando de saída, uma nova linha do formulário de impressão é posicionada. Portanto, existem inúmeras formas de apresentação dos dados na unidade de saída. Algumas destas formas são mostradas a seguir:

COMANDO

escreva X,A,Y,B,Z,C

FORMULÁRIO

JOSÉ, 80, MARIA, 75, PAULO, 100

escreva X,A
escreva Y,B
escreva Z,C

JOSÉ, 80 MARIA, 75 PAULO, 100

escreva X
escreva A
escreva Y
escreva B
escreva Z
escreva C

JOSÉ 80 MARIA 75 PAULO 100

Os comandos são executados na ordem em que aparecem.

1.8.1. Exercício de fixação

○ Supondo

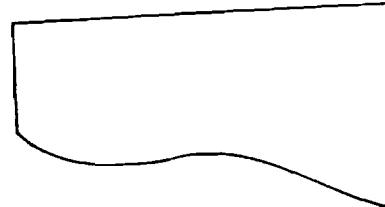
N e P variáveis do tipo literal

X e A variáveis do tipo numérico

e uma linha digitada contendo os valores

MMAA, 25

interpretar a seqüência de comandos, a seguir, e preencher o formulário de impressão com os valores que serão impressos na unidade de saída.



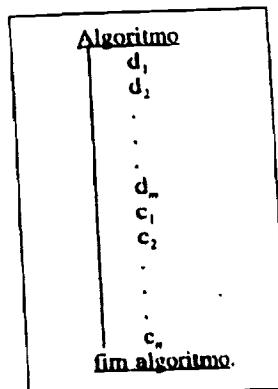
```

X ← 0
leia N,A
X ← X + A
P ← N
escreva P,X
X ← X + A
escreva X
A ← X
escreva N,X,A
  
```

Como já foi visto na seção 0.3, um algoritmo completo é constituído por declarações, comandos e estruturas. As declarações e os principais comandos acabaram de ser mostrados, faltando, para concluir este capítulo, apenas algumas considerações sobre as principais estruturas que serão utilizadas ao longo deste livro. Para melhor explicá-las, além do texto do algoritmo, será mostrado também o diagrama de Chapin correspondente.

1.9. ESTRUTURA SEQUENCIAL

Num algoritmo aparecem em primeiro lugar as declarações seguidas por comandos que, se não houver indicação em contrário, deverão ser executados numa seqüência linear, seguindo-se o texto em que estão escritos, de cima para baixo. Neste livro, os algoritmos são iniciados com a palavra **Algoritmo** e terminados com a expressão **fim algoritmo**.



Texto do algoritmo

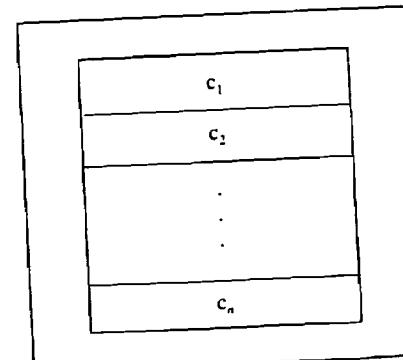


Diagrama de Chapin

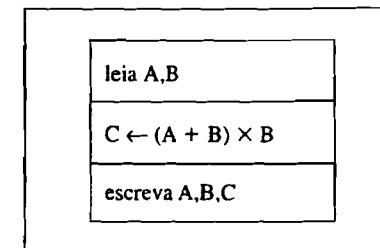
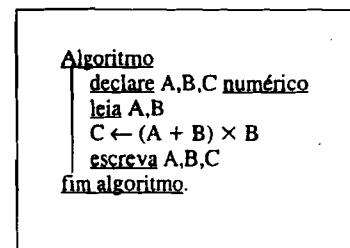
onde:

d₁, d₂, ..., d_n
c₁, c₂, ..., c_n

são declarações
são comandos

Observação: As declarações não são representadas no diagrama de Chapin.

□ Exemplo 1.28



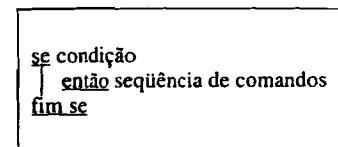
Neste exemplo, após serem definidos os tipos das variáveis A,B,C, os valores de A e B serão lidos, o valor de C calculado e os valores contidos em A, B e C serão escritos.

1.10. ESTRUTURA CONDICIONAL

A estrutura condicional permite a escolha do grupo de ações e estruturas a ser executado quando determinadas condições, representadas por expressões lógicas, são ou não satisfeitas. Neste livro, esta estrutura é delimitada pelo comando **se** e pela expressão **fim se**.

Esta estrutura pode se apresentar de duas formas.

1.10.1. Estrutura condicional simples



Texto do algoritmo

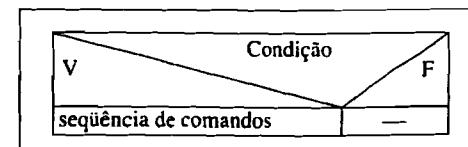
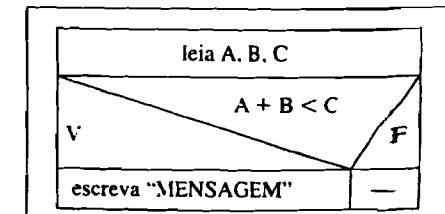
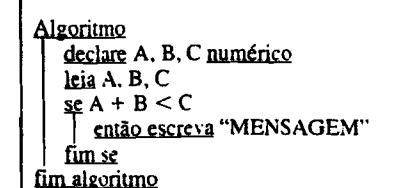


Diagrama de Chapin

Neste caso, a seqüência de comandos só será executada se a condição for verdadeira.

Observação: É bom que fique claro que, até o final deste capítulo, ao ser mencionada "seqüência de comandos", está implícito que ela contém um ou mais comandos e pode conter uma ou mais estruturas.

□ Exemplo 1.29



Neste exemplo, após serem definidos os tipos das variáveis A, B, C, os valores de A, B, C serão lidos; em seguida, caso a soma de A + B seja menor do que C, uma mensagem será escrita. Se A + B for maior ou igual a C, nenhuma ação será empreendida.

1.10.2. Estrutura condicional composta

```
se condição
  então seqüência A de comandos
  senão seqüência B de comandos
fim se
```

Texto do algoritmo

Neste caso, a seqüência A de comandos só será executada se a condição (expressão lógica) for verdadeira e a seqüência B de comandos só será executada se a condição for falsa.

Exemplo 1.30

Algoritmo

```
declare A, B, X, Y numérico
leia A, B
se A = B
  então X ← 1,5
        Y ← 2,5
  senão X ← -1,5
        Y ← -2,5
fim se
escreva X, Y
fim algoritmo.
```

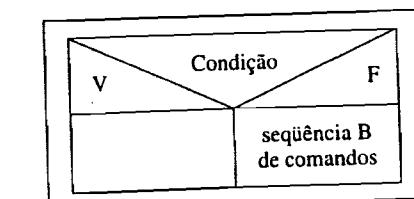
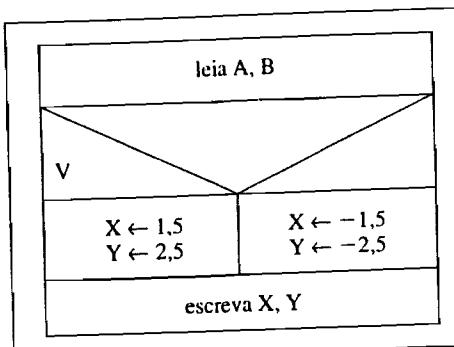


Diagrama de Chapin



Neste exemplo, após serem definidos os tipos das variáveis A, B, X, Y, os valores de A, B serão lidos; em seguida, se o valor contido em A for igual ao valor contido em B, serão atribuídos a X e Y os valores 1,5 e 2,5, respectivamente; caso contrário, se o valor contido em A for diferente do valor contido em B, serão atribuídos a X e Y os valores -1,5 e -2,5, respectivamente; finalmente, os valores contidos em X e Y serão escritos.

Nos exemplos seguintes, será usada a técnica de refinamentos sucessivos.

Exemplo 1.31

Fazer um algoritmo que leia três valores inteiros, determine e imprima o menor deles:

Algoritmo

```
Defina os tipos das variáveis
Leia os números
Determine o menor número
Escreva o menor número
fim algoritmo.
```

Após a leitura do problema, foi esboçado o algoritmo acima com as principais ações a serem executadas. Como se pode observar, essas ações não estão detalhadas o suficiente para tornar o algoritmo

fácilmente executável. É isto que será feito, agora, aplicando-se a cada uma das ações um ou mais refinamentos.

```
Ref. Leia os números
| leia A, B, C
fim ref.
```

```
Ref. Determine o menor número
| se A < B e A < C
|   então MENOR ← A
|   senão Determine o menor dentre B e C
| fim se
fim ref.
```

```
Ref. Determine o menor dentre B e C
| se B < C
|   então MENOR ← B
|   senão MENOR ← C
| fim se
fim ref.
```

```
Ref. Escreva o menor número
| escreva MENOR
fim ref.
```

```
Ref. Defina os tipos das variáveis
| declare A,B,C,MENOR numérico
fim ref.
```

O que foi escrito até aqui reflete o que deve ser feito durante o desenvolvimento do algoritmo. Para se ter uma visão geral do algoritmo, de como ele será executado, inserem-se os refinamentos nos seus respectivos lugares e colocam-se comentários onde se fizerem necessários. Assim, obtém-se:

Algoritmo	(Definição do tipo das variáveis)
declare A,B,C, MENOR numérico	(Leitura dos números)
leia A,B,C	(Determinação do menor número)
se A < B e A < C	
então MENOR ← A	
senão se B < C	
então MENOR ← B	
senão MENOR ← C	
fim se	
escreva MENOR	(Escrita do menor número)
fim algoritmo.	

Exemplo 1.32

Dados três valores, X, Y, Z, verificar se eles podem ser os comprimentos dos lados de um triângulo e, se forem, verificar se é um triângulo equilátero, isósceles ou escaleno. Se eles não formarem um triângulo, escrever uma mensagem.

Antes de começar a elaboração do algoritmo, torna-se necessária a revisão de algumas propriedades e definições.

Propriedade — O comprimento de cada lado de um triângulo é menor do que a soma dos comprimentos dos outros dois lados.

Definição 1 — Chama-se triângulo equilátero ao triângulo que tem os comprimentos dos três lados iguais.

Definição 2 — Chama-se triângulo isósceles ao triângulo que tem os comprimentos de dois lados iguais. Portanto, todo triângulo equilátero é também isósceles.

Definição 3 — Chama-se triângulo escaleno ao triângulo que tem os comprimentos de seus três lados diferentes.

Algoritmo

Defina os tipos das variáveis

Leia os números

Se existe triângulo

então Verifique o tipo do triângulo

senão Escreva mensagem

fim se

fim algoritmo.

Ref. Leia os números

leia X,Y,Z

fim ref.

Ref. Se existe triângulo

se $X < Y + Z \wedge Y < X + Z \wedge Z < X + Y$

fim ref.

Ref. Verifique o tipo do triângulo

se $X = Y \wedge X = Z$

então escreva "TRIÂNGULO EQÜILÁTERO"

senão Verifique se ele é escaleno ou isósceles

fim se

fim ref.

Ref. Verifique se ele é escaleno ou isósceles

se $X = Y \text{ ou } X = Z \text{ ou } Y = Z$

então escreva "TRIÂNGULO ISÓSCELES"

senão escreva "TRIÂNGULO ESCALENO"

fim se

fim ref.

Ref. Escreva mensagem

escreva "NÃO EXISTE TRIÂNGULO"

fim ref.

Ref. Defina o tipo das variáveis

declare X,Y,Z numérico

fim ref.

Finalmente, inserindo-se os refinamentos no algoritmo, tem-se:

Algoritmo

{Definição do tipo das variáveis}

declare X,Y,Z numérico

{Leitura dos números}

leia X,Y,Z

{Verificação da existência de triângulo}

se $X < Y + Z \wedge Y < X + Z \wedge Z < X + Y$

então

se $X = Y \wedge X = Z$

então escreva "TRIÂNGULO EQÜILÁTERO"

senão se $X = Y \text{ ou } X = Z \text{ ou } Y = Z$

então escreva "TRIÂNGULO ISÓSCELES"

senão escreva "TRIÂNGULO ESCALENO"

fim se

senão

escreva "NÃO EXISTE TRIÂNGULO"

fim se

fim algoritmo.

Exemplo 1.33

Dados três valores distintos, colocá-los em ordem crescente:

Algoritmo

Defina o tipo das variáveis

Leia os números

Ordene esses números

Escreva o resultado

fim algoritmo.

Ref. Leia os números

leia L,M,N

fim ref.

Ref. Ordene esses números

Armazene em L o menor valor

Armazene em M o valor intermediário e em N o maior valor

fim ref.

Ref. Armazene em L o menor valor

se $L > M \text{ ou } L > N$

então se $M < N$

então troque L com M

senão troque L com N

fim se

fim se

fim ref.

Ref. Troque L com M

AUXILIAR \leftarrow L

L \leftarrow M

M \leftarrow AUXILIAR

fim ref.

A troca de conteúdo entre duas variáveis é muito comum no desenvolvimento de algoritmos, portanto torna-se necessário uma explicação da utilização de uma variável auxiliar, que neste caso recebeu o nome de AUXILIAR. Se, simplesmente, fossem feitas as seguintes atribuições $L \leftarrow M$, $M \leftarrow L$ o que ocorreria?

Supondo-se que, inicialmente, em L estivesse o valor 27 e em M o valor 15, ao se executar o comando $L \leftarrow M$ em L seria armazenado o valor 15 e o valor 27 desapareceria.

Ao se executar o comando $M \leftarrow L$ em M, seria armazenado o valor contido, atualmente, em L, ou seja, o valor 15 e, portanto, L e M armazenariam o mesmo valor, e o número 27 teria desaparecido.

Agora, supondo-se que, inicialmente, em L estivesse o valor 27, em M, o valor 15 e executando o refinamento anterior, obtém-se:

- em AUXILIAR é armazenado o número 27;
- em L é armazenado o número 15;
- em M é armazenado o que está contido em AUXILIAR, ou seja, o valor 27.

Logo, a troca foi processada, pois L passou a conter o valor 15 e M, o valor 27.

Feito este esclarecimento, pode-se continuar a refinar as ações iniciais.

Ref. Troque L com N

AUXILIAR \leftarrow L

L \leftarrow N

N \leftarrow AUXILIAR

fim ref.

Ref. Armazene em M o valor intermediário e em N o maior valor

se $M > N$

então AUXILIAR \leftarrow M

M \leftarrow N

N \leftarrow AUXILIAR

fim se

fim ref.

Ref. Escreva o resultado

escreva L,M,N

fim ref.

Ref. Defina o tipo das variáveis

declare AUXILIAR,L,M,N numérico

fim ref.

Finalmente, inserindo-se os refinamentos em seus devidos lugares, obtém-se:

Algoritmo

declare AUXILIAR,L,M,N numérico

{Definição do tipo das variáveis}

leia L,M,N

{Leitura dos números}

se $L > M$ ou $L > N$

então se $M < N$

então AUXILIAR \leftarrow L

L \leftarrow M

M \leftarrow AUXILIAR

senão AUXILIAR \leftarrow L

L \leftarrow N

N \leftarrow AUXILIAR

fim se

fim se

se $M > N$

então AUXILIAR \leftarrow M

M \leftarrow N

N \leftarrow AUXILIAR

fim se

escreva L,M,N

fim algoritmo

{Escrita do resultado}

1.10.3. Exercícios de fixação

1.10.3.1. Após a execução do seguinte trecho de um algoritmo

Algoritmo

se $A2 \leq B3$

então TESTE \leftarrow verdadeiro

senão TESTE \leftarrow falso

fim se

C \leftarrow TESTE

.

.

.

fim algoritmo

em C estará armazenado o valor falso se, originalmente:

a) $A2 < B3$

b) $A2 \leq B3$

c) $A2 \geq B3$

d) $A2 > B3$

e) $A2 = B3$

Q 1.10.3.2. Após a execução do seguinte trecho de um algoritmo

Algoritmo

```
D ← 0
se A ≤ B e C ≥ B
    então D ← 5
fim se
fim algoritmo.
```

em D estará armazenado o valor 5 se:

- a) $A < B < C$
- b) $A \leq B \leq C$
- c) $A < B \leq C$
- d) $B < C < A$
- e) nenhuma das respostas acima.

Q 1.10.3.3. Dado o seguinte algoritmo:

Algoritmo A,B,C,I,J,K numérico

```
declare A,B,C,I,J,K numérico
A ← 32
C ← 2
I ← 5
B ← A1/5
J ← C × 3/4
se B > J
    então K ← 8 ×  $\frac{I}{6^2/C}$ 
    senão K ← A + 1/A - 1
fim se
escreva B,J,K
fim algoritmo.
```

que valores serão escritos?

Q 1.10.3.4. Dado o seguinte algoritmo:

Algoritmo declare A,B,I,M numérico

```
leia M
se M ≠ 0
    então I ← TRUNCA (M/12)
        A ← M/12
        B ← ARREDONDA (M/12)
        se RESTO (M,12) ≥ 6
            então I ← I + 1
        fim se
        escreva A,B,I
fim se
fim algoritmo.
```

que valores seriam escritos se, em sucessivas execuções, fossem lidos os valores 30; 19; 27; 60; 0?

Q 1.10.3.5. Quais os valores escritos pelo algoritmo a seguir?

Algoritmo

```
declare A,B,C,MAT,XX,Y,YY numérico
X ← 0
XX ← 0
Y ← 0
YY ← 0
A ← 2
B ← 5
C ← 4
MAT ← o último algarismo do dia em que você nasceu
se MAT ≥ 5
    então X ← (A + B)1/2 + C/2 × A
        Y ← RESTO (B,A)/A - QUOCIENTE (C,B)
    senão XX ← QUOCIENTE (B,A)/A2 + RESTO (B2,C)
        YY ← A/2 × C + (3 + A)1/2/A
fim se
escreva X,Y,XX,YY
fim algoritmo.
```

1.11. ESTRUTURA DE REPETIÇÃO

A estrutura de repetição permite que uma sequência de comandos seja executada repetidamente até que uma determinada condição de interrupção seja satisfeita. Neste livro, esta estrutura é delimitada pelo comando **repita** e pela expressão **fim repita** e a interrupção é feita através do comando **interrompa**.

A condição de interrupção que deve ser satisfeita é representada por uma expressão lógica. Esta estrutura pode se apresentar nas formas que se seguem:

Primeira forma. Interrupção de início

```
repita
    se condição
        então interrompa
    fim se
    sequência B de comandos
fim repita
```

Texto do algoritmo

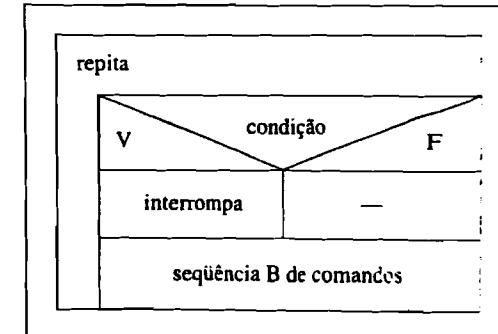


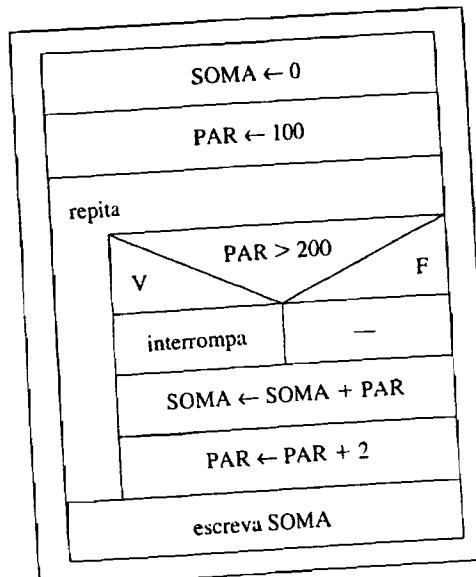
Diagrama de Chapin

Nesta estrutura, a sequência B de comandos será repetida até que a condição seja satisfeita. Quando isto ocorrer, a repetição é interrompida e a sequência de comandos que vier logo após a expressão **fim repita** passa a ser executada.

Nos três exemplos 1.34, 1.35 e 1.36 são mostrados três algoritmos que, embora ligeiramente diferentes, calculam por um processo repetitivo a soma dos números pares desde 100 até 200, inclusive.

□ Exemplo 1.34

Algoritmo
declare PAR,SOMA numérico
 SOMA ← 0
 PAR ← 100
repita
 se PAR > 200
 então interrompa
 fim se
 SOMA ← SOMA + PAR
 PAR ← PAR + 2
fim repita
escreva SOMA
fim algoritmo.



Neste algoritmo, após os tipos das variáveis PAR e SOMA serem definidos, em SOMA é armazenado o valor zero e em PAR, o valor 100. Em seguida, o conjunto de comandos que gera os números pares e efetua a soma deles é repetido até que o valor armazenado em PAR seja maior do que 200, quando, então, a estrutura de repetição é interrompida e o comando, que vem logo após a expressão **fim repita**, é executado, ou seja, o valor contido em SOMA é escrito.

Segunda forma. Interrupção no interior

repita
 seqüência A de comandos
 se condição
 então interrompa
 fim se
 seqüência B de comandos
fim repita

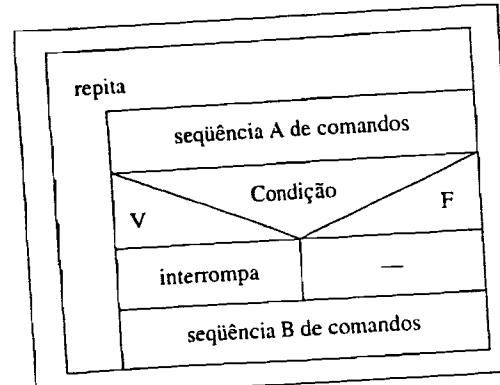
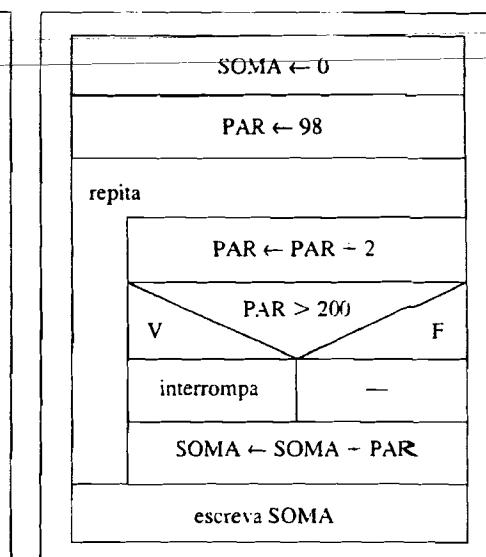


Diagrama de Chapin

Nesta estrutura, as seqüências A e B de comandos serão repetidas até que a condição seja satisfeita. Quando isto ocorrer, a repetição é interrompida e a seqüência de comandos que vier logo após a expressão **fim repita** passa a ser executada.

□ Exemplo 1.35

Algoritmo
declare PAR, SOMA numérico
 SOMA ← 0
 PAR ← 98
repita
 PAR ← PAR + 2
 se PAR > 200
 então interrompa
 fim se
 SOMA ← SOMA + PAR
 fim repita
escreva SOMA
fim algoritmo.



Neste algoritmo, após os tipos das variáveis PAR e SOMA serem definidos, em SOMA é armazenado o valor 0 e em PAR, o valor 98. Em seguida, os comandos que geram os números pares e o que calcula a soma deles são repetidos até que o valor armazenado em PAR seja maior que 200, quando, então, a estrutura de repetição é interrompida e o comando que vem logo após a expressão **fim repita** é executado, ou seja, o valor contido em SOMA é escrito.

Terceira forma. Interrupção no fim

repita
 seqüência A de comandos
 se condição
 então interrompa
 fim se
fim repita

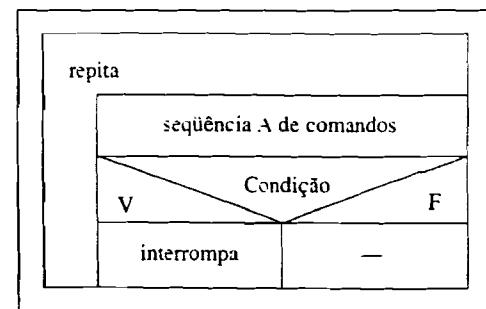


Diagrama de Chapin

Nesta estrutura, a seqüência A de comandos será repetida até que a condição seja satisfeita. Quando isto ocorrer, a repetição é interrompida e a seqüência de comandos que vier logo após a expressão **fim repita** passa a ser executada.

□ Exemplo 1.36

Algoritmo
declare PAR, SOMA numérico
 SOMA ← 0
 PAR ← 100

```

repita
    SOMA ← SOMA + PAR
    PAR ← PAR + 2
    se PAR > 200
        então interrompa
    fim se
fim repita
escreva SOMA
fim algoritmo.

```

Neste algoritmo, após o tipo das variáveis PAR e SOMA ser definido, em SOMA é armazenado o valor zero e em PAR, o valor 100. Em seguida, o conjunto de comandos que gera os números pares e efetua a soma deles é repetido até que o valor armazenado em PAR seja maior do que 200, quando, então, efetua a estrutura de repetição é interrompida e o comando que vem logo após a expressão **fim repita** é executado, ou seja, o valor contido em SOMA é escrito.

Observação: Deve-se ressaltar que a estrutura condicional

```

se condição
    então interrompa
fim se

```

é parte integrante da estrutura de repetição. Em benefício da maior clareza e simplicidade do algoritmo, ela deve ser usada apenas como estrutura condicional simples (sem a parte referente ao senão). Além disso, ela deve ocorrer, explicitamente, uma e uma só vez para cada estrutura de repetição e dentro do mesmo refinamento.

1.11.1. Exercícios de fixação

○ 1.11.1.1. Executando o seguinte algoritmo

Algoritmo

```

declare N.QUADRADO numérico
N ← 10
repita
    QUADRADO ← N2
    escreva QUADRADO
    se N = 1
        então interrompa
    fim se
    N ← N - 1
fim repita
fim algoritmo.

```

que valores serão escritos?

○ 1.11.1.2. Executando o seguinte algoritmo

Algoritmo

```

declare A,Q,TERMO numérico
A ← 1
Q ← 3
TERMO ← A
repita
    se TERMO ≥ 2000

```

```

    fim se
    escreva TERMO
    TERMO ← TERMO × Q
fim repita
fim algoritmo.

```

que valores serão escritos?

○ 1.11.1.3. Executando o seguinte algoritmo

Algoritmo

```

declare D,K,NÚMERO,SOMA numérico
leia NÚMERO
SOMA ← NÚMERO + 1
K ← QUOCIENTE (NÚMERO,2)
D ← 2
repita
    se D > K
        então interrompa
    fim se
    se RESTO (NÚMERO,D) = 0
        então SOMA ← SOMA + D
    fim se
    D ← D + 1
fim repita
escreva SOMA
fim algoritmo.

```

que valor seria escrito se fosse lido o valor 30?

Nos exemplos seguintes, será usada a técnica de refinamentos sucessivos.

□ Exemplo 1.37

Fazer um algoritmo que calcule o valor de $N!$ (fatorial de N), sendo que o valor inteiro de N encontra-se disponível em uma unidade de entrada e que:

- $N! = 1 \times 2 \times 3 \times \dots \times (N - 1) \times N;$
- $0! = 1$, por definição.

Algoritmo

```

Defina tipo das variáveis
leia N
Calcule o fatorial de N
Escreva o fatorial de N
fim algoritmo.

```

Ref. Calcule o fatorial de N

```

FATORIAL ← 1
I ← 2
repita
    se I > N
        então interrompa
    fim se

```

```
FATORIAL ← FATORIAL × I  
I ← I + 1  
fun repita  
fun ref.
```

Neste último refinamento, a variável I assume os valores 2, 3, 4,... até atingir o valor N + 1, quando, então, a estrutura de repetição é interrompida. Ainda dentro da estrutura, à medida que I passa a assumir um novo valor, este é multiplicado pelo conteúdo da variável FATORIAL e o produto é armazenado em FATORIAL.

Ref. Escreva o fatorial de N
| escreva FATORIAL
fim ref.

Ref. Defina tipo das variáveis
| declare FATORIAL,I,N numérico
sim ref.

Finalmente, inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

{Definição do tipo das variáveis}

declare FATORIAL,I,N numérico

leia N {Cálculo do fatorial de N}

FATORIAL ← 1

I ← 2

repita

se I > N
então interrompa

fim se

FATORIAL ← FATORIAL × I

I ← I + 1

fim repita {Escrita do fatorial de N}

escreva FATORIAL

fim algoritmo.

□ Exemplo 1.38

Exemplo 1.38
 Uma pessoa aplicou seu capital a juros e deseja saber, trimestralmente, a posição de seu investimento C inicial. Chamando de i a taxa de juros do trimestre, escrever uma tabela que dê para cada trimestre o rendimento auferido e o saldo acumulado durante um período de X anos, supondo-se que nenhuma retirada tenha sido feita.

Observações:

- a) Os valores de C , I e X estão disponíveis numa unidade de entrada.
 b) A fórmula para capitalização composta é dada por:

$$M_t = C(1+i)^t$$

onde:

M	montante após terem decorridos n trimestres;
C	capital inicial investido;
i	taxa de juros;
n	número de períodos decorridos (trimestres)

Algoritmo

Defina tipo das variáveis
leia CAPITAL,TAXA,X
Atribua valores iniciais necessários
repita

Calcule rendimento trimestral
Calcule montante trimestral
Escreva rendimento e montante trimestrais
se TRIMESTRE = N
 | enfão interrompa
 fim se
Modifique valor de TRIMESTRE
fim repita
n algoritmo.

Ref. Calcule rendimento trimestral
| RENDIMENTO ← TAXA × MONTANTE
|
| Últ.ref.

Ref. Calcule montante trimestral
 $\text{MONTANTE} \leftarrow \text{CAPITAL} \times (1 + \text{TAXA})^{\text{TRIMESTRE}}$
 fim ref

Ref. Escreva rendimento e montante trimestrais
escreva RENDIMENTO,MONTANTE
fim ref.

Ref. Modifique valor de TRIMESTRE
| TRIMESTRE ← TRIMESTRE + 1
fin ref.

```

Ref. Atribua valores iniciais necessários
  | MONTANTE ← CAPITAL
  | TRIMESTRE ← 1
  | N ← 4 × X
fim ref.

```

Ref. Defina tipo das variáveis
| declare CAPITAL,MONTANTE,N,RENDIMENTO,TAXA,
| TRIMESTRE,X número
sim ref.

Finalmente, inserindo-se no algoritmo inicial os refinamentos realizados, tem-se:

Algoritmo

```
{Definição do tipo das variáveis}  
declare CAPITAL,MONTANTE,N,RENDIMENTO,  
TAXA,TRIMESTRE,X, número  
leia CAPITAL,TAXA,X  
(Atribuição de valores iniciais necessários)  
MONTANTE ← CAPITAL  
TRIMESTRE ← 1  
N ← 4 × X  
repita  
    {Cálculo do montante trimestral}  
    RENDIMENTO ← TAXA × MONTANTE  
    {Cálculo do montante trimestral}  
    MONTANTE ← CAPITAL × (1 + TAXA)TRIMESTRE  
    {Escrita do rendimento e montante trimestrais}  
    escreva RENDIMENTO,MONTANTE  
    se TRIMESTRE = N  
        então interrompa  
    fim se  
    {Modifique valor de TRIMESTRE}  
    TRIMESTRE ← TRIMESTRE + 1  
fim repita  
fim algoritmo.
```

Exemplo 1.39

Num frigorífico existem 90 bois. Cada boi traz preso em seu pescoço um cartão contendo seu número de identificação e seu peso.

Fazer um algoritmo que escreva o número e peso do boi mais gordo e do boi mais magro.

Algoritmo

```
Defina tipo das variáveis  
Atribua valores iniciais necessários  
repita  
    se o número de bois examinados é igual a 90  
        então interrompa  
    fim se  
    Leia número de identificação e o peso do boi  
    Determine o boi mais gordo  
    Determine o boi mais magro  
    Conte o boi  
fim repita  
Escreva o número e o peso do boi mais gordo  
Escreva o número e o peso do boi mais magro  
fim algoritmo.
```

Ref. Leia número de identificação e peso do boi
| leia NÚMERO,PESO
fim ref.

Ref. Determine o boi mais gordo
se PESO > MAIORPESO
então MAIORPESO ← PESO
GORDO ← NÚMERO
fim se
fim ref.

Ref. Determine o boi mais magro
se PESO < MENORPESO
então MENORPESO ← PESO
MAGRO ← NÚMERO
fim se
fim ref.

Ref. Conte este boi
BOIS ← BOIS + 1
fim ref.

Ref. Escreva o número e o peso do boi mais gordo
escreva GORDO,MAIORPESO
fim ref.

Ref. Escreva o número e o peso do boi mais magro
escreva MAGRO,MENORPESO
fim ref.

Ref. Atribua valores iniciais necessários
BOIS ← 0
MAIORPESO ← 0
MENORPESO ← 100000
fim ref.

Ref. Defina tipo das variáveis
declare BOIS,GORDO,MAGRO,MAIORPESO,MENORPESO,NÚMERO,
PESO número
fim ref.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

```
{Definição do tipo das variáveis}  
declare BOIS,  
GORDO,  
MAGRO,  
MAIORPESO,  
MENORPESO,  
NÚMERO.  
(número de bois examinados)  
(número do boi mais gordo)  
(número do boi mais magro)  
(peso do boi mais gordo)  
(peso do boi mais magro)  
(número de identificação dos bois)
```

PESO
número

{peso dos bois}

{Atribuição de valores iniciais necessários}

BOIS ← 0
MAIORPESO ← 0
MENORPESO ← 100000

repita
| se BOIS = 90
| então interrompa
fim se

(Leitura do número de identificação e do peso do boi)

leia NÚMERO,PESO

(Determinação do boi mais gordo)

se PESO > MAIORPESO
então MAIORPESO ← PESO
GORDO ← NÚMERO

fim se

(Determinação do boi mais magro)

se PESO < MENORPESO
então MENORPESO ← PESO
MAGRO ← NÚMERO

fim se

(Contagem deste boi)

BOIS ← BOIS + 1

fim repita

(Escrita do número e do peso do boi mais gordo)

escreva GORDO,MAIORPESO

(Escrita do número e do peso do boi mais magro)

escreva MAGRO,MENORPESO

fim algoritmo.

1.11.2. Exercício de fixação

○ Se houver dois ou mais bois com o mesmo peso, maior que todos os demais, este algoritmo escreverá o número de qual deles?

□ Exemplo 1.40

Uma pesquisa sobre algumas características físicas da população de uma determinada região coletou os seguintes dados, referentes a cada habitante, para serem analisados:

- sexo (masculino, feminino);
- cor dos olhos (azuis, verdes, castanhos);
- cor dos cabelos (loiros, castanhos, pretos);
- idade em anos.

Para cada habitante, foi digitada uma linha com esses dados e a última linha, que não corresponde a ninguém, conterá o valor de idade igual a -1.

Fazer um algoritmo que determine e escreva:

- a) a maior idade dos habitantes;
- b) a porcentagem de indivíduos do sexo feminino cuja idade está entre 18 e 35 anos inclusive e que tenham olhos verdes e cabelos louros.

Algoritmo

Defina tipo das variáveis

Atribua valores iniciais necessários

repita

leia SEXO,OLHOS,CABELOS,IDADE

se IDADE = -1

então interrompa

fim se

Calcule maior idade

Conte os indivíduos pesquisados

Conte os indivíduos entre 18 e 35 anos, do sexo feminino, com olhos verdes e cabelos louros

fim repita

Calcule porcentagem de indivíduos entre 18 e 35 anos, do sexo feminino, com olhos

verdes e cabelos louros

Escreva a maior idade

Escreva a porcentagem calculada

fim algoritmo.

Ref. Calcule maior idade

se IDADE > MAIORIDADE

então MAIORIDADE ← IDADE

fim se

fim ref.

Ref. Conte os indivíduos pesquisados

TOTALINDIVÍDUOS ← TOTALINDIVÍDUOS + 1

fim ref.

Ref. Conte os indivíduos entre 18 e 35 anos, do sexo feminino, com olhos verdes

e cabelos louros

se IDADE ≥ 18 e IDADE ≤ 35 e SEXO = "FEMININO" e OLHOS = "VERDES"

e CABELOS = "LOUROS"

então INDIVÍDUOS ← INDIVÍDUOS + 1

fim se

fim ref.

Ref. Calcule porcentagem de indivíduos entre 18 e 35 anos, do sexo feminino,

com olhos verdes e cabelos louros

PORCENTAGEM ← INDIVÍDUOS × 100 / TOTALINDIVÍDUOS

fim ref.

Ref. Escreva a maior idade

escreva MAIORIDADE

fim ref.

Ref. Escreva a porcentagem calculada

escreva PORCENTAGEM

fim ref.

Ref. Atribua valores iniciais necessários
MAIORIDADE ← 0
TOTALINDIVÍDUOS ← 0
INDIVÍDUOS ← 0
fim ref.

Ref. Defina tipo das variáveis
declare IDADE,INDIVÍDUOS,MAIORIDADE,PORCENTAGEM,
TOTALINDIVÍDUOS numérico
declare CABELOS,OLHOS,SEXO literal
fim ref.

Finalmente, inserindo os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo {Definição do tipo das variáveis}

declare IDADE,
INDIVÍDUOS, {número de indivíduos com idade entre 18 e}
{35 anos, com olhos verdes e cabelos louros}
MAIORIDADE,
PORCENTAGEM {porcentagem de indivíduos com idade entre}
{18 e 35 anos, com olhos verdes e cabelos}
{louros}
TOTALINDIVÍDUOS {número de indivíduos pesquisados}
numérico
declare CABELOS,
OLHOS {cor de cabelos}
SEXO {cor dos olhos}
literal

{Atribuição de valores iniciais necessários}

MAIORIDADE ← 0
TOTALINDIVÍDUOS ← 0
INDIVÍDUOS ← 0

repita leia SEXO,OLHOS,CABELOS,IDADE

se IDADE = -1

então interrompa

fim se

{Cálculo da maior idade}

se IDADE > MAIORIDADE

então MAIORIDADE ← IDADE

fim se

{Contagem dos indivíduos pesquisados}

TOTALINDIVÍDUOS ← TOTALINDIVÍDUOS + 1

{Contagem dos indivíduos com idade entre 18 e 35 anos, do sexo}

{feminino, com olhos verdes e cabelos louros}

se IDADE ≥ 18 e IDADE ≤ 35 e SEXO = "FEMININO" e OLHOS = "VERDES"
e CABELOS = "LOUROS"

então INDIVÍDUOS ← INDIVÍDUOS + 1

fim se

fim repita

{Cálculo da porcentagem de indivíduos entre 18 e 35 anos,}
{do sexo feminino, com olhos verdes e cabelos louros}

PORCENTAGEM ← INDIVÍDUOS × 100 / TOTALINDIVÍDUOS

{Escrita da maior idade}

escreva MAIORIDADE

{Escrita da porcentagem calculada}

escreva PORCENTAGEM

fim algoritmo.

Exemplo 1.41

Para se determinar o número de lâmpadas necessárias para cada cômodo de uma residência, existem normas que dão o mínimo de potência de iluminação exigida por metro quadrado (m^2) conforme a utilização deste cômodo.

Seja a seguinte tabela tomada como exemplo:

UTILIZAÇÃO	CLASSE	POTÊNCIA/ m^2
quarto	1	15
sala de TV	1	15
salas	2	18
cozinha	2	18
varandas	2	18
escritório	3	20
banheiro	3	20

Supondo que só serão usadas lâmpadas de 60W, fazer um algoritmo que:

a) Leia um número indeterminado de linhas contendo cada uma:

- cômodo de uma residência;
- classe de iluminação deste cômodo;
- as duas dimensões do cômodo.

b) Calcule e escreva:

- b.1) para cada cômodo:
 - o cômodo;
 - a área do cômodo;
 - potência de iluminação;
 - número de lâmpadas necessárias;
- b.2) para toda a residência:
 - total de lâmpadas;
 - total de potência.

Observações:

- 1) Se o número calculado de lâmpadas for fracionário, considerar o menor inteiro que contenha esse número. Ex.: 8,3 → 9; 8,7 → 9.
- 2) A última linha, que não entrará nos cálculos, conterá no lugar do cômodo a palavra vazio.

Algoritmo

Defina tipo das variáveis

Atribua valores iniciais necessários

repita

leia CÔMODO,CLASSE,COMPRIMENTO,LARGURA

se CÔMODO = "VAZIO"

então interrompa

fim se

Calcule a área do cômodo

Calcule a potência de iluminação do cômodo
 Calcule o número de lâmpadas necessárias ao cômodo
 Calcule o total de lâmpadas da residência
 Calcule o total de potência da residência
 Escreva CÔMODO, área do cômodo, potência de iluminação, número de lâmpadas necessárias
fim repita
 Escreva o total de lâmpadas e o total de potência da residência
fim algoritmo.

Ref. Calcule a área do cômodo
 $\text{ÁREA} \leftarrow \text{COMPRIMENTO} \times \text{LARGURA}$
fim ref.

Ref. Calcule a potência de iluminação do cômodo
 se CLASSE = 1
 então POTÊNCIA $\leftarrow \text{ÁREA} \times 15$
 senão se CLASSE = 2
 então POTÊNCIA $\leftarrow \text{ÁREA} \times 18$
 senão POTÊNCIA $\leftarrow \text{ÁREA} \times 20$
 fim se
 fim se
fim ref.

Ref. Calcule o número de lâmpadas necessárias ao cômodo
 $\text{LÂMPADAS} \leftarrow \text{QUOCIENTE}(\text{POTÊNCIA}, 60)$
 Calcule o menor inteiro que contém LÂMPADAS
fim ref.

Ref. Calcule o menor inteiro que contém LÂMPADAS
 $\text{AUXILIAR} \leftarrow \text{POTÊNCIA}/60$
 se AUXILIAR \neq LÂMPADAS
 então LÂMPADAS $\leftarrow \text{LÂMPADAS} + 1$
 fim se
fim ref.

Ref. Calcule o total de lâmpadas da residência
 $\text{TOTALÂMPADAS} \leftarrow \text{TOTALÂMPADAS} + \text{LÂMPADAS}$
fim ref.

Ref. Calcule o total de potência da residência
 $\text{TOTALPOTÊNCIA} \leftarrow \text{TOTALPOTÊNCIA} + \text{POTÊNCIA}$
fim ref.

Ref. Escreva CÔMODO, área do cômodo, potência de iluminação, número de lâmpadas necessárias
 escreva CÔMODO, ÁREA, POTÊNCIA, LÂMPADAS
fim ref.

Ref. Escreva o total de lâmpadas e total de potência da residência
 escreva TOTALÂMPADAS, TOTALPOTÊNCIA
fim ref.

Ref. Atribua valores iniciais necessários
 $\text{TOTALÂMPADAS} \leftarrow 0$
 $\text{TOTALPOTÊNCIA} \leftarrow 0$
fim ref.

Ref. Defina tipo das variáveis
declare ÁREA, AUXILIAR, CLASSE, COMPRIMENTO,
 LÂMPADAS, LARGURA, POTÊNCIA, TOTALÂMPADAS,
 TOTALPOTÊNCIA número
declare CÔMODO literal
fim ref.

Finalmente, inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtem-se:

Algoritmo	
	{Definição do tipo de variáveis}
<u>declare</u>	ÁREA, {área do cômodo} AUXILIAR, {variável auxiliar para cálculo} CLASSE, {classe de iluminação} COMPRIMENTO, {comprimento do cômodo} LÂMPADAS, {número de lâmpadas necessárias por cômodos} LARGURA, {largura do cômodo} POTÊNCIA, {potência necessária para cada cômodo} TOTALÂMPADAS, {número de lâmpadas necessárias para a residência} TOTALPOTÊNCIA, {potência necessária para a residência}
<u>declare</u>	CÔMODO {tipo do cômodo} <u>literal</u>
	{Atribuição de valores iniciais necessários}
	TOTALÂMPADAS $\leftarrow 0$ TOTALPOTÊNCIA $\leftarrow 0$
<u>repita</u>	<u>leia</u> CÔMODO, CLASSE, COMPRIMENTO, LARGURA se CÔMODO = "VAZIO" então interrompa fim se
	{Cálculo da área do cômodo}
	ÁREA $\leftarrow \text{COMPRIMENTO} \times \text{LARGURA}$
	{Cálculo da potência de iluminação do cômodo}
se	CLASSE = 1 então POTÊNCIA $\leftarrow \text{ÁREA} \times 15$ senão se CLASSE = 2 então POTÊNCIA $\leftarrow \text{ÁREA} \times 18$ senão POTÊNCIA $\leftarrow \text{ÁREA} \times 20$ fim se
	{Cálculo do número de lâmpadas necessárias ao cômodo}

LÂMPADAS ← QUOCIENTE(POTÊNCIA,60)

AUXILIAR ← POTÊNCIA/60

se AUXILIAR ≠ LÂMPADAS

então LÂMPADAS ← LÂMPADAS + 1

fim se

{Cálculo do total de lâmpadas da residência}

TOTALÂMPADAS ← TOTALÂMPADAS + LÂMPADAS

{Cálculo do total de potência da residência}

TOTALPOTÊNCIA ← TOTALPOTÊNCIA + POTÊNCIA

{Escrita do CÔMODO, área do cômodo, potência de}

{iluminação, número de lâmpadas necessárias}

escreva CÔMODO,ÁREA,POTÊNCIA,LÂMPADAS

fim repita

{Escrita do total de lâmpadas e total de potência da}

{residência}

escreva TOTALÂMPADAS,TOTALPOTÊNCIA

fim algoritmo.

Exemplo 1.42

Fazer um algoritmo que:

- leia e escreva o nome e a altura das moças inscritas em um concurso de beleza. Para cada moça, existe uma linha contendo seu nome e sua altura. A última linha que não corresponde a nenhuma moça, conterá a palavra VAZIO no lugar do nome;
- calcule e escreva as duas maiores alturas e quantas moças as possuem.

Algoritmo

Defina tipo das variáveis

Atribua valores iniciais necessários

repita

leia NOME,ALTURA

se NOME = "VAZIO"

então interrompa

fim se

escreva NOME,ALTURA

Compare ALTURA com as duas maiores alturas

fim repita

Escreva resultados

fim algoritmo.

Ref. Compare ALTURA com as duas maiores alturas

se ALTURA > MAIOR1

então Atualize a 2.^a maior altura e o número de moças que a possuem

Atualize a 1.^a maior altura e o número de moças que a possuem

senão Verifique se ALTURA é igual a MAIOR1

fim se

fim ref.

Ref. Atualize a 2.^a maior altura e o número de moças que a possuem

MAIOR2 ← MAIOR1

CONTADOR2 ← CONTADOR1

fim ref.

Ref. Atualize a 1.^a maior altura e o número de moças que a possuem

MAIOR1 ← ALTURA

CONTADOR1 ← 1

fim ref.

Ref. Verifique se ALTURA é igual a MAIOR1

se ALTURA = MAIOR1

então CONTADOR1 ← CONTADOR1 + 1

senão Compare ALTURA com MAIOR2

fim se

fim ref.

Ref. Compare ALTURA com MAIOR2

se ALTURA > MAIOR2

então MAIOR2 ← ALTURA

CONTADOR2 ← 1

senão se ALTURA = MAIOR2

então CONTADOR2 ← CONTADOR2 + 1

fim se

fim ref.

Ref. Escreva resultados

escreva MAIOR1,CONTADOR1,MAIOR2,CONTADOR2

fim ref.

Ref. Atribua valores iniciais necessários

CONTADOR1 ← 0

CONTADOR2 ← 0

MAIOR1 ← -1

MAIOR2 ← -1

fim ref.

Ref. Defina tipo das variáveis

declare ALTURA,CONTADOR1,CONTADOR2,MAIOR1,MAIOR2

 número

declare NOME literal

fim ref.

Finalmente, inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, tem-se:

Algoritmo

{Definição do tipo das variáveis}

declare ALTURA,CONTADOR1,CONTADOR2,MAIOR1,MAIOR2

 número

declare NOME literal

 {Atribuição de valores iniciais necessários}

CONTADOR1 ← 0

```

CONTADOR2 ← 0
MAIOR1 ← -1
MAIOR2 ← -1
repita
    leia NOME,ALTURA
    se NOME = "VAZIO"
        então interrompa
    fim se
    escreva NOME,ALTURA
        {Comparação da ALTURA com as duas maiores alturas)
    se ALTURA > MAIOR1
        então MAIOR2 ← MAIOR1
        CONTADOR2 ← CONTADOR1
        MAIOR1 ← ALTURA
        CONTADOR1 ← 1
    senão se ALTURA = MAIOR1
        então CONTADOR1 ← CONTADOR1 + 1
    senão se ALTURA > MAIOR2
        então MAIOR2 ← ALTURA
        CONTADOR2 ← 1
    senão se ALTURA = MAIOR2
        então CONTADOR2 ← CONTADOR2 + 1
    fim se
    fim se
fim se
fim repita
escreva MAIOR1,CONTADOR1,MAIOR2,CONTADOR2
fim algoritmo.

```

Exemplo 1.43

Escrever um algoritmo para fazer uma tabela de sen A, com A variando de 0 a 1.6 radiano de décimo em décimo de radiano, usando a série

$$\text{sen } A = A - \frac{A^3}{3!} + \frac{A^5}{5!} - \dots$$

com erro inferior a 0.0001. Imprimir também o número de termos usados.

Em séries alternadas, o valor absoluto do erro cometido com a interrupção da série é inferior ao valor absoluto do primeiro termo abandonado. Daí o seguinte algoritmo:

```

Algoritmo
Declaração das variáveis
escreva "A   SEN A   N"
A ← 0
repita
    se A > 1.6
        então interrompa
    fim se
    Calcule SENA
    escreva A,SENA,N
    A ← A + 0,1
fim repita
fim algoritmo.

```

os dizeres entre aspas.

escreva A,SENA,N é um comando para escrever o valor das variáveis A,SENA e N.

Ref. Calcule SENA

Atribua valores iniciais necessários
repita
 se ABS(T) < 0,0001
 então interrompa
 fim se
 SENA ← SENA + T
 N ← N + 1
 Calcule o próximo T
fim repita
fim ref.

Ref. Calcule o próximo T

$T ← -T \times \frac{A^2}{2 \times N \times (2 \times N + 1)}$
fim ref.

Ref. Atribua valores iniciais necessários

SENA ← 0
T ← A
N ← 0
fim ref.

Ref. Declare as variáveis

declare SENA,A,N,T numérico
fim ref.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

(Declaração das variáveis)
declare A,N,SENA,T numérico
escreva "A SEN A N"
A ← 0
repita
 se A > 1.6
 então interrompa
 fim se
 (Cálculo de SENA)
 SENA ← 0
 T ← A
 N ← 0
 repita
 se ABS(T) < 0,0001
 então interrompa
 fim se
fim ref.

```

SENA ← SENA + T
N ← N + 1
T ← -T ×  $\frac{A^2}{2 \times N \times (2 \times N + 1)}$ 

```

```

fim repita
escreva A, SENA, N
A ← A + 0,1
fim repita
fim algoritmo.

```

Exemplo 1.44

Para cada aluno da disciplina Programação de Computadores deste semestre, será digitada uma linha com:

- a identificação da turma (A,B,...,R, nesta ordem);
- número de matrícula;
- nota final.

Após o último aluno de cada turma, virá uma linha, que não corresponde a nenhum aluno, contendo zero no lugar do número de matrícula. Deseja-se, através de um computador, ler estas linhas e imprimir, para cada turma, a sua identificação, o número de alunos aprovados (nota final ≥ 60), a média das notas e a melhor nota. Após todas as turmas serem processadas, deseja-se imprimir também o total de alunos aprovados, a média geral e a melhor nota na disciplina, neste semestre.

Em problemas deste tipo, sugere-se pensar primeiro em um nível mais geral (disciplina) e depois, gradativamente, em níveis mais particulares (turma, aluno). Em cada nível, deverá haver, quando necessário, uma parte inicial, uma parte intermediária e uma parte final, que terão de ser repetidas ou não. No caso de se repetir, deverá existir uma condição de interrupção.

Algoritmo

```

Declare as variáveis
Atribua valores iniciais às variáveis relativas à disciplina
Execute os cálculos intermediários relativos à disciplina
Execute os cálculos finais relativos à disciplina
fim algoritmo.

```

A nível de disciplina não há repetição, já que só existe uma. Neste exemplo, o desenvolvimento dos refinamentos será iniciado pelo último, especificando-se o que se deseja obter neste nível.

```

Ref. Execute os cálculos finais relativos à disciplina
MEDIADISC ← SOMADISC / NDISC
escreva APROVDISC,MEDIADISC,MELHORDISC
fim ref.

```

As letras DISC no final do nome das variáveis indicarão, neste algoritmo, que elas se referem à disciplina Programação de Computadores. Neste momento, já se sabe quais variáveis devem ter seus valores iniciais atribuídos a nível de disciplina.

```

Ref. Atribua valores iniciais às variáveis relativas à disciplina
SOMADISC ← 0
NDISC ← 0
APROVDISC ← 0
MELHORDISC ← -1
fim ref.

```

na, levando-se em conta ser a disciplina formada de várias turmas, sendo A e R as identificações da primeira e última turmas, respectivamente.

Ref. Execute cálculos intermediários relativos à disciplina

repita

```

Atribua valores iniciais às variáveis relativas a uma turma
Execute cálculos intermediários relativos a uma turma
Execute cálculos finais relativos a uma turma
se TURMA = "R"
então interrompa
fim se
fim repita
fim ref.

```

Novamente, o desenvolvimento dos refinamentos será iniciado pelo último.

Ref. Execute os cálculos finais relativos a uma turma

```

MÉDIA ← SOMA / N
escreva TURMA,APROV,MÉDIA,MELHOR
SOMADISC ← SOMADISC + N
APROVDISC ← APROVDISC + APROV
se MELHOR > MELHORDISC
então MELHORDISC ← MELHOR
fim se
fim ref.

