Augusto de Abreu Pires

ALGORITMOS

Fundamentos e construção

EDUNISO

ALGORITMOS

Fundamentos e construção

Editora da Universidade de Sorocaba - Eduniso

Biblioteca "Aluísio de Almeida"
Rodovia Raposo Tavares KM 92,5
18023-000 – Jardim Novo Eldorado
Sorocaba | SP | Brasil
Fone: 15 – 21017018
E-mail: eduniso@uniso.br
Site: https://editora.uniso.br

Augusto de Abreu Pires

ALGORITMOS

Fundamentos e construção

Sorocaba/SP
Eduniso
2024

Ficha técnica

Capa, projeto gráfico e diagramação: Eliezer Silva Proença
Normalização: Vilma Franzoni
Produção Editorial: Silmara Pereira da Silva Martins

Ficha Catalográfica

Pires, Augusto de Abreu.

Algoritmos: fundamentos e construção / Augusto de Abreu Pires. – Sorocaba, SP: Eduniso, 2024.

224p.

e-ISBN: 978-65-89550-20-4

DOI: 10.22482/eduniso.48

1. Algoritmos. 2. Lógica matemática. I. Título.

Elaborada por Vilma Franzoni – CRB 8/4485

Dedico este livro a todos, que de uma forma ou de outra, colaboraram com a construção desta obra e a todos que nunca tiveram uma obra a si dedicada!

SUMÁRIO

1	**APRESENTAÇÃO**	**9**
2	**UMA INTRODUÇÃO AO CÁLCULO PROPOSICIONAL**	**14**
2.1	Operação negação	16
2.2	Operação conjunção	17
2.3	Operação disjunção	18
2.3.1	Operação disjunção inclusiva	18
2.3.2	Operação disjunção exclusiva	19
2.4	Parênteses	26
2.5	Tabela verdade	27
2.6	A negação de uma conjunção	31
2.7	A negação de uma disjunção inclusiva	32
2.8	A negação de uma disjunção exclusiva	35
2.9	Exercícios propostos	38
2.10	Gabarito dos exercícios propostos	40
3	**A CONSTRUÇÃO DE ALGORITMOS**	**42**
3.1	Expressões e condições	43
3.2	Identificadores, declaração de variáveis e comandos atribuição, entrada e saída	55
3.3	Estrutura sequencial	63
3.4	Exercícios propostos	74
3.5	Gabarito dos exercícios propostos	76
3.6	Estruturas de seleção	79
3.6.1	Seleção simples	79
3.6.2	Seleção composta	84

3.6.3 Seleção encadeada 87
3.6.3.1 Seleção encadeada homogênea 87
3.6.3.2 Seleção encadeada heterogênea 101
3.6.4 Seleção de múltipla escolha 109
3.7 Exercícios propostos 117
3.8 Gabarito dos exercícios propostos 119
3.9 Estruturas de repetição 128
3.9.1 Repetição com teste no início 128
3.9.2 Repetição com teste no final 136
3.9.3 Repetição com variável de controle 142
3.9.4 Modularização 146
3.10 Exercícios propostos 153
3.11 Gabarito dos exercícios propostos 155

4 ESTRUTURA DE DADOS 165

4.1 Variáveis compostas homogêneas 167
4.1.1 Vetores 167
4.1.2 Matrizes 172
4.2 Variáveis compostas heterogêneas 183
4.3 Exercícios propostos 197
4.4 Gabarito dos exercícios propostos 199

5 CONSTRUÇÃO E SIMPLIFICAÇÃO DE EXPRESSÕES 208

5.1 Exercícios propostos 221
5.2 Gabarito dos exercícios propostos 222
BIBLIOGRAFIA RECOMENDADA 224

1 APRESENTAÇÃO

O conteúdo principal desta obra é a construção de algoritmos, o qual tratamos a partir do segundo capítulo. Os fundamentos necessários a esta construção estudaremos no primeiro capítulo, para a qual outras habilidades matemáticas amplamente estudadas na Educação Básica, como funções e operadores aritméticos ou relacionais, também são requisitadas. No último capítulo voltamos a trabalhar fundamentos matemáticos, mas agora com o objetivo tornar mais eficiente a construção de algoritmos. A seguir apresentamos uma breve descrição de cada um dos capítulos desta obra.

No capítulo primeiro, *uma introdução ao Cálculo Proposicional*, apresentamos de forma breve os primeiros passos do estudo da Lógica Matemática, as operações essenciais do Cálculo Proposicional a programação: A negação, a conjunção e as disjunções inclusiva e exclusiva, uma vez que estas não foram formalmente tratadas na Educação Básica da maioria dos ingressantes do Ensino Superior nos cursos afins a Ciência da Computação, a quem esta obra se destina. Tratamos também nesse capítulo da negação de conjunções e de disjunções, inclusiva ou exclusiva, uma vez que estas são motivos de equívocos praticados mesmo por quem já teve algum contato inicial com a Lógica Matemática. Tratamos ainda da construção das tabelas verdade, a forma mais completa para a análise de uma fórmula do Cálculo Proposicional, geratriz das famosas tabelas de decisão estudadas na construção de algoritmos. Além disso, as tabelas verdade nos permitem a detecção das tautologias e das contradições, fórmulas que podem ser traduzidas por verdades ou falsidades absolutas, e que por isto não podem ser utilizadas com condições ou parte delas em nossos algoritmos.

O capítulo segundo, *a construção de algoritmos*, podemos dividi-lo em três partes: A estrutura sequencial, as estruturas de seleção e as estruturas de repetição, além dos pré-requisitos a estes temas: as expressões, as condições, os identificadores, a declaração de variáveis e os comandos atribuição, entrada e saída.

As estruturas de seleção são apresentadas de acordo com o grau de complexidade: a seleção simples, a seleção composta e as seleções encadeadas, homogênea e heterogênea, além da seleção de múltipla escolha.

Quanto às estruturas de repetição tratamos dos três casos: A repetição com teste no início, a repetição com teste no final e a repetição com variável de controle. Destaca-se também neste capítulo a comparação entre diversas estruturas, como também o estudo detalhado de cada uma delas.

Finalizando este capítulo apresentamos, de forma breve, a ideia de modularização.

No capítulo terceiro, *estrutura de dados*, continuamos a construir algoritmos, agora organizando as informações. Para tanto estudamos as variáveis compostas homogêneas, conhecidas como vetores e matrizes, como também as variáveis compostas heterogêneas.

Nestes dois últimos capítulos todos os algoritmos construídos são apresentados em Português Estruturado, que nada mais é do que uma pré-linguagem de programação cujos comandos são expressos através da Língua Portuguesa, facilitando, para nós brasileiros, a compreensão da execução do nosso programa. Existem ambientes nos quais os algoritmos podem ser executados e assim definitivamente testados.

Escolhemos o Portugol Studio, software livre, do nosso ponto de vista, um bom ambiente para aprender a programar, destinado aos estudantes em fase inicial da aprendizagem e que falam o idioma Português. Em toda esta obra o que se refere a este software estará destacado com fundo cinza, como este que agora você lê, facilitando a identificação caso o leitor escolha outro ambiente de programação e não deseje estuda-lo.

Caso contrário, recomendamos o download deste software, para melhor exploração dos algoritmos abordados nesta obra ou outros. A escolha deste software se deve não somente ao seu desempenho, mas principalmente ao fato das adaptações necessárias ao algoritmo em Português Estruturado para a sua inserção no Portugol Studio deixa-lo muito próximo ao respectivo programa quando escrito em linguagens comerciais, como a linguagem C, por exemplo.

Devido a essas adaptações, para que o leitor comece a familiarizar-se com o Portugol Studio, alguns dos algoritmos desta obra, desenvolvidos em Português Estruturados são apresentados também convertidos para o Portugol Studio. Escolhemos um problema particular, um clássico no estudo de algoritmos, o de calcular a média entre cinco notas, pois com ele partimos de uma situação bem simples, a de apenas calcular essa média, e avançamos para situações mais complexas, a de tratar uma turma de alunos. Isso faz com que as preocupações com a construção do algoritmo e com a respectiva implementação no Portugol Studio sejam reduzidas pois novos recursos são incluídos aos poucos.

Assim, o leitor perceberá a importância da estrutura do algoritmo e que para implementá-lo, em qualquer ambiente, são necessárias apenas adaptações próprias de cada linguagem, sem mudança alguma

na essência do raciocínio empregado para a resolução do problema em estudo. Destacamos que é este o nosso único objetivo na utilização deste ambiente, não temos a pretensão de tornar nosso leitor um profundo conhecedor deste software.

No último capítulo, o quarto, *construção e simplificação de expressões*, voltamos a estudar, também de forma introdutória, conceitos do Cálculo Proposicional da Lógica Matemática, que nos permites a construção de fórmulas a partir de uma tabela verdade e a simplificação destas por meio da aplicação de equivalências lógicas. Este estudo é muito importante visto que muitas situações práticas são melhores compreendidas quando apresentadas através de uma tabela verdade e a partir desta as fórmulas obtidas são por vezes extensas, e a simplificação delas nos garantirá mais legibilidade ao nosso algoritmo, o que, como veremos, é uma das duas características fundamentais. A outra é a portabilidade, ou seja, a independência do algoritmo em relação a qualquer linguagem de programação, o que o torna fundamental a qualquer uma delas.

2 UMA INTRODUÇÃO AO CÁLCULO PROPOSICIONAL

Definição: Chamamos de *valor verdade* os estados de qualquer situação biestável.

Exemplo: Quente e frio, aceso e apagado, verdadeiro e falso, entre outros.

Trabalharemos sempre com os valores verdade *verdadeiro*, geralmente indicado por V ou por 1, e *falso*, geralmente indicado por F ou por 0, salvo menção contrária.

Definição: *Proposição* (no Cálculo Proposicional) é toda expressão sobre a qual faz sentido estabelecer o seu valor verdade, ou seja, é qualquer expressão na qual se tenha sentido afirmar se seu conteúdo é verdadeiro ou falso.

Exemplo: São proposições:

"A neve é branca"

"3 é um número par"

"4>5"

"2>1".

Para representar uma proposição, usaremos as letras minúsculas p, q, r, acompanhadas ou não de índices inferiores que pertençam ao conjunto dos números inteiros não negativos, ou seja, p_0, q_0, r_0, p_1, q_1, r_1, etc..

Exemplo: Dada a proposição "2 é menor que 3", podemos representá-la por p, ou p_0 (lê-se p zero), ou p_1(lê-se p um), ou mais genericamente, por qualquer p ou q ou r ou ainda por p_i ou q_i ou r_i em que $i \in Z_+$.

Notação: p: 2 é menor que 3 significará p representa a proposição "2 é menor que 3", e a notação é análoga para outras representações.

Definição: Se p é uma proposição que assume o valor verdade verdadeiro, dizemos que o *valor lógico* da proposição p é verdadeiro, o que indicamos por vl(p) = V. Analogamente, se p é uma proposição que assume o valor verdade falso, dizemos que o *valor lógico* da proposição p é falso, o que indicamos por vl(p) = F. É importante observar que dada uma certa proposição, ela assume um dos dois valores verdade, verdadeiro (V), ou falso (F), nunca ambos simultaneamente, e não existe um terceiro valor.

2.1 Operação negação

É uma operação unitária, isto é, que opera uma proposição e fornece como resultado uma segunda proposição. É o exemplo mais simples de uma operação. Se p é uma proposição, indica-se a negação de p por não p, ou por - p, ou por ~ p, ou ainda por p´, entre outras notações existentes. Daremos preferência a primeira delas. Assim, se p representa a proposição "a neve é branca", denotamos por:

p: a neve é branca.

e daí não p representa a proposição "a neve não é branca", e denotamos:

não p: a neve não é branca.

Note que se vl(p) = V, então vl(não p) = F, e se vl(p) = F, então vl(não p) = V.

Podemos descrever uma tabela que resuma como se comporta a operação chamada *negação*. Para tanto seja p uma proposição qualquer:

p	não p
V	F
F	V

A tabela acima nos diz que se vl(p) = V, então vl(não p) = F, e que se vl(p) = F, então vl(não p) = V.

Observe ainda que vl(não não p) = vl(p).

2.2 Operação conjunção

É uma operação binária, isto é, que opera duas proposições e fornece como resultado uma terceira proposição. Essa operação, chamada *conjunção*, evidencia o uso da palavra "e" na linguagem usual e é denotada pelo próprio e, ou por $\wedge$, ou por $\cdot$, ou ainda por &, entre outras notações existentes. Daremos preferência a primeira delas. Assim, se p e q são duas proposições, diremos que p e q representa a conjunção de p com q, e lê-se p e q.

Veja que se afirmamos "vou a Uniso e vou estudar" se você não for a Uniso você será acusado de mentiroso, como também se você não estudar a acusação será a mesma, e ainda mais força terá a acusação se você não for Uniso e tão pouco estudar.

É fácil se convencer que o valor lógico da proposição p e q é verdadeiro somente quando ambos os valores lógicos de p e de q forem também verdadeiros. Assim analisaremos o valor lógico de p e q de acordo com os valores verdade (V ou F) que são atribuídos a p e a q, conforme a tabela abaixo:

p	q	p e q
V	V	V
V	F	F
F	V	F
F	F	F

Note que só quando vl(p) = vl(q) = V, temos vl(p e q) = V, nos demais casos vl(p e q) = F.

Na conjunção p e q, p e q recebem o nome de *conjuntivos*.

2.3 Operação disjunção

Essa operação é representada pelo uso da palavra "ou".

Existem dois tipos de disjunção, a disjunção inclusiva e a disjunção exclusiva. A inclusiva é a mais utilizada, inclusive muitos textos sobre Lógica Matemática não abordam a disjunção exclusiva. Vamos aos seus significados.

2.3.1 Operação disjunção inclusiva

Essa operação, chamada disjunção inclusiva, representa o uso da palavra "ou" e é representada pelo próprio ou, ou por $\vee$, ou por +, entre outras notações. Daremos preferência a primeira delas. Assim, se p representa a proposição "João é muito esperto" e q representa a proposição "João é muito sortudo", a utilização de "p ou q" resulta na proposição::

"João é muito esperto ou João é muito sortudo".

Nesta frase vemos que nada impede que João seja esperto e sortudo, isto é, não se exclui a possibilidade de João ser tanto esperto como sortudo, ou ainda podemos dizer que o uso de "p ou q" no sentido inclusivo faz com que a situação se prevaleça para p, ou para q, ou para ambos p e q.

2.3.2 Operação disjunção exclusiva

Essa operação, chamada disjunção exclusiva, também representa o uso da palavra "ou" e é representada por xou, ou por x$\veebar$. Daremos preferência a primeira delas. Assim, se p representa a proposição "João vai passar esta tarde no clube" e q representa a proposição "João vai passar esta tarde em sua casa", a utilização de "p xou q" resulta na proposição:

"João vai passar esta tarde no clube ou João vai passar esta tarde em sua casa".

É evidente que não se tem o caso em que João vai passar esta tarde no clube e João vai passar esta tarde em sua casa, ao mesmo tempo. Isto é o que caracteriza o uso de "ou" no sentido exclusivo, isto é, vale para p, ou vale para q, mas não é válido para ambos. Veja que em nosso exemplo contextualizado, o fato de ambas proposições assumirem o valor lógico verdadeiro é um absurdo, nem pode ser considerado.

Considerando agora outra contextualização, sendo n um número Natural qualquer é correto afirmar:

"n é um número par ou n é um número ímpar".

e neste caso, ambas proposições assumirem o valor lógico verdadeiro como também ambas proposições assumirem o valor lógico falso são absurdos, nem podem ser considerados.

Em linguagem menos formal é comum destacar o emprego da disjunção exclusiva com a colocação da palavra ou também antes da primeira proposição. Assim nossos exemplos se tornam:

"Ou João vai passar esta tarde no clube ou João vai passar esta tarde em sua casa",

e:

"Ou n é um número par ou n é um número ímpar".

Pelo exposto temos que p ou q representará "p ou q ou ambos" e sua tabela será dada por:

p	q	p ou q
V	V	V
V	F	V
F	V	V
F	F	F

Note que vl(p ou q) = F somente quando vl(p) = vl(q) = F. Nos demais casos vl(p ou q) = V.

Agora, p xou q representará "somente p ou somente q" ou "p ou q e não ambos" e sua tabela será dada por:

p	q	p xou q
V	V	F
V	F	V
F	V	V
F	F	F

Note que vl(p xou q) = F somente quando vl(p) = vl(q). Nos demais casos vl(p xou q) = V.

Na disjunção (inclusiva e exclusiva), p e q são chamados de *disjuntivos*.

Quando cita-se simplesmente *disjunção*, devemos considerar a disjunção inclusiva, e no caso de situações contextualizadas se assim a interpretação permitir.

Até aqui, introduzimos as operações e as representamos por símbolos:

não, e, ou e xou

os quais também são chamados de *conectivos*.

Quando temos apenas uma proposição, digamos p, sabemos que ela pode possuir o valor lógico verdadeiro ou falso, e assim uma tabela para uma única proposição possui apenas duas linhas de valores verdade:

p
V
F

Agora, quando temos duas proposições, digamos p e q, existem quatro possibilidades diferentes de atribuirmos os valores verdade verdadeiro ou falso a estas duas proposições, simultaneamente consideradas. Assim, uma tabela para duas proposições possui quatro linhas de valores verdade:

p	q
V	V
V	F
F	V
F	F

Nada, porém, impede que uma proposição seja constituída por várias proposições distintas. Então quantas possibilidades diferentes existem de atribuirmos os valores verdade verdadeiro ou falso simultaneamente à n proposições?

Observe que para uma proposição existem duas possibilidades distintas ($2 = 2^1$) e para duas proposições existem quatro possibilidades diferentes ($4 = 2^2$).

Uma regra pode ser deduzida disso: para n proposições, teremos 2^n linhas.

Quando n = 3, isto é, para três proposições, digamos p, q e r, temos 2^3 possibilidades apresentadas na seguinte tabela:

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

Para a construção destas oito possibilidades tomamos as primeiras quatro linhas, ou seja, a metade do total, e atribuímos o valor verdade verdadeiro à primeira proposição, no caso p e em seguida construímos as quatro possibilidades diferentes de atribuirmos os valores lógicos verdadeiro ou falso às outras proposições, no caso p e q,

simultaneamente consideradas. Finalizando, copiamos as quatro primeiras linhas já construídas para as outras quatro linhas da tabela, substituindo V por F e F por V. Esta mesma regra pode ser utilizada para um número maior de proposições.

Outras regras existem para esta mesma finalidade, como por exemplo, preencher a primeira coluna colocando na primeira metade das 2^n linhas o valor verdade V e na segunda metade o valor verdade F, e depois repetindo o procedimento nas demais colunas alternando a colocação de V e F em quantidades iguais a metade das quantidades de V ou F utilizados na coluna anterior, até preencher a última coluna, alternando um V e um F. Nesta metodologia, com n = 3, isto é, para três proposições, digamos p, q e r, temos a seguinte tabela:

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

Na Matemática, quando denotamos um número por uma letra, digamos x, esta letra recebe o nome de variável. Da mesma forma, no Cálculo Proposicional as letras minúsculas p, q, r, acompanhadas ou não de índices inferiores que pertençam ao conjunto dos números inteiros não negativos, utilizadas para representar as proposições, são chamadas de *variáveis proposicionais (v. p.)*.

Nosso próximo passo é saber identificar, sem ambiguidade, expressões construídas com o uso dos conectivos. Por exemplo, a expressão p ou q e r pode ser compreendida como (p ou q) e r ou como p ou (q e r), e o que é mais grave, esta ambiguidade pode prejudicar a sua análise, pois sendo vl(p) = V, vl(q) = V e vl(r) = F, podemos ter que vl(p ou q e r) = V ou vl(p ou q e r) = F, dependendo de qual operação executamos primeiro. Para resolvermos este problema, vamos introduzir o conceito de fórmula.

Definição: Definimos como *fórmula* qualquer expressão construída com variáveis proposicionais pela aplicação de um número finito de conectivos, satisfazendo:

1) Toda variável proposicional sozinha é fórmula;

2) Se A e B são fórmulas, então não A, (A e B), (A ou B) ou (A xou B), também são fórmulas;

3) Somente são fórmulas as expressões que são determinadas por meio das condições 1 e 2.

As fórmulas serão denotadas por letras maiúsculas do alfabeto, acompanhadas ou não de índices inferiores que pertençam ao conjunto dos números inteiros não negativos.

Observações:

1) não A não é necessariamente uma fórmula. não A será uma fórmula sempre que A o for. Isto é não A é uma fórmula somente se A é uma fórmula.

2) (A e B), (A ou B) ou (A xou B) não são necessariamente fórmulas. (A e B), (A ou B) ou (A xou B) só serão fórmulas se A e B são fórmulas.

3) Uma fórmula A qualquer assume um dos dois valores verdade, verdadeiro (V), ou falso (F), nunca ambos simultaneamente, e não existe um terceiro valor. Quando assume o valor verdade verdadeiro, dizemos que o *valor lógico* da fórmula A é verdadeiro, o que indicamos por vl(A) = V. Analogamente, se A é uma fórmula que assume o valor verdade falso, dizemos que o *valor lógico* da fórmula A é falso, o que indicamos por vl(A) = F.

4) As tabelas das operações negação (não), conjunção (e), disjunção inclusiva (ou) e disjunção exclusiva (xou), são ainda válidas se substituirmos as proposições p e q por fórmulas A e B quaisquer.

Exemplo: Segundo as condições enunciadas p, não p, r_{23}, (p ou q), não (p e q), ((r_{23} xou p) e (p ou q)), são consideradas fórmulas, mas r_{23}, p ou q, (r_{23} → p) e (p ou q), não p e q, s ou q, A, (B ou C) não são consideradas fórmulas.

Definição: A última operação que realizamos em uma fórmula determina o seu tipo. Assim, sendo A e B fórmulas quaisquer, temos três tipos de fórmula:

Negação: as fórmulas do tipo não A;

Conjunção: as fórmulas do tipo (A e B) e

Disjunção: as fórmulas do tipo (A ou B) ou do tipo (A xou B).

2.4 Parênteses

Principalmente por motivos estéticos expressões podem ser fornecidas sem todos os pares de parênteses necessários a uma fórmula. Para compreendê-la perfeitamente devemos recolocar esses parênteses faltantes observando as duas seguintes regras:

1) Para qualquer conectivo, adotaremos o princípio de associação à esquerda. Por exemplo, para restabelecer os parênteses na fórmula p ou q ou r, fazemos uma associação à esquerda, isto é, como primeiro passo a fórmula torna-se (p ou q) ou r, e em seguida, a próxima associação à esquerda, ou seja, como segundo passo a fórmula torna-se ((p ou q) ou r);

2) Adotaremos uma ordem hierárquica de força entre os conectivos. Os conectivos não, e, ou, xou, estão em ordem crescente de força, isto é, numa restauração de parênteses em uma fórmula, coloca-se parênteses primeiro nos mais fracos e assim sucessivamente até esgotar todos os conectivos. Para restaurar os parênteses na seguinte fórmula:

$$p \text{ xou não } p_1 \text{ ou } p_2 \text{ e } p_3,$$

procedemos assim:

$$\text{primeiro passo: } p \text{ xou não } p_1 \text{ ou } (p_2 \text{ e } p_3);$$

$$\text{segundo passo: } p \text{ xou (não } p_1 \text{ ou } (p_2 \text{ e } p_3));$$

$$\text{terceiro passo: (p xou (não } p_1 \text{ ou } (p_2 \text{ e } p_3))).$$

Observamos que nas fórmulas do tipo conjunção ou disjunção o par de parênteses externos podem ser omitidos, sem prejuízo a compreensão das mesmas.

2.5 Tabela verdade

Podemos dizer que toda fórmula A para cada atribuição de valores verdade feita às variáveis proposicionais de A, podemos calcular o valor lógico correspondente a fórmula A. Este cálculo pode ser exposto por meio de uma tabela chamada *tabela verdade*.

Para se construir a tabela verdade de uma fórmula A qualquer, devemos seguir os seguintes passos:

1º passo: Determine todas as subfórmulas de A, isto é, todas as partes de A que são fórmulas;

2º passo: Escreva na primeira linha da tabela todas as subfórmulas de A, começando com as variáveis proposicionais e, em seguida, as demais subfórmulas, sendo que a última será a própria fórmula A, obedecendo a regra de que uma subfórmula é colocada nesta primeira linha da tabela somente depois que todas as suas subfórmulas já foram colocadas;

3º passo: Complete as colunas das variáveis proposicionais com todas as possibilidades distintas de atribuirmos os valores lógicos verdadeiro (V) e falso (F) simultaneamente à estas variáveis, lembrando que se temos n variáveis proposicionais existem 2^n atribuições distintas;

4º passo: Complete as demais colunas, na sequência em que você escreveu as subfórmulas, usando as tabelas das operações.

Vamos agora ver em alguns exemplos como se constrói tal tabela.

Exemplo: Seja A a fórmula:

((p e q) xou (p ou q)).

Suas subfórmulas são: p, q, (p e q), (p ou q) e ((p e q) xou (p ou q)).

Sua tabela verdade é:

p	q	(p e q)	(p ou q)	((p e q) xou (p ou q))
V	V	V	V	F
V	F	F	V	V
F	V	F	V	V
F	F	F	F	F

Exemplo: Construa a tabela verdade para a fórmula:

((não p ou q) xou (não q e p)).

Suas subfórmulas são: p, q, não p, não q, (não p ou q), (não q e p) e ((não p ou q) xou (não q e p)).

Sua tabela verdade é:

p	q	não p	não q	(não p ou q)	(não q e p)	((não p ou q) xou (não q e p))
V	V	F	F	V	F	V
V	F	F	V	F	V	V
F	V	V	F	V	F	V
F	F	V	V	V	F	V

Exemplo: Construa a tabela verdade para a fórmula:

((p ou q) e r).