```

Deve-se observar que a nível de turma, e também a níveis inferiores aos de turma, a execução de cálculos finais se refere não só à manipulação dos totais da turma, como também à vinculação destes totais com os valores relativos ao nível imediatamente superior, neste caso, a disciplina.

Ref. Atribua valores iniciais às variáveis relativas a uma turma

```

N ← 0
APROV ← 0
MELHOR ← -1
SOMA ← 0
fim ref.

```

A seguir, desenvolve-se o refinamento relativo aos cálculos intermediários de uma turma, levando-se em conta que uma turma é formada de vários alunos, vindo após o último um aluno com matrícula nula.

Ref. Execute os cálculos intermediários relativos a uma turma

repita

```

leia TURMA,ALUNO,NOTA
se ALUNO = 0
então interrompa
fim se
Execute os cálculos relativos a um aluno
fim repita
fim ref.

```

Ref. Execute os cálculos relativos a um aluno

```
N ← N + 1  
se NOTA ≥ 60  
  então APROV ← APROV + 1  
fim se  
se NOTA > MELHOR  
  então MELHOR ← NOTA  
fim se  
SOMA ← SOMA + NOTA  
fim ref.
```

Agora, falta apenas se definir o tipo das variáveis utilizadas

Ref. Declare as variáveis

```
declare MEDIADISC, {média das notas de todos os alunos da disciplina}  
        SOMADISC, {soma de todas as notas da disciplina}  
        NDISC, {número de alunos matriculados na disciplina}  
        APROVDISC, {número de alunos aprovados na disciplina}  
        MELHORDISC, {melhor nota na disciplina}  
        MÉDIA, {média das notas de todos os alunos de uma turma}  
        SOMA, {soma de todas as notas de uma turma}  
        N, {número de alunos de uma turma}  
        APROV, {número de alunos aprovados em uma turma}  
        MELHOR, {melhor nota de uma turma}  
        ALUNO, {número de matrícula de um aluno}  
        NOTA  
        numérico  
        declare TURMA {identificação de uma turma}  
        literal  
fim ref.
```

Finalmente, inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, tem-se:

Algoritmo

```
{Declaração das variáveis}  
declare MEDIADISC, {média das notas de todos os alunos da disciplina}  
        SOMADISC, {soma de todas as notas da disciplina}  
        NDISC, {número de alunos matriculados na disciplina}  
        APROVDISC, {número de alunos aprovados na disciplina}  
        MELHORDISC, {melhor nota na disciplina}  
        MÉDIA, {média das notas de todos os alunos de uma turma}  
        SOMA, {soma de todas as notas de uma turma}  
        N, {número de alunos de uma turma}  
        APROV, {número de alunos aprovados em uma turma}  
        MELHOR, {melhor nota de uma turma}  
        ALUNO, {número de matrícula de um aluno}  
        NOTA  
        numérico  
        declare TURMA {identificação de uma turma}  
        literal  
(Atribuição de valores iniciais às variáveis relativas à disciplina)  
SOMADISC ← 0  
NDISC ← 0  
APROVDISC ← 0
```

(Execução dos cálculos intermediários relativos à disciplina)

repita

SOMA ← 0

N ← 0

APROV ← 0

MELHOR ← -1

(Execução dos cálculos intermediários relativos a uma turma)

repita

leia TURMA,ALUNO,NOTA

se ALUNO = 0

então interrompa

fim se

(Execução dos cálculos relativos a um aluno)

N ← N + 1

se NOTA ≥ 60

então APROV ← APROV + 1

fim se

se NOTA > MELHOR

então MELHOR ← NOTA

fim se

SOMA ← SOMA + NOTA

fim repita

(Execução dos cálculos finais relativos a uma turma)

MÉDIA ← SOMA/N

escreva TURMA,APROV,MÉDIA,MELHOR

SOMADISC ← SOMADISC + SOMA

NDISC ← NDISC + N

APROVDISC ← APROVDISC + APROV

se MELHOR > MELHORDISC

então MELHORDISC ← MELHOR

fim se

se TURMA = "R"

então interrompa

fim se

fim repita

(Execução dos cálculos relativos à disciplina)

MEDIADISC ← SOMADISC / NDISC

escreva APROVDISC,MEDIADISC,MELHORDISC

fim algoritmo.

1.12. EXERCÍCIOS PROPOSTOS

PROBLEMAS GERAIS

△ 1.12.1. Fazer um algoritmo que:

- Leia um número indeterminado de linhas contendo cada uma a idade de um indivíduo. A última linha, que não entrará nos cálculos, contém o valor da idade igual a zero.
- Calcule e escreva a idade média deste grupo de indivíduos.

△ 1.12.2. Tem-se um conjunto de dados contendo a altura e o sexo (masculino, feminino) de 50 pessoas.

Fazer um algoritmo que calcule e escreva:

- a maior e a menor altura do grupo;
- a média de altura das mulheres;
- o número de homens.

△ 1.12.3. A conversão de graus Farenheit para centígrados é obtida por

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

Fazer um algoritmo que calcule e escreva uma tabela de centígrados em função de graus Farenheit, que variam de 50 a 150 de 1 em 1.

▲ 1.12.4. Um comerciante deseja fazer o levantamento do lucro das mercadorias que ele comercializa. Para isto, mandou digitar uma linha para cada mercadoria com o nome, preço de compra e preço de venda das mesmas. Fazer um algoritmo que:

- determine e escreva quantas mercadorias proporcionam:
 - lucro < 10%
 - 10% ≤ lucro ≤ 20%
 - lucro > 20%
- determine e escreva o valor total de compra e de venda de todas as mercadorias, assim como o lucro total.

Observação: o aluno deve adotar um flag.

△ 1.12.5. Supondo que a população de um país A seja da ordem de 90.000.000 de habitantes com uma taxa anual de crescimento de 3% e que a população de um país B seja, aproximadamente, de 200.000.000 de habitantes com uma taxa anual de crescimento de 1,5%, fazer um algoritmo que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas essas taxas de crescimento.

▲ 1.12.6. Um determinado material radioativo perde metade de sua massa a cada 50 segundos. Dada a massa inicial, em gramas, fazer um algoritmo que determine o tempo necessário para que essa massa se torne menor do que 0,5 grama. Escreva a massa inicial, a massa final e o tempo calculado em horas, minutos e segundos.

▲ 1.12.7. Deseja-se fazer um levantamento a respeito da ausência de alunos à primeira prova de Programação de Computadores para cada uma das 14 turmas existentes. Para cada turma, é fornecido um conjunto de valores, sendo que os dois primeiros valores do conjunto correspondem à identificação da turma (A, ou B, ou C,...) e ao número de alunos matriculados, e os demais valores deste conjunto contêm o número de matrícula do aluno e a letra A ou P para o caso de o aluno estar ausente ou presente, respectivamente. Fazer um algoritmo que:

- para cada turma, calcule a porcentagem de ausência e escreva a identificação da turma e a porcentagem calculada;
- determine e escreva quantas turmas tiveram porcentagem de ausência superior a 5%.

△ 1.12.8. Uma certa firma fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um novo produto lançado no mercado. Para isso, forneceu o sexo do entrevistado e sua resposta (sim ou não). Sabendo-se que foram entrevistadas 2.000 pessoas, fazer um algoritmo que calcule e escreva:

- o número de pessoas que responderam sim;
- o número de pessoas que responderam não;
- a porcentagem de pessoas do sexo feminino que responderam sim;
- a porcentagem de pessoas do sexo masculino que responderam não.

▲ 1.12.9. Foi feita uma pesquisa para determinar o índice de mortalidade infantil em um certo período. Fazer um algoritmo que:

- leia inicialmente o número de crianças nascidas no período;
- leia, em seguida, um número indeterminado de linhas, contendo, cada uma, o sexo de uma criança morta (masculino, feminino) e o número de meses de vida da criança. A última linha, que não entrará nos cálculos, contém no lugar do sexo a palavra "vazio";
- determine e imprima:
 - a) a porcentagem de crianças mortas no período;
 - b) a porcentagem de crianças do sexo masculino mortas no período;
 - c) a porcentagem de crianças que viveram 24 meses ou menos no período.

△ 1.12.10. 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 (4, 5, 7, 12) e o número de pessoas que o estavam assistindo naquela casa. Se a televisão estivesse desligada, nada era anotado. ou seja, esta casa não entra na pesquisa. Fazer um algoritmo que:

- leia um número indeterminado de dados, sendo que o "FLAG" corresponde ao número do canal igual a zero;

- calcule a porcentagem de audiência para cada emissora;
- escreva o número do canal e a sua respectiva porcentagem.

△ 1.12.11. Uma universidade deseja fazer um levantamento a respeito de seu concurso vestibular. Para cada curso, é fornecido o seguinte conjunto de valores:

- o código do curso;
- número de vagas;
- número de candidatos do sexo masculino;
- número de candidatos do sexo feminino.

O último conjunto, para indicar fim de dados, contém o código do curso igual a zero. Fazer um algoritmo que:

- calcule e escreva, para cada curso, o número de candidatos por vaga e a porcentagem de candidatos do sexo feminino (escreva também o código correspondente do curso);
- determine o maior número de candidatos por vaga e escreva esse número juntamente com o código do curso correspondente (supor que não haja empate);
- calcule e escreva o total de candidatos.

▲ 1.12.12. O sistema de avaliação de uma determinada disciplina obedece aos seguintes critérios:

- durante o semestre são dadas três notas;
- a nota final é obtida pela média aritmética das notas dadas durante o curso;
- é considerado aprovado o aluno que obtiver a nota final superior ou igual a 60 e que tiver comparecido a um mínimo de 40 aulas.

Fazer um algoritmo que:

- a) Leia um conjunto de dados contendo o número de matrícula, as três notas e a frequência (número de aulas freqüentadas) de 100 alunos.

b) Calcule:

- a nota final de cada aluno;
- a maior e a menor nota da turma;
- a nota média da turma;
- o total de alunos reprovados;
- a porcentagem de alunos reprovados por infreqüência.

c) Escreva:

- para cada aluno, o número de matrícula, a freqüência, a nota final e o código (aprovado ou reprovado);
- o que foi calculado no item b (2, 3, 4 e 5).

△ 1.12.13. Deseja-se fazer uma pesquisa a respeito do consumo mensal de energia elétrica em uma determinada cidade. Para isso, são fornecidos os seguintes dados:

- preço do kWh consumido;
- número do consumidor;
- quantidade de kWh consumidos durante o mês;
- código do tipo de consumidor (residencial, comercial, industrial).

O número do consumidor igual a zero deve ser usado como flag. Fazer um algoritmo que:

• leia os dados descritos acima;

• calcule:

- a) para cada consumidor, o total a pagar,
- b) o maior consumo verificado,
- c) o menor consumo verificado,
- d) o total do consumo para cada um dos três tipos de consumidores,
- e) a média geral de consumo;

• escreva:

- a) para cada consumidor, o seu número e o total a pagar,
- b) o que foi calculado nos itens b, c, d, e acima especificados.

▲ 1.12.14. Tem-se uma estrada ligando várias cidades. Cada cidade tem seu marco quilométrico. Fazer um algoritmo que:

- leia vários pares de dados, contendo cada par os valores dos marcos quilométricos, em ordem crescente, de duas cidades. O último par contém estes dois valores iguais;
- calcule os tempos decorridos para percorrer a distância entre estas duas cidades, com as seguintes velocidades: 20, 30, 40, 50, 60, 70 e 80 km/hora, sabendo-se que

$$t = \frac{e}{v}, \text{ onde } t = \text{tempo}; e = \text{espaço}; v = \text{velocidade};$$

- escreva os marcos quilométricos, a velocidade e o tempo decorrido entre as duas cidades, apenas quando este tempo for superior a 2 horas.

△ 1.12.15. Os bancos atualizam diariamente as contas de seus clientes. Essa atualização envolve a análise dos depósitos e retiradas de cada conta. Numa conta de saldo mínimo, uma taxa de serviço é deduzida se a conta cai abaixo de uma certa quantia especificada.

Suponha que uma conta particular comece o dia com um saldo de R\$ 60,00. O saldo mínimo exigido é R\$ 30,00 e se o saldo de fim de dia for menor do que isso, uma taxa é reduzida da conta. A fim de que essa atualização fosse feita utilizando computador, é fornecido, para cada conta, o seguinte conjunto de dados:

- a primeira linha contém o número da conta, o valor do saldo atual e do saldo mínimo diário, quantidade de transações e taxa de serviço;
- as linhas seguintes contêm o valor e o código da transação (depósito ou retirada).

Escrever um algoritmo que:

- calcule o saldo (saldo/débito) da conta ao fim do dia (se o resultado for negativo, isto significa insuficiência de fundos na conta);
- escreva, para cada conta, o seu número e o saldo calculado. Se não houver fundos, imprima o número da conta e a mensagem "NÃO HÁ FUNDOS";
- utilize como flag o número da conta igual a zero.

△ 1.12.16. Uma empresa decidiu fazer um levantamento em relação aos candidatos que se apresentarem para preenchimento de vagas no seu quadro de funcionários, utilizando processamento eletrônico. Sendo que você seja o programador encarregado desse levantamento, fazer um algoritmo que:

- leia um conjunto de dados para cada candidato contendo:
 - número de inscrição do candidato,
 - idade,
 - sexo (masculino, feminino),
 - experiência no serviço (sim ou não).

O último conjunto contém o número de inscrição do candidato igual a zero.

- calcule:
 - o número de candidatos do sexo feminino,
 - o número de candidatos do sexo masculino,
 - idade média dos homens que já têm experiência no serviço,
 - porcentagem dos homens com mais de 45 anos entre o total de homens,
 - número de mulheres que têm idade inferior a 35 anos e com experiência no serviço;
 - a menor idade entre mulheres que já têm experiência no serviço;

- escreva:
 - o número de inscrição das mulheres pertencentes ao grupo descrito no item e,
 - o que foi calculado em cada item acima especificado.

△ 1.12.17. Uma companhia de teatro planeja dar uma série de espetáculos. A direção calcula que, a R\$ 5,00 o ingresso, serão vendidos 120 ingressos, e as despesas montarão em R\$ 200,00. A uma diminuição de R\$ 0,50 no preço dos ingressos espera-se que haja um aumento de 26 ingressos vendidos.

Fazer um algoritmo que escreva uma tabela de valores do lucro esperado em função do preço do ingresso, fazendo-se variar este preço de R\$ 5,00 a R\$ 1,00 de R\$ 0,50 em R\$ 0,50. Escreva, ainda, o lucro máximo esperado, o preço e o número de ingressos correspondentes.

△ 1.12.18. A comissão organizadora de um rallye automobilístico dedicou apurar os resultados da competição através de um processamento eletrônico.

Um dos algoritmos necessários para a classificação das equipes concorrentes é o que emite uma lista geral do desempenho das equipes, atribuindo pontos segundo determinadas normas:

O algoritmo deverá:

- Ler:
 - uma linha contendo os tempos-padrão (em minutos decimais) para as três fases de competição;
 - um conjunto de linhas contendo cada uma o número de inscrição da equipe e os tempos (em minutos decimais) que as mesmas despendem ao cumprir as três diferentes etapas. A última linha (flag), que não entrará nos cálculos, contém o número 9999 como número de inscrição.
- Calcular:
 - os pontos de cada equipe em cada uma das etapas, segundo o seguinte critério:

— valor absoluto da diferença entre o tempo-padrão (lido na primeira linha) e o tempo despendido pela equipe numa etapa:

- | | |
|--------------------------------|--|
| $\Delta < 3$ minutos | — atribuir 100 pontos à etapa |
| $3 \leq \Delta \leq 5$ minutos | — atribuir 80 pontos à etapa |
| $\Delta > 5$ minutos | — atribuir $80 - \frac{\Delta - 5}{5}$ pontos à etapa; |

b.2) o total de pontos de cada equipe nas três etapas;

b.3) a equipe vencedora.

c) Escrever:

- para cada equipe, o número de inscrição, os pontos obtidos em cada etapa e o total de pontos obtidos.

△ 1.12.19. Numa certa loja de eletrodomésticos, o comerciário encarregado da seção de televisores recebe, mensalmente, um salário fixo mais comissão. Essa comissão é calculada em relação ao tipo e ao número de televisores vendidos por mês, obedecendo à tabela abaixo:

TIPO	N.º DE TELEVISORES VENDIDOS	COMISSÕES
a cores	maior ou igual a 10 menor do que 10	R\$ 50,00 por televisor vendido R\$ 5,00 por televisor vendido
preto e branco	maior ou igual a 20 menor do que 20	R\$ 20,00 por televisor vendido R\$ 2,00 por televisor vendido

Sabe-se, ainda, que ele tem um desconto de 8% sobre seu salário fixo para o INSS. Se o seu salário total (fixo + comissões - INSS) for maior ou igual a R\$ 500,00 ele ainda terá um desconto de 15%, sobre esse salário total, relativo ao imposto de renda retido na fonte. Sabendo-se que existem 20 empregados nesta seção, leia o valor do salário fixo e, para cada comerciário, o número de sua inscrição, o número de televisores a cores e o número de televisores preto e branco vendidos; calcule e escreva o número de inscrição de cada empregado, seu salário bruto e seu salário líquido.

△ 1.12.20. O dia da semana para uma data qualquer pode ser calculado pela seguinte fórmula:

$$\text{Dia da semana} = \text{RESTO}(\text{TRUNCA}(2,6 \times M - 0,1) + D - A + \text{QUOCIENTE}(A, 4) - \text{QUOCIENTE}(S, 4) - 2 \times S), 7)$$

onde:

M — representa o número do mês. Janeiro e fevereiro são os meses 11 e 12 do ano precedente. Março é o mês 1 e dezembro é o mês 10;

D — representa o dia do mês;

A — representa o número formado pelos dois últimos algarismos do ano;

S — representa o número formado pelos dois primeiros algarismos do ano.

Os dias da semana são numerados de zero a seis: domingo corresponde a 0, segunda a 1, e assim por diante.

Fazer um algoritmo que:

- leia um conjunto de 50 datas (dia, mês, ano);
- determine o dia da semana correspondente à data lida, segundo o método especificado;
- escreva, para cada data lida, o dia, mês, ano e o dia da semana calculado.

△ 1.12.21. Numa fábrica trabalham homens e mulheres divididos em três classes:

A — os que fazem até 30 peças por mês;

B — os que fazem de 31 a 35 peças por mês;

C — os que fazem mais de 35 peças por mês.

A classe A recebe salário-mínimo. A classe B recebe salário-mínimo e mais 3% do salário-mínimo por peça, acima das 30 iniciais. A classe C recebe salário-mínimo e mais 5% do salário-mínimo por peça acima das 30 iniciais.

Fazer um algoritmo que:

a) leia várias linhas, contendo cada uma:

- o número do operário,
- o número de peças fabricadas por mês,
- o sexo do operário;

b) calcule e escreva:

- o salário de cada operário,
- o total da folha mensal de pagamento da fábrica,
- o número total de peças fabricadas por mês,
- a média de peças fabricadas pelos homens em cada classe,
- a média de peças fabricadas pelas mulheres em cada classe,
- o número do operário ou operária de maior salário (não existe empate).

Observação: A última linha, que servirá de flag, terá o número do operário igual a zero.

△ 1.12.22. Uma determinada fábrica de rádios possui duas linhas de montagem distintas: *standard* e luxo. A linha de montagem *standard* comporta um máximo de 24 operários; cada rádio *standard* dá um lucro de X reais e gasta um homem-dia para sua confecção. A linha de montagem luxo comporta no máximo 32 operários; cada rádio luxo dá um lucro de Y reais e gasta 2 homens-dia para sua confecção. A fábrica possui 40 operários. O mercado é capaz de absorver toda a produção e o fabricante deseja saber qual esqueleto de produção a adotar de modo a maximizar seu lucro diário.

Fazer um algoritmo que leia os valores de X e Y e escreva, para esse esquema de lucro máximo, o número de operários na linha *standard* e na linha luxo, o número de rádios *standard* e luxo produzidos e o lucro.

▲ 1.12.23. Fazer um algoritmo para calcular o número de dias decorridos entre duas datas (considerar também a ocorrência de anos bissextos), sabendo-se que:

- cada par de datas é lido numa linha, a última linha contém o número do dia negativo;
- a primeira data na linha é sempre a mais antiga;
- o ano está digitado com quatro dígitos;
- um ano será bissexto se for divisível por 400, ou se for divisível por 4 e não o for por 100.

PROBLEMAS ENVOLVENDO CÁLCULO DE SOMATÓRIOS

△ 1.12.24. Fazer um algoritmo que calcule e escreva o valor de S:

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \dots + \frac{99}{50}$$

△ 1.12.25. Fazer um algoritmo que calcule e escreva a seguinte soma:

$$\frac{2^1}{50} + \frac{2^2}{49} + \frac{2^3}{48} + \dots + \frac{2^{50}}{1}$$

▲ 1.12.26. Fazer um algoritmo para calcular e escrever a seguinte soma:

$$S = \frac{37 \times 38}{1} + \frac{36 \times 37}{2} + \frac{35 \times 36}{3} + \dots + \frac{1 \times 2}{37}$$

△ 1.12.27. Fazer um algoritmo que calcule e escreva o valor de S onde:

$$S = \frac{1}{1} - \frac{2}{4} + \frac{3}{9} - \frac{4}{16} + \frac{5}{25} - \frac{6}{36} \dots - \frac{10}{100}$$

△ 1.12.28. Fazer um algoritmo que calcule e escreva a soma dos 50 primeiros termos da seguinte série:

$$\frac{1000}{1} - \frac{997}{2} + \frac{994}{3} - \frac{991}{4} + \dots$$

▲ 1.12.29. Fazer um algoritmo que calcule e escreva a soma dos 30 primeiros termos da série:

$$\frac{480}{10} - \frac{475}{11} + \frac{470}{12} - \frac{465}{13} + \dots$$

△ 1.12.30. Escrever um algoritmo para gerar e escrever uma tabela com os valores do seno de um ângulo A em radianos, utilizando a série de Mac-Laurin truncada, apresentada a seguir:

$$\sin A = A - \frac{A^3}{6} + \frac{A^5}{120} - \frac{A^7}{5040}$$

Condições: os valores dos ângulos A devem variar de 0.0 a 6.3, inclusive, de 0.1 em 0.1.

▲ 1.12.31. Fazer um algoritmo para calcular e escrever o valor do número π , com precisão de 0,0001, usando a série:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Para obter a precisão desejada, adicionar apenas os termos cujo valor absoluto seja maior ou igual a 0,0001. △ 1.12.32. O valor aproximado de π pode ser calculado usando-se a série

$$S = \frac{1}{1^3} - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \frac{1}{9^3} \dots$$

sendo $\pi = \sqrt[4]{S \times 32}$

Fazer um algoritmo para calcular e escrever o valor de π com 51 termos.

△ 1.12.33. Fazer um algoritmo que:

- leia o valor de X de uma unidade de entrada;
- calcule e escreva o valor do seguinte somatório:

$$\frac{X^{25}}{1} - \frac{X^{24}}{2} + \frac{X^{23}}{3} - \frac{X^{22}}{4} + \dots + \frac{X}{25}$$

△ 1.12.34. Fazer um algoritmo que calcule e escreva o valor de S no seguinte somatório:

$$S = \frac{1}{225} - \frac{2}{196} + \frac{4}{169} - \frac{8}{144} + \dots + \frac{16384}{1}$$

△ 1.12.35. Fazer um algoritmo que calcule e escreva a soma dos 20 primeiros termos da série:

$$\frac{100}{0!} + \frac{99}{1!} + \frac{98}{2!} + \frac{97}{3!} + \dots$$

▲ 1.12.36. Elaborar um algoritmo que:

- calcule e escreva o valor da série abaixo com precisão menor que um décimo de milionésimo (0.0000001);
- indique quantos termos foram usados.

$$S = 63 + \frac{61}{1!} + \frac{59}{2!} + \frac{57}{3!} + \dots$$

△ 1.12.37. Fazer um algoritmo que calcule e escreva a soma dos 50 primeiros termos da série:

$$\frac{1!}{1} - \frac{2!}{3} + \frac{3!}{7} - \frac{4!}{15} + \frac{5!}{31} - \dots$$

△ 1.12.38. Fazer um algoritmo que calcule o valor de e^x através da série:

$$e^x = x^0 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

de modo que o mesmo difira do valor calculado através da função EXP de, no máximo, 0,0001. O valor de x deve ser lido de uma unidade de entrada. O algoritmo deverá escrever o valor de x, o valor calculado através da série, o valor dado pela função EXP e o número de termos utilizados da série.

▲ 1.12.39. Fazer um algoritmo para determinar e escrever o valor do seguinte somatório:

$$S = X - \frac{X^2}{3!} + \frac{X^4}{5!} - \frac{X^6}{7!} + \dots$$

usando os 20 primeiros termos do somatório. O valor de X é lido de uma unidade de entrada.

△ 1.12.40. Fazer um algoritmo que:

a) calcule o valor do co-seno de x através de 20 termos da série seguinte:

$$\text{co-seno}(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

b) calcule a diferença entre o valor calculado no item a e o valor fornecido pela função $\text{COS}(X)$.

c) imprima o que foi calculado nos itens a e b.

Observação: o valor de x é fornecido como entrada.

PROBLEMAS DE APLICAÇÃO EM CIÊNCIAS EXATAS

△ 1.12.41. Escrever um algoritmo que:

- leia várias linhas, cada uma delas contendo um valor a ser armazenado em X.
- para cada valor X lido, calcule o valor de Y dado pela fórmula:

$$Y = 2,5 * \cos |X/2|$$

- escreva os valores de X e Y.

Observação: A última linha de dados, cujo conteúdo não será processado, deverá conter um valor negativo. Use esta condição para testar o fim do processamento.

△ 1.12.42. Sejam P(x_1, y_1) e Q(x_2, y_2) dois pontos quaisquer do plano. A sua distância é dada por

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Escrever então um algoritmo que, lendo várias linhas onde cada uma contém as coordenadas dos dois pontos, escreva para cada par de pontos lidos a sua distância. A última linha contém as coordenadas x_1, x_2, y_1, y_2 iguais a zero.

△ 1.12.43. A solução x, y para o sistema de equações lineares abaixo:

$$ax + by = u$$

$$cx + dy = v$$

é dada por:

$$x = \frac{d}{ad - bc} u - \frac{b}{ad - bc} v \quad y = \frac{-c}{ad - bc} u + \frac{a}{ad - bc} v$$

Escrever um algoritmo que:

- leia várias linhas, onde cada uma contém os parâmetros a, b, c, d, u, v do sistema (a última linha contém os valores de a, b, c, d iguais a zero);
- calcule a solução x, y de cada sistema dado por seus parâmetros;
- escreva os parâmetros lidos e os valores calculados.

△ 1.12.44. Fazer um algoritmo que, tendo em uma unidade de entrada os parâmetros A e B de uma reta no plano dado pela equação $Y = AX + B$, determine a área do triângulo formado por esta reta e os eixos coordenados.

O algoritmo lerá um número indeterminado de linhas, cada linha contendo um par de parâmetros (A, B), e para cada par lido deverá escrever: os parâmetros A e B e a área do triângulo.

A execução do algoritmo deverá terminar quando ler uma linha com um par de zeros.

Observação: Se, em uma linha (à exceção da última), um dos parâmetros for igual a zero, não haverá triângulo — assim, o programa deverá imprimir A, B, e 0 (zero).

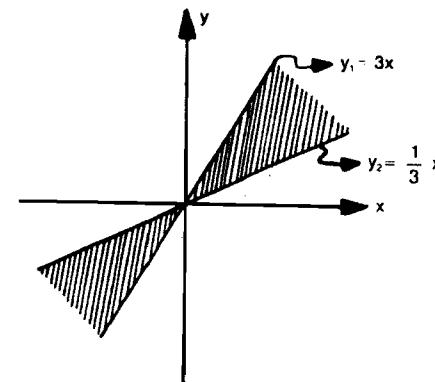
▲ 1.12.45. Fazer um algoritmo para tabular a função $y = f(x) + g(x)$, para $x = 1, 2, 3, \dots, 10$ onde:

$$h(x) = x^2 - 16$$

$$f(x) = \begin{cases} h(x), & \text{se } h(x) \geq 0 \\ 1, & \text{se } h(x) < 0 \end{cases}$$

$$g(x) = \begin{cases} x^2 + 16, & \text{se } f(x) = 0 \\ 0, & \text{se } f(x) > 0 \end{cases}$$

△ 1.12.46. As coordenadas de um ponto (x, y) estão disponíveis em uma unidade de entrada. Ler esses valores (até quando um flag ocorrer) e escrever "INTERIOR" se o ponto estiver dentro da região hachurada mostrada abaixo ($y_2 < |y| < y_1$) caso contrário, escrever "EXTERIOR".



△ 1.12.47. Fazer um algoritmo para calcular e escrever a soma dos cubos dos números pares compreendidos entre B e A. Suponha que os valores de B e A ($B > A$) são dados em uma linha.

△ 1.12.48. Fazer um algoritmo que calcule o volume de uma esfera em função do raio R. O raio deverá variar de 0 a 20 cm de 0,5 em 0,5 cm.

$$V = \frac{4}{3}\pi R^3$$

△ 1.12.49. Fazer um algoritmo para calcular e escrever a área de um polígono regular de N lados inscrito numa circunferência de raio R. O número de polígonos será fornecido na primeira linha de dados e nas linhas seguintes serão fornecidos os valores de N e R.

▲ 1.12.50. Para um polígono regular inscrito numa circunferência, quanto maior o número de lados do polígono, mais seu perímetro se aproxima do comprimento da circunferência. Se o número de lados for muito grande e o raio da circunferência for unitário, o semiperímetro do polígono terá um valor muito próximo de π .

Fazer um algoritmo que escreva uma tabela do semiperímetro em função do número de lados, para polígonos regulares inscritos, numa circunferência de raio unitário. O número de lados deverá variar de 5 a 100 de 5 em 5.

△ 1.12.51. Construir uma tabela de perda de carga em tubulações para vazões que variem de 0,1 l/s a 10,0 l/s, de 0,1 em 0,1, através da fórmula de Hanzen-Willians dada abaixo:

$$J = Q^{1.85} \times 10,643 \times D^{-1.87} \times C^{-1.85}$$

onde:

J = perda de carga (m/1000m);

Q = vazão (m³/s);

D = diâmetro de tubo (m²);

C = coeficiente de rugosidade.

Os valores de D e C serão lidos de uma unidade de entrada. Considerar como flag o valor D = 0.

△ 1.12.52. Fazer um algoritmo que calcule e escreva o número de grãos de milho que se pode colocar num tabuleiro de xadrez, colocando 1 no primeiro quadro e nos quadros seguintes o dobro do quadro anterior.

1	2	4	8	

São 64 quadros

$$\text{Fórmula: } \sum_{n=1}^{63} 2^n$$

△ 1.12.53. Um certo aço é classificado de acordo com o resultado de três testes, que devem verificar se o mesmo satisfaz às seguintes especificações:

Teste 1 — conteúdo de carbono abaixo de 7%;

Teste 2 — dureza Rokwell maior que 50;

Teste 3 — resistência à tração maior do que 80.000 psi.

Ao aço é atribuído o grau 10, se passa pelos três testes; 9, se passa apenas nos testes 1 e 2; 8, se passa nos testes 1 e 7, se não passou nos três testes. Supondo que sejam lidos de uma unidade de entrada: número de amostra, conteúdo de carbono (em %), a dureza Rokwell e a resistência à tração (em psi) — fazer um algoritmo que dê a classificação de 112 amostras de aço que foram testadas, escrevendo o número da amostra e o grau obtido.

△ 1.12.54. Fazer um algoritmo para calcular a raiz quadrada de um número positivo, usando o roteiro abaixo, baseado no método de aproximações sucessivas de Newton:

Seja Y o número:

$$\text{• a primeira aproximação para a raiz quadrada de } Y \text{ é } X_1 = \frac{Y}{2};$$

$$\text{• as sucessivas aproximações serão: } X_{n+1} = \frac{X_n^2 + Y}{2X_n}$$

O algoritmo deverá prever 20 aproximações.

△ 1.12.55. Dada a equação $x^3 - 3x^2 + 1 = 0$, pode-se encontrar qualquer uma de suas raízes reais através de aproximações sucessivas utilizando a seguinte fórmula:

$$X_{n+1} = X_n - \frac{X_n^3 - 3X_n^2 + 1}{3X_n^2 - 6X_n}$$

Fazer um algoritmo que:

- considere como primeira aproximação $X = 1,5$;
- calcule e escreva a trigésima aproximação da raiz.

△ 1.12.56. Fazer um algoritmo que tabule a seguinte função:

$$f(x, y) = \frac{x^2 + 3x + y^2}{xy - 5y - 3x + 15}$$

para $x = 1, 4, 9, 16, \dots, 100$;

e $y = 0, 1, 2, \dots, 5$ para cada valor de x .

△ 1.12.57. Tem-se 10 conjuntos de valores, onde cada conjunto é formado pelo número de um aluno, a nota provisória do seu trabalho prático e a data em que foi entregue.

Fazer um algoritmo para:

a) Calcular e imprimir a nota final de cada aluno, sabendo-se que os trabalhos entregues:

- até 20/04, nota final = nota provisória + 10 pontos;
- até 02/05, nota final = nota provisória;
- até 30/05, nota final = nota provisória/2;
- até 30/06, nota final = 0.

b) Calcular a média e o desvio padrão das notas provisória e final.

$$\text{Observação: Desvio padrão} = \sqrt{\frac{1}{N-1} \left[\sum_{i=1}^N X_i^2 - \frac{1}{N} \left(\sum_{i=1}^N X_i \right)^2 \right]}$$

△ 1.12.58. Números complexos podem ser escritos na forma cartesiana $Z = x + iy$ ou na forma exponencial $Z = r e^{i\theta}$. Multiplicações e divisões de números complexos na forma exponencial ficam muito mais fáceis de serem feitas, pois assumem a seguinte forma:

$$Z_1 \times Z_2 = r_1 e^{i\theta_1} \times r_2 e^{i\theta_2} = (r_1 \times r_2) e^{i(\theta_1 + \theta_2)}$$

$$\frac{Z_1}{Z_2} = \frac{r_1 e^{i\theta_1}}{r_2 e^{i\theta_2}} = \frac{r_1}{r_2} e^{i(\theta_1 - \theta_2)}$$

Bastando, portanto, operar os módulos (r_1 e r_2) e os argumentos (θ_1 e θ_2).

Razer um algoritmo que leia um conjunto de linhas, cada uma contendo um código e quatro valores. Código MULTIPLICA indica que se quer operar a multiplicação dos dois números complexos representados pelos quatro valores ($r_1, \theta_1, r_2, \theta_2$). Código DIVIDE indica que a operação desejada é a divisão. E código VAZIO vai indicar fim de dados. Para cada operação completada, escrever todos os valores lidos e os valores obtidos.

△ 1.12.59. O cálculo do valor de uma integral definida, usando o método da aproximação por trapézios, é feito dividindo o intervalo de integração em n partes iguais e aproximando a função, em cada subintervalo obtido, por um segmento de reta. O valor da integral é calculado, então, como a soma das áreas dos diversos trapézios formados.

$$A = \frac{y_i + y_{i+1}}{2} \cdot h, \text{ área de cada trapézio}$$

$$h = x_{i+1} - x_i = \frac{b - a}{n} = \text{constante}$$

Fazer um algoritmo para determinar e escrever o valor de π , o qual pode ser calculado pela integral:

$$\pi = 4 \int_0^1 \frac{1}{1+x^2} dx$$

△ 1.12.60. Fazer um algoritmo que:

- leia um conjunto de 25 linhas, contendo, cada uma três números inteiros positivos (em qualquer ordem);
- calcule o máximo divisor comum entre os três números lidos, utilizando o método das divisões sucessivas;
- escreva os três números lidos e o m.d.c. entre eles.

△ 1.12.61. O número 3025 possui a seguinte característica:

$$\begin{cases} 30 + 25 = 55 \\ 55^2 = 3025 \end{cases}$$

Fazer um algoritmo para um programa que pesquise e imprima todos os números de quatro algoritmos que apresentam tal característica.

△ 1.12.62. Dada uma equação diferencial $y = f(x, y)$ e a condição inicial $y(x_0) = y_0$, pode-se encontrar uma solução aproximada desta equação, usando o seguinte método:

$$\begin{aligned} y_1 &= y_0 + hf(x_0, y_0) \\ y_2 &= y_1 + hf(x_1, y_1) \end{aligned}$$

$$y_{k+1} = y_k + hf(x_k, y_k)$$

onde h é um acréscimo que se dá aos valores de x .

$$h = \frac{x_n - x_0}{n}$$

x_n limite superior do intervalo;

x_0 limite inferior do intervalo;

n número de subintervalos.

Fazer, portanto, um algoritmo que encontre e escreva as soluções aproximadas da equação $y' = xy$ com $y(0) = 1$ no intervalo fechado de 0 a 1, com $n = 10$ subintervalos.

△ 1.12.63. Fazer um algoritmo que:

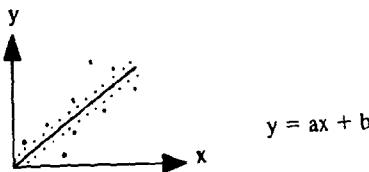
- calcule o número de divisores dos números compreendidos entre 300 e 400.
- escreva cada número e o número de divisores correspondentes.

△ 1.12.64. Fazer um algoritmo que, dados 100 números inteiros positivos, calcule e imprima os que são números perfeitos.

Nota: Número perfeito é aquele cuja soma de seus divisores, exceto ele próprio, é igual ao número.

Exemplo: 6 é perfeito porque $1 + 2 + 3 = 6$.

△ 1.12.65. Regressão linear é uma técnica estatística que ajusta uma equação linear (da forma $y = ax + b$) a um conjunto de pontos dados. O problema consiste em achar uma equação linear que melhor se ajuste aos pontos dados. Um dos métodos empregados é o dos mínimos quadrados, que consiste em minimizar a soma dos quadrados dos desvios verticais dos pontos para a linha reta.



As fórmulas para os coeficientes a e b , dado um conjunto de n pares de pontos (x, y) , são:

$$a = \frac{n\sum xy - \sum x \cdot \sum y}{n \sum x^2 - (\sum x)^2}$$

$$b = \frac{\sum y \cdot \sum x^2 - \sum x \cdot \sum xy}{n \sum x^2 - (\sum x)^2}$$

sendo: $\sum x = \sum_{i=1}^n x_i$, $\sum xy = \sum_{i=1}^n x_i y_i$

$$\sum y = \sum_{i=1}^n y_i$$

Uma vez achada a equação da reta, é importante determinar a precisão de ajustamento dessa linha nos dados reais. Uma medida disso é o coeficiente de correlação R , dado pela fórmula

$$R = \frac{n \sum xy - \sum x \cdot \sum y}{\sqrt{n \sum x^2 - (\sum x)^2} \cdot \sqrt{n \sum y^2 - (\sum y)^2}}$$

O intervalo de variação de R é de $-1 \leq R \leq 1$. Quanto mais próximo de 1 ou -1 ficar o valor de R , melhor terá sido o ajustamento da reta.

Fazer um algoritmo para ler e imprimir um conjunto de pares de pontos (x, y) e calcular e escrever os valores de a , b e R .

△ 1.12.66. Capicuas são números que têm o mesmo valor, se lidos da esquerda para a direita ou da direita para a esquerda. Ex.: 44, 232 etc.

Fazer um algoritmo que determine e escreva todos os números inteiros menores que 10.000 que são quadrados perfeitos e capicuas ao mesmo tempo.

△ 1.12.67. Número primo é aquele que só é divisível por ele mesmo e pela unidade.

Fazer um algoritmo que determine e escreva os números primos compreendidos entre 5.000 e 7.000.

△ 1.12.68. Fazer um algoritmo que:

- leia um conjunto de linhas contendo, cada uma, um número inteiro, na base 10, de até cinco dígitos. A última linha contém o valor zero;
- transforme esse número da base 10 para a base 2;
- escreva o número na base 10 e na base 2.

△ 1.12.69. Fazer um algoritmo que:

- leia um conjunto de linhas contendo, cada uma, um número inteiro na base 3. A última linha contém o valor zero;
- transforme esse número na base 3 para a base 10;
- escreva o número na base 3 e na base 10.

Capítulo 2

Estruturas de Dados

2.1. INTRODUÇÃO

Como já foi visto, variáveis correspondem a posições de memória, às quais o programador tem acesso, através de um algoritmo, visando atingir resultados propostos. Uma variável passa a existir a partir de sua declaração, quando, então, lhe são associados um nome ou identificador e a respectiva posição de memória por ela representada. Qualquer referência ao seu identificador significa o acesso ao conteúdo de uma única posição de memória.

Este capítulo tratará da forma de se ter acesso a conjuntos de dados agrupados segundo o algoritmo adotado para a solução de um determinado problema.

□ Exemplo 2.1

Imaginando que se deseja calcular a média das notas de 10 alunos de uma disciplina e determinar o número de alunos que tiveram nota superior à média calculada, pode-se desenvolver o seguinte algoritmo:

Algoritmo

Defina tipo de variáveis

Atribua valores iniciais necessários

repita

se $I > 10$

então interrompa

fim se

leia NOTA

Acumule NOTA

Conte as notas lidas

fim repita

Calcule a média das notas

Conte os alunos com nota superior à média

Escreva a média e o número de alunos com nota superior à média

fim algoritmo.

Ref. Acumule NOTA

SOMA \leftarrow SOMA + NOTA

fim ref.

Ref. Conte as notas lidas

```
I ← I + 1  
fim ref.
```

Ref. Calcule a média das notas

```
MÉDIA ← SOMA/10  
fim ref.
```

Ref. Atribua valores iniciais necessários

```
I ← 0  
SOMA ← 0  
QUANTIDADE ← 0  
fim ref.
```

Ref. Defina tipo das variáveis

```
declare I, MÉDIA, NOTA, QUANTIDADE, SOMA numérico  
fim ref.
```

Para se proceder à solução do refinamento “Conte os alunos com nota superior à média”, faz-se necessária a comparação de cada uma das 10 notas com o conteúdo de MÉDIA. Mas surge um problema, o conjunto de notas utilizado para o cálculo da média estava disponível na unidade de entrada, e uma vez lidos os dados, não se tem mais acesso a todos eles, pois só se consegue manipular a última nota lida que se encontra armazenada na variável NOTA.

Com os recursos do capítulo anterior, é possível solucionar o impasse de duas maneiras. Uma delas seria a divisão do algoritmo em dois, ou seja, um lerá as notas e calcularia a média, o outro lerá a média e, novamente, todas as notas para as comparações pudessem ser feitas. Os dois algoritmos ficariam assim:

Algoritmo

{Definição do tipo das variáveis}

```
declare I, MÉDIA, NOTA, SOMA numérico  
{Atribuição de valores iniciais necessários}
```

```
I ← 0  
SOMA ← 0
```

repita

```
se I > 10  
então interrompa
```

fim se

leia NOTA

{Acumulação de nota}

```
SOMA ← SOMA + NOTA
```

{Contagem de notas lidas}

```
I ← I + 1
```

fim repita

{Cálculo da média}

```
MÉDIA ← SOMA/10
```

{Escrita da média}

escreva MÉDIA

fim algoritmo.

Algoritmo

{Definição do tipo das variáveis}

```
declare I, MÉDIA, NOTA, QUANTIDADE numérico  
{Atribuição de valores iniciais necessários}
```

I ← 0

QUANTIDADE ← 0

leia MÉDIA

repita

```
se I > 10
```

então interrompa

fim se

leia NOTA

{Contagem dos alunos com nota superior à média}

```
se NOTA > MÉDIA
```

então QUANTIDADE ← QUANTIDADE + 1

fim se

I ← I + 1

fim repita

escreva QUANTIDADE

fim algoritmo.

Os dois programas a serem criados a partir desses dois algoritmos podem ocasionar situações pouco práticas em problemas mais complexos.

A outra forma de resolver o problema, com as ferramentas disponíveis até o momento, seria declarar 10 variáveis para o armazenamento das notas, ter um comando de leitura, um de acumulação e outros 10 de comparação. O algoritmo, segundo este enfoque, fica assim:

Algoritmo

```
declare MÉDIA, NOTA1, NOTA2, NOTA3, NOTA4, NOTA5, NOTA6,  
NOTA7, NOTA8, NOTA9, NOTA10, QUANTIDADE, SOMA  
numérico
```

QUANTIDADE ← 0

```
leia NOTA1, NOTA2, NOTA3, NOTA4, NOTA5, NOTA6, NOTA7,  
NOTA8, NOTA9, NOTA10
```

```
SOMA ← NOTA1 + NOTA2 + NOTA3 + NOTA4 + NOTA5 + NOTA6  
+ NOTA7 + NOTA8 + NOTA9 + NOTA10
```

MÉDIA ← SOMA/10

```
se NOTA1 > MÉDIA
```

então QUANTIDADE ← QUANTIDADE + 1

fim se

```
se NOTA2 > MÉDIA
```

então QUANTIDADE ← QUANTIDADE + 1

fim se

```
se NOTA3 > MÉDIA
```

então QUANTIDADE ← QUANTIDADE + 1

fim se

```
se NOTA4 > MÉDIA
```

então QUANTIDADE ← QUANTIDADE + 1

fim se

```
se NOTA5 > MÉDIA
```

então QUANTIDADE ← QUANTIDADE + 1

fim se

```

se NOTA6 > MÉDIA
  então QUANTIDADE ← QUANTIDADE + 1
fim se

se NOTA7 > MÉDIA
  então QUANTIDADE ← QUANTIDADE + 1
fim se

se NOTA8 > MÉDIA
  então QUANTIDADE ← QUANTIDADE + 1
fim se

se NOTA9 > MÉDIA
  então QUANTIDADE ← QUANTIDADE + 1
fim se

se NOTA10 > MÉDIA
  então QUANTIDADE ← QUANTIDADE + 1
fim se

escreva MÉDIA, QUANTIDADE
fim algoritmo.

```

Novamente, tem-se uma solução bem pouco prática, em virtude do tamanho do texto que seria obtido. Se, ao invés de 10 notas, houvesse 100 ou 1.000 notas, a elaboração deste algoritmo seria impraticável.

A introdução do conceito de variáveis compostas visa facilitar a manipulação de conjunto de dados de mesma natureza, na memória do computador. Embora aumente o grau de abstração dos algoritmos, representa um grande passo no desenvolvimento das técnicas para confecção de algoritmos.

2.2. VARIÁVEIS COMPOSTAS HOMOGENEAS

Variáveis compostas homogêneas correspondem a posições de memória, identificadas por um mesmo nome, individualizadas por índices e cujo conteúdo é de mesmo tipo.

O conjunto de 10 notas dos alunos de uma disciplina pode constituir uma variável composta. A este conjunto associa-se o identificador NOTA que passará a identificar não uma única posição de memória, mas 10.

A referência ao conteúdo do n -ésimo elemento do conjunto será indicada pela notação NOTA[n], onde n é um número inteiro ou uma variável numérica contendo um valor inteiro.

Exemplo 2.2

Supondo-se que em um dado instante a variável composta NOTA contivesse os seguintes valores.

NOTA

60	70	90	60	55	91	100	47	74	86
1	2	3	4	5	6	7	8	9	10

NOTA[3] estaria referenciando o terceiro elemento do conjunto cujo conteúdo é 90.

É possível, também, a utilização de uma forma de acesso genérico, através de uma variável que contenha um valor inteiro.

Exemplo 2.3

No mesmo conjunto NOTA, esboçado no exemplo 2.2, utilizando-se da variável I, tem-se a possibilidade de acesso a qualquer uma das notas armazenadas.

I
6

A uma referência à NOTA[I], antes de ser consultada a variável composta NOTA, I seria substituído pelo seu conteúdo no dado instante. Neste caso, está-se referenciando ao elemento do conjunto que tem o índice 6 associado, ou seja, a nota 91.

Exemplo 2.4

Escrever o trecho do algoritmo que faça a leitura de 10 notas dos alunos de uma disciplina e armazene-as numa variável composta NOTA:

```

Algoritmo
.
.
I ← 1
repita
  se I > 10
    então interrompa
  fim se
  leia NOTA[I]
  I ← I + 1
fim repita
.
.
fim algoritmo.

```

Supondo que as notas fossem 70, 75, 80, 90, 65, 85, 75, 85, 91, 92 e estivessem uma em cada linha de um dispositivo de entrada, após a execução do trecho do algoritmo anterior, a variável NOTA ficaria assim:

NOTA

70	75	80	90	65	85	75	85	91	92
1	2	3	4	5	6	7	8	9	10

Exemplo 2.5

Escrever o trecho do algoritmo que leia um conjunto de 10 notas, armazene-as na variável composta NOTA e calcule a sua média.

```

Algoritmo
.
.
I ← 1
repita
  se I > 10
    então interrompa
  fim se
  leia NOTA[I]
  SOMA ← SOMA + NOTA[I]
  I ← I + 1
fim repita
MÉDIA ← SOMA/10
.
.
fim algoritmo.

```

Exemplo 2.6

Escrever a estrutura que compare o conteúdo da sétima posição da variável NOTA com o valor 60.

```

se NOTA[7] > 60
  entao
    _____
    _____
    _____
  senao
    _____
    _____
  fim se

```

De forma genérica, utilizando a variável ÍNDICE, tem-se:

```

INDICE ← 7
se NOTA[ÍNDICE] > 60
  entao
    _____
    _____
  senao
    _____
    _____
  fim se

```

Exemplo 2.7

Escrever o trecho do algoritmo que, tendo em vista a variável composta montada no exemplo 2.5, calcule e escreva o número de alunos com nota superior à MÉDIA.

A variável QUANTIDADE será usada para armazenar o número de alunos que estão nas condições especiais indicadas pelo problema.

Algoritmo

```

QUANTIDADE ← 0
I ← 1
repita
  se I > 10
    então interrompa
  senao
    se NOTA[I] > MÉDIA
      então QUANTIDADE ← QUANTIDADE + 1
    fim se
    I ← I + 1
  fim se
  escreva MEDIA, QUANTIDADE
fim algoritmo

```

A partir do conceito fundamental de variáveis compostas são criadas estruturas de dados mais complexas, visando adequar o instrumento aos vários problemas do dia-a-dia do programador. Assim, surgem as variáveis compostas bidimensionais, que necessitam de dois índices para individualização de seus elementos. São muito úteis quando se tem que manipular matrizes e tabelas. O primeiro índice representa o número da linha e o segundo, o número da coluna.

Exemplo 2.8

A variável MATRIZ é constituída por nove elementos dispostos em três linhas de três colunas cada:

MATRIZ		
1	2	3
1	12	19
2	69	14
3	11	112
	32	42

MATRIZ[2,2] referencia o elemento da segunda linha e segunda coluna, cujo conteúdo é 14; MATRIZ[2,3] referencia o elemento da segunda linha e terceira coluna, cujo conteúdo é 37.

O conceito de variáveis compostas bidimensionais é estendido para as de n dimensões. Trata-se, apenas, de uma facilidade a mais, visto que o conjunto de índices, conceitualmente, pode ser encarado como um refinamento da individualização do elemento do conjunto. A seguir, serão abordadas as variáveis compostas unidimensionais e as multidimensionais.

2.2.1. Variáveis compostas unidimensionais

Conjuntos de dados referenciados por um mesmo nome e que necessitam de somente um índice para que seus elementos sejam endereçados são ditos compostos unidimensionais.

Exemplo 2.9

São unidimensionais as variáveis NOTA, PESO, IDADE e NOME mostradas a seguir.

NOTA

72	43	91	67	92	84	76	79	81	20
----	----	----	----	----	----	----	----	----	----

PESO

46,5	66,1	99,1	33,1	14,2	132,0	67,5	58,8
------	------	------	------	------	-------	------	------

IDADE

27	28	15	12	64
----	----	----	----	----

NOME

J	O	S	E
---	---	---	---

2.2.1.1 DECLARAÇÃO

A criação de variáveis compostas unidimensionais é feita através da seguinte declaração:

declare lista de identificadores [ℓ_i : ℓ_s])

onde:

-) declare lista de identificadores é uma palavra-chave; são nomes associados às variáveis que se deseja declarar;
-) ℓ_i é o limite inferior do intervalo de variação dos índices;
-) ℓ_s é o limite superior do intervalo de variação dos índices;
-) t é o tipo dos componentes da variável (numérico, literal, lógico)

As regras para a formação dos identificadores das variáveis compostas são as mesmas dos identificadores das variáveis simples. Quanto aos índices, a única restrição é que o limite superior não seja menor do que o limite inferior ($\ell_s \geq \ell_i$).

Exemplo 2.10

Declarar uma variável composta de 10 elementos numéricos de nome NOTA.

declare NOTA[1:10] numérico

Com esta declaração, cria-se uma estrutura de dados constituída por 10 elementos numéricos, endereçáveis por índices que podem variar de 1 a 10.

Exemplo 2.11

Preencher o arranjo M, abaixo, com o valor 2.

M

1	2	3	4	5	6
---	---	---	---	---	---

Algoritmo

```
declare M[1:6] numérico
declare I numérico
I ← 1
repita
    se I > 6
        então interrompa
    fim se
    M[I] ← 2
    I ← I + 1;
fim repita
escreva M[1], M[2] ... M[6]
fim algoritmo.
```

Após a execução do "repita", a variável composta M ficaria assim:

M

2	2	2	2	2	2
1	2	3	4	5	6

A execução do comando "escreva" ocasionaria o aparecimento de

2,2,2,2,2,2

2,2,2,2,2,2

na unidade de saída.

Exemplo 2.12

Ler um conjunto de cinco elementos numéricos, armazene-los na variável A e escrevê-los.

7, 40, 90, 37, 18

Dados na unidade de entrada

Algoritmo

```
declare A[1:5] numérico
leia A[1] ... A[5]
escreva A[1] ... A[5]
fim algoritmo.
```

Exemplo 2.13

Ler uma variável de 100 elementos numéricos e verificar se existem elementos iguais a 30. Se existirem, escrever as posições em que estão armazenados.

Algoritmo

```
Defina tipo das variáveis
leia NÚMEROS[1] ... NÚMEROS[100]
Verifique se existem elementos iguais a 30
fim algoritmo.
```

Ref. Verifique se existem elementos iguais a 30

```
I ← 1
repita
    se I > 100
        então interrompa
    fim se
    se NÚMEROS[I] = 30
        então escreva I
    fim se
    I ← I + 1
fim repita
fim ref.
```

Ref. Defina tipo das variáveis

```
declare NÚMEROS[1:100] numérico
declare I numérico
fim ref.
```

Finalmente, inserindo-se os refinamentos em seus respectivos lugares, obtém-se:

Algoritmo

```
{Definição do tipo das variáveis}
declare NÚMEROS[1:100] numérico
declare I numérico
leia NÚMEROS[1] ... NÚMEROS[100]
```

{Verificação da existência de elementos iguais a 30}

```

I ← 1
repita
  se I > 100
    então interrompa
  fim_se
  se NÚMEROS[I] = 30
    então escreva I
  fim_se
  I ← I + 1
fim_repita
fim_algoritmo.

```

Exemplo 2.14

O Instituto de Ciências Exatas da UFMG deseja saber se existem alunos cursando, simultaneamente, as disciplinas "Programação de Computadores" e "Cálculo Numérico". Existem disponíveis na unidade de entrada os números de matrícula dos alunos de "Programação de Computadores" (no máximo 150 alunos) e de "Cálculo Numérico" (no máximo 220 alunos). Cada conjunto dos números de matrícula dos alunos de uma disciplina tem a matrícula fictícia 9999 no final.

Formular um algoritmo que imprima o número de matrícula dos alunos que estão cursando estas disciplinas simultaneamente.

Trata-se, então, da verificação da ocorrência de um elemento de um conjunto em um outro conjunto. Assim, após a leitura dos dados, poderiam estar montadas as seguintes variáveis compostas unidimensionais PC e CN contendo, respectivamente, os números de matrícula dos alunos que estão cursando: "Programação de Computadores" e "Cálculo Numérico".

PC

							...		9999
1	2	3	4	5	6			150	151

CN

							...		9999
1	2	3	4	5	6			220	221

Algoritmo

```

Declare variáveis
Leia a matrícula dos alunos que estão cursando PC
Leia a matrícula dos alunos que estão cursando CN
Verifique a matrícula simultânea em PC e CN
fim_algoritmo.

```

Ref. Leia a matrícula dos alunos que estão cursando PC

```

I ← 1
repita
  leia PC[I]
  se PC[I] = 9999
    então interrompa
  fim_se
  I ← I + 1
fim_repita
fim_ref.

```

Ref. Leia a matrícula dos alunos que estão cursando CN

```

I ← 1
repita
  leia CN[I]
  se CN[I] = 9999
    então interrompa
  fim_se
  I ← I + 1
fim_repita
fim_ref.

```

Ref. Verifique a matrícula simultânea em PC e CN

```

I ← 1
repita
  se PC[I] = 9999
    então interrompa
  fim_se
  J ← 1
repita
  se CN[J] = 9999 ou PC[I] = CN[J]
    então interrompa
  fim_se
  J ← J + 1
fim_repita
se PC[I] = CN[J]
  então escreva PC[I]
fim_se
I ← I + 1
fim_repita
fim_ref.

```

Ref. Declare as variáveis

```

declare PC[1:151], {matrícula em Programação de Computadores}
CN[1:221] {matrícula em Cálculo Numérico}
declare I, J numérico
fim_ref.

```

Pode-se observar que, mesmo para o caso extremo em que haja 150 alunos matriculados em Programação de Computadores e 220 alunos matriculados em Cálculo Numérico, existe tanto na variável composta PC quanto na variável composta CN uma posição a mais devido à existência da matrícula fictícia 9999.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo, tem-se:

Algoritmo

```

declare {Declaração das variáveis}
PC[1:151], {matrícula em Programação de Computadores}
CN[1:221] {matrícula em Cálculo Numérico}

```

```

declare I, J numérico
(Lerda matrícula dos alunos que estão cursando PC)

```

```

I ← 1
repita
    leia PC[I]
    se PC[I] = 9999
        então interrompa
    fim se
    I ← I + 1
fim repita

```

{Lerda matrícula dos alunos que estão cursando CN}

```

I ← 1
repita
    leia CN[I]
    se CN[I] = 9999
        então interrompa
    fim se
    I ← I + 1
fim repita

```

{Verificação da matrícula simultânea em PC e CN}

```

I ← 1
repita
    se PC[I] = 9999
        então interrompa
    fim se

```

J ← 1

```

repita
    se CN[J] = 9999 ou PC[I] = CN[J]
        então interrompa
    fim se
    J ← J + 1
fim repita

```

```

se PC[I] = CN[J]
    então escreva PC[I]
fim se
I ← I + 1

```

```

fim repita
fim algoritmo.

```

Exemplo 2.15

Escrever um algoritmo que faça reserva de passagens aéreas de uma companhia. Além da leitura do número dos vôos e quantidade de lugares disponíveis, ler vários pedidos de reserva, constituídos do número da carteira de identidade do cliente e do número do voo desejado.

Para cada cliente, verificar se há disponibilidade no voo desejado. Em caso afirmativo, imprimir o número da identidade do cliente, e o número do voo, atualizando o número de lugares disponíveis. Caso contrário, avisar ao cliente da inexistência de lugares.

Indicando o fim dos pedidos de reserva, existe um passageiro cujo número da carteira de identidade é 9999. Considerar fixo e igual a 37 o número de vôos da companhia.

Estrutura de dados necessária:

1	727	15	
2	442	16	
3	331	0	
.	.	.	
35	4471	90	
36	221	16	
37	291	15	

N.º DOS VÔOS LUGARES DISPONÍVEIS

Algoritmo

```

Declare as variáveis
leia VÔOS[1] ...VÔOS[37], LDISP[1] ...LDISP[37]
repita
    leia CLIENTE, NVÔO
    se CLIENTE = 9999
        então interrompa
    fim se
    Verifique a existência do voo
    Verifique a existência de lugar
fim repita
fim algoritmo.

```

Ref. Verifique a existência do voo

```

I ← 0
repita
    I ← I + 1
    se I = 37 ou VÔOS[I] = NVÔO
        então interrompa
    fim se
fim repita
se VÔOS[I] = NVÔO
    então APONT ← I
    senão escreva "VÔO INEXISTENTE"
        APONT ← 0
fim se
fim ref.

```

Ref. Verifique a existência de lugar

```

se APONT ≠ 0
    então se LDISP[APONT] > 0
        então escreva CLIENTE, NVÔO
            Atualize o número de lugares disponíveis
        senão escreva NVÔO, "LOTADO"
    fim se
fim se
fim ref.

```

Ref. Atualize o número de lugares disponíveis
LDISP[APONT] ← LDISP[APONT] - 1
fim ref.

Ref. Declare as variáveis
declare VÔOS, {número dos vôos existentes}
 LDISP[1:37] {lugares disponíveis}
declare numérico
 APONT, CLIENTE,I,NVÔO numérico
 fim ref.

Pode-se observar que se duas ou mais variáveis compostas de mesmo tipo têm o mesmo número de elementos, somente o último identificador terá o par de limites dos índices.

```

Algoritmo {Reserva de passagens aéreas}
    {Declaração das variáveis}
    declare VÔOS, {número dos vôos existentes}
        LDISP[1:37] {lugares disponíveis}
            numérico
    declare APONT, CLIENTE, I, NVÔO numérico
    leia VÔOS[1] ... VÔOS[37], LDISP[1] ... LDISP[37]
    repita
        leia CLIENTE, NVÔO
        se CLIENTE = 9999
            então interrompa
        fim se {Verificação da existência do vôo}
        I ← 0
        repita
            I ← I + 1
            se I = 37 ou VÔOS[I] = NVÔO
                então interrompa
            fim se
        fim repita
        se VÔOS[I] = NVÔO
            então APONT ← I
            senão escreva "VÔO INEXISTENTE"
            APONT ← 0
        fim se {Verificação da existência de lugar}
        se APONT ≠ 0
            então se LDISP(APONT) > 0
                então escreva CLIENTE, NVÔO
                LDISP(APONT) ← LDISP(APONT) - 1
                senão escreva NVÔO, "LOTADO"
            fim se
        fim se
    fim repita
fim algoritmo.

```

□ Exemplo 2.16

Fazer um algoritmo para corrigir provas de múltipla escolha. Cada prova tem 10 questões, cada questão valendo um 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 fiktício, será 9999. O algoritmo deverá calcular e imprimir:

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

Estrutura de dados principal:

GABARITO	<input type="text"/>	NÚMERO	<input type="text"/>
RESPOSTAS	<input type="text"/>	APROVADOS	<input type="text"/>
FREQÜÊNCIA	<input type="text"/>	MAIOR	<input type="text"/>

Algoritmo

Declare as variáveis
Atribua valores iniciais necessários
leia GABARITO[1] ... GABARITO[10]
repita
 leia NÚMERO, RESPOSTAS[1] ... RESPOSTAS[10]
 se NÚMERO = 9999
 então interrompa
 fim se
 Corrija a prova
 escreva NÚMERO, NOTA
 TOTAL ← TOTAL + 1
 Determine a freqüência das notas
fim repita
Determine a porcentagem de aprovação
Determine a nota de maior freqüência
escreva NOTAMAIOR, PORCENT
fim algoritmo.

```

Ref. Corija a prova
NOTA ← 0
L ← 1
repita
    se L > 10
        então interrompa
    fim se
    se GABARITO[L] = RESPOSTAS[L]
        então NOTA ← NOTA + 1
    fim se
    L ← L + 1
fim repita
fim ref.

```

Ref. Determine a frequência das notas
 $\text{FREQÜÊNCIA}[\text{NOTA}] \leftarrow \text{FREQÜÊNCIA}[\text{NOTA}] + 1$
fim ref.

Este comando de atribuição armazena na variável composta unidimensional FREQÜÊNCIA o número de vezes que cada nota está ocorrendo. Observe-se, ainda, que a própria nota determina a posição do elemento de FREQÜÊNCIA que será alterado.

Ref. Determine a porcentagem de aprovação
 $\text{APROVADOS} \leftarrow 0$
 $L \leftarrow 6$
repita
se $L > 10$
então interrompa
fim se
 $\text{APROVADOS} \leftarrow \text{APROVADOS} + \text{FREQÜÊNCIA}[L]$
 $L \leftarrow L + 1$
fim repita
 $\text{PORCENT} \leftarrow \text{APROVADOS}/\text{TOTAL} \times 100$
fim ref.

Ref. Determine a nota de maior freqüência
 $L \leftarrow 0$
 $\text{MAIOR} \leftarrow 0$
repita
se $L > 10$
então interrompa
fim se
se $\text{FREQÜÊNCIA}[L] > \text{MAIOR}$
então $\text{MAIOR} \leftarrow \text{FREQÜÊNCIA}[L]$
 $\text{NOTAMAIOR} \leftarrow L$
fim se
 $L \leftarrow L + 1$
fim repita
fim ref.

Ref. Atribua valores iniciais necessários
 $\text{TOTAL} \leftarrow 0$
 $L \leftarrow 0$
repita
se $L > 10$
então interrompa
fim se
 $\text{FREQÜÊNCIA}[L] \leftarrow 0$
 $L \leftarrow L + 1$
fim repita
fim ref.

Ref. Declare as variáveis
declare GABARITO, RESPOSTAS[1:10] literal
declare FREQÜÊNCIA [0:10] numérico
declare APROVADOS,L,MAIOR,NOTA,NOTAMAIOR,NÚMERO,
PORCENT,TOTAL numérico
fim ref.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo, tem-se:

Algoritmo {Correção de provas de múltipla escolha}
{Declaração das variáveis}
declare GABARITO,RESPOSTAS[1:10] literal
declare FREQÜÊNCIA [0:10] numérico
declare APROVADOS,L,MAIOR,NOTA,NOTAMAIOR,NÚMERO,
PORCENT,TOTAL numérico
{Atribuição de valores iniciais necessários}
 $\text{TOTAL} \leftarrow 0$
 $L \leftarrow 0$
repita
se $L > 10$
então interrompa
fim se
 $\text{FREQÜÊNCIA}[L] \leftarrow 0$
 $L \leftarrow L + 1$
fim repita
leia GABARITO[1] ... GABARITO[10]
repita
leia NÚMERO,RESPOSTAS[1]...RESPOSTAS[10]
se NÚMERO = 9999
então interrompa
fim se
{Correção da prova}
 $\text{NOTA} \leftarrow 0$
 $L \leftarrow 1$
repita
se $L > 10$
então interrompa
fim se
se $\text{GABARITO}[L] = \text{RESPOSTAS}[L]$
então $\text{NOTA} \leftarrow \text{NOTA} + 1$
fim se
 $L \leftarrow L + 1$
fim repita
escreva NÚMERO, NOTA
 $\text{TOTAL} \leftarrow \text{TOTAL} + 1$
{Determinação da freqüência das notas}
 $\text{FREQÜÊNCIA}[\text{NOTA}] \leftarrow \text{FREQÜÊNCIA}[\text{NOTA}] + 1$
fim repita
{Determinação da porcentagem de aprovação}
 $\text{APROVADOS} \leftarrow 0$
 $L \leftarrow 6$
repita
se $L > 10$
então interrompa
fim se

```

APROVADOS ← APROVADOS + FREQÜÊNCIA[L]
L ← L + 1
fim repita
PORCENT ← APROVADOS/TOTAL × 100
(Determinação da nota de maior freqüência)
L ← 0
MAIOR ← 0
repita
  se L > 10
    então interrompa
  fim se
  se FREQÜÊNCIA[L] > MAIOR
    então MAIOR ← FREQÜÊNCIA[L]
    NOTAMAIOR ← L
  fim se
  L ← L + 1
fim repita
escreva NOTAMAIOR, PORCENT
fim algoritmo.

```

Exemplo 2.17

Dado um conjunto A de n valores numéricos ($n \leq 200$), fazer um algoritmo que imprima seus valores em ordem crescente.

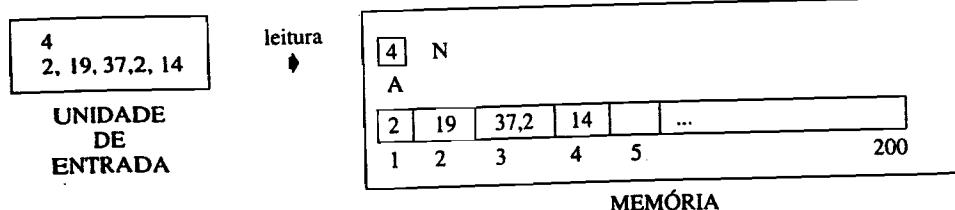
Algoritmo

```

Declare as variáveis
leia N
leia A[1]...A[N]
Ordene A
escreva A[1]...A[N]
fim algoritmo.

```

Este algoritmo lê a quantidade de dados disponíveis (N), e os transfere para a memória, armazenando-os na variável composta unidimensional A.
Suponha N = 4, tem-se a seguinte simulação desta situação:



Para se refinar o comando "Ordene A", será usado o método de inserção direta. Este método adota um procedimento análogo ao do jogador de baralho que, ao receber um conjunto de cartas desordenadas, ordena-as, partindo das cartas colocadas mais à direita na sua mão, inserindo-as na posição mais adequada.

Ref. Ordene A

```

J ← 2
repita
  se J > N
    então interrompa
  fim se
  Insira A[J] dentre os elementos já ordenados, deslocando os maiores que A[J]
  J ← J + 1
fim repita
fim ref.

```

Ref. Insira A[J] dentre os elementos já ordenados, deslocando os maiores que A[J]

```

AUX ← A[J]
I ← J
repita
  I ← I - 1
  se I = 1 ou AUX > A[I]
    então interrompa
  fim se
  A[I + 1] ← A[I]
fim repita
se AUX > A[I]
  então A[I + 1] ← AUX
  senão A[I + 1] ← A[I]
  A[I] ← AUX
fim se
fim ref.

```

Para inserir A[J], supõe-se A[1], A[2]...A[J - 1] já ordenados e compara-se A[J] com A[J - 1], A[J - 2]..., até se descobrir que ele deve ser inserido entre A[I] e A[I + 1]. Move-se, então, A[I + 1] e os elementos seguintes, já ordenados, atribuindo-se a A[I + 1] o valor original de A[J].

Ref. Declare as variáveis

```

declare A[1:200] numérico
declare AUX, I, J, N numérico
fim ref.

```

O algoritmo final é dado a seguir e foi obtido a partir dos refinamentos:

Algoritmo {Declaração das variáveis}

```

declare A[1:200] numérico
declare AUX,I,J,N numérico
leia N
leia A[1]...A[N]
{Ordenação de A}
J ← 2
repita
  se J > N
    então interrompa
  fim se

```

```

AUX ← A[J]
I ← J
repita
  I ← I - 1
  se I = 1 ou AUX > A[I]
    então interrompa
  fim se
  A[I + 1] ← A[I]
fim repita
se AUX > A[I]
  então A[I + 1] ← AUX
senão A[I + 1] ← A[I]
  A[I] ← AUX
fim se
J ← J + 1
fim repita
escreva A[1] ...A[N]
fim algoritmo.

```

Exemplo 2.18

Está disponível num equipamento de entrada de dados o estoque de mercadorias de uma loja. São dados os códigos das mercadorias e as respectivas quantidades existentes. A seguir, estão os pedidos dos clientes. Fazer um algoritmo para atualização do estoque, tal que:

- seja lido e listado o estoque inicial (máximo de 100 mercadorias);
- sejam lidos os pedidos dos clientes, constituído, cada um, do número do cliente, código da mercadoria e quantidade desejada;
- seja verificado, para cada pedido, se ele pode ser integralmente atendido. Em caso negativo, imprima o número do cliente e a mensagem "NÃO TEMOS A MERCADORIA EM ESTOQUE SUFICIENTE";
- seja atualizado o estoque após cada operação;
- seja listado o estoque final.

Observação: Considerar que, separando os dados do estoque inicial dos de pedidos, exista um dado cujo código de mercadoria é 9999 e encerrando os pedidos haja um cliente fictício, cujo número é 9999.

Algoritmo

```

Declare as variáveis
Leia e escreva o estoque inicial
repita
  leia CLIENTE, MERCADORIA, QUANTIDADE
  se CLIENTE = 9999
    então interrompa
  fim se
  Verifique a existência da mercadoria
fim repita
Escreva o estoque final
fim algoritmo.

```

Ref. Leia e escreva o estoque inicial

```

I ← 1
repita
  leia CÓDIGO(I),ESTOQUE(I)

```

```

se CÓDIGO(I) = 9999
  então interrompa
fim se
escreva CÓDIGO(I),ESTOQUE(I)
I ← I + 1
fim repita
N ← I - 1
fim ref.

```

Ref. Verifique a existência de mercadoria

```

I ← 0
repita
  I ← I + 1
  se I = N ou MERCADORIA = CÓDIGO(I)
    então interrompa
  fim se
fim repita
se MERCADORIA = CÓDIGO(I)
  então Verifique o estoque
  senão escreva "NÃO EXISTE A MERCADORIA PEDIDA"
fim se
fim ref.

```

Ref. Verifique o estoque

```

se ESTOQUE(I) ≥ QUANTIDADE
  então atualize o estoque
  senão escreva "NAO TEMOS A MERCADORIA EM ESTOQUE
  SUFICIENTE"
fim se
fim ref.

```

Ref. Atualize o estoque

```

ESTOQUE(I) ← ESTOQUE(I) - QUANTIDADE
fim ref.

```

Ref. Escreva o estoque final

```

I ← 1
repita
  se I > N
    então interrompa
  fim se
  escreva CÓDIGO(I),ESTOQUE(I)
  I ← I + 1
fim repita
fim ref.

```

Ref. Declare as variáveis

```

declare ESTOQUE,CÓDIGO[1:101] numérico
declare CLIENTE,I,MERCADORIA,N,QUANTIDADE numérico
fim ref.

```

A seguir, pode ser visto o texto final do algoritmo:

```

Algoritmo
  {Declaração das variáveis}
  declare ESTOQUE,CÓDIGO[1:101] numérico
  declare CLIENTE,I,MERCADORIA,N,QUANTIDADE numérico
  {Leitura e escrita do estoque inicial}

  I ← 1
  repita
    leia CÓDIGO[I],ESTOQUE[I]
    se CÓDIGO[I] = 9999
      então interrompa
    fim se
    escreva CÓDIGO[I],ESTOQUE[I]
    I ← I + 1
  fim repita
  N ← I - 1
  repita
    leia CLIENTE,MERCADORIA,QUANTIDADE
    se CLIENTE = 9999
      então interrompa
    fim se
    {Verificação da existência da mercadoria}
    I ← 0
    repita
      I ← I + 1
      se I = N ou MERCADORIA = CÓDIGO[I]
        então interrompa
      fim se
    fim repita
    se MERCADORIA = CÓDIGO[I]
      então se ESTOQUE[I] ≥ QUANTIDADE
        então ESTOQUE[I] ← ESTOQUE[I] - QUANTIDADE
        senão escreva "NÃO TEMOS A MERCADORIA EM
          ESTOQUE SUFICIENTE"
      fim se
      senão escreva "NÃO EXISTE A MERCADORIA PEDIDA"
    fim se
  fim repita
  {Escrita do estoque final}

  I ← 1
  repita
    se I > N
      então interrompa
    fim se
    I ← I + 1
    escreva CÓDIGO[I],ESTOQUE[I]
  fim repita
fim algoritmo.

```

2.2.1.2. EXERCÍCIOS DE FIXAÇÃO

2.2.1.2.1. Resolver o exemplo 2.1 utilizando variáveis compostas unidimensionais e calcular e escrever a porcentagem de alunos que tiveram notas abaixo da média.

2.2.1.2.2. Fazer um algoritmo que calcule e escreva o somatório dos valores armazenados numa variável composta unidimensional A, de 100 elementos numéricos a serem lidos do dispositivo de entrada.

Exemplo:

A

32	17	...	193,7	15,8
1	2		99	100

$$\text{SOMATÓRIO} = 32 + 17 + \dots + 193,7 + 15,8 = \sum_{i=1}^{100} a_i$$

● 2.2.1.2.3. Fazer um algoritmo que:

- leia duas variáveis compostas unidimensionais, contendo, cada uma, 25 elementos numéricos;
- intercale os elementos destes dois conjuntos formando uma nova variável composta unidimensional de 50 elementos;
- escreva o novo conjunto, assim obtido.

Exemplo:

A

1800	225,9	667	...	320	11	0,8
1	2	3		23	24	25

B

7	4	11	..	3	1	8
1	2	3		23	24	25

C

1800	7	225,9	4	667	11	...	320	3	11	1	0,8	8
1	2	3	4	5	6		45	46	47	48	49	50

● 2.2.1.2.4. Fazer um algoritmo que:

- leia 100 valores numéricos e os armazene numa variável composta unidimensional A;
- calcule e escreva

$$S = \sum_{i=1}^{100} \frac{i}{a_i}, \text{ onde } a_i \text{ é o } i\text{-ésimo valor armazenado na variável A};$$

- calcule e escreva quantos termos da série têm o numerador inferior ao denominador.

Exemplo:

A

-48	37	99,2	...	16,3
1	2	3		100

SOMALINHA

1	0	2	-1	3
4	3	2	1	0
1	-2	3	4	5
8	5	1	3	2

TOTAL 45

5

10

11

19

Ref. Escreva a soma de todos os elementos

escreva TOTAL

fim ref.

Ref. Atribua valores iniciais necessários

TOTAL ← 0

fim ref.

Ref. Defina tipo das variáveis

declare MAT[1:4,1:5] numérico

declare I,J,SOMALINHA, TOTAL numérico

fim ref.

Algoritmo

Defina tipo das variáveis

Atribua valores iniciais necessários

leia MAT[1,1] ...MAT[4,5]

I ← 1

repita

se I > 4
então interrompa

fim se

Calcule e escreva a soma dos elementos da linha

Calcule a soma de todos os elementos

I ← I + 1

fim repita

Escreva a soma de todos os elementos

fim algoritmo.

Chamando-se de SOMALINHA a uma variável que conterá a soma dos elementos da linha, e usando-se notação matemática, tem-se:

$$\text{SOMALINHA} = \sum_{j=1}^5 \text{MAT}_{ij}, i = 1, 2, 3, 4$$

Ref. Calcule e escreva a soma dos elementos da linha

SOMALINHA ← 0

J ← 1

repita

se J > 5
então interrompa

fim se

SOMALINHA ← SOMALINHA + MAT[I,J]

J ← J + 1

fim repita

escreva SOMALINHA

fim ref.

Ref. Calcule a soma de todos os elementos

TOTAL ← TOTAL + SOMALINHA

fim ref.

Algoritmo

{Definição do tipo das variáveis}

declare MAT[1:4,1:5] numérico

declare I,J,SOMALINHA,TOTAL numérico

TOTAL ← 0

leia MAT[1,1] ...MAT[4,5]

I ← 1

repita

se I > 4
então interrompa

fim se

{Cálculo da soma dos elementos da linha}

SOMALINHA ← 0

J ← 1

repita

se J > 5
então interrompa

fim se

SOMALINHA ← SOMALINHA + MAT[I,J]

J ← J + 1

fim repita

escreva SOMALINHA

{Cálculo da soma de todos os elementos}

TOTAL ← TOTAL + SOMALINHA

I ← I + 1

fim repita

{Escrita da soma de todos os elementos}

escreva TOTAL

fim algoritmo.

 Exemplo 2.25

A multiplicação de duas matrizes A e B só é possível se o número de colunas da matriz A for igual ao número de linhas da matriz B. Assim, se A é uma matriz m × n e B, uma matriz n × p, a multiplicação será possível e o produto será uma matriz C, m × p.

O produto matricial pode ser muito útil em várias aplicações como, por exemplo, na situação descrita a seguir.

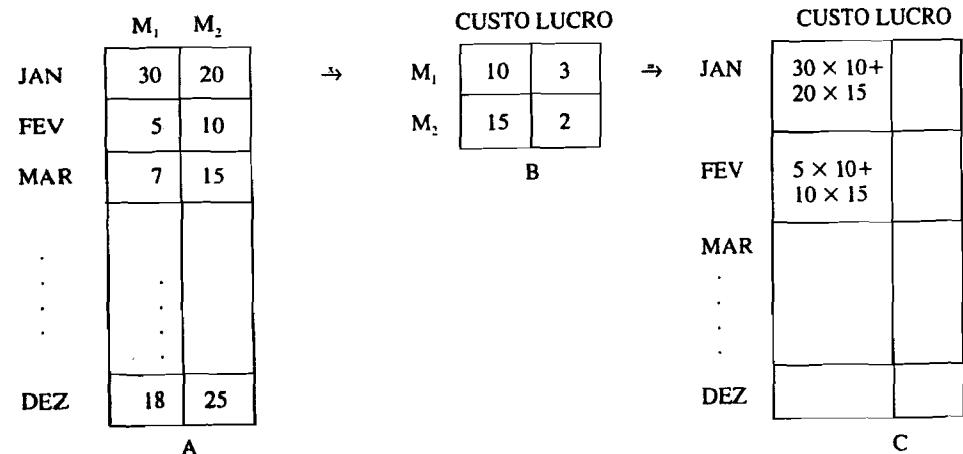
Uma certa fábrica produziu dois tipos de motores M_1 e M_2 nos meses de janeiro, ..., dezembro e o número de motores produzidos foi registrado na tabela a seguir:

	M_1	M_2
JAN	30	20
FEV	5	10
MAR	7	15
.	.	
.	.	
.	.	
DEZ	18	25

O setor de controle de vendas tem uma tabela do custo e do lucro (em unidades monetárias) obtidos com cada motor.

	CUSTO	LUCRO
M_1	10	3
M_2	15	2

Para saber o custo e o lucro nos meses de janeiro, ..., dezembro, basta que se faça o produto matricial das duas tabelas.



Ou seja:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

$$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$

Portanto,

$$C_{ij} = \sum_{k=1}^2 A_{ik} B_{kj}$$

Fazer um algoritmo que, a partir da produção mensal de motores M_1 e M_2 e seus respectivos custos e lucros, calcule o custo e lucro em cada um dos meses e o custo e lucro anuais.

Algoritmo

```

Declare as variáveis
leia PRODUÇÃO[1,1] ...PRODUÇÃO[12,2]
leia VALORES[1,1] ...VALORES[2,2]
Calcule custo e lucro mensais
Calcule custo e lucro anuais
Escreva os resultados obtidos
fim algoritmo.

```

Ref. Calcule custo e lucro mensais

```

I ← 1
repita
  se I > 12
    então interrompa
  fim se
  J ← 1
  repita
    se J > 2
      então interrompa
    fim se
    CLM[I,J] ← 0
    K ← 1
    repita
      se K > 2
        então interrompa
      fim se
      CLM[I,J] ← CLM[I,J] + PRODUÇÃO[I,K] × VALORES[K,J]
      K ← K + 1
    fim repita
    J ← J + 1
  fim repita
I ← I + 1
fim repita
fim ref.

```

Para se calcular o custo e lucro anuais, somam-se os elementos das colunas 1 e 2, respectivamente, da matriz de custo e lucro mensais.

Ref. Calcule custo e lucro anuais

J ← 1

repita

se J > 2
então interrompa

fim se

CLA[J] ← 0

I ← 1

repita

se I > 12
então interrompa

fim se

CLA[J] ← CLA[J] + CLM[I,J]

I ← I + 1

fim repita

J ← J + 1

fim repita

fin ref.

Ref. Escreva os resultados obtidos

escreva CLM[1,1] ...CLM[12,2]

escreva CLA[1],CLA[2]

fin ref.

Ref. Declare as variáveis

declare PRODUÇÃO
CLM[1:12,1:2],
VALORES[1:2,1:2],
CLA[1:2]
número

declare I,J,K número

fin ref.

{produção mensal de cada motor}
{custo e lucro mensais dos motores}
{custo e lucro de cada motor}
{custo e lucro anuais dos motores}

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo, tem-se:

Algoritmo

{Declaração das variáveis}
declare PRODUÇÃO,
CLM[1:12,1:2],
VALORES[1:2,1:2],
CLA[1:2]
número

declare I,J,K número
leia PRODUÇÃO[1,1] ...PRODUÇÃO[12,2]
leia VALORES[1,1] ...VALORES[2,2]
{Cálculo do custo e do lucro mensais}

I ← 1

repita

se I > 12
então interrompa

fim se

J ← 1

repita

se J > 2
então interrompa

fim se

CLM[I,J] ← 0

K ← 1

repita

se K > 2
então interrompa

fim se

CLM[I,J] ← CLM[I,J] + PRODUÇÃO[I,K] × VALORES[K,J]

K ← K + 1

fim repita

J ← J + 1

fim repita

I ← I + 1

fim repita

{Cálculo do custo e do lucro anuais}

J ← 1

repita

se J > 2
então interrompa

fim se

CLA[J] ← 0

I ← 1

repita

se I > 12
então interrompa

fim se

CLA[J] ← CLA[J] + CLM[I,J]

I ← I + 1

fim repita

J ← J + 1

fim repita

{Escrita dos resultados obtidos}

escreva CLM[1,1] ...CLM[12,2]

escreva CLA[1],CLA[2]

fin algoritmo.

Exemplo 2.26

Na loteria esportiva da década de 80 o jogador tinha que escolher resultados para 16 jogos. Fazer um algoritmo que leia os palpites de um jogador e calcule o valor a ser pago. O algoritmo deverá ler e imprimir os dados referentes ao teste e ao revendedor como especificado na página seguinte.

O valor a ser pago é calculado por:

$$\text{VALOR} = V \times 2^D \times 3^T$$

onde:

V — constante a ser lida;

D — número de palpites duplos;

T — número de palpites triplas.

088781 0084 0933 002731 0561 066846
0000 000000 0035 054313

LOTERIA ESPORTIVA FEDERAL -TESTE 0933

REV 11.1.00236-2 NMAQ 08878-1
BILHETE Nº 002.732 44776497

-PROGNOSTICOS-

	1	1	1	1	1	1	1										
JOGOS/ COL.	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	
COL. 2	2	1	X	1	1	X	X	X	1	X	X	1	1				
COL. 3	X	X	1	X	X	1	1	1	X	1	1	1	1				
COL. 4	1	X	1	X	1	1	X	1	X	1	1	X	X				
VALOR PAGO																	CZ\$40,00

A estrutura de dados a ser manipulada pelo algoritmo consiste principalmente de duas variáveis compostas bidimensionais esquematizadas abaixo:

	JOGO 1	JOGO 2	...	JOGO 16
COL. 2				
COL. DO MEIO				
COL. 1				

MATRIZ DE APOSTAS

A matriz de apostas conterá X na coluna correspondente ao palpite dos apostadores em cada uma das linhas correspondentes a cada jogo.

Algoritmo

Declare as variáveis
Leia os dados do teste e revendedor
leia APOSTAS[1,1] ...APOSTAS[3,16]
Verifique o número de duplos e triplos
VALOR $\leftarrow V \times 2^6 \times 3^7$
Escreva resultados obtidos
fim algoritmo.

Ref. Leia os dados do teste e revendedor
leia NTESTE, NMAQ, NREV, NAPOSTA, V
fim ref.

Ref. Verifique o número de duplos e triplos

```
D ← 0
T ← 0
J ← 1
repita
  se J > 16
    então interrompa
  fim se
  N ← 0
  se APOSTAS[I,J] = "X"
    então N ← N + 1
  fim se
  se APOSTAS[2,J] = "X"
    então N ← N + 1
  fim se
  se APOSTAS[3,J] = "X"
    então N ← N + 1
  fim se
  se N = 3
    então T ← T + 1
    senão se N = 2
      então D ← D + 1
    fim se
  fim se
  J ← J + 1
fim repita
fim ref.
```

Ref. Escreva resultados obtidos
escreva "LOTERIA ESPORTIVA FEDERAL", NTESTE, NMAQ, NREV,
NAPOSTA

```
I ← 1
repita
  se I > 3
    então interrompa
  fim se
  escreva APOSTAS[I,1]...APOSTAS[I,16]
  I ← I + 1
fim repita
escreva VALOR
fim ref.
```

Ref. Declare as variáveis
declare APOSTAS[1:3,1:16] literal
declare D,I,N,NAPOSTA,NMAQ,NREV,NTESTE,T,V,VALOR numérico
fim ref.

Inserindo-se os refinamentos em seus respectivos lugares, tem-se:

Algoritmo

```

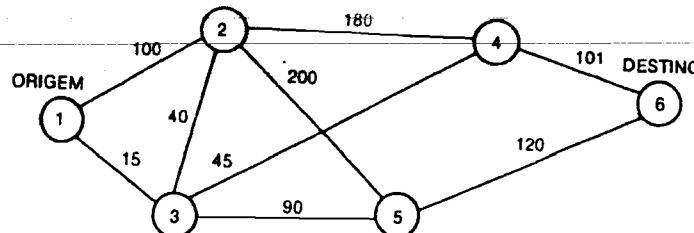
{Declaração das variáveis}
declare APOSTAS[1:3,1:16] literal
declare D,I,N,NAPOSTA,NMAQ,NREV,NTESTE,T,V,VALOR numérico
{Leitura dos dados do teste e revendedor}
leia NTESTE,NMAQ,NREV,NAPOSTA, V
leia APOSTAS
{Verificação do número de duplos e triplos}

D ← 0
T ← 0
J ← 1
repita
  se J > 16
    então interrompa
  fim se
  N ← 0
  se APOSTAS[1,J] = "X"
    então N ← N + 1
  fim se
  se APOSTAS[2,J] = "X"
    então N ← N + 1
  fim se
  se APOSTAS[3,J] = "X"
    então N ← N + 1
  fim se
  se N = 3
    então T ← T + 1
    senão se N = 2
      então D ← D + 1
    fim se
  fim se
  J ← J + 1
fim repita
VALOR ← V × 2D × 3T
{Escrita dos resultados obtidos}
escreva "LOTERIA ESPORTIVA FEDERAL",NTESTE,NMAQ,NREV,
NAPOSTA
I ← 1
repita
  se I > 3
    então interrompa
  fim se
  escreva APOSTAS[I,1]...APOSTAS[I,16]
  I ← I + 1
fim repita
escreva VALOR
fim algoritmo.

```

Exemplo 2.27

Os nós do grafo a seguir representam cidades e os arcos, a presença de uma estrada ligando duas destas cidades. Os números ao lado dos arcos são a distância medida em quilômetros.



Pode-se representar numericamente este grafo por uma variável composta bidimensional, na qual a distância entre duas cidades i, j é indicada pelo elemento $D[i,j]$; se $i = j$ ou se não houver conexão entre i e j , $D[i,j]$ será zero. Desta forma tem-se:

	1	2	3	4	5	6
1	0'	100	15	0	0	0
2	100	0	40	180	200	0
3	15	40	0	45	90	0
4	0	180	45	0	0	101
5	0	200	90	0	0	120
6	0	0	0	101	120	0

Matriz D de distância entre as cidades

Seja proposto agora o problema de se achar o caminho mais curto entre duas cidades quaisquer. Este problema foi resolvido por Dijkstra [DIJKSTRA, 1971] e tem uma série de aplicações em questões de otimização.

Além da matriz D das distâncias, considera-se a variável composta unidimensional DA, cuja componente DA[I] representa a distância acumulada em um caminho percorrido desde a origem até a cidade I. Cada uma destas componentes será iniciada com um valor bem grande, por exemplo, 10.000.

Ainda serão consideradas mais duas variáveis compostas unidimensionais. A primeira, designada ANT, será tal que a sua componente ANT[I] indica qual é a cidade antecedente de I no caminho considerado. A outra, EXP terá componentes lógicas, todas elas inicialmente com o valor falso, indicando que as cidades ainda não foram "expandidas".

Partindo de uma cidade C, inicialmente igual à origem, calcula-se a nova distância acumulada (NOVADA) de cada uma das cidades adjacentes a C ainda não expandidas. A nova distância acumulada prevalecerá sobre o valor anterior se lhe for inferior; neste caso, C será atribuído à componente ANT[I]. Quando terminar a expansão de C, registra-se que EXP[C] é verdadeiro.

Em seguida, procura-se, dentre as cidades ainda não expandidas, aquela que tenha a menor distância acumulada. Esta será a nova cidade C e a sua distância acumulada é, então, a menor que possa ser conseguida a partir da ORIGEM.

O processo será repetido até que a cidade C seja o DESTINO ou que não se encontre nenhuma cidade ainda não expandida, cuja distância acumulada seja inferior a 10.000. Neste último caso, isto significa que não existe caminho ligando a ORIGEM ao DESTINO.

Algoritmo

Declare as variáveis
Atribua os valores iniciais necessários
 $C \leftarrow ORIGEM$
 $DA[C] \leftarrow 0$
repita
 se $(C = DESTINO)$ ou $(C = 0)$
 então interrompa
 fim se
 Expanda C
 Determine o próximo C
fim repita
se $C = DESTINO$
 então Descreva o caminho mínimo
 senão escreva "Não existe caminho unindo as duas cidades"
fim se
fim algoritmo.

Ref. Descreva o caminho mínimo

repita
 escreva C
 se $C = ORIGEM$
 então interrompa
 fim se
 $C \leftarrow ANT[C]$
fim repita
fim ref.

Ref. Expanda C

$I \leftarrow 1$
repita
 se $I > N$
 então interrompa
 fim se
 se $D[C,I] \neq 0$ e não $EXP[I]$
 então $NOVADA \leftarrow DA[C] + D[C,I]$
 se $NOVADA < DA[I]$
 então $DA[I] \leftarrow NOVADA$
 $ANT[I] \leftarrow C$
 fim se
 fim se
 $I \leftarrow I + 1$
fim repita
 $EXP[A] \leftarrow verdadeiro$
fim ref.

Ref. Determine o próximo C

$MIN \leftarrow 10.000$
 $C \leftarrow 0$
 $I \leftarrow 1$

repita

 se $I > N$
 então interrompa
 fim se
 se (não $EXP[A]$) e ($DA[I] < MIN$)
 então $MIN \leftarrow DA[I]$
 $C \leftarrow I$
 fim se
 $I \leftarrow I + 1$
fim repita
fim ref.

Ref. Atribua os valores iniciais necessários

leia N
leia $D[1,1]...D[N,N]$
leia $ORIGEM, DESTINO$
 $I \leftarrow 1$
repita
 se $I > N$
 então interrompa
 fim se
 $EXP[A] \leftarrow \text{falso}$
 $DA[I] \leftarrow 10.000$
 $I \leftarrow I + 1$
fim repita
fim ref.

Ref. Declare as variáveis

declare $D[1:100,1:100], DA[1:100], ANT[1:100]$ numérico
declare $EXP[1:100]$ lógico
declare $N, ORIGEM, DESTINO, I, C, NOVADA, MIN$ numérico
fim ref.

O texto final do algoritmo é o seguinte:

Algoritmo

{Declaração das variáveis}
declare $D[1:100,1:100], DA[1:100], ANT[1:100]$ numérico
declare $EXP[1:100]$ lógico
declare $N, ORIGEM, DESTINO, I, C, NOVADA, MIN$ numérico
(Atribuição dos valores iniciais necessários)
leia N
leia $D[1,1]...D[N,N]$
leia $ORIGEM, DESTINO$
 $I \leftarrow 1$
repita
 se $I > N$
 então interrompa
 fim se
 $EXP[A] \leftarrow \text{falso}$
 $DA[I] \leftarrow 10.000$
 $I \leftarrow I + 1$

fim repita
 $C \leftarrow \text{ORIGEM}$
 $\text{DA}[C] \leftarrow 0$

repita
 se $(C = \text{DESTINO}) \text{ ou } (C = 0)$
 então interrompa

fim se
 {Expansão de C}

$I \leftarrow 1$

repita
 se $I > N$
 então interrompa

fim se

se $D[C,J] \neq 0 \text{ e } \text{não } \text{EXP}[I]$
 então $\text{NOVADA} \leftarrow \text{DA}[C] + D[C,I]$
 se $\text{NOVADA} < \text{DA}[I]$
 então $\text{DA}[I] \leftarrow \text{NOVADA}$
 $\text{ANT}[I] \leftarrow C$

fim se

$I \leftarrow I + 1$

fim repita

$\text{EXPA}[C] \leftarrow \text{verdadeiro}$
 {Determinação do próximo C}

$\text{MIN} \leftarrow 10000$

$C \leftarrow 0$

$I \leftarrow 1$

repita

se $I > N$
 então interrompa

fim se

se $(\text{não } \text{EXPA}[I]) \text{ e } (\text{DA}[I] < \text{MIN})$

então $\text{MIN} \leftarrow \text{DA}[I]$

$C \leftarrow 1$

fim se

$I \leftarrow I + 1$

fim repita

fim repita

se $C = \text{DESTINO}$

então {Descrição do caminho mínimo}

repita

escreva C

se $C = \text{ORIGEM}$

então interrompa

fim se

$C \leftarrow \text{ANT}[C]$

fim repita

senão escreva "Não existe caminho unindo as duas cidades"

fim se

fim algoritmo.

Exemplo 2.28

Visando fazer um levantamento das atuais condições de tráfego de uma malha rodoviária e definir alternativas para uma reformulação futura, foi realizada uma pesquisa de tráfego do tipo ORIGEM-DESTINO. Esta pesquisa submeteu aos motoristas que trafegavam pela região em estudo um questionário com as seguintes perguntas:

1 - De onde veio?
 (origem)

- 1 - Belo Horizonte
- 2 - São Paulo
- 3 - Santos Dumont
- 4 - Brasília

2 - Para onde vai?
 (destino)

- 1 - Rio de Janeiro
- 2 - Petrópolis
- 3 - Juiz de Fora
- 4 - Barbacena

3 - Qual o tipo de veículo?

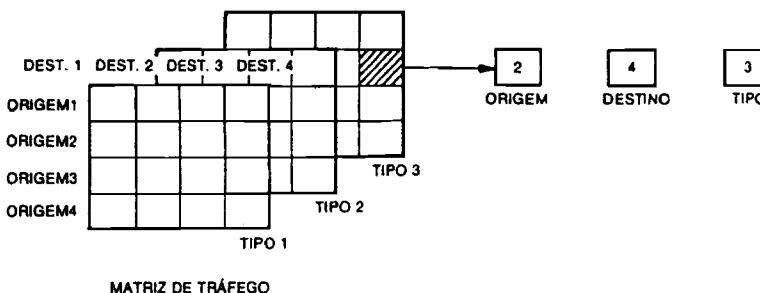
- 1 - automóvel
- 2 - caminhão
- 3 - ônibus

As respostas a estas três questões de cada entrevista, utilizando códigos numéricos (1, 2, 3, 4), estão disponíveis numa unidade de entrada. A última entrevista possui as três respostas iguais a zero.

A fim de fornecer respostas a algumas questões formuladas pelos engenheiros responsáveis pelo estudo do problema, fazer um algoritmo que calcule e imprima:

- a matriz origem-destino para cada tipo de veículo;
- o número de veículos que tem Belo Horizonte como origem;
- a quantidade de ônibus que se originam de Brasília e se destinam a Petrópolis ou Barbacena

A principal estrutura de dados utilizada é uma variável composta tridimensional, constituída de três matrizes bidimensionais, ou seja, três matrizes origem × destino, uma para cada veículo, esquematizadas a seguir:



Algoritmo

Declare as variáveis

Atribua valores iniciais à matriz de tráfego

Monte a matriz de tráfego

Escrive a matriz origem × destino para cada tipo de veículo

Calcule e escreva o número de veículos originários de Belo Horizonte

Calcule e escreva o número de ônibus originários de Brasília e que se destinam a Petrópolis ou Barbacena

fim algoritmo.

Ref. Atribua valores iniciais à matriz de tráfego

$I \leftarrow 1$

```

repita
  se I > 4
    então interrompa
  fim se
  J ← 1
  repita
    se J > 4
      então interrompa
    fim se
    K ← 1
    repita
      se K > 3
        então interrompa
      fim se
      TRÁFEGO[I,J,K] ← 0
      K ← K + 1
    fim repita
    J ← J + 1
  fim repita
  I ← I + 1
fim repita
fim ref.

```

Ref. Monte a matriz de tráfego

```

repita
  leia ORIGEM, DESTINO, TIPO
  se ORIGEM = 0 e DESTINO = 0 e TIPO = 0
    então interrompa
  fim se
  TRÁFEGO[ORIGEM, DESTINO, TIPO] ←
    TRÁFEGO [ORIGEM, DESTINO, TIPO] + 1
  fim repita
fim ref.

```

Ref. Escrita da matriz origem × destino para cada tipo de veículo

```

K ← 1
repita
  se K > 3
    então interrompa
  fim se
  escreva "VEÍCULO DO TIPO", K
  I ← 1
  repita
    se I > 4
      então interrompa
    fim se
    escreva TRÁFEGO[I,I,K]...TRÁFEGO[I,4,K]
    I ← I + 1
  fim repita
  K ← K + 1
fim repita
fim ref.

```

```

Ref. Calcule e escreva o número de veículos originários de Belo Horizonte
Q ← 0
J ← 1
repita
  se J > 4
    então interrompa
  fim se
  K ← 1
  repita
    se K > 3
      então interrompa
    fim se
    Q ← Q + TRÁFEGO[1,J,K]
    K ← K + 1
  fim repita
  J ← J + 1
fim repita
escreva Q
fim ref.

```

```

Ref. Calcule e escreva o número de ônibus originários de Brasília e que se destinam
a Petrópolis ou Barbacena
Q ← TRÁFEGO[4, 2, 3] + TRÁFEGO[4, 4, 3]
escreva Q
fim ref.

```

```

Ref. Declare as variáveis
declare TRÁFEGO[1:4,1:4,1:3] numérico
declare DESTINO,I,J,K,ORIGEM,Q,TIPO numérico
fim ref.

```

O texto final do algoritmo é dado a seguir:

Algoritmo

(Declaração das variáveis)

```

declare TRÁFEGO[1:4,1:4,1:3] numérico
declare DESTINO,I,J,K,ORIGEM,Q,TIPO numérico
  (Atribuição de valores iniciais à matriz de tráfego)
I ← 1
repita
  se I > 4
    então interrompa
  fim se
  J ← 1
  repita
    se J > 4
      então interrompa
    fim se
    K ← 1
    repita
      se K > 3
        então interrompa
      fim se

```

```

então interrompa
fim se
TRÁFEGO[I,J,K] ← 0
K ← K + 1
fim repita
J ← J + 1
fim repita
I ← I + 1
fim repita
    {Montagem da matriz de tráfego}
repita
leia ORIGEM, DESTINO, TIPO
se ORIGEM = 0 e DESTINO = 0 e TIPO = 0
    então interrompa
fim se
TRÁFEGO [ORIGEM, DESTINO, TIPO] ←
    TRÁFEGO [ORIGEM, DESTINO, TIPO] + 1
fim repita
    {Escrita da matriz origem × destino para cada tipo de veículo}
K ← 1
repita
    se K > 3
        então interrompa
    fim se
    escreva "VEÍCULO DO TIPO", K
    I ← 1
    repita
        se I > 4
            então interrompa
        fim se
        escreva TRÁFEGO[I,1,K]...TRÁFEGO[I,4,K]
        I ← I + 1
    fim repita
    K ← K + 1
fim repita
    {Cálculo e escrita do número de veículos originários
     {de Belo Horizonte}}
Q ← 0
J ← 1
repita
    se J > 4
        então interrompa
    fim se
    K ← 1
    repita
        se K > 3
            então interrompa
        fim se
        Q ← Q + TRÁFEGO[I,J,K]
        K ← K + 1
    fim repita
    J ← J + 1
fim repita
escreva Q
    {Cálculo e escrita do número de ônibus originários de Brasília
     {e que se destinam a Petrópolis ou Barbacena}}

```

$Q \leftarrow \text{TRÁFEGO}[4,2,3] + \text{TRÁFEGO}[4,4,3]$
 escreva Q
 fim algoritmo.

2.2.2.2. EXERCÍCIOS DE FIXAÇÃO

- 2.2.2.2.1. A variável composta X, de N linhas por quatro colunas, contém informações sobre alunos da Universidade. Os elementos da primeira, segunda, terceira e quarta colunas são, respectivamente, o número de matrícula, sexo (0 ou 1), número do curso e a média geral no curso.

Fazer um algoritmo que:

- leia o número N de alunos ($N \leq 2000$);
- leia as informações sobre os alunos;
- determine e imprima o número da matrícula do aluno de sexo 1, curso 153 que obteve a melhor média.

Supor a inexistência de empate.

- 2.2.2.2.2. Fazer um algoritmo que efetue um produto matricial. Seja A ($m \times n$) e B ($n \times m$) as matrizes-fatores, sendo $m \leq 40$ e $n \leq 70$. Deverão ser impressas as matrizes A, B e a matriz-produto obtida.

- 2.2.2.2.3. A tabela dada a seguir contém vários itens que estão estocados em vários armazéns de uma companhia. É fornecido, também, o custo de cada um dos produtos armazenados.

	PRODUTO 1 (unid.)	PRODUTO 2 (unid.)	PRODUTO 3 (unid.)
ARMAZÉM 1	1200	3700	3737
ARMAZÉM 2	1400	4210	4224
ARMAZÉM 3	2000	2240	2444
CUSTO (\$)	260,00	420,00	330,00

Fazer um algoritmo que:

- leia o estoque inicial;
- determine e imprima quantos itens estão armazenados em cada armazém;
- qual o armazém que possui a maior quantidade de produto 2 armazenado;
- o custo total de:
 - cada produto em cada armazém;
 - estoque em cada armazém;
 - cada produto em todos os armazéns.

2.3. VARIÁVEIS COMPOSTAS HETEROGÊNEAS

2.3.1. Registros

São conjuntos de dados logicamente relacionados, mas de tipos diferentes (numérico, literal, lógico)

Exemplo 2.29

Numa dada aplicação pode-se ter os seguintes dados de funcionários de uma empresa (figuras 2.1 e 2.2).

Em cada uma destas figuras, os dados estão logicamente relacionados entre si, pois constituem as informações cadastrais do mesmo indivíduo.

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

INSC 224901	NOME FULANO DE TAL		ENDERÉCOS RUA, Nº, APTO, FONE BÁO JOÃO, 237/4				CEP 30.000	CIDADE BH	UF MG
SITUAÇÃO NORMAL	DOC. MILITAR 3 04 11 024561	TÍTULO 278 MG 86121	DOCUMENTOS CPF 274965106	CART. PROFISSIONAL 365 4421321	C. IDENT. MG G 44121				
CONTA BANCÁRIA 13350 104 821	ESTADO CIVIL SOLTEIRO	CARGO PROFESSOR		SALÁRIO 2.132.240					

Fig. 2.1

INSC 66721	NOME SICRANO DE QUIL		ENDERÉCOS RUA, Nº, APTO, FONE SÃO PEDRO, 100/27				CEP 30.000	CIDADE BH	UF MG
SITUAÇÃO NORMAL	DOC. MILITAR 1 07 12 114211	TÍTULO 23A MG 44121	DOCUMENTOS CPF 666102275	CART. PROFISSIONAL 621 9942143	C. IDENT. MG L 331214				
CONTA BANCÁRIA 124510 102 081	ESTADO CIVIL CASADO	CARGO OPERADOR		SALÁRIO 2.120.732					

Fig. 2.2

Dentre os vários componentes de FICHA podem ser citados os seguintes:

Identificador da variável do componente	Tipo
NOME	literal
ESTADO CIVIL	literal
SALÁRIO	numérico

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

Registros correspondem a conjuntos de posições de memória conhecidos por um mesmo nome e individualizados por identificadores associados a cada conjunto de posições.

O registro é um caso mais geral de variável composta na qual os elementos do conjunto não precisam ser, necessariamente, homogêneos ou do mesmo tipo. O registro é constituído por componentes.

Na variável composta homogênea, a individualização de um elemento é feita através de índices, já no registro cada componente é individualizado pela explicitação de seu identificador.

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

identificador do registro	identificador do componente
---------------------------	-----------------------------

Exemplo 2.30

Supondo-se que o registro FICHA, num dado instante, contivesse os valores a seguir:

FICHA

INSCRIÇÃO	NOME	
RUA	NÚMERO	CEP
CPF		
SEXO	DATA DE NASCIMENTO	
TEM DEP.		HORAS TRAB.

09214	JOSÉ DA SILVA			
SAGRES	210	30000		
274165106-44				
M	03 02 55			
FALSO				
25,5				

FICHA.INSCRIÇÃO estaria se referenciando ao conteúdo do componente INSCRIÇÃO do registro FICHA, isto é, 09214.

2.3.1.1. DECLARAÇÃO

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

declare lista-de-identificadores registro (componentes)

onde:

declare
lista-de-identificadores

é uma palavra-chave;
são os nomes que estão sendo associados aos registros
que se deseja declarar;

componentes

são declarações e/ou identificadores de variáveis compostas,
separados por vírgula;

registro

é uma palavra-chave.