Suas subfórmulas são: p, q, r, (p ou q) e ((p ou q) e r).

Sua tabela verdade é:

p	q	r	(p ou q)	((p ou q) e r)
V	V	V	V	V
V	V	F	V	F
V	F	V	V	V
V	F	F	V	F
F	F	F	F	F
F	F	V	F	F
F	V	F	V	F
F	V	V	V	V

Uma fórmula A recebe o nome de *tautologia* se ela assume o valor lógico verdadeiro (V) para todas as atribuições de valores verdade feitas às suas variáveis proposicionais. É o caso da fórmula ((não p ou q) xou (não q e p)). Outro exemplo de tautologia é a fórmula (p ou não p), cuja tabela verdade é:

p	não p	(p ou não p)
V	F	**V**
F	V	**V**

Observe que se B é uma fórmula qualquer então toda fórmula do tipo (B ou não B) é uma tautologia.

Por outro lado, uma fórmula A recebe o nome de *contradição* se ela assume o valor lógico falso (F) para todas as atribuições de valores verdade feitas às suas variáveis proposicionais. Assim, a fórmula não ((não p ou q) xou (não q e p)) e a fórmula não (p ou não p) são exemplos de contradições. Outro exemplo de contradição é a fórmula (p e não p), cuja tabela verdade é:

p	não p	(p e não p)
V	F	**F**
F	V	**F**

Observe que, se B é uma fórmula qualquer, então toda fórmula do tipo (B e não B) é uma contradição.

Finalmente, as fórmulas que não são contradições, nem tautologias, são chamadas *contingências*, as quais são particularmente interessantes a programação de computadores, na composição de condições.

Exemplo: Seja A a fórmula:

((p ou q) e (q xou p)).

Suas subfórmulas são: p, q, (p ou q), (p xou q) e ((p ou q) e (p xou q)).

Sua tabela verdade é:

p	q	(p ou q)	(p xou q)	((p ou q) e (p xou q))
V	V	V	F	**F**
V	F	V	V	**V**
F	V	V	V	**V**
F	F	F	F	**V**

Logo, a fórmula A é uma contingência.

Observe que tanto as tautologias como as contradições não podem ser utilizadas como condições a serem testadas, visto que o resultado já é conhecido, sempre verdadeiro ou sempre falso, e por esse motivo não figuram em muitas das aplicações da Lógica Matemática, em particular, não constituem condições ou parte delas na construção de algoritmos, o que começaremos a estudar em nosso próximo capítulo.

Antes de começarmos a estudar a construção de algoritmos, veremos situações de negação nas quais muitas vezes a compreensão é incorreta, a negação de uma conjunção e a negação das disjunções.

2.6 A negação de uma conjunção

Veja que a negação de uma conjunção, digamos a negação de p e q é não (p e q) e não não p e q. Neste último caso a negação atua somente sobre a proposição p e não sobre a conjunção de p com q.

A partir da tabela de p e q, cuja tabela é:

p	q	p e q
V	V	V
V	F	F
F	V	F
F	F	F

obtemos a tabela de não (p e q), adicionando uma coluna:

p	q	p e q	não (p e q)
V	V	V	F
V	F	F	V
F	V	F	V
F	F	F	V

e lembrando que a operação negação troca o valor lógico de uma proposição, e assim onde vl(p e q) = V temos vl(não (p e q)) = F e onde vl(p e q) = F temos vl(não (p e q)) = V.

Considere o exemplo “vou a Uniso e vou estudar” a sua negação é algo do tipo “não é verdade que vou a Uniso e vou estudar” ou “é falso que vou a Uniso e vou estudar” o que é verdade desde que você deixe de fazer uma das coisas, ir a Uniso ou estudar.

2.7 A negação de uma disjunção inclusiva

Veja que a negação de uma disjunção inclusiva, digamos a negação de p ou q é não (p ou q) e não não p ou q. Neste último caso a negação atua somente sobre a proposição p e não sobre a disjunção inclusiva de p com q.

A partir da tabela de p ou q:

p	q	p ou q
V	V	V
V	F	V
F	V	V
F	F	F

obtemos a tabela de não (p ou q) basta adicionando uma coluna:

p	q	p ou q	não (p ou q)
V	V	V	F
V	F	V	F
F	V	V	F
F	F	F	V

e lembrando que a operação negação troca o valor lógico de uma proposição, e assim onde vl(p ou q) = V temos vl(não (p ou q)) = F e onde vl(p ou q) = F temos vl(não (p ou q)) = V.

Negando a expressão “vou a Uniso ou vou estudar” obtemos algo do tipo “não é verdade que vou a Uniso ou vou estudar” ou “é falso que vou a Uniso ou vou estudar” o que é verdade desde não faça nenhuma das coisas, ir a Uniso e estudar.

Vamos agora obter expressões equivalentes para a negação de disjunção e para a negação de conjunção.

A tabela de não (p ou q) é:

p	q	p ou q	não (p ou q)
V	V	V	F
V	F	V	F
F	V	V	F
F	F	F	V

e a tabela de não p e não q é:

p	q	não p	não q	não p e não q
V	V	F	F	F
V	F	F	V	F
F	V	V	F	F
F	F	V	V	V

e assim as expressões não (p ou q) e não p e não q representam exatamente a mesma situação, aquela onde o resultado é verdadeiro somente quanto as duas proposições iniciais, no caso p e q, são falsas. Esta é a primeira lei de De Morgan.

A tabela de não (p e q) é:

p	q	não (p e q)
V	V	F
V	F	V
F	V	V
F	F	V

e a tabela de não p ou não q é:

p	q	não p ou não q
V	V	F
V	F	V
F	V	V
F	F	V

e assim as expressões não (p e q) e não p ou não q representam exatamente a mesma situação, aquela onde o resultado é falso somente quanto as duas proposições iniciais, no caso p e q, são verdadeiras. Esta é a segunda lei de De Morgan.

Estas duas leis de De Morgan nos deixam bem claro que as famosas placas:

"não jogue lixo e entulho"

como também as famosas propagandas:

"sem adição de açúcares e conservantes"

não significam o que desejam representar. A grafia correta é "não jogue lixo ou entulho" e "sem adição de açúcares ou conservantes".

Na importante tarefa dos engenheiros ou dos programadores estes equívocos não podem ocorrer. Por exemplo, ao descrever o funcionamento de um conjunto motor-bomba ao escrever ou programar:

"se é falso que a lâmpada de temperatura está acesa e o reservatório está cheio então mantenha o sistema ligado",

o sistema será mantido acionado mesmo quando a lâmpada de temperatura estiver acesa enquanto o reservatório não encher e também quando a lâmpada de temperatura não estiver acesa mas o reservatório já estiver cheio, e desta forma o prejuízo não será apenas na escrita. O correto é:

"se é falso que a lâmpada de temperatura está acesa ou o reservatório está cheio então mantenha o sistema ligado",

que nos diz que o sistema será mantido acionado somente enquanto a lâmpada de temperatura não estiver acesa enquanto o reservatório não estiver cheio.

2.8 A negação de uma disjunção exclusiva

Veja que a negação de uma disjunção exclusiva, digamos a negação de p xou q é não (p xou q) e não não p xou q. Neste último caso a negação atua somente sobre a proposição p e não sobre a disjunção exclusiva de p com q.

Vimos que p xou q representa "somente p ou somente q" e sua tabela é:

p	q	p xou q
V	V	F
V	F	V
F	V	V
F	F	F

Note que vl(p xou q) = F somente quando vl(p) = vl(q). Nos demais casos vl(p xou q) = V.

Assim, a tabela da negação da disjunção exclusiva de p com q, ou seja, de não (p xou q) é:

p	q	não (p xou q)
V	V	V
V	F	F
F	V	F
F	F	V

Note que vl(não (p xou q)) = V somente quando vl(p) = vl(q). Nos demais casos vl(p xou q) = F.

A Lógica Matemática sugere que para melhor compreensão de uma negação de uma disjunção exclusiva, digamos não (p xou q), seja traduzida para p se e somente se q, concordando com a tabela acima. No entanto, em problemas contextualizados devemos tomar cuidado.

Voltando ao exemplo:

"João vai passar esta tarde no clube ou João vai passar esta tarde em sua casa",

para o qual sabemos que o fato de ambas proposições assumirem o valor lógico verdadeiro é um absurdo.

A negação desta disjunção exclusiva é:

"não é verdade que João vai passar esta tarde no clube ou João vai passar esta tarde em sua casa",

a qual, seguindo a sugestão da Lógica Matemática, traduzimos para:

"João vai passar esta tarde no clube se e somente se João vai passar esta tarde em sua casa",

e temos que esta negação assume o valor lógico verdadeiro quando ambas proposições assumirem o valor lógico verdadeiro, o que já sabíamos ser um absurdo.

Em nosso outro exemplo já visto sobre a disjunção exclusiva, sendo n um número Natural qualquer é correto afirmar:

"n é um número par ou n é um número ímpar",

no qual, ambas proposições assumirem o valor lógico verdadeiro como também ambas proposições assumirem o valor lógico falso são absurdos.

A negação desta disjunção exclusiva é:

"é falso que n é um número par ou n é um número ímpar",

a qual, seguindo a sugestão da Lógica Matemática, traduzimos para:

"n é um número par se e somente se n é um número ímpar",

e agora temos que esta negação assume o valor lógico verdadeiro em ambas situações já sabidamente absurdas.

2.9 Exercícios propostos

1) Sejam as proposições:

p: João é bom aluno;

q: João é estudioso.

i) Escreva na linguagem usual as seguintes sentenças:

a) p ou q

b) p e q

c) p xou não q

d) não p e não q

e) não não p

f) não (não p e não q)

ii) Escreva na linguagem simbólica as seguintes sentenças:

a) João é bom aluno mas não é estudioso.

b) João é estudioso ou João é bom aluno.

c) Não é verdade que João é bom aluno e estudioso.

d) É falso que João não é estudioso.

2) Reescreva as sentenças seguintes procurando esclarecer o seu significado:

a) Não é verdade que João é bom aluno e estudioso.

b) É falso que João não é bom aluno e não é estudioso.

c) Não é verdade que João é bom aluno ou estudioso.

d) Não é verdade que João não é bom aluno ou é estudioso.

e) É falso que João não é estudioso.

3) Sabendo que vl(p) = V e vl(q) = F, determinar o valor lógico de cada uma das proposições:

a) p e não q

b) p ou não q

c) não p e q

d) não p e não q

e) não p xou não q

f) p e (não p ou q)

4) Classifique as fórmulas dadas abaixo como tautologia, contradição ou contingência:

a) ((p xou q) e (não p e não q))

b) (p e não (p xou q))

c) não ((p xou q) e (não p e não q))

2.10 Gabarito dos exercícios propostos

1)

i) a) João é bom aluno ou é estudioso.

b) João é bom aluno e é estudioso.

c) João é bom aluno ou não é estudioso.

d) João não é bom aluno e não é estudioso.

e) Não é verdade que João não é bom aluno.

f) Não é verdade que João não é bom aluno e não é estudioso.

ii) a) p e não q

b) q ou p

c) não (p e q)

d) não (não q)

---

2) Nesta resolução foi considerado que reprovado é o oposto de aprovado, não existindo uma terceira opção.

a) João não é bom aluno ou não é estudioso.

b) João é bom aluno ou é estudioso.

c) João não é bom aluno e não é estudioso.

d) João é bom aluno e não é estudioso.

e) João é estudioso.

---

3) a) vl (p e não q) = V

b) vl (p ou não q) = V

c) vl(não p e q) = F

d) vl (não p e não q) = F

e) vl(não p xou não q) = V

f) vl(p e (não p ou q)) = F

---

4)

a)

p	q	⋯	((p xou q) e (não p e não q))
V	V		F
V	F		F
F	V		F
F	F		F

Contradição

b)

p	q	⋯	(p e não (p xou q))
V	V		V
V	F		F
F	V		F
F	F		F

Contingência

c)

p	q	⋯	não ((p xou q) e (não p e não q))
V	V		V
V	F		V
F	V		V
F	F		V

Tautologia

3 A CONSTRUÇÃO DE ALGORITMOS

3.1 Expressões e condições

A construção de algoritmo não está necessariamente relacionada a programação de computadores, embora seja esse o eixo norteador do nosso estudo. De forma geral, um algoritmo é uma sequência de passos que visam atingir um objetivo bem definido, e assim a algoritmização está presente em todo planejamento. Também, o ato de programar não é exclusivo da computação, no entanto, este capítulo foi concebido como os primeiros passos no estudo da programação de computadores.

Ao construir um algoritmo organizamos a execução de ações através de estruturas de controle, as quais recebem o nome de *estruturas de controle de execução* ou *estruturas de controle de fluxo*. Essas estruturas estão divididas em três tipos básicos:

- Estrutura sequencial: conjunto de ações que devem ser executadas, todas passo a passo, uma após a outra, compondo uma ordem sequencial de execução;

- Estrutura de seleção: é uma estrutura que seleciona conforme o resultado de uma condição, qual conjunto de ações deve ser executado ou qual conjunto de ações deve ser evitado;

- Estrutura de repetição: é uma estrutura que repete um mesmo trecho do algoritmo, fazendo com que este passe pelo mesmo trecho diversas vezes, mediante a verificação de uma condição.

As estruturas de seleção e de repetição estão vinculadas a verificação de condições, ou seja, de expressões que após executadas produzem como resultado o valor-lógico verdadeiro (V) ou falso (F). Estas expressões não envolvem apenas as fórmulas do Cálculo Proposicional,

as quais já conhecemos bem, mas elas envolvem também os operadores aritméticos e os relacionais. Por isso faremos um breve estudo sobre estes operadores.

Os operadores aritméticos são as operações básicas da matemática:

+	-	adição,
–	-	subtração,
*	-	multiplicação,
/	-	divisão,
**	-	potenciação,
//	-	radiciação,

e as não convencionais:

mod	-	resto da divisão,
div	-	quociente da divisão inteira.

Exemplos:

1) 17 div 2 resulta 8.

2) 17 mod 2 resulta 1.

3) 21 div 3 resulta 7.

4) 21 mod 3 resulta 0.

Com esses operadores e com as funções matemáticas:

sen(x)	-	seno de x,
cos(x)	-	cosseno de x,
tg(x)	-	tangente de x,
arcsen(x)	-	arco cujo seno é x,
arccos(x)	-	arco cujo cosseno é x,
arctg(x)	-	arco cuja tangente é x,
abs(x)	-	valor absoluto (módulo) de x,
int(x)	-	a parte inteira de um número fracionário,
frac(x)	-	a parte fracionária de x,
ard(x)	-	transforma, por arredondamento, um número fracionário em inteiro,
sinal(x)	-	fornece o valor – 1, + 1 ou zero conforme o valor de x seja negativo, positivo ou nulo,

onde, nestas funções como na Matemática, x, que recebe o nome de variável, é um número, representa um número, expressão aritmética ou outra função matemática, construímos as expressões aritméticas.

Observamos que as variáveis nas expressões aritméticas também podem ser representadas por outras letras minúsculas.

Exemplos:

1) int(44.125) resulta 44.

2) frac(44.125) resulta 83872.

3) ard(44.125) resulta 44.

4) ard(44.725) resulta 45.

5) abs(– 5) resulta 5.

6) sinal(– 4.6) resulta – 1.

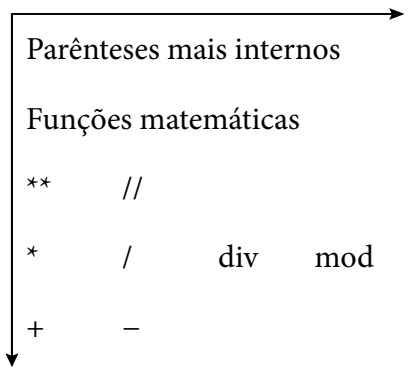
7) abs(x – 10).

8) int(y/5).

9) abs(int(y/10)).

Observe que nos três últimos exemplos não obtivemos um valor numérico como resultado, pois as expressões envolvem as variáveis x ou y, cujos valores são desconhecidos.

A prioridade em expressões aritméticas é estipulada, de cima para baixo e da esquerda para a direita, pelo esquema:



e para alterar a prioridade devemos utilizar parênteses.

Exemplos:

1) 5 + 9 + 7 + 10 / 4 = 5 + 9 + 7 + 2.5 = 23.5.

2) 10 – 4 * 3 / 6 – 2 ** 3 = 10 – 4 * 3 / 6 – 8 = 10 – 12 / 6 – 8 = 10 – 2 – 8 = 0.

3) 3 ** 3 – 4 / 2 + abs(5 – 3 * 5) / 2 = 3 ** 3 – 4 / 2 + abs(5 – 15) / 2 = 3 ** 3 – 4 / 2 + abs(– 10) / 2 = 3 ** 3 – 4 / 2 + 10 / 2 = 27 – 4 / 2 + 10 / 2 = 27 – 2 + 5 = 30.

Os operadores relacionais são aqueles utilizados para realizar comparações entre dois números, formando uma relação, variáveis ou expressões aritméticas, obtendo como resultado um valor-lógico.

Os operadores relacionais são:

=	-	igual a,
<>	-	diferente de,
>	-	maior que,
>=	-	maior igual a,
<	-	menor que,
<=	-	menor igual a.

Quando um operador relacional compara expressões aritméticas, devemos primeiro executar as expressões aritméticas.

Exemplo: Determine o valor lógico das seguintes expressões:

a) vl(3 * 4 = 24 / 3) = vl(12 = 8) = F.

b) vl(23 mod 4 < 19 mod 6) = vl(3 < 1) = F.

c) vl(3 * 5 div 4 <= 3 ** 3/0.5) = vl(15 div 4 <= 27/0.5) = vl(3 <= 54) = V.

d) vl(3 + 8 mod 5 >= 3 * 6 – 15) = vl(3 + 3 >= 18 – 15) = vl(6 >= 3) = V.

Na construção de algoritmos usamos também operações lógicas negação, conjunção, disjunção inclusiva e disjunção exclusiva, ou simplesmente os conectivos não, e, ou e xou, cujas tabelas, conforme o estudo no capítulo *Uma introdução ao Cálculo Proposicional*, são respectivamente:

p	não p
V	F
F	V

p	q	p e q
V	V	V
V	F	F
F	V	F
F	F	F

p	q	p ou q
V	V	V
V	F	V
F	V	V
F	F	F

p	q	p xou q
V	V	F
V	F	V
F	V	V
F	F	F

para proposições quaisquer p e q.

Entre todos operadores a prioridade se dá de cima para baixo, conforme o esquema:

Parênteses mais internos

Funções matemáticas

Operadores aritméticos

Operadores racionais e

Operadores lógicos,

e para alterar a prioridade devemos utilizar parênteses.

Exemplo: Determine o valor lógico das expressões abaixo:

a) vl(não 3 ** 3 < 4 ** 2 ou abs(int(15 / (– 2))) < 10) =

vl(não 27 < 16 ou abs(int(– 7.5)) < 10) =

vl(não F ou abs(– 7) < 10) =

vl(V ou 7 < 10) =

vl(V ou V) =

V

b) vl(não (5 <> 12 / 2 ou verdadeiro e 2 – 5 > 5 – 2 xou falso)) =

vl(não (5 <> 6 ou V e – 3 > 3 xou F)) =

vl(não (V ou V e F xou F)) =

vl(não (F ou F xou F)) =

vl(não (F xou F)) =

vl(não (F)) =

V

c) vl(int(20 / 2) <> 4 + 2 xou 2 + 3 * 5 / 3 mod 5 > 0) =

vl(int(10) <> 6 xou 2 + 15 / 3 mod 5 > 0) =

vl(10 <> 6 xou 2 + 5 mod 5 > 0) =

vl(V xou 2 + 0 > 0) =

vl(V xou 2 > 0) =

vl(V xou V) =

F

Todas as expressões que envolvem os operadores relacionais ou lógicos recebem o nome, na teoria dos algoritmos, de *condições*. Será usual a construção de uma *tabela de decisão*, ou seja, uma parte de tabela verdade da expressão, formada apenas pela(s) linha(s) possível(is) àquela expressão.

Exemplo: A tabela de decisão da expressão 10 <> 6 xou 2 + 5 mod 5 > 0 é:

10 <> 6	2 + 5 mod 5 > 0	10 <> 6 xou 2 + 5 mod 5 > 0
V	V	F

e a tabela de decisão da expressão x <> 6 xou 2 + 5 mod 5 > 0 terá duas linhas:

x	x <> 6	2 + 5 mod5 > 0	x <> 6 xou 2 + 5 mod 5 > 0
6	F	V	V
outros	V	V	F

e mais uma coluna, pois na expressão temos a variável x, e nesta nova coluna indicaremos os valores possíveis desta variável.

Aqui encerramos o nosso estudo sobre as expressões, e sobre este conteúdo não apresentaremos exercícios propostos, visto que este capítulo possui como foco principal a lógica na construção de algoritmos, no entanto, se o leitor não se sentir suficientemente dotado de habilidades para o trato destas expressões, sugerimos a busca de textos complementares, alguns citados na bibliografia desta obra.

Mais adiante relacionaremos ações a condições, nas estruturas de seleção e de repetição. Daí, uma tabela de decisão objetivará basicamente relacionar as ações que dependem de alguma(s) condição(ões) com essa(s) própria(s) condição(ões), a fim de esclarecer e visualizar facilmente quais valores o conjunto de condições deve assumir para que se efetue, ou não, sua respectiva ação.

Antes de iniciarmos os estudos para a construção de algoritmos, apresentamos duas características fundamentais de qualquer algoritmo:

- a legibilidade, ou seja, o todo algoritmo deve ser compreendido por qualquer outra pessoa, com habilidades semelhantes às de quem o construiu, ou seja, a legibilidade garante a clareza com que sua lógica está exposta, e,

- a portabilidade, que garante a independência do algoritmo com qualquer linguagem de programação. Por isso, os algoritmos são construídos em uma pseudo-linguagem denominada Português Estruturado. Assim, além de se preocupar apenas com a lógica do problema, pode ser convertido com facilidade para qualquer linguagem de programação.

A pseudo-linguagem citada no parágrafo acima recebe o nome de Português Estruturado, devido ao idioma escolhido e a formação de suas estruturas, as quais visando aumentar a legibilidade são definidas utilizando-se deslocamentos à direita que facilitarão a percepção dos respectivos início e fim, como também das suas principais partes, como veremos em breve.

Alguns passos devem ser seguidos para a construção de um algoritmo:

i) ler atentamente o enunciado,

ii) retirar do enunciado a relação das entradas de dados, ou seja, as informações fornecidas pelo enunciado,

iii) retirar do enunciado a relação das saídas de dados, ou seja, as informações solicitadas no problema, ou ainda, qual a resposta desejada,

iv) determinar o que deve ser feito para transformar as entradas determinadas nas saídas desejadas, ou seja, construir, propriamente dito, o algoritmo,

v) reescrever o determinado no passo anterior em Português Estruturado, e,

vi) executar o *teste de mesa*, ou seja, executar todas as ações descritas seguindo o fluxo de execução estabelecido, verificando se os resultados obtidos correspondem ao esperado quando da montagem do algoritmo, detectando então algum possível erro no desenvolvimento deste. Muitas vezes, a tabela de decisão de uma estrutura é suficiente para testar o algoritmo e assim o teste de mesa fica reduzido a essa tabela de decisão.

Na construção de um algoritmo, devemos sempre levar em conta, que muitas vezes é interessante dividir o problema em partes e então tratamos cada uma dessas partes, construindo o algoritmo de cada uma delas, para depois unifica-las em um todo, construindo o algoritmo desejado. Algumas vezes, o problema é composto somente por partes disjuntas, e daí este procedimento é trivial. Outras vezes, algumas partes do problema não são disjuntas, ou seja, a execução de uma parte está incorporada por outra(s), e daí a divisão deve ser feita levando em conta que precisamos começar a construção pela parte mais interna do problema, e depois, passamos para a parte que contém a já construída, e assim sucessivamente, até obter o algoritmo desejado.

Exemplo: Um dos problemas que será bastante explorado neste texto, consiste em construir um algoritmo que calcule a média aritmética entre cinco notas quaisquer de cinco alunos, fornecidas pelo usuário, e emita uma avaliação quanto à situação do aluno, sendo que a aprovação é obtida atingindo-se média superior ou igual a seis. Daí podemos primeiro pensar em um algoritmo que calcula a média entre cinco notas de um aluno, depois, tratar da análise da situação do aluno, e finalmente tratar do problema todo, ou seja, da turma com cinco alunos.

Muitas vezes realizamos a divisão em partes acima citada mentalmente ou na própria construção do algoritmo completo, mas sem dúvida alguma, formalizando ou não a divisão proposta, esta será uma prática presente em nossas construções de algoritmos.

Um algoritmo tem como objeto de cálculo as informações. A quantidade de informações a operar, cada vez maior, é o que tornou necessário a existência dos computadores, e uma ciência que busca constantemente o aprimoramento destas máquinas, a Ciência da Computação, a qual divide as informações em quatro *tipos primitivos*:

1) inteiro: toda e qualquer informação numérica que pertença ao conjunto dos números inteiros.

2) real: toda e qualquer informação numérica que pertença ao conjunto dos números reais.

3) caractere: toda e qualquer informação composta por um conjunto de caracteres alfanuméricos ou especiais, por exemplo, #, $, %, &, *, ?, ~, <, >, !, @, etc..

4) lógico: toda e qualquer informação biestável, isto é, que pode apenas assumir duas situações.

5) cadeia: toda e qualquer informação formada por uma cadeia de caracteres.

Uma informação pode sofrer ou não variação durante o decorrer do tempo e assim definimos, como *constantes* a informação que não sofre nenhuma variação no decorrer do tempo e como *variável* a informação que tem a possibilidade de ser alterada em algum instante no decorrer do tempo.

Para diferenciar as informações constantes do tipo caractere ou do tipo cadeia dos outros tipos de informação, geralmente elas são delimitá-las por um par de aspas e convencionaremos, para facilitar o raciocínio, que as informações de tipo lógico são sempre verdadeiras (verdadeiro ou V) ou falsas (falso ou F), embora toda a Ciência da Computação esteja estruturada na Álgebra Booleana Mínima, a álgebra dos zeros e uns.

Exemplo: 10, "Não fume", – 0.5, falso e "X%21" são informações constantes, do tipo, inteiro, cadeia, real, lógico e caractere, respectivamente. As informações cotação do dólar e índice de inflação são variáveis.

3.2 Identificadores, declaração de variáveis e comandos atribuição, entrada e saída

Para utilizarmos uma informação variável não podemos representá-la por números, palavras, V ou F, já que não são constantes e assim necessitaremos criar *identificadores*, ou seja, nomes das informações de caráter variável, e para tanto devemos obedecer algumas regras:

- devem começar por um caractere alfabético,
- podem ser seguidos por mais caracteres alfabéticos ou numéricos,
- não é permitido o uso de caracteres especiais,
- os caracteres alfabéticos devem obrigatoriamente ser escritos em maiúsculo.

Exemplo: ALFA, X, NOTA, MÉDIA, TEMP e CON, são considerados identificadores válidos e 5X, A(1), A − B, A +, e B −, não são considerados válidos.

Observamos que é aceito a colocação de acentos nas letras, em concordância com a Língua Portuguesa, como também a letra C acompanhada da cedilha (Ç).

Os identificadores devem ser autoexplicativos, ou seja, devem permitir a identificação imediata do seu conteúdo. Por exemplo, ao calcular a média entre cinco notas de um aluno devemos criar o identificador MÉDIA, mas não o identificador X, pois este último dificultaria a recordação de que nessa variável há o valor de uma média das notas.

No ambiente computacional, as informações variáveis são guardadas em dispositivos eletrônicos chamados de memória. Assim, ao iniciarmos um algoritmo, em primeiro lugar devemos informar as variáveis que utilizaremos e o tipo de cada uma delas, pois em um programa computacional isso promoverá a reserva de uma determinada quantidade de memória necessária para gravar os valores que cada uma dessas variáveis venham a assumir durante a execução do programa. Portanto, um algoritmo começa com a *declaração de variáveis*.

Uma comparação interessante é a com um armário sapateiro. A memória do computador é o armário, cada colmeia desse armário é uma variável, cada sapato a ser colocado nesse armário é a informação e para que não seja necessário abrir várias colmeias para encontrar um determinado sapato, colocamos etiquetas em cada colmeia, sendo então estas etiquetas os identificadores das variáveis. Atente-se que para colocar um sapato em uma determinada colmeia já ocupada por outro sapato, o que aí está deve ser retirado dessa colmeia, e assim, guardado em outra colmeia ou descartado.

Para declarar as variáveis que utilizaremos em nosso algoritmo seguimos a seguinte regra sintática:

tipo primitivo**:** lista de variáveis**;**

onde, relembrando**,** os tipos primitivos são inteiro, real, caractere, lógico e cadeia, e a lista de variáveis deve conter somente variáveis criadas segundo os critérios de formação de identificadores vistos anteriormente, separadas por vírgulas, e ao final de uma declaração colocamos ponto e vírgula.

Exemplo: Ao escrevermos:

inteiro: CON;

cadeia: NOME, ENDEREÇO;

real: DÓLAR, INFL;

lógico: RESPOSTA;

estamos declarando que usaremos as variávies CON do tipo inteiro, NOME e ENDEREÇO do tipo cadeia, DÓLAR e INFL do tipo real e RESPOSTA do tipo lógico.

Não devemos permitir que mais de uma variável possuam o mesmo identificador, visto que ficaríamos sem saber que variável utilizar. Em nossa analogia, seria o mesmo que etiquetar duas ou mais colmeias da mesma forma.

Lembramos ainda que em nosso sapateiro, nas colmeias destinadas a chinelo não podemos colocar botas, pois elas não cabem lá. Da mesma forma, só podemos guardar informações em variáveis do mesmo tipo primitivo.

Como já mencionamos uma variável assume valores durante a execução do programa. Assim, também no algoritmo isto acontece, através de duas maneiras: do comando de atribuição ou do comando de entrada.

O *comando de atribuição* permite fornecer um valor a uma certa variável, onde o tipo dessa informação a ser atribuída deve ser compatível com o tipo da variável. O comando de atribuição é análogo ao ato de colocar um sapato em uma colmeia. Ao final de uma atribuição devemos colocar ponto e vírgula.

Exemplo: Ao escrevermos:

```
lógico: RESPOSTA;
inteiro: CON;
RESPOSTA := verdadeiro;
CON := 0;
```

estamos informando que usaremos as variávies CON do tipo inteiro, e RESPOSTA do tipo lógico e atribuindo à variável identificada com RESPOSTA o valor verdadeiro e a variável identificada como CON o valor nulo.

O símbolo := pode ser substituído pelo símbolo ←.

É possível ainda realizar a atribuição por meio de uma expressão aritmética ou lógica. No entanto, devemos tomar o cuidado que estas expressões sejam resolvidas em primeiro lugar, ou seja, antes da atribuição, para que depois o resultado possa ser armazenado na variável.

Exemplo: Ao escrevermos:

```
lógico: RESPOSTA;
inteiro: CON, SOMA;
RESPOSTA := verdadeiro;
CON := 0;
SOMA := 10 + CON;
```

tomamos o cuidado de atribuir antes o valor nulo à variável CON para depois utilizá-la na atribuição de valor à variável SOMA, o que foi realizado através de uma expressão aritmética.

O *comando de entrada* permite ao usuário do programa fornecer qualquer tipo de informação a variáveis, desde que do mesmo tipo primitivo da variável. Assim realiza-se a entrada de informações ou entrada de dados para que o programa possa processá-las. Este comando possui a regra sintática:

leia ( );

ou

receba ( );

sendo que entre os parênteses colocamos a lista de variáveis às quais desejamos atribuir valores, separadas por vírgulas.

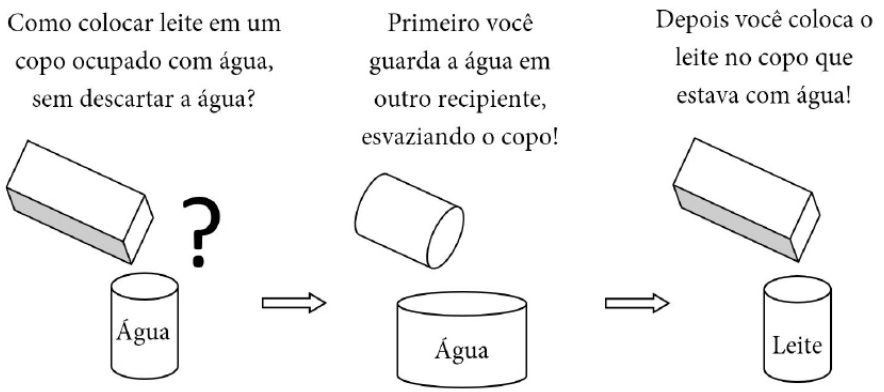
Exemplo: Ao escrevermos:

```
lógico: RESPOSTA;
inteiro: CON, SOMA;
RESPOSTA := verdadeiro;
leia (CON);
SOMA := 10 + CON;
```

o usuário poderá atribuir qualquer valor inteiro à variável CON.

É muito importante frisar que quando uma variável assume um valor fixo, através do comando de atribuição ou através do comando de entrada, um possível valor até então assumido por essa variável é descartado, e a partir daí sempre que figurar essa variável será processado o valor assumido por ela, até que o seu valor seja modificado por alguma outra ação.

Cabe muito bem aqui o exemplo da utilização de um copo. Se nosso copo está com água, para utilizarmos esse copo para armazenar leite, a água será descartada ou guardada em outro copo.



Após o processamento das informações o computador necessita nos mostrar o resultado esperado. No algoritmo isto é realizado através da *saída de dados*. Assim como havia um comando para atribuir um valor à determinada variável, temos também um comando que realiza a saída de dados. Este comando possui a regra sintática:

mostre ( );

ou

escreva ( );

ou ainda,

exiba ( );

sendo que entre os parênteses colocamos a lista de variáveis às quais desejamos atribuir valores, separadas por vírgulas.

Exemplo: Ao escrevermos:

```
lógico: RESPOSTA;
inteiro: CON, SOMA;
RESPOSTA := verdadeiro;
leia (CON);
SOMA := 10 + CON;
escreva (CON, SOMA, RESPOSTA);
```

se, por exemplo, à variável CON o usuário atribuir o valor 5, teremos como resposta, ou seja, a saída será:

```
5
15
verdadeiro.
```

Antes de realizarmos o estudo estrutural dos algoritmos definimos *blocos*, como um conjunto de ações com uma função definida.

Observe que, neste caso, um algoritmo pode ser visto como um bloco. Ele serve também para definir os limites nos quais as variáveis declaradas em seu interior são conhecidas.

Para delimitar um bloco, utilizaremos os delimitadores: início e fim. A regra sintática é:

início {início do bloco ou algoritmo}
⋮
⋮{declaração de variáveis}
⋮
⋮{sequência de ações}
⋮
fim. {final do bloco ou algoritmo}

na qual a declaração de variáveis e a sequência de ações devem estar deslocados à direita em relação à palavra início, a qual deve estar no mesmo alinhamento vertical da palavra fim. Quando o bloco for apenas uma parte do algoritmo "fim." deve ser substituído por "fim;".

Observamos também que todos os comentários devem ser colocados entre chaves. Assim, são apenas comentários e não serão executados.

Exemplo: Ao escrevermos:

```
início
        lógico: RESPOSTA;
        inteiro: CON, SOMA;
        RESPOSTA := verdadeiro;
        leia (CON);
        SOMA := 10 + CON;
        escreva (CON, SOMA, RESPOSTA);
fim.
```

determinamos o início e o fim do bloco ou do algoritmo que construímos em nosso último exemplo.

Agora passaremos ao estudo das estruturas de controle do fluxo de execução de um algoritmo. Como já vimos essas estruturas podem ser divididas em três grupos:

sequênciação,

seleção e

repetição,

sendo as demais combinações destas.

Antes observamos que os exemplos que utilizaremos neste texto, não visam privilegiar esta ou aquela área. Na verdade são exemplos clássicos, já consagrados como eficientes na aprendizagem da construção de algoritmos, e também adotaremos o procedimento de utilizar, sempre que possível, os mesmos problemas nas diversas estruturas que estudaremos, para que o leitor possa efetivamente se dedicar ao estudo, compreensão e comparação das estruturas, sem ter como foco principal de reflexão a compreensão do problema, pois isto ele praticará muito nos exercícios propostos, quando as estruturas já estarão suficientemente esclarecidas.

3.3 Estrutura sequencial

É uma estrutura que permite a execução de um conjunto de ações ou comandos numa sequência linear, de cima para baixo e da esquerda para a direita, na qual todas as ações devem ser seguidas por um ponto e vírgula, que objetiva separar uma ação de outra e auxiliar a organização sequencial das ações. A regra sintática geral, para a execução de n comandos é:

início {começo do algoritmo}
{declaração de variáveis}
comando a;
comando b;
comando c;
⋮
⋮
comando n;
fim. {final do algoritmo}

na qual a declaração de variáveis e a sequência de comandos estão deslocados à direita em relação ao alinhamento das palavras início e fim, com a finalidade de aumentar a legibilidade, a qual, como já dissemos, é uma qualidade fundamental em qualquer algoritmo.

O leitor verificará que em todas as estruturas que serão definidas de agora em diante estes deslocamentos estarão presentes, visando facilitar o início e o fim do bloco, da estrutura ou do algoritmo, sendo que a palavra fim ao final de um bloco ou do algoritmo, ou a palavra finalizadora de uma certa estrutura, será seguida de um ponto final quanto for o final do algoritmo e nos demais casos de um ponto e vírgula.

Exemplos:

1) Construa um algoritmo que calcule a média aritmética entre cinco notas quaisquer fornecidas pelo usuário.

Para construir este algoritmo vamos executar os passos descritos anteriormente.

i) Ler atentamente o enunciado:

O que é média aritmética?

É a soma dos elementos divididos pela quantidade deles. Em nosso caso particular: (N1 + N2 + N3 + N4 + N5) / 5 onde N1, N2, N3, N4 e N5 são as cinco notas.

ii) Retirar do enunciado a relação das entradas de dados:

N1, N2, N3, N4 e N5.

iii) Retirar do enunciado a relação das saídas de dados:

Média aritmética: MA.

iv) Determinar o que deve ser feito para transformar as entradas determinadas nas saídas desejadas:

Utilizar a fórmula da média aritmética: MA = (N1 + N2 + N3 + N4 + N5) / 5.

v) Reescrever o determinado no passo anterior em Português Estruturado:

```
início {algoritmo}
        real: N1, N2, N3, N4, MA;
        leia (N1, N2, N3, N4, N5);
        MA := (N1 + N2 + N3 + N4 + N5) / 5;
        escreva (MA);
fim. {algoritmo}
```

vi) Executar o teste de mesa:

N1	N2	N3	N4	N5	MA
7	10	5	6	7	7

Agora vamos apresentar este algoritmo escrito para o ambiente Portugol Studio em três versões, nas quais algumas especificidades do ambiente e outras para melhorar a inserção e apresentação dos dados e resultados são aos poucos introduzidas, das quais o leitor perceberá facilmente as respectivas funcionalidades ao transcrever o que abaixo apresentamos a esse ambiente. Deixamos a cargo do leitor o estudo das muitas outras especificidades deste ambiente, as quais serão próprias também em cada linguagem de programação que leitor escolher, no entanto, em qualquer ambiente de programação os passos para a solução do problema permanecem inalterados.

Versão 1:

```
programa
{
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		leia (N1, N2, N3, N4, N5)
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva (MA)
	}
}
```

Versão 2:

```
programa
{
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		escreva ("digite N1 = ")
		leia (N1)
		escreva ("digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
		escreva ("digite N4 = ")
		leia (N4)
		escreva ("digite N5 = ")
		leia (N5)
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva ("a media e = ")
		escreva (MA)
	}
}
```

Versão 3:

```
programa
{
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		escreva ("digite N1 = ")
		leia (N1)
		escreva ("digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
		escreva ("digite N4 = ")
		leia (N4)
		escreva ("digite N5 = ")
		leia (N5)
		limpa()
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva ("a media e = ")
		escreva (MA)
	}
}
```

Atente-se a aba:

do seu Portugol Studio. Os seus importantes botões vão exibir um texto autoexplicativo ao você colocar o marcador do seu mouse sobre eles. Eles permitem, por exemplo, salvar e executar o programa, através do botão representado por um pen-drive e do botão representado por um triângulo, respectivamente, para o que nos referenciaremos apenas como rodar o programa.

Daí, para as notas 4, 3.5, 4.8, 8 e 9, ou seja, inserindo as entradas utilizando a tecla “enter” do teclado:

digite N1 = 4

digite N2 = 3.5

digite N3 = 4.8

digite N4 = 8

digite N5 = 9

obtemos como resposta “a media e = 5.86”.

2) Construa um algoritmo que calcule a quantidade de combustível necessária para uma viagem, quando a distância a ser percorrida e o consumo do veículo são fornecidos pelo usuário.

i) Ler atentamente o enunciado:

A quantidade de combustível - QTDE é fornecida pela distância - DIST dividido pelo consumo do veículo - CONS: $QTDE = \frac{DIST}{CONS}$.

ii) Retirar do enunciado a relação das entradas de dados:

DIST, CONS.

iii) Retirar do enunciado a relação das saídas de dados:

QTDE

iv) Determinar o que deve ser feito para transformar as entradas determinadas nas saídas desejadas:

Utilizar a fórmula $QTDE = \frac{DIST}{CONS}$.

v) Reescrever o determinado no passo anterior em Português Estruturado:

```
início {algoritmo}
        real: QTDE, DIST, CONS;
        leia (DIST, CONS);
        QTDE := DIST/CONS;
        escreva (QTDE);
fim. {algoritmo}
```

vi) Executar o teste de mesa:

DIST	CONS	QTDE
250	10	25

3) Construa um algoritmo que calcule a quantidade de latas de tinta necessárias e o custo para pintar externamente tanques cilíndricos de coleta de recicláveis, onde são fornecidos a altura e o raio desses tanques e a quantidade de tanques, sabendo-se que a lata de tinta custa R$ 100,00, cada lata contém 3.6 litros e com cada litro de tinta pinta-se 5 metros quadrados.

i) Quantidade de latas é dada por:

quantidade total de litros / 3.6;

Quantidade total de litros é dada por:

(área do tanque * quantidade de tanques) / 5;

Área do cilindro é dada por:

área da base + área lateral

sendo:

Área da base:

3.14 * R ** 2, onde R é o raio;

Área lateral:

H * comprimento = 2 * 3.14 * R onde H é a altura;

ii) Entradas: QTDETANQ, H, R;

iii) Dados de saída: custo C e quantidade de latas QTDELT;

iv) área = ((3.14 *R ** 2) + (H *2 * 3.14 * R)) * QTDETANQ;

litros = área / 5;

quantidade de latas = litros / 3.6;

custo = QTDELT * 100;

v) início {algoritmo}

```
real: H, R, QTDETANQ, CUST, QTDELT, AREA, LITRO;
leia (H, R);
AREA := ((3.14 * R ** 2) + (H * 2 * 3.14 * R)) * QTDETANQ;
LITRO := AREA / 5;
QTDELT := LITRO / 3.6;
CUST := QTDELT * 100;
escreva (CUST, QTDELT);
```

fim. {algoritmo}

vi)

H	R	QTDETANQ	AREA	LITRO	QTDELT	C
1	0.3	3	6.4998	1.2996	0.43332	43.332

Destacamos que, como fizemos nos algoritmos acima, é interessante ao criamos os identificadores das variáveis, levarmos em conta quais as informações que essas variáveis representam. Neste último exemplo se usássemos como variáveis A, B, C, D, E, F e G em vez de H, R, QTDETANQ, CUST, QTDELT, AREA, LITRO não estaria errado, mas prejudicaria a legibilidade do algoritmo, e portanto não devemos proceder assim.

Ressaltamos também que ao executar o teste de mesa é interessante colocarmos na linha de cabeçalho da tabela as variáveis na sequência que elas aparecem nos comandos do algoritmo.

A seguir apresentamos vários exercícios propostos, entre eles o clássico da troca de valores entre duas variáveis, nos quais a prática é aprimorada. Em seguida a estes apresentamos uma resolução desses exercícios com o objetivo principal de o leitor analisar uma construção, muito provavelmente, diferente da sua, visto que a construção de um algoritmo não é única.

3.4 Exercícios propostos

1) Construa um algoritmo para calcular as raízes de uma equação do segundo grau do tipo $Ax^2 + Bx + C = 0$, sendo que os coeficientes A, B e C são fornecidos pelo usuário.

2) Construa um algoritmo que, tendo como dados de entrada dois pontos quaisquer do plano P = (x1,y1) e Q = (x2,y2), imprima a distância entre eles, sabendo-se que a fórmula da distância é.

$$D = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}\ .$$

3) Faça um algoritmo para calcular o volume de uma esfera de raio R, em que R é um valor dado.

4) Faça um algoritmo que calcule a quantidade de litros de combustível gastos em uma viagem, utilizando-se um automóvel que faz 12 Km por litro em média. O usuário deverá fornecer o tempo gasto na viagem e a velocidade média em Km/hora.

5) Faça um algoritmo que leia uma temperatura em graus Centígrados (C) e apresente-a convertida em graus Fahrenheit (F), pela fórmula:

$$F = (9 * C + 160) / 5.$$

6) Faça um algoritmo que calcule e apresente o volume de uma lata de óleo, utilizando a fórmula:

VOLUME: = 3.14159 * R ** 2 * H.

7) Faça um algoritmo que leia dois valores para as variáveis A e B e efetue a troca dos valores de forma que a variável A passe a possuir o valor da variável B e a variável B passe a possuir o valor da variável A, e apresente os valores trocados.

8) Efetuar o cálculo do valor de uma prestação em atraso, utilizando a fórmula:

PRESTAÇÃO := VALOR + (VALOR * (TAXA / 100) * TEMPO).

3.5 Gabarito dos exercícios propostos

1)

```
início
        real: A, B, C, X1, X2, D;
        leia (A, B, C);
        D :=(B**2-4*A*C);
        X1 := (– B + 2 // D) / (2 * A);
        X2 := (– B – 2 // D) / (2 * A);
        escreva (X1, X2);
fim.
```

---

2)

```
início
        real: X1, Y1, X2, Y2, DIST;
        leia (X1, Y1);
        leia (X2,Y2);
        DIST := 2 // ((X2 – X1) ** 2 + (Y2 – Y1) ** 2);
        escreva (DIST);
fim.
```

---

3)

```
início
        real: VOL, PI, R;
        leia (R);
        PI := 3.14;
        VOL := 4 / 3 * PI * R ** 3;
        escreva (VOL);
fim.
```

---

4)
```
início
        real: QTDL, DP, TG, VM;
        leia (TG, VM);
        DP := TG * VM;
        QTDL := DP / 12;
        escreva (QTDL);
fim.
```

---

5)
```
início
        real: C, F;
        leia (C);
        F := (9 * C + 160) / 5;
        escreva (F);
fim.
```

---

6)
```
início
        real: R, VOL, H;
        leia (R, H);
        VOL := 3,14 * R ** 2 * H;
        escreva (VOL);
fim.
```

---

7) início

```
        real: A, B, TROCA;
        leia (A, B);
        TROCA := A;
        A := B;
        B := TROCA;
        escreva (A, B);
fim.
```

Comentário: Veja que foi utilizada a variável TROCA foi utilizada para guardar o valor da variável A para que está receba o valor da variável B.

---

8) início

```
        real: VAL, TEM, TX, PREST;
        leia (VAL, TEM, TX);
        PREST := VAL + (VAL * (TX / 100) * TEM);
        escreva (PREST);
fim.
```

3.6 Estruturas de seleção

São estruturas que permitem que um grupo de ações ou comandos selecionados sejam executados quando determinadas condições, representadas por expressões relacionais ou lógicas, forem ou não satisfeitas. O grupo de ações ou comandos selecionados podem, inclusive, formar outras estruturas.

As estruturas de seleção estão divididas em três grupos básicos:

seleção simples;
seleção composta; e
seleção encadeada,

e a seleção encadeada pode ser homogênea, heterogênea ou de múltipla escolha.

3.6.1 Seleção simples

A regra sintática geral, para a execução de n comandos, ou seja, sua estrutura é:

```
se <condição>
        então
                início {bloco verdade}
                        C1;
                        C2;    {sequência de comandos}
                        ⋮
                        Cn;
                fim;   {bloco verdade}
    fimse;
```

onde <condição> representa uma expressão relacional ou lógica, que, quando inspecionada, gera um resultado verdadeiro ou (exclusivo) falso,

sendo que se <condição> for verdadeira, o *bloco verdade*, ou seja, a sequência de comandos C1, C2, ..., Cn será executado e caso contrário, isto é, se <condição> for falsa, nada é executado, isto é, encerra a estrutura. A palavra se delimita o início desta estrutura e a palavra fimse marca o seu final.

Veja que a sequência de comandos C1, C2, ..., Cn constitui um bloco e portanto deve ser delimitado por início e fim. Isso é necessário pois há um conjunto com várias ações no bloco verdade, mas caso exista apenas uma ação nesse bloco não será necessário delimitar o bloco.

Exemplo: Voltando ao primeiro exemplo da estrutura sequencial, vamos construir um algoritmo que calcule a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, agora com uma avaliação quanto à aprovação, neste caso, obtida atingindo-se média superior ou igual a seis.

Como este problema já foi por nós tratado passo a passo, vamos direto a construção do algoritmo em Português Estruturado, levando em conta que teremos também como informação de saída, além da média aritmética, MA, uma informação adicional, se o aluno for aprovado.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA;
        leia (N1, N2, N3, N4, N5);
        MA := (N1 + N2 + N3 + N4 + N5) / 5;
        escreva (MA);
        se MA >= 6
                então
                        escreva (“Aluno aprovado!”);
        fimse;
fim. {algoritmo}
```

O teste de mesa é:

N1	N2	N3	N4	N5	MA	MA >= 6	Mensagem
7	10	5	6	7	7	V	Aluno aprovado!
6	5	4	7	3	0	F	

Veja que neste exemplo não temos mensagem quando a média for menor que seis. Se assim desejarmos devemos usar duas estruturas de seleção simples, por enquanto:

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA;
        leia (N1, N2, N3, N4, N5);
        MA := (N1 + N2 + N3 + N4 + N5) / 5;
        escreva (MA);
        se MA >= 6
                então
                        escreva (“Aluno aprovado!”);
        fimse;
        se MA < 6
                então
                        escreva (“Aluno reprovado!”);
        fimse;
fim. {algoritmo}
```

O teste de mesa é:

N1	N2	N3	N4	N5	MA	MA >= 6	MA < 6	Mensagem
7	10	5	6	7	7	V	F	Aluno aprovado!
6	5	4	7	3	0	F	V	Aluno reprovado!