Exemplo 2.31

Declarar o registro cadastro que tem a seguinte forma:

NOME		
RUA	NÚMERO	CEP
CPF		
SEXO	HT ₁	HT ₂
HT ₃		
NASCIMENTO		
TEMDEP		

**declare CADASTRO registro (NOME, RUA literal,
NÚMERO, CEP, CPF numérico,
SEXO literal,
HT [1:3] numérico,
NASCIMENTO numérico,
TEMDEP lógico)**

Exemplo 2.32

Declarar o registro CAD que tem a seguinte forma:

NOME	
ENDERECO	
CPF	
SEXO	HT
NASCIMENTO	
TEMDEP	

onde endereço é um registro contendo o nome da rua, número e código de endereçamento postal. HT é um arranjo numérico de três elementos e TEM DEP é uma variável lógica.

```
declare CAD registro (NOME literal,  
                     ENDEREÇO,  
                     CPF numérico,  
                     SEXO literal,  
                     HT [1:3] numérico,  
                     NASCIMENTO numérico,  
                     TEMDEP lógico)
```

```
declare ENDEREÇO registro  
        (RUA literal,  
         NÚMERO, CEP numérico)
```

Comparando-se os exemplos 2.31 e 2.32, vê-se a possibilidade que existe de se aplicar a filosofia da técnica de refinamentos sucessivos à declaração dos registros.

No caso de estruturas complexas, como o do exemplo 2.32, quando na declaração do registro se encontra um outro registro, o acesso aos elementos é feito hierarquicamente. Primeiro, faz-se menção aos registros mais externos, depois aos mais internos e, finalmente, ao identificador do componente.

Exemplo 2.33

O acesso ao componente CEP do registro declarado no exemplo 2.32 é feito através da menção ao nome do registro mais externo, o CAD, passando pelo ENDEREÇO e finalizando com o identificador do componente, ou seja, CEP.

De acordo com a notação adotada, tal referência seria indicada por

CAD.ENDEREÇO.CEP

2.3.2. Conjunto de registros

Podem-se ter conjuntos de registros referenciáveis por um mesmo nome e individualizados por índices.

Exemplo 2.34

Considerando-se o registro de uma mercadoria de uma loja como descrito a seguir

CÓDIGO	NOME
PREÇO	ESTOQUE

7721	CAMISA AZUL
148200	2732

o conjunto de mercadorias da loja poderia ser agrupado numa variável composta heterogênea MERCADORIAS, onde cada elemento deste conjunto é um registro constituído por quatro componentes (CÓDIGO, NOME, PREÇO e ESTOQUE).

A referência ao i-ésimo elemento ou registro do conjunto é feito pela seguinte notação:

identificador da variável[i].identificador do registro.identificador do componente

onde:

- identificador da variável i é o nome associado a esta estrutura;
- i é um conjunto de um ou mais índices capazes de individualizar um registro do conjunto de registros;
- identificador do registro é o nome associado ao registro, que, dependendo da declaração feita, pode ser omitido;
- identificador do componente é o nome associado ao componente, cuja referência ao conteúdo está sendo feita.

2.3.2.1. DECLARAÇÃO

A declaração de conjuntos de registros tem a seguinte forma:

```
declare lista de identificadores [li1:ls1, li2:ls2, ..., lin:lsn] t
```

onde:

- declare é uma palavra-chave;
- lista de identificadores são nomes associados às variáveis compostas heterogêneas que se deseja declarar;
- li₁:ls₁, ..., li_n:ls_n são os limites dos intervalos de variação dos índices da variável, onde cada par está associado a um índice: identificador ou descrição do registro.
- t

Exemplo 2.35

Declarar a seguinte estrutura de dados

CONTAS

1	
2	
3	○
4	
5	

CLIENTE

NOME		
RUA	NÚMERO	CPF
SALDO		

declare CONTAS[1:5] CLIENTE

declare CLIENTE registro(NOME, RUA literal,

NÚMERO, CEP, SALDO numérico)

ou, équivalentement:

declare CONTAS[1:5] registro (NOME, RUA literal)
(NÚMERO, CEP, SALDO numérico)

Com uma declaração como a anterior, passa a existir a variável CONTAS, onde cada elemento do conjunto é um registro, com componentes NOME, RUA, NÚMERO, CEP e SALDO.

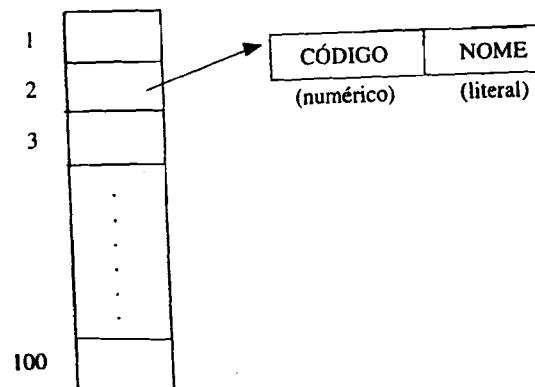
Exemplo 2.36

Dada a tabela a seguir:

CÓDIGO	NOME
1 001000	ALFAIATE
2 001050	ALMOXARIFE
3 002000	ANALISTA
4 002050	ANTROPÓLOGO
5 003000	BOMBEIRO
6 003050	CARPINTEIRO
.	.
.	.
.	.
99 198 050	TORNEIRO
100 199 000	VENDEDOR

Escrever um algoritmo que, dados 500 CÓDIGOS DE PROFISSÃO (fornecidos em 500 linhas) emita o NOME das profissões correspondentes. A tabela acima também deve ser lida.

Estrutura de dados



Algoritmo

```

Defina o tipo das variáveis
leia TABELA[1]...TABELA[100]
K ← 1
repita
    se K > 500
        então interrompa
    fim se
    leia CÓDIGO DESEJADO
    Escreva o nome da profissão
    K ← K + 1
fim repita
fim algoritmo.

```

Ref. Escreva o nome da profissão

```

I ← 0
repita
    I ← I + 1
    se TABELA[I].CÓDIGO = CÓDIGODESEJADO então I = 100
        então interrompa
    fim se
fim repita
se TABELA[I].CÓDIGO = CÓDIGODESEJADO
    então escreva CÓDIGODESEJADO,TABELA[I].NOME
    senão escreva "CÓDIGO INVÁLIDO"
fim se
fim ref.

```

Ref. Defina tipo das variáveis

```
declare TABELA[1:100] registro (CÓDIGO número,  
NOME literal)  
declare CÓDIGODESEJADO,I,K, número  
fim ref
```

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

```

{Definição do tipo das variáveis}
declare TABELA[1:100] registro (CÓDIGO numérico,
NOME literal)
declare CÓDIGODESEJADO,I,K numérico
leia TABELA[I]..TABELA[100]
K ← 1
repita
    se K > 500
        então interrompa
    fim se
    leia CÓDIGODESEJADO
        (Escrita do nome da profissão)
    I ← 0
    repita
        I ← I + 1

```

```

    se TABELA[I].CÓDIGO = CÓDIGO DESEJADO ou I = 100
        então interrompa
    fim se
    fim repita
    se TABELA[I].CÓDIGO = CÓDIGO DESEJADO
        então escreva CÓDIGO DESEJADO, TABELA[I].NOME
        senão escreva "CÓDIGO INVÁLIDO"
    fim se
    K ← K + 1
    fim repita
fim algoritmo.

```

Exemplo 2.37

Em certo município, vários proprietários de imóveis estão em atraso com o pagamento do imposto predial. Desenvolver um algoritmo que calcule e escreva o valor da multa a ser paga por esses proprietários considerando que:

- os dados de cada imóvel: identificação (literal), valor do imposto e número de meses em atraso estão à disposição numa unidade de entrada;
- as multas devem ser calculadas a partir do valor do imposto e de acordo com a seguinte tabela (também à disposição numa unidade de entrada);

Valor do imposto	% por mês em atraso
até R\$ 50,00	1
de R\$ 51,00 a R\$ 180,00	2
de R\$ 181,00 a R\$ 500,00	4
de R\$ 501,00 a R\$ 1.200,00	7
acima de R\$ 1.200,00	10

- o último registro lido, que não deve ser considerado, contém a identificação do imóvel igual a vazio;
- na saída deverão ser impressos: a identificação do imóvel, valor do imposto, meses em atraso e a multa a ser paga.

As estruturas de dados a serem adotadas para a solução do problema são:

TABELA

LIMITES		PERCENTUAL
DE	ATÉ	
0	50,00	1
51,00	180,00	2
181,00	500,00	4
501,00	1.200,00	7
1.201,00		10

Variável composta
homogênea de 5 linhas
por 3 colunas

Para os dados de cada imóvel, será adotado o seguinte registro (variável composta heterogênea)

IMÓVEL

IDENTIFIC	IMPOSTO	MESESAT
↓ literal	↓ número	↓ número

Algoritmo

```

Defina tipo das variáveis
leia TABELA[1,1]...TABELA[5,3]
repita
    leia IMÓVEL
    se IMÓVEL.IDENTIFIC = "VAZIO"
        então interrompa
    fim se
    Calcule MULTA
    Escreva resultados obtidos
fim repita
fim algoritmo.

```

Ref. Calcule MULTA

```

I ← 6
repita
    I ← I - 1
    se IMÓVEL.IMPOSTO ≥ TABELA[I,1] ou I = 1
        então interrompa
    fim se
fim repita
se IMÓVEL.IMPOSTO ≥ TABELA[1,1]
    então MULTA ← TABELA[I,3] × IMÓVEL.IMPOSTO/100 × IMÓVEL
        MESESAT
    fim se
fim ref

```

Ref. Escreva resultados obtidos

```

escreva IMÓVEL, MULTA
fim ref.

```

Ref. Defina tipo das variáveis

```

declare TABELA[1:5,1:3] número
declare IMÓVEL registro (IDENTIFIC literal,
                           IMPORTO número,
                           MESESAT número)
declare I, MULTA número
fim ref.

```

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, tem-se:

Algoritmo

(Definição do tipo das variáveis)

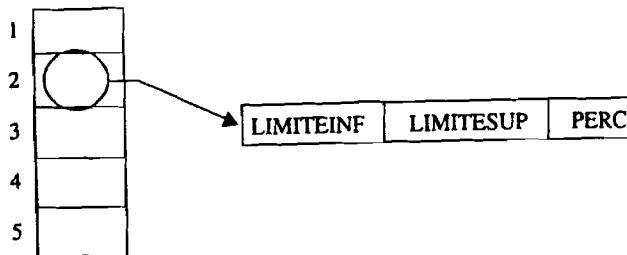
```

declare TABELA[1:5;1:3] numérico
declare IMÓVEL registro (IDENTIFIC literal,
                         IMPOSTO numérico,
                         MESESAT numérico)
declare I, MULTA numérico
leia TABELA
repita
    leia IMÓVEL
    se IMÓVEL.IDENTIFIC = "VAZIO"
        então interrompa
    fim se
    {Cálculo da MULTA}
    I ← 6
    repita
        I ← I - 1
        se IMÓVEL.IMPOSTO ≥ TABELA[I,1] ou I = 1
            então interrompa
        fim se
    fim repita
    se IMÓVEL.IMPOSTO ≥ TABELA[I,1]
        então MULTA ← TABELA[I,3] × IMÓVEL.IMPOSTO/100 ×
              IMÓVEL.MESESAT
    fim se
    {Escrita dos resultados obtidos}
    escreva IMÓVEL,MULTA
fim repita
fim algoritmo.

```

Pode-se observar que, para efeito de documentação e clareza do algoritmo, a estrutura de dados TABELA, embora variável composta homogênea, poderia ser declarada da seguinte forma:

TABELA



```

declare TABELA[1:5] REG
declare REG registro (LIMITEINF,
                     LIMITESUP,
                     PERC numérico)

```

O algoritmo, além das declarações das estruturas de dados vistas acima, sofreria as seguintes transformações:

TABELA[I,1] seria substituído por TABELA[I].REG.LIMITEINF
TABELA[I,2] seria substituído por TABELA[I].REG.LIMITESUP
TABELA[I,3] seria substituído por TABELA[I].REG.PERC

Exemplo 2.38

O diagrama abaixo representa o pátio de um depósito de uma empresa de construções, que armazena os seguintes materiais: cimento, areia, tubos, blocos, madeira, cal e saibro:

20	cimento	00		80	saibro	40	madeiras
00		23	blocos	00		30	cimento
00		00		30	areia	25	areia
53	tubos	15		45	tubos	20	cal

Em cada espaço do depósito estão colocados a quantidade do material e o nome correspondente. Assim, 30 cimento significa que naquele local existem 30 sacos de cimento.

Escrever um algoritmo capaz de contar quantos elementos de cada material existem no pátio (observe que pode haver mais de um local com o mesmo material).

Sabe-se que os dados estão dispostos em linhas.

O algoritmo deve imprimir os resultados conforme o diagrama a seguir:

PRODUTO	QUANTIDADE
CIMENTO	XXXXX
AREIA	XXXXX
TUBOS	XXXXX
BLOCOS DE CONCRETO	XXXXX
MADEIRA	
CAL	
SAIBRO	

Estrutura de dados

PÁTIO	1	2	3	4	MATERIAL	QUANTIDADE	NOME
1							
2							
3							
4							

PRODUTOS

CIMENTO	AREIA	TUBOS	BLOCOS	MADEIRA	CAL	SAIBRO
1	2	3	4	5	6	7

Algoritmo

Defina tipo das variáveis
Atribua valores iniciais necessários

leia PÁTIO[1,1] ...PÁTIO[4,4]
 leia PRODUTO[1] ...PRODUTO[7]
 Determine a quantidade de cada material existente no pátio
 Escreva a quantidade calculada
 fim algoritmo.

Ref. Determine a quantidade de cada material existente no pátio

```

I ← 1
repita
  se I > 7
    então interrompa
  fim se
  J ← 1
  repita
    se J > 4
      então interrompa
    fim se
    K ← 1
    repita
      se K > 4
        então interrompa
      fim se
      se PÁTIO[J,K].MATERIAL.NOME = PRODUTO[I]
        então OCORRÊNCIA[I] ← OCORRÊNCIA[I] +
          PÁTIO[J,K].MATERIAL.QUANTIDADE
      fim se
      K ← K + 1
    fim repita
    J ← J + 1
  fim repita
  I ← I + 1
fim repita
fim ref.
  
```

Ref. Escreva a quantidade calculada

```

I ← 1
repita
  se I > 7
    então interrompa
  fim se
  escreva PRODUTO[I], OCORRÊNCIA[I]
  I ← I + 1
fim repita
fim ref.
  
```

Ref. Atribua valores iniciais necessários

```

I ← 1
repita
  se I > 7
    então interrompa
  fim se
  OCORRÊNCIA[I] ← 0
  
```

$I \leftarrow I + 1$
 fim repita
 fim ref.

Ref. Defina tipo das variáveis
declare PÁTIO[1:4,1:4] MATERIAL
declare MATERIAL registro (QUANTIDADE numérico,
NOME literal)
declare OCORRÊNCIA[1:7] numérico
declare PRODUTO[1:7] literal
declare I,J,K numérico
 fim ref.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

{Definição do tipo das variáveis}
declare PÁTIO[1:4,1:4] MATERIAL
declare MATERIAL registro (QUANTIDADE numérico,
NOME literal)
declare OCORRÊNCIA[1:7] numérico
declare PRODUTO[1:7] literal
declare I,J,K numérico
 {Atribuição dos valores iniciais necessários}

```

I ← 1
repita
  se I > 7
    então interrompa
  fim se
  OCORRÊNCIA[I] ← 0
  I ← I + 1
fim repita
leia PÁTIO[1,1]...PÁTIO[4,4]
leia PRODUTO[1]...PRODUTO[7]
{Determinação da quantidade de cada material}
{existente no pátio}
I ← 1
repita
  se I > 7
    então interrompa
  fim se
  J ← 1
  repita
    se J > 4
      então interrompa
    fim se
    K ← 1
    repita
      se K > 4
        então interrompa
      fim se
      se PÁTIO[J,K].MATERIAL.NOME = PRODUTO[I]
        então OCORRÊNCIA[I] ← OCORRÊNCIA[I] +
          PÁTIO[J,K].MATERIAL.QUANTIDADE
      fim se
      K ← K + 1
    fim repita
    J ← J + 1
  fim repita
  I ← I + 1
fim repita
  
```

PÁTIO[J,K].MATERIAL.QUANTIDADE

```

    fim se
    K ← K + 1
fim repita
J ← J + 1
fim repita
I ← I + 1
fim repita
    (Escrita da quantidade calculada)

I ← 1
repita
    se I > 7
        então interrompa
    fim se
    escreva PRODUTO[I],OCORRÊNCIA[I]
    I ← I + 1
fim repita
fim algoritmo.

```

Exemplo 2.39

Escriver um algoritmo que leia um conjunto de linhas contendo, cada uma, uma data:

dia/mês/ano (flag = 00/jan/0000)

verifique se a data está correta (se o mês é um dos meses do ano e se o dia está de acordo com o mês) e se o ano é igual a 1985. Se a data não estiver correta, imprimir "DATA INCORRETA" e os valores de dia, mês e ano. Se a data estiver correta e o ano for 1985, verificar qual é a diferença (em dias) para com a data 21/jun/1985, e imprimir esta diferença (positiva ou negativa).

DATA FORNECIDA = 25/jun/1985 — Diferença = + 4
 DATA FORNECIDA = 20/jun/1985 — Diferença = -1
 DATA FORNECIDA = 31/fev/1985 — Data incorreta

Estrutura de dados
 MESES (variável composta homogênea)

JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
1	2	3	4	5	6	7	8	9	10	11	12

DIAS DO MÊS (variável composta homogênea)

31	28	31	30	31	30	31	31	30	31	30	31
1	2	3	4	5	6	7	8	9	10	11	12

DATA (variável composta não homogênea)

DIA	BARRA 1	MÊS	BARRA 2	ANO

Algoritmo

Defina tipo das variáveis
 leia MESES[1]...MESES[12]
 leia DIASDOMÊS[1]...DIASDOMÊS[12]

Calcule TOTALDIAS1

```

repita
    leia DATA
    se DATA.DIA = 0 e DATA.BARRA1 = "/" e DATA.MÊS = "JAN"
        e DATA.BARRA2 = "/" e DATA.ANO = 0
        então interrompa
    fim se
    se DATA.ANO = 1985
        então verifique se DATA está correta
        senão escreva "DATA FORNECIDA", DATA, "DATA INCORRETA"
    fim se
    fim repita
fim algoritmo.

```

Ref. Calcule TOTALDIAS1

```

TOTALDIAS1 ← 21
I ← 1
repita
    se I = 6
        então interrompa
    fim se
    TOTALDIAS1 ← TOTALDIAS1 + DIASDOMÊS[I]
    I ← I + 1
fim repita
fim ref.

```

Ref. Verifique se DATA está correta

```

Verifique se DATA.MÊS está correto
se MÊSVÁLIDO
    então Verifique se DATA.DIA está correto
    senão escreva "DATA FORNECIDA =", DATA, "DATA INCORRETA"
fim se
fim ref.

```

Ref. Verifique se DATA.MÊS está correto

```

M ← 0
repita
    M ← M + 1
    se DATA.MÊS = MESES[M] ou M = 12
        então interrompa
    fim se
fim repita
se DATA.MÊS = MESES[M]
    então MÊS VÁLIDO ← verdadeiro
    senão MÊS VÁLIDO ← falso
fim se
fim ref.

```

Ref. Verifique se DATA.DIA está correto

```

se DATA.DIA > 0 e DATA.DIA ≤ DIASDOMÊS[M]
    então Calcule a DIFERENÇA para 21/jun/1985

```

```

    DATA FORNECIDA ,DATA,"DIFERENÇA =",
    DIFERENÇA
    sendo escreva "DATA FORNECIDA =", DATA, "DATA INCORRETA"
fim se
fim ref.

```

Ref. Calcule a DIFERENÇA para 21/jun/85

```

TOTALDIAS2 ← DATA.DIA
I ← 1
repita
  se I = M
    então interrompa
  fim se
  TOTALDIAS2 ← TOTALDIAS2 + DIASDOMÊS[I]
  I ← I + 1
fim repita
DIFERENÇA ← TOTALDIAS1 - TOTALDIAS2
fim ref.

```

Ref. Defina tipo das variáveis

```

declare DATA registro (DIA numérico,
                      BARRA1 literal,
                      MÊS literal,
                      BARRA2 literal,
                      ANO numérico)
declare MESES[1:12] literal
declare DIASDOMÊS[1:12] numérico
declare I,M,TOTALDIAS1,TOTALDIAS2,DIFERENÇA numérico
declare MÊSVÁLIDO lógico
fim ref.

```

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

```

{Definição do tipo das variáveis}
declare DATA registro (DIA numérico,
                      BARRA1 literal,
                      MÊS literal,
                      BARRA2 literal,
                      ANO numérico)
declare MESES[1:12] literal
declare DIASDOMÊS[1:12] numérico
declare I,M,TOTALDIAS1,TOTALDIAS2,DIFERENÇA numérico
declare MÊSVÁLIDO lógico
leia MESES[1]...MESES[12]
leia DIASDOMÊS[1]...DIASDOMÊS[12]
(Cálculo de TOTALDIAS1)
TOTALDIAS1 ← 21
I ← 1
repita
  se I = 6
    então interrompa
  fim se
  TOTALDIAS2 ← TOTALDIAS1 + DIASDOMÊS[I]
  I ← I + 1
fim repita
DIFERENÇA ← TOTALDIAS1 - TOTALDIAS2
escreva "DATA FORNECIDA =", DATA,
        "DIFERENÇA =", DIFERENÇA
senão escreva "DATA FORNECIDA =", DATA,
        "DATA INCORRETA"
fim se
fim repita
fim algoritmo.

```

```

fim se
TOTALDIAS1 ← TOTALDIAS1 + DIASDOMÊS[I]
I ← I + 1
fim repita
repita
leia DATA
se DATA.DIA = 0 e DATA.BARRA1 = "/" e DATA.MÊS = "JAN" e
  DATA.BARRA2 = "/" e DATA.ANO = 0
  então interrompa
fim se
se DATA.ANO = 1985
  {Verificação da correção da data}
então M ← 0
repita
  M ← M + 1
  se DATA.MÊS = MESES[M] ou M = 12
    então interrompa
  fim se
fim repita
se DATA.MÊS = MESES[M]
  então MÊSVÁLIDO ← verdadeiro
  senão MÊSVÁLIDO ← falso
fim se
se MÊSVÁLIDO
  então se DATA.DIA > 0 e DATA.DIA ≤ DIASDOMÊS[M]
    então TOTALDIAS2 ← DATA.DIA
    I ← 1
    repita
      se I = M
        então interrompa
      fim se
      TOTALDIAS2 ← TOTALDIAS2 +
                    DIASDOMÊS[I]
      I ← I + 1
    fim repita
    DIFERENÇA ← TOTALDIAS1 -
                  TOTALDIAS2
    escreva "DATA FORNECIDA =", DATA,
            "DIFERENÇA =", DIFERENÇA
    senão escreva "DATA FORNECIDA =", DATA,
            "DATA INCORRETA"
  fim se
  senão escreva "DATA FORNECIDA =", DATA,
            "DATA INCORRETA"
  fim se
fim se
fim repita
fim algoritmo.

```

2.4. ARQUIVOS

Como já foi dito, um aspecto fundamental na formulação de um algoritmo é a estrutura de dados usada para representar as informações do problema em resolução. Variáveis simples ou compostas, e re-

registros são as entidades normalmente usadas para declarar estruturas de dados alocadas no mesmo ambiente do algoritmo.

Entretanto, nem sempre é possível, ou desejável, construir e/ou manter uma estrutura de dados no ambiente do algoritmo. Considerando a memória do computador como o ambiente comum ao algoritmo e à estrutura de dados, não é difícil perceber, por exemplo, que pode não existir espaço para armazenar um volume de dados muito grande. Em outra situação poderá haver necessidade de armazenamento de um conjunto de informações por um período de tempo longo (da ordem de minutos, horas e/ou, anos), não fazendo sentido, portanto, mantê-lo na memória principal por ser este um recurso caro e limitado.

A alternativa para estruturas de dados que não podem ou não necessitam estar alocadas no ambiente do algoritmo é o arquivo. Este é uma estrutura de dados conhecida pelo algoritmo, mas fisicamente alojada em um meio secundário de armazenamento, sendo mais comum o uso de meios magnéticos como fitas, discos ou disquetes.

2.4.1. Conceito de arquivo

Arquivo é um conjunto de registros armazenados em um dispositivo de memória secundária. Registro é um conjunto de unidades de informação logicamente relacionadas. Cada unidade de informação constitui um campo do registro.

Considere-se a cédula de identidade, a seguir:

INSTITUTO DE IDENTIFICAÇÃO	
REGISTRO GERAL A-9876	
NOME: Fulano de Tal	
PAI: Sicrano de Tal	
FILIAÇÃO:	MÃE: Beltrana de Tal
DESCOBERTO-MG	29/02/1900
NATURALIDADE	NASCIMENTO

Nela encontram-se alguns dados referentes a um cidadão: nome, filiação, registro geral, naturalidade, nascimento etc. O conjunto de informações apresentado por estes dados pode formar um registro:

R.G.	Nome	Pai	Mãe	Naturalidade	Nascimento
A-9876	Fulano de Tal	Sicrano de Tal	Beltrana de Tal	Descoberto	29/02/1900

A relação lógica entre estas informações é a sua associação a uma pessoa. O armazenamento do conjunto formado pelas regiões contendo as informações das carteiras de identidade dos membros de uma comunidade, por exemplo, poderá formar um arquivo.

A-9876	Fulano de Tal	Sicrano de Tal	Beltrana de Tal	Descoberto	29/02/1900
B-3686	José da Silva	João da Silva	Maria Silva	Belém	30/03/1944
C-3384	Maria da Glória	Afonso Mendes	Glória Maria	Manaus	03/04/1950
D-6666	Márcia Lopes	Mário Lopes	Sílvia Lopes	Brasília	15/06/1958

"ARQUIVO DE IDENTIDADES"

O registro conforme apresentado neste capítulo é denominado registro lógico ou, simplesmente, registro. Na verdade, o arquivo é constituído de um conjunto de registros físicos que, por questão de eficiência, é um conjunto de um ou mais registros lógicos. Esta questão, entretanto, não interessa ao projetista de algoritmos, que poderá continuar pensando no arquivo como uma coleção de registros. As questões relativas aos registros físicos, e a outros conceitos, poderão ser resolvidas quanto à implementação do algoritmo em uma linguagem de programação.

O fato de o arquivo ser armazenado em uma memória secundária o torna independente de qualquer algoritmo. Isto é, um arquivo pode ser criado, consultado, processado e eventualmente removido por algoritmos distintos.

Sendo o arquivo uma estrutura fora do ambiente do algoritmo, para que esse tenha acesso aos dados do arquivo é necessária a operação de leitura do registro no arquivo. Analogamente, para que o algoritmo coloque alguma informação no arquivo, é necessária a operação de escrita do registro no arquivo. Estas operações de acesso ao arquivo são básicas nos algoritmos, e os usuários devem procurar minimizá-las, pois são muito demoradas e podem tornar o algoritmo ineficiente. A escolha e o conhecimento das organizações de arquivo ajuda na minimização destas operações.

2.4.2. Organização de arquivos

As operações básicas que podem ser feitas em um arquivo através de um algoritmo são: obtenção de um registro do arquivo, inserção de um novo registro, modificação ou exclusão de um registro. Dependendo do tipo de problema, estas operações poderão ocorrer em maior ou menor número de vezes.

A disposição dos registros no arquivo pode favorecer determinadas operações em detrimento de outras. O conhecimento das possibilidades de organização dos registros nos arquivos permite ao projetista de algoritmos escolher aquela que seja mais adequada à solução do seu problema em termos de eficiência e eficiência.

Basicamente, existem duas possibilidades de organização de arquivos: a organização seqüencial, na qual os registros são obtidos ou inseridos no arquivo em ordem seqüencial, e a organização direta, em que o acesso do registro é feito em ordem aleatória. Outros tipos de organização são, na verdade, variações destas e, ainda que possam ser importantes em algumas aplicações, não serão tratados neste livro, em nome da simplicidade.

2.4.3. Declaração

O acesso a um arquivo dentro do algoritmo é feito através da leitura e escrita de registros. No algoritmo, o arquivo deve ser declarado e aberto, antes que tal acesso possa ser feito. No final do algoritmo, ou quando houver necessidade, o arquivo deve ser fechado.

A declaração de um arquivo é feita através da seguinte especificação:

declare lista-de-identificadores arquivo organização de nome

onde:

- | | |
|--------------------------|--|
| declare | é uma palavra-chave; |
| lista-de-identificadores | são nomes que serão usados pelo algoritmo para referenciar os arquivos; |
| arquivo | é uma palavra-chave; |
| organização | indica o tipo de organização do arquivo, que pode ser <i>seqüencial</i> ou <i>direta</i> ; |
| de | é uma palavra-chave; |
| nome | é o nome do registro que será usado para se ter acesso ao arquivo. |

Exemplo 2.40

declare AGENDA arquivo seqüencial de ENDEREÇO

declare ENDEREÇO registro (NOME, RUA literal,

NÚMERO numérico,

CIDADE literal)

Neste exemplo, foi declarado um arquivo de nome AGENDA, composto de registros de nome EN-DEREÇO. Cada registro é composto dos campos NOME, RUA, NÚMERO e CIDADE. Esquematicamente, este arquivo pode ser representado pela figura 2.3.

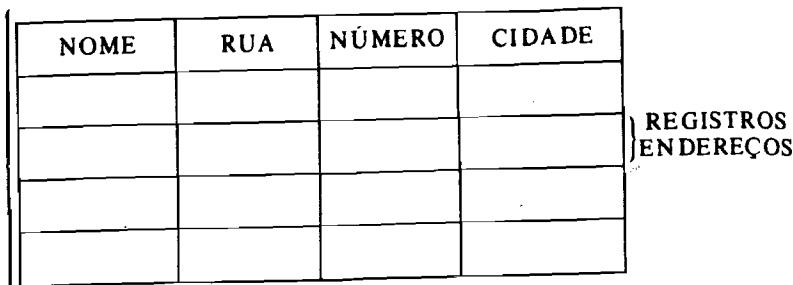


Fig. 2.3

Exemplo 2.41

declare COMPRA, PEDIDOS arquivo sequencial de PAPELETA
declare PAPELETA registro (QUANTIDADE numérico,
ESPECIFICAÇÃO literal)

O exemplo mostra a declaração de dois arquivos sequenciais com os nomes de COMPRA e PEDIDOS. Ambos possuem registros idênticos ao registro PAPELETA.

É bom lembrar que a declaração de registro define no ambiente do algoritmo um espaço para o mesmo. O fato de se declarar um arquivo associando-o a um registro é apenas para definir os componentes dos registros do arquivo. Eventualmente, pode-se ter acesso a registros de um arquivo utilizando-se um registro cuja declaração não foi explicitamente associada ao arquivo. Veja o exemplo 2.47.

2.4.3.1. EXERCÍCIOS DE FIXAÇÃO

○ 2.4.3.1.1. Declarar um arquivo com organização seqüencial para registros com o seguinte formato:

NÚMERO DA CONTA	NOME DO CLIENTE	SALDO	DATA DA ÚLTIMA OPERAÇÃO

○ 2.4.3.1.2. Declarar dois arquivos seqüenciais usando o mesmo registro dado abaixo:

TELEFONE	CIDADE	TEMPO DE LIGAÇÃO

○ 2.4.3.1.3. Escrever, agora, um trecho de algoritmo que compare um registro de um arquivo com um registro do outro arquivo, campo a campo. Caso os registros sejam iguais, atribuir ao campo "tempo de ligação" do registro correspondente ao primeiro arquivo a soma dos tempos de ligação dos dois registros, e atribuir zero ao campo de "tempo de ligação" do registro correspondente ao segundo arquivo.

2.4.4. Abertura de arquivo

A declaração do arquivo é a definição, para o algoritmo, do modelo e dos nomes que estarão associados à estrutura de dados, isto é, ao arquivo. A associação deste modelo ao arquivo físico é uma questão que, no algoritmo, pode ser resolvida pelo comando:

abra lista-de-identificadores-de-arquivo tipo-de-utilização

onde:

abra
lista-de-identificadores-de-arquivo
tipo-de-utilização

é uma palavra-chave;
são os nomes declarados dos arquivos e pelos quais o algoritmo os referenciará;
especifica se o arquivo será usado somente para leitura, somente para escrita ou ambos, simultaneamente.

Exemplo 2.42

- a) abra AGENDA leitura;
- b) abra AGENDA escrita;
- c) abra AGENDA.

No exemplo 2.42a, o arquivo AGENDA foi aberto somente para leitura. Analogamente, no exemplo 2.42b, o mesmo arquivo foi aberto somente para escrita. No último exemplo, o arquivo foi aberto para leitura e/ou escrita.

2.4.4.1. EXERCÍCIO DE FIXAÇÃO

○ Escrever o comando de abertura para o arquivo declarado no exercício 2.4.3.1.1., de modo a permitir a atualização do registro, isto é, a leitura, modificação dos campos e escrita do registro no mesmo arquivo.

2.4.5. Fechamento de arquivo

Para se desfazer a associação entre o modelo e o arquivo físico, usa-se o comando:

feche lista-de-identificadores-de-arquivos

onde:

feche
lista-de-identificadores-de-arquivos

é uma palavra-chave;
são os nomes declarados dos arquivos e pelos quais o algoritmo os referencia.

Utiliza-se este comando, geralmente, no fim do algoritmo ou quando se deseja alterar o tipo de utilização do arquivo.

Exemplo 2.43

Algoritmo

```
declare NOTAS arquivo sequencial de DADOS
declare DADOS registro (MATRÍCULA numérico,
NOMEALUNO literal,
NOTATOTAL numérico)
```

abra NOTAS escrita

feche NOTAS

abra NOTAS leitura

feche NOTAS

fim algoritmo.

Neste exemplo é declarado um arquivo seqüencial de nome NOTAS, composto do registro DADOS. Inicialmente, o arquivo é aberto para escrita e, após alguns comandos, é fechado e, em seguida, é aberto para leitura. Ao fim do algoritmo, o arquivo é fechado.

Os formatos para leitura e escrita de um arquivo são dependentes do tipo de organização do arquivo e por isto serão vistos nos itens seguintes.

2.4.6. Organização seqüencial

A principal característica da organização seqüencial é a de que os registros são armazenados contiguamente, isto é, um após o outro. Como consequência, o acesso aos registros do arquivo, tanto na leitura quanto na escrita, são feitos seqüencialmente, ou seja, a leitura de um registro só é possível após a leitura de todos os registros anteriores e a escrita de um registro só é feita após o último registro.

As operações de entrada e saída em um arquivo de organização seqüencial são indicadas nos algoritmos pelos comandos dados a seguir:

2.4.6.1. COMANDO DE ENTRADA

Sua forma geral é dada por:

```
leia nome-do-arquivo . nome-do-registro
```

Onde:

leia	é uma palavra-chave;
nome-do-arquivo	é o nome do arquivo declarado no algoritmo;
nome-do-registro	é o nome do registro que será usado para armazenar, no ambiente do algoritmo, o registro do arquivo, após a leitura.

2.4.6.2. COMANDO DE SAÍDA

SUA FORMA GERAL É DADA POR:

```
escreva nome-do-arquivo nome-do-registro
```

Onde:

escreva	é uma palavra-chave;
nome-do-arquivo	é o nome do arquivo declarado no algoritmo;
nome-do-registro	é o nome do registro usado para armazenar no ambiente do algoritmo o registro do arquivo, antes da escrita.

Exemplo 2.44

Supondo-se a existência de um arquivo A, com registros com campos de NOME e SALÁRIO, o algoritmo a seguir cria um arquivo B, através da cópia dos registros de A.

Algoritmo

```

declare A, B arquivo seqüencial de T
declare T registro (NOME literal,
                   SALÁRIO numérico)

abra A leitura
abra B escrita
repita
    (Leitura do arquivo)
    leia A.T
    (Teste para detectar fim do arquivo A)
    se A.FDA
        então interrompa
    fim se
fim repita

```

(Escrita do registro T no arquivo B)

Escreva B.T

```

fim repita
feche A
feche B
fim algoritmo.

```

Neste exemplo surgiu a necessidade de se determinar a leitura do último registro do arquivo. Para resolver esta questão, que será comum na elaboração de algoritmos, supõe-se a existência de uma variável lógica associada ao arquivo, de nome FDA (Fim de Arquivo), que assumirá o valor verdadeiro quando se tentar ler um registro após a leitura do último.

Não há necessidade de se declarar esta variável, já que ela é um atributo que está associado ao arquivo. Este tipo de acesso é largamente utilizado pelas linguagens mais recentes, para resolver alguns problemas análogos, tais como condições de erro, registro não encontrado etc. O sistema operacional encarrega-se de atualizar estes atributos a cada acesso do arquivo, de tal forma que o usuário possa testá-los de acordo com as suas necessidades.

Um arquivo seqüencial pode ser implementado em qualquer tipo de dispositivo. Inclusive, pode-se tratar uma impressora, a tela de um terminal, uma leitora de cartões ou um teclado como um arquivo seqüencial, respeitadas as características de cada um quanto à capacidade de leitura e/ou escrita. Fitas magnéticas, discos e disquetes são os meios mais comuns para tais arquivos.

Exemplo 2.45

Escriver um algoritmo para criar um arquivo seqüencial, com dados de uma agenda, sendo que os dados serão lidos a partir de um teclado. O registro de entrada do arquivo possui os seguintes campos:

NOME	TELEFONE	LOGRADOURO	NÚMERO	CIDADE	ESTADO
------	----------	------------	--------	--------	--------

Algoritmo

```

Defina as estruturas de dados
Abra arquivos
Leia dados e crie arquivos
Feche arquivos
fim algoritmo

```

Ref. Defina as estruturas de dados

```

declare TECLADO, (arquivo do qual serão lidos os dados)
AGENDA {arquivo onde serão armazenados os dados}
arquivo seqüencial de ENDEREÇO
declare ENDEREÇO registro (NOME literal,
                           TELEFONE numérico,
                           LOGRADOURO literal,
                           NÚMERO numérico,
                           CIDADE, ESTADO literal)
fim ref.

```

Ref. Abra arquivos

```

abra TECLADO leitura
abra AGENDA escrita
fim ref.

```

Ref. Leia dados e crie arquivos

```
repita
    leia TECLADO.ENDEREÇO
    se TECLADO.FDA
        então interrompa
    fim se
    escreva AGENDA.ENDEREÇO
fim repita
fim ref.
```

Ref. Feche arquivos

```
feche TECLADO
feche AGENDA
fim ref.
```

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo

{Definição das estruturas de dados}

```
declare TECLADO, {arquivo do qual serão lidos os dados}
        AGENDA {arquivo onde serão armazenados os dados}
        arquivo seqüencial de ENDEREÇO
```

```
declare ENDEREÇO registro (NOME literal,
                            TELEFONE numérico,
                            LOGRADOURO literal,
                            NÚMERO numérico,
                            CIDADE,ESTADO literal)
```

{Abertura dos arquivos}

```
abra TECLADO leitura
abra AGENDA escrita
{Leitura de dados e criação de arquivos}
```

```
repita
    leia TECLADO.ENDEREÇO
    se TECLADO.FDA
        então interrompa
    fim se
    escreva AGENDA.ENDEREÇO
fim repita
```

{Fechamento dos arquivos}

```
feche TECLADO
feche AGENDA
fim algoritmo.
```

Exemplo 2.46

Uma companhia resolveu diminuir sua folha de pagamentos. Para tal, mandou criar um arquivo, a partir do arquivo cadastro da empresa, com todos os funcionários que recebam mais de 30 salários mínimos. Sabendo-se que os registros possuem os campos mostrados na figura abaixo, escrever um algoritmo para criar o arquivo pedido.

NOME DO FUNCIONÁRIO	CARGO	SALÁRIO
---------------------	-------	---------

Algoritmo

Defina a estrutura de dados

Abra arquivos

Leia salário mínimo

Crie arquivo de funcionários acima de 30 salários

Feche arquivos

fim algoritmo.

Ref. Defina a estrutura de dados

```
declare CADASTRO, {arquivo com dados de todos os funcionários}
        BEMPAGOS {arquivo dos funcionários acima de 30 salários}
```

arquivo seqüencial de DADOS

```
declare DADOS registro (FUNCIONÁRIOS,CARGO literal,
                        SALÁRIO numérico)
```

```
declare ENTRADA {arquivo de onde será lido o salário mínimo}
arquivo seqüencial de REG
```

```
declare REG registro (SALÁRIOMÍNIMO numérico)
```

fim ref.

No refinamento anterior, além dos arquivos CADASTRO e BEMPAGOS, foi declarado o arquivo ENTRADA, que será usado para a leitura do valor do salário mínimo. Este arquivo pode ser, por exemplo, o teclado de um terminal.

Ref. Abra arquivos

abra CADASTRO leitura

abra BEMPAGOS escrita

abra ENTRADA leitura

fim ref.

Ref. Leia salário mínimo

leia ENTRADA.REG

fim ref.

Ref. Crie arquivo de funcionários acima de 30 salários

repita

leia CADASTRO.DADOS

se CADASTRO.FDA

então interrompa

fim se

Escreva registro de funcionários acima de 30 salários

fim repita

fim ref.

Ref. Escreva registro de funcionários acima de 30 salários

se CADASTRO.DADOS.SALÁRIO > 30 × REG.SALÁRIOMÍNIMO

então escreva BEMPAGOS.DADOS

fim se

fim ref.

Ref. Feche arquivos
Feche CADASTRO
Feche BEMPAGOS
Feche ENTRADA
fim ref.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo {Criação de um arquivo com dados selecionados}

{Definição da estrutura de dados}

declare CADASTRO, {arquivo com dados de todos os funcionários}
BEMPAGOS {arquivo dos funcionários acima de 30 salários}
arquivo seqüencial de DADOS

declare DADOS registro (FUNCIONÁRIOS,CARGO literal,
SALÁRIO número)

declare ENTRADA {arquivo de onde será lido o salário mínimo}
arquivo seqüencial de REG

declare REG registro (SALÁRIO MÍNIMO número)
(Abertura de arquivos)

abra CADASTRO **leitura**
abra BEMPAGOS **escrita**
abra ENTRADA **leitura**
(Leitura do salário mínimo)

leia ENTRADA.REG
(Criação de arquivos de funcionários acima de 30)
(salários)

repita
 leia CADASTRO.DADOS
 se CADASTRO.FDA
 então interrompa
 fim se
 se CADASTRO.DADOS.SALÁRIO > 30 × REG.SALÁRIO MÍNIMO
 então escreva BEMPAGOS.DADOS
 fim se
fim repita

(Fechamento de arquivos)

feche CADASTRO
feche BEMPAGOS
feche ENTRADA

fim algoritmo.

Exemplo 2.47

Uma instituição de pesquisa recolheu amostras em três regiões a respeito do nível de vida da população dessas regiões. Cada amostra constitui um registro com os seguintes componentes: sexo, idade, salário, estado civil, número de dependentes, valor do patrimônio, quantidade de calorias absorvidas por dia, grau de instrução.

Em cada região, os dados foram armazenados em um arquivo seqüencial, sendo os registros colocados em ordem crescente de idade.

Escrever um algoritmo que intercale estes arquivos de modo que o arquivo resultante permaneça ordenado.

Solução:

A intercalação de arquivos é um problema muito comum em processamento de arquivos seqüenciais. A solução aqui apresentada certamente servirá de referência para problemas semelhantes.

Uma característica importante, e que tornará o algoritmo mais eficiente, é a de que os três arquivos estão ordenados segundo a idade do entrevistado.

A idéia básica do algoritmo consiste, inicialmente, na leitura de um registro de cada arquivo. Em seguida, seleciona-se entre os três registros lidos aquele que possui a menor idade, escrevendo-o no arquivo de saída. Lê-se, um novo registro do arquivo correspondente ao registro que foi gravado e repete-se o processo de seleção, escrita e leitura, até que os três arquivos tenham sido esgotados. Ter-se-á, então, o arquivo de saída com os registros intercalados e ordenados.

Algoritmo

Defina as estruturas de dados
Abra arquivos
Leia um registro de cada arquivo
Gere o arquivo de saída
Feche arquivos
fim algoritmo.

Ref. Defina as estruturas de dados

declare SAÍDA {arquivo onde serão escritos os registros intercalados}
REGIÃO1, REGIÃO2, REGIÃO3 {arquivos de entrada por região}
arquivo seqüencial de PESQUISA1

declare PESQUISA1, PESQUISA2, PESQUISA3 {um registro por arquivo}
registro (IDADE,SALÁRIO,NDEPENDENTES,PATRIMÔNIO,
CALORIAS número,
SEXO,ESTADOCIVIL,INSTRUÇÃO literal)

declare NARQUIVO, {indica a qual arquivo pertence o registro com menor idade}
MENORIDADE número

fim ref.

Observe-se na declaração de arquivos que todos foram definidos como tendo o registro do formato do registro PESQUISA1. Porém, foram declarados outros registros (PESQUISA2 e PESQUISA3) que serão usados para manter no ambiente do algoritmo os registros lidos dos arquivos REGIÃO2 e REGIÃO3, respectivamente. Também o arquivo saída usará estes mesmos registros para escrita.

Ref. Abra arquivos

abra SAÍDA **escrita**
abra REGIÃO1,REGIÃO2,REGIÃO3 **leitura**
fim ref.

Ref. Leia um registro de cada arquivo

leia REGIÃO1.PESQUISA1
leia REGIÃO2.PESQUISA2
leia REGIÃO3.PESQUISA3
fim ref.

Ref. Gere o arquivo de saída

repita
 se REGIÃO1.FDA & REGIÃO2.FDA & REGIÃO3.FDA
 então interrompa
 fim se
Inicie variável para seleção do registro com a menor idade

Selecionar o registro com menor idade
Escreva no arquivo o registro selecionado e leia o próximo

fim repita
fim ref.

Ref. Inicie variável para seleção do registro com a menor idade

MENORIDADE ← 9999

fim ref.

Ref. Selecionar o registro com menor idade
se PESQUISA1.IDADE < MENORIDADE e não REGIÃO1.FDA

então MENORIDADE ← PESQUISA1.IDADE
NARQUIVO ← 1

fim se
se PESQUISA2.IDADE < MENORIDADE e não REGIÃO2.FDA

então MENORIDADE ← PESQUISA2.IDADE
NARQUIVO ← 2

fim se
se PESQUISA3.IDADE < MENORIDADE e não REGIÃO3.FDA

então MENORIDADE ← PESQUISA3.IDADE
NARQUIVO ← 3

fim se

fim ref.

Ref. Escreva no arquivo o registro selecionado e leia o próximo

se NARQUIVO = 1
então escreva SAÍDA.PESQUISA1
leia REGIÃO1.PESQUISA1

fim se

se NARQUIVO = 2
então escreva SAÍDA.PESQUISA2
leia REGIÃO2.PESQUISA2

fim se

se NARQUIVO = 3
então escreva SAÍDA.PESQUISA3
leia REGIÃO3.PESQUISA3

fim se

fim ref.

Ref. Feche arquivos

feche SAÍDA,REGIÃO1,REGIÃO2,REGIÃO3

fim ref.

Inserindo-se os refinamentos em seus respectivos lugares no algoritmo inicial, obtém-se:

Algoritmo {Intercalação de arquivos}

{Definição das estruturas de dados}

declare SAÍDA, (arquivo onde serão escritos os registros intercalados)
REGIÃO1,REGIÃO2,REGIÃO3 (arquivos de entrada por região)
arquivo seqüencial de PESQUISA1

declare PESQUISA1,PESQUISA2,PESQUISA3 {um registro por arquivo}
registro (IDADE,SALÁRIO,NDEPENDENTES,PATRIMÔNIO,
CALORIAS numérico,
SEXO,ESTADOCIVIL,INSTRUÇÃO literal)
declare NARQUIVO, {indica a qual arquivo pertence o registro com menor
idade}
MENORIDADE numérico

{Abertura dos arquivos}

abra SAÍDA escrita

abra REGIÃO1,REGIÃO2,REGIÃO3 leitura

{Leitura de um registro de cada arquivo}

leia REGIÃO1.PESQUISA1

leia REGIÃO2.PESQUISA2

leia REGIÃO3.PESQUISA3

{Geração do arquivo de saída}

repita

se REGIÃO1.FDA e REGIÃO2.FDA e REGIÃO3.FDA
então interrompa

fim se

{Iniciação das variáveis para seleção do registro com a menor idade}

MENORIDADE ← 9999

{Seleção do registro com a menor idade}

se PESQUISA1.IDADE < MENORIDADE e não REGIÃO1.FDA
então MENORIDADE ← PESQUISA1.IDADE
NARQUIVO ← 1

fim se

se PESQUISA2.IDADE < MENORIDADE e não REGIÃO2.FDA
então MENORIDADE ← PESQUISA2.IDADE
NARQUIVO ← 2

fim se

se PESQUISA3.IDADE < MENORIDADE e não REGIÃO3.FDA
então MENORIDADE ← PESQUISA3.IDADE
NARQUIVO ← 3

fim se

{Escrita no arquivo do registro selecionado e leitura do próximo}

se NARQUIVO = 1
então escreva SAÍDA.PESQUISA1
leia REGIÃO1.PESQUISA1

fim se

se NARQUIVO = 2
então escreva SAÍDA.PESQUISA2
leia REGIÃO2.PESQUISA2

fim se

se NARQUIVO = 3
então escreva SAÍDA.PESQUISA3
leia REGIÃO3.PESQUISA3

fim se

fim repita

{Fechamento dos arquivos}

feche SAÍDA,REGIÃO1,REGIÃO2,REGIÃO3

fim algoritmo.

2.4.6.3. EXERCÍCIOS DE FIXAÇÃO

● 2.4.6.3.1. Escrever um algoritmo para listar o arquivo AGENDA criado no exemplo 2.45.

○ 2.4.6.3.2. A seção de controle de produção de uma fábrica mantém um arquivo de registros de produção por máquinas. Cada registro contém o número da máquina e o número de peças produzidas em um

dia. Supondo que a fábrica possua três máquinas, escreva um algoritmo que separe o arquivo em três outros arquivos, um para cada máquina.

2.4.6.3.3. Reescrever o algoritmo do exemplo 2.47, substituindo os três registros declarados por uma variável composta heterogênea.

2.4.6.3.4. Uma agência bancária cria diariamente dois arquivos: o primeiro com as operações de crédito e o segundo com as operações de débito. Ambos os arquivos são ascendenteamente ordenados, pelo número da conta do cliente. Para atualizar o arquivo de contas correntes, é necessário intercalar o arquivo de débito e o de crédito, e gerar um único arquivo.

Supondo que o registro de cada um dos arquivos tenha as seguintes informações: número da conta, código da operação (débito ou crédito), valor da operação e data da operação, escrever um algoritmo que crie um arquivo intercalando débitos e créditos.

2.4.6.3.5. Supondo que o arquivo de contas correntes seja ascendenteamente ordenado, escrever um algoritmo que atualize este arquivo a partir do arquivo criado no exercício anterior.

2.4.7. Organização direta

A principal característica da organização direta é a facilidade de acesso a um registro desejado. Ao contrário da organização seqüencial, para se ter acesso a um registro de um arquivo direto, não é necessário pesquisar-se pelo registro, pois este pode ser obtido diretamente. Isto é possível porque a posição do registro no espaço físico do arquivo é univocamente determinada a partir de um dos campos do registro, escolhido no momento de criação do arquivo direto como "chave".

Exemplo 2.48

Suponha-se um arquivo de registros de alunos de um colégio. Cada registro possui um campo para o número de matrícula e uma sequência de outros campos com outras informações. A figura 2.4 representa a organização deste arquivo seqüencialmente. Um algoritmo para a obtenção de um registro neste arquivo, por exemplo procurando um registro com um determinado número de matrícula, deverá preocupar-se em pesquisar cada registro até encontrar, ou não, o registro desejado.

MATRÍCULA	OUTRAS	INFORMAÇÕES
820001	ZZZADDEF	00000
820007	KKKZDXZ	140300
820003	ACKYDYZ	111111
820005	ZKD AAAA	99990
830008	ZZZ DEEK	1111
830001	KYZ ZZZZ	0010

Fig. 2.4 Organização seqüencial do arquivo de alunos.

Um algoritmo para encontrar o registro com a matrícula 830008 poderia ser o seguinte:

Algoritmo {Pesquisa pelo registro cuja matrícula seja 830008, num arquivo seqüencial}
 {Definição do arquivo de alunos}
declare ALUNOS arquivo seqüencial de DADOS
declare DADOS registro (MATRÍCULA numérico,
OUTROS DADOS literal)
 {Definição do arquivo para impressão}
declare LISTAGEM arquivo seqüencial de LINHA

```

declare LINHA registro (COMPONENTE literal)
{Abertura dos arquivos}
abre ALUNOS leitura
abre LISTAGEM escrita
{Pesquisa do registro}

repita
  leia ALUNOS.DADOS
  se ALUNOS.FDA ou DADOS.MATRÍCULA = 830008
    então interrompa
  fim se
fim repita
se ALUNOS.FDA
então LINHA.COMPONENTE ← "REGISTRO NÃO ENCONTRADO"
  escreva LISTAGEM.LINHA
senão escreva LISTAGEM.DADOS
fim se
{Fechamento dos arquivos}
feche ALUNOS,LISTAGEM
fim algoritmo.

```

Se a organização for direta, a disposição dos registros no arquivo não será necessariamente apresentada pelo arquivo seqüencial. Através de funções internas ao computador, cada registro será locado em uma posição univocamente determinada pela chave escolhida. A figura 2.5 mostra como poderiam ser dispostos os registros num arquivo direto.

Para se ter acesso a um registro, basta efetuar-se a leitura ao arquivo usando a chave desejada, no caso, o número de matrícula desejado. Não há necessidade de o algoritmo prover nenhum tipo de pesquisa ao registro procurado. Caso a chave não exista, uma condição de CHAVE INVÁLIDA será possível de ser testada, semelhantemente ao mecanismo de FIM DE ARQUIVO.