Para o Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		escreva ("digite N1 = ")
		leia (N1)
		escreva ("digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
		escreva ("digite N4 = ")
		leia (N4)
		escreva ("digite N5 = ")
		leia (N5)
		limpa()
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva ("a media e = ")
		escreva (MA)
		limpa()
		se (MA >= 6)
		{
			escreva ("Aluno aprovado! A sua media e = ",
mat.arredondar(MA, 2))
		}
		se (MA < 6)
		{
			escreva ("Aluno reprovado! A sua media e = ",
mat.arredondar(MA, 2))
		}
```

```
        }
}
```

Após rodar o programa, para as notas 4, 5.5, 7.3, 8 e 9:

digite N1 = 4

digite N2 = 5.5

digite N3 = 7.3

digite N4 = 8

digite N5 = 9

obtemos como resposta “Aluno aprovado! A sua media e = 6.76”, e para as notas 4, 3.5, 4.8, 8 e 9:

digite N1 = 4

digite N2 = 3.5

digite N3 = 4.8

digite N4 = 8

digite N5 = 9

obtemos como resposta “Aluno reprovado! A sua media e = 5.86”.

Observamos a utilização da biblioteca “Matemática” usada para o arredondamento da média final, neste caso, com até duas casas decimais. O Portugol Studio possui muitas outras bibliotecas, as quais deixamos a critério da curiosidade do leitor o conhecimento e a utilização de cada uma delas.

Nos próximos problemas implementados nesta linguagem não apresentaremos as entradas e saídas deste ambiente, pois, como o leitor perceberá, o número de simulações para provocar todas os tipos saídas aumenta conforme tornamos o nosso problema mais complexo.

Veja que neste último algoritmo temos a mensagem da reprova, mas para tanto, como usamos duas estruturas de seleção simples tivemos que testar duas condições, quando na verdade, sabemos que se uma for verdadeira a outra será falsa e quando uma é falsa a outra é verdadeira. Neste caso dizemos que essas condições são *complementares*, pois elas cobrem todas as possibilidades. Nesta situação é mais vantajoso usar a seleção composta, a qual descrevemos a seguir.

3.6.2 Seleção composta

Sua estrutura é:

```
se <condição>
        então
                início  {bloco verdade}
                        C1;
                        C2;     {sequência de comandos}
                        ⋮
                        Cn;
                fim;    {bloco verdade}
        senão
                início  {bloco falso}
                        D1;
                        D2;     {sequência de comandos}
                        ⋮
                        Dm;
                fim;    {bloco falso}
fimse;
```

onde <condição> representa uma expressão relacional ou lógica, que, quando inspecionada, gera um resultado verdadeiro ou (exclusivo) fal-

so, sendo que se <condição> for verdadeira, o *bloco verdade*, ou seja, a sequência de comandos C1, C2, ..., Cn, será executado e caso contrário, ou seja, se <condição> for falsa, o *bloco falso*, ou seja, a sequência de comandos D1, D2, ..., Dm, será executado, e daí encerra a estrutura, a qual é finalizada com a palavra fimse, e inicializada com a palavra se.

A existência dos blocos verdade e falso, demarcados por início e fim, faz-se necessária devido à existência de um conjunto com várias ações no bloco verdade e no bloco falso. Caso exista apenas uma ação no bloco não será necessário delimita-lo.

Exemplo: Reescrevendo nosso último algoritmo, agora utilizando a seleção composta, ele fica assim:

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA;
        leia (N1, N2, N3, N4, N5);
        MA := (N1 + N2 + N3 + N4 + N5) / 5;
        escreva (MA);
        se MA >= 6
                então
                        escreva (“Aluno aprovado!”);
                senão
                        escreva (“Aluno reprovado!”);
        fimse;
fim. {algoritmo}
```

O teste de mesa é:

N1	N2	N3	N4	N5	MA	MA >= 6	Mensagem
7	10	5	6	7	7	V	Aluno aprovado!
6	5	4	7	3	0	F	Aluno reprovado!

No Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		escreva ("digite N1 = ")
		leia (N1)
		escreva ("digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
		escreva ("digite N4 = ")
		leia (N4)
		escreva ("digite N5 = ")
		leia (N5)
		limpa()
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva ("a media e = ")
		escreva (MA)
		limpa()
		se (MA >= 6)
		{
			escreva ("Aluno aprovado! \nA sua media e = ",
			mat.arredondar(MA, 2))
		}
		senao
		{
			escreva ("Aluno reprovado! \nA sua media e = ",
			mat.arredondar(MA, 2))
		}
	}
}
```

Ao executar este algoritmo no ambiente Portugol Studio o leitor perceberá a função de cada comando que aos poucos estamos introduzindo em nossos algoritmos, como é o caso de "\n" no algoritmo acima, utilizado para inserir quebra de linhas.

3.6.3 Seleção encadeada

É uma estrutura utilizada quanto uma determinada ação ou bloco de ações está vinculado a várias condições, ou seja, deve ser executado quando um de condições for satisfeito. Está dividida em três tipos básicos:

seleção encadeada homogênea;

seleção encadeada heterogênea; e

seleção de múltipla escolha,

obtidos a partir da seleção simples e da seleção composta. Por isso, a delimitação dos blocos e das estruturas segue o que mencionamos para estas duas já estudadas.

3.6.3.1 Seleção encadeada homogênea

É uma seleção encadeada tal que sua estrutura segue um determinado padrão lógico. Existem dois tipos de seleção encadeada homogênea:

se então se; e

se senão se.

Na seleção encadeada homogênea se então se a sua estrutura é:

```
se <condição 1>
        então
                se <condição 2>
                        então
                                se <condição 3>
                                                ⋮
                                                        ⋮
                                                        se <condição n>
                                                                então
                                        {bloco verdade}         início
                                                                        C1;
                                        {sequência de comandos}         C2;
                                                                        ⋮
                                                                        Cn;
                                        {bloco verdade}         fim;
                                                        fimse;
                                                        ⋮
                                                ⋮
                                fimse;
                fimse;
fimse;
```

Observe que o padrão desta estrutura é após cada então existe outro se, e não existem senões.

Podemos simplificar esta estrutura, pois o conjunto de ações C1, C2, ..., Cn será executado somente quando as n condições forem satisfeitas, ou seja, quando todas as condições forem simultaneamente verdadeiras, portanto, seria equivalente a escrever, simplificadamente:

```
se <condição 1> e <condição 2>  e ... e <condição n>
        então
                início {bloco verdade}
                        C1;
                        C2;      {sequência de comandos}
                        ⋮
                        Cn;
                fim; {bloco verdade}
fimse;
```

A tabela de decisão desta estrutura é:

condição 1	condição 2	...	condição n	ação
V	V	...	V	bloco verdade

sendo que para todas as outras $2^n - 1$ possibilidades de cada uma dessas n condições assumirem os valores V ou F o bloco de ações não é executado, ou seja, se pelo menos uma das n condições for falsa (F) nenhuma ação será realizada.

Se avaliarmos a eficiência de um algoritmo apenas pelo número de operações que ele realiza, se por um lado a segunda forma de apresentação desta estrutura é mais simples, ela também é menos eficiente, pois na primeira delas s e a i-ésima condição for falsa, a condição i + 1, ..., condição n não é(são) testada(s), para qualquer $1 \leq i \leq n-1$, enquanto que nesta última forma todas as n condições são sempre verificadas.

Esta estrutura deve ser utilizada quando temos que uma determinada ação (ou bloco de ações) será executada quando n condições forem satisfeitas.

Exemplo: Dados o sexo, a altura e a idade de uma pessoa, verificar se ela pode assumir na carreira Militar um determinado posto,

admitindo-se que para este posto há a exigência de ser homem, com no mínimo 1.65 m de altura e idade entre 18 e 35 anos.

i) Exigências para o posto:
SEXO = Masculino,
ALT >= 1.65,
IDADE >= 18 e IDADE <= 35.

ii) Dados de entrada: SEXO, ALT, IDADE.

iii) Dados de saída: mensagem-Apto.

iv) O que devemos fazer para transformar os dados de entrada em saídas?

Se as três exigências forem verdadeiras a mensagem deve ser exibida.

v) Construção do algoritmo:

```
início {algoritmo}
        inteiro: IDADE;
        real: ALT;
        cadeia: SEXO;
        leia (SEXO, ALT, IDADE);
        se SEXO = "Masculino"
              então
                    se ALT >= 1.65
                          então
                                se IDADE >= 18 e IDADE <= 35
                                      então
                                            escreva ("Apto");
                                fimse;
                    fimse;
        fimse;
fim.{algoritmo}
```

vi) O teste de mesa é:

SEXO	ALT	IDADE	SEXO = "Masculino"	ALT >= 1.65	IDADE >= 18 e IDADE <= 35	mensagem
Feminino	1.5	30	F			
Masculino	1.5	30	V	F		
Masculino	1.7	40	V	V	F	
Masculino	1.7	30	V	V	V	Apto

onde uma célula em branco significa que a respectiva condição não foi testada. Assim, para cada célula em branco, considerando que a respectiva condição pode ser verdadeira ou falsa, essa linha da tabela de decisão incorpora duas linhas da tabela verdade, uma é obtida admitindo essa condição como falsa e a outra é obtida admitindo essa condição como verdadeira. Por isso que esta tabela de decisão contém apenas quatro linhas, enquanto a tabela verdade contempla, neste caso, oito possibilidades.

Veja que temos outras infinitas combinações de atribuições às variáveis SEXO, ALT e IDADE. Na verdade escolhemos uma combinação que torne a condição SEXO = "Masculino" falsa e depois outra que torne essa condição verdadeira, e assim agimos escolhendo combinações que tornem as demais condições ora falsa e depois verdadeira. Por isso que ao definirmos o teste de mesa, citamos que muitas vezes podemos nos dar por satisfeitos ao testar um algoritmo com uma tabela de decisão. No caso do algoritmo acima teríamos:

SEXO = "Masculino"	ALT >= 1.65	IDADE >= 18 e IDADE <= 35	mensagem
F			
V	F		
V	V	F	
V	V	V	Apto

Observe que como o algoritmo acima envolve a condição IDADE >= 18 e IDADE <= 35 podemos reescrevê-lo de forma a torná-lo mais eficiente:

```
início {algoritmo}
        inteiro: IDADE;
        real: ALT;
        cadeia: SEXO;
        leia (SEXO, ALT, IDADE);
        se SEXO = "Masculino"
            então
                se ALT >= 1.65
                        então
                                se IDADE >= 18
                                        então
                                                se IDADE <= 35
                                                        então
                                                          escreva ("Apto");
                                                fimse;
                                fimse;
                se ALT >= 1.65
                        então
                                se IDADE >= 18
                                        então
                                                se IDADE <= 35
                                                        então
                                                          escreva ("Apto");
                                                fimse;
                                fimse;
        fimse;
fim.{algoritmo}
```

cujo teste é:

SEXO = "Masculino"	ALT >= 1.65	IDADE >= 18	IDADE <= 35	mensagem
F				
V	F			
V	V	F		
V	V	V	F	
V	V	V	V	Apto

O algoritmo

```
início {algoritmo}
        inteiro: IDADE;
        real: ALT;
        cadeia: SEXO;
        leia (SEXO, ALT, IDADE);
        se SEXO = "Masculino" e ALT >= 1.65 IDADE >= 18 e IDADE <= 35
                                                        então
                                                            escreva ("Apto");
        fimse;
fim.{algoritmo}
```

realiza a mesma tarefa que os dois anteriores, mas, embora de mais fácil compreensão, testa a cada execução, as quatro condições SEXO = "Masculino", ALT >= 1.65, IDADE >= 18 e IDADE <= 35 mesmo com a informação de que basta uma delas ser falsa para que a mensagem não seja exibida.

Na seleção encadeada homogênea se senão se a sua estrutura é:

```
se <condição 1>
        então
            início {bloco verdade 1}
                C11;
                C12;      {sequência de comandos}
                ⋮
                C1p;
            fim;   {bloco verdade 1}
```

```
        senão
                se <condição 2>
                        então
                                início {bloco verdade 2}
                                        C21;
                                        C22;{sequência de comandos}
                                        ⋮
                                        C2q;
                                fim;    {bloco verdade 2}
                        senão
                                se <condição 3>
                                        então
                                                ⋮
                                                    ⋮
                                                    senão
                                                          se <condição n>
                                                              então
                                                    {bloco verdade n}  início
                                                                       Cn1;
                                                    {sequência de comandos} Cn2;
                                                                       ⋮
                                                                       Cnr;
                                                    {bloco verdade n}    fim;
                                                          fimse;
                                                    ⋮
                                                ⋮
                                fimse;
                fimse;
fimse;
```

Para esta estrutura o padrão é após cada senão existe outro comando se, e depois do então existe uma ação ou bloco de ações que não é outra seleção.

Esta estrutura realiza os testes das condições somente até encontrar uma que seja satisfeita e neste caso a respectiva ação (ou bloco de ações) é executada e a estrutura é finalizada. Assim, ela deve ser usada sempre que soubermos que somente uma das n condições será satisfeita.

Sua tabela de decisão é:

condição 1	condição 2	condição 3	....	condição n	ação executada
V					bloco verdade 1
F	V				bloco verdade 2
F	F	V			bloco verdade 3
⋮					
F	F	F		V	bloco verdade n

Nesta tabela as células vazias indicam que a respectiva condição não foi testada. Assim, para cada célula em branco, considerando que a respectiva condição pode ser verdadeira ou falsa, essa linha da tabela de decisão incorpora duas linhas da tabela verdade, uma é obtida admitindo essa condição como falsa e a outra é obtida admitindo essa condição como verdadeira.

Exemplo: Vamos reescrever nosso algoritmo do cálculo da média das cinco notas, incluindo agora um teste no caso da reprovação, de forma que se a média do aluno for maior que 3 ele fará um exame final:

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA;
        leia (N1, N2, N3, N4, N5);
        MA := (N1 + N2 + N3 + N4 + N5) / 5;
        escreva (MA);
        se MA >= 6
                então
                        escreva (“Aluno aprovado!”);
                senão
                        início
                                escreva (“Aluno reprovado!”);
                                se MA >= 3
                                        então
                                                escreva (“Exame final!”);
                        fim;
        fimse;
fim. {algoritmo}
```

Vamos então testar este algoritmo apenas com a tabela de decisão:

MA >= 6	MA >= 3	Mensagem
V		Aluno aprovado!
F	V	Aluno reprovado! Exame final!
F	F	Aluno reprovado!

No Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		escreva ("digite N1 = ")
		leia (N1)
		escreva ("digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
		escreva ("digite N4 = ")
		leia (N4)
		escreva ("digite N5 = ")
		leia (N5)
		limpa()
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva ("a media e = ")
		escreva (MA)
		limpa()
		se (MA >= 6)
		{
			escreva ("Aluno aprovado! \nA sua media e = ",
mat.arredondar(MA, 2))
		}
```

```
                senao
                {
                        escreva ("Aluno reprovado! \nA sua media e = ",
mat.arredondar(MA, 2))
                        se (MA >= 3)
                        {
                                escreva ("\nExame final!")
                        }
                }
        }
}
```

Exemplo: Dada a idade de um atleta, entre 7 e 25 anos, verificar o nível de enquadramento dele dentro de um determinado clube, sendo entre 7 e 11 anos o nível é dente de leite, entre 12 e 15 anos o nível é infantil, entre 16 e 18 anos é juvenil e entre 19 e 25 anos é profissional.

i)

Idade	Nível de enquadramento
7 a 11	Dente de leite
12 a 15	Infantil
16 a 18	Juvenil
19 a 25	Profissional

ii) Dados de entrada: IDADE.

iii) Dados de saída: mensagem: Nível de enquadramento.

iv) O que devemos fazer para transformar os dados de entrada em saídas?

De acordo com a idade um nível de enquadramento deve ser exibido:

Se IDADE >= 7 e IDADE <= 11 exibir “dente de leite”;
Se IDADE >= 12 e IDADE <= 15 exibir “infantil”;
Se IDADE >= 16 e IDADE <= 18 exibir “juvenil”;
Se IDADE >= 19 e IDADE <= 25 exibir “profissional”;

sendo que após encontrar uma condição verdadeira a respectiva mensagem é exibida e o algoritmo será finalizado.

v) Construção do algoritmo:

```
início {algoritmo}
    inteiro: IDADE;
    leia (IDADE);
    se IDADE >= 7 e IDADE <= 11
        então
            escreva (“dente de leite”);
        senão
            se IDADE >= 12 e IDADE <= 15
                então
                    escreva (“infantil”);
                senão
                    se IDADE >= 16 e IDADE <= 18
                        então
                            escreva (“juvenil”);
                            senão
                                se IDADE >= 19 e IDADE<= 25
```

```
                                        então
                                            escreva (“profissional”);
                                    fimse;
                            fimse;
                    fimse;
    fimse;
fim.{algoritmo}
```

vi) O teste da tabela de decisão é:

Se IDADE >= 7 e IDADE <= 11	IDADE >= 12 e IDADE <= 15	Se IDADE >= 16 e IDADE <= 18	IDADE >= 19 e IDADE <= 25	mensagem
V				dente de leite
F	V			infantil
F	F	V		juvenil
F	F	F	V	profissional

onde as células em branco significa que a respectiva condição não foi testada. Assim, para cada célula em branco, considerando que a respectiva condição pode ser verdadeira ou falsa, essa linha da tabela de decisão incorpora duas linhas da tabela verdade, uma é obtida admitindo essa condição como falsa e a outra é obtida admitindo essa condição como verdadeira. Por isso que esta tabela de decisão contém apenas quatro linhas, enquanto a tabela verdade contempla, neste caso, dezesseis possibilidades.

Observe que neste último exemplo foi exigido que a idade do atleta esteja entre 7 e 25 anos. Podemos desejar que o tratamento para outras idades também seja feito, e que atletas com idade inferior a 7 anos ou superior a 25 anos sejam considerados não classificados nesse clube, mas isso somente pode ser feito com uma das duas próximas estruturas que estudaremos.

3.6.3.2 Seleção encadeada heterogênea

Nesta estrutura o programador pode utilizar o encadeamento de seleção simples e seleção composta de acordo com a sua necessidade. Por isso ela não possui uma estrutura pré-definida.

Apenas para exemplificá-la e praticarmos a construção da tabela de decisão consideremos o encadeamento heterogêneo seguinte.

```
se <condição 1>
        então
                se <condição 2>
                        então
                                início  {bloco verdade 1}
                                        C1;
                                        C2; {sequência de comandos}
                                        ⋮
                                        Cn;
                                fim;    {bloco verdade 1}
                fimse;
        senão
                se <condição 3>
                        então
                                início  {bloco verdade 2}
                                        D1;
        {sequência de comandos}         D2;
                                        ⋮
                                        Dm;
                                fim;    {bloco verdade 2}
```

```
                    senão
                        se <condição 4>
                                então
                    {bloco verdade 3}       início
                                                    E1;
                    {sequência de comandos}         E2;
                                                    ⋮
                                                    Ep;
                    {bloco verdade 3}       fim;
                                senão
                        {bloco falso}       início
                                                    F1;
                    {sequência de comandos}         F2;
                                                    ⋮
                                                    Fq;
                        {bloco falso}       fim;
                        fimse;
            fimse;
fimse;
```

Observe que não conseguimos identificar um padrão lógico de construção da estrutura acima.

A tabela de decisão deste encadeamento é:

condição 1	condição 2	condição3	condição 4	ação executada
V	V			bloco verdade 1
F		V		bloco verdade 2
F		F	V	bloco verdade 3
F		F	F	bloco falso

Nesta tabela as células vazias indicam que a respectiva condição não foi testada.

Na construção da estrutura acima envolvemos 4 condições, porém podemos construir uma estrutura de seleção encadeada heterogênea para n condições, onde n é um número natural finito fixo e maior do que 1.

Exemplo: Retomaremos o nosso problema do cálculo de média. Agora desejamos um algoritmo que calcule a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, com uma avaliação quanto à aprovação, obtida atingindo-se média superior ou igual a seis, e quanto à reprovação, obtida quando a média é inferior a seis, e que forneça o conceito final do aluno, de acordo com a tabela:

Média (MA)	Conceito
8 <= MA <= 10	A
6 <= MA < 8	B
0 <= MA < 6	R

Vamos incluir ainda um teste que verificará se a média obtida está entre zero e dez.

```
início {algoritmo}
    real: N1, N2, N3, N4, N5, MA;
    leia (N1, N2, N3, N4, N5);
    MA := (N1 + N2 + N3 + N4 + N5) / 5;
    escreva (MA);
    se MA >= 8 e MA <= 10
      então
```

```
            início
                escreva (“Aluno aprovado!”);
                escreva (“Conceito A!”);
            fim;
        senão
            se MA >= 6 e MA < 8
                então
                    início
                        escreva(“Alunoaprovado!”);
                        escreva (“Conceito B!”);
                    fim;
                senão
                    se MA >= 0 e MA < 6
                        então
                            início
                                escreva (“Aluno reprovado!”);
                                escreva (“Conceito R!”);
                            fim;
                        senão
                                escreva (“Média não válida!”)
                    fimse;
            fimse;
    fimse;
fim. {algoritmo}
```

A partir deste algoritmo os passos i, ii), iii) e iv) para a construção de um algoritmo serão por nós omitidos, mas tomaremos o cuidado de que após a apresentação do algoritmo esses passos fiquem claros ao leitor, quem deve continuar os executando na sua prática.

MA >= 8 e MA <= 10	MA >= 6 e MA < 8	MA >= 0 e MA < 6	mensagens
V			Aluno aprovado! Conceito A!
F	V		Aluno aprovado! Conceito B!
F	F	V	Aluno reprovado! Conceito R!
F	F	F	Nota não válida!

Em Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		escreva ("digite N1 = ")
		leia (N1)
		escreva ("digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
		escreva ("digite N4 = ")
		leia (N4)
		escreva ("digite N5 = ")
		leia (N5)
		limpa()
		MA = (N1 + N2 + N3 + N4 + N5) / 5
		escreva ("a media e = ")
```

```
            escreva (MA)
            limpa()
            se (MA >= 8 e MA <= 10)
            {
                  escreva ("Aluno aprovado! \nA sua media e = ", mat.
arredondar(MA, 2))
                  escreva ("\n Conceito A!")
            }
            senao
                  se (MA >= 6 e MA < 8)
                  {
                        escreva ("Aluno aprovado! \nA sua media e = ",
mat.arredondar(MA, 2))
                        escreva ("\n Conceito B!")
                  }
                  senao
                        se (MA >= 0 e MA < 6)
                        {
                              escreva ("Aluno reprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
                              escreva ("\nConceito R!")
                        }
                        senao
                              {
                                    escreva ("\nMédia não válida!")
                              }

      }
}
```

Exemplo: Retomando o exemplo da classificação de um atleta, consideremos agora que dada a idade de um atleta, verificar o nível de enquadramento dele dentro de um determinado clube, sendo que de 7 a 11 anos o nível é dente de leite, de 12 a 15 anos o nível é infantil, de 16 a 18 anos é juvenil, de 19 a 25 anos é profissional e atletas com idade inferior a 7 anos ou superior a 25 anos são considerados não classificados.

O enunciado acima é resumido pela tabela:

Idade	Nível de enquadramento
7 a 11	Dente de leite
12 a 15	Infantil
16 a 18	Juvenil
19 a 25	Profissional
Inferior a 7 ou superior a 25	Não classificado

e como já mencionamos, os passos i, ii), iii) e iv) para a construção de um algoritmo serão por nós omitidos, vamos direto a construção do algoritmo:

```
início {algoritmo}
    inteiro: IDADE;
    leia (IDADE);
    se IDADE >= 7 e IDADE <= 11
        então
            escreva (“dente de leite”);
        senão
            se IDADE >= 12 e IDADE <= 15
                então
```

```
                    escreva (“infantil”);
                senão
                    se IDADE >= 16 e IDADE <= 18
                        então
                            escreva (“juvenil”);
                        senão
                            se IDADE >= 19 e IDADE <= 25
                                então
                                    escreva (“profissional”);
                                senão
                                    escreva (“não classificado”);
                            fimse;
                    fimse;
            fimse;
    fimse;
fim.{algoritmo}
```

O teste é:

Se IDADE >= 7 e IDADE <= 11	IDADE >= 12 e IDADE <= 15	Se IDADE >= 16 e IDADE <= 18	IDADE >= 19 e IDADE <= 25	mensagem
V				dente de leite
F	V			infantil
F	F	V		juvenil
F	F	F	V	profissional
F	F	F	F	não classificado

onde as células em branco significa que a respectiva condição não foi testada.

Neste último algoritmo apenas acrescentamos uma ação a ser realizada quando todas as condições forem falsas, o que descaracterizou a estrutura se senão se utilizada anteriormente.

Reescreveremos estes dois últimos problemas com a estrutura que estudaremos a seguir, preferida por muitos programadores, mas, no entanto, como veremos é aplicável a apenas uma parte dos problemas tratados pelos encadeamentos.

3.6.4 Seleção de múltipla escolha

Esta estrutura é geralmente utilizada para simplificar a estrutura se senão se quando todas as condições são constituídas do operador relacional = (igual a) comparando uma variável X a n valores Xi que podem ser assumidos por X sendo que uma respectiva ação Ai é executada somente quando a comparação de X com o valor Xi produzir o resultado verdadeiro, para i: 1, 2, ...,N.

Sua estrutura é:

```
escolha X
        caso X1: A1;
        caso X2: A2;
              ⋮
        caso XN: An;
fimescolha;
```

Caso o conteúdo da variável X seja igual ao valor Xi, para $1 \leq i \leq N$ então a ação Ai, para $1 \leq i \leq n$ será executada e a estrutura é encerrada.

Se uma ação, A, deve ser realizada para mais de um valor da variável X, todos esses valores devem ser agrupados em um único caso. Além disso, se uma ação A estiver vinculada a todos valores inteiros da variável X entre a e b podemos escrever:

caso a . . b: A.

Observe que esta estrutura é delimitada pelas palavras escolha e fimescolha.

Exemplo: Reescrevendo o nosso ante último problema, agora com a estrutura de múltipla escolha temos o algoritmo:

```
início {algoritmo}
        inteiro: IDADE;
        leia (IDADE);
        escolha IDADE
                caso 7 . . 11: escreva (“dente de leite”);
                caso 12 . . 15: escreva (“infantil”);
                caso 16 . . 18: escreva (“juvenil”);
                caso 19 . . 25: escreva (“profissional”);
        fimescolha;
fim.{algoritmo}
```

Para executarmos uma ação que se verifica com todos os outros valores, exceto os descriminados caso a caso, incluímos outra situação, a caso contrário.

Sua estrutura é:

```
escolha X
        caso X1, Y1: A1;
        caso X2: A2;
                ⋮
        caso XN: An;
        caso contrário: B1;
fimescolha;
```

na qual caso o conteúdo da variável X seja igual ao valor Xi, para $1 \leq i \leq N$ então a ação Ai, para $1 \leq i \leq n$ será executada, e caso X seja diferente dos valores X1, ..., XN então a ação B1 será executada, e a estrutura é encerrada.

Da mesma forma que na estrutura anterior, se uma ação deve ser realizada para mais de um valor da variável X, todos esses valores devem ser agrupados em um único caso. Além disso, se uma ação A estiver vinculada a todos valores inteiros da variável X entre a e b podemos escrever:

```
caso a . . b: A.
```

Exemplo: Considerando novamente nosso último problema, com a opção de que atletas com idade inferior a 7 anos ou superior a 25 anos são considerados não classificados, temos o seguinte algoritmo:

```
início {algoritmo}
        inteiro: IDADE;
        leia (IDADE);
        escolha IDADE
```

```
                caso 7 . . 11: escreva (“dente de leite”);
                caso 12 . . 15: escreva (“infantil”);
                caso 16 . . 18: escreva (“juvenil”);
                caso 19 . . 25: escreva (“profissional”);
                caso contrário: escreva (“não classificado”);
        fimescolha;
fim.{algoritmo}
```

Como nesta estrutura compara-se uma variável X a n valores que podem ser assumidos por ela muitos ambientes de programação exigem que essa variável seja do tipo inteiro, cadeia ou caractere.

Retomemos o já bastante explorado algoritmo do cálculo da média aritmética entre cinco notas quaisquer fornecidas pelo usuário, com o teste que verifica se a média obtida está entre zero e dez, e que agora forneça apenas o conceito final do aluno, de acordo com a tabela:

Média (MA)	Conceito
8 <= MA <= 10	A
6 <= MA < 8	B
0 <= MA < 6	R

Para viabilizar a utilização desta estrutura criaremos uma variável auxiliar, AUX, do tipo inteiro, que receberá valores, uma para cada faixa de médias, de acordo com a tabela acima.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA;
        inteiro: AUX;
        leia (N1, N2, N3, N4, N5);
        MA := (N1 + N2 + N3 + N4 + N5) / 5;
        escreva (MA);
        se MA >= 8 e MA <= 10
                então
                        AUX := 1;
                senão
                        se MA >= 6 e MA < 8
                                então
                                        AUX := 2;
                                senão
                                        se MA >= 0 e MA < 6
                                                então
                                                        AUX := 3;
                                        fimse;
                        fimse;
        fimse;
        escolha AUX
                caso 1: escreva (“Aluno aprovado!”, “Conceito A”);
                caso 2: escreva (“Aluno aprovado!”, “Conceito B”);
                caso 3: escreva (“Aluno reprovado!”, “Conceito R”);
                caso contrário: escreva (“Média não válida!”);
        fimescolha;
fim. {algoritmo}
```

MA >= 8 e MA <= 10	MA >= 6 e MA < 8	MA >= 0 e MA < 6	mensagens
V			Aluno aprovado! Conceito A!
F	V		Aluno aprovado! Conceito B!
F	F	V	Aluno reprovado! Conceito R!
F	F	F	Média não válida!

Vamos a prática com o Portugol Studio, destacando que neste ambiente a variável escolhida pode assumir e ser comparada somente com valores inteiros, um de cada vez, e por isso utilizamos a variável AUX, a qual deve sempre assumir algum valor antes do início da estrutura de seleção de múltipla escolha, ou seja, mesmo para a execução do caso contrário a variável AUX deve possuir algum valor inteiro, por isso atribuímos o valor 0 a essa variável já após a sua declaração.

```
programa
{
	funcao inicio()
	{
		real N1, N2, N3, N4, N5, MA
		inteiro AUX
		AUX = 0
		escreva ("digite N1 = ")
		leia (N1)
		escreva (“digite N2 = ")
		leia (N2)
		escreva ("digite N3 = ")
		leia (N3)
```

```
escreva ("digite N4 = ")
leia (N4)
escreva ("digite N5 = ")
leia (N5)
limpa()
MA = (N1 + N2 + N3 + N4 + N5) / 5
limpa()
se (MA >= 8 e MA <= 10)
{
        AUX = 1
}
senao
{
        se (MA >= 6 e MA < 8)
        {
                AUX = 2
        }
        senao
        {
                se (MA >= 0 e MA < 6)
                {
                        AUX = 3
                }
        }
}
escolha (AUX)
{
        caso 1:
                escreva ("Aluno aprovado!\nConceito A!")
```

```
                    pare  // Impede que as instruções do caso 2 ou
posteriores sejam executadas
                caso 2:
                    escreva ("Aluno aprovado!\nConceito B!")
                    pare  // Impede que as instruções do caso 3 ou
posterior sejam executadas
                caso 3:
                    escreva ("Aluno reprovado!\nConceito R!")
                    pare  // Impede que a instrução do caso
contrário seja executada
                caso contrario: // Será executado para qualquer
opção diferente de 1, 2 ou 3
                    escreva ("Média não válida!")
            }
        }
}
```

Quanto ao passo vi), o teste do algoritmo, ele será ainda mais importante a partir da inclusão em nossos algoritmos das estruturas de repetição que estudaremos a seguir. Nele devemos ter atenção especial quanto ao número de repetições.

Seguem outros exercícios propostos para o aprimoramento da prática através de problemas clássicos no estudo de algoritmos e outros nem tanto. Em seguida apresentamos uma resolução desses exercícios, a qual, como já mencionamos, não é única, e agora a possibilidade de diferentes resoluções é ainda maior, pois a nossa criatividade aumenta à medida que conhecemos novas estruturas.

3.7 Exercícios propostos

1) Construa um algoritmo para calcular as raízes de uma equação do segundo grau do tipo $Ax^2+Bx+C=0$, sendo que os coeficientes A, B e C são fornecidos pelo usuário, levando em consideração a análise da existência de raízes.

2) Tendo como dados de entrada a altura e o sexo de uma pessoa, construa um algoritmo que calcule seu peso ideal, utilizando as seguintes fórmulas:

para homens: (72.7 * h) – 58,

para mulheres: (62.1 * h) – 44.7.

3) Faça um algoritmo que leia dois valores numéricos e apresente a diferença do maior pelo menor.

4) Faça um algoritmo que leia um valor inteiro e apresente o módulo do valor lido.

5) Faça um algoritmo que leia uma senha fornecida pelo usuário e compare-a com uma senha previamente estabelecida, e forneça mensagens a respeito da veracidade da senha fornecida.

6) Elabore um algoritmo que, dada a idade de um nadador, classifique-o em uma das seguintes categorias:

dente de leite: 5 - 7 anos,

infantil: 8 - 10 anos,

juvenil: 11 - 13 anos,

amador: 14 - 17 anos,

sênior: maiores de 18 anos.

7) Faça um algoritmo que leia três valores e apresente os valores dispostos em ordem crescente.

8) Dados três valores A, B e C verificar se eles podem ser os comprimentos dos lados de um triângulo, e se forem, verificar se compõem um triângulo equilátero, isósceles ou escaleno.

9) Construa um algoritmo que, tendo como dados de entrada o preço de um produto e um código de origem, emita o preço e a sua origem. Caso o código não seja nenhum dos especificados abaixo, o produto deve ser encarado como importado. Os códigos de origem são:

1, 2 ou 3 para a região Sul,

4 ou 5 para a região Norte,

6, 7 ou 8 para a região Leste, e

9 para a região Oeste.

3.8 Gabarito dos exercícios propostos

1)

```
início
        real: A, B, C, X1, X2, D;
        leia (A, B, C);
        D := B ** 2 – 4 *A * C;
        se D < 0
                então
                        escreva (“Não existem raízes”);
                 senão
                        se delta = 0
                                então
                                        início
                                                X1 := – B / (2 *A);
                                                escreva (X1);
                                        fim;
                                 senão
                                        início
                                                X1 : = (– B + (2 // D) / (2 *A);
                                                X2 := (– B – (2 // D) / (2 *A);
                                                escreva (X1, X2);
                                        fim;
                        fimse;
        fimse;
fim.
```

2)

```
início
    cadeia: SEXO;
    real: PESO, ALTURA;
    leia (SEXO, ALTURA);
    se SEXO <> "M" e SEXO <> "F"
                                então
                                    início
                                        escreva ("Digite apenas M ou F");
                                        leia (SEXO);
                                    fim;
    fimse;
    se SEXO = "M"
            então
                PESO := (72.7 * ALTURA) – 58;
            senão
                PESO := (62.1 * ALTURA) – 44.7;
    fimse;
    escreva ("Seu peso ideal é:", PESO);
fim.
```

---

3)

```
início
        real: NUM1, NUM2, DIF;
        leia (NUM1, NUM2);
        se NUM1 = NUM2
                então
                        escreva (“Números iguais, não há diferença”);
                senão
                        se NUM1 > NUM2
                                então
                                        início
                                                DIF: = NUM1 – NUM2;
                                                escreva (DIF);
                                        fim;
                                senão
                                        início
                                                DIF: = NUM2 – NUM1;
                                                escreva (DIF);
                                        fim;
                        fimse;
        fimse;
fim.
```

Comentário: Outra forma de construir este algoritmo é utilizar a função abs.

---

4)

```
início
        inteiro: NUM;
        leia (NUM);
        se NUM > 0 ou NUM = 0
                                        então
                                                escreva (NUM);
                                         senão
                                                escreva (abs(NUM));
        fimse;
fim.
```

Comentário: Não apresentamos abs(NUM) de forma direta pois no caso de números não negativos a avaliação desta função é desnecessária.

---

5)

```
início
        caractere: SENHA;
        leia (SENHA);
        se SENHA = "AAP1"
                                then
                                        escreva ("senha correta");
                                  senão
                                        escreva ("senha incorreta");
        fimse;
fim.
```

---

6)

```
início
inteiro: IDADE;
leia (IDADE);
se (IDADE >= 5) e (IDADE <= 7)
    então
        escreva (“Dente de leite”);
    senão
        se (Idade >= 8) e (IDADE <= 10)
            então
                escreva (“Infantil”);
            senão
                se (IDADE >= 11) e (IDADE <= 13)
                    então
                        escreva (“Juvenil”);
                    senão
                        se (IDADE >=14) e (IDADE<=17)
                            então
                                escreva (“Amador”);
                            senão
                                se (IDADE > 18)
                                    então
                                        escreva
(“Profissional”);
                                    senão
                                        escreva
(“valor inválido”);
                                fimse;
                        fimse;
                fimse;
        fimse;
fimse;
fim.
```

7)

```
início
    real: A, B, C;
    leia (A, B, C);
    se A > B
            então
                se A > C
                        então
                            se B > C
                                    então
                                        escreva (C, B, A);
                                    senão
                                        escreva (B, C, A);
                            fimse;

                        senão
                            escreva (B, A, C);
                fimse;
            senão
                se B > C
                        então
                            se C > A
                                    então
                                        escreva (A, C, B);
                                    senão
                                        escreva (C, A, B);
                            fimse;
                        senão
                            escreva (A, B, C);
                fimse;
    fimse;
fim.
```

Comentário: Este problema pode ser resolvido ordenando-se dois valores e depois compara-se o terceiro com os dois já ordenados. Esta ideia é apresentada a seguir.

```
início
    real: A, B, C, MAIOR, MENOR;
    leia (A, B, C);
    se A > B
            então
                início
                    MAIOR := A;
                    MENOR := B;
                fim;
            senão
                início
                    MAIOR := B;
                    MENOR := A;
                fim;
    fimse;
    se C > MAIOR
            então
                escreva (MENOR, MAIOR, C);
            senão
                se C < MENOR
                        então
                            escreva (C, MENOR, MAIOR);
                        senão
                            escreva ( MENOR, C, MAIOR);
                fimse;
    fimse;
fim.
```

8)

```
início
        inteiro: A, B, C;
        leia (A, B, C);
        se (A < B + C) e (B < A + C) e (C < A + B)
                então
                        se (A = B) e (B = C)
                                então
                                        escreva (“triângulo equilátero”);
                                senão
                                        se (A = B) ou (A = C) ou (B = C)
                                                então
                                                        escreva (“triângulo isósceles”);
                                                senão
                                                        escreva (“triângulo escaleno”);
                                        fimse;
                        fimse;
                senão
                        escreva (“não compõem triângulo”);
        fimse;
fim.
```

---

9)

```
início
        real: PREÇO;
        inteiro: ORIGEM;
        leia (PREÇO, ORIGEM);
        escolha ORIGEM
                caso 1 . . 3: escreva (PREÇO, "- produto do Sul");
                caso 4, 5: escreva (PREÇO, "- produto do Norte");
                caso 6 . . 8: escreva (PREÇO, "- produto do Leste");
                caso 9: escreva (PREÇO, "- produto do Oeste");
                caso contrário: escreva (PREÇO, "- produto importado");
        fimescolha;
fim.
```

3.9 Estruturas de repetição

São estruturas que permitem que um trecho do algoritmo seja repetido um número finito de vezes, fazendo com que o fluxo de execução passe por esse mesmo trecho diversas vezes, mediante a verificação de uma condição. Essas estruturas se apresentam em três tipos:

epetição com teste no início,

repetição com teste no final, e

repetição com variável de controle.

Como nos próximos algoritmos retomaremos problemas já tratados em exemplos anteriores, não realizaremos o tratamento passo a passo, iremos direto a construção do algoritmo.

3.9.1 Repetição com teste no início

É uma estrutura de controle do fluxo lógico que permite executar um número finito de vezes um mesmo trecho do algoritmo verificando antes de cada execução uma condição que quando atendida permite repetir o trecho. Sua estrutura é:

```
enquanto <condição> faça
        C1;
        C2;      {sequência de comandos}
        ⋮
        Cn;
fimenquanto;
```

a qual recebe o nome de *estrutura enquanto*. Essa estrutura permite que um bloco de ações ou uma ação seja repetido(a) um número finito de

vezes enquanto a condição for verdadeira e quando o resultado da verificação da condição for falso a estrutura é abandonada. Assim, se já da primeira vez o resultado for falso, os comandos não são executados nenhuma vez. Veja que esta estrutura é delimitada pelas palavras enquanto e fimenquanto.

Exemplo: Voltando ao exemplo de construir um algoritmo que calcule a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, e emita uma avaliação quanto à aprovação, neste caso, obtida atingindo-se média superior ou igual a seis, com a informação que provém do caso falso da condição MA >= 6, ou seja, a reprovação do aluno, consideraremos agora uma turma de 5 alunos.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA;
        inteiro: CON;
        CON := 0;
        enquanto CON < 5 faça
                leia (N1, N2, N3, N4, N5);
                MA := (N1 + N2 + N3 + N4 + N5) / 5;
                escreva (MA);
                se MA >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
                CON := CON + 1;
        fimenquanto;
fim. {algoritmo}
```

O teste é:

CON	CON< 5	N1	N2	N3	N4	N5	MA	MA>= 6	Mensagem
0	V	7	10	5	6	7	7	V	Aluno aprovado!
1	V	6	5	4	7	3	5	F	Aluno reprovado!
2	V	5	5	5	5	5	5	F	Aluno reprovado!
3	V	3	5	5	2	5	4	F	Aluno reprovado!
4	V	10	8	8	10	4	8	V	Aluno aprovado!
5	F								

Observe neste teste que o número de repetições foi exatamente 5, como desejado.

Em Portugol Studio temos:

```
programa
{
inclua biblioteca Matematica --> mat
inclua biblioteca Util
funcao inicio()
	{
		inteiro CON
		real N1, N2, N3, N4, N5, MA
		CON = 0
		enquanto (CON < 5)
		{
			escreva ("\ndigite N1 = ")
			leia (N1)
			escreva ("digite N2 = ")
```

```
                        leia (N2)
                        escreva ("digite N3 = ")
                        leia (N3)
                        escreva ("digite N4 = ")
                        leia (N4)
                        escreva ("digite N5 = ")
                        leia (N5)
                        MA = (N1 + N2 + N3 + N4 + N5) / 5
                        escreva ("a media e = ")
                        escreva (MA)
                        se (MA >= 6)
                        {
                                escreva ("\nAluno aprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
                        }
                        senao
                        {
                                escreva ("\nAluno reprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
                        }
                        Util.aguarde(2000)
                        /*
                         * Aguarda 2000 milisegundos
                         */
                        CON = CON + 1
                        limpa()
                }
        }
}
```

Exemplo: Consideremos novamente o problema que dada a idade de um atleta, devemos verificar o nível de enquadramento dele dentro de um determinado clube, sendo que de 7 a 11 anos o nível é dente de leite, de 12 a 15 anos o nível é infantil, de 16 a 18 anos é juvenil, de 19 a 25 anos é profissional e atletas com idade inferior a 7 anos ou superior a 25 anos são considerados não classificados, sendo que agora isso será feito para um grupo de atletas, cuja quantidade é desconhecida e o termino da lista é indicado pela digitação da palavra FIM como nome de atleta.

```
início {algoritmo}
    inteiro: IDADE;
    cadeia: NOME;
    NOME := ENTRE;
    enquanto NOME <> “FIM” faça
        leia (IDADE, NOME);
        se IDADE >= 7 e IDADE <= 11
            então
                escreva (“dente de leite”);
            senão
                se IDADE >= 12 e IDADE <= 15
                    então
                        escreva (“infantil”);
                    senão
                        se IDADE >= 16 e IDADE <= 18
                            então
                                escreva (“juvenil”);
                            senão
                                se IDADE >= 19 e IDADE <= 25
```

```
                                        então
                                            escreva (“profissional”);
                                        senão
                                            escreva (“não classificado”);
                                    fimse;
                                fimse;
                            fimse;
                        fimse;
                    fimenquanto;
fim.{algoritmo}
```

O teste é:

IDADE	NOME	IDADE >=7 e IDADE <=11	IDADE >=12 e IDADE <=5	IDADE >=16 e IDADE <=18	IDADE >=19 e IDADE <=25	mensagem
	ENTRE					
10	João	V				dente de leite
13	Antonio	F	V			infantil
17	Beto	F	F	V		juvenil
20	Claudio	F	F	F	V	profissional
5	Júnior	F	F	F	F	não classificado
	FIM					

Observe que neste algoritmo atribuímos ENTRE à variável NOME apenas para iniciar a repetição.

Cada execução de uma estrutura de repetição recebe o nome de loop.

Observe que no último exemplo, estabelecemos um modo de contagem, o que é feito através de um *contador* representado por uma variável com um dado valor inicial, o qual é incrementado a cada repetição. Sua estrutura geral é:

```
inteiro: CON;          {declaração do contador}
CON := VI;             {inicializar o contador}
CON := CON + INC;      {incrementar o contador de INC}
```

na qual, VI é um número inteiro fixo, o valor inicial da contagem, geralmente zero ou um, e INC é o valor a ser incrementado a cada repetição do trecho, chamado de *incremento do contador*. Nesta estrutura geral o contador foi declarado como um dado do tipo inteiro, pois geralmente contamos de um em um, ou de dois em dois, ou se preferir de n em n, onde n é um inteiro, no entanto podemos declarar o contador como real, se desejarmos contar de tal forma que ele assuma valores não inteiros.

Caso seja necessário guardar cumulativamente o(s) valor(es) calculado(s) em cada loop, isto pode ser feito através da criação de uma variável chamada *acumulador*. Sua estrutura geral é:

```
inteiro: ACM;          {declaração do acumulador}
ACM := VI;             {inicializar o acumulador}
ACM := ACM + INC;      {incrementar o acumulador de X}
```

na qual, VI é um número inteiro fixo, o valor inicial do acumulador, e INC é o valor a ser acumulado a cada repetição do trecho, que pode ser um valor inserido pelo usuário ou calculado pelo algoritmo. Nesta estrutura geral o acumulador foi declarado como um dado do tipo inteiro, no entanto é comum declarar o acumulador como real, quando ele assume valores não inteiros.

Exemplo: Vamos refazer o algoritmo do exemplo anterior, atendendo a exigência de ser calculada a média da turma de 5 alunos.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA, ACM, MEDIATUR;
        inteiro: CON;
        CON := 0;
        ACM := 0;
        enquanto CON < 5 faça
                leia (N1, N2, N3, N4, N5);
                MA := (N1 + N2 + N3 + N4 + N5) / 5;
                escreva (MA);
                se MA >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
                CON := CON + 1;
                ACM := ACM + MA;
        fimenquanto;
        MEDIATUR := ACM / 5;
        escreva (MEDIATUR);
fim. {algoritmo}
```

O teste é:

CON	CON < 5	N1	N2	N3	N4	N5	MA	MA >= 6	Mensagem	ACM	MEDIATUR
0	V	7	10	5	6	7	7	V	Aluno aprovado!	7	
1	V	6	5	4	7	3	5	F	Aluno reprovado!	12	
2	V	5	5	5	5	5	5	F	Aluno reprovado!	17	
3	V	3	5	5	2	5	4	F	Aluno reprovado!	21	
4	V	10	8	8	10	4	8	V	Aluno aprovado!	29	5,8
5	F										

A partir deste ponto deixaremos o teste do algoritmo por conta do leitor.

3.9.2 Repetição com teste no final

É uma estrutura de controle do fluxo lógico que permite executar um número finito de vezes um mesmo trecho do algoritmo verificando ao final de cada execução uma condição que quando atendida encerra a repetição do trecho. Sua estrutura é:

```
repita
        C1;
        C2;       {sequência de comandos}
        ⋮
        Cn;
até <condição>;
```

a qual recebe o nome de *estrutura repita*. Essa estrutura permite que um bloco de ações ou uma ação seja repetido(a) um número finito de vezes enquanto a condição for falsa e quando o resultado da verificação da condição for verdadeiro a estrutura é abandonada. Assim, o bloco de ações C1, C2, ..., Cn é executado pelo menos uma vez, independente da validade da condição. Isto ocorre porque a primeira verificação da condição é feita após a primeira execução do bloco. Esta estrutura é delimitada pelas palavras repita e até.

Vamos reescrever o exemplo do algoritmo que calcula a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, emite uma avaliação quanto à aprovação, neste caso, obtida atingindo-se média superior ou igual a seis, com a informação que provém do caso falso

da condição MA >= 6, ou seja, a reprovação do aluno, considerando uma turma de 5 alunos, para a qual deve ser calculada a média, usando a estrutura repita.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA, ACM, MEDIATUR;
        inteiro: CON;
        CON := 0;
        ACM := 0;
        repita
                leia (N1, N2, N3, N4, N5);
                MA := (N1 + N2 + N3 + N4 + N5) / 5;
                escreva (MA);
                se MA >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
                CON := CON + 1;
                ACM := ACM + MA;
        até CON = 5;
                MEDIATUR  := ACM / 5;
                escreva (MEDIATUR);
fim. {algoritmo}
```

Em Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	inclua biblioteca Util
	funcao inicio()
	{
		inteiro CON
		real N1, N2, N3, N4, N5, MA, ACM, MEDIATUR
		CON = 0
		ACM = 0
		faca
		{
			escreva ("\ndigite N1 = ")
			leia (N1)
			escreva ("digite N2 = ")
			leia (N2)
			escreva ("digite N3 = ")
			leia (N3)
			escreva ("digite N4 = ")
			leia (N4)
			escreva ("digite N5 = ")
			leia (N5)
			MA = (N1 + N2 + N3 + N4 + N5) / 5
			escreva ("a media e = ")
			escreva (MA)
			se (MA >= 6)
			{
```

```
                    escreva ("\nAluno aprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
                }
                senao
                {
                    escreva ("\nAluno reprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
                }
                Util.aguarde(2000)
                limpa()
                CON = CON + 1
                ACM = ACM + MA
            }
            enquanto (CON < 5)
            MEDIATUR = ACM / 5
            escreva ("\nA media da turma e = ", mat.
arredondar(MEDIATUR, 2))
        }
}
```

Observe que no Portugol Studio a repetição com teste no final:

```
repita
    C1;
    C2;      {sequência de comandos}
    ⋮
    Cn;
até <condição>;
```

possui a sintática alterada para:

```
faça
{
        C1;
        C2;     {sequência de comandos}
        ⋮
        Cn;
}
enquanto <condição>;
```

e por isso a condição para encerrar a repetição pode sofrer um pequeno ajuste. No último exemplo a condição CON = 5 foi ajustada para CON < 5.

Agora vamos reescrever o algoritmo da classificação do atleta com a estrutura repita:

```
início {algoritmo}
    inteiro: IDADE;
    cadeia: NOME;
    repita
        leia (NOME);
        se NOME < > "FIM"
            então
                leia (IDADE);
                se IDADE >= 7 e IDADE <= 11
                    então
                        escreva ("dente de leite");
                    senão
                        se IDADE >= 12 e IDADE <= 15
```