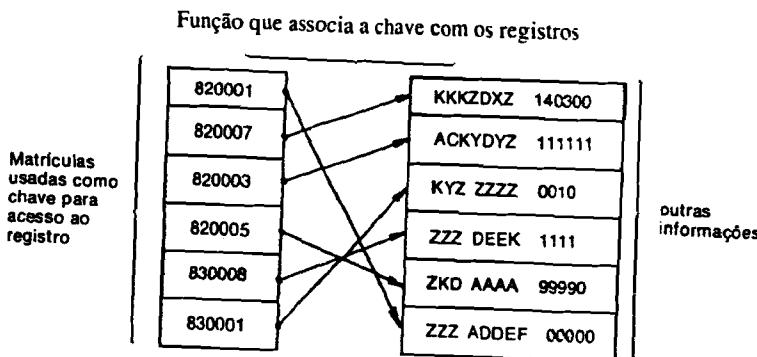


Fig. 2.5 Arquivo de acesso direto.

A escolha da chave é feita pelo usuário no momento da criação do arquivo. Em geral, escolhe-se como chave um dos campos do registro. Cada registro deverá ser gravado usando sua chave. Não pode haver registros usando a mesma chave, já que não haverá uma função capaz de distingui-los.

As operações de entrada e saída em um arquivo de organização direta são indicadas nos algoritmos pelos comandos dados a seguir.

2.4.7.1. COMANDO DE ENTRADA

Sua forma geral é dada por

leia item [chave] nome-do-arquivo . nome-do-registro

onde:

leia item chave	é uma palavra-chave; é uma palavra-chave; é o nome ou o valor do campo do registro escolhido como chave para endereçar o registro;
nome-do-arquivo nome-do-registro	é o nome do arquivo declarado no algoritmo; é o nome do registro declarado para o arquivo e que será usado para armazenar no ambiente do algoritmo o registro após a leitura ou antes da escrita.

2.4.7.2. COMANDO DE SAÍDA

Sua forma geral é dada por:

escreva item [chave] nome-do-arquivo nome-do-registro

onde:

escreva item chave	é uma palavra-chave; é uma palavra-chave; é o nome ou o valor do campo do registro escolhido como chave para endereçar o registro;
nome-do-arquivo nome-do-registro	é o nome do arquivo declarado no algoritmo; é o nome do registro declarado no algoritmo e que será usado para armazenar no ambiente do algoritmo o registro após a leitura ou antes da escrita.

O algoritmo desenvolvido anteriormente para encontrar o registro com a matrícula 830008 no arquivo seqüencial poderá ser reescrito com a seguinte forma para um arquivo de acesso direto.

Exemplo 2.49

Algoritmo {Pesquisa pelo registro cuja matrícula seja 830008, num arquivo direto}

{Definição do arquivo de alunos}

declare ALUNOS arquivo direto de DADOS
declare DADOS registro (MATRÍCULA numérico,
RESTO literal)

{Definição do arquivo para impressão}

declare SAÍDA arquivo seqüencial de LINHA
declare LINHA registro (COMPONENTE literal)
(Abertura dos arquivos)

abra ALUNOS leitura

abra SAÍDA escrita

{Leitura do registro, usando a matrícula 830008 como chave}

leia item [830008] ALUNOS.DADOS

{Se esta chave não for encontrada no arquivo, então existe erro}

se ALUNOS.INV

então LINHA ← "CHAVE NÃO EXISTE NO ARQUIVO"

escreva SAÍDA.LINHA

senão LINHA ← DADOS.RESTO

escreva SAÍDA.LINHA

fiim se

{Fechamento dos arquivos}

feche ALUNOS,SAÍDA
fiim algoritmo.

Exemplo 2.50

Criar, a partir de um arquivo seqüencial, um arquivo de acesso direto, com registros contendo a placa, tipo e nome do proprietário de um veículo. Usar como chave a placa do veículo.

Algoritmo {Criação de arquivo de acesso direto a partir de arquivo seqüencial}

{Definição das estruturas de dados}

declare FONTE {arquivo seqüencial contendo os registros dos veículos}

arquivo seqüencial de REG

declare VEÍCULOS {arquivo de acesso direto com os registros dos veículos}

arquivo direto de REG

declare REG registro (PLACA,TIPO,PROPRIETÁRIO literal)

{Abertura dos arquivos}

abra FONTE leitura

abra VEÍCULOS escrita

{Criação do arquivo direto}

repita

leia FONTE.REG

se FONTE.FDA

então interrompa

fiim se

escreva item [PLACA] VEÍCULOS.REG

fiim repita

{Fechamento de arquivos}

feche FONTE

feche VEÍCULOS

fiim algoritmo.

Exemplo 2.51

Um terminal, composto de uma tela e um teclado, pode ser tratado em um algoritmo, e também em um programa, como um arquivo. Pela própria natureza do equipamento, a tela deverá ser tratada como um arquivo de saída, e o teclado, como um arquivo de entrada. Ambos são arquivos seqüenciais.

Agora, supondo-se que em um sistema computacional bancário exista um arquivo de acesso direto, e que cada registro gravado usa como chave o número da conta do cliente, escrever um algoritmo para um programa que receba através do teclado o número da conta do cliente e devolva, na tela, o nome do cliente e o saldo da conta. Supor que o registro do arquivo contenha somente estas informações.

Algoritmo {Exibição de saldo em conta corrente}

{Definição da estrutura de dados}

declare CONTAS CORRENTES {arquivo contendo registro das contas correntes}

arquivo direto de CLIENTE

declare TECLADO {arquivo para leitura do número da conta}

arquivo seqüencial de CONTA

declare TELA {arquivo para exibição dos dados do cliente}

arquivo seqüencial de LINHA

declare CLIENTE registro (NOME DO CLIENTE literal,

SALDO numérico)

declare TEXTO registro (COMPONENTE literal)

declare CONTA registro (CAMPO numérico)

declare LINHA registro (NOME DO CLIENTE literal).

NÚMERODACONTA,SALDO numérico)

{Abertura dos arquivos}

abre TECLADO leitura

abre TELA escrita

abre CONTASCORRENTES leitura

(Leitura dos arquivos)

repita

leia TECLADO.CONTA

se TECLADO.FDA

então interrompa

final se

leia item [CONTA] CONTASCORRENTES.CLIENTE

se CONTASCORRENTES.INV

então TEXTO ← "NÚMERO DA CONTA NÃO EXISTE"

escreva TELA.TEXTO

senão LINHA.NOMEDOCliente ← CLIENTE.NOMEDOCiente

LINHA.NÚMERODACONTA ← CONTA.CAMPO

LINHA.SALDO ← CLIENTE.SALDO

escreva TELA.LINHA

final se

final repita

{Fechamento dos arquivos}

feche TECLADO,TELA,CONTASCORRENTES

final algoritmo.

2.5. EXERCÍCIOS PROPOSTOS

2.5.1. Variáveis compostas unidimensionais

PROBLEMAS GERAIS

△ 2.5.1.1. Em uma cidade do interior, sabe-se que, de janeiro a abril de 1976 (121 dias), não ocorreu temperatura inferior a 15°C nem superior a 40°C. As temperaturas verificadas em cada dia estão disponíveis em uma unidade de entrada de dados.

Fazer um algoritmo que calcule e imprima:

- a menor temperatura ocorrida;
- a maior temperatura ocorrida;
- a temperatura média;
- o número de dias nos quais a temperatura foi inferior à temperatura média.

△ 2.5.1.2. Fazer um algoritmo que:

- leia uma frase de 80 caracteres, incluindo brancos;
- conte quantos brancos existem na frase;
- conte quantas vezes a letra A aparece;
- conte quantas vezes ocorre um mesmo par de letras na frase e quais são elas;
- imprima o que foi calculado nos itens b, c e d.

△ 2.5.1.3. Fazer um algoritmo que:

- leia o valor de n ($n \leq 1000$) e os n valores de uma variável composta A de valores numéricos, ordenados de forma crescente:

b) determine e imprima, para cada número que se repete no conjunto, a quantidade de vezes em que ele aparece repetido;

c) elimine os elementos repetidos, formando um novo conjunto;

d) imprima o conjunto obtido no item c.

△ 2.5.1.4. Dado um conjunto de 100 valores numéricos disponíveis num meio de entrada qualquer, fazer um algoritmo para armazená-los numa variável composta B, e calcular e imprimir o valor do somatório dado a seguir:

$$S = (b_1 - b_{100})^3 + (b_2 - b_{99})^3 + (b_3 - b_{98})^3 + \dots + (b_{50} - b_{51})^3$$

Exemplo:

B

210	160	...	33	97
1	2		99	100

$$S = (210 - 97)^3 + (160 - 33)^3 + \dots$$

▲ 2.5.1.5. Fazer um algoritmo que:

- leia um conjunto de valores inteiros correspondentes a 80 notas dos alunos de uma turma, notas estas que variam de 0 a 10;
- calcule a freqüência absoluta e a freqüência relativa de cada nota;
- imprima uma tabela contendo os valores das notas (de 0 a 10) e suas respectivas freqüências absoluta e relativa.

Observações:

1. Freqüência absoluta de uma nota é o número de vezes em que ela aparece no conjunto de dados.
2. Freqüência relativa é a freqüência absoluta dividida pelo número total de dados.
3. Utilizar como variável composta somente aquelas que forem necessárias.

▲ 2.5.1.6. Fazer um algoritmo que leia diversos pares de conjuntos numéricos e que imprima a identificação dos pares de conjuntos disjuntos (aqueles que não possuem elementos comuns a ambos). Os elementos de cada par de conjuntos são precedidos pelo nome que identifica o par e pelo número de elementos de cada conjunto. Após o último par de conjuntos vem como identificação do par o literal VAZIO. O número máximo de elementos de cada conjunto é 250.

△ 2.5.1.7. Um armazém trabalha com 100 mercadorias diferentes identificadas pelos números inteiros de 1 a 100. O dono do armazém anota a quantidade de cada mercadoria vendida durante o mês. Ele tem uma tabela que indica, para cada mercadoria, o preço de venda. Escreva um algoritmo para calcular o faturamento mensal do armazém. A tabela de preços é fornecida seguida pelos números das mercadorias e as quantidades vendidas. Quando uma mercadoria não tiver nenhuma venda, é informado o valor zero no lugar da quantidade.

▲ 2.5.1.8. Uma grande firma deseja saber quais os três empregados mais recentes. Fazer um algoritmo para ler um número indeterminado de informações (máximo de 300) contendo o número do empregado e o número de meses de trabalho deste empregado e imprimir os três mais recentes.

Observações: A última informação contém os dois números iguais a zero. Não existem dois empregados admitidos no mesmo mês.

Exemplo:

EMPREGADOS

224	1731	2210	4631	...	526
1	2	3	4		300

MESES

17	3	9	2		10
----	---	---	---	--	----

Empregado mais recente: 4631.

△ 2.5.1.9. Fazer um algoritmo que:

- leia uma variável composta A com 30 valores numéricos;
- leia uma outra variável composta B com 30 valores numéricos;
- leia o valor de uma variável X;
- verifique qual o elemento de A que é igual a X;
- imprima o elemento de B de posição correspondente à do elemento de A igual a X.

- ▲ 2.5.1.10. Intercalação é o processo utilizado para construir uma tabela ordenada, de tamanho $n + m$, a partir de duas tabelas já ordenadas de tamanhos n e m . Por exemplo, a partir das tabelas:

1	3	6	7
---	---	---	---

2	4	5
---	---	---

construímos a tabela

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Fazer um algoritmo que:

- leia NA, número de elementos do conjunto A ($NA \leq 100$);
- leia, em seguida, os elementos do conjunto A;
- leia, logo após o valor de NB, número de elementos do conjunto B ($NB \leq 100$);
- leia, finalmente, os elementos do conjunto B;
- crie e imprima um conjunto C, ordenado, de tamanho $NA + NB$, a partir dos conjuntos originais A e B.

Observações:

- Considerar os elementos de A e B como inteiros.
- Os elementos de A e B já são lidos ordenados.

PROBLEMAS DE APLICAÇÃO EM CIÊNCIAS EXATAS

▲ 2.5.1.11. Fazer um algoritmo que:

- leia o valor de M ($M \leq 30$) e os M valores de uma variável composta A;
- leia o valor de N ($N \leq 20$) e os N valores de uma variável composta B;
- determine o conjunto C = A ∪ B (união de A com B), onde C não deverá conter elementos repetidos (A e B não contêm elementos repetidos);
- imprima os elementos contidos em A, B e C.

▲ 2.5.1.12. Seja

$$P = a_n X^n + a_{n-1} X^{n-1} + a_{n-2} X^{n-2} + \dots + a_1 X + a_0$$

Escrever um algoritmo que:

- leia o valor de n, sendo $n \leq 20$;
- leia os coeficientes a_i , $i = 0, 1, 2, \dots, n$;
- calcule o valor de P para 10 valores lidos para x;
- imprima o valor de x e o valor de P correspondente.

2.5.2. Variáveis compostas multidimensionais

PROBLEMAS GERAIS

△ 2.5.2.1. Seja a seguinte variável composta bidimensional A:

A	1	2	3	4	5	6
1	175	225	10	9000	3,7	4,75
2	9,8	100	363	432	156	18
3	40	301	30,2	6381	1	0
4	402	4211	7213	992	442	7321
5	21	3	2	1	9000	2000

a) Quais elementos fazem parte do conjunto.

- Qual o conteúdo do elemento identificado por A[4,5]?
- Qual o conteúdo de X após a execução do comando $X \leftarrow A[3,2] + A[5,1]$?
- O que aconteceria caso fosse referenciado o elemento A[6,2] no algoritmo?

e) Somar os elementos da quarta coluna
(A[1,4] + A[2,4] + A[3,4] + A[4,4] + A[5,4]).

f) Somar os elementos da terceira linha:
(A[3,1] + A[3,2] + A[3,3] + A[3,4]).

△ 2.5.2.2. Dada a variável bidimensional B, de 100 linhas por 200 colunas, escrever o trecho de algoritmo que calcula o somatório dos elementos da quadragésima coluna.

△ 2.5.2.3. Com a mesma variável composta do exercício anterior, escrever o trecho de algoritmo que calcula o somatório dos elementos da trigésima linha.

△ 2.5.2.4. Dadas as variáveis compostas A e B abaixo:

A	7	8	4	9
	2	1	7	3

B	6	9	11	15
	32	19	3	4

Calcular o conjunto $C = A + B$.

▲ 2.5.2.5. Fazer um algoritmo que:

- leia duas variáveis compostas bidimensionais de dimensão $m \times n$ ($m \leq 20, n \leq 30$). Os valores de m e n são fornecidos inicialmente. A seguir, são informadas cada uma das linhas de cada uma das variáveis;
- calcule e imprima a soma dessas variáveis.

△ 2.5.2.6. Fazer um algoritmo que:

- leia uma matriz A, de dimensão $M \times N$ ($M \leq 20; N \leq 50$). Os valores de M e N são dados e, a seguir, são fornecidas as linhas da matriz;
- determine a matriz transposta de A;
- imprima a matriz A e a sua transposta.

Exemplo:

A	9	16	34
	32	11	17

TRANPOSTA DE A	9	32
	16	11
	34	17

▲ 2.5.2.7. Fazer um algoritmo que:

- leia uma matriz inteira A de $M \times N$, onde os elementos de cada linha e os valores de M e N são fornecidos ($M \leq 20, N \leq 10$);
- imprima a matriz lida;
- calcule e imprima uma matriz modificada B ($M \times N + 1$), sendo que os elementos da $(N + 1)$ -ésima coluna são formados com o produto dos elementos da mesma linha.

Exemplo:

A	2	3
	4	5

B	2	3	6
	4	5	20

△ 2.5.2.8. Uma biblioteca possui oito departamentos. Cada departamento contém 40 estantes capazes de conter, cada uma, 150 livros. Supondo que o livro-padrão tenha 200 páginas de 35 linhas por 60 colunas de caracteres, declarar uma variável composta capaz de conter todos os caracteres presentes nos livros da biblioteca.

△ 2.5.2.9. Um grupo de pessoas respondeu a um questionário composto de 10 perguntas. Cada pergunta contém cinco opções ou respostas possíveis, codificadas de 1 a 5. Cada pergunta é respondida com a escolha de apenas uma opção dentre as cinco opções possíveis.

São fornecidos os nomes das pessoas e suas respectivas respostas. A última informação, utilizada como flag, contém o nome da pessoa igual a "VAZIO".

Fazer um algoritmo para ler e imprimir os dados lidos e calcular e imprimir o número de pessoas que responderam a cada uma das cinco opções de cada pergunta.

△ 2.5.2.10. Fazer um algoritmo para controlar as reservas de passagem dos vôos de uma companhia aérea e verificar os lucros e prejuízos da mesma.

O algoritmo deverá:

1. Ler os dados de 10 vôos. Os dados de cada voo são formados pelo:
 - número de voo;
 - tipo de avião utilizado (707, 727, 737);
 - preço da passagem.

2. Ler um número indeterminado de pedidos de reservas, contendo cada um:
 - número da identidade do passageiro; e
 - número do voo desejado;

(flag: número da identidade = 0).

3. Verificar, para cada passageiro, se há disponibilidade no voo. Em caso afirmativo, atualizar o número de lugares disponíveis e imprimir:
 - número da identidade do passageiro;
 - número do voo desejado;
 - preço da passagem;
 - a mensagem "RESERVA CONFIRMADA".

Em caso negativo, imprimir os dois primeiros itens e a mensagem "VÔO LOTADO".

4. Ao final, imprimir uma estatística de lucros e prejuízos por voo e no total da companhia aérea. Considerar que a lotação de 60% de capacidade de cada avião não produz lucros nem prejuízos. Sendo assim, acima deste valor é lucro e abaixo é prejuízo.

Observação: Capacidade de cada avião: 707 — 200 lugares; 727 — 170 lugares; 737 — 120 lugares.

- ▲ 2.5.2.11. Fazer um algoritmo que leia e imprima uma variável composta bidimensional cujo conteúdo é a população dos 10 municípios mais populosos de cada um dos 23 estados brasileiros.

1	2	...	10

POPULAÇÃO [i,j]

população do j-ésimo município
do i-ésimo estado.

Determinar e imprimir o número do município mais populoso e o número do estado a que pertence. Considerando que a primeira coluna contém sempre a população da capital do estado, calcular a média da população das capitais dos 23 estados.

▲ 2.5.2.12. A composição dos custos das diversas atividades de construção de um prédio é feita a partir da elaboração de um quadro de quantitativos dos diversos recursos envolvidos em cada atividade. Estes recursos são de vários tipos e envolvem principalmente os custos mais diretos, como, por exemplo, matérias-primas, mão-de-obra, hora de equipamento etc.

Sendo conhecidos os custos unitários para cada recurso envolvido, chega-se facilmente ao custo final unitário de cada atividade. A este custo são acrescidos os percentuais de custos indiretos (adminis-

tivos), impostos, depreciação de equipamentos, leis sociais etc., totalizando o preço final para a execução de cada fase.

Este procedimento básico é adotado em várias empreiteiras de obras e o objetivo deste trabalho é fazer um algoritmo que execute estes cálculos para auxiliar o analista de custos de uma empreiteira.

Supondo-se que na execução do prédio são realizados quatro tipos de atividades e que cada uma consome os recursos especificados na tabela dada a seguir

Atividade \ Recurso	Cimento (kg)	Areia m ³	Brita m ³	Hora de pedreiro (h)	Hora de servente (h)	Tijolo (u)	Betoneira (h)
1 — Fundação m ³	50	0,4	0,6	5	3	0	3
2 — Alvenaria m ²	20	0,3	0	2	1	100	1
3 — Estrutura m ³	70	0,3	0,7	6	3	0	35
4 — Acabamento m ²	40	0,2	0	9	5	0	1

e que as despesas indiretas (administração) são dados levantados a cada mês, fazer um algoritmo que:

- a) leia o percentual de administração do mês;
- b) leia os custos unitários dos sete recursos envolvidos;
- c) leia um conjunto indeterminado de dados (máximo de 15 atividades) contendo os quantitativos de recursos envolvidos em cada atividade;
- d) calcule e imprima:
 - d.1) o preço unitário de custo (direto + administração) de cada atividade;
 - d.2) o preço unitário que a empreiteira deve cobrar em cada atividade, para que tenha 36% de lucro;
 - d.3) considerando o percentual de 16% para as leis sociais, incidentes sobre a mão-de-obra, quanto deve ser recolhido para cada unidade de atividade;
 - d.4) considerando o percentual de administração fornecido + 36% de lucro + 16% de leis sociais, qual será o preço a ser cobrado pela empreiteira para a construção de uma obra que envolva as seguintes atividades:
 50 m³ de fundação,
 132 m² de alvenaria,
 200 m³ de estrutura,
 339 m² de acabamento;
 - d.5) para a mesma obra acima, qual será a quantidade total de cada recurso envolvido?

△ 2.5.2.13. Desenvolver um algoritmo para imprimir uma tabela com o índice de afinidade existente entre cada moça e cada rapaz de um grupo de M moças e um grupo de R rapazes ($R \leq 50$ e $M \leq 60$).

Foi distribuído entre eles um questionário de 100 perguntas, tais como:

1. Você se incomoda que seu parceiro fume?
2. Você é vibrado em música sertaneja?
3. Você gosta de cebola?

...

100. Você gosta do AMÉRICA FUTEBOL CLUBE?

Cada resposta tem as seguintes opções:

SIM

INDIFERENTE

NÃO

O índice de afinidade de um rapaz com uma moça é dado pelo número de perguntas em que ambos deram a mesma resposta ou em que um deles deu a resposta indiferente.

O algoritmo poderá ler:

- os valores de R e M;
- as 100 respostas de cada rapaz;
- as 100 respostas de cada moça.

A tabela que será impressa deverá ter o aspecto:

	1	2	3	...	M
1	60	70	20	...	
2	10	30	82	...	
3	41	73	91	...	
.	
.	
R					

onde se pode observar, por exemplo, que o índice de afinidade do segundo rapaz com a terceira moça é 82.

PROBLEMAS DE APLICAÇÃO EM CIÊNCIAS EXATAS

▲ 2.5.1.4. Fazer um algoritmo que:
a) leia uma matriz quadrada real A, de dimensão $M \times M$ ($M \leq 20$). O valor de M é fornecido inicialmente;

b) verifique se a matriz é simétrica, ou seja, se $A[i,j] = A[j,i]$ para $\forall i, j \leq M$;
c) imprima a palavra simétrica, se a matriz A for simétrica, e não-simétrica, em caso contrário.

▲ 2.5.2.15. Escrever um algoritmo que calcule e imprima as "n" raízes do seguinte sistema particular de "n" equações com "n" incógnitas.

$$\begin{array}{lcl} a_{11}X_1 & = & b_1 \\ a_{21}X_1 + a_{22}X_2 & = & b_2 \\ a_{31}X_1 + a_{32}X_2 + a_{33}X_3 & = & b_3 \\ \dots & & \dots \\ a_{n1}X_1 + a_{n2}X_2 + \dots + a_{nn}X_n & = & b_n \end{array}$$

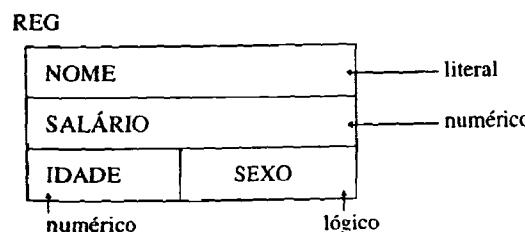
Para isto, ler:

- número de equações, $N \leq 20$;
- a matriz triangular A dos coeficientes;
- o vetor B dos termos independentes.

2.5.3. Registros

PROBLEMAS GERAIS

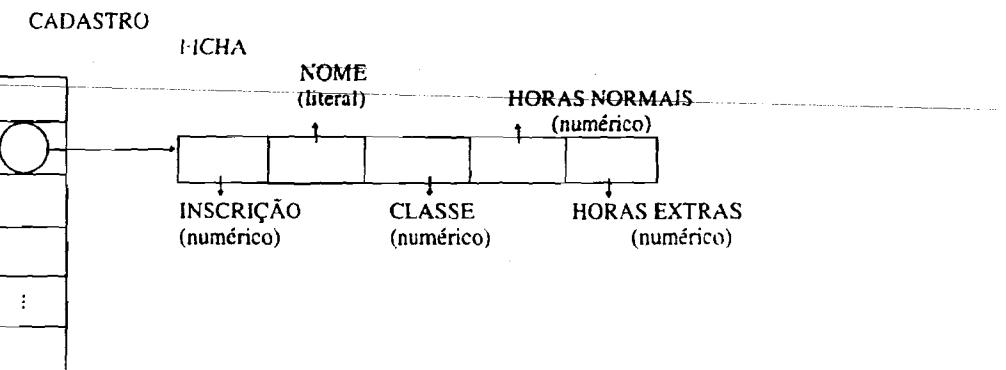
△ 2.5.3.1. Declarar o registro cuja representação gráfica é dada a seguir.



△ 2.5.3.2. Escrever o comando que atribui 7840212,00 ao campo de nome SALÁRIO do registro REG.

△ 2.5.3.3. Uma indústria faz a folha mensal de pagamentos de seus empregados baseada no seguinte:

Existe uma tabela com os dados do funcionário



Fazer um algoritmo que processe a tabela e emita, para cada funcionário, seu contracheque, cujo formato é dado a seguir:

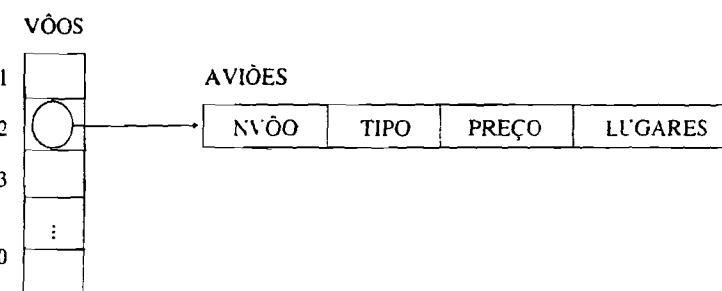
NÚMERO DE INSCRIÇÃO:	NOME:
SALÁRIO HORAS NORMAIS:	
SALÁRIO HORAS EXTRAS:	
DEDUÇÃO INSS:	
SALÁRIO LÍQUIDO:	

O salário de referência deverá ser lido previamente.

O salário referente às horas extras é calculado acrescentando 30% ao salário-hora normal. O desconto do INSS é de 8% do salário bruto (salário correspondente às horas normais trabalhadas - salário correspondente às horas extras).

Para o cálculo do salário, considerar que existem duas classes de funcionários, a classe 1, cujo salário é 1,3 vez o salário de referência, e a classe 2, cujo salário é 1,9 vez o salário de referência.

△ 2.5.3.4. Resolver novamente o exercício 2.5.2.9., considerando as seguintes estruturas de dados:



▲ 2.5.3.5. Uma empresa de transporte interestadual deseja calcular a distância percorrida pelos ônibus. Para isto, é fornecido o percurso de cada ônibus com os seguintes dados:

- número do ônibus;
- número de cidades percorridas;
- códigos de todas as cidades percorridas.

Assim:

103, 08, 01, 05, 07, 03, 09, 03, 08, 05

indica que o ônibus n.º 103 percorreu oito cidades, na seguinte ordem:

- da cidade 01 para a cidade 05;
- da cidade 05 para a cidade 07;
- da cidade 07 para a cidade 03.

etc.

Cada ônibus percorre um máximo de 24 cidades.

Para calcular a distância entre cidades, a empresa possui uma tabela de distâncias (30×30):

	01	02	03	04	30	
01	0	15	10	18	...	90
02	15	0	25	42	...	
03	10	25	0	12	...	
04	18	42	12	0	...	
...
...
...

A 2.5.3.6. Para evitar erros de digitação de seqüências de números de importância fundamental, como a matrícula de um aluno, o CPF do Imposto de Renda, o número de conta bancária, geralmente se adiciona ao número um dígito verificador. Por exemplo, o número de matrícula 811057 é usado como 8110573, onde 3 é o dígito verificador, calculado da seguinte maneira:

- a) cada algarismo do número é multiplicado por um peso começando de 2 e crescendo de 1, da direita para a esquerda:

$$8 \times 7, 1 \times 6, 1 \times 5, 0 \times 4, 5 \times 3, 7 \times 2;$$

- b) somam-se as parcelas obtidas:

$$56 + 6 + 5 + 0 + 15 + 14 = 96;$$

- c) calcula-se o resto da divisão desta soma por 11:

$$96 \text{ dividido por } 11 \text{ dá resto } 8 (96 = 8 \times 11 + 8);$$

- d) subtrai-se de 11 o resto obtido:

$$11 - 8 = 3;$$

- e) se o valor encontrado for 10 ou 11, o dígito verificador será 0; nos outros casos, o dígito verificador é o próprio valor encontrado.

Escrever um algoritmo capaz de:

1. Ler um conjunto de registros contendo, cada um, o número de uma conta bancária, o dígito verificador deste número, o saldo da conta e o nome do cliente. O último registro, que não deve ser considerado, contém o número de conta igual a zero.
2. Utilizando o esquema de verificação acima, imprimir duas listas de clientes distintas no seguinte formato de saída:

CONTAS DE NÚMERO CORRETO

413599-7 987,30 Débora Neuenschander
111118-06 121,99 Juliana Berg
...

CONTAS DE NÚMERO ERRADO

765432-1 335,66 Júnia Faria
...

Δ 2.5.3.7. Seja um conjunto de dados, cada um com valores

N.º do pedido	N.º do produto	Quantidade
---------------	----------------	------------

classificados em ordem crescente pelo número do pedido e contendo uma linha para cada produto pedido.

Exemplo:

	0002	435	010
0001	103	200	
0001	697	150	
0001	435	400	

Existe uma tabela de preços para cada produto e esta tabela está em linhas com o formato:

N.º do produto	Preço unitário
----------------	----------------

Para se descobrir o preço de cada produto, basta pesquisar esta tabela. (Não existe nenhum produto que não esteja na tabela, nem existe produto na tabela que não tenha preço.)

Escrever um algoritmo para emitir o relatório abaixo:

N.º do pedido	N.º do produto	Quantidade	Preço unitário	Valor

△ 2.5.3.8. Supondo-se que o cadastro dos funcionários de uma empresa seja do tipo do exemplo 2.29, fazer um algoritmo que leia o cadastro, disponível numa unidade de entrada, e liste todos os funcionários cujo cargo seja "INSTRUTOR", situação "AFASTADO" e salário superior a R\$ 4.000,00.

2.5.4. Arquivos

PROBLEMAS GERAIS

△ 2.5.4.1. Copiar o arquivo seqüencial denominado FONTE para um arquivo seqüencial chamado FON-ITENOVA. Em ambos os arquivos os registros têm os seguintes campos:

CHAVE — numérico
INFORMAÇÃO — literal
DATA — numérico

△ 2.5.4.2. Listar um arquivo seqüencial denominado DADOS que possui registros com campos de NOME, ENDEREÇO, CEP e TELEFONE.

△ 2.5.4.3. Dado o arquivo CADASTRO com registros com os campos NOME, SEXO, COR-DE-OLHOS, ALTURA, PESO E DATA-DE-NASCIMENTO, separá-los em dois arquivos: um chamado HOMENS, com registros cujo campo SEXO apresente o valor 1 (sexo masculino), e outro chamado MULHERES, com registros cujo campo SEXO seja igual a 2. Os registros dos novos arquivos deverão possuir os seguintes campos: NOME, COR-DE-OLHOS, PESO e DATA-DE-NASCIMENTO.

△ 2.5.4.4. O arquivo BOLETA contém registros das operações de clientes na Bolsa de Valores. Cada operação de compra ou venda que um cliente realiza na bolsa gera um registro com o NÚMERO do cliente, o CÓDIGO da operação (V para venda e C para compra), a DESCRIÇÃO do título comercializado, a QUANTIDADE de títulos comercializados e o VALOR unitário de cada título. Estes registros estão seqüencialmente organizados no arquivo BOLETA, de tal modo que todos os registros de um mesmo cliente estão juntos. Escrever um algoritmo para gerar o arquivo seqüencial RESULTADO, onde, para cada cliente, apareça um registro da forma:

NÚMERO do cliente, SALDO apurado, TIPO de saldo

O TIPO de saldo será igual a C (de CREDOR), se o valor comprado for maior ou igual que o vendido. Caso contrário, o saldo será D (de DEVEDOR).

△ 2.5.4.5. O arquivo seqüencial ENTRADA contém registros RS com os campos CÓDIGO, NOME e DISPONÍVEL e registros RM com os campos CÓDIGO, NOME, DEPÓSITOS ou RETIRADAS. O registro RS fornece a situação, ou o saldo, do cliente de um banco e RM fornece o movimento do dia. Supondo que os registros estejam ordenados por nome, e que o último registro referente a um mesmo cliente seja um registro RS, escreva um algoritmo que crie um arquivo SAIDA somente de registros RS, sendo cada registro atualizado pelas operações indicadas nos registros RM correspondentes ao cliente. O campo CÓDIGO identifica cada registro da seguinte forma:

CÓDIGO = 1 — registro RS;
CÓDIGO = 2 — registro RM operação depósito;
CÓDIGO = 3 — registro RM operação retirada.

△ 2.5.4.6. No exercício de fixação 2.4.6.3.4. foi criado um arquivo com o movimento de débitos e créditos, ordenados por número de contas bancárias. Escrever agora um algoritmo que leia os registros daquele arquivo, calcule o resultado do movimento por número de conta e atualize o saldo em um arquivo de acesso direto em que a chave é o número da conta. Escolher um formato adequado para o registro.

Capítulo 3

Modularização

3.1. INTRODUÇÃO

No fim da década de 60, um conjunto de problemas no desenvolvimento de sistemas de programação levou os países desenvolvidos à chamada "crise de software".

Os custos das atividades de programação mostravam a cada ano uma clara tendência a se elevar em muito em relação aos custos dos equipamentos. Essa tendência era devida, em grande parte, ao rápido avanço tecnológico na fabricação dos equipamentos de computação, em contrapartida com a lenta evolução das técnicas aplicadas ao desenvolvimento de software.

A ausência de uma metodologia para a construção de programas conduz a um software geralmente cheio de erros e com alto custo de desenvolvimento que, consequentemente, exige um custo elevado para sua correção e manutenção futuras.

A **programação estruturada** é hoje o resultado de uma série de estudos e propostas de disciplinas e metodologias para o desenvolvimento de software. Conceitos associados como **técnica de refinamentos sucessivos e modularização de programas** integram o ferramental para a elaboração de programas visando, principalmente, os aspectos de confiabilidade, legibilidade, manutenibilidade e flexibilidade.

Pode-se reunir as idéias da programação estruturada em três grupos:

- desenvolvimento de algoritmos por fases ou refinamentos sucessivos;
- uso de um número muito limitado de estruturas de controle;
- transformação de certos refinamentos sucessivos em módulos.

O desenvolvimento de algoritmo por refinamentos sucessivos e o uso de número limitado de estruturas de controle são técnicas largamente usadas nos capítulos anteriores do presente texto.

Este capítulo aborda os aspectos de decomposição de algoritmos em módulos, tendo em vista os refinamentos já desenvolvidos, para dominar a complexidade e organizar o processo de programação.

Quando se desenvolve um algoritmo através de refinamentos sucessivos, faz-se uma opção pela divisão do algoritmo; este procedimento conduz à modularização da solução do problema.

Um **módulo** é, então, um grupo de comandos, constituindo um trecho de algoritmo, com uma função bem definida e o mais independente possível em relação ao resto do algoritmo.

Assim sendo, ao se elaborar um algoritmo para calcular o salário líquido de um empregado, tem-se as seguintes etapas:

Algoritmo
Leia os dados do funcionário
Determine o salário
Escreva o salário
Final algoritmo

O comando "determine o salário" pode ser refinado como a seguir:

```
Ref. Determine o salário
Calcule as vantagens
Calcule as deduções
SALARIOLIQUIDO ← VANTAGENS — DEDUÇOES
fim ref.
```

Na elaboração do refinamento acima, não houve preocupação de como o processo de cálculo das vantagens e das deduções seria efetuado. Essas ações constituem funções bem definidas no algoritmo e que serão executadas por módulos específicos. Nesta fase do projeto do algoritmo, pode-se, então, optar pela elaboração de módulos para o cálculo das vantagens e cálculo das deduções. O mesmo refinamento ficaria, então:

```
Ref. Determine o salário
Ative o módulo "Cálculo das vantagens"
Ative o módulo "Cálculo das deduções"
SALARIOLIQUIDO ← VANTAGENS — DEDUÇOES
fim ref.
```

Os módulos seriam:

```
módulo {Cálculo das vantagens}
Calcule as vantagens
fim módulo
```

```
módulo {Cálculo das deduções}
Calcule as deduções
fim módulo
```

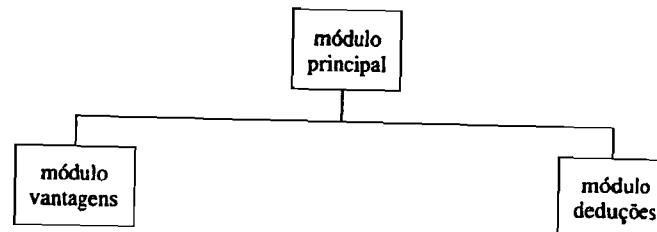
A versão final modularizada do algoritmo ficaria então constituída do algoritmo seguinte, mais os módulos refinados:

```
Algoritmo
Leia os dados do funcionário
Ative o módulo "Cálculo das vantagens"
Ative o módulo "Cálculo das deduções"
SALARIOLIQUIDO ← VANTAGENS — DEDUÇOES
escreva SALARIOLIQUIDO
fim algoritmo.
```

```
módulo {Cálculo das vantagens}
SALARIOBRUTO ← NHORAS × SALARIOHORA
SALARIOFAMILIA ← NFILHOS × VALORPORFILHO
VANTAGENS ← SALARIOBRUTO + SALARIOFAMILIA
fim módulo
```

```
módulo {Cálculo das deduções}
INSS ← SALARIOBRUTO × 0,08
IRPF ← SALARIOBRUTO × TAXA
DEDUÇOES ← INSS + IRPF
fim módulo
```

A descrição estrutural da modularização pode ser feita através do diagrama hierárquico, como a seguir:



A maneira mais intuitiva de proceder à modularização de problemas é feita definindo-se um módulo principal de controle e módulos específicos para as funções do algoritmo. No diagrama anterior, o módulo principal tem a função de receber os dados, escrever os resultados e exercer o controle na execução das funções do algoritmo. A determinação das vantagens e deduções é delegada a módulos específicos.

As linguagens de programação hoje existentes dispõem de recursos que facilitam a construção e manipulação de módulos. Estes recursos permitem não só a modularização dos comandos do programa, como também a modularização dos dados utilizados.

A experiência recomenda que os módulos de um programa devem ter um tamanho limitado. Módulos muito grandes são difíceis de ser compreendidos e, em geral, são multifuncionais.

Um outro aspecto importante é a possibilidade de cada módulo poder definir as próprias estruturas de dados, suficientes e necessárias apenas para atingir o objetivo final do módulo.

Todo módulo é constituído por uma sequência de comandos que operam sobre um conjunto de objetos, que podem ser globais ou locais.

Objetos globais são entidades que podem ser usadas em módulos internos a outro módulo do algoritmo onde foram declaradas.

Objetos locais são entidades que só podem ser usadas no módulo do algoritmo onde foram declaradas. Estes objetos não possuem qualquer significado fora deste módulo. São exemplos de objetos globais ou locais: variáveis, arquivos, outros módulos etc.

Um módulo pode usar objetos globais ou locais em relação a ele. Porém, não pode usar objetos declarados em módulos que não o abrangem. Isto significa que objetos globais, declarados em módulos mais externos ou mesmo a nível do módulo principal, podem ser também utilizados em módulos mais internos.

A comunicação entre módulos deverá ser feita através de vínculos, utilizando-se objetos globais ou transferência de parâmetros (vide 3.2.1).

A decisão pela divisão do algoritmo em módulos traz benefícios tais como:

- a independência do módulo permite uma manutenção mais simples e evita efeitos colaterais em outros pontos do algoritmo;
- b) a elaboração do módulo pode ser feita independentemente e em época diferente do restante do algoritmo;
- c) testes e correções dos módulos podem ser feitos em separado;
- d) um módulo independente pode ser utilizado em outros algoritmos que requeiram o mesmo processamento por ele executado.

3.2. FERRAMENTAS PARA MODULARIZAÇÃO

Dentre as ferramentas de modularização pode-se destacar:

sub-rotinas
funções

As sub-rotinas e as funções são módulos de programação que servem basicamente a três objetivos:

- evitar que uma certa sequência de comandos necessária em vários locais de um algoritmo tenha que ser escrita repetidamente nestes locais;
- dividir e estruturar um algoritmo em partes fechadas e logicamente coerentes;
- aumentar a legibilidade de um algoritmo.

A divisão de um algoritmo em módulos, além de facilitar a sua elaboração, permite uma melhor documentação e verificação de sua correção. Cada módulo, implementado como uma sub-rotina ou uma função, deve conter sua própria documentação e pode ser verificado independentemente. Isto torna eficiente o trabalho de equipes de programação, já que um algoritmo pode ser dividido entre vários programadores que podem implementar, testar e catalogar suas respectivas partes, separadamente.

Sub-rotinas e funções são módulos hierarquicamente subordinados a um algoritmo, comumente chamado de módulo principal. Da mesma forma, uma sub-rotina ou uma função pode conter outras sub-rotinas e funções aninhadas, como pode ser visto no diagrama hierárquico a seguir:

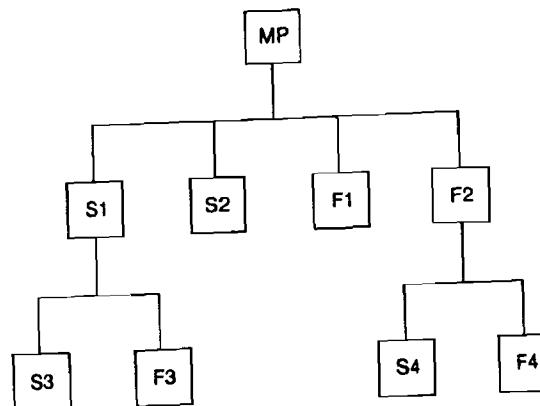


Fig. 3.1 Diagrama hierárquico.

A figura 3.1 ilustra um algoritmo constituído de um módulo principal MP, as sub-rotinas S1, S2, S3 e S4 e as funções F1, F2, F3 e F4.

A modularização, indicada no diagrama hierárquico anterior, é mostrada, a seguir, com a notação algorítmica.

```

Algoritmo (MP)
  subrotina S1
    subrotina S3
      ...
      fim subrotina
      função F3
      ...
      fim função
      fim subrotina
      subrotina S2
      ...
      fim subrotina
  
```

```

  função F1
  fim função
  função F2
  subrotina S4
    ...
    fim subrotina
    função F4
    ...
    fim função
    fim função
  
```

Tendo em vista os conceitos de objeto local e global, a função F3, declarada dentro da sub-rotina S1, é conhecida apenas em S1. Entretanto, a sub-rotina S2, declarada no módulo principal, é conhecida em todo o algoritmo e pode ser ativada dentro de S1.

A sub-rotina e a função são criadas através das suas declarações em um algoritmo. Para serem executadas, necessitam de ativação provocada por um comando de chamada.

A declaração de uma sub-rotina ou função é constituída de um **cabeçalho** e de um **corpo**. O cabeçalho, que identifica a sub-rotina ou função, contém o seu nome e a lista de parâmetros formais. O corpo contém declarações locais e os comandos da sub-rotina ou função. A ativação de uma sub-rotina ou função é feita através da referência a seu nome e a indicação dos parâmetros atuais.

3.2.1. Sub-rotina

Uma sub-rotina é declarada como a seguir:

```

subrotina NOME(lista-de-parâmetros-formais)
  declarações dos objetos locais à sub-rotina
  comandos da sub-rotina
  fim subrotina
  
```

onde:

subrotina	é uma palavra-chave;
NOME	é o nome dado à sub-rotina;
lista-de-parâmetros-formais	é a lista dos objetos que serão substituídos por outros objetos, fornecidos quando da chamada da sub-rotina.

A chamada de uma sub-rotina é feita com uma referência a seu nome e a indicação dos parâmetros atuais no local do algoritmo onde a sub-rotina deve ser ativada, ou seja, onde a sua execução deve ser iniciada.

A forma geral para o comando de ativação de uma sub-rotina é a seguinte:

```

NOME(lista-de-parâmetros-atuais)
  
```

onde:

NOME	é o nome dado à sub-rotina;
lista-de-parâmetros-atuais	é a lista de objetos que substituirão os parâmetros formais durante a execução da sub-rotina.
	Os parâmetros atuais devem concordar em número, ordem e tipo com os parâmetros formais.

Ao terminar a execução dos comandos de uma sub-rotina, o fluxo de controle retorna ao comando seguinte àquele que provocou a chamada. Desta maneira, a execução de uma sub-rotina se constitui em uma transferência temporária da execução do módulo chamador para a sub-rotina, retornando depois ao algoritmo que a chamou.

Exemplo 3.1

O exemplo 1.33 apresenta o problema "Dados três valores distintos, colocá-los em ordem crescente", cuja solução algorítmica encontrada é dada por:

Algoritmo

{Definição do tipo das variáveis}

declare AUXILIAR,L,M,N número

{Leitura dos números}

leia L,M,N
se L > M ou L < N
então se M < N
então AUXILIAR ← L
L ← M
M ← AUXILIAR
senão AUXILIAR ← L
L ← N
N ← AUXILIAR
final se

se M > N
então AUXILIAR ← M
M ← N
N ← AUXILIAR
final se

{Escrita do resultado}

escreva L,M,N
final algoritmo

Observa-se que há repetição de um grupo de comandos que diferem entre si devido às variáveis utilizadas:

AUXILIAR ← L
L ← M
M ← AUXILIAR

AUXILIAR ← L
L ← N
N ← AUXILIAR

AUXILIAR ← M
M ← N
N ← AUXILIAR

Todas estas repetições têm objetivo que é a troca de valores de 2 variáveis, podendo ser substituídas por uma única sub-rotina.

Algoritmo

(Declaração da sub-rotina TROCA)

subrotina TROCA(A,B)

declare A,B,AUX número

AUX ← A

A ← B

B ← AUX

fim subrotina

{Definição do tipo das variáveis}

declare L,M,N número

{Leitura dos números}

leia L,M,N

se L > M ou L > N

então se M < N

então TROCA(L,M)

senão TROCA(L,N)

fim se

fim se

se M > N

então TROCA(M,N)

fim se

{Escrita do resultado}

escreva L,M,N

final algoritmo.

Cada vez que a sub-rotina TROCA for ativada, os comandos dentro dela são executados tendo em vista os valores contidos nos parâmetros atuais e, em seguida, a seqüência do algoritmo retorna ao comando imediatamente seguinte ao da ativação.

Os parâmetros de uma sub-rotina classificam-se em:

- de entrada — são aqueles que têm seus valores estabelecidos fora da sub-rotina e não podem ser modificados dentro da sub-rotina;
- de saída — são aqueles que têm seus valores estabelecidos dentro da sub-rotina;
- de entrada-saída — são aqueles que têm seus valores estabelecidos fora da sub-rotina, mas podem ter seus valores alterados dentro dela.

A vinculação entre módulos pode ser feita através da transferência ou passagem de parâmetros, que associam parâmetros atuais com os parâmetros formais.

Dentre os modos de transferência de parâmetros, pode-se destacar a passagem por valor, a passagem por resultado e a passagem por referência.

Na passagem de parâmetros por valor, as alterações feitas nos parâmetros formais, dentro da sub-rotina, não se refletem nos parâmetros atuais. O valor do parâmetro atual é copiado no parâmetro formal, na chamada da sub-rotina. Assim, quando a passagem é por valor, isto significa que o parâmetro é de entrada.

Na passagem de parâmetros por resultado, as alterações feitas nos parâmetros formais, dentro da sub-rotina, refletem-se nos parâmetros atuais. O valor do parâmetro formal é copiado no parâmetro atual, ao retornar da sub-rotina. Assim, quando a passagem é por resultado, isto significa que o parâmetro é de saída.

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

No presente texto, será adotada apenas a passagem de parâmetros por referência, por atender a todas as necessidades. Desta forma, ao se elaborar uma sub-rotina, deve-se tomar cuidado em classificar os parâmetros entre os tipos mencionados (entrada, saída e entrada-saída), para evitar os efeitos colaterais decorrentes do modo de passagem utilizado.

Finalmente, é importante ressaltar que nas linguagens de programação existem, muitas vezes, mais de um modo de passagem de parâmetros com regras específicas para indicar qual o modo que se aplica a cada parâmetro, sendo necessário, às vezes, realizar adaptações no algoritmo.

Exemplo 3.2

No exemplo 2.14, pode-se observar que existem dois refinamentos de leitura que diferem entre si apenas na variável composta utilizada. Neste caso, pode ser usada uma sub-rotina:

```
subrotina LEITURA(NMAX,A)
    declare NMAX,I numérico
    declare A[I:NMAX] numérico
    I ← 1
    repita
        leia A[I]
        se A[I] = 9999
            então interrompa
        fim se
        I ← I + 1
    fim repita
fim subrotina
```

Na sub-rotina anterior, a declaração do parâmetro A é feita com o dimensionamento em função de um outro parâmetro de entrada, NMAX, que contém o número máximo de elementos que a variável composta A pode ter, ou seja, o valor do limite superior com que o parâmetro atual a ela associado foi declarado no módulo principal.

Como esta sub-rotina será utilizada para a leitura de duas variáveis unidimensionais, com número diferente de elementos (151 e 221), ao parâmetro atual NMAX deverá ser atribuído o valor 221.

Tendo em vista a sub-rotina LEITURA, o algoritmo final para o problema proposto no exemplo 2.14 passou a ser o seguinte:

Algoritmo

{Declaração da sub-rotina de leitura}

```
subrotina LEITURA(NMAX,A)
    declare NMAX,I numérico
    declare A[I:NMAX] numérico
    I ← 1
    repita
        leia A[I]
        se A[I] = 9999
            então interrompa
        fim se
        I ← I + 1
    fim repita
fim subrotina
```

{Declaração das variáveis}

```
declare PC[1:151], {matrícula em Programação de Computadores}
    CN[1:221] {matrícula em Cálculo Numérico}
    numérico
declare NMAX,I,J, numérico
NMAX ← 151
```

{Leitura da matrícula dos alunos que estão cursando PC}

```
LEITURA(NMAX,PC)
NMAX ← 221
```

{Leitura da matrícula dos alunos que estão cursando CN}

```
LEITURA(NMAX,CN)
{Verificação da matrícula simultânea em PC e CN}
```

I ←
repita

```
se PC[I] = 9999
    então interrompa
fim se
J ← 1
repita
    se CN[J] = 9999 ou PC[I] = CN[J]
        então interrompa
    fim se
    J ← J + 1
fim repita
se PC[I] = CN[J]
    então escreva PC[I]
fim se
I ← I + 1
fim repita
fim algoritmo.
```

É bom ressaltar, então, que, se uma variável composta é passada como parâmetro numa função ou sub-rotina, os limites superiores de seus índices, declarados no módulo principal, também têm de ser passados como parâmetros.

3.2.2. Função

As funções, embora bastante semelhantes às sub-rotinas, têm a característica especial de retornar ao algoritmo que as chamou um valor associado ao nome da função. Esta característica permite uma analogia com o conceito de função da Matemática.

No item 1.4 do capítulo 1, quando foram apresentadas as expressões aritméticas, foram usadas algumas funções que já são consideradas como itens fundamentais da notação algorítmica, adotada neste texto, como, por exemplo: RESTO, ABS, LOG etc.

A utilização de outras funções no algoritmo, como por exemplo, factorial, ou qualquer outra função especial, pode ser feita declarando-se uma função. A declaração de uma função é idêntica à de uma sub-rotina, com exceção de que é necessário o seu tipo, ou seja, o tipo do valor que será retornado. Além de numéricas, as funções podem ser lógicas ou literais.

Apresenta-se seguir a forma de declaração de funções:

```
função tipo NOME(lista-de-parâmetros-formais)
    declaração dos objetos locais à função
    comandos da função
fim função
```

onde:

função	é uma palavra-chave;
tipo	é o tipo do valor que será retornado;
NOME	é o nome dado à função;
lista-de-parâmetros-formais	é a lista dos objetivos que serão substituídos por outros objetos, fornecidos quando da chamada da função.

A chamada de uma função é feita com uma referência a seu nome e a indicação dos parâmetros atuais em uma expressão. A seguir, apresenta-se a forma geral para a ativação de uma função:

NOME(*Lista-de-parâmetros-atuais*)

onde:

NOME **lista-de-parâmetros-atuais**

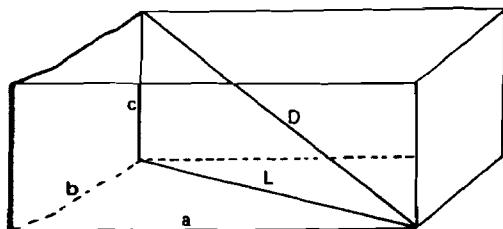
é o nome dado à função;
é a lista de objetos que substituirão os parâmetros formais durante a execução da função. Os parâmetros atuais devem concordar em números, ordem e tipo com os parâmetros formais.

O fluxo de controle é desviado para a função, no momento em que ela é ativada. Ao terminar a execução dos comandos da função, o fluxo de controle retorna ao comando seguinte àquele onde ela foi ativada.

São válidas aqui as considerações já feitas sobre passagem de parâmetros.

Exemplo 3.3

Escrever um algoritmo que leia as medidas dos três lados a , b e c de um paralelepípedo, calcule e escreva o valor de sua diagonal.



$$L = \sqrt{a^2 + b^2}$$

$$D = \sqrt{L^2 + c^2}$$

Fig. 3.2

função numérico HIPOTENUSA(*A,B*)

declare *A,B* numéricos

$$\text{HIPOTENUSA} \leftarrow \sqrt{A^2 + B^2}$$

fim função

O algoritmo principal é dado por:

Algoritmo

{Definição da função que calcula a hipotenusa}

função numérico HIPOTENUSA(*A,B*)

declare *A,B* numérico

$$\text{HIPOTENUSA} \leftarrow \sqrt{A^2 + B^2}$$

fim função

(Declaração do tipo das variáveis)

declare *A,B,C,D* numérico

(Leitura das dimensões do paralelepípedo)

leia *A,B,C*

(Cálculo do valor da diagonal do paralelepípedo)

$$D \leftarrow \text{HIPOTENUSA}(\text{HIPOTENUSA}(A,B),C)$$

(Escrita do valor da diagonal do paralelepípedo)

escreva *D*

fim algoritmo.