```
                                então
                                    escreva ("infantil");
                                senão
                                    se IDADE >= 16 e IDADE <= 18
                                        então
                                            escreva ("juvenil");
                                        senão
                                            se IDADE >= 19 e
IDADE <= 25
                                                então
                                                    escreva
("profissional");
                                                senão
                                                    escreva
("não classificado");
                                            fimse;
                                    fimse;
                            fimse;
                    fimse;
            fimse;
    até NOME = "FIM";
fim. {algoritmo}
```

Veja que neste algoritmo, foi inserida uma seleção simples para verificar o nome digitado antes de solicitar a idade do atleta, pois quando é digitado FIM nada deve ser executado.

3.9.3 Repetição com variável de controle

É uma estrutura de controle do fluxo lógico que permite executar um número finito e pré-definido de vezes um mesmo trecho do algoritmo verificando ao final de cada execução uma condição que quando atendida encerra a repetição do trecho. Sua estrutura é:

```
para CON de vi até vf passo p faça
        C1;
        C2;      {sequência de comandos}
        ⋮
        Cn;
fimpara;
```

onde CON é a variável de controle, a qual deve ser informada na declaração de variáveis, vi é o valor inicial da variável CON, vf é o valor final da variável CON, ou seja, o valor até o qual ela pode chegar e p é o valor do incremento dado à variável CON. Esta estrutura é delimitada pelas palavras para e fimpara.

Como definimos, esta estrutura repete o trecho do algoritmo um número finito e pré-definido de vezes, pois ela possui limites fixos, vi e vf.

Veja que esta estrutura já possui um contador para o qual temos uma inicialização (vi), um teste para verificar se este atingiu o limite (vf) e um incremento (p) após cada execução do bloco de repetição.

Esta estrutura recebe o nome de *estrutura para*.

Exemplo: Voltando novamente ao exemplo das notas, usando agora a estrutura para, temos o algoritmo:

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA, ACM, MEDIATUR;
        inteiro: CON;
        ACM := 0;
        Para CON de 0 até 4 passo 1 faça
                leia (N1, N2, N3, N4, N5);
                MA := (N1 + N2 + N3 + N4 + N5) / 5;
                escreva (MA);
                se MA >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
                ACM := ACM + MA;
        fimpara;
        MEDIATUR := ACM / 5;
        escreva (MEDIATUR);
fim. {algoritmo}
```

Em Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	inclua biblioteca Util
	funcao inicio()
	{
		inteiro CON
		real N1, N2, N3, N4, N5, MA, ACM, MEDIATUR
		ACM = 0
		para (CON = 0; CON < 5; CON++)
		{
			escreva ("\ndigite N1 = ")
			leia (N1)
			escreva ("digite N2 = ")
			leia (N2)
			escreva ("digite N3 = ")
			leia (N3)
			escreva ("digite N4 = ")
			leia (N4)
			escreva ("digite N5 = ")
			leia (N5)
			MA = (N1 + N2 + N3 + N4 + N5) / 5
			escreva ("a media e = ")
			escreva (MA)
			se (MA >= 6)
			{
				escreva ("\nAluno aprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
```

```
                        }
                        senao
                        {
                                escreva ("\nAluno reprovado! \nA sua
media e = ", mat.arredondar(MA, 2))
                        }
                        ACM = ACM + MA
                        Util.aguarde(2000)
                        limpa()
                }
                MEDIATUR = ACM / 5
                escreva ("\nA media da turma e = ", mat.
arredondar(MEDIATUR, 2))
        }
}
```

Veja que no Portugol Studio o limite final (vf) é substituído por uma condição sobre o contador a qual deve ser atendida para que ocorra a execução do bloco de ações. No exemplo acima, o limite final 4 para o contador CON foi substituído pela condição CON < 5. Também, o incremento (p) do contador após cada execução do bloco de repetição no Portugol Studio é substituído pelo incremento "++" unitário. Se desejarmos outro incremento podemos obtê-lo através do incremento do contador no interior da estrutura de repetição.

Observe que para utilizarmos a estrutura para precisamos saber, antes de iniciarmos a repetição, quantas vezes devemos repetir o trecho do algoritmo. Assim, o nosso problema de classificar uma lista de atletas, cuja quantidade não sabemos e o término da lista é indicado pela palavra FIM, não pode ser tratado por esta estrutura.

Toda estrutura enquanto pode ser convertida para repita e vice-versa, e toda estrutura para pode ser convertida para enquanto ou repita, mas nem, toda estrutura enquanto ou repita pode ser convertida em para.

3.9.4 Modularização

Em geral, problemas mais complexos exigem algoritmos extensos. Neste casos, muitas vezes, mesmo quando conseguimos um eficiente algoritmo, a sua legibilidade fica muito comprometida. Uma solução para este problema é dividir estes algoritmos mais complexos em pequenos algoritmos conhecidos como *módulos* ou *subalgoritmos*, buscando aumentar a funcionalidade das partes do conjunto, facilitando o seu entendimento e possibilitando a reutilização destas partes.

Para tanto, devemos dividir algoritmo completo em módulo principal e diversos módulos ou subalgoritmos, tantos quantos forem necessários, sendo o módulo principal aquele onde a execução é inicializada, e os subalgoritmos são os outros módulos que são acionados (ativados ou chamados) pelo algoritmo principal e que por sua vez poderão também acionar outros subalgoritmos, sendo que ao acionarmos um subalgoritmo as ações contidas nele são executadas e, em seguida a execução retorna ao ponto da sua chamada.

Não existe ordem para definição dos módulos, e variáveis que são utilizadas apenas em um determinado módulo podem ser declaradas nesse módulo, através de uma *declaração local de variáveis*.

A cada módulo ou subalgoritmo damos um nome e para acioná-lo o mencionamos como uma linha de execução no local desejado.

É muito útil ter um subalgoritmo que tenha a função de calcular um resultado, como funções matemáticas. Este tipo de módulo tem o objetivo de retornar uma única informação. Como nos outros módulos já estudados este recebe um nome que o identifica, sendo que já em seguida ao nome, entre parênteses ocorre a declaração das variáveis a serem utilizadas nesse módulo.

A sua estrutura é:

módulo <nome do módulo>(declaração das variáveis);
início
C1;
C2; {sequência de comandos}
⋮
Cn;
fim;

Observe que após o nome do módulo há um par de parênteses para a declaração das variáveis.

Quando o subalgoritmo tem a função de calcular um resultado para utilizá-lo usamos:

retorne (X);

onde X representa uma informação numérica, alfanumérica ou uma expressão aritmética.

Exemplo: Construa um algoritmo no qual é fornecido o raio de circunferências para o cálculo do seu comprimento, sendo que o final da lista será identificado pelo raio nulo.

```
início {algoritmo}
        módulo comp(real: R);
                início
                        retorne (2 * 3.14 * R);
                fim;
        real: RAIO;
        repita
                leia (RAIO);
                retorne (comp(RAIO));
        até RAIO = 0;
fim. {algoritmo}
```

Como prática retomaremos em seguida a construção do algoritmo que calcula a média aritmética entre cinco notas quaisquer fornecidas pelo usuário utilizando a modularização, mas evidentemente, não há grandes vantagens em relação àquele que realizava todas as ações necessárias dentro do algoritmo principal. Na verdade, a modularização é bastante vantajosa quando se trata de problemas complexos, multitarefas, capazes de realizar diversas tarefas independentes, mas relacionadas. Por exemplo, um algoritmo que gerencie o estoque de diversos produtos em uma grande empresa utilizados na produção de um determinado artigo. Naturalmente, embora a produção seja de um determinado artigo, o controle de cada produto do estoque deve ser realizado de forma independente, uma vez que a empresa pode produzir vários artigos, compostos por número distinto de produtos do estoque.

Assim, o algoritmo deve possuir um módulo para o controle de cada um dos produtos do estoque, ficando a cargo do algoritmo principal

a chamada de um ou mais módulos de acordo com o artigo em produção. Isto permite que vários programadores desenvolvam o algoritmo, cada um desenvolvendo alguns módulos, facilitando a realização dos testes do algoritmo, pois os módulos podem ser testados separadamente e alterações posteriores podem ficar restritas a modificações em alguns módulos.

Exemplo: O algoritmo abaixo, construído anteriormente, calcula a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, e emite uma avaliação quanto à aprovação, neste caso, obtida atingindo-se média superior ou igual a seis, com a informação que provém do caso falso da condição MA >= 6, ou seja, a reprovação do aluno, considerando uma turma de 5 alunos.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, MA, ACM, MEDIATUR;
        inteiro: CON;
        CON := 0;
        ACM := 0;
        repita
                leia (N1, N2, N3, N4, N5);
                MA := (N1 + N2 + N3 + N4 + N5) / 5;
                escreva (MA);
                se MA >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
               CON := CON + 1;
               ACM := ACM + MA;
```

```
        até CON = 5;
        MEDIATUR := ACM / 5;
        escreva (MEDIATUR);
fim. {algoritmo}
```

Agora vamos reescrevê-lo utilizando modularização.

```
início {algoritmo}
        módulo média:
                início
                        real: N1, N2, N3, N4, N5, MA;
                        MA := (N1 + N2 + N3 + N4 + N5) / 5;
                        escreva (MA);
                        retorne (MA);
                fim;
        real: N1, N2, N3, N4, N5, MA, MA, ACM, MEDIATUR;
        inteiro: CON;
        CON := 0;
        ACM := 0;
        repita
                leia (N1, N2, N3, N4, N5);
                módulo média;
                se MA >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
                CON := CON + 1;
                ACM := ACM + MA;
        até CON = 5;
        MEDIATUR := ACM / 5;
        escreva (MEDIATUR);
fim. {algoritmo}
```

Observe que agora o cálculo e a exibição da média, de cada aluno são realizados no módulo média, localizado no início do algoritmo principal, no qual foi realizada também a declaração das variáveis N1, N2, N3, N4, N5 e MA, uma vez que todas elas são utilizadas no algoritmo principal e também neste módulo.

Comparando as duas construções logo acima confirmamos que para este problema a modularização não apresenta grandes vantagens.

Em Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	inclua biblioteca Util
	funcao inicio()
	{
		inteiro CON
		real N1, N2, N3, N4, N5, ACM, MEDIATUR
		CON = 0
		ACM = 0
		faca
		{
			escreva ("\ndigite N1 = ")
			leia (N1)
			escreva ("digite N2 = ")
			leia (N2)
			escreva ("digite N3 = ")
			leia (N3)
			escreva ("digite N4 = “)
			leia (N4)
```

```
                    escreva ("digite N5 = ")
                    leia (N5)
                    se (media(N1,N2,N3,N4,N5) >= 6)
                    {
                        escreva ("\nAluno aprovado! \nA sua
media e = ", mat.arredondar(media(N1,N2,N3,N4,N5), 2))
                    }
                    senao
                    {
                        escreva ("\nAluno reprovado! \nA sua
media e = ", mat.arredondar(media(N1,N2,N3,N4,N5), 2))
                    }
                    Util.aguarde(2000)
                    limpa()
                    CON = CON + 1
                    ACM = ACM + media(N1,N2,N3,N4,N5)
                }
                enquanto (CON < 5)
                MEDIATUR = ACM / 5
                escreva ("\nA media da turma e = ", mat.
arredondar(MEDIATUR, 2))
        }
        funcao real media(real N1, real N2, real N3, real N4, real N5)
        {
            real MA
            MA = (N1 + N2 + N3 + N4 + N5)/5
            retorne MA
        }
}
```

3.10 Exercícios propostos

1) Faça um algoritmo que leia um número N, some todos os números inteiros entre 1 e N, e mostre o resultado obtido.

2) Faça um algoritmo que leia um número e divida-o por 2 sucessivamente até que o resultado seja menor que 1 e mostre o resultado da última divisão efetuada.

3) Faça um algoritmo que leia dois números X e N. A seguir, mostre o resultado das divisões de X por N, onde após cada divisão, X passa a ter como conteúdo o resultado da divisão anterior e N é decrementado de 1 em 1, até chegar a 2.

4) Faça um algoritmo que leia uma lista de números terminada pelo número 0 e mostre cada número lido. Ao final o algoritmo deve mostrar a média aritmética de todos os números da lista.

5) Faça um algoritmo que leia um número e, a seguir, leia uma lista de números até achar um igual ao primeiro lido e mostre todos os números lidos.

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

7) Faça um algoritmo que leia uma lista de N números, e mostre o quadrado de todos os números lidos.

8) Faça um algoritmo que leia um número N e uma lista de N números inteiros positivos e mostre o maior número da lista.

9) Construa um algoritmo, que dado um conjunto de valores determine qual o menor valor do conjunto. O final do conjunto de valores é conhecido através do valor zero.

10) Faça um algoritmo que leia uma lista de números e some todos os números pares desta lista. A lista será finalizada pelo número 0 (zero).

11) Faça um algoritmo que leia uma lista de números e some todos os números pares e todos os números ímpares desta lista. A lista será finalizada pelo número 0 (zero).

12) Faça um algoritmo que leia uma lista de letras terminada pela letra "z", e ao final mostre a quantidade de vogais desta lista.

13) Faça um programa que leia uma lista de letras terminadas em "z" e ao final mostre a quantidade de vogais e consoantes desta lista.

14) Sendo H = 1 + 1/2 + 1/3 + ... + 1/N, prepare um algoritmo para gerar o número H, sendo que o número N é fornecido pelo usuário.

15) Elabore um algoritmo que calcule N! (fatorial de N), sendo que o valor de N é fornecido pelo usuário. Sabendo-se que:

$$N! = 1 * 2 * 3 * ... * (N - 1) * N,$$

$$0! = 1, \text{ por definição.}$$

16) Utilize o conceito de modularização para elaborar um algoritmo que leia duas listas formadas por duplas de números e calcule a soma dos números de cada dupla da primeira lista e a multiplicação dos números da dupla respectiva na segunda lista, ou seja, posição a posição. As listas serão finalizadas quando o resultado da soma dos números de uma dupla da primeira lista for igual ao resultado da multiplicação dos números da respectiva dupla da segunda lista.

3.11 Gabarito dos exercícios propostos

1)

```
início
        real:  NUM;
        inteiro: SOMA,CONT;
        CONT := 1;
        SOMA := 0;
        leia (NUM);
        repita
                SOMA := SOMA + CONT;
                CONT := CONT + 1;
        até (CONT > NUM);
        escreva (SOMA);
fim.
```

---

2)

```
início
        real: NUM;
        leia (NUM);
        repita
                NUM := NUM / 2;
        até NUM < 1;
        escreva (NUM);
fim.
```

---

3)

```
início
        real: X, N;
        repita
                leia (X, N);
        até N <> 0,
        repita
                X := X / N;
                N := N − 1;
                escreva (X);
        até N < 2;
fim.
```

---

4)

```
início
        real: NUM, QTDE, SOMA;
        QTDE := 0;
        SOMA := 0;
        repita
                leia (NUM);
                escreva (NUM);
                SOMA := SOMA + NUM;
                QTDE := QTDE + 1;
        até NUM = 0;
        escreva (SOMA/QTDE);
fim.
```

---

5)

```
início
        real: LISTNUM, N;
        leia (N);
        repita
                leia (LISTNUM);
                escreva (“Numero:”, LISTNUM);
        até (LISTNUM = N)
        escreva (“Primeiro novamente!”);
fim.
```

---

6)

```
início
        inteiro: N, CONT;
        real: NUM, SOMA;
        SOMA := 0;
        leia (N);
        para CONT de 1 até N passo1 faça
                leia (NUM);
                SOMA := SOMA + NUM;
        fimpara;
        escreva (SOMA);
fim.
```

---

7)

```
início
        inteiro: QTDE;
        real: NUM;
        leia (N);
        repita
                leia (NUM);
                escreva (NUM * NUM);
                N := N − 1;
        até N = 0;
fim.
```

---

8)

```
início
        inteiro: CONT, MAIOR;
        real: NUM,
        MAIOR := 0;
        CONT := 0;
        leia (N) ;
        repita
                leia (NUM);
                se NUM > 0 e int(NUM) = NUM
                        então
                                início
                                        CONT := CONT + 1;
                                        se NUM  > MAIOR
                                                então
```

```
                                                    MAIOR := NUM;
                                            fimse;
                                    fim;
                fim se;
        até CONT = N;
        escreva (MAIOR);
fim.
```

Comentário: A variável MAIOR foi inicializada com o valor nulo, pois os N números são inteiros positivos, o que é verificado na primeira condição da seleção.

---

9)

```
início
        real: NUM, MENOR;
        leia (NUM);
        MENOR := NUM;
        enquanto NUM <> 0
                leia (NUM);
                se NUM < MENOR
                                então
                                        MENOR := NUM;
                fimse;
        fimenquanto;
        escreva ("Menor número:", MENOR);
fim.
```

Comentário: Ao ler o primeiro número este é o menor e portanto ele é atribuído a variável MENOR.

10)

```
início
        inteiro: NUM, SOMA;
        repita
                leia (NUM);
                se INT(NUM / 2) = NUM / 2
                                        então
                                                SOMA := SOMA + NUM;
                fim se;
        até NUM = 0;
        escreva (SOMA);
fim.
```

Comentário: Foi utilizado que um número x é par se int(x / 2) = x / 2.

---

11)

```
início
        inteiro: NUM, NP, NI;
        NUM := 1;
        NP := 0;
        NI := 0;
        enquanto NUM <> 0 faça
                leia (NUM);
                se (NUM / 2) = int(NUM / 2)
                        então
                                NP := NP + NUM;
                        senão
                                se frac(NUM) = 0
```

```
                                então
                                    NI := NI + NUM;
                            fimse;
                    fimse;
            fimenquanto;
fim.
```

Comentário: Foi atribuído o valor unitário a variável NUM apenas para inicializar a estrutura enquanto. Com a condição frac(NUM) = 0 somente números inteiros serão somados como ímpares.

---

12)

```
início
    cadeia: LETRA;
    inteiro: VOGAIS;
    repita
        leia (LETRA);
        se LETRA = "a" ou LETRA = "e" ou LETRA = "i" ou LETRA = "o" ou
                                                    LETRA = "u"
            então
                VOGAIS := VOGAIS + 1;
        fimse;
    até LETRA = "z";
    escreva (VOGAIS);
fim.
```

---

13)

```
início
    cadeia : LETRA;
    inteiro : VOGAL, CONSOANTE;
    VOGAL := 0;
    CONSOANTE := 0;
    repita
        exiba ("Digite uma letra");
        leia (LETRA);
        se LETRA = "a" ou LETRA = "e" ou LETRA = "i" ou LETRA = "o" ou
                                                    LETRA = "u"
            então
                VOGAL := VOGAL + 1;
            senão
                CONSOANTE := CONSOANTE + 1;
        fimse;
    até LETRA = "z";
    exiba ("Você digitou ", VOGAL, " vogal(is)");
    exiba ("Você digitou", CONSOANTE, "consoante(s)");
fim.
```

---

14)

```
início
        real: H;
        inteiro: N, CONT;
        leia (N);
        H: = 0;
        CONT: = 1;
```

```
        repita
                H = H + (1 / CONT);
                CONT := CONT + 1;
        até CONT > N;
        escreva (H);
fim.
```

---

15)

```
início
        inteiro: N, FAT, CON;
        FAT := 1;
        Leia (N);
        se N = 0
                então
                        escreva (“O fatorial do número é 1”);
                senão
                        início
                                para CON de 1 até N passo 1
                                        FAT  := FAT * CON;
                                fimpara;
                                escreva (“O fatorial do número é”, FAT );
                        fim;
        fimse;
fim.
```

---

16)

```
início {algoritmo}
        módulo soma:
                início
                        real: N1, N2, SOMA;
                        SOMA := N1 + N2;
                        escreva (SOMA);
                        retorne (SOMA);
                fim;
        módulo mult:
                início
                        real: N3, N4, MULT;
                        MULT := N3 * N4;
                        escreva (MULT);
                        retorne (MULT);
                fim;
        real: N1, N2, N3, N4, SOMA, MULT;
                repita
                        leia (N1, N2);
                        leia (N3, N4);
                        módulo soma;
                        módulo mult;
                até SOMA = MULT;
fim. {algoritmo}
```

4 ESTRUTURA DE DADOS

Cada tipo de variável já estudado permite armazenar um único valor do mesmo tipo primitivo de cada vez. Retomando o nosso problema de calcular a média entre cinco notas de um determinado aluno, no qual usamos as variáveis N1, N2, N3, N4 e N5, quando as notas de um determinado aluno são inseridas, imediatamente a média á calculada e atribuída à variável MA. Ao lançarmos as notas de outro aluno, as cinco notas de aluno anterior são descartadas e ao calcularmos a média desse novo aluno a média anterior também é descartada. Se desejarmos não descartar as notas lançadas e as médias calculadas de uma turma com 5 alunos, com o que até agora estudamos, necessitaremos de 25 variáveis do tipo real. Lembre-se que uma turma pode ter bem mais do que cinco alunos!

Para viabilizar esse nosso desejo, e suprir a necessidade de muitos outros problemas, nos quais a quantidade de variáveis dos tipos primitivos estipulados até o momento exigida é muito grande, ou mesmo insuficiente, inviabilizando a algoritmização do problema, construiremos, ou melhor, estruturaremos maneiras de agrupar as variáveis dos tipos primitivos, visando suprir essa necessidade e facilitar o manuseio. Em outras palavras estruturaremos as informações, ou melhor ainda, estudaremos estrutura de dados. Resumindo, uma variável pode ser interpretada como um elemento e uma estrutura de dados como um conjunto de elementos.

As estruturas de dados recebem o nome de *variáveis compostas*, e podem ser *homogêneas*, que são aquelas nas quais as variáveis que as compõe são do mesmo tipo primitivo, formando um conjunto homogêneo de dados, e as *heterogêneas*, que são aquelas nas quais as variáveis que as compõe não são do mesmo tipo primitivo.

4.1 Variáveis compostas homogêneas

As variáveis compostas homogêneas estão divididas em dois grupos, as unidimensionais, que recebem o nome de *vetores*, e as multidimensionais, que recebem o nome de *matrizes*.

4.1.1 Vetores

A sintática da sua declaração é:

tipo IDENTIFICADOR = vetor [LI . . LF] de <tipo primitivo>;

IDENTIFICADOR: lista de variáveis;

onde:

IDENTIFICADOR é o nome associado ao vetor criado,

LI é o limite inicial do vetor,

LF é o limite final do vetor,

<tipo primitivo> representa qualquer um dos tipos primitivos anteriormente definidos, e lista de variáveis é a lista dos elementos, separados por vírgulas, que terão o mesmo tipo denotado por IDENTIFICADOR.

Observamos que LI e LF devem ser obrigatoriamente constantes inteiras tais que LI < LF e como as posições são identificadas a partir de LI, com incrementos unitários, até LF, o número de elementos do vetor será dado por LF – LI + 1.

Visto que os vetores são compostos por diversas variáveis e, como podem existir muitos vetores do mesmo tipo, torna-se necessário determinar primeiro qual vetor contém o dado desejado e depois especificar em qual posição este se encontra.

O nome do vetor é determinado por meio do identificador que foi utilizado na lista de variáveis, e a posição, por meio da constante,

expressão aritmética ou variável que estiver dentro dos colchetes, também denominada *índice*.

Exemplo: Para estruturar, ou simplesmente criar, um vetor, por exemplo para guardar as cinco médias do nosso exemplo acima comentado, fazemos:

tipo CLASSE = vetor [0 . . 4] de reais;

CLASSE: VETORMEDIA;.

Fazendo isso na verdade criamos uma estrutura de dados equivalente a cinco variáveis do tipo real. Neste vetor VETORMEDIA temos: VETORMEDIA[0], VETORMEDIA[1], VETORMEDIA[2], VETORMEDIA[3] e VETORMEDIA[4].

Não necessariamente precisamos utilizar todas as posições de um vetor. Após isolar elementos do vetor, poderemos manipulá-los através de qualquer operação de entrada, saída ou atribuição, e as demais permanecem como estão. Para manipular alguns elementos devemos selecioná-los e então realizar com esse elemento o desejado. Se escrevemos VETORMEDIA[1] := 7, considerando o vetor criado no exemplo logo acima, estamos atribuindo o valor 7 ao elemento VETORMEDIA[1] do vetor VETORMEDIA. Para percorrer todo o vetor VETORMEDIA usamos uma estrutura de repetição, como faremos no exemplo abaixo.

Exemplo: Agora construiremos o algoritmo que calcula a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, e emite uma avaliação quanto à aprovação, neste caso, obtida atingindo-se média superior ou igual a seis, com a informação que provém do caso falso da condição MA >= 6, ou seja, a reprovação do aluno, considerando uma turma de 5 alunos, de forma que as médias calculadas não sejam descartadas, e ao final sejam exibidas novamente.

```
início {algoritmo}
        real: N1, N2, N3, N4, N5, ACM, MEDIATUR;
        inteiro: CON;
        tipo CLASSE = vetor [0 . . 4] de reais;
        CLASSE: VETORMEDIA;
        CON := 0;
        ACM := 0;
        repita
                leia (N1, N2, N3, N4, N5);
                VETORMEDIA[CON] := (N1 + N2 + N3 + N4 + N5) / 5;
                escreva (VETORMEDIA[CON]);
                se VETORMEDIA[CON] >= 6
                        então
                                escreva (“Aluno aprovado!”);
                        senão
                                escreva (“Aluno reprovado!”);
                fimse;
                ACM := ACM + VETORMEDIA[CON] ;
                CON := CON + 1;
        até CON = 5;
        CON := 0;
        MEDIATUR := ACM / 5;
        escreva (MEDIATUR);
        Para CON de 0 até 4 passo 1 faça
                escreva (VETORMEDIA[CON];
        fimpara;
fim. {algoritmo}
```

Em Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	inclua biblioteca Util
	funcao inicio()
	{
		inteiro CON
		real N1, N2, N3, N4, N5, ACM, MEDIATUR
		real vetormedia[5]
		CON = 0
		ACM = 0
		faca
		{
			escreva ("digite N1 = ")
			leia (N1)
			escreva ("digite N2 = ")
			leia (N2)
			escreva ("digite N3 = ")
			leia (N3)
			escreva ("digite N4 = ")
			leia (N4)
			escreva ("digite N5 = ")
			leia (N5)
			vetormedia[CON] = (N1 + N2 + N3 + N4 + N5) / 5
			se (vetormedia[CON] >= 6)
			{
				escreva ("\nAluno aprovado! \nA sua
media e = ", mat.arredondar(vetormedia[CON], 2))
```

```
                    }
                    senao
                    {
                        escreva ("\nAluno reprovado! \nA sua
media e = ", mat.arredondar(vetormedia[CON], 2))
                    }
                    Util.aguarde(2000)
                    limpa()
                    ACM = ACM + vetormedia[CON]
                    CON = CON + 1
                }
                enquanto (CON < 5)
                CON = 0
                MEDIATUR = ACM / 5
                escreva ("\nA media geral da turma e = ", mat.
arredondar(MEDIATUR, 2))
                Util.aguarde(2000)
                escreva ("\nAs medias da turma são:")
                para (CON = 0; CON < 5; CON++)
                {
                    escreva ("\n" ,mat.arredondar(vetormedia[CON], 2))
                }
        }
}
```

No Portugol Studio ao criar o vetor VETORMEDIA[5], cria-se automaticamente um vetor com cinco posições, numeradas de 0 a 4:

VETORMEDIA[0]	VETORMEDIA[1]	VETORMEDIA[2]	VETORMEDIA[3]	VETORMEDIA[4]

Este algoritmo não despreza nenhuma das 5 médias calculadas, mas não permite, após o cálculo das cinco médias, saber a partir de quais notas cada uma dessas médias foi calculada, pois as cinco notas de cada um dos cinco alunos foram inseridas utilizando-se as mesmas variáveis N1, N2, N3, N4 e N5. Como é claro que não desejamos criar vinte e cinco variáveis com esta finalidade, vamos, antes de resolver este problema, estudar as variáveis compostas multidimensionais, ou seja, as matrizes.

4.1.2 Matrizes

A sintática da sua declaração é:

tipo IDENTIFICADOR = matriz [LI1 . . LF1, LI2 . . LF2, ..., LIn . . LFn] de <tipo primitivo>;

IDENTIFICADOR: lista de variáveis;

onde:

IDENTIFICADOR é o nome associado à matriz criada,

LI1 . . LF1, LI2 . . LF2, ..., LIn . . LFn são os limites, inicial e final, dos intervalos de variação dos índices da variável, onde cada par de limites está associado a um índice, <tipo primitivo> representa qualquer um dos tipos primitivos anteriormente definidos, e lista de variáveis é a lista dos elementos, separados por vírgulas, que terão o mesmo tipo denotado por IDENTIFICADOR.

Observamos que LIi e LFi devem ser obrigatoriamente constantes inteiras tais que LIi < LFi, para i: 1, 2, ..., n, e, como as posições são identificadas a partir de LIi, com incrementos unitários, até LFi, para i: 1, 2, ..., n, o número de elementos é igual ao produto do número de elementos de cada dimensão:

(LF1 – LI1 + 1) * (LF2 – LI2 + 1) * ... * (LFn – Lin + 1).

Visto que as matizes são compostas por diversas variáveis e, como podem existir muitas matrizes do mesmo tipo, torna-se necessário determinar primeiro qual matriz contém o dado desejado e depois especificar em qual posição este se encontra.

O nome da matriz é determinado por meio do identificador que foi utilizado na lista de variáveis, e a posição, por meio de constantes, expressões aritméticas ou variáveis que estiverem dentro dos colchetes, também denominadas *índice*s. Como temos n limites iniciais ou n limites finais isso nos indica que a nossa matriz é n-dimensional ou de dimensão n, e assim n necessitaremos de n índices.

Exemplo: Para estruturar, ou simplesmente criar, uma matriz, por exemplo para guardar cinco notas de cinco alunos fazemos:

tipo CLASSE1 = matriz [0 . . 4, 0 . . 4] de reais;

CLASSE1: NOTAS;.

Fazendo isso na verdade criamos uma estrutura de dados equivalente a vinte e cinco variáveis do tipo real. Nesta matriz NOTAS temos:

NOTAS[0,0]	NOTAS[0,1]	NOTAS[0,2]	NOTAS[0,3]	NOTAS[0,4]
NOTAS[1,0]	NOTAS[1,1]	NOTAS[1,2]	NOTAS[1,3]	NOTAS[1,4]
NOTAS[2,0]	NOTAS[2,1]	NOTAS[2,2]	NOTAS[2,3]	NOTAS[2,4]
NOTAS[3,0]	NOTAS[3,1]	NOTAS[3,2]	NOTAS[3,3]	NOTAS[3,4]
NOTAS[4,0]	NOTAS[4,1]	NOTAS[4,2]	NOTAS[4,3]	NOTAS[4,4]

Após isolar elementos da matriz, poderemos manipulá-los através de qualquer operação de entrada, saída ou atribuição, e as demais permanecem como estão. Para manipular alguns elementos devemos selecioná-los e então realizar com esse elemento o desejado. Se escrevemos NOTAS[1,1] := 7, considerando a matriz criada no exemplo logo

acima, estamos atribuindo o valor 7 ao elemento NOTAS[1,1] da matriz NOTAS, o que indica que a primeira nota do primeiro aluno foi sete. Para percorrer toda a matriz usamos estruturas de repetição, como faremos no próximo exemplo.

A representação acima apresentada de uma matriz é muito útil para organizarmos o nosso raciocínio, mas limita-se ao caso bidimensional, e lembre-se que nossa variável matriz pode ser n-dimensional, com n > 2.

Exemplo: Agora construiremos o algoritmo que calcula a média aritmética entre cinco notas quaisquer fornecidas pelo usuário, e emite uma avaliação quanto à aprovação, neste caso, obtida atingindo-se média superior ou igual a seis, com a informação que provém do caso falso da condição MA >= 6, ou seja, a reprovação do aluno, considerando uma turma de cinco alunos, de forma que as médias e as notas de cada um dos alunos não sejam descartadas, e sejam exibidas ao final.

```
início {algoritmo}
inteiro: I, J
real: MEDIATUR, ACM;
tipo CLASSE = vetor [0 . . 4] de reais;
CLASSE: VETORMEDIA;
tipo CLASSE1 = matriz [0 . . 4, 0 . . 4] de reais;
CLASSE1: NOTAS;
ACM := 0;
Para I de 0 até 4 passo 1 faça
        VETORMEDIA[I] := 0
        Para J de 0 até 4 passo 1 faça
```

```
                leia (NOTAS[I, J]);
                VETORMEDIA[I] := VETORMEDIA[I]+NOTAS[I, J]
        fimpara;
        VETORMEDIA[I] := VETORMEDIA[I] / 5;
        escreva (VETORMEDIA[I])
        se VETORMEDIA[I] >= 6
                então
                        escreva ("Aluno aprovado!");
                senão
                        escreva ("Aluno reprovado!");
        fimse;
fimpara;
Para I de 0 até 4 passo 1 faça
        ACM = ACM + VETORMEDIA[I]
fimpara;
MEDIATUR := ACM / 5;
Para I de 0 até 4 passo 1 faça
        Para J de 0 até 4 passo 1 faça
        escreva (NOTAS[I, J])
        fimpara;
        escreva (VETORMEDIA[I];
fimpara;
escreva (MEDIATUR);
fim. {algoritmo}
```

Em Portugol Studio temos:

```
programa
{
	inclua biblioteca Matematica --> mat
	inclua biblioteca Util
	funcao inicio()
	{
		inteiro I, J
		real MEDIATUR, ACM
		real VETORMEDIA[5]
		real NOTAS[5][5]
		ACM = 0
		para (I = 0; I < 5; I++)
		{
			VETORMEDIA[I] = 0
			para (J = 0; J < 5; J++)
			{
				escreva ("digite a nota ")
				escreva (I,",",J," = ")
				leia(NOTAS[I][J])
				VETORMEDIA[I] = VETORMEDIA[I]
+ NOTAS[I][J]
			}
			VETORMEDIA[I] = VETORMEDIA[I] / 5
			escreva ("\n")
			limpa()
		}
		para (I = 0; I < 5; I++)
		{
			se (VETORMEDIA[I] >= 6)
			{
```

```
                                        escreva ("\n Aluno ",I+1," aprovado! A
sua media e = ", mat.arredondar(VETORMEDIA[I], 2))
                                }
                                senao
                                {
                                        escreva ("\n Aluno ",I+1," reprovado! A
sua media e = ", mat.arredondar(VETORMEDIA[I], 2))
                                }
                                escreva("\n")
                        }
                        para (I = 0; I < 5; I++)
                        {
                                ACM = ACM + VETORMEDIA[I]
                        }
                        MEDIATUR = ACM / 5
                        escreva("\n","As notas e medias de cada aluno são:","\n")
                        para (I = 0; I < 5; I++)
                        {
                                para (J = 0; J < 5; J++)
                                {
                                        escreva("[",mat.arredondar(NOTAS[I]
[J], 2),"] ")
                                }
                                escreva ("media ")
                                escreva (I," = ")
                                escreva (mat.arredondar(VETORMEDIA[I], 2))
                                escreva(" ")
                                escreva ("\n")
                        }
                        escreva ("\nA media da turma e = ", mat.
arredondar(MEDIATUR, 2))
                }
}
```

No Portugol Studio ao criar a matriz NOTAS[5][5], cria-se automaticamente uma matriz com vinte e cinco posições, tais que para melhor compreensão e raciocínio podemos imaginar como:

NOTAS[0,0]	NOTAS[0,1]	NOTAS[0,2]	NOTAS[0,3]	NOTAS[0,4]
NOTAS[1,0]	NOTAS[1,1]	NOTAS[1,2]	NOTAS[1,3]	NOTAS[1,4]
NOTAS[2,0]	NOTAS[2,1]	NOTAS[2,2]	NOTAS[2,3]	NOTAS[2,4]
NOTAS[3,0]	NOTAS[3,1]	NOTAS[3,2]	NOTAS[3,3]	NOTAS[3,4]
NOTAS[4,0]	NOTAS[4,1]	NOTAS[4,2]	NOTAS[4,3]	NOTAS[4,4]

Caso o leitor prefira não inicializar a contagem de posição com zero, podemos criar uma matriz NOTAS[6][6], com trinta e seis posições, tais que para melhor compreensão e raciocínio podemos imaginar como:

NOTAS[0,0]	NOTAS[0,1]	NOTAS[0,2]	NOTAS[0,3]	NOTAS[0,4]	NOTAS[0,5]
NOTAS[1,0]	NOTAS[1,1]	NOTAS[1,2]	NOTAS[1,3]	NOTAS[1,4]	NOTAS[1,5]
NOTAS[2,0]	NOTAS[2,1]	NOTAS[2,2]	NOTAS[2,3]	NOTAS[2,4]	NOTAS[2,5]
NOTAS[3,0]	NOTAS[3,1]	NOTAS[3,2]	NOTAS[3,3]	NOTAS[3,4]	NOTAS[3,5]
NOTAS[4,0]	NOTAS[4,1]	NOTAS[4,2]	NOTAS[4,3]	NOTAS[4,4]	NOTAS[4,5]
NOTAS[5,0]	NOTAS[5,1]	NOTAS[5,2]	NOTAS[5,3]	NOTAS[5,4]	NOTAS[5,5]

e então desprezar a posição 0 de linha e a posição 0 de coluna, e desta forma utilizará apenas os elementos:

NOTAS[1,1]	NOTAS[1,2]	NOTAS[1,3]	NOTAS[1,4]	NOTAS[1,5]
NOTAS[2,1]	NOTAS[2,2]	NOTAS[2,3]	NOTAS[2,4]	NOTAS[2,5]
NOTAS[3,1]	NOTAS[3,2]	NOTAS[3,3]	NOTAS[3,4]	NOTAS[3,5]
NOTAS[4,1]	NOTAS[4,2]	NOTAS[4,3]	NOTAS[4,4]	NOTAS[4,5]
NOTAS[5,1]	NOTAS[5,2]	NOTAS[5,3]	NOTAS[5,4]	NOTAS[5,5]

Desta forma os elementos NOTAS[0,0], NOTAS[0,1], NOTAS[0,2], NOTAS[0,3], NOTAS[0,4], NOTAS[0,5], NOTAS[1,0], NOTAS[2,0], NOTAS[3,0], NOTAS[4,0] e NOTAS[5,0] serão criados, mas não utilizados.

Ideia análoga vale na criação de vetores, ao criar no Portugol Studio o vetor VETORMEDIA[5] cria-se automaticamente um vetor com cinco posições, numeradas de 0 a 4:

VETORMEDIA[0]	VETORMEDIA[1]	VETORMEDIA[2]	VETORMEDIA[3]	VETORMEDIA[4]

mas caso o leitor prefira não inicializar a contagem de posição com zero, podemos criar um vetor VETORMEDIA[6] com seis posições, numeradas de 0 a 5:

VETORMEDIA[0]	VETORMEDIA[1]	VETORMEDIA[2]	VETORMEDIA[3]	VETORMEDIA[4]	VETORMEDIA[5]

e então desprezar o elemento VETORMEDIA[0].

Esta é uma ocorrência muito comum a vários ambientes de programação.

Neste problema votamos a apresentar as entradas e saídas de uma execução deste programa. Após rodar o programa, para as notas a seguir:

digite a nota 0,0 = 1
digite a nota 0,1 = 2
digite a nota 0,2 = 3
digite a nota 0,3 = 4
digite a nota 0,4 = 5
digite a nota 1,0 = 6
digite a nota 1,1 = 7
digite a nota 1,2 = 8
digite a nota 1,3 = 9
digite a nota 1,4 = 10
digite a nota 2,0 = 1

digite a nota 2,1 = 3
digite a nota 2,2 = 5
digite a nota 2,3 = 7
digite a nota 2,4 = 9
digite a nota 3,0 = 2
digite a nota 3,1 = 4
digite a nota 3,2 = 6
digite a nota 3,3 = 8
digite a nota 3,4 = 10
digite a nota 4,0 = 10
digite a nota 4,1 = 10
digite a nota 4,2 = 10
digite a nota 4,3 = 10
digite a nota 4,4 = 10
obtemos como respostas:
“Aluno 1 reprovado! A sua media e = 3.0”
“Aluno 2 aprovado! A sua media e = 8.0”
“Aluno 3 reprovado! A sua media e = 5.0”
“Aluno 4 aprovado! A sua media e = 6.0”
“Aluno 5 aprovado! A sua media e = 10.0”
“As notas e medias de cada aluno são:
[1.0] [2.0] [3.0] [4.0] [5.0] media 0 = 3.0
[6.0] [7.0] [8.0] [9.0] [10.0] media 1 = 8.0
[1.0] [3.0] [5.0] [7.0] [9.0] media 2 = 5.0
[2.0] [4.0] [6.0] [8.0] [10.0] media 3 = 6.0
[10.0] [10.0] [10.0] [10.0] [10.0] media 4 = 10.0”
“A media da turma e = 6.4”.

De forma geral, para utilizar um vetor inserimos um único laço de repetição, fazendo com que haja variação de seu índice. Em uma matriz n dimensional possuímos n índices, e assim faz-se necessário a utilização de n laços de repetição.

Na verdade, uma estrutura composta multidimensional é um conjunto de vetores que são determinados por cada intervalo que compõe o tipo matriz.

As matrizes mais utilizadas são as bidimensionais, devido a sua direta relação com muitas aplicações, como, por exemplo, tabelas, que como vimos exigem dois laços de repetição.

Não necessariamente precisamos utilizar todas as posições de um vetor ou de uma matriz. No exemplo abaixo, utilizaremos apenas parte dos vetores definidos.

Exemplo: Considere o problema de classificar uma lista de atletas dada a idade de cada um, cuja quantidade é desconhecida e o termino da lista é indicado pela digitação da palavra FIM como nome de atleta.

Vamos definir dois vetores, um para receber os nomes e outro para receber as idades. Como neste caso não sabemos a quantidade de atletas, devemos definir vetores que suportem qualquer lista de atletas que possa vir a ser analisada pelo nosso algoritmo, sendo que neste caso julgamos que esta não ultrapassará cem elementos.

```
início {algoritmo}
    tipo NAME = vetor [0 . . 99] de cadeia;
    NAME: NOME;
    tipo ID = vetor [0 . . 99] de inteiro;
    ID: IDADE;
    inteiro: I,
    I := 0;
    NOME[I] := “ENTRE”;
    enquanto NOME[I] <> “FIM” faça
        leia (IDADE[I], NOME[I]);
        se NOME[I] <> “FIM”
            então
                se IDADE[I] >= 7 e IDADE[I] <= 11
                    então
                        escreva (“dente de leite”);
                    senão
                        se IDADE[I] >= 12 e IDADE[I] <= 15
                            então
                                escreva (“infantil”);
                            senão
                                se IDADE[I] >= 16 e
IDADE[I] <= 18
                                    então
                                        escreva (“juvenil”);
                                    senão
                                        se IDADE[I] >=
19 e IDADE[I] <= 25
                                            então
                                                escreva
(“profissional”);
```

```
                                                    senão
                                                        escreva
("não classificado");
                                                fimse;
                                            fimse;
                                        fimse;
                    fimse;
                    I := I +1;
            fimse;
    fimenquanto;
fim. {algoritmo}
```

4.2 Variáveis compostas heterogêneas

Observe que no nosso último exemplo armazenamos os nomes dos atletas em um vetor e a idade em outro. Retomemos agora o problema de calcular a média geral e a média de cada aluno de uma turma com cinco alunos considerando que devemos inserir também os nomes dos cinco alunos. Este problema também não pode ser tratado definindo uma única matriz para receber o nome, as notas e as médias de cada aluno, pois para tanto necessitamos das variáveis compostas heterogêneas, as quais ainda não estudamos. Vamos então tratá-lo definindo um vetor para receber os nomes, outro para receber as médias e uma matriz para receber a notas.

```
início {algoritmo}
        inteiro: I, J;
        real: MEDIATUR, ACM;
        tipo NAME = vetor [0 . . 4] de cadeia;
        NAME: NOME;
```

```
tipo CLASSE1 = matriz [0 . . 4, 0 . . 4] de reais;
CLASSE1: NOTAS;
tipo CLASSE = vetor [0 . . 4] de reais;
CLASSE: VETORMEDIA;
ACM := 0;
Para I de 0 até 4 passo 1 faça
        VETORMEDIA[I] := 0;
        leia (NOME[I]);
        Para J de 0 até 4 passo 1 faça
                leia (NOTAS[I, J]);
                VETORMEDIA[I] := VETORMEDIA[I]
+ NOTAS[I, J];
        fimpara;
        VETORMEDIA[I] := VETORMEDIA[I] / 5;
fimpara;
Para I de 0 até 4 passo 1 faça
        se VETORMEDIA[I] >= 6
                então
                        escreva (“Aluno aprovado!”);
                senão
                        escreva (“Aluno reprovado!”);
        fimse;
        ACM := ACM + VETORMEDIA[I];
fimpara;
MEDIATUR := ACM / 5;
escreva (MEDIATUR);
Para I de 0 até 4 passo 1 faça
        escreva (NOME[I]);
```

```
                    Para J de 0 até 4 passo 1 faça
                        escreva (NOTAS[I, J]);
                    fimpara;
                    escreva (VETORMEDIA[I]);
                fimpara;
            fim. {algoritmo}
```

A sua versão em Portugol Studio é:

```
programa
{
    inclua biblioteca Matematica --> mat
    inclua biblioteca Util
    funcao inicio()
    {
        inteiro I, J
        real MEDIATUR, ACM
        cadeia NOME[5]
        real NOTAS[5][5]
        real VETORMEDIA[5]
        ACM = 0
        para (I = 0; I < 5; I++)
        {
            VETORMEDIA[I] = 0
            escreva ("\ndigite o nome o aluno ",I,":")
            leia(NOME[I])
            para (J = 0; J < 5; J++)
            {
```

```
                                escreva ("digite a nota ")
                                escreva (I,",",J," = ")
                                leia(NOTAS[I][J])
                                VETORMEDIA[I] = VETORMEDIA[I]
+ NOTAS[I][J]
                        }
                        VETORMEDIA[I] = VETORMEDIA[I] / 5
                        escreva ("\n")
                        limpa()
                }
                para (I = 0; I < 5; I++)
                {
                        se (VETORMEDIA[I] >= 6)
                        {
                                escreva ("\n Aluno ",NOME[I],"
aprovado! A sua media e = ", mat.arredondar(VETORMEDIA[I], 2))
                        }
                        senao
                        {
                                escreva ("\n Aluno ",NOME[I],"
reprovado! A sua media e = ", mat.arredondar(VETORMEDIA[I], 2))
                        }
                        escreva("\n")
                        ACM = ACM + VETORMEDIA[I]
                }
                MEDIATUR = ACM / 5
                escreva("\n","O nome, as suas notas e media de cada
aluno são:","\n")
```

```
                para (I = 0; I < 5; I++)
                {
                        escreva("\nAluno ",NOME[I],": ")
                        escreva("\n")
                        para (J = 0; J < 5; J++)
                        {
                                escreva("[",mat.arredondar(NOTAS[I]
[J], 2),"] ")
                        }
                        escreva ("media ")
                        escreva (I," = ")
                        escreva (mat.arredondar(VETORMEDIA[I], 2))
                        escreva(" ")
                        escreva ("\n")
                }
                escreva ("\nA media da turma e = ", mat.
arredondar(MEDIATUR, 2))
        }
}
```

Rodando este programa para as mesmas notas do último exemplo rodado no Portugol Studio, agora atribuídas aos alunos João, Antonio, Pedro, Paulo e Junior, nesta ordem, obtemos as repostas:

“Aluno João reprovado! A sua media e = 3.0”

“Aluno Antonio aprovado! A sua media e = 8.0”

“Aluno Pedro reprovado! A sua media e = 5.0”

“Aluno Paulo aprovado! A sua media e = 6.0”

“Aluno Junior aprovado! A sua media e = 10.0”

“O nome, as suas notas e media de cada aluno são:

Aluno João:

[1.0] [2.0] [3.0] [4.0] [5.0] media 0 = 3.0

Aluno Antonio:

[6.0] [7.0] [8.0] [9.0] [10.0] media 1 = 8.0

Aluno Pedro:

[1.0] [3.0] [5.0] [7.0] [9.0] media 2 = 5.0

Aluno Paulo:

[2.0] [4.0] [6.0] [8.0] [10.0] media 3 = 6.0

Aluno Junior:

[10.0] [10.0] [10.0] [10.0] [10.0] media 4 = 10.0”

“A media da turma e = 6.4”.

A desvantagem do tratamento dado neste último exemplo é que não há relação entre as variáveis que guardam informações a respeito do mesmo aluno. Por exemplo, se desejarmos saber as notas e a média de um aluno devemos, na variável NOME procurar a sua posição, e daí na variável NOTAS e na variável VETORMEDIA verificar suas notas e sua média.

Por isso estudaremos as variáveis compostas heterogêneas, que recebem o nome de *registro*.

A sintática da sua declaração é:

tipo IDENTIFICADOR = registro

<tipo primitivo 1>**:** campo 1**;**

<tipo primitivo 2>**:** campo 2**;**

⋮

<tipo primitivo n>**:** campo n**;**

IDENTIFICADOR: lista de variáveis;

onde:

IDENTIFICADOR é o nome associado ao registro criado,

<tipo primitivo1>, ..., <tipo primitivo n> representam qualquer um dos tipos primitivos anteriormente definidos,

campo 1, ..., campo n são os nomes associados a cada campo do registro, e lista de variáveis é a lista dos elementos separados por vírgulas, que terão o mesmo tipo denotado por IDENTIFICADOR.

Exemplo: Para criar um registro que deverá receber o nome e a nota de um grupo de alunos fazemos:

tipo NOMENOTA = registro

cadeia: NOME;

real: NOTA;

NOMENOTA: NONNOT;

A forma visual abaixo é bastante esclarecedora.

NONNOT	NOME
	NOTA

Como um registro contém mais de uma informação, para acessar um campo específico é necessário indicar o nome da variável e o nome do campo desejado, separados por um ponto. No exemplo acima, para acessar a nota do aluno fazemos:

NONNOT.NOTA,

e para acessar o nome do aluno fazemos:

NONNOT.NOME,

e daí, por exemplo, fazendo:

NONNOT.NOTA := 7,5;

NONNOTNOME := “João”;

nosso primeiro registro NONNOT deve ser assim compreendido visualmente:

NONNOT	João
	7,5

Se desejássemos que a informação acima fosse inserida pelo usuário usaríamos o comando leia e de forma análogo procedemos com o comando escreva.

Se precisamos de todas as informações contidas no registro acessamos genericamente o registro, sem especificar o campo. Fazendo:

leia (NONNOT);

estamos acessando o nome e a nota de todo registro NONNOT.

Nos registros podem estar presentes também campos que são variáveis compostas, ou seja, formados por outros tipos construídos.