Pode-se observar neste algoritmo que o parâmetro atual de uma função pode ser a ativação de uma função. Neste caso, ao ser ativada no módulo principal a função HIPOTENUSA, o fluxo do algoritmo se transfere para a função, a fim de calcular a hipotenusa referente aos catetos a , b , c , em seguida, a função é novamente ativada para o cálculo da hipotenusa, referente aos catetos L e c .

Esta observação aplica-se, também, às sub-rotinas.

Exemplo 3.4

Escriver uma função lógica que receba:

- uma variável composta unidimensional *M*;
- o número *N* de elementos de *M*;
- um valor *X*

e devolva o valor verdadeiro, se *X* for igual a algum dos elementos de *M*, ou falso, em caso contrário.

Escriver um algoritmo que leia um conjunto de 20 números inteiros, seguido de outro conjunto de 10 números inteiros, e determine quais destes 10 números são iguais a um dos 20 primeiros.

função lógico EXISTE(*M,N,X*)

declare *M[1:N]* numérico

declare *N,X,I* numérico

I \leftarrow 1

repita

se *I* $>$ *N* ou *X* = *M[I]*
então interrompa

fim se

I \leftarrow *I* + 1

fim repita

EXISTE \leftarrow (*I* \leq *N*)

fim função

O módulo principal pode ser dado por:

Algoritmo

{Definição da função que verifica a existência de um número em um dado conjunto}

função lógico EXISTE(*M,N,X*)

declare *M[1:N]* numérico

declare *N,X,I* numérico

I \leftarrow 1

repita

se *I* $>$ *N* ou *X* = *M[I]*
então interrompa

fim se

I \leftarrow *I* + 1

fim repita

EXISTE \leftarrow (*I* \leq *N*)

fim função

{Definição do tipo das variáveis}

declare *M[1:20]* numérico

declare *N,X,J* numérico

leia *M[1]...M[20]*

N \leftarrow 20

J \leftarrow 1

```

repita
  se J > 10
    então interrompa
  fim se
  {Leitura do número}
  leia X
  se EXISTE(M,N,X)
    então escreva X
  fim se
  J ← J + 1
fim repita
fim algoritmo.

```

O valor da expressão lógica $I < N$ será o valor a ser retornado pela função EXISTE.

Exemplo 3.5

Escrever uma função que calcule a distância entre dois pontos de um plano, sendo fornecidas as coordenadas X_1, Y_1 e X_2, Y_2 . Escrever, ainda, um algoritmo que leia 10 linhas contendo, cada uma, as coordenadas de três pontos, $X_1, Y_1, X_2, Y_2, X_3, Y_3$ e determine a área do triângulo formado por estes pontos. Escreva, para cada linha, as coordenadas lidas e a área determinada.

Algoritmo

{Definição da função que calcula a distância}
função numérico DISTANCIA(A1,B1,A2,B2)

declare A1,B1,A2,B2 numérico

$$\text{DISTANCIA} \leftarrow \sqrt{(A2 - A1)^2 + (B2 - B1)^2}$$

fim função

{Declaração do tipo das variáveis}

declare X1,Y1,X2,Y2,X3,Y3,ÁREA,

LA,

(Lado A)

LB,

(Lado B)

LC,

(Lado C)

SP,

(Semiperímetro)

N

(Contador)

numérico

N ← 0

repita

{Leitura das coordenadas dos pontos}

leia X1,Y1,X2,Y2,X3,Y3

{Cálculo das medidas dos lados do triângulo}

LA ← DISTANCIA(X1,Y1,X2,Y2)

LB ← DISTANCIA(X1,Y1,X3,Y3)

LC ← DISTANCIA(X2,Y2,X3,Y3)

{Cálculo do semiperímetro}

SP ← (LA + LB + LC)/2

{Cálculo da área}

ÁREA ← $\sqrt{SP \times (SP - LA) \times (SP - LB) \times (SP - LC)}$

{Escrita dos resultados}

escreva X1,Y1,X2,Y2,X3,Y3,ÁREA

N ← N + 1

se N = 10

então interrompa

fim se
fim repita
fim algoritmo.

Exemplo 3.6

Fazer um algoritmo que:

- leia vários pares de números inteiros positivos, M e P. O último par terá o valor zero para M e P;
- calcule e escreva o número de arranjos e combinações de M elementos P a P, dado pelas fórmulas:

$$A_M^P = \frac{M!}{(M - P)!}$$

$$C_M^P = \frac{M!}{P!(M - P)!}$$

Se $M < P$, por definição

$$A_M^P = 0 \quad \text{e} \quad C_M^P = 0$$

Algoritmo

Defina o tipo das variáveis

repita

leia M,P

se M = 0 e P = 0

então interrompa

fim se

se M < P

então A ← 0

C ← 0

senão Calcule o fatorial de M

Calcule o fatorial de $(M - P)$

Calcule o arranjo de M elementos P a P

Calcule o fatorial de P

Calcule a combinação de M elementos P a P

fim se

escreva A,C

fim repita

fim algoritmo.

Pode-se observar que neste algoritmo existem três comandos, a serem refinados, que são iguais, exceto pelas variáveis envolvidas, o que demonstra a necessidade de se usar uma sub-rotina ou uma função. Como o módulo deverá retornar apenas um resultado, que será utilizado em expressões aritméticas, a escolha recaiu sobre uma função.

Transformando-se o algoritmo de cálculo de fatorial, desenvolvido no exemplo 1.37, em função e inserindo-a no algoritmo anterior, juntamente com o desenvolvimento dos refinamentos, tem-se:

Algoritmo

{Definição da função que calcula o fatorial de um n°}

função numérico FATORIAL(N)

{Definição do tipo do parâmetro e da variável local}

declare I,N numérico

FATORIAL ← 1

I ← 2

repita

se I > N

```

    | então interrompa
  fim se
  FATORIAL ← FATORIAL × I
  I ← I + 1
  fim repita
fim função
        (Definição do tipo das variáveis)
declare A,C,M,P,MP,FATM,FATMP numérico
repita
  leia M,P
  se M = 0 e P = 0
    então interrompa
  fim se
  se M < P
    então A ← 0
    C ← 0
    senão FATM ← FATORIAL(M)
      MP ← M - P
      FATMP ← FATORIAL(MP)
      {Cálculo do arranjo de M elementos P a P}
      A ← FATM/FATMP
      {Cálculo da combinação de M elementos P a P}
      C ← FATM/(FATORIAL(P) × FATMP)
  fim se
  escreva A,C
fim repita
fim algoritmo

```

Exemplo 3.7

Escrever uma função que integre numericamente, sobre um intervalo dado $(a, b) \subset (0, \pi)$, a função $\text{seno}(x)$ descrita na figura 3.3. Escrever, ainda, um algoritmo que leia várias linhas contendo, cada uma, um intervalo (X_1, X_N) de integração. Para cada intervalo de integração, calcular a área descrita pela função SENO usando 10, 20, 30, 40 e 50 subintervalos. A última linha, que não será considerada, contém os valores de X_1 e X_N iguais a zero.

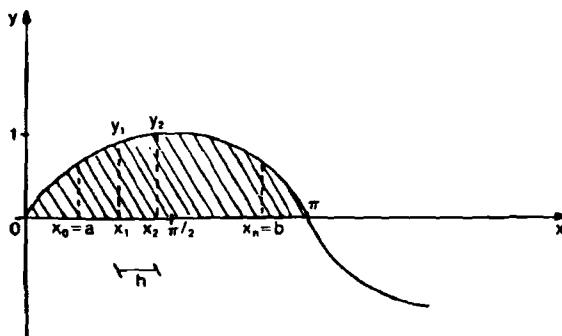


Fig. 3.3

Para o cálculo da integral, será utilizada a primeira regra de Simpson, dada por:

$$S = \int_a^b \text{sen}(x) dx = \frac{h}{3} (y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \dots + y_n)$$

onde:

h é constante e determinada por $h = x_{i+1} - x_i = \frac{b-a}{n}$;
 n é o número de subintervalos, $n > 1$ e par
 $y_i = \text{seno}(x_i)$

O algoritmo de integração é o seguinte:

```

função numérico INTEGRAL(A,B,N)
declare A,B,N,H,C,I,SOMA,X,Y numérico
  H ←  $\frac{B-A}{N}$ 
  SOMA ← SENO(A) + SENO(B)
  I ← 1
repita
  X ← A + I × H
  Y ← SENO(X)
  C ← 2 × (RESTO(I,2) + 1)
  SOMA ← SOMA + C × Y
  I ← I + 1
  se I = N
    então interrompa
  fim se
fim repita
INTEGRAL ←  $\frac{\text{SOMA} \times H}{3}$ 
fim função

```

Inserindo-se a função INTEGRAL no módulo principal, tem-se:

Algoritmo (Declaração da função que calcula a integral)

```

função numérico INTEGRAL(A,B,N)
declare A,B,N,H,C,I,SOMA,X,Y numérico
  H ←  $\frac{B-A}{N}$ 
  SOMA ← SENO(A) + SENO(B)
  I ← 1
repita
  X ← A + I × H
  Y ← SENO(X)
  C ← 2 × (RESTO(I,2) + 1)
  SOMA ← SOMA + C × Y
  I ← I + 1
  se I = N
    então interrompa
  fim se
fim repita
INTEGRAL ←  $\frac{\text{SOMA} \times H}{3}$ 
fim função
(Definição do tipo das variáveis utilizadas)

```

```

declare AREA,          {resultado da integração}
X1,                  {limite inferior do intervalo}
XN,                  {limite superior do intervalo}
N                   {número de subintervalos para integração}
numérico

repita               {Leitura do intervalo de integração}
    leia X1,XN
    se X1 = 0 e XN = 0
        então interrompa
    fim se
    N ← 10
    repita
        se N > 50
            então interrompa
        fim se
        ÁREA ← INTEGRAL(X1,XN,N)
        escreva "A ÁREA SOB A SENOIDE NO INTERVALO",X1,XN,
                "INTEGRADA COM",N,"PONTOS E IGUAL A",ÁREA
        N ← N + 10
    fim repita
fim repita
fun algoritmo

```

Exemplo 3.8

Fazer uma função que transforme horas, minutos e segundos em segundos. Ex. 2 h 40min 10seg → 9.610 segundos.

Fazer uma sub-rotina que transforme segundo em horas, minutos e segundos. Ex. 11.030 segundos → 3h 3min 50seg.

Fazer um algoritmo que:

- leia um conjunto de linhas contendo, cada uma, o número de um empregado, a hora de início (horas, minutos e segundos) e a hora do término (horas, minutos e segundos) de uma determinada tarefa. A última linha (FLAG) conterá o número do empregado negativo;
- calcule, para cada empregado, a duração da tarefa que ele executou, num mesmo dia, utilizando os dois módulos anteriormente definidos;
- escreva, para cada empregado, o seu número e a duração de sua tarefa em horas, minutos e segundos.

```

função numérico SEGUNDOS(H,M,S)
declare H,M,S numérico
SEGUNDOS ← H × 3600 + M × 60 + S
fim função

```

```

subrotina HORAS(SEG,H,M,S)
declare SEG,H,M,S numérico
H ← QUOCIENTE(SEG,3600)
M ← QUOCIENTE(RESTO(SEG,3600),60)
S ← RESTO(RESTO(SEG,3600),60)
fim subrotina

```

Inserindo-se estes módulos no módulo principal, tem-se:

Algoritmo {Transformação de horas, minutos e segundos em segundos} função numérico SEGUNDOS(H,M,S) declare H,M,S numérico SEGUNDOS ← H × 3600 + M × 60 + S fim função	{Transformação de segundos em horas, minutos e segundos} subrotina HORAS(SEG,H,M,S) declare SEG,H,M,S numérico H ← QUOCIENTE(SEG,3600) M ← QUOCIENTE(RESTO(SEG,3600),60) S ← RESTO(RESTO(SEG,3600),60) fim subrotina
{Definição do tipo das variáveis}	
declare NEMP, {número do empregado} HI, {horas de início} MI, {minutos de início} SI, {segundos de início} HT, {horas de término} MT, {minutos de término} ST, {segundos de término} DUR, {duração da tarefa em segundos} HD, {horas de duração} MD, {minutos de duração} SD, {segundos de duração} numérico	repita {Leitura dos horários de início e término da tarefa} leia NEMP,HI,MI,SI,HT,MT,ST se NEMP < 0 então interrompa fim se {Cálculo da duração da tarefa} DUR ← SEGUNDOS(HT,MT,ST) — SEGUNDOS(HI,MI,SI) HORAS(DUR,HD,MD,SD) escreva NEMP,HD,MD,SD fim repita fim algoritmo.

Exemplo 3.9

Fazer uma sub-rotina que calcule o máximo divisor comum entre dois números inteiros (método das divisões sucessivas ou “jogo da velha”).

Fazer um algoritmo que:

- leia 50 linhas, contendo, cada uma, um par de números inteiros;
- escreva, utilizando a sub-rotina anterior, os números primos entre si.

Dois números são primos entre si quando só admitem como divisor comum a unidade.

Algoritmo {Definição da sub-rotina que calcula o m.d.c.}

```

subrotina MDC(A,B,M)
  declare A,B,M,AUX1,AUX2 numérico
  M  $\leftarrow$  B
  AUX1  $\leftarrow$  RESTO(A,B)
  repita
    se AUX1 = 0
      então interrompa
    fim se
    AUX2  $\leftarrow$  M
    M  $\leftarrow$  AUX1
    AUX1  $\leftarrow$  RESTO(AUX2,M)
  fim repita
fim subrotina
  
```

{Definição do tipo das variáveis}

```

declare I,M,N1,N2 numérico
I  $\leftarrow$  1
repita
  {Leitura dos números}
  leia N1,N2
  MDC(N1,N2,M)
  se M = 1
    então escreva N1,N2
  fim se
  se I = 50
    então interrompa
  fim se
  I  $\leftarrow$  I + 1
fim repita
fim algoritmo.
  
```

Exemplo 3.10

Fazer uma sub-rotina que, dado um número inteiro N, retorne a soma dos divisores deste número, exceto ele próprio.

Fazer um algoritmo que, utilizando a sub-rotina anterior, determine e escreva todos os pares de números amigos em um intervalo [A,B]. Os valores de A e B ($A < B$), inteiros maiores que zero, deverão ser lidos.

Dois números inteiros M e N são amigos se a soma dos divisores de M, excluindo M, é igual a N e a soma dos divisores de N, excluindo N, é igual a M.

Antes de se elaborar um algoritmo para este problema, algumas observações se fazem necessárias:

- Se um número inteiro X possui um divisor Y menor que sua raiz quadrada, o quociente da divisão de X por Y será maior que a raiz quadrada de X e será, também, um divisor de X.

Exemplo:

$$X = 64$$

$$Y = 4$$

$$X/Y = 16 > \sqrt{64} \text{ e é, também, divisor de } 64.$$

- Se um número inteiro X possui um divisor Y igual a sua raiz quadrada, o quociente da divisão de X por Y será o próprio divisor Y.

Exemplo:

$$X = 64$$

$$Y = 8$$

$$X/Y = 8, \text{ que é igual a } \sqrt{64}$$

```

subrotina SDIV(N,SOMA)
  declare N,SOMA,DIVISOR numérico
  {Cálculo da soma dos divisores menores que a raiz quadrada de N e de seus respectivos quocientes}
  SOMA  $\leftarrow$  1
  DIVISOR  $\leftarrow$  2
  repita
    se DIVISOR  $\geq \sqrt{N}$ 
      então interrompa
    fim se
    se RESTO(N,DIVISOR) = 0
      então SOMA  $\leftarrow$  SOMA + DIVISOR + QUOCIENTE(N,DIVISOR)
    fim se
    DIVISOR  $\leftarrow$  DIVISOR + 1
  fim repita
  se DIVISOR =  $\sqrt{N}$ 
    então SOMA  $\leftarrow$  SOMA + DIVISOR
  fim se
fim subrotina
  
```

Inserindo-se a sub-rotina no módulo principal, tem-se:

Algoritmo

{Definição da sub-rotina que calcula os divisores de um número, exceto ele próprio}

```

subrotina SDIV(N,SOMA)
  declare N,SOMA,DIVISOR numérico
  {Cálculo da soma dos divisores menores que a raiz quadrada de N e de seus respectivos quocientes}
  SOMA  $\leftarrow$  1
  DIVISOR  $\leftarrow$  2
  repita
    se DIVISOR  $\geq \sqrt{N}$ 
      então interrompa
    fim se
    se RESTO(N,DIVISOR) = 0
      então SOMA  $\leftarrow$  SOMA + DIVISOR + QUOCIENTE(N,DIVISOR)
    fim se
    DIVISOR  $\leftarrow$  DIVISOR + 1
  fim repita
  se DIVISOR =  $\sqrt{N}$ 
    então SOMA  $\leftarrow$  SOMA + DIVISOR
  fim se
fim subrotina
  
```

{Definição do tipo das variáveis}

```

declare A,          {limite inferior do intervalo}
      B,          {limite superior do intervalo}
      SA,         {soma dos divisores próprios de A}
      SAMIGO     {soma dos divisores próprios de SA}
      numérico
  
```

{Leitura dos limites do intervalo}

leia A,B

{Cálculo da soma dos divisores próprios de A}

SDIV(A,SA)

{Verificação se SA está dentro do intervalo}

```
se SA ≥ A e SA ≤ B
  {Cálculo da soma dos divisores próprios de SA}
  então SDIV(SA,SAMIGO)
    se A = SAMIGO
      então escreva A,SA
      fim se
    fim se
    A ← A + 1
    se A = B
      então interrompa
    fim se
  fim repita
fim algoritmo.
```

NOTEX[1:NMAX]

{podem fazer exame especial}
{nota dos alunos que podem fazer}
{exame especial}

numérico

```
I ← 1
NEX ← 0
repita
  se I > N
    então interrompa
  fim se
  se F[I] ≥ FM e NOTA[I] ≥ 40 e NOTA[I] < 60
    então NEX ← NEX + 1
    NUMEX[NEX] ← NUM[I]
    NOTEX[NEX] ← NOTA[I]
  fim se
  I ← I + 1
fim repita
fim subrotina
```

Exemplo 3.11

Escrever um algoritmo que emita um relatório e forneça dados estatísticos a respeito de uma turma (no máximo, 50 alunos) do 1.º período do Ciclo Básico do Instituto de Ciências Exatas da UFMG. O algoritmo deve:

- ler a freqüência mínima exigida como requisito parcial para a aprovação do aluno em Geometria Analítica, Programação de Computadores e Cálculo I, nesta ordem;
- ler o número de matrícula de cada aluno e sua nota final em cada disciplina (na mesma ordem citada na alínea a), além de sua freqüência (ou seja, a quantas aulas assistiu) por disciplina;
- conter uma sub-rotina que verifique se um aluno tem direito de fazer exame especial numa disciplina (isto é, se o aluno é freqüente e obteve, no mínimo, 40 pontos);
- conter uma sub-rotina para calcular a percentagem dos alunos que já foram aprovados (sem exame especial) em uma disciplina, cuja nota mínima de aprovação é 60;
- conter uma sub-rotina que classifique os elementos de um conjunto de informações de duas grandes diferentes em ordem crescente de uma delas;
- determinar (utilizando a sub-rotina da alínea c) quais alunos podem fazer exame especial em cada disciplina;
- escrever, para cada disciplina, o número de matrícula dos alunos que têm condições legais de fazer o exame especial, em ordem crescente, com suas notas respectivas, utilizando a sub-rotina definida na alínea e;
- calcular e escrever, para cada disciplina, a percentagem de alunos já aprovados, utilizando a sub-rotina da alínea d e a percentagem dos que têm direito ao exame especial.

subrotina EXAME(NMAX,N,FM,NUM,NOTA,F,NUMEX,NOTEX,NEX)

```
declare NMAX,          {limite superior das variáveis compostas}
       N,            {número de alunos da turma}
       FM,           {freqüência mínima}
       NEX,          {número de alunos que podem fazer o exame}
                   {especial}
       I             {variável auxiliar}

declare NUM[1:NMAX],   {número de matrícula dos alunos}
       NOTA[1:NMAX], {nota dos alunos}
       F[1:NMAX],     {freqüência dos alunos}
       NUMEX[1:NMAX], {número de matrícula dos alunos que}
```

subrotina PERCENT(NMAX,N,FM,NOTA,PA)

```
declare NMAX,          {limite superior da variável composta}
       N,            {número de alunos da turma}
       FM,           {freqüência mínima}
       PA,           {percentagem de alunos aprovados}
       NA,           {número de alunos aprovados}
       I             {variável auxiliar}

declare NOTA[1:NMAX], {nota dos alunos da turma}
       F[1:NMAX]     {freqüência dos alunos da turma}
                   numérico

NA ← 0
I ← 1
repita
  se I > N
    então interrompa
  fim se
  se NOTA[I] ≥ 60 e F[I] ≥ FM
    então NA ← NA + 1
  fim se
  I ← I + 1
fim repita
PA ←  $\frac{NA \times 100}{N}$ 
fim subrotina
```

Transformando o exemplo 2.17, que coloca em ordem crescente os elementos de um conjunto, em sub-rotina e fazendo algumas adaptações para este problema, tem-se:

subrotina CLASSIFICA(NMAX,N,NUM,NOTA)

```
declare NMAX,          {limite superior das variáveis compostas}
       N,            {número de alunos}
```

```

I,J,  

AUX1,AUX2  

número  

declare  

  I,NUM[1:NMAX],  

  NOTA[1:NMAX]  

  número  

J ← 2  

repita  

  se J > N  

    então interrompa  

  fim se  

  AUX1 ← NUM[J]  

  AUX2 ← NOTA[J]  

  I ← J  

  repita  

    I ← I - 1  

    se I = 1 ou AUX1 > NUM[I]  

      então interrompa  

    fim se  

    NUM[I + 1] ← NUM[I]  

    NOTA[I + 1] ← NOTA[I]  

  fim repita  

  se I = 1  

    então NUM[I] ← AUX1  

    NOTA[I] ← AUX2  

  senão NUM[I + 1] ← AUX1  

    NOTA[I + 1] ← AUX2  

  fim se  

  J ← J + 1  

fim repita
fim subrotina

```

Como o número de alunos não foi dado, será utilizado como FLAG um número de matrícula negativo.
O algoritmo é dado por:

Algoritmo

```

{Declaração da sub-rotina que verifica quais os alunos
{que podem fazer exame especial}
subrotina EXAME(NMAX,N,FM,NUM,NOTA,F,NUMEX,NOTEK,NEX)
  declare
    NMAX,          {limite superior das variáveis compostas}
    N,             {número de alunos da turma}
    FM,            {freqüência mínima}
    NEX,           {número de alunos que podem fazer o exame}
    {especial}
    I,             {variável auxiliar}
  declare
    NUM[1:NMAX],   {número de matrícula dos alunos}
    NOTA[1:NMAX],  {nota dos alunos}
    F[1:NMAX],     {freqüência dos alunos}
    NUMEX[1:NMAX], {número de matrícula dos alunos}
    {que podem fazer exame especial}
    NOTEK[1:NMAX], {nota dos alunos que podem fazer}
    {exame especial}
    número

```

```

I ← 1
NEX ← 0
repita
  se I > N
    então interrompa
  fim se
  se F[I] ≥ FM e NOTA[I] ≥ 40 e NOTA[I] < 60
    então NEX ← NEX + 1
    NUMEX[NEX] ← NUM[I]
    NOTEK[NEX] ← NOTA[I]
  fim se
  I ← I + 1
fim repita
fim subrotina
{Declaração da sub-rotina que calcula a percentagem de
{alunos aprovados}
subrotina PERCENTE(NMAX,N,FM,NOTA,F,PA)
  declare
    NMAX,          {limite superior da variável composta}
    N,             {número de alunos da turma}
    FM,            {freqüência mínima}
    PA,            {percentagem de alunos aprovados}
    NA,            {número de alunos aprovados}
    I,             {variável auxiliar}
  declare
    NOTA[1:NMAX], {nota dos alunos da turma}
    F[1:NMAX],     {freqüência dos alunos da turma}
    número
  NA ← 0
  I ← 1
  repita
    se I > N
      então interrompa
    fim se
    se NOTA[I] ≥ 60 e F[I] ≥ FM
      então NA ← NA + 1
    fim se
    I ← I + 1
  fim repita
  PA ←  $\frac{NA \times 100}{N}$ 
fim subrotina

```

{Declaração da sub-rotina que classifica em ordem
{crescente a relação dos alunos que podem fazer exame
{especial}}

```

subrotina CLASSIFICA(NMAX,N,NUM,NOTA)
  declare
    NMAX,          {limite superior das variáveis
{compostas}}
    N,             {número de alunos}
    I,J,           {variáveis auxiliares}
    AUX1,AUX2     {variáveis auxiliares}
  declare
    NUM[1:NMAX],   {número de matrícula dos alunos}
    NOTA[1:NMAX]   {nota dos alunos}
    número
  J ← 2

```

```

repita
  se J > N
    então interrompa
  fim se
  AUX1 ← NUM[J]
  AUX2 ← NOTA[J]
  I ← J
  repita
    I ← I - 1
    se I = 1 ou AUX1 > NUM[I]
      então interrompa
    fim se
    NUM[I + 1] ← NUM[I]
    NOTA[I + 1] ← NOTA[I]
  fim repita
  se I = 1
    então NUM[I] ← AUX1
    NOTA[I] ← AUX2
  senão NUM[I+1] ← AUX1
    NOTA[I + 1] ← AUX2
  fim se
  J ← J + 1
  fim repita
fim subrotina

```

{Declaração das variáveis utilizadas}

```

declare FM[1:3].
  MAT[1:51].
  NOTAG[1:51].
  NOTAP[1:51].
  NOTAC[1:51].
  FREQG[1:51].
  FREQP[1:51].
  FREQC[1:51].
  NUMEX[1:51].
  NOTAEX[1:51]

  numérico
  NMAX,
  I,
  N,
  PA,
  NEX

```

{freqüência mínima exigida}

{número de matrícula dos alunos}

{nota dos alunos de Geom. Analítica}

{nota dos alunos de P. de Computadores}

{nota dos alunos de Cálculo I}

{freqüência dos alunos de Geom. Anal.}

{freqüência dos alunos de P. de Comp.}

{freqüência dos alunos de Cálculo I}

{número de matrícula dos alunos que podem fazer exame especial}

{nota dos alunos que podem fazer exame}

{especial}

{limite superior das variáveis compostas}

{variável auxiliar}

{número de alunos da turma}

{percentagem}

{número de alunos que podem fazer exame}

{especial}

NMAX ← 51

(Leitura da freqüência mínima exigida nas 3 disciplinas)

leia FM[1]...FM[3]

(Leitura da matrícula, notas e freqüências dos alunos)

I ← 1

repita

leia MAT[I],NOTAG[I],NOTAP[I],
 NOTAC[I],FREQG[I],FREQP[I],FREQC[I]

se MAT[I] < 0
 então interrompa

fim se

I ← I + 1

N ← I - 1

{Para a disciplina Geometria Analítica:}

{Verificação dos alunos que podem fazer exame especial}

EXAME(NMAX,N,FM[1],NUM,NOTAG,FREQG,NUMEX,NOTAEX,NEX)

{Escrita dos alunos que podem fazer exame especial}

CLASSIFICA(NMAX,NEX,NUMEX,NOTAEX)

I ← 1

repita

se I > NEX
 então interrompa

fim se

escreva NUMEX[I],NOTAEX[I]

I ← I + 1

fim repita

{Escrita da percentagem dos alunos já aprovados}

PERCENT(NMAX,FM[1],N,NOTAG,FREQG,PA)

escreva PA

{Escrita da percentagem dos alunos que podem fazer exame especial}

PA ← $\frac{NEX \times 100}{N}$

escreva PA

{Para a disciplina Programação de Computadores:}

{Verificação dos alunos que podem fazer exame especial}

EXAME(NMAX,N,FM[2],NUM,NOTAP,FREQP,NUMEX,NOTAEX,NEX)

{Escrita dos alunos que podem fazer exame especial}

CLASSIFICA(NMAX,NEX,NUMEX,NOTAEX)

I ← 1

repita

se I > NEX
 então interrompa

fim se

escreva NUMEX[I],NOTAEX[I]

I ← I + 1

fim repita

{Escrita da percentagem dos alunos já aprovados}

PERCENT(NMAX,N,FM[2],NOTAP,FREQP,PA)

escreva PA

{Escrita da percentagem dos alunos que podem fazer exame especial}

PA ← $\frac{NEX \times 100}{N}$

escreva PA

{Para a disciplina Cálculo I:}

{Verificação dos alunos que podem fazer exame especial}

EXAME(NMAX,N,FM[3],NUM,NOTAC,FREQC,NUMEX,NOTAEX,NEX)

{Escrita dos alunos que podem fazer exame especial}

CLASSIFICA(NMAX,NEX,NUMEX,NOTAEX)

I ← 1

repita

se I > NEX
 então interrompa

fim se

escreva NUMEX[I],NOTAEX[I]

| I ← I + 1
fin repita

PERCEN(NMAX,N,FM[3],NOTAC,FREQC,PA)
escreva PA

(Escrita da percentagem dos alunos já aprovados)
(exame especial)

$$PA \leftarrow \frac{NEX \times 100}{N}$$

escreva PA

fim algoritmo.

Pode-se notar que, para cada disciplina, ainda existe um grupo de comandos que se repete, diferindo, apenas, pelas variáveis envolvidas. Portanto, faz-se necessária a utilização de mais uma sub-rotina declarada como se segue:

subrotina DISCIPLINA[NMAX,N,FM,NUM,NOTA,FREQ]

{Declaração dos parâmetros e variáveis utilizadas}
(limite superior das variáveis compostas)

declare NMAX,
N,
FM,
NEX,
I
numérico

{número de alunos da turma}
(frequência mínima)

{número de alunos que podem fazer o exame}
(especial)

{variável auxiliar}

declare NUM[1:NMAX],
NOTA[1:NMAX],
FREQ[1:NMAX],
NUMEX[1:NMAX],
NOTAEX[1:NMAX]
numérico

{número de matrícula dos alunos}
(nota dos alunos)

{frequência dos alunos}

{número de matrícula dos alunos}

{que podem fazer exame especial}

{nota dos alunos que podem fazer}
(exame especial)

EXAME(NMAX,N,FM,NUM,NOTA,FREQ,NUMEX,NOTAEX,NEX)

{Verificação dos alunos que podem fazer exame especial}
(Escrita dos alunos que podem fazer exame especial)

CLASSIFICA(NMAX,NEX,NUMEX,NOTAEX)

I ← 1

repita

se I > NEX

então interrompa

fin se

escreva NUMEX[I],NOTAEX[I]

I ← I + 1

fin repita

PERCEN(NMAX,N,FM,NOTA,FREQ,PA)

escreva PA

(Escrita da percentagem dos alunos já aprovados)
(exame especial)

$$PA \leftarrow \frac{NEX \times 100}{N}$$

escreva PA

fim subrotina

O algoritmo é o seguinte:

Algoritmo

{Declaração da sub-rotina que verifica quais os alunos}
(que podem fazer exame especial)

subrotina EXAME(NMAX,N,FM,NUM,NOTA,F,NUMEX,NOTELEX,NEX)

declare NMAX,
N,
FM,
NEX,

{limite superior das variáveis compostas}
(número de alunos da turma)

{frequência mínima}
(número de alunos que podem fazer o exame)

{especial}

{variável auxiliar}

I

numérico

declare

NUM[1:NMAX],
NOTA[1:NMAX],

{número de matrícula dos alunos}
(nota dos alunos)

F[1:NMAX],
NUMEX[1:NMAX],

{frequência dos alunos}
(número de matrícula dos alunos)

{que podem fazer exame especial}
(nota dos alunos que podem fazer)

NOTELEX[1:NMAX]
numérico

I ← 1

NEX ← 0

repita

se I > N

então interrompa

fin se

se F[I] ≥ FM e NOTA[I] ≥ 40 e NOTA[I] < 60

então NEX ← NEX + 1

NUMEX[NEX] ← NUM[I]

NOTELEX[NEX] ← NOTA[I]

fin se

I ← I + 1

fin repita

fim subrotina

{Declaração da sub-rotina que calcula a percentagem de}
(alunos aprovados)

subrotina PERCEN(NMAX,N,FM,NOTA,F,PA)

declare NMAX,
N,
FM,
PA,

{limite superior da variável composta}
(número de alunos da turma)

{frequência mínima}

{percentagem de alunos aprovados}

{número de alunos aprovados}

I,

numérico

declare

NOTA[1:NMAX],

{nota dos alunos da turma}

F[1:NMAX],

{frequência dos alunos da turma}

numérico

NA ← 0

I ← 1

repita

se I > N

então interrompa

fin se

se NOTA[I] ≥ 60 e F[I] ≥ FM

```

    | então NA ← NA + 1
    | fim se
    |   I ← I + 1
    | fim repita
    PA ←  $\frac{NA \times 100}{N}$ 
fim subrotina
    {Declaração da sub-rotina que classifica em ordem}
    {crescente a relação dos alunos que podem fazer exame}
    {especial}

subrotina CLASSIFICA(NMAX,N,NUM,NOTA)
declare NMAX,          {limite superior das variáveis}
       {compostas}
       N,           {número de alunos}
       I,J,         {variáveis auxiliares}
       AUX1,AUX2  {variáveis auxiliares}
numérico
declare NUM[1:NMAX],  {número de matrícula dos alunos}
       NOTA[1:NMAX] {nota dos alunos}
numérico
J ← 2
repita
  se J > N
    então interrompa
  fim se
  AUX1 ← NUM[J]
  AUX2 ← NOTA[J]
  I ← J
  repita
    I ← I - 1
    se I = 1 ou AUX1 > NUM[I]
      então interrompa
    fim se
    NUM[I + 1] ← NUM[I]
    NOTA[I + 1] ← NOTA[I]
  fim repita
  se I = 1
    então NUM[I] ← AUX1
    NOTA[I] ← AUX2
  senão NUM[I + 1] ← AUX1
    NOTA[I + 1] ← AUX2
  fim se
  J ← J + 1
fim repita
fim subrotina
    {Declaração da sub-rotina que escreve os resultados}
    {para cada disciplina}

subrotina DISCIPLINA(NMAX,N,FM,NUM,NOTA,FREQ)
    {Declaração dos parâmetros e variáveis utilizadas}
declare NMAX,          {limite superior das variáveis compostas}
       N,           {número de alunos da turma}
       FM,          {freqüência mínima}
       NEX,         {número de alunos que podem fazer o exame}
       I,           {especial}
       I            {variável auxiliar}

```

```

declare NUM[1:NMAX],  {número de matrícula dos alunos}
       NOTA[1:NMAX], {nota dos alunos}
       FREQ[1:NMAX], {freqüência dos alunos}
       NUMEX[1:NMAX], {número de matrícula dos alunos}
       NOTAEX[1:NMAX] {que podem fazer exame especial}
       NOTAEX[1:NMAX] {nota dos alunos que podem fazer exame}
       {especial}
numérico
       {Verificação dos alunos que podem fazer exame especial}
EXAME(NMAX,N,FM,NUM,NOTA,FREQ,NUMEX,NOTAEX,NEX)
       {Escrita dos alunos que podem fazer exame especial}
CLASSIFICA(NMAX,NEX,NUMEX,NOTAEX)
I ← 1
repita
  se I > NEX
    então interrompa
  fim se
  escreva NUMEX[I],NOTAEX[I]
  I ← I + 1
fim repita
    {Escrita da percentagem dos alunos já aprovados}
PERCEN(NMAX,N,FM,NOTA,FREQ,PA)
escreva PA
    {Escrita da percentagem dos alunos que podem fazer}
    {exame especial}
PA ←  $\frac{NEX \times 100}{N}$ 
escreva PA
fim subrotina
    {Declaração das variáveis utilizadas}
declare FM[1:3],        {freqüência mínima exigida}
       MAT[1:51],       {número de matrícula dos alunos}
       NOTAG[1:51],     {nota dos alunos de Geom. Analítica}
       NOTAP[1:51],     {nota dos alunos de P. de Computadores}
       NOTAC[1:51],     {nota dos alunos de Cálculo I}
       FREQG[1:51],     {freqüência dos alunos de Geom. Anal.}
       FREQP[1:51],     {freqüência dos alunos de P. de Comp.}
       FREQC[1:51],     {freqüência dos alunos de Cálculo I}
numérico
declare NMAX,          {limite superior das variáveis compostas}
       I,             {variável auxiliar}
       N,             {número de alunos da turma}
       NMAX ← 51
       {Leitura da freqüência mínima exigida nas 3 disciplinas}
leia FM[1]...FM[3]
       {Leitura da matrícula, notas e freqüências dos alunos}
I ← 1
repita
  leia MAT[I],NOTAG[I],NOTAP[I],NOTAC[I],FREQG[I],FREQP[I],FREQC[I]
  se MAT[I] < 0
    então interrompa
  fim se
  I ← I + 1

```

Final repita
N ← 1 - 1

{Para a disciplina Geometria Analítica:}

DISCIPLINA(NMAX,N,FM[1],NUM,NOTAG,FREQG)
(Para a disciplina Programação de Computadores)

DISCIPLINA(NMAX,N,FM[2],NUM,NOTAP,FREQP)
(Para a disciplina Cálculo I:)

DISCIPLINA(NMAX,N,FM[3],NUM,NOTAC,FREQC)

Final algoritmo

3.3. CONSIDERAÇÕES SOBRE A MODULARIZAÇÃO DE PROGRAMAS

Existem várias vantagens e, naturalmente, algumas desvantagens na modularização de programas. Algumas destas desvantagens são específicas de cada linguagem de programação que se escolha para a implementação do algoritmo.

Dentre as vantagens da modularização de programas podem-se citar:

1. **Partes comuns** a vários programas ou que se repetem dentro de um mesmo programa, quando modularizadas em uma sub-rotina ou função, são programadas e testadas uma só vez, mesmo que tenham que ser executadas com variáveis diferentes.
2. Podem-se constituir bibliotecas de programas, isto é, uma coleção de módulos que podem ser usados em diferentes programas sem alteração e mesmo por outros programadores. Por exemplo, as funções pré-definidas das linguagens de programação, como citadas no item anterior, constituem uma biblioteca de módulos.
3. A modularização dos programas permite preservar na sua implementação os refinamentos obtidos durante o desenvolvimento dos algoritmos.
4. **Economia** de memória do computador, uma vez que o módulo é armazenado uma única vez, mesmo que utilizado em diferentes partes do programa. Permite, também, que, em um determinado instante da execução do programa, estejam na memória principal apenas o módulo ou os módulos necessários à execução desse trecho de programa.

Dentre as desvantagens pode-se citar que existe um acréscimo de tempo na execução de programas constituídos de módulos, devido ao tratamento adicional de ativação do módulo etc.

Conclui-se que, sem dúvida, a modularização dos programas é uma técnica altamente recomendável, tanto pela eficiência no projeto e desenvolvimento dos mesmos quanto pela quantidade, confiabilidade e manutenção do produto elaborado.

Finalmente, o critério de modularização, utilizado nos exemplos apresentados, foi o de "funcionalidade", isto é, a tarefa de construção do algoritmo foi dividida em partes, segundo a função que estas teriam de executar. Entretanto, existem outros critérios para modularização. Dentre eles, pode-se citar o "ocultamento da informação" [Parnas, 1972], que geralmente produz algoritmos cuja manutenção posterior é mais efetiva.

Estes e outros detalhes, referentes à modularização, são tratados de forma mais adequada na disciplina denominada "Engenharia de Programas" [STAA, 1983], cujo conteúdo foge ao escopo deste texto.

3.4. EXERCÍCIOS PROPOSTOS

PROBLEMAS GERAIS

△ 3.4.1. Refazer o exercício proposto em 1.12.18, do capítulo 1, usando uma função para calcular o número de pontos de uma etapa. Esta função deve ser ativada para se obter os pontos de cada etapa para cada equipe.

△ 3.4.2. Reescrever o algoritmo do exemplo 1.41, do capítulo 1, sendo que o cálculo da potência de cada comando deve ser feito por uma função.

△ 3.4.3. A avaliação de aproveitamento de uma certa disciplina é feita através de 4 provas mensais no valor de 20 pontos e uma prova final no valor de 40 pontos. A nota final é obtida somando-se as 3 melhores notas, dentre as provas mensais, com a nota da prova final.

O conceito final é dado atendendo-se ao seguinte critério:

de 90 a 100 — conceito A
de 80 a 89 — conceito B
de 70 a 79 — conceito C
de 60 a 69 — conceito D
de 40 a 59 — conceito E
de 0 a 39 — conceito F

Fazer uma sub-rotina que, recebendo como parâmetro 4 (quatro) números inteiros, devolva ao módulo que a chamou a soma dos 3 (três) maiores números dentre os 4 (quatro) números recebidos.

Fazer um algoritmo que:

- leia um conjunto de 80 linhas contendo, cada uma, o número do aluno, as 4 notas mensais e a nota da prova final;
- calcule, para cada aluno, sua nota final, utilizando a sub-rotina anterior;
- verifique o conceito obtido;
- escreva, para cada aluno, o seu número, a sua nota final e o seu conceito.

△ 3.4.4. Fazer um algoritmo para um programa de apostas da LOTO. O algoritmo deverá ler, inicialmente, as cinco dezenas sorteadas e, a seguir, ler várias linhas, uma para cada aposta, contendo:

- número da aposta;
- quantidade de dezenas apostadas (no máximo, 10);
- as dezenas apostadas.

A última linha, que não entrará nos cálculos, conterá o número da aposta igual a zero.

O algoritmo deverá escrever o número de todas as apostas que tiverem três, quatro ou cinco dezenas sorteadas e, ao final, a quantidade de apostadores que fizeram o terço (três dezenas sorteadas), a quadra (quatro dezenas sorteadas) e a quina (cinco dezenas sorteadas). Neste algoritmo, deverá ser utilizada uma sub-rotina que faça a avaliação do número de pontos de cada aposta.

△ 3.4.5. Construir uma função que receba como parâmetro de entrada um número inteiro positivo e devolva um dígito verificador conforme o processo de cálculo descrito no exercício 2.5.3.6.

Escrever um algoritmo capaz de:

- ler um conjunto indeterminado de linhas contendo, cada uma, o nome de uma pessoa e seu número de CPF (n.º de inscrição no Cadastro de Pessoas Físicas);
- imprimir, para cada pessoa, os seus dados de entrada mais a mensagem "VÁLIDO" ou "INVÁLIDO", conforme a situação do número de CPF.
- utilize a função acima para calcular os dígitos verificadores.

Obs: Um n.º de CPF é validado através de seus dois últimos dígitos (dígitos verificadores, denominados controle). Por exemplo, o CPF de número 23086025620 é validado pelos dígitos verificadores 20. O esquema de verificação é o seguinte:

função	→ dígito verificador igual a 2
função	→ dígito verificador igual a 0

△ 3.4.6. Fazer uma sub-rotina que, dados N números, determine o número que apareceu mais vezes. Supõe que os valores possíveis de cada número estão entre 1 e 6, inclusive, e que sempre haverá um único número vencedor.

Sabendo-se que um jogo de dados ocorre 40 vezes por dia e que a cada dia é digitada uma linha contendo os 40 números que saíram, fazer um algoritmo que:

- leia os dados contidos em 30 linhas, correspondentes a um mês de jogo;
- determine o número ganhador do dia, utilizando-se a sub-rotina anterior;
- escreva este número e a mensagem "RESULTADO DIARIO";
- verifique também qual o número ganhador do mês;
- escreva este número e a mensagem "RESULTADO MENSAL DO JOGO".

△ 3.4.7. Fazer uma sub-rotina, cujo cabeçalho é dado por:

QUANTOSDIAS(DIA,MES,ANO,N)

onde:

DIA, MES e ANO são parâmetros de entrada;

N é um parâmetro de saída que conterá o número de dias do ano até a data fornecida.

cada mês. O mês de fevereiro pode ter 28 ou 29 dias, dependendo de o ANO ser bissexto ou não (ver o exercício 1.12.23).

Fazer um algoritmo que:

- leia um conjunto de linhas contendo, cada uma, duas datas. A última linha, que será utilizada como flag, conterá os valores 0, 0, 0, 0, 0, 0;
- verifique se as datas estão corretas ($1 \leq MES \leq 12$, dia de acordo com o mês e se ambas estão dentro do mesmo ano). Se alguma das datas não estiver correta, escreva "DATA INCORRETA" e os valores de DIA, MES e ANO das duas datas;
- verifique, se as datas estiverem corretas, qual a diferença, em dias, entre essas duas datas;
- escreva as datas lidas e a diferença entre elas.

△ 3.4.8. Escrever um algoritmo que leia um conjunto de 1.000 linhas contendo, cada uma, uma palavra em inglês e a sua tradução em português. Em seguida, leia um número indeterminado de linhas contendo, cada uma:

- a letra I (indicando inglês) e uma palavra qualquer das 1.000 em inglês; ou
- a letra P (indicando português) e uma palavra qualquer das 1.000 em português.

Para cada uma dessas linhas, escrever a palavra lida e a sua tradução. A última linha, indicando o fim de dados, terá a primeira letra diferente de I e P.

A tradução da palavra lida deve ser feita através de uma sub-rotina que recebe as listas de palavras em inglês e português, a letra I ou P e a palavra que se deseja traduzir, devolvendo a tradução da mesma.

PROBLEMAS DE APLICAÇÃO EM CIÉNCIAS EXATAS

▲ 3.4.9. Escrever uma função que receba dois números inteiros, positivos, e determine o produto dos mesmos, utilizando o seguinte método de multiplicação:

- dividir, sucessivamente, o primeiro número por 2, até que se obtenha 1 como quociente;
- paralelamente, dobrar, sucessivamente, o segundo número;
- somar os números da segunda coluna que tenham um número ímpar na primeira coluna. O total obtido é o produto procurado.

Exemplo:

$$\begin{array}{r} 9 \times 6 \\ 9 \quad 6 \rightarrow \quad 6 \\ 4 \quad 12 \\ 2 \quad 24 \\ 1 \quad 48 \rightarrow \quad + 48 \\ \hline 54 \end{array}$$

A seguir, escrever um algoritmo que leia 10 pares de números, calcule e escreva os respectivos produtos, usando a função anterior.

▲ 3.4.10. Determinar os números inteiros, menores que 5.000, que são quadrados perfeitos e, também, são capicuas.

Capicuas são números que têm o mesmo valor se lidos da esquerda para a direita ou da direita para a esquerda. Exemplo: 44, 232, 1661, etc.

Deverão ser escritos os seguintes algoritmos:

- um módulo principal;
- uma função que calcule quantos algarismos tem um determinado número inteiro;
- uma sub-rotina para separar um número em n algarismos;
- uma sub-rotina para formar o número na ordem inversa.

△ 3.4.11. Dado um polinômio na forma

$$P = a_1x^n + a_2x^{n-1} + \dots + a_nx + a_{n+1}$$

a) Fazer uma sub-rotina que retorne o valor do polinômio e o de sua derivada no ponto x, recebendo como parâmetros de entrada a ordem do polinômio, os coeficientes e o x.

b) Fazer um algoritmo que:

- leia a ordem do polinômio e os seus coeficientes;
- utilizando a sub-rotina da alínea a, calcule o valor do polinômio e o de sua derivada para diversos valores de x. Esses valores deverão estar digitados um por linha, sendo que a primeira linha desse conjunto de dados contém o número de valores de x a serem lidos;

△ 3.4.12. Escrever uma sub-rotina que calcule a distância entre dois vetores:

$$X = [x_1, x_2, \dots, x_n] \text{ e } Y = [y_1, y_2, \dots, y_n]$$

Escrever um algoritmo que, utilizando a sub-rotina anterior, calcule e escreva a distância entre M pares de vetores. O valor de M será fornecido.

▲ 3.4.13. Calcular e escrever a área total de 10 tetraedros, dadas as coordenadas de cada um de seus quatro vértices. Para tanto, deverão ser utilizadas as seguintes sub-rotinas:

- que calcula a distância entre dois pontos do espaço;
- que calcula a área de um triângulo em função de seus lados ($\text{AREA} = \sqrt{\rho \times (\rho - a) \times (\rho - b) \times (\rho - c)}$, onde ρ é o semi-perímetro do triângulo).

△ 3.4.14. Escrever uma sub-rotina que calcule o valor de π através da série

$$S = 1 - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \dots \text{, sendo } \pi = \sqrt{32 \times S}$$

Deverá ser fornecido à sub-rotina o número de termos da série para o cálculo de π .

Escrever um algoritmo que, fornecendo à sub-rotina, sucessivamente, o número de termos (1, 2, 3, ..., N), escreva uma tabela com o valor de π e número de termos utilizados. O valor de N deverá ser lido.

▲ 3.4.15. Escrever uma sub-rotina que calcule o valor de e através da série:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

O número de termos da série deverá ser fornecido à sub-rotina como parâmetro.

Escrever um algoritmo que, utilizando a sub-rotina anterior, determine o número de termos da série necessário para calcular o valor de e, cuja diferença em relação ao valor obtido através da função EXP(1) seja menor que 0,0001.

△ 3.4.16. Escrever uma sub-rotina que determine o conjunto interseção entre dois conjuntos A e B de caracteres.

Escrever uma sub-rotina que determine o conjunto união entre esses mesmos conjuntos A e B.

Escrever um algoritmo que leia 50 pares de conjuntos de 100 caracteres cada, determine e escreva a interseção e a união desses conjuntos, utilizando as sub-rotinas anteriormente definidas.

▲ 3.4.17. Segundo a conjectura de Goldbach, qualquer número par, maior que 2, pode ser escrito como a soma de dois números primos.

Exemplo:

$$8 = 3 + 5, 16 = 11 + 5, 68 = 31 + 37 \text{ etc.}$$

Dado um conjunto de números inteiros positivos, pares, fazer um algoritmo que calcule, para cada número, um par de números primos cuja soma seja igual ao próprio número. Adotar como flag um número negativo.

Para verificar se um número é primo, fazer uma sub-rotina que deverá retornar em uma variável lógica o valor verdadeiro, se o número for primo, e falso, em caso contrário.

△ 3.4.18. Fazer uma sub-rotina que, recebendo como parâmetro dois conjuntos de números inteiros e os tamanhos destes conjuntos, devolva ao módulo principal um outro conjunto de números inteiros, contendo a interseção dos dois conjuntos recebidos e o tamanho desse novo conjunto formado.

Fazer uma sub-rotina que, recebendo como parâmetro um número inteiro, devolva ao módulo principal um conjunto de números inteiros, contendo todos os divisores do número recebido e o tamanho desse conjunto.

Fazer um algoritmo que:

- leia um conjunto de 30 pares de números inteiros;
- escreva, para cada par de números lidos, os seus valores e os seus divisores comuns, fazendo uso das sub-rotinas anteriormente definidas.

Conclusões

A técnica apresentada neste livro para o desenvolvimento de algoritmos é útil e indicada mesmo quando não se cogita da utilização dos computadores. Ao se defrontar com problemas de organização de uma empresa, do estabelecimento de uma política econômica, do projeto de uma obra de engenharia, da fixação do plano de ação de uma equipe de pesquisa, do planejamento de uma novela, romance ou mesmo um livro didático, como este, as técnicas de desenvolvimento estruturado de algoritmos podem e devem ser empregadas. Os problemas devem ser abordados, primeiro, na sua generalidade e somente depois, gradativamente, nas suas particularidades. As ações devem ser consideradas na sua sequenciação, na sua condicionalidade e na sua repetição.

Por outro lado, o desenvolvimento de um algoritmo capaz de fazer com que as operações básicas de um computador conduzam à solução de um problema, até a obtenção dos seus resultados finais, é apenas uma das fases necessárias para o uso do computador.

Particularmente, para se resolver um problema num computador, é necessário passar pelas fases descritas a seguir.

4.1. DEFINIÇÃO DO PROBLEMA

Antes de se utilizar o computador, é necessário saber qual é o problema que se deseja resolver. É necessário caracterizá-lo de uma maneira clara e completa.

4.2. DESENVOLVIMENTO DE UM ALGORITMO

Definido o problema, passa-se em seguida à fase de desenvolvimento de um algoritmo capaz de produzir os resultados desejados. Nesta fase, quanto mais complexo for o problema, mais se recomenda a utilização da técnica apresentada neste livro.

4.3. CODIFICAÇÃO

Desenvolvido satisfatoriamente um algoritmo, faz-se nesta fase a sua transcrição para uma linguagem de programação aceita pelo computador.

4.4. DIGITAÇÃO

Escrito o programa, na linguagem escolhida, é necessário introduzi-lo no computador, a fim de executá-lo. Faz-se então a sua digitação num teclado, ligado a um computador diretamente ou através de um terminal.

4.5. PROCESSAMENTO DO PROGRAMA

Esta fase é feita pelo próprio computador, verificando a correção sintática do programa, considerando o significado de suas sentenças e procedendo a sua execução. Em alguns casos, a análise das sentenças do programa e a sua execução são realizadas alternadamente: diz-se, então, que o programa está sendo interpretado. Em outros casos, é feita, antes, a tradução de todo o programa para a linguagem de máquina e somente depois se processa a sua execução: diz-se, neste caso, que o programa foi compilado.

4.6. ANÁLISE DOS RESULTADOS

A obtenção de resultados através de um computador não é a garantia de que eles estejam corretos, mas apenas uma indicação de que o programa está escrito sem erros de linguagem. Geralmente, o computador não executa programas com erro de linguagem, mas emite mensagens, tentando localizar o erro e determinar a sua natureza. Por consequência, os primeiros resultados de cada alternativa contida num programa devem ser cuidadosamente verificados.

No item 0.5 faz-se uma referência às principais linguagens de programação em uso, atualmente, e de como proceder-se à sua escolha. Estas linguagens serão apresentadas de uma maneira sistemática, uniforme e unificada nos volumes seguintes da coleção que ora se inicia e tem por base este livro *Algoritmos Estruturados*.

Apêndices

Exercícios Resolvidos

Neste apêndice são apresentadas possíveis soluções para alguns dos exercícios formulados no livro. Foram escolhidos os exercícios mais significativos de cada capítulo e que representam uma classe de problemas semelhantes.

Todo exercício, cuja solução está incluída neste apêndice, é destacado, na lista correspondente do capítulo, através de uma marca em negrito. Por exemplo, o exercício de número 3.4.17 aparece na lista de exercícios propostos do capítulo 3 na forma ▲ 3.4.17.

Os algoritmos são agrupados por capítulos na mesma ordem em que foram originalmente formulados.

Todos os algoritmos presentes neste apêndice foram implementados e devidamente testados na linguagem Pascal, sendo que os programas encontram-se listados no Apêndice A do livro *Pascal Estruturado*.

Capítulo 1 - Exercícios propostos

▲ 1.12.2.

Algoritmo

```

declare ALTURA, {Definição do tipo das variáveis}
          PESSOAS, {Altura das pessoas}
          MAIOR, {Número de pessoas}
          MENOR, {Maior altura do grupo}
          MULHERES, {Menor altura do grupo}
          HOMENS, {Número de mulheres}
          SOMA, {Número de homens}
          MÉDIA, {Soma das alturas das mulheres}
          numérico {Média das alturas das mulheres}

declare SEXO {Sexo: masculino ou feminino}
          literal

          {Atribuição dos valores iniciais necessários}

MULHERES ← 0
SOMA ← 0
MAIOR ← 0
MENOR ← 5
PESSOAS ← 1

repita
    {Leitura da altura e sexo de cada pessoa}
    leia ALTURA, SEXO
    {Determinação da soma das alturas das mulheres}
    se SEXO = "FEMININO"
        então SOMA ← SOMA + ALTURA
        MULHERES ← MULHERES + 1
    fim se
    {Determinação da maior e menor alturas do grupo}
    se ALTURA > MAIOR
        então MAIOR ← ALTURA
    fim se
    se ALTURA < MENOR
        então MENOR ← ALTURA
    fim se
    PESSOAS ← PESSOAS + 1
    se PESSOAS > 50
        então interrompa
    fim se
fim repita
    {Cálculo da média de altura das mulheres}
    MÉDIA ← SOMA / MULHERES
    {Cálculo do número de homens}
    HOMENS ← PESSOAS - MULHERES
    {Escrita da maior e da menor altura}
    escreva MAIOR, MENOR

```

{Escrita da média de altura das mulheres}

escreva MÉDIA

{Escrita do número de homens}

escreva HOMENS

fim algoritmo.

▲ 1.12.4.

Algoritmo

```

declare COMPRA, {Definição do tipo das variáveis;
                  VENDA, {Preço de compra de cada mercadoria;
                  LUCRO, {Preço de venda de cada mercadoria;
                  LUCRO10, {Porcentagem de lucro de cada mercadoria;
                  LUCRO1020, {Nº de mercadorias com lucro < 10%;
                  LUCRO20, {Nº de mercadorias com 10 ≤ lucro ≤ 20%;
                  TOTALCOMPRA, {Nº de mercadorias com lucro > 20%;
                  TOTALVENDA, {Valor total de compras;
                  LUCROTOTAL, {Valor total de vendas;
                  numérico {Lucro total;

declare NOME {Nome de cada mercadoria;
          literal

          {Atribuição dos valores iniciais necessários}

LUCRO10 ← 0
LUCRO1020 ← 0
LUCRO20 ← 0
TOTALCOMPRA ← 0
TOTALVENDA ← 0

repita
    {Leitura}
    leia NOME, COMPRA, VENDA
    {Verificação de fim de dados para leitura}
    se NOME = "VAZIO"
        então interrompa
    fim se
    {Cálculo da porcentagem de lucro}
    LUCRO ← (VENDA - COMPRA) × 100 / COMPRA
    {Cálculo do nº de mercadorias em cada intervalo de lucro}
    se LUCRO < 10
        então LUCRO10 ← LUCRO10 + 1
    senão se LUCRO ≤ 20
        então LUCRO1020 ← LUCRO1020 + 1
    senão LUCRO20 ← LUCRO20 + 1
    fim se
fim se
    {Cálculo dos valores totais de compra e de venda}
    TOTALCOMPRA ← TOTALCOMPRA + COMPRA
    TOTALVENDA ← TOTALVENDA + VENDA
fim repita

```

{Cálculo do lucro total}
 $LUCROTOTAL \leftarrow TOTALVENDA - TOTALCOMPRA$
 {Escrita dos valores calculados}
escreva LUCRO10
escreva LUCRO1020
escreva LUCRO20
escreva TOTALCOMPRA, TOTALVENDA, LUCROTOTAL
fim algoritmo.

▲ 1.12.6.

Algoritmo

{Definição do tipo das variáveis}
declare INICIAL, {Massa inicial em gramas do material}
 FINAL, {Massa final em gramas do material}
 TEMPO, {Tempo necessário para massa final ficar < 0,5g}
 H, {Horas}
 MIN, {Minutos}
 SEG {Segundos}
numérico
 {Atribuição dos valores iniciais necessários}
 $TEMPO \leftarrow 0$
 {Leitura da massa inicial do material radioativo}
leia INICIAL
 $FINAL \leftarrow INICIAL$
 {Cálculo do tempo necessário para que FINAL fique < 0,5g}
repita
 se FINAL < 0,5
 então interrompa
 fim se
 $FINAL \leftarrow FINAL/2$
 $TEMPO \leftarrow TEMPO + 50$
fim repita
 se TEMPO = 0
 então escreva INICIAL, "MASSA INICIAL MENOR QUE 0,5G"
 senão H $\leftarrow QUOCIENTE(TEMPO,3600)$
 SEG $\leftarrow RESTO(TEMPO,3600)$
 MIN $\leftarrow QUOCIENTE(SEG,60)$
 SEG $\leftarrow RESTO(SEG,60)$
escreva INICIAL, FINAL, H, MIN, SEG
 fim se
fim algoritmo.

▲ 1.12.7.

Algoritmo

{Definição do tipo das variáveis}
declare CONTURMA, {Número de turmas}
 ALUNOS, {Número de alunos de cada turma}
 CONTALUNOS, {Contador de alunos}
 MATRÍCULA, {Número de matrícula de cada aluno}

AUSÊNCIA, {Número de alunos ausentes em cada turma}
 PERCAUS, {Porcentagem de alunos ausentes}
 PERC5, {Turmas com mais de 5% de alunos ausentes}
declare TURMA, {Identificação da turma}
 FREQUÊNCIA {Frequência de cada aluno: A ou P}
literal
 {Atribuição dos valores iniciais necessários}
 $PERC5 \leftarrow 0$
 $CONTURMA \leftarrow 1$
repita
 se CONTURMA > 14
 então interrompa
 fim se
 {Leitura da identificação da turma e do nº de seus alunos}
leia TURMA, ALUNOS
 {Atribuição dos valores iniciais necessários}
 $CONTALUNOS \leftarrow 0$
 $AUSÊNCIA \leftarrow 0$
repita
 se CONTALUNOS > ALUNOS
 então interrompa
 fim se
 {Leitura dos dados de cada aluno}
leia MATRÍCULA, FREQUÊNCIA
 se FREQUÊNCIA = "A"
 então $AUSÊNCIA \leftarrow AUSÊNCIA + 1$
 fim se
 $CONTALUNOS \leftarrow CONTALUNOS + 1$
 fim repita
 $PERCAUS \leftarrow AUSÊNCIA \times 100/ALUNOS$
 se PERCAUS > 5
 então $PERC5 \leftarrow PERC5 + 1$
 fim se
escreva TURMA, PERCAUS
 $CONTURMA \leftarrow CONTURMA + 1$
 fim repita
escreva PERC5
fim algoritmo.

▲ 1.12.9.

Algoritmo

{Definição do tipo das variáveis}
declare NASCIMENTOS, {Número de crianças nascidas no período}
 MESES, {Número de meses de vida}
 MORTAS, {Número de crianças mortas}
 MASCULINOS, {Número de meninos mortos}
 VIDA24, {Nº de crianças que viveram até 24 meses}

PERCMORTAS, {Porcentagem de crianças mortas}
PERCMASC, {Porcentagem de meninos mortos}
PERC24 {Porcentagem de crianças que viveram até 24 meses}
numérico

declare SEXO {Sexo das crianças}
literal
 {Atribuição dos valores iniciais necessários}

MORTAS \leftarrow 0
 MASCULINOS \leftarrow 0
 VIDA24 \leftarrow 0

{Leitura do número de crianças nascidas no período}

leia NASCIMENTOS

repita

{Leitura do sexo e dos meses de vida de cada criança}

leia SEXO, MESES
se SEXO = "VAZIO"
então interrompa
fim se
 MORTAS \leftarrow MORTAS + 1
se SEXO = "MASCULINO"
então MASCULINOS \leftarrow MASCULINOS + 1
fim se
se MESES \leq 24
então VIDA24 \leftarrow VIDA24 + 1
fim se

fim repita
 PERCMORTAS \leftarrow MORTAS \times 100/NASCIMENTOS
 PERCMASC \leftarrow MASCULINOS \times 100/MORTAS
 PERC24 \leftarrow VIDA24 \times 100/MORTAS
escreva PERCMORTAS, PERCMASC, PERC24
fim algoritmo.

▲ 1.12.12.

Algoritmo

declare ALUNOS, {Definição do tipo das variáveis}
MATRÍCULA, {Número de alunos}
NOTA1, {Número da matrícula de cada aluno}
NOTA2, {Primeira nota de cada aluno}
NOTA3, {Segunda nota de cada aluno}
FREQUÊNCIA, {Terceira nota de cada aluno}
NOTAFINAL, {Nº de aulas frequentadas por aluno}
MAIOR, {Nota final de cada aluno}
MENOR, {Maior nota final da turma}
TOTAL, {Menor nota final da turma}
MÉDIA, {Soma das notas finais da turma}
REPROVADOS, {Média das notas finais da turma}
INFREQUENTES, {Total de alunos reprovados}
ALUNOS, {Total de alunos infrequentes}

PERC {Porcentagem de alunos infrequentes}
numérico
declare CÓDIGO {Código: aprovado ou reprovado}
literal

{Atribuição dos valores iniciais necessários;}

REPROVADOS \leftarrow 0

INFREQUENTES \leftarrow 0

TOTAL \leftarrow 0

MAIOR \leftarrow 0

MENOR \leftarrow 100

ALUNOS \leftarrow 1

repita

leia MATRÍCULA, NOTA1, NOTA2, NOTA3, FREQUÊNCIA

NOTAFINAL \leftarrow (NOTA1 + NOTA2 + NOTA3)/3

se (NOTAFINAL < 60) ou (FREQUÊNCIA < 40)

então CÓDIGO \leftarrow "REPROVADO"

REPROVADOS \leftarrow REPROVADOS + 1

se FREQUÊNCIA < 40

então INFREQUENTES \leftarrow INFREQUENTES + 1

fim se

senão CÓDIGO \leftarrow "APROVADO"

fim se

escreva MATRÍCULA, FREQUÊNCIA, NOTAFINAL, CÓDIGO

TOTAL \leftarrow TOTAL + NOTAFINAL

{Leitura das notas e frequência dos outros alunos;}

TOTAL \leftarrow TOTAL + NOTAFINAL

se NOTAFINAL > MAIOR

então MAIOR \leftarrow NOTAFINAL

fim se

se NOTAFINAL < MENOR

então MENOR \leftarrow NOTAFINAL

fim se

ALUNOS \leftarrow ALUNOS + 1

se ALUNOS > 100

então interrompa

fim se

fim repita

MÉDIA \leftarrow TOTAL/100

PERC \leftarrow INFREQUENTES \times 100/100

escreva MAIOR, MENOR, MÉDIA, REPROVADOS, PERC

fim algoritmo.

▲ 1.12.14.

Algoritmo

declare MARCO1, {Definição do tipo das variáveis}
MARCO2, {Marco quilométrico da 1^a cidade;
E, {Marco quilométrico da 2^a cidade;
Distância entre as duas cidades;

V, {Velocidade}
T, {Tempo decorrido entre as cidades};
numérico

repita
 {Leitura dos marcos quilométricos de duas cidades}
 leia MARCO1, MARCO2
 se (MARCO1 = 0) e (MARCO2 = 0)
 então interrompa
 fim se
 E \leftarrow MARCO2 - MARCO1
 V \leftarrow 20
 repita
 T \leftarrow E/V
 se T > 2
 então escreva MARCO1, MARCO2, V, T
 fim se
 V \leftarrow V + 10
 se V > 80
 então interrompa
 fim se
 fim repita
 fim algoritmo.

▲ 1.12.16.

Algoritmo

declare INSCRIÇÃO, {Definição do tipo das variáveis}
 IDADE, {Número de inscrição do candidato}
 FEMININOS, {Idade do candidato}
 MASCULINOS, {Número de candidatos do sexo feminino}
 TIDADE, {Número de candidatos do sexo masculino}
 IDADEMED, {Total de idades de homens c/experiência}
 THOMENS45, {Idade média dos homens c/experiência}
 THOMEXP, {Total de homens com mais de 45 anos}
 PERC45, {Total de homens c/experiência}
 MULHEXP3535, {Percentual de homens c/mais de 45 anos}
 MULHEXP35, {Total de mulheres c/experiência e idade < 35 anos}
 MENOR, {Total de mulheres c/experiência}
 numérico
declare SEXO, {Sexo do candidato}
 EXPERIÊNCIA {Experiência no serviço: sim ou não}
 literal
 {Atribuição dos valores iniciais necessários}
 FEMININOS \leftarrow 0
 MASCULINOS \leftarrow 0
 THOMEXP \leftarrow 0
 TIDADE \leftarrow 0
 THOMENS45 \leftarrow 0

MULHEXP35 \leftarrow 0
MENOR \leftarrow 150
repita

{Leitura dos dados de cada candidato}

leia INSCRIÇÃO, IDADE, SEXO, EXPERIÊNCIA
se INSCRIÇÃO = 0
 então interrompa
fim se
se SEXO = "FEMININO"
 então FEMININOS \leftarrow FEMININOS + 1
 se EXPERIÊNCIA = "SIM"
 então se IDADE < MENOR
 então MENOR \leftarrow IDADE
 fim se
 se IDADE < 35
 então escreva INSCRIÇÃO
 MULHEXP35 \leftarrow MULHEXP35 + 1
 fim se
 fim se
 senão MASCULINOS \leftarrow MASCULINOS + 1
 se EXPERIÊNCIA = "SIM"
 então THOMEXP \leftarrow THOMEXP + 1
 TIDADE \leftarrow TIDADE + IDADE
 fim se
 se IDADE > 45
 então THOMENS45 \leftarrow THOMENS45 + 1
 fim se
 fim se
 fim repita
 IDADEMED \leftarrow TIDADE/THOMEXP
 PERC45 \leftarrow THOMENS45 \times 100/MASCULINOS
escreva FEMININOS, MASCULINOS, IDADEMED, PERC45, MULHEXP35,
 MENOR
fim algoritmo.

▲ 1.12.17.

Algoritmo

{Definição do tipo das variáveis}
declare PREÇO, {Preço do ingresso}
 INGRESSOS, {Número provável de ingressos vendidos}
 DESPESAS, {Despesas com o espetáculo}
 LUCRO, {Lucro provável}
 LUCROMAX, {Lucro máximo provável}
 PREÇOMAX, {Preço máximo}
 INGRESSOSMAX {Número máximo de ingressos vendidos}
 numérico
 {Atribuição dos valores iniciais necessários}
 LUCROMAX \leftarrow 0

```

PREÇO ← 5
INGRESSOS ← 120
DESPESAS ← 200
repita
    se PREÇO < 1
        então interrompa
    fim se
    LUCRO ← INGRESSOS × PREÇO - DESPESAS
    escreva LUCRO, PREÇO
    se LUCRO > LUCROMAX
        então LUCROMAX ← LUCRO
        PREÇOMAX ← PREÇO
        INGRESSOSMAX ← INGRESSOS
    fim se
    INGRESSOS ← INGRESSOS + 26
    PREÇO ← PREÇO - 0,5
fim repita
escreva LUCROMAX, PREÇOMAX, INGRESSOSMAX
fim algoritmo.

```

▲ 1.12.20.

Algoritmo

```

{Definição do tipo das variáveis}
declare DATAS, {Nº de datas pesquisadas}
    DIA, {Dia da semana a ser determinado}
    MES, {Mês lido}
    ANO, {Ano lido}
    M, {Número do mês}
    D, {Dia do mês}
    A, {Dois últimos algarismos do ano}
    S {Dois primeiros algarismos do ano}
    numérico
{Atribuição dos valores iniciais necessários}
DATAS ← 1
repita
    se DATAS > 50
        então interrompa
    fim se
    leia D, MÊS, ANO
    S ← QUOCIENTE(ANO,100)
    A ← RESTO(ANO,100)
    se MÊS ≤ 2
        então M ← MÊS + 10
        A ← A - 1
    senão M ← MÊS - 2
fim se
{Leitura de uma data}

```

```

DIA ← RESTO((TRUNCA(2,0 × M - 0,1) + D + A + QUOCIENTE(A,4) -
    QUOCIENTE(S,4) - 2 × S),7)
se DIA < 0
    então DIA ← DIA + 7
fim se
escreva D, MÊS, ANO, DIA
DATAS ← DATAS + 1
fim repita
fim algoritmo.

```

▲ 1.12.21.

Algoritmo

```

{Definição do tipo das variáveis;
declare NÚMERO, {Número de cada operário}
    PEÇAS, {Número de peças produzidas;
    SALÁRIO, {Salário de cada operário}
    SALMÍNIMO, {Salário mínimo}
    TFOLHA, {Total da folha mensal}
    TPEÇAS, {Total de peças produzidas no mês}
    HOMENSA, {Total de homens - classe A}
    HPEÇASA, {Total de peças produzidas pelos homens - classe A}
    MÉDIAHA, {Média de peças produzidas pelos homens - classe A}
    HOMENSB, {Total de homens - classe B}
    HPEÇASB, {Total de peças produzidas pelos homens - classe B}
    MÉDIAHB, {Média de peças produzidas pelos homens - classe B}
    HOMENSC, {Total de homens - classe C}
    HPEÇASC, {Total de peças produzidas pelos homens - classe C}
    MÉDIAHC, {Média de peças produzidas pelos homens - classe C}
    MULHERESA, {Total de mulheres - classe A}
    MPEÇASA, {Total de peças produzidas pelas mulheres - classe A}
    MÉDIAMA, {Média de peças produzidas pelas mulheres - classe A}
    MULHERESB, {Total de mulheres - classe B}
    MPEÇASB, {Total de peças produzidas pelas mulheres - classe B}
    MÉDIAMB, {Média de peças produzidas pelas mulheres - classe B}
    MULHERESC, {Total de mulheres - classe C}
    MPEÇASC, {Total de peças produzidas pelas mulheres - classe C}
    MÉDIAMC, {Média de peças produzidas pelas mulheres - classe C}
    MAIOR, {Maior salário}
    MAIORNUM {Número do operário com maior salário;
    numérico
declare SEXO {Sexo do operário: masculino ou feminino}
literal
{Atribuição dos valores iniciais necessários;
TFOLHA ← 0
TPEÇAS ← 0
HOMENSA ← 0
HOMENSB ← 0
HOMENSC ← 0

```

```

HPEÇASB ← 0
HPEÇASC ← 0
MULHERESA ← 0
MULHERESB ← 0
MULHERESC ← 0
MPEÇASA ← 0
MPEÇASB ← 0
MPEÇASC ← 0
MAIOR ← 0
SALMÍNIMO ← 120
repita

```

{Leitura dos dados de cada operário}

```

leia NÚMERO, PEÇAS, SEXO
se NÚMERO = 0
  então interrompa
fim se
se PEÇAS ≤ 30
  então SALÁRIO ← SALMÍNIMO
    se SEXO = "MASCULINO"
      então HOMENSA ← HOMENSA + 1
        HPEÇASA ← HPEÇASA + PEÇAS
      senão MULHERESA ← MULHERESA + 1
        MPEÇASA ← MPEÇASA + PEÇAS
    fim se
  senão se PEÇAS ≤ 35
    então SALÁRIO ← SALMÍNIMO + 0,03 × SALMÍNIMO
      × (PEÇAS - 30)
    se SEXO = "MASCULINO"
      então HOMENSB ← HOMENSB + 1
        HPEÇASB ← HPEÇASB + PEÇAS
      senão MULHERESB ← MULHERESB + 1
        MPEÇASB ← MPEÇASB + PEÇAS
    fim se
    senão SALÁRIO ← SALMÍNIMO + 0,05 × SALMÍNIMO
      × (PEÇAS - 30)
    se SEXO = "MASCULINO"
      então HOMENSC ← HOMENSC + 1
        HPEÇASC ← HPEÇASC + PEÇAS
      senão MULHERESC ← MULHERESC + 1
        MPEÇASC ← MPEÇASC + PEÇAS
    fim se
  fim se
escreva NÚMERO, SALÁRIO
TFOLHA ← TFOLHA + SALÁRIO
TPEÇAS ← TPEÇAS + PEÇAS

```

```

se SALARIO > MAIOR
  então MAIOR ← SALARIO
  MAIORNUM ← NÚMERO
fim se
fim repita
MÉDIAHA ← HPEÇASA/HOMENSA
MÉDIAHB ← HPEÇASB/HOMENSB
MÉDIAHC ← HPEÇASC/HOMENSC
MÉDIAMA ← MPEÇASA/MULHERESA
MÉDIAMB ← MPEÇASB/MULHERESB
MÉDIAMC ← MPEÇASC/MULHERESC
escreva TFOLHA
escreva TPEÇAS
escreva MÉDIAHA, MÉDIAHB, MÉDIAHC
escreva MÉDIAMA, MÉDIAMB, MÉDIAMC
escreva MAIORNUM
fim algoritmo.

```

▲ 1.12.23.

Algoritmo

{Definição do tipo das variáveis}

```

declare DIA1, {Dia da primeira data}
  MÊS1, {Mês da primeira data}
  ANO1, {Ano da primeira data}
  DIA2, {Dia da segunda data}
  MÊS2, {Mês da segunda data}
  ANO2, {Ano da segunda data}
  DIA, {Dia de uma data}
  MÊS, {Mês de uma data}
  ANO, {Ano de uma data}
  D, {Número de dias contidos em uma data}
  DIAS, {Nº de dias decorridos entre duas datas}
  RESÍDUO, {Resíduo dos anos bissextos}
  M, {Contador}
  DATA, {Contador}
  I {Variável de controle}
  numérico

```

repita

{Leitura das datas}

```

leia DIA1, MÊS1, ANO1, DIA2, MÊS2, ANO2

```

```

se DIA1 < 0

```

```

  então interrompa

```

```

fim se

```

{Cálculo do número de dias contidos em cada uma das datas}

```

DIA ← DIA1

```

```

MÊS ← MÊS1

```

```

ANO ← ANO1

```

```

DATA ← 1

```

```

repita
  se DATA > 2
    então interrompa
  fim se
  {Cálculo do nº de anos bissextos ocorridos entre os anos 1 e}
  {o anterior à data pesquisada}
  RESÍDUO ← 0
  I ← 4
  repita
    se I > ANO - 1
      então interrompa
    fim se
    se (RESTO(I,400) = 0) ou (RESTO(I,100) ≠ 0)
      então RESÍDUO ← RESÍDUO + 1
    fim se
    I ← I + 4
  fim repita
  {Transformação dos anos anteriores ao da data em dias}
  D ← (ANO - 1) * 365 + RESÍDUO + DIA
  {Transformação dos meses anteriores ao da data em dias}
  se MÊS ≠ 1
    então D ← D + 31 {Dias de janeiro}
    M ← 2
    repita
      se M > MÊS - 1
        então interrompa
      fim se
      se M = 2 {Dias de fevereiro}
        então se ((RESTO(ANO,4) = 0) e (RESTO(ANO,100) ≠ 0))
          ou (RESTO(ANO,400))
          então D ← D + 29
        senão D ← D + 28
      fim se
      senão se (M = 4) ou (M = 6) ou (M = 9) ou (M = 11)
        então D ← D + 30
      senão D ← D + 31
      fim se
    fim se
    M ← M + 1
  fim repita
fim se
se DATA = 1
  então DIAS1 ← D
  DIA ← DIA2
  MÊS ← MÊS2
  ANO ← ANO2

```

```

  | senão DIAS2 ← D
  | fim se
  | DATA ← DATA + 1
  | fim repita
  | DIAS ← DIAS2 - DIAS1
  | escreva DIA1, MÊS1, ANO1, DIA2, MÊS2, ANO2, DIAS
  | fim repita
  | fim algoritmo.

```

▲ 1.12.26.

Algoritmo {Definição do tipo das variáveis}

```

declare DENOMINADOR, {Valor do denominador}
  SOMA {Valor do somatório}
  numérico {Atribuição dos valores iniciais necessários}
  SOMA ← 0
  DENOMINADOR ← 1
  repita
    SOMA ← SOMA + ((38 - DENOMINADOR) × (39 - DENOMINADOR)) /
      DENOMINADOR
    DENOMINADOR ← DENOMINADOR + 1
    se DENOMINADOR > 37
      então interrompa
    fim se
  fim repita
  escreva SOMA {Escrita da soma}
fim algoritmo.

```

▲ 1.12.29.

Algoritmo {Definição do tipo das variáveis}

```

declare NUMERADOR, {Valor do numerador}
  DENOMINADOR, {Valor do denominador}
  QUANT, {Quantidade de termos}
  SINAL, {Sinal da parcela}
  SOMA {Valor do somatório}
  numérico {Atribuição dos valores iniciais necessários}
  NUMERADOR ← 480
  DENOMINADOR ← 10
  SOMA ← 0
  SINAL ← 1
  QUANT ← 1
  repita
    SOMA ← SOMA + (NUMERADOR / DENOMINADOR) × SINAL
    NUMERADOR ← NUMERADOR - 5
    DENOMINADOR ← DENOMINADOR + 1

```

```

SINAL ← SINAL × (-1)
QUANT ← QUANT + 1
se QUANT > 30
  então interrompa
fim se
fim repita

```

{Escrita da soma}

```

escreva SOMA
fim algoritmo.

```

▲ 1.12.31.

Algoritmo

{Definição do tipo das variáveis}

```

declare DENOMINADOR,  [Valor do denominador]
  PARCELA,            [Parcela do somatório]
  PI,                 [Valor do somatório para calculo da PI]
  SINAL,              [Sinal da parcela]
  numérico

```

{Atribuição dos valores iniciais necessários}

```

DENOMINADOR ← 1
PI ← 0
SINAL ← 1
PARCELA ← 4
repita
  se PARCELA < 0,0001
    então interrompa
  fim se
  PI ← PI + PARCELA × SINAL
  DENOMINADOR ← DENOMINADOR + 2
  SINAL ← SINAL × (-1)
  PARCELA ← 4 / DENOMINADOR
fim repita

```

{Escrita do somatório de PI}

```

escreva PI
fim algoritmo.

```

▲ 1.12.36.

Algoritmo

{Definição do tipo das variáveis}

```

declare NUMERADOR,      [Valor do numerador]
  PARCELA,             [Parcela do somatório]
  PARCELAANTERIOR,    [Parcela anterior do somatório]
  DENOMINADOR,         [Valor do denominador]
  SOMA,                [Valor do somatório]
  QUANT,               [Quantidade de termos]
  numérico

```

{Atribuição dos valores iniciais necessários}

```

NUMERADOR ← 61
DENOMINADOR ← 1

```

```

SOMA ← 63
PARCELA ← 61
PARCELAANTERIOR ← 63
QUANT ← 1
repita
  se | PARCELA - PARCELAANTERIOR | < 0,0000001
    então interrompa
  fim se
  SOMA ← SOMA + PARCELA
  QUANT ← QUANT + 1
  NUMERADOR ← NUMERADOR - 2
  DENOMINADOR ← DENOMINADOR × QUANT
  PARCELAANTERIOR ← PARCELA
  PARCELA ← NUMERADOR / DENOMINADOR
fim repita

```

{Escrita da soma}

```

escreva SOMA
fim algoritmo.

```

▲ 1.12.37.

Algoritmo

{Definição do tipo das variáveis}

```

declare NUMERADOR,      [Valor do numerador]
  DENOMINADOR,         [Valor do denominador]
  SOMA,                [Valor do somatório]
  SINAL,               [Sinal da parcela]
  QUANT,               [Quantidade de parcelas]
  numérico

```

{Atribuição dos valores iniciais necessários}

```

NUMERADOR ← 1
DENOMINADOR ← 1
SOMA ← 1
SINAL ← 1
QUANT ← 1
repita

```

```

  NUMERADOR ← NUMERADOR × QUANT
  DENOMINADOR ← DENOMINADOR × 2 - 1
  SINAL ← SINAL × (-1)
  SOMA ← SOMA + NUMERADOR / DENOMINADOR × SINAL
  QUANT ← QUANT + 1
  se QUANT > 50
    então interrompa
  fim se
fim repita

```

{Escrita do somatório }

```

escreva SOMA
fim algoritmo.

```

Algoritmo

{Definição do tipo das variáveis;}

declare X,

NUMERADOR,	{Valor do numerador}
DENOMINADOR,	{Valor do denominador}
SOMA,	{Valor do somatório}
SINAL,	{Sinal da parcela}
QUANT	{Quantidade de parcelas}
numérico	

{Leitura dos dados }

leia X

{Atribuição dos valores iniciais necessários;

NUMERADOR $\leftarrow 1$

DENOMINADOR $\leftarrow 1$

SOMA $\leftarrow X$

SINAL $\leftarrow 1$

QUANT $\leftarrow 2$

repita

SINAL $\leftarrow SINAL \times (-1)$

NUMERADOR $\leftarrow NUMERADOR \times X \times X$

DENOMINADOR $\leftarrow DENOMINADOR \times (2 \times QUANT - 2) \times (2 \times QUANT - 1)$

SOMA $\leftarrow SOMA + NUMERADOR / DENOMINADOR \times SINAL$

QUANT $\leftarrow QUANT + 1$

se QUANT > 20

então interrompa

fim se

fim repita

{Escrita do somatório }

escreva SOMA

fim algoritmo.

▲ 1.12.45.

Algoritmo

{Definição do tipo das variáveis;}

declare X, {Valor do incremento}

Y, H, F, G {Valor das funções}

numérico

X $\leftarrow 1$

repita

H $\leftarrow X^2 - 16$

se H ≥ 0

então F $\leftarrow H$

senão F $\leftarrow 1$

fim se

se F > 0

então G $\leftarrow 0$

senão se r = 0

então F $\leftarrow X^2 + 16$

fim se

Y $\leftarrow F + G$

escreva X, Y

X $\leftarrow X + 1$

se X > 10

então interrompa

fim se

fim repita

fim algoritmo.

▲ 1.12.50.

Algoritmo

{Definição do tipo das variáveis;}

declare LADOS, {Número de lados do polígono}

SEMIPERÍMETRO

numérico

LADOS $\leftarrow 5$

repita

{Cálculo do semi-perímetro}

SEMIPERÍMETRO $\leftarrow LADOS \times \text{SEN}(3,1416 / LADOS)$

escreva LADOS, SEMIPERÍMETRO

LADOS $\leftarrow LADOS + 5$

se LADOS > 100

então interrompa

fim se

fim repita

fim algoritmo.

▲ 1.12.54.

Algoritmo

{Definição do tipo das variáveis;}

declare Y, {Número lido}

RAIZ, {Valor da raiz quadrada}

I {Contador de aproximações }

numérico

{Leitura dos dados }

leia (Y)

RAIZ $\leftarrow Y / 2$

I $\leftarrow 2$

repita

RAIZ $\leftarrow (RAIZ^2 - Y) / (2 \times RAIZ)$

I $\leftarrow I + 1$

se I > 20

então interrompa

fim se

fim repita

{Escrita da raiz }

escreva Y, RAIZ
fim algoritmo.

▲ 1.12.57

Algoritmo

declare NÚMERO,
DIA,
MÊS,
NOTAPROVISÓRIA,
NOTAFINAL,
SOMAPROVISÓRIA,
SOMAPROVISÓRIA2,
SOMAFINAL,
SOMAFINAL2,
MÉDIAPROVISÓRIA,
MÉDIAFINAL,
DESVIOPROVISÓRIO,
DESVIOFINAL,
N,
I
numérico
 {Definição do tipo das variáveis}

N → 10
SOMAPROVISÓRIA → 0
SOMAPROVISÓRIA2 → 0
SOMAFINAL → 0
SOMAFINAL2 → 0
 {Atribuição dos valores iniciais necessários}

I → 1
repita
 se MÊS < 4 ou (DIA ≤ 20 e MÊS = 4)
 então NOTAFINAL ← NOTAPROVISÓRIA + 10
 senão se MÊS < 5 ou (DIA ≤ 2 e MÊS = 5)
 então NOTAFINAL ← NOTAPROVISÓRIA
 senão se DIA ≤ 30 e MÊS = 5
 então NOTAFINAL ← NOTAPROVISÓRIA / 2
 senão NOTAFINAL ← 0
 fim se
fim se
 {Leitura dos dados}

escreva NÚMERO, NOTAFINAL
 {Escrita da nota dos alunos }

escreva NÚMERO, NOTAFINAL
 {Acúmulo das notas}

SOMAPROVISÓRIA ← SOMAPROVISÓRIA + NOTAPROVISÓRIA
SOMAFINAL ← SOMAFINAL + NOTAFINAL

SOMAPROVISÓRIA2 ← SOMAPROVISÓRIA2 + NOTAPROVISÓRIA2
SOMAFINAL2 ← SOMAFINAL2 + NOTAFINAL2

I ← I + 1

se I > N

 então interrompa
fim se

fim repita

MÉDIAPROVISÓRIA ← SOMAPROVISÓRIA / N

MÉDIAFINAL ← SOMAFINAL / N

DESVIOPROVISÓRIO ← $\sqrt{\frac{1}{(N-1)} \times (SOMAPROVISÓRIA2 - 1/N \times SOMAPROVISÓRIA^2)}$

DESVIOFINAL ← $\sqrt{\frac{1}{(N-1)} \times (SOMAFINAL2 - 1/N \times SOMAFINAL^2)}$
escreva MÉDIAPROVISÓRIA, MÉDIAFINAL, DESVIOPROVISÓRIO,
DESVIOFINAL

fim algoritmo.

▲ 1.12.58.

Algoritmo

declare R1, R2,
T1, T2,
PR,
PI
numérico
declare OPERAÇÃO {Operação a ser realizada}
literal
 {Definição do tipo das variáveis;
 {Módulos dos números complexos};
 {Argumentos dos números complexos};
 {Parte real};
 {Parte imaginária}}

leia OPERAÇÃO, R1, R2, T1, T2
repita
 se OPERAÇÃO = "VAZIO"
 então interrompa
 fim se

 se OPERAÇÃO = "MULTIPLICA"
 então {multiplicação dos números};
 PR ← R1 x R2
 PI ← T1 + T2
 senão {divisão dos números};
 PR ← R1 / R2
 PI ← T1 - T2
 fim se
 escreva R1, R2, T1, T2, PR, "EXP(", PI, ")"
 leia OPERAÇÃO, R1, R2, T1, T2
fim repita

fim algoritmo.

▲ 1.12.59.

Algoritmo

declare A, B,
 {Definição do tipo das variáveis;
 {Limites de integração};

N, {Número de intervalos}
 H, {Largura dos intervalos}
 X1, X2, Y1, Y2, {Abscissas e ordenadas da função}
 I, {Contador de intervalos}
 INT {Valor da Integral}
numérico
leia A, B, N
 $H \leftarrow (B - A)/N$
 INT $\leftarrow 0$
 X1 $\leftarrow A$
 I $\leftarrow 1$
repita
 $X2 \leftarrow X1 + H$
 $Y1 \leftarrow 1 / (1 + X1^2)$
 $Y2 \leftarrow 1 / (1 + X2^2)$
 $INT \leftarrow INT + Y1 + Y2$
 $X1 \leftarrow X2$
 $I \leftarrow I + 1$
se I = N
 \quad então interrompa
fim se
fim repita
 $INT \leftarrow INT \times 4 \times H / 2$
escreva INT
fim algoritmo.

▲ 1.12.60.

Algoritmo

declare LINHA, {Definição do tipo das variáveis}
 A, B, C, {Variável de controle das linhas a serem lidas}
 MDC, {Variáveis usadas no cálculo do M.D.C.}
 RESTO, {Máximo Divisor Comum}
 AUX, {Resto da divisão}
numérico
declare JACALCULOU {Variável de controle do cálculo do MDC entre A e B, e C}
lógico
 LINHA $\leftarrow 1$
 {Processamento de 25 linhas contendo valores de A, B e C}
repita
se LINHA ≤ 25
 \quad então interrompa
fim se
leia A, B, C
escreva A, B, C
 JACALCULOU := verdadeiro

repita
se A < B {Ordena: A > B}
 \quad então AUX $\leftarrow A$
 \quad A $\leftarrow B$
 \quad B \leftarrow AUX
fim se
 RESTO $\leftarrow 1$
 { Cálculo do MDC pelo método das divisões sucessivas }
repita
se RESTO = 0
 \quad então interrompa
fim se
 $RESTO \leftarrow RESTO(A,B)$
se RESTO = 0
 \quad então MDC $\leftarrow B$
senão A $\leftarrow B$
 \quad B \leftarrow RESTO
fim se
fim repita
se JACALCULOU { Inclusão do C:
 \quad então A $\leftarrow MDC$
 \quad B $\leftarrow C$
 \quad JACALCULOU \leftarrow falso
senão JACALCULOU \leftarrow verdadeiro
fim se
se JACALCULOU
 \quad então interrompa
fim se
fim repita
escreva MDC
 LINHA $\leftarrow LINHA + 1$
fim repita
fim algoritmo.

▲ 1.12.64.

Algoritmo

declare NÚMERO, {Definição do tipo das variáveis}
 DIVISOR, {Valor a ser testado}
 SOMA, {Divisores do número}
numérico
declare NÚMERO, {Teste dos números de 1 a 100}
 DIVISOR, {Soma dos divisores}
numérico
 NÚMERO $\leftarrow 1$
repita
se NÚMERO > 100
 \quad então interrompa
fim se

{Atribuição de valores iniciais para teste do próximo número}
LIMITE \leftarrow ARREDONDA($\sqrt{NÚMERO}$)

DIVISOR \leftarrow 2

SOMA \leftarrow 1

{Pesquisa e soma dos divisores do número}

repita

se DIVISOR \geq LIMITE
| então interrompa

fim se

se RESTO(NÚMERO,DIVISOR) = 0

| (Acúmulo do DIVISOR e do quociente, que também é divisor)

SOMA \leftarrow SOMA + DIVISOR + NÚMERO / DIVISOR

fim se

DIVISOR \leftarrow DIVISOR + 1

fim repita

se SOMA = NÚMERO {Verificação se número é perfeito}
| então escreva NÚMERO

fim se

NÚMERO \leftarrow NÚMERO + 1 {Próximo número}

fim repita

fim algoritmo.

▲ 1.12.67.

Algoritmo

{Definição do tipo das variáveis}

declare NÚMERO, {Valor a ser testado}
LIMITE, {Limite do teste do divisor}
DIVISOR {Divisores do número}
numérico

declare EPRIMO {Variável lógica auxiliar}
lógico

NÚMERO \leftarrow 5000

repita

se NÚMERO \leq 7000

| então interrompa

fim se

LIMITE \leftarrow ARREDONDA($\sqrt{NÚMERO}$)

DIVISOR \leftarrow LIMITE

EPRIMO \leftarrow verdadeiro

repita

se RESTO(NÚMERO,DIVISOR) = 0

| EPRIMO \leftarrow falso

fim se

DIVISOR \leftarrow DIVISOR - 1

se DIVISOR = 0 ou não EPRIMO

| então interrompa

fim se

fim repita

{Escrita do número primo}

se EPRIMO

| então escreva NÚMERO

fim se

NÚMERO \leftarrow NÚMERO + 1

fim repita

fim algoritmo.

▲ 1.12.68.

Algoritmo

{Definição do tipo das variáveis}

declare NÚMERO, {Valor a ser convertido}

RESTO {Resto da divisão}

numérico

declare BINÁRIO {Representação em binário}

literal

repita

| leia NÚMERO

escreva NÚMERO

se NÚMERO = 0

| então interrompa

fim se

BINÁRIO \leftarrow "

{Geração da representação binária do valor lido pelo método}

{ das divisões sucessivas por 2}

repita

RESTO \leftarrow RESTO(NÚMERO,2)

{Concatenação do caractere 0, ou 1, à esquerda, dependendo do resto.}

{ Por ex.: "0" | "11" = "011" e "1" | "11" = "111"}

se RESTO = 0

| então BINÁRIO \leftarrow "0" | BINÁRIO

senão BINÁRIO \leftarrow "1" | BINÁRIO

fim se

NÚMERO \leftarrow QUOCIENTE(NÚMERO,2)

se NÚMERO = 0

| então interrompa

fim se

fim repita

escreva BINÁRIO

fim repita

fim algoritmo.

▲ 2.2.1.2.1.

Algoritmo

```

{Declaração de das variáveis utilizadas}
declare NOTAS[0:10] numérico
declare SOMA, MÉDIA, QUANTIDADE, I numérico
SOMA, {Soma das notas}
MÉDIA, {Média das notas}
QUANTIDADE, {Quantidade de notas acima da média}
I, {Variável auxiliar}

I ← 0
SOMA ← 0
repita {Leitura e soma das notas}
  se I = 10
    então interrompa
  fim se
  leia NOTA[I] {Leitura e acumulação das notas}
  SOMA ← SOMA + NOTA[I]
  I ← I + 1
fim repita
MÉDIA ← SOMA/10 {Cálculo da média}
{Contagem dos alunos com nota superior à média}
I ← 0
QUANTIDADE ← 0
repita
  se I = 10
    então interrompa
  fim se
  {Contagem dos alunos com nota inferior à média}
  se NOTA[I] > MÉDIA
    então QUANTIDADE ← QUANTIDADE + 1
  fim se
  I ← I + 1
fim repita
escreva MÉDIA, QUANTIDADE {Escrita dos resultados}
fim algoritmo.

```

▲ 2.2.1.2.3.

Algoritmo

```

{Definição do tipo das variáveis}
declare A, B[1:25], C[1:50] numérico
{Conjuntos lidos}
{Conjunto intercalado}

declare I numérico
{Variável usada como índice}

leia A[1], A[2], ..., A[25]
leia B[1], B[2], ..., B[25]
{Leitura dos conjuntos}

```

escreva A[1], A[2], ..., A[25]escreva B[1], B[2], ..., B[25]

I ← 1

repita

se I = 25

| então interrompa

fim se

{Intercalação dos conjuntos}

C[2 × I - 1] ← A[I]

C[2 × I] ← B[I]

I ← I + 1

fim repita

{Escrita do conjunto intercalado}

escreva C[1], C[2], ..., C[50]fim algoritmo.

▲ 2.2.1.2.5.

Algoritmo

{Definição do tipo das variáveis}

declare A[1:200] numéricodeclare AUX, I numéricoleia A[1], A[2], ..., A[200]escreva A[1], A[2], ..., A[200] {Escrita do conjunto lido}

I ← 1

repita

{Inversão do conjunto}

se I = 101

| então interrompa

fim se

AUX ← A[I]

A[I] ← A[201 - I]

A[201 - I] ← AUX

I ← I + 1

fim repita

{Escrita do conjunto invertido}

escreva A[1], A[2], ..., A[200]fim algoritmo.

▲ 2.2.2.2.1.

Algoritmo

{Definição do tipo das variáveis}

declare X[1:2000, 4] numérico {Estrutura contendo informações}declare MELHORMÉDIA numérico {Melhor média}declare MELHOR, N, I, J numérico {Índice de identificação da melhor média}N, I, J numérico {Variáveis auxiliares}

{Leitura do número de alunos e das demais informações sobre cada aluno}

leia N

leia X[1, 1], X[1,2], ..., X[N,3], X[N,4]

{Identificação da melhor média}

MELHORMÉDIA ← 0

I ← 1

repita

se I > N

então interrompa

fim se

se X[I,3] = 153 e X[I,4] > MELHORMÉDIA

então MELHOR ← I

MELHORMÉDIA ← X[I,4]

fim se

I ← I + 1

fim repita

{Escrita dos resultados}

escreva X[MELHOR,1], X[MELHOR,2], X[MELHOR,3], X[MELHOR,4]

fim algoritmo.

▲ 2.2.2.2.3.

Algoritmo

{Definição do tipo das variáveis}

declare ESTOQUE[1:3,1:3]

CUSTOPRODUTO[1:3]

TABELACUSTO[1:3]

numérico

declare CUSTO,

CUSTOESTOQUE,

TOTALITENS,

ARMAZÉM,

PRODUTO,

MAIORARMAZÉM

numérico

leia ESTOQUE[1,1], ESTOQUE[1,2], ..., ESTOQUE[3,2], ESTOQUE[3,3]

leia TABELACUSTO[1], TABELACUSTO[2], TABELACUSTO[3]

{Determinação e escrita da quantidade de itens por armazém}

ARMAZÉM ← 1

repita

se ARMAZÉM > 3

então interrompa

fim se

PRODUTO ← 1

TOTALITENS ← 0

repita

se PRODUTO > 3

então interrompa

fim se

TOTALITENS ← TOTALITENS + ESTOQUE[ARMAZÉM,PRODUTO]

PRODUTO ← PRODUTO + 1

fim repita

escreva "Armazém", ARMAZÉM, "TOTAL ESTOCADO", TOTALITENS

ARMAZÉM ← ARMAZÉM + 1

fim repita

{Pesquisa do armazém com o maior estoque do produto 2}

ARMAZÉM ← 1

MAIORARMAZÉM ← 1

PRODUTO ← 2

repita

se ARMAZÉM > 3

então interrompa

fim repita

se ESTOQUE[ARMAZÉM,PRODUTO] >

ESTOQUE[MAIORARMAZÉM,PRODUTO]

então MAIORARMAZÉM ← ARMAZÉM

fim se

ARMAZÉM ← ARMAZÉM + 1

fim repita

escreva "O maior estoque do produto 2 está no armazém: ", MAIORARMAZÉM

{Cálculo dos custos}

PRODUTO ← 1

repita

se PRODUTO > 3

então interrompa

fim se

CUSTOPRODUTO[PRODUTO] ← 0

fim repita

ARMAZÉM ← 1

repita

se ARMAZÉM > 3

então interrompa

fim se

CUSTOESTOQUE ← 0

PRODUTO ← 1

repita

se PRODUTO > 3

então interrompa

fim se

CUSTOPRODUTO[PRODUTO] ← CUSTOPRODUTO[PRODUTO] × TABELACUSTO[PRODUTO]

escreva "O custo do produto ", PRODUTO , " do armazém ",

ARMAZÉM, CUSTO

CUSTOPRODUTO[PRODUTO] ← CUSTOPRODUTO[PRODUTO] + CUSTO

CUSTOESTOQUE ← CUSTOESTOQUE + CUSTO

PRODUTO ← PRODUTO + 1

fim repita

escreva "Custo do estoque do armazém ", ARMAZEM , CUSTOESTOQUE
ARMAZÉM ← ARMAZÉM + 1

fim repita

{Escrita do custo total por produto}

PRODUTO ← 1

repita

se PRODUTO > 3

então interrompa

fim se

escreva "Custo total do produto ", PRODUTO , CUSTOPRODUTO[PRODUTO]

PRODUTO ← PRODUTO + 1

fim repita

fim algoritmo.

Exercícios de fixação - Arquivo - Organização Seqüencial

▲ 2.4.6.3.1.

Algoritmo

{Declaração das estruturas de dados}

declare AGENDA {Arquivo onde estão armazenados os dados}

arquivo sequencial de ENDEREÇO

declare ENDEREÇO registro (NOME literal,

TELEFONE numérico,

LOGRADOURO literal,

NÚMERO numérico

CIDADE, ESTADO literal)

{Abertura do arquivo para leitura}

abra AGENDA leitura

{Leitura do arquivo e Escrita dos dados}

repita

leia AGENDA.ENDEREÇO

se AGENDA.FDA

então interrompa

fim se

escreva AGENDA.ENDEREÇO

fim repita

{Fechamento do arquivo}

feche AGENDA

fim algoritmo.

▲ 2.4.6.3.4.

Algoritmo

{Definição do tipo das variáveis}

declare CRÉDITO,

DÉBITO registro (CONTA numérico

OPERAÇÃO literal

VALOR numérico

DATA literal)

declare ARQCREDITO, ARQDEBITO, ARQATUALIZA

arquivo sequencial de DÉBITO, CRÉDITO

declare CONTAFINAL numérico

{Atribuição dos valores iniciais necessários;

CONTAFINAL ← 9999

CRÉDITO.CONTA ← CONTAFINAL

DÉBITO.CONTA ← CONTAFINAL

{Abertura dos arquivos}

abra ARQCRÉDITO leitura

abra ARQDÉBITO leitura

abra ARQATUALIZA escrita

{Leitura dos primeiros registros;

leia ARQCRÉDITO.CRÉDITO

leia ARQDÉBITO.DÉBITO

{Intercalação dos registros de débito e crédito}

repita

se CRÉDITO.CONTA = CONTAFINAL e DÉBITO.CONTA = CONTAFINAL

então interrompa

fim se

se CRÉDITO.CONTA ≤ DÉBITO.CONTA

então escreva ARQATUALIZA.CRÉDITO

se ARQCRÉDITO.FDA

então CRÉDITO.CONTA ← CONTAFINAL

senão leia ARQCRÉDITO.CRÉDITO

fim se

fim se

se DÉBITO.CONTA < CRÉDITO.CONTA

então escreva ARQATUALIZA.DÉBITO

se ARQDÉBITO.FDA

então DÉBITO.CONTA ← CONTAFINAL

senão leia ARQDÉBITO.DÉBITO

fim se

fim se

fim repita

{Fechamento dos arquivos;

feche ARQCRÉDITO, ARQDÉBITO, ARQATUALIZA

fim algoritmo.

Capítulo 2 - Exercícios propostos

▲ 2.5.1.2.

Algoritmo

{Definição do tipo das variáveis}

declare FRASE literal

UMPAR literal

BRANCOS numérico

CONTAA numérico

I, J, K numérico

```

declare PARES[1:20] registro (PARLETRAS literal,
    CONTAPAR numérico)
    {Atribuição de valores iniciais}

K ← 0
CONTAA ← 0
leia FRASE
    {Pesquisa dos brancos, A's e repetições}

I ← 1
repita
    se I > 80
        então interrompa
    fim se
    se FRASE[I] = " "
        então BRANCOS ← BRANCOS + 1
    senão se FRASE[I] = 'A'
        então CONTAA ← CONTAA + 1
    fim se
    se I < 80 e FRASE[I] = FRASE[I+1]
        então UMPAR ← FRASE[I] + FRASE[I+1]
    fim se
    J ← 1
    repita
        se J > K ou UMPAR = PARES.PARLETRAS[J]
            então interrompa
        fim se
        J ← J + 1
    fin repita
    se J > K
        entao K ← K + 1
        PARES.PARLETRAS[K] ← UMPAR
        PARES.CONTAPAR[K] ← 1
    senão PARES.CONTAPAR[K] ← PARES.CONTAPAR[K] + 1
    fim se
    fim se
    I ← I + 1
fin repita
    {Escrita dos resultados}

escreve FRASE
escreve CONTAA
escreve BRANCOS
escreve PARES[1], ..., PARES[K]
fin algoritmo.

```

▲ 2.5.1.4.

Algoritmo

{Definição do tipo das variáveis}

```

declare B[1:100] numérico
declare S, {Soma}

```

```

I {Índice do vetor B}
    numérico
leia B[1], B[2], ..., B[100]
    {Leitura do vetor}
    S ← 0
    {Cálculo da soma}

I ← 1
repita
    se I > 50
        então interrompa
    fim se
    S ← S + (B[I] + B[101 - I])3
    I ← I + 1
fim repita
fin algoritmo.

```

▲ 2.5.1.5.

Algoritmo

{Definição do tipo das variáveis}

```

declare N, NTOT, NOTA, FREQREL numérico
declare FREQABS[0:10] numérico
    {Atribuição dos valores iniciais necessários}

NTOT ← 80
NOTA ← 0

```

```

repita
    se NOTA > 10
        então interrompa
    fim se
    FREQABS[NOTA] ← 0
    NOTA ← NOTA + 1
fin repita
    {Leitura de cada NOTA e sua contagem em FREQABS}

```

```

N ← 0
repita
    se N > NTOT
        então interrompa
    fim se
    leia NOTA
    FREQABS[NOTA] ← FREQABS[NOTA] + 1
    N ← N + 1
fin repita
    {Escrita de cada FREQABS e cada FREQREL}

```

```

NOTA ← 0
repita
    se NOTA > 10
        então interrompa
    fim se
    FREQREL ← FREQABS[NOTA] / NOTA
    escreva NOTA, FREQABS[NOTA], FREQREL

```

```

    NOTA ← NOTA + 1
    fim repita
    fim algoritmo.

```

▲ 2.5.1.6.

Algoritmo

{Definição do tipo das variáveis}

```

declare X, Y [1:250] numérico
declare N, NTOT, M, MTOT numérico
declare NOME literal
declare DISJUNTOS lógico
repita {para cada par de conjuntos}
    {Leitura do nome de cada par de conjuntos}
    escreva "Digite o NOME de um par de conjuntos"
    leia NOME
    se NOME = "VAZIO"
        então interrompa
    fim se
    {Leitura do número de elementos de cada conjunto}
    escreva "Digite o número de elementos de cada conjunto"
    leia NTOT, MTOT
    {Leitura dos elementos de cada conjunto}
    escreva "Digite os ", NTOT, "elementos do 1º conjunto"
    N ← 1
    repita
        se N > NTOT
            então interrompa
        fim se
        leia X[N]
        N ← N + 1
    fim repita
    escreva "Digite os ", MTOT, "elementos do 2º conjunto"
    M ← 1
    repita
        se M > MTOT
            então interrompa
        fim se
        leia Y[M]
        M ← M + 1
    fim repita
    {Verificação se X e Y são disjuntos}
    DISJUNTOS ← verdadeiro
    N ← 1
    repita
        se N > NTOT ou DISJUNTOS = falso
            então interrompa
        fim se
        M ← 1

```

```

repita
    se M > MTOT ou DISJUNTOS = falso
        então interrompa
    fim se
    se X[N] = Y[M]
        então DISJUNTOS ← falso
    fim se
    M ← M + 1
fim repita
N ← N + 1
fim repita
fim repita
{Escrita do nome dos pares disjuntos}
se DISJUNTOS
    então escreva NOME
fim se
fim repita
fim algoritmo.

```

▲ 2.5.1.8.

Algoritmo

{Definição do tipo das variáveis}

```

declare I, ITOT, N, NTOT, AUX numérico
declare EMPREGADO, MESES [1:301] numérico
{Leitura das informações dos empregados}
escreva "Digite o nº de cada EMPREGADO e o nº de MESES"
N ← 1
repita
    leia EMPREGADO[N], MESES[N]
    se EMPREGADO[N] = 0 e MESES[N] = 0
        então interrompa
    fim se
    N ← N + 1
fim repita
NTOT ← N - 1
ITOT ← 3
{Colocação dos empregados mais recentes na frente}
se ITOT > NTOT
    então ITOT ← NTOT
fim se
I ← 1
repita
    se I > ITOT
        então interrompa
    fim se
    N ← I
    repita
        se N > NTOT
            então interrompa
        fim se

```

```

    se MESES[N] < MESES[I]
    então AUX ← MESES[I]
    MESES[I] ← MESES[N]
    MESES[N] ← AUX
    {Trocada de MESES[I] com MESES[N]}
    AUX ← EMPREGADO
    EMPREGADO[I] ← EMPREGADO[N]
    EMPREGADO[N] ← AUX

    fim se
fim repita
fim repita

{Escrita do nº dos empregados mais recentes}
escreva "EMPREGADOS MAIS RECENTES"
I ← 1
repita
    se I > ITOT
    então interrompa
fim se
escreva EMPREGADO[I]
fim repita
fim algoritmo.

```

▲ 2.5.1.10.

Algoritmo

```

{Definição do tipo das variáveis}
declare I, IA, IB, IC, NA, NB numérico
declare A, B [1:100] numérico
declare C [1:200] numérico

{Leitura do nº de elementos do conjunto A, seguido de seus elementos}
leia NA
IA ← 1
repita
    se IA > NA
    então interrompa
fim se
leia A[IA]
IA ← IA + 1
fim repita

{Leitura do nº de elementos do conjunto B, seguido de seus elementos}
leia NB
IB ← 1
repita
    se IB > NB
    então interrompa
fim se
leia B[IB]
IB ← IB + 1
fim repita

```

{Trocada de MESES[I] com MESES[N]}

```

{Intercalação dos conjuntos A com B obtendo o conjunto C}
IA ← 1
IB ← 1
IC ← 0
{Transferência de valores enquanto houver elementos em A e em B}
repita
    se IA > NA ou IB > NB
    então interrompa
fim se
    IC ← IC + 1
    se A[IA] < B[IB]
    então C[IC] ← A[IA]
        IA ← IA + 1
    senão C[IC] ← B[IB]
        IB ← IB + 1
    fim se
fim repita

{Transferência do menor dentre A e B para C}
IC ← IC + 1
se A[IA] < B[IB]
então C[IC] ← A[IA]
IA ← IA + 1
senão C[IC] ← B[IB]
IB ← IB + 1
fim se
fim repita

I ← IA
{Transferência dos elementos restantes de A}
repita
    se I > NA
    então interrompa
fim se
    IC ← IC + 1
    C[IC] ← A[I]
    I ← I + 1
fim repita

I ← IB
{Transferência dos elementos restantes de B}
repita
    se I > NB
    então interrompa
fim se
    IC ← IC + 1
    C[IC] ← B[I]
    I ← I + 1
fim repita

{Escrita dos elementos intercalados}
escreva "VALORES INTERCALADOS"
I ← 1
repita
    se I > IC
    então interrompa
fim se
escreva C[I]
I ← I + 1
fim repita
fim algoritmo.

```

▲ 2.5.1.11.

Algoritmo

{Definição do tipo das variáveis;

```
declare I, J, M, N, P numérico
declare A[1:30] numérico
declare B[1:20] numérico
declare C[1:50] numérico
declare PRESENTE lógico
```

{Leitura de M e dos M valores de A}

escreva "Digite M seguido dos M valores de A"

leia M

I \leftarrow 1

repita

```
se I  $>$  M  
| então interrompa
```

fim se

leia A[I]

I \leftarrow I + 1

fim repita

{Leitura de N e dos N valores de B}

escreva "Digite N seguido dos N valores de B"

leia N

I \leftarrow 1

repita

```
se I  $>$  N  
| então interrompa
```

fim se

leia B[I]

I \leftarrow I + 1

fim repita

{Colocação em C dos elementos resultantes da união de A com B;

{Transferencia dos elementos de A para C;

I \leftarrow 1

repita

```
se I  $>$  M  
| então interrompa
```

fim se

C[I] \leftarrow A[I]

I \leftarrow I + 1

fim repita

{Transferência para C dos elementos de B que não estão em C;

P \leftarrow M

I \leftarrow 1

repita

```
se I  $>$  N  
| então interrompa
```

fim se

PRESENTE \leftarrow falso

J \leftarrow 1

repita

```
se J  $>$  P ou PRESENTE  
| então interrompa
```

fim se

se B[I] = C[J]

| então PRESENTE \leftarrow verdadeiro

fim se

J \leftarrow J + 1

fim repita

se PRESENTE = falso

| então P \leftarrow P + 1

C[P] \leftarrow B[I]

fim se

fim repita

{Escrita de C;

escreva "União de A com B"

J \leftarrow 1

repita

```
se J  $>$  P  
| então interrompa
```

fim se

escreva C[J]

J \leftarrow J + 1

fim repita

fim algoritmo

▲ 2.5.1.12.

Algoritmo

{Definição do tipo das variáveis;

declare P, X, I, N, IX, NX numérico

declare A[0:20] numérico

{Leitura de N seguido de A[0] ... A[N];

escreva "Digite N seguido dos valores A[0] ... A[N]"

leia N

I \leftarrow 0

repita

```
se I  $>$  N  
| então interrompa
```

fim se

leia A[I]

I \leftarrow I + 1

fim repita

{Cálculo e escrita de P para 10 valores de X;

NX \leftarrow 10

IX \leftarrow 1

```

repita
  se IX > NX
    então interrompa
  fim se

```

leia X

{Leitura de um valor de X;}

P ← 0

I ← N

repita

se I < 0

então interrompa

fim se

P ← P + X + A[I]

I ← I - 1

fim repita

escreva "P =", P

IX ← IX + 1

fim repita

fim algoritmo.

{Cálculo de P = ((A₀x + A₁)x + A₂)x + ...}

{Escrjta de P;

2.5.2.5.

Algoritmo

{Definição do tipo das variáveis;

declare A, B, C[1:20,1:30] numérico

declare M, N, I, J numérico

{Leitura de M e N e da matriz A;

escreva "Digite as dimensões M e N das matrizes"

leia M, N

escreva "Digite as M linhas de A, cada uma com N valores"

I ← 1

repita

se I > M

então interrompa

fim se

J ← 1

repita

se J > N

então interrompa

fim se

leia A[I, J]

J ← J + 1

fim repita

I ← I - 1

fim repita

{Leitura da matriz B;

escreva "Digite as M linhas de B, cada uma com N valores"

I ← 1

repita

se I > M

então interrompa

fim se

J ← 1

repita

se J > N

então interrompa

fim se

leia B[I, J]

J ← J + 1

fim repita

I ← I - 1

fim repita

I ← 1

{Cálculo da matriz soma C = A + B;

repita

se I > M

então interrompa

fim se

J ← 1

repita

se J > N

então interrompa

fim se

C[I, J] ← A[I, J] + B[I, J]

J ← J + 1

fim repita

I ← I - 1

fim repita

escreva "SOMA DAS MATRIZES"

{Escrjta da matriz soma C;

I ← 1

repita

se I > M

então interrompa

fim se

Mude de linha

J ← 1

repita

se J > N

então interrompa

fim se

escreva C[I, J]

J ← J + 1

fim repita

I ← I - 1

fim repita

fim algoritmo.

▲ 2.5.2.7.

Algoritmo

{Definição do tipo das variáveis}

```

declare A[1:20, 1:10] numérico
declare B[1:20, 1:11] numérico
declare M, N, I, J numérico

    (Leitura de M e N e da matriz A)
escreva "Digite as dimensões M e N da matriz A"
leia M, N
escreva "Digite as M linhas de A, cada uma com N valores"
I ← 1
repita
    se I > M
        então interrompa
    fim se
    J ← 1
    repita
        se J > N
            então interrompa
        fim se
        leia A[I, J]
        J ← J + 1
    fim repita
    I ← I + 1
fim repita
escreva "MATRIZ LIDA"
I ← 1
repita
    se I > M
        então interrompa
    fim se
    Mude de linha
    J ← 1
    repita
        se J > N
            então interrompa
        fim se
        escreva A[I, J]
        J ← J + 1
    fim repita
    I ← I + 1
fim repita
I ← 1
repita
    se I > M
        então interrompa
    fim se
Cálculo da matriz modificada B:

```

B[I, N+1] ← 1

J ← 1

repita

se J > N

então interrompa

fim se

B[I, J] ← A[I, J]

B[I, N + 1] ← B[I, N + 1] × B[I, J]

J ← J + 1

fim repita

I ← I + 1

fim repita

{Escrita da matriz modificada B}

escreva "MATRIZ MODIFICADA"

I ← 1

repita

se I > M

então interrompa

fim se

J ← 1

repita

se J > N + 1

então interrompa

fim se

escreva B[I, J]

J ← J + 1

fim repita

I ← I + 1

fim repita

fim algoritmo

▲ 2.5.2.11.

Algoritmo

{Definição do tipo das variáveis}

declare POP[1:26, 1:10] numérico

declare MUN, EST, NMUN, NEST, MUNMAIS, ESTMAIS numérico

declare MAIS, SOMA, MÉDIA numérico

{Leitura da Matriz POP, com a população}

NEST ← 26

NMUN ← 10

escreva "Digite a população de ", NMUN, " Municípios de ", NEST, "Estados"

EST ← 1

repita

se EST > NEST

então interrompa

fim se

MUN ← 1

```

repita
    se MUN > NMUN
        então interrompa
    fim se
    leia POP[EST, MUN]
    MUN ← MUN + 1
fim repita
    EST ← EST + 1
fim repita
escreva "MATRIZ LIDA"
EST ← 1
repita
    se EST > NEST
        então interrompa
    fim se
    Muda de linha
    MUN ← 1
repita
    se MUN > NMUN
        então interrompa
    fim se
    escreva POP[EST, MUN]
    MUN ← MUN + 1
fim repita
    EST ← EST + 1
fim repita
    {Município mais populoso do país}

```

{Escrita de POP}

```

    EST ← EST + 1
fim repita
    {Escrita de MUNMAIS, ESTMAIS}
    escreva "Municipio ", MUNMAIS, " do Estado ", ESTMAIS, "é o mais populoso"
    {Média da população das capitais}.

    SOMA ← 0
    EST ← 1
    repita
        se EST > NEST
            então interrompa
        fim se
        SOMA ← SOMA + POP[EST, 1]
        EST ← EST + 1
    fim repita
    MÉDIA ← SOMA / NEST
    {Escrita da MÉDIA}
    escreva "Média da população das capitais = ", MÉDIA
fim algoritmo.

```

▲ 2.5.2.12.

Algoritmo

{Definição do tipo das variáveis}

```

declare TABELA[1:15, 1:7] numérico
declare CUSTOUNITREC[1:7] numérico
declare CUSTOUNTATIV, INSSRECUNITATIV, QUANTIDADE [1:15] numérico
declare ADM, LEISSOC, LUCRO, OBRA, TOTAL, ATIV, NATIV, REC numérico
    {Leitura de ADM}
escreva "Digite o percentual de administração"
leia ADM
    {Leitura de CUSTOUNTREC}
escreva "Digite o custo unitário dos 7 recursos"
REC ← 1
repita
    se REC > 7
        então interrompa
    fim se
    leia CUSTOUNTREC[REC]
    REC ← REC + 1
fim repita
    {Leitura da TABELA de Quantitativos}
escreva "Digite o número total de atividades"
leia NATIV
escreva "Digite para cada atividade (NATIV), a quantidade de recursos (?)"
ATIV ← 1
repita
    se ATIV > NATIV
        então interrompa
    fim se

```

```

REC ← 1
repita
    se REC > 7
        então interrompa
    fim se
    leia TABELA[ATIV, REC]
    REC ← REC + 1
fim repita
ATIV ← ATIV + 1
fim repita
{Cálculo e escrita de CUSTOUNITATIV}
escreva "1- PREÇO DE CUSTO UNITÁRIO (DIRETO+ADM) DE CADA",
    "ATIVIDADE"
ATIV ← 1
repita
    se ATIV > NATIV
        então interrompa
    fim se
    CUSTOUNITATIV[ATIV] ← 0
    REC ← 1
    repita
        se REC > 7
            então interrompa
        fim se
        CUSTOUNITATIV[ATIV] ← CUSTOUNITATIV[ATIV] + TABELA[ATIV, REC]
            × CUSTOUNITREC[REC]
        REC ← REC + 1
    fim repita
    CUSTOUNITATIV[ATIV] ← CUSTOUNITATIV[ATIV] × (1 + ADM / 100)
    escreva CUSTOUNITATIV[ATIV]
    ATIV ← ATIV + 1
fim repita
{Cálculo e escrita de CUSTOUNITTIV × 1,36}
escreva "2- PREÇO UNITÁRIO A SER COBRADO EM CADA ATIVIDADE"
ATIV ← 1
repita
    se ATIV > NATIV
        então interrompa
    fim se
    escreva CUSTOUNITATIV[ATIV] × 1,36
    ATIV ← ATIV + 1
fim repita
{Recolhimento ao INSS por atividade}
escreva "3- A SER RECOLHIDO AO INSS EM CADA UNID DE ATIV"
ATIV ← 1

```

```

repita
    se ATIV > NATIV
        então interrompa
    fim se
    INSSRECUNITATIV[ATIV] ← (TABELA[ATIV,4] × CUSTOUNITREC [4]
        + TABELA[ATIV,5] × CUSTOUNITREC[5]) × 0,16
    escreva INSSRECUNITATIV[ATIV]
    ATIV ← ATIV + 1
fim repita
{Cobrança por uma obra}
escreva "Digite a quantidade de cada atividade usada na obra"
ATIV ← 1
repita
    se ATIV > NATIV
        então interrompa
    fim se
    leia QUANTIDADE[ATIV]
    ATIV ← ATIV + 1
fim repita
escreva "4- A SER COBRADO PELA OBRA"
OBRA ← 0
ATIV ← 1
repita
    se ATIV > NATIV
        então interrompa
    fim se
    OBRA ← OBRA + (CUSTOUNITATIV[ATIV] + 1,36 + INSSRECUNITATIV[ATIV])
        × QUANTIDADE[ATIV]
    ATIV ← ATIV + 1
fim repita
escreva OBRA
{Total de cada recurso envolvido na obra}
escreva "5- QUANTIDADE TOTAL DE CADA RECURSO ENVOLVIDO"
REC ← 1
repita
    se REC > 7
        então interrompa
    fim se
    TOTAL ← 0
    ATIV ← 1
    repita
        se ATIV > NATIV
            então interrompa
        fim se
        TOTAL ← TOTAL + TABELA[ATIV, REC] × QUANTIDADE[ATIV]
        ATIV ← ATIV + 1
    fim repita

```

```

escreva TOTAL
REC  $\leftarrow$  REC + 1
fim repita
fim algoritmo.

```

▲ 2.5.2.14.

Algoritmo

{Definição do tipo das variáveis}

```
declare A[1:20, 1:20] numérico
```

```
declare M, I, J numérico
```

```
declare SIMÉTRICA lógica
```

{Leitura de M e A}

```
escreva "Digite M, seguido de M linhas de A, cada uma com M elementos"
```

```
leia M
```

```
I  $\leftarrow$  1
```

```
repita
```

```
  se I > M
```

```
    | então interrompa
```

```
  fim se
```

```
  J  $\leftarrow$  1
```

```
repita
```

```
  | se J > M
```

```
    | então interrompa
```

```
  | fim se
```

```
  | leia A[I, J]
```

```
  | J  $\leftarrow$  J + 1
```

```
  fim repita
```

```
  I  $\leftarrow$  I + 1
```

```
fim repita
```

{Verificação da simetria da matriz A}

```
SIMÉTRICA  $\leftarrow$  verdadeiro
```

```
I  $\leftarrow$  2
```

```
repita
```

```
  se I > M
```

```
    | então interrompa
```

```
  fim se
```

```
  J  $\leftarrow$  1
```

```
repita
```

```
  | se J > I ou SIMÉTRICA = falso
```

```
    | então interrompa
```

```
  | fim se
```

```
  | se A[I, J]  $\neq$  A[J, I]
```

```
    | então SIMÉTRICA  $\leftarrow$  falso
```

```
  | fim se
```

```
  | J  $\leftarrow$  J + 1
```

```
  fim repita
```

```
  I  $\leftarrow$  I + 1
```

```
fim repita
```

{Escrita da simetria de A}

```
se SIMÉTRICA
```

```
| então escreva "SIMÉTRICA"
```

```
| senão escreva "NÃO SIMÉTRICA"
```

```
fim se
```

```
fim algoritmo.
```

▲ 2.5.2.15.

Algoritmo

{Definição do tipo das variáveis}

```
declare N, I, J, SOMA numérico
```

```
declare A[1:20, 1:20] numérico
```

```
declare B, X[1:20] numérico
```

{Leitura de N e da matriz A}

```
leia N
```

```
I  $\leftarrow$  1
```

```
repita
```

```
  se I > N
```

```
    | então interrompa
```

```
  fim se
```

```
  J  $\leftarrow$  1
```

```
repita
```

```
  | se J > I
```

```
    | então interrompa
```

```
  | fim se
```

```
  | leia A[I, J]
```

```
  | J  $\leftarrow$  J + 1
```

```
  fim repita
```

```
  I  $\leftarrow$  I + 1
```

```
fim repita
```

{Leitura de B}

```
I  $\leftarrow$  1
```

```
repita
```

```
  se I > N
```

```
    | então interrompa
```

```
  fim se
```

```
  | leia B[I]
```

```
  | I  $\leftarrow$  I + 1
```

```
fim repita
```

{Cálculo de X}

```
X[1]  $\leftarrow$  B[1] / A[1,1]
```

```
I  $\leftarrow$  2
```

```
repita
```

```
  se I > N
```

```
    | então interrompa
```

```
  fim se
```

```
  | SOMA  $\leftarrow$  0
```

```
  | J  $\leftarrow$  1
```

```

repita
  se J > I - 1
    então interrompa
  fim se
  SOMA ← SOMA + A[I, J] × X[J]
  J ← J + 1
fim repita
X[I] ← (B[I] - SOMA) / A[I, I]
I ← I + 1
fim repita
{Escrita de X}
escreva "RAÍZES DO SISTEMA DE EQUAÇÕES"
I ← 1
repita
  se I > N
    então interrompa
  fim se
  escreva X[I]
  I ← I + 1
fim repita
fim algoritmo.

```

▲ 2.5.3.5.

Algoritmo

{Definição do tipo das variáveis}

```

declare TABDIST[1:30,1:30] numérico
declare CODCID[1:24] numérico
declare I, J, N, TOTLINHAS, ÔNIBUS, NCID, DIST numérico
{Leitura da tabela de distâncias}

TOTLINHAS ← 30
escreva "Digite em ", TOTLINHAS - 1, "linhas a parte triangular inferior da ",
"tabela de distâncias"
TABDIST[1, 1] ← 0
I ← 2
repita
  se I > TOTLINHAS
    então interrompa
  fim se
  J ← 1
  repita
    se J > I - 1
      então interrompa
    fim se
    leia TABDIST[I, J]
    TABDIST[J, I] ← TABDIST[I, J]
    J ← J + 1
  fim repita
TABDIST[I, I] ← 0

```

| 1 ← 1 + 1
fim repita

{Escrita da Tabela de Distâncias;
escreva "TABELA DE DISTÂNCIAS:"}

```

I ← 1
repita
  se I > TOTLINHAS
    então interrompa
  fim se
  J ← 1
  repita
    se J > TOTLINHAS
      então interrompa
    fim se
    escreva TABDIST[I, J]
    J ← J + 1
  fim repita
  I ← I + 1
fim repita
repita
  leia ÔNIBUS {Leitura do número de uma linha de ônibus}
  se ÔNIBUS = 0
    então interrompa
  fim se
  {Leitura do percurso da linha de ônibus}
  escreva "Digite o nº de cidades percorridas e o código de cada uma a"
  leia NCID
  N ← 1
  repita
    se N > NCID
      então interrompa
    fim se
    leia CODCID[N]
  fim repita
  DIST ← 0 {Cálculo da distância percorrida}
  N ← 1
  repita
    se N > NCID - 1
      então interrompa
    fim se
    DIST ← DIST + TABDIST[CODCID[N], CODCID[N-1]]
    N ← N + 1
  fim repita
  {Escrita da distância percorrida}
  escreva "DISTÂNCIA PERCORRIDA = ", DIST
fim repita
fim algoritmo.

```

▲ 2.5.3.6.

Algoritmo

```

{Definição do tipo das variáveis;
declare VÁLIDAS[1:100], INVÁLIDAS[1:100] CONTA
declare CONTA registro (NUM,      {Número da conta bancária}
                      DV,        {Digito verificador do nº da conta}
                      SALDO     {Saldo da conta}
                      numérico,
                      NOME      {Nome do titular da conta;
                      literal }

declare NÚMERO, SOMA, PESO, DIGVER, D, I, J, K numérico
I ← 0
J ← 0
repita
    leia CONTA.NUM, CONTA.DV, CONTA.SALDO, CONTA.NOME
    se CONTA.NUM = 0
        então interrompa
    fim se
    SOMA ← 0      {Cálculo do dígito verificador do número da conta}
    PESO ← 2
    NÚMERO ← CONTA.NUM
    repita
        se NÚMERO = 0
            então interrompa
        fim se
        D ← RESTO(NÚMERO,10)
        SOMA ← SOMA + (D × PESO)
        NÚMERO ← QUOCIENTE(NÚMERO,10)
        PESO ← PESO + 1
    fim repita
    DIGVER ← 11 - RESTO(SOMA,11)
    se DIGVER > 9
        então DIGVER ← 0
    fim se
    se DIVGER = CONTA.DV      {Verificação da validade do número da conta}
        então I ← I + 1
        VÁLIDAS[I] ← CONTA
    senão J ← J + 1
        INVÁLIDAS[J] ← CONTA
    fim se
fim repita
K ← 1                  {Escrita das contas válidas;
escreva "CONTAS DE NÚMERO CORRETO"
repita
    se K > I
        então interrompa
    fim se

```

escreva VÁLIDAS[K].CONTA.NUM, "-", VÁLIDAS[K].CONTA.DV,
VÁLIDAS[K].CONTA.SALDO, VÁLIDAS[K].CONTA.NOME

fim repita

K ← 1 {Escrita das contas inválidas;

escreva "CONTAS DE NÚMERO ERRADO"

repita

se K > J

então interrompa

fim se

escreva INVÁLIDAS[K].CONTA.NUM, "-", INVÁLIDAS[K].CONTA.DV,
INVÁLIDAS[K].CONTA.SALDO, INVÁLIDAS[K].CONTA.NOME

fim repita

fim algoritmo.

▲ 2.5.4.4.

Algoritmo

{Definição do tipo das variáveis}

declare BOLETA {Operações dos clientes na Bolsa de Valores}

arquivo seqüencial de OPERAÇÃO

declare RESULTADO {Resultados totais das operações dos clientes}

arquivo seqüencial de SALDOCLIENTE

declare OPERAÇÃO registro (NÚMERO, {Identificação do cliente}

CÓDIGO, {V-venda, C-compra}

DESCRIÇÃO {Título negociado}

literal,

VALOR {Valor unitário do título negociado}

QUANTIDADE {Quantidade de títulos negociados}

numérico)

declare SALDOCLIENTE registro (NÚMERO, {Identificação do cliente}

TIPO, {C-credor, D-devedor}

literal,

SALDO, {Saldo apurado do cliente}

numérico)

declare CLIENTEATUAL literal

declare TCOMPRAS, TVENDAS numérico

abra BOLETA leitura

abra RESULTADO escrita

TCOMPRAS ← 0

TVENDAS ← 0

leia BOLETA.OPERAÇÃO

CLIENTEATUAL ← OPERAÇÃO.NÚMERO

repita

se OPERAÇÃO.FDA

então interrompa

fim se

repita

se OPERAÇÃO.CÓDIGO = "C"

{Acúmulo de operação}

```

    compradas ← compras +
    (OPERAÇÃO.QUANTIDADE × OPERAÇÃO.VALOR)
    senão TVENDAS ← VENDAS +
    (OPERAÇÃO.QUANTIDADE × OPERAÇÃO.VALOR)
  fim se
  leia BOLETA.OPERAÇÃO
  se (OPERAÇÃO.FDA) ou (CLIENTEATUAL ≠ OPERAÇÃO.NÚMERO)
    então interrompa
  fim se
  fim repita
  se CLIENTEATUAL ≠ OPERAÇÃO.NÚMERO
    então se TCOMPRAS > TVENDAS           {Cálculo do saldo do cliente}
      então SALDOCLIENTE.SALDO ← TCOMPRAS - TVENDAS
          SALDOCLIENTE.TIPO ← "C"
      senão se TCOMPRAS < TVENDAS
        então SALDOCLIENTE.SALDO ← TVENDAS - TCOMPRAS
            SALDOCLIENTE.TIPO ← "D"
        senão SALDOCLIENTE.SALDO ← 0
            SALDOCLIENTE.TIPO ← " "
      fim se
    fim se
    SALDOCLIENTE.NÚMERO ← CLIENTEATUAL
    escreva RESULTADO, SALDOCLIENTE
    TCOMPRAS ← 0
    TVENDAS ← 0
    CLIENTEATUAL ← OPERAÇÃO.NÚMERO
  fim se
  fim repita
  se TCOMPRAS > TVENDAS
    então SALDOCLIENTE.SALDO ← TCOMPRAS - TVENDAS
        SALDOCLIENTE.TIPO ← "C"
    senão se TCOMPRAS < TVENDAS
      então SALDOCLIENTE.SALDO ← TVENDAS - TCOMPRAS
          SALDOCLIENTE.TIPO ← "D"
      senão SALDOCLIENTE.SALDO ← 0
          SALDOCLIENTE.TIPO ← " "
    fim se
  fim se
  SALDOCLIENTE.NÚMERO ← CLIENTEATUAL
  escreva RESULTADO, SALDOCLIENTE
  feche BOLETA
  feche RESULTADO
fim algoritmo.

```

Capítulo 3 - Exercícios propostos

▲ 3.4.3.

Algoritmo

{Declaração da sub-rotina que soma os três maiores valores}

```

subrotina SOMATRESMAIORES(A,B,C,D,SOMA)
  declare A, B, C, D,      {Números fornecidos}
        SOMA             {Soma dos três maiores números;
                           numérico}
  se A < B e A < C e A < D
    então SOMA ← B + C + D
  senão se B < C e B < D
    então SOMA ← A + C + D
  senão se C < D
    então SOMA ← A + B + D
  senão SOMA ← A + B + C
  fim se
fim se
fim subrotina

```

{Definição do tipo das variáveis utilizadas}

```

declare N1, N2, N3, N4,      {Notas das provas mensais}
      NF,                  {Nota da prova final}
      NALUNO,               {Número do aluno}
      NOTAFINAL,            {Nota final do aluno}
      SOMA,                 {Soma das três maiores notas mensais}
      I                      {Contador}
      numérico
declare CONCEITO            {Conceito final obtido pelo aluno}
      literal
I ← 0
repita
  leia NALUNO, N1, N2, N3, N4, NF
  SOMATRESMAIORES(N1,N2,N3,N4,SOMA)
  NOTAFINAL ← SOMA + NF
  se NOTAFINAL ≤ 39
    então CONCEITO ← "F"
  senão se NOTAFINAL ≤ 59
    então CONCEITO ← "E"
  senão se NOTAFINAL ≤ 69
    então CONCEITO ← "D"
  senão se NOTAFINAL ≤ 79
    então CONCEITO ← "C"
  senão se NOTAFINAL ≤ 89
    então CONCEITO ← "B"
  senão CONCEITO ← "A"
fim se

```

```

    fim se
fim se
escreva NALUNO, NOTAFINAL, CONCEITO
I ← I + 1
se I = 80
    então interrompa
fim se
fim se
fim repita
fim algoritmo.

```

3.4.5.

Algoritmo

{Declaração da função que calcula um dígito verificador}

```

função numérico DÍGITOVERIFYCADOR(NÚMERO)
declare NÚMERO, SOMA, PESO, D, DV numérico
SOMA ← 0
PESO ← 2
repita
    D ← RESTO(NÚMERO, 10)
    SOMA ← SOMA + (D × PESO)
    NÚMERO ← QUOCIENTE(NÚMERO, 10)
    se NÚMERO = 0
        então interrompa
    fim se
    PESO ← PESO + 1
fim repita
DV ← 11 - RESTO(SOMA, 11)
se DV > 9
    então DÍGITOVERIFYCADOR ← 0
    senão DÍGITOVERIFYCADOR ← DV
fim se
fim função

declare CPF,          {Definição do tipo das variáveis}
      NUM,           {Nº de inscrição no Cadastro de Pessoas Físicas}
      DV1, DV2,       {Parte do CPF sem os dígitos de controle}
      numérico        {Dígitos verificadores}

declare NOME           {Nome da pessoa}
      literal

repita
    leia NOME, CPF
    se CPF = 0
        então interrompa
    fim se
    NUM ← QUOCIENTE(CPF, 100)
    DV1 ← DÍGITOVERIFYCADOR(NUM)

```

```

NUM ← NUM × 10 + DV1
DV2 ← DÍGITOVERIFYCADOR(NUM)
NUM ← NUM × 10 + DV2
se NUM = CPF
    então escreva NOME, CPF, "VÁLIDO"
    senão escreva NOME, CPF, "INVÁLIDO"
fim se
fim repita
fim algoritmo.

```

3.4.7.

Algoritmo

{Declaração da sub-rotina que calcula o número de dias do ano
(até uma data fornecida)}

```

subrotina QUANTOSDIAS(DIA, MÊS, ANO, N)
declare DIA, MÊS, ANO, N, I numérico
{Data fornecida como parâmetro de entrada;
{Número de dias do ano até a data fornecida;
{Contador}

N ← DIA
I ← 1
repita
    se I > MÊS - 1
        então interrompa
    fim se
    se I = 2
        então se RESTO(ANO, 400) = 0 ou
            (RESTO(ANO, 4) = 0 e RESTO(ANO, 100) = 0)
            então N ← N + 29
            senão N ← N + 28
        fim se
        senão se I = 4 ou I = 6 ou I = 9 ou I = 11
            então N ← N + 30
            senão N ← N + 31
        fim se
    fim se
    I ← I + 1
fim repita
fim subrotina

{Declaração da função que verifica se uma data é válida }

função lógico DATAVÁLIDA(DIA, MÊS, ANO)
declare DIA, MÊS, ANO numérico
declare DIASDOMÊS[1:12] {Tabela do número de dias em cada mês;
                        numérico
DIASDOMÊS[1] ← 31
DIASDOMÊS[2] ← 29
DIASDOMÊS[3] ← 31
DIASDOMÊS[4] ← 30

```

```

DIASDOMÊS[0] ← 31
DIASDOMÊS[6] ← 30
DIASDOMÊS[7] ← 31
DIASDOMÊS[8] ← 31
DIASDOMÊS[9] ← 30
DIASDOMÊS[10] ← 31
DIASDOMÊS[11] ← 30
DIASDOMÊS[12] ← 31
DATAVÁLIDA ← verdadeiro
se ANO < 0
    então DATAVÁLIDA ← falso
    senão se I < 1 ou I > 12
        então DATAVÁLIDA ← falso
        senão se (DIA < 1) ou (DIA > DIASDOMÊS[MÊS]) ou
            (MÊS = 2 e DIA = 29 e não((RESTO(ANO,400) = 0 ou
            (RESTO(ANO,4) = 0 e RESTO(ANO,100) ≠ 0)))
                então DATAVÁLIDA ← falso
            fim se
        fim se
    fim se
fim função

{Definição do tipo das variáveis}
declare D1, M1, A1, D2, M2, A2, {Datas de um mesmo ano}
    N1, N2, {Número de dias do ano para cada data}
    DIFDIAS {Diferença em dias entre as datas}
    numérico
repita
    leia D1, M1, A1, D2, M2, A2
    se D1=0 e M1=0 e A1=0 e D2=0 e M2=0 e A2=0
        então interrompa
    fim se
    se não DATAVÁLIDA(D1, M1, A1)
        então escreva "DATA INCORRETA", D1, "/", M1, "/", A1
    senão se não DATAVÁLIDA(D2, M2, A2)
        então escreva "DATA INCORRETA", D2, "/", M2, "/", A2
        senão se A1 ≠ A2
            então escreva "DATAS INCORRETAS: "
            "ANOS DIFERENTES"
            senão QUANTOSDIAS(D1, M1, A1, N1)
                QUANTOSDIAS(D2, M2, A2, N2)
                DIFDIAS ← ABS(N1 - N2)
                escreva DIFDIAS
            fim se
        fim se
    fim se
fim repita
fim algoritmo.

```

Algoritmo

{Declaração da função que calcula o produto de dois números inteiros}

função numérico PRODUTO(X,Y)

declare X, Y, {Fatores da multiplicação}
 P {Produto}
 numérico

P ← 0

repita

 se RESTO(X, 2) ≠ 0
 então P ← P + Y

 fim se

 X ← QUOCIENTE(X, 2)

 Y ← Y × 2

 se X = 0
 então interrompa

 fim se

fim repita

PRODUTO ← P

fim função

{Definição do tipo das variáveis}

declare A, B, {Par de números inteiros positivos}
 PAB, {Produto A × B}
 I {Contador}
 numérico

I ← 0

repita

 leia A, B

 PAB ← PRODUTO(A,B)

 escreva A, B, PAB

 I ← I + 1

 se I = 10
 então interrompa

 fim se

fim repita

fim algoritmo.

▲ 3.4.10.

Algoritmo

{Declaração da função que calcula o número de algarismos existentes em um número inteiro}

função numérico NUMALGARISMOS(N)

declare N, NA numérico

NA ← 1

repita

 se QUOCIENTE(N,10) = 0
 então interrompa

 fim se

```

    NA ← NA + 1
    N ← QUOCIENTE(N,10)
  fim repita
  NUMALGARISMOS ← NA
fim função

```

{Declaração da sub-rotina que separa os algarismos de um número inteiro }

subrotina SEPARAALGARISMOS(NUM,NA,ALGARISMOS)

```

declare NUM, {Número fornecido}
      NA, {Quantidade de algarismos de NUM}
      I {Contador}
      numérico
declare ALGARISMOS[1:NA] {Algarismos do número NUM}
      numérico

```

I ← 0

repita

```

    I ← I + 1
    ALGARISMOS[I] ← RESTO(NUM,10)
    NUM ← QUOCIENTE(NUM,10)
  se I = NA
    então interrompa
  fim se
fim repita

```

fim subrotina

{Declaração da sub-rotina que forma um número na ordem inversa}

subrotina INVERTENÚMERO(NA,ALGS,N)

```

declare N, {Número inverso de saída}
      NA, {Quantidade de algarismos de N}
      I, ORDEM {Contadores}
      numérico
declare ALGS[1:NA] {Algarismos do número N}
      numérico

```

N ← 0

ORDEM ← 1

I ← NA

repita

```

    N ← N + ALGS[I] × ORDEM
    I ← I - 1
    ORDEM ← ORDEM × 10
  se I = 0
    então interrompa
  fim se
fim repita

```

fim subrotina

{Definição do tipo das variáveis}

```

declare N, {Número quadrado perfeito}
      K, {Contador}
      INVERSON {Número N invertido}

```

numérico
declare ALGARISMOS[1:4] {Algarismos de N}

numérico

K ← 0

repita

```

    N ← K2
    SEPARAALGARISMOS(N,NUMALGARISMOS(N),ALGARISMOS)
    INVERTENÚMERO(NUMALGARISMOS(N),ALGARISMOS,INVERSON)
  se N = INVERSON
    então escreva N
  fim se

```

K ← K + 1

se K > 70

então interrompa

fim se

fim repita

3.4.13.

Algoritmo

{Declaração da função que calcula a distância entre dois pontos no espaço}

função numérica DISTÂNCIA(X1,Y1,Z1,X2,Y2,Z2)

declare X1,Y1,Z1, X2, Y2, Z2 {Coordenadas dos pontos}

numérico

DISTÂNCIA ← $\sqrt{(X2 - X1)^2 + (Y2 - Y1)^2 + (Z2 - Z1)^2}$

fim função

{Declaração da sub-rotina que calcula a área de um triângulo}

subrotina ÁREATRIÂNGULO(A,B,C,ÁREA)

declare A, B, C, {Lados do triângulo}

ÁREA, {Área do triângulo}

P {Semiperímetro do triângulo}

numérico

P ← (A + B + C) / 2

ÁREA ← $\sqrt{P \times (P - A) \times (P - B) \times (P - C)}$

fim subrotina

{Definição do tipo das variáveis}

declare X1, Y1, Z1,

{Coordenadas do vértice 1 do tetraedro}

X2, Y2, Z2,

{Coordenadas do vértice 2 do tetraedro}

X3, Y3, Z3,

{Coordenadas do vértice 3 do tetraedro}

X4, Y4, Z4,

{Coordenadas do vértice 4 do tetraedro}

AT1, AT2, AT3, AT4,

{Áreas das faces de um tetraedro}

ÁREATOTAL,

{Área total do tetraedro}

I {Contador}

numérico

I ← 0

repita

leia X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4

```

AREATRIÂNGULO(DISTÂNCIA(X1,Y1,Z1,X2,Y2,Z2),
    DISTÂNCIA(X1,Y1,Z1,X3,Y3,Z3),
    DISTÂNCIA(X2,Y2,Z2,X3,Y3,Z3), AT1)
AREATRIÂNGULO(DISTÂNCIA(X1,Y1,Z1,X2,Y2,Z2),
    DISTÂNCIA(X1,Y1,Z1,X4,Y4,Z4),
    DISTÂNCIA(X2,Y2,Z2,X4,Y4,Z4), AT2)
AREATRIÂNGULO(DISTÂNCIA(X1,Y1,Z1,X3,Y3,Z3),
    DISTÂNCIA(X1,Y1,Z1,X4,Y4,Z4),
    DISTÂNCIA(X3,Y3,Z3,X4,Y4,Z4), AT3)
AREATRIÂNGULO(DISTÂNCIA(X2,Y2,Z2,X3,Y3,Z3),
    DISTÂNCIA(X2,Y2,Z2,X4,Y4,Z4),
    DISTÂNCIA(X3,Y3,Z3,X4,Y4,Z4), AT4)
AREATOTAL ← AT1 + AT2 + AT3 + AT4
escreva AREATOTAL
I ← I + 1
se I = 10
    então interrompa
fim se
fim repita
fim algoritmo.

```

▲ 3.4.15.

Algoritmo

{Declaração da sub-rotina que calcula o valor da série}

```

subrotina CALCULANÚMEROE(NT,E )
declare NT, {Número de termos da série}
        E, {Valor da série – número neperiano}
        I, {Contador de termos da série}
        DEN {Denominador de um termo da série}
            numérico
E ← 0
DEN ← 1
I ← 0
repita
    se I = NT
        então interrompa
    fim se
    E ← E + ( 1 / DEN )
    I ← I + 1
    DEN ← DEN × I
fim repita
fim subrotina

```

{Definição do tipo das variáveis}

```

declare NTERMOS, {Número de termos da série}
        I, {Contador }
        E {Valor do número neperiano calculado pela série}
            numérico
NTERMOS ← 1

```

repita

```

CALCULANÚMEROE(NTERMOS,E)
se ABS( EXP(1) - E ) < 0,0001
    então interrompa
fim se
NTERMOS ← NTERMOS + 1
fim repita
escreva NTERMOS
fim algoritmo.

```

▲ 3.4.17.

Algoritmo

{Declaração da função que determina se um dado número é primo;
função lógico PRIMO(N)}

```

declare N, {Número a ser pesquisado}
        D {Possível divisor de N}
            numérico
D ← 2
repita
    se D > √N ou RESTO(N,D) = 0
        então interrompa
    fim se
    D ← D + 1
fim repita
se RESTO(N,D) ≠ 0
    então PRIMO ← verdadeiro
    senão PRIMO ← falso
fim se
fim função

```

{Definição do tipo das variáveis}

```

declare NPAR, {Número inteiro positivo par fornecido na entrada}
        J, K {Candidatos a números primos cuja soma é igual a NPAR}
            numérico

```

repita

```

leia NPAR
se NPAR < 0
    então interrompa
fim se
J ← 1
repita
    K ← NPAR - J
    se PRIMO(K) e PRIMO(J)
        então interrompa
    fim se
    J ← J + 2
fim repita
escreva NPAR, J, K
fim repita
fim algoritmo.

```

Declaração de registros

declare	lista-de-identificadores	registro	(componentes)
onde:			
declare	é uma palavra-chave;		
lista-de-identificadores	são os nomes que estão sendo associados aos registros que se deseja declarar;		
componentes	são declarações e/ou identificadores de variáveis compostas, separados por vírgula;		
registro	é uma palavra-chave.		

Declaração de conjuntos de registros

declare	lista-de-identificadores [li ₁ :ls ₁ , ..., li _n :ls _n] tipo
onde:	
declare	é uma palavra-chave;
lista-de-identificadores	são os nomes associados às variáveis compostas heterogêneas que se deseja declarar;
li ₁ :ls ₁ , ..., li _n :ls _n	são os limites dos intervalos de variação dos índices da variável, onde cada par de limites está associado a um índice;
tipo	identificador ou descrição do registro.

Referência ao conteúdo de variáveis compostas

(a) Variável composta homogênea

ident-da-variável [i]

(b) Variável composta heterogênea

ident-da-variável · ident-do-componente

(c) Conjunto de variáveis compostas heterogêneas

ident-da-variável [i] · ident-do-registro · ident-do-componente

onde:	ident-da-variável	é o nome associado à estrutura;
	ident-do-registro	é o nome associado ao registro;
	i	é uma lista de um ou mais índices capazes de individualizar um elemento do conjunto;
	ident-do-componente	é o nome associado ao componente cuja referência ao conteúdo está sendo feita.

Apêndice B

Este **apêndice** resume o conjunto de regras e convenções estabelecidas para a hotação algorítmica **definida neste** livro. Apresentam-se aqui as declarações e formas de referência às estruturas de dados, à **sintaxe dos** comandos existentes e às ferramentas de modularização disponíveis.

Declaração de variáveis simples

declare	lista-de-identificadores	nome-do-tipo
---------	--------------------------	--------------

onde:

- declare é uma palavra-chave;
- lista-de-identificadores são os nomes escolhidos para as variáveis, que devem estar separados por vírgula;
- nome-do-tipo é uma das três palavras-chaves: numérico, lógico ou literal.

Declaração de variáveis compostas homogêneas

declare	lista-de-identificadores [li ₁ :ls ₁ , ..., li _n :ls _n] tipo
---------	---

onde:

- declare é uma palavra-chave;
- lista-de-identificadores são os nomes que irão ser associados às variáveis múltiplas;
- li₁:ls₁, ..., li_n:ls_n são os limites dos intervalos de variação dos índices da variável, onde cada par de limites está associado a um índice;
- tipo tipo a que pertencem todos os componentes do conjunto.

Observações:

- li_k limite inferior do intervalo de variação do índice k;
- ls_k limite superior do intervalo de variação do índice k.

Operações aritméticas

PRIORIDADE	OPERAÇÃO
1.	potenciação, radiciação
2.	multiplicação, divisão
3.	adição, subtração

Principais funções aritméticas

NOME	RESULTADO FORNECIDO
LOG (EA)	logaritmo na base 10 de EA
LN (EA)	logaritmo neperiano de EA
EXP (EA)	o número e (base dos logaritmos neperianos) elevado a EA
ABS (EA)	valor absoluto de EA
TRUNCA (EA)	parte inteira de um número fracionário
ARREDONDA (EA)	transforma, por arredondamento, um número fracionário em inteiro
SINAL (EA)	fornecer o valor -1, +1 ou zero conforme o valor de EA seja negativo, positivo ou igual a zero
QUOCIENTE (EAx, EAy)	quociente inteiro da divisão de EAx por EAy
RESTO (EAx, EAy)	resto da divisão de EAx por EAy

Prioridade entre os operadores

PRIORIDADE	OPERADOR
1.	aritmético
2.	relacional
3.	não
4.	e
5.	ou

Estrutura de um algoritmo

```

Algoritmo
d1
d2
d3
.
.
dm
c1
c2
c3
.
.
cn
fim algoritmo

```

onde:

d1, d2, ..., dm são declarações
c1, c2, ..., cn são comandos

Comando de atribuição

identificador ← expressão

onde:

identificador é o nome da variável à qual está sendo atribuído o valor;
← é o símbolo de atribuição;
expressão pode ser uma expressão aritmética, expressão lógica ou
expressão literal de cuja avaliação é obtido o valor a ser
atribuído à variável.

Comando de entrada

leia lista-de-identificadores

onde:

leia é uma palavra-chave;
lista-de-identificadores são os nomes das variáveis, separados por vírgula,
nas quais serão armazenados os valores
provenientes do meio de entrada.

Comando de saída

escreva lista-de-identificadores e/ou constantes

onde:

escreva é uma palavra-chave;
lista-de-identificadores são os nomes das variáveis cujos conteúdos serão
mostrados ao usuário através de um meio de
saída. Além dos conteúdos das variáveis, o valor
de uma constante pode ser emitido diretamente.

Comando condicional simples

```

se condição
| então seqüência de comandos
fim se

```

Comando condicional composto

```

se condição
| então seqüência A de comandos
| senão seqüência B de comandos
fim se

```

Comando de repetição (interrupção no início)

```
repita
|   se condição
|   |   então interrompa
|   fim se
|   sequência B de comandos
fim repita
```

Comando de repetição (interrupção no meio)

```
repita
|   sequência A de comandos
|   se condição
|   |   então interrompa
|   fim se
|   sequência B de comandos
fim repita
```

Comando de repetição (interrupção no final)

```
repita
|   sequência A de comandos
|   se condição
|   |   então interrompa
|   fim se
fim repita
```

Declaração de arquivos

```
declare lista-de-identificadores arquivo organização de nome
```

onde:

declare é uma palavra-chave;
lista-de-identificadores são os nomes que serão usados pelo algoritmo para referenciar os arquivos;
arquivo é uma palavra-chave;
organização indica o tipo de organização do arquivo, que pode ser seqüencial ou direta;
de é uma palavra-chave;
nome é o nome do registro que será usado para se ter acesso ao arquivo.

Comandos de entrada e saída (arquivos de acesso seqüencial)

a) Comando de entrada

```
leia nome-do-arquivo · nome-do-registro
```

b) Comando de saída

```
escreva nome-do-arquivo · nome-do-registro
```

onde:

leia e escreva são palavras-chaves;
nome-do-arquivo é o nome do arquivo declarado no algoritmo;
nome-do-registro é o nome do registro usado para armazenar no ambiente de algoritmo o registro do arquivo após a leitura ou antes da escrita.

Comandos de entrada e saída (arquivos de acesso direto)

a) Comando de entrada

```
leia item [chave] nome-do-arquivo · nome-do-registro
```

b) Comando de saída

```
escreva item [chave] nome-do-arquivo · nome-do-registro
```

onde:

escreva, leia e item são palavras-chaves;
chave é o nome ou o valor do campo escolhido como chave para endereçar o registro;
nome-do-arquivo é o nome do arquivo declarado no algoritmo;
nome-do-registro é o nome do registro declarado para o arquivo e que será usado para armazenar no ambiente do algoritmo o registro após a leitura ou antes da escrita.

Declaração de funções

```
função tipo NOME (lista-de-parâmetros)
|   declaração dos objetos locais à função
|   comandos da função
fim função
```

onde:

função é uma palavra-chave;
tipo é a especificação do tipo do valor de retorno;
NOME é o identificador da função;
lista-de-parâmetros é a lista de objetos globais que serão utilizados dentro do procedimento e que têm seus valores passados do algoritmo ou procedimento chamador para este procedimento, e vice-versa.

Declaração de sub-rotinas

```
sub-rotina NOME (lista-de-parâmetros)
|   declaração dos objetos locais à sub-rotina
|   comandos da sub-rotina
fim sub-rotina
```

onde:

sub-rotina é uma palavra-chave;
NOME é o identificador da sub-rotina;
lista-de-parâmetros é a lista de objetos globais que serão utilizados dentro do procedimento e que têm seus valores passados do algoritmo ou procedimento chamador para este procedimento, e vice-versa.

Bibliografia

- AUERBACH. *Objectives of Structured Programming*. Philadelphia, Pa., 1974 (Computer Programming Management Portfolio, 14-02-01).
- BARTEE, T. C. *Fundamentos de Computadores Digitais*. Rio de Janeiro, Guanabara, 1980.
- DAHL, O. J.; DIJKSTRA, E. W.; HORRE, C. A. R. *Structured Programming*. London, Academic Press, 1972.
- DIJKSTRA, E. W. *A Discipline of Programming*. Englewood Cliffs, N. J., Prentice-Hall, 1976.
- GEAR, C. W. *Organização e Programação de Computadores*. Rio de Janeiro, Guanabara, 1980.
- MAIA, M. L. *Curso de Programação de Computadores*. Belo Horizonte, Publicação Interna do ICEx-UFMG, 1977.
- PARNAS, D. L. *On the Criteria to be used in Decomposing Systems into Modules*. Communications of the ACM, 15(12), 1972.
- STAA, A. V. *Engenharia de Programas*. Rio de Janeiro, LTC — Livros Técnicos e Científicos, 1983.
- WIRTH, N. *Program Development by Stepwise Refinement*. Communications of the ACM, 14(4), April, 1971.
- WIRTH, N. *From Programming Techniques to Programming Methods*. Gunther, A. et alii — International Computing Symposium, Amsterdam, North-Holland, 1974.

Índice Afabético

- A**
- Ação, conceito de, 14
 - ADA, 27
 - Álgebra das Proposições, 36
 - ALGOL, 25, 26
 - Algoritmos
 - conceituação, 14
 - estruturados, 19
 - refinamentos sucessivos, 15
 - Analógicos, 1
 - APL, 27
 - Aritméticas, expressões, 32
 - funções, 33
 - Arquivos
 - abertura, 150
 - conceito, 148
 - declaração, 149
 - fechamento, 151
 - organização
 - direta, 149, 160
 - seqüencial, 152
 - Atribuição, comando de, 40
- B**
- BASIC, 24
 - Blocos, diagrama de, 20
- C**
- Chapin, diagrama de, 26, 45
 - Clareza, 32
 - COBOL, 23
 - Codificação, 209
 - Comandos, 15, 44
 - Comentários, 32
 - Compilador, 23, 27
 - Comportamento, padrão de, 14
 - Condicional, estrutura, 16
 - composta, 46
 - simples, 45
 - Conjunção, 36
 - Constantes
 - literal, 30
 - lógica, 30
- D**
- Dados, estruturas de, 87
 - arquivos, 147
 - variáveis compostas heterogêneas, 131
 - variáveis compostas homogêneas, 90
 - Declarações, 44
 - arquivos, 149
 - variáveis compostas, 94, 111, 133
 - Diagrama hierárquico, 177
 - Digitação, 201
 - Digitais, estrutura dos, 2
 - Discos magnéticos e disquetes, unidades de, 8
 - Disjunção, 37
 - EAROM, 6
- E**
- Entrada, unidades de, 41, 181
 - Entrada, unidade de, 2, 7
 - arquivos, 152, 161, 162
 - sub-rotina, 180-181
 - EPROM, 6
 - Estruturada, programação, 175
- F**
- Falso, valor, 30, 35
 - Fatorial, 57
 - FDA (fim de arquivo), 152
 - Fitas magnéticas e cassetes, unidades de, 10
 - FORTH, 26
 - FORTRAN, 21, 24, 25
 - Funções, 33, 183
 - Fundamentais, itens, 29
- G**
- Gráficos, traçador de, 13
- I**
- Identificadores, 30
 - Impressora, 8
 - Interpretador, 27
 - Interrupção
 - no fim, 55
 - no início, 53
 - no interior, 54
- L**
- Leitoras/perfuradoras de cartão, 12
 - de caracteres magnéticos, 13
 - de caracteres ópticos, 13
 - de marcas ópticas, 13
 - Linguagem de programação
 - de máquina, 22
 - simbólica, 22
 - Literais, expressões, 39
 - Lógica e aritmética, unidades (ULA), 2
 - Lógicas, expressões
 - operadores lógicos, 36
 - prioridade, 38
 - relações, 35
 - Lógico, valor, 35
 - LOGO, 27
- M**
- Memórias, 2, 3
 - magnéticas, 3
 - semicondutoras, 4
 - Microprocessadores, 7
 - Modularização de programas, 175
 - ferramentas, 177
- N**
- Negação, 37
- O**
- Objetos
 - globais, 177
 - locais, 177
 - Operadores

- de concatenação, 39
- lógicos, 36
- relacionais, 35

P

Palavra-chave, 32

Parâmetros, passagem de
- por referência, 181
- por resultado, 181
- por valor, 181

Paridade, bit de, 3

PASCAL, 26

Periféricos, 7

PL/I, 24

Prioridade das operações, 38

Processamento, unidade central de
(UCP), 6

PROM, 6

R

RAM, 5

Registros, 131
- conjunto, 134
- físico, 148
- lógico, 148

Relações, 35

Repetição, estrutura de, 16, 53

ROM, 5, 6

RPG, 26

S

Saída, unidade de, 2, 7

- arquivos, 152
- sub-rotina, 179

Seqüencial, estrutura, 15, 44

- parâmetros, 180, 181

T

Teclado, 7

Tipos

- constantes, 29
- variáveis, 30

V

Variáveis

- compostas heterogêneas, 131
-- registros, 131
- compostas homogêneas, 90
-- multidimensionais, 110
-- unidimensionais, 93
- declaração de, 31
- identificadores, 31

Verdadeiro, valor, 30, 40

Vídeo, 7