Neste caso definimos a(s) variável(is) composta(s) a ser(em) utilizada(s) e depois definimos o registro. A sintática da sua declaração é:

declaração da variável composta 1;
declaração da variável composta 2;
⋮
declaração da variável composta m;
tipo IDENTIFICADOR = registro
<tipo primitivo 1>**:** campo 1**;**
<tipo primitivo 2>**:** campo 2**;**
⋮
<tipo primitivo n>**:** campo n**;**
variálvel composta 1: lista de variáveis 1;
variálvel composta 2: lista de variáveis 2;
⋮
variálvel composta m: lista de variáveis m;
IDENTIFICADOR: lista de variáveis;

onde

declaração da variável composta 1, ..., declaração da variável composta m são as declarações da m variáveis compostas a serem utilizadas no registro,

IDENTIFICADOR é o nome associado ao registro criado,

<tipo primitivo1>, ..., <tipo primitivo n> representam qualquer um dos tipos primitivos anteriormente definidos,

campo 1, ..., campo n são os nomes associados a cada campo do registro,

variável composta 1, ..., variável composta n são as variáveis compostas criadas na declaração das m variáveis compostas,

lista de variáveis 1, ..., lista de variáveis m são as listas dos elementos, separados por vírgulas, que terão o mesmo tipo das m variáveis compostas, e

lista de variáveis é a lista dos elementos, separados por vírgulas, que terão o mesmo tipo denotado por IDENTIFICADOR

Exemplo: Para criar um registro que deverá receber o nome e as cinco notas de um aluno fazemos:

```
tipo NOTAS = vetor [0 . . 4] de reais;
        NOTAS: NOTA;
        tipo NOMENOTA = registro
        cadeia: NOME;
        NOTAS: NOTA;
NOMENOTA: NONNOT;
```

A forma visual abaixo é bastante esclarecedora.

NONNOT	NOME
	NOTA[0]
	NOTA[1]
	NOTA[2]
	NOTA[3]
	NOTA[4]

na qual, se o primeiro aluno é João que tem notas, 7.5, 6, 8, 7 e 10 ficamos com a seguinte representação visual:

NONNOT	João
	7,5
	6
	8
	7
	10

Exemplo: Para criar um registro que deverá receber o nome e as cinco notas de um grupo com cinco alunos fazemos:

```
tipo NAME = vetor [0 . . 4] de cadeia;
        NAME: NOME;
tipo NOTAS = matriz [0 . . 4, 0 . . 4] de reais;
        NOTAS: NOTA;
tipo MEDIA = vetor [0 . . 4] de reais;
        MEDIA: VETORMEDIA;
tipo NOMENOTA = registro
        NAME: NOME;
        NOTAS: NOTA;
        MEDIA: VETORMEDIA;
NOMENOTA: NONNOT;
```

cuja representação visual é:

NONNOT	NOME[0]	NOME[1]	NOME[2]	NOME[3]	NOME[4]
	NOTA[0,0]	NOTA[1,0]	NOTA[2,0]	NOTA[3,0]	NOTA[4,0]
	NOTA[0,1]	NOTA[1,1]	NOTA[2,1]	NOTA[3,1]	NOTA[4,1]
	NOTA[0,2]	NOTA[1,2]	NOTA[2,2]	NOTA[3,2]	NOTA[4,2]
	NOTA[0,3]	NOTA[1,3]	NOTA[2,3]	NOTA[3,3]	NOTA[4,3]
	NOTA[0,4]	NOTA[1,4]	NOTA[2,4]	NOTA[3,4]	NOTA[4,4]
	VETORMEDIA [0]	VETORMEDIA [1]	VETORMEDIA [2]	VETORMEDIA [3]	VETORMEDIA [4]

Agora vamos retomar o problema de calcular a média geral e a média de cada aluno de uma turma com cinco alunos considerando que devemos inserir também os nomes dos cinco alunos, utilizando uma variável do tipo registro.

```
início {algoritmo}
        real: ACM, MEDIATUR;
        inteiro: I, J;
        tipo NAME = vetor [0 . . 4] de cadeia;
                NAME: NOME;
        tipo NOTAS = matriz [0 . . 4, 0 . . 4] de reais;
                NOTAS: NOTA;
        tipo MEDIA = vetor [0 . . 4] de reais;
                MEDIA: MEDIATUR;
        tipo NOMENOTA = registro
                NAME: NOME;
                NOTAS: NOTA;
                MEDIA: MEDIATUR;
        NOMENOTA: NONNOT;
        ACM := 0;
```

```
            Para I de 0 até 4 passo 1 faça
                  VETORMEDIA[I] := 0;
                  leia (NOME[I]);
                  Para J de 0 até 4 passo 1 faça
                        leia (NOTA[I,J]);
                        VETORMEDIA[I] := VETORMEDIA[I]
+ NOTA[I, J];
                  fimpara;
                  VETORMEDIA[I] := VETORMEDIA[I] / 5;
            fimpara;
            Para I de 0 até 4 passo 1 faça
                  se VETORMEDIA[I] >= 6
                        então
                              escreva (“Aluno aprovado!”);
                        senão
                              escreva (“Aluno reprovado!”);
                  fimse;
                  ACM := ACM + VETORMEDIA[I];
            fimpara;
            MEDIATUR := ACM / 5;
            escreva (MEDIATUR);
            Para I de 0 até 4 passo 1 faça
                  escreva (NOME[I]);
                  Para J de 0 até 4 passo 1 faça
                        escreva (NOTA[I, J]);
                  fimpara;
                  escreva(VETORMEDIA[I]);
            fimpara;
      fim. {algoritmo}
```

A diferença entre o algoritmo acima e o construído no início deste tópico se dá apenas na forma de como a matriz das notas é percorrido, além da utilização de registro, é claro. Assim propomos ao leitor o desafio de modificar a definição do registro deste último algoritmo de forma que esta diferença seja eliminada.

Observe que agora todas as informações dessa turma de cinco alunos estão armazenadas no registro NONNOT. Assim se desejamos a segunda nota do aluno Pedro, sabendo que a posição dele na turma é a terceira, basta acessarmos:

NONNOT.NOTA[3,2],

e assim acessamos a posição [3,2] do campo NOTA do registro NONNOT. Para acessar a média desse aluno fazemos:

NONNOT.MA[3].

No parágrafo acima estipulamos ser conhecida a posição do aluno na turma, e assim surge a importância da ordenação de elementos. Este problema não foi por nós tratado neste texto, pois conforme já mencionamos, deixamos de lado os problemas para focar na compreensão das estruturas, mas este e outros algoritmos importantes serão estudados através dos exercícios propostos.

Finalizando, na Ciência da Computação, mais do que em qualquer outra, a aprendizagem jamais é concluída, e assim propomos leitor o estudo de *arquivos*, que pode ser definido, de maneira simplista, como um conjunto de registros, e daí o nosso leitor poderá resolver o problema de uma escola, na qual, para cada classe teremos um registro realizado pelo nosso último algoritmo.

4.3 Exercícios propostos

1) Faça um algoritmo que receba um vetor unidimensional formado por 100 números reais.

2) Construa um algoritmo que receba uma matriz quadrada bidimensional, formada por 100 números reais, dispostos em 10 colunas e 10 linhas.

3) Construa um algoritmo que receba uma matriz quadrada tridimensional, formada por 1000 números reais.

4) Construa um algoritmo que receba uma matriz quadrada bidimensional, formada por 100 números reais, dispostos em 10 colunas e 10 linhas, e calcule a sua transposta.

5) Construa um algoritmo que receba uma matriz quadrada bidimensional, formada por 100 números reais, dispostos em 10 colunas e 10 linhas, e calcule o seu maior elemento.

6) Construa um algoritmo que receba uma matriz quadrada bidimensional, formada por 100 números reais, dispostos em 10 colunas e 10 linhas, e multiplique-a por uma constante a ser definida pelo usuário.

7) Construa um algoritmo que receba duas matrizes quadradas bidimensionais, formadas por 100 números reais cada, dispostos em 10 colunas e 10 linhas, e some-as.

8) Construa um algoritmo que multiplique duas matrizes quadradas bidimensionais, formadas por 100 números reais cada, dispostos em 10 colunas e 10 linhas.

9) Construa um algoritmo que receba uma lista de nomes e ao final escreva a lista dos nomes digitados acompanhados pela sua posição na lista. O final da lista será determinado pelo nome “fim”.

10) Construa um algoritmo que ordene, em ordem crescente, um vetor com 100 elementos.

11) Construa um algoritmo que ordene, em ordem decrescente, um vetor com 100 elementos.

12) Construa um algoritmo que calcule a norma linha de uma matriz quadrada de ordem inferior ou igual a 100.

13) Construa um algoritmo que calcule a norma coluna de uma matriz quadrada de ordem inferior ou igual a 100.

4.4 Gabarito dos exercícios propostos

1)

```
início
        tipo VET = vetor  [0 . . 99] de reais;
        VET: V;
        inteiro: I;
        para I de 0 até 99 passo 1 faça
                leia (V[I]);
        fimpara;
fim.
```

---

2)

```
início
        tipo MAT = matriz [0 . . 9, 0 . . 9] de reais;
        MAT: M;
        inteiro: I, J;
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        leia (M[I,J]);
                fimpara;
        fimpara;
fim.
```

---

3)

```
início
        tipo MAT = matriz [0 . . 9, 0 . . 9, 0 . . 9] de reais;
        MAT: M;
```

```
        inteiro: I, J, K;
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        para K de 0 até 9 passo 1 faça
                                leia (M[I,J,K]);
                        fimpara;
                fimpara;
        fimpara;
fim.
```

---

4)

```
início
        tipo MAT = matriz [1 . . 10, 1 . . 10] de reais;
        MAT: M, TR;
        inteiro: I, J;
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        leia (M[I,J]);
                fimpara;
        fimpara;
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        TR[I,J] := M[J,I];
                        escreva (TR[I,J]);
                fimpara;
        fimpara;
fim.
```

5)

```
início
        tipo MAT = matriz [0 . . 9, 0 . . 9] de reais;
        MAT: M;
        inteiro: I, J;
        real: MAIOR;
        leia (M[0,0]);
        MAIOR := M[0,0];
        para I de 0 até 9 passo 1 faça;
                para J de 0 até 9 passo 1 faça;
                        se (I <> 0) e (J <> 0)
                                então
                                        início
                                                leia (M[I,J]);
                                                se M[I,J] > MAIOR
                                                        então
                                                                MAIOR := M[i,j];
                                                fimse;
                                        fim;
                        fimse;
                fimpara;
        fimpara;
        escreva (MAIOR);
fim.
```

Comentário: Foi usado uma seleção simples apenas para não ler novamente o elemento M[0,0].

6)

```
início
        tipo MAT = matriz [0 . . 9, 0 . . 9] de reais;
        MAT: M, N;
        inteiro: I, J;
        real: NUM;
        leia (NUM);
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        leia (M[I,J]);
                        N[I,J] := NUM * M[I,J];
                        escreva (N[I,J]);
                fimpara;
        fimpara;
fim.
```

---

7)

```
início
        tipo MAT = matriz [0 . . 9, 0 . . 9] de reais;
        MAT: M, N, SOMA;
        inteiro: I, J;
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        leia (M[I,J], N[I,J]);
                        SOMA[I,J] := M[I,J] + N[I,J];
                        escreva (SOMA[I,J]);
                fimpara;
        fimpara;
fim.
```

8)

```
início
        tipo MAT = matriz [0 . . 9, 0 . . 9] de reais;
        MAT: M, N, MULT;
        inteiro: I, J, K;
        real: PROD;
        para I de 0 até 9 passo 1 faça
                para J de 0 até 9 passo 1 faça
                        leia (M[I,J], N[I,J]);
                        Para K de 0 até 9 passo 1 faça
                                PROD := M[I,K] * N[K,J];
                                SOMA := SOMA + PROD;
                        fimpara;
                        MULT[I,J] := SOMA;
                fimpara;
        fimpara;
fim.
```

---

9)

```
início
        tipo VET = vetor [0 .. 999] de cadeia;
        VET: NOME;
        inteiro: I, END;
        I = 0;
        repita
                leia (nome[I]);
                I := I + 1;
        até nome[I] = “fim”;
        END := I – 1;
        para I de 0 até END passo 1 faça
```

```
            escreva (I, NOME[I]);
        fimpara;
fim.
```

10)

```
início
        tipo VET = vetor [0 . . 99] de reais
        VET: V;
        real: AUX;
        inteiro: I, J;
        para I de 0 até 99 passo 1 faça
            leia (V[I]);
        fimpara;
        para I de 0 até 99 passo 1 faça
                para J de I + 1 até 99 passo 1 faça
                        se V[I] > V[J]
                                        então
                                                início
                                                        AUX := V[I];
                                                        V[I] := V[J];
                                                        V [J] :=AUX;
                                                fim;
                        fimse;
                fimpara;
        fimpara;
        para I de 0 até 99 passo 1 faça
                escreva (V[I]);
        fimpara;
fim.
```

11)

```
início
        tipo VET = vetor [0 . . 99] de reais
        VET: V;
        real: AUX;
        inteiro: I, J;
        para I de 0 até 99 passo 1 faça
            leia (V[I]);
        fimpara;
        para I de 0 até 99 passo 1 faça
                para J de I + 1 até 99 passo 1 faça
                        se V[I] < V[J]
                                        então
                                                início
                                                    AUX := V[I];
                                                   V[I] := V[J];
                                                   V [J] :=AUX;
                                                fim;
                        fimse;
                fimpara;
        fimpara;
        para I de 0 até 99 passo 1 faça
                escreva (V[I]);
        fimpara;
fim.
```

12)

```
início
        tipo MAT = matriz [0 . . 99, 0 . . 99] de reais
        MAT: A;
        tipo VET = vetor [0 . . 99]
        Vet: S;
        inteiro: I, J, M, N;
        real: M1, M2, NORMA;
        repita
                leia (M, N);
        até M >= 1 e M <= 100 e N >= 1 e N <= 100 e frac(M) = 0 e frac(N) = 0;
        para I de 0 até M – 1 passo 1 faça
                S[I] := 0;
        fimpara;
        Para I de 0 até M – 1  passo 1 faça
                Para J de 0 até N – 1  passo 1 faça
                        leia (A[I,J]);
                        S[I] := S[I] + abs(A[I,I]);
                fimpara;
        fimpara;
        NORMA := S[0];
        para I de 1 até M – 1 passo 1 faça
                se S[I] > NORMA
                                então
                                        NORMA := S[I];
                fimse;
        fimpara;
        escreva (“A NORMA LINHA É”, NORMA);
fim.
```

13)

```
início
        tipo MAT = matriz [0 . . 99, 0 . . 99] de reais
        MAT: A;
        tipo VET = vetor [0, . . 99]
        Vet: S;
        inteiro: I, J, M, N;
        real: M1, M2, NORMA;
        repita
                leia (M, N);
        até M >= 1 e M <= 100 e N >= 1 e N <= 100 e frac(M) = 0 e frac(N) = 0;
        para J de 0 até N – 1 passo 1 faça
                S[J] := 0;
        fimpara;
        Para I de 0 até M – 1 passo 1 faça
                Para J de 0 até N – 1 passo 1 faça
                        leia (A[I,J]);
                        S[J] := S[J] + abs(A[I,I]);
                fimpara;
        fimpara;
        NORMA := S[0];
        para J de 1 até N – 1 passo 1 faça
                se S[J] > NORMA
                                então
                                        NORMA := S[J];
                fimse;
        fimpara;
        escreva (“A NORMA COLUNA É”, NORMA);
fim.
```

5 CONSTRUÇÃO E SIMPLIFICAÇÃO DE EXPRESSÕES

Vamos aprender a obter uma fórmula do Cálculo Proposicional a partir da sua tabela verdade, uma vez que muitas situações reais são melhores interpretadas quando descritas por uma tabela verdade. Como exemplo considere que um determinado comando C, depende da verificação de três condições, condição 1, condição 2 e condição 3, e que este comando deve ser executado sempre que pelo menos duas das três condições forem verdadeiras. Esta situação é melhor compreendida através da tabela verdade:

Condição 1	Condição 2	Condição 3	C
V	V	V	V
V	V	F	V
V	F	V	V
V	F	F	F
F	F	F	F
F	F	V	F
F	V	F	F
F	V	V	V

Para obtermos uma fórmula que represente uma tabela verdade temos dois processos, extremamente simples.

Primeiro processo: Seja A a fórmula procurada. Para construir a fórmula desejada, escrevemos inicialmente uma fórmula para cada linha da tabela dada que contenha o valor lógico verdadeiro (V) na coluna dos vl(A). Estas fórmulas parciais são formadas pela conjunção das variáveis proposicionais negadas ou não, usando-se a própria variável proposicional quando o seu valor verdade for verdadeiro (V), ou a sua negação quando o seu valor verdade for falso (F). Finalmente, obtém-se a fórmula procurada reunindo-se todas as fórmulas parciais, através da

disjunção inclusiva. As fórmulas obtidas por este processo possuem uma forma especial, denominada Forma Normal Disjuntiva Padronizada.

Segundo processo: Seja A a fórmula procurada. Para construir a fórmula desejada, escrevemos inicialmente uma fórmula para cada linha da tabela dada que contenha o valor lógico falso (F) na coluna dos vl(A). Estas fórmulas parciais são formadas pela disjunção inclusiva das variáveis proposicionais negadas ou não, usando-se a própria variável proposicional quando o seu valor verdade for falso (F), ou a sua negação quando o seu valor verdade for verdadeiro (V). Finalmente, obtém-se a fórmula procurada reunindo-se todas as fórmulas parciais, através da conjunção. As fórmulas obtidas por este processo possuem uma forma especial, denominada Forma Normal Conjuntiva Padronizada.

Exemplo: Consideremos a seguinte tabela verdade:

p	q	A
V	V	F
V	F	V
F	V	V
F	F	F

Usando-se a primeira regra proposta, obtemos que as fórmulas parciais são (p e não q) para a segunda linha e (não p e q) para a terceira linha. Finalmente obtemos que a fórmula A procurada é ((p e não q) ou (não p e q)).

Usando-se a segunda regra proposta, obtemos que as fórmulas parciais são (não p ou não q) para a primeira linha e (p ou q) para a quarta linha. Finalmente obtemos que a fórmula A procurada é ((não p ou não q) e (p ou q)).

Assim para o nosso problema quando um determinado comando C, depende da verificação de três condições, condição 1, condição 2 e condição 3, e que este comando deve ser executado sempre que pelo menos duas das três condições forem verdadeiras, o qual foi compreendido através da tabela verdade:

Condição 1	Condição 2	Condição 3	C
V	V	V	V
V	V	F	V
V	F	V	V
V	F	F	F
F	F	F	F
F	F	V	F
F	V	F	F
F	V	V	V

Usando-se o primeiro processo para a construção de fórmulas, obtemos:

(<condição 1> e <condição 2> e <condição 3>) ou (<condição 1> e <condição 2> e não <condição 3>) ou (<condição 1> e não <condição 2> e <condição 3>) ou (não <condição 1> e <condição 2> e <condição 3>).

O que estudamos até o momento nos permite o estudo das mais diversas linguagens, tornando-nos programadores. No entanto muitas expressões, como as obtidas pelos dois processos de construção de fórmulas, as quais integrarão condições em nossos algoritmos, podem e devem ser simplificadas, quando nos tornamos então bons programadores. Em tais simplificações utilizamos as equivalências lógicas.

Definição: Dizemos que as fórmulas A e B são *logicamente equivalentes* se e somente se A e B sempre tomam o mesmo valor lógico para

qualquer atribuição de valores verdade feita às variáveis proposicionais de A e B. Isto é, a última coluna das tabelas verdade de A e de B são, necessariamente, as mesmas.

Notação: Se A e B são duas fórmulas logicamente equivalentes, indica-se por A eq B ou A ⇔ B.

Exemplo: Seja A a fórmula:

não (p ou q),

e seja B a fórmula:

(não p e não q).

As subfórmulas da fórmula não (p ou q) são: p, q, (p ou q) e não (p ou q).

As subfórmulas da fórmula (não p e não q) são: p, q, não p, não q e (não p e não q).

Daí as respectivas tabelas verdade são:

p	q	(p ou q)	não (p ou q)	não p	não q	(não p e não q)
V	V	V	**F**	F	F	**F**
V	F	V	**F**	F	V	**F**
F	V	V	**F**	V	F	**F**
F	F	F	**V**	V	V	**V**

Logo A eq B, isto é, as fórmulas A e B são logicamente equivalentes, isto é:

não (p ou q) eq (não p e não q).

Observe que utilizamos a mesma atribuição de valores verdades para p e q na construção das tabelas verdade de ambas as fórmulas.

Algumas equivalências lógicas se destacam pela grande utilização, principalmente na compreensão e na simplificação de expressões, e por isto recebem o nome de *equivalências notáveis*:

1) Dupla Negação - DN:

não não A eq A

2) Leis Idempotentes - LI:

i) (A ou A) eq A

ii) (A e A) eq A

3) Leis Comutativas - CL:

i) (A ou B) eq (B ou A)

ii) (A e B) eq (B e A)

4) Leis Associativas - AL:

i) ((A ou B) ou C) eq (A ou (B ou C))

ii) ((A e B) e C) eq (A e (B e C))

5) Leis de De Morgan - DL:

i) não (A ou B) eq (não A e não B)

ii) não (A e B) eq (não A ou não B)

6) Leis Distributivas - LD:

i) (A ou (B e C)) eq ((A ou B) e (A ou C))

ii) (A e (B ou C)) eq ((A e B) ou (A e C))

7) **Elementos Neutros - EN:**

A ou F eq A

A e V eq A

8) Elementos Complementares - EC:

A ou não A eq V

A e não A eq F

nas quais A, B, e C são fórmulas quaisquer e V e F são tautologias e contradições, respectivamente.

A comprovação de cada uma dessas equivalências notáveis pode ser feita através da construção das respectivas tabelas verdade, lembrando que como A, B, e C são fórmulas quaisquer elas podem assumir apenas os valores verdade verdadeiro (V) ou falso (F).

Assim, podemos aplicar equivalências lógicas a qualquer fórmula, sempre com o objetivo de simplifica-la ou melhorar a sua legibilidade. Para isto, é interessante observar que, sendo A uma fórmula qualquer, C uma contradição e T uma tautologia então (A e C) eq C e (A ou T) eq T.

Exemplo:

1) Consideremos agora a tabela verdade:

p	q	r	A
V	V	V	V
V	V	F	V
V	F	V	V
V	F	F	V
F	F	F	F
F	F	V	F
F	V	F	F
F	V	V	F

Usando-se a primeira regra dada acima, obtemos que as fórmulas parciais são (p e q e r), (p e q e não r), (p e não q e r) e (p e não q e não r) para a primeira, segunda, terceira e quarta linhas respectivamente. Finalmente obtemos que a fórmula A procurada é:

(p e q e r) ou (p e q e não r) ou (p e não q e r) ou (p e não q e não r).

Utilizando as equivalências notáveis "Leis Distributivas" e "Leis Associativas", obtemos:

p e ((q e r) ou (q e não r) ou (não q e r) ou (não q e não r)).

Utilizando novamente a equivalência notável "Leis Distributivas" obtemos:

p e ((q e (r ou não r)) ou (não q e (r ou não r)).

Como (r ou não r) eq T, em que T é tautologia, temos:

p e ((q e T) ou (não q e T)).

Agora:

(q e T) eq q

e

(não q e T) eq não q,

e assim temos:

p e (q ou não q).

Como (q ou não q) eq T, temos:

p e T eq p.

Assim, demonstramos que a fórmula:

(p e q e r) ou (p e q e não r) ou (p e não q e r) ou (p e não q e não r).

é logicamente equivalente à fórmula:

p,

muito mais simples.

2) Considere a tabela verdade:

p	q	r	A
V	V	V	V
V	V	F	V
V	F	V	V
V	F	F	F
F	F	F	F
F	F	V	V
F	V	F	V
F	V	V	V

Para esta tabela, o segundo procedimento é mais adequado, pois existem apenas dois valores lógicos F. Utilizando-se então este procedimento, obtemos as seguintes fórmulas parciais para a quarta e para a quinta linha, respectivamente:

(não p ou q ou r) e (p ou q ou r).

Assim a fórmula final é:

(não p ou q ou r) e (p ou q ou r).

Utilizando a equivalência notável “Leis Comutativas”, obtemos:

(q ou r ou não p) e (q ou r ou p).

Pela equivalência notável “Leis Distributivas”, obtemos:

(q ou r) ou (não p e p).

Como (não p e p) eq C, em que C é contradição, obtemos:

(q ou r) ou C eq (q ou r).

Assim obtemos a fórmula:

(q ou r),

que é muito mais simples que a fórmula:

(não p ou q ou r) e (p ou q ou r).

As fórmulas obtidas pelos dois processos de construção de fórmulas são, obviamente, logicamente equivalentes, e uma pode ser obtida a partir da outra pela aplicação conveniente de equivalências lógicas.

Exemplo: Consideremos a seguinte tabela verdade:

p	q	A
V	V	V
V	F	F
F	V	F
F	F	V

Usando-se a segunda regra proposta, obtemos que as fórmulas parciais são (não p ou q) para a segunda linha e (p ou não q) para a terceira linha. Finalmente obtemos que a fórmula A procurada é ((não p ou q) e (p ou não q)).

Pela equivalência notável, “Leis Distributivas”, obtemos:

((não p ou q) e p) ou ((não p ou q) e não q)).

Pela equivalência notável, “Leis Comutativas”, obtemos:

(p e (não p ou q)) ou (não q e (não p ou q)).

E novamente pela equivalência notável “Leis Distributivas”, obtemos:

((p e não p) ou (p e q)) ou ((não q e não p) ou (não q e q)).

Como:

(p e não p) eq C

e

(não q e q) eq C,

em que C é contradição, temos:

(C ou (p e q)) ou ((não q e não p) ou C).

E, finalmente, como:

(C ou (p e q)) eq (p e q)

e

((não q e não p) ou C) eq (não q e não p)

temos:

(p e q) ou (não q e não p),

a qual é a fórmula que obtemos se aplicarmos o primeiro processo a tabela verdade deste exemplo.

Retomemos agora o problema quando um determinado comando C, depende da verificação de três condições, condição 1, condição 2 e condição 3, e que este comando deve ser executado sempre que pelo menos duas das três condições forem verdadeiras, o qual foi compreendido através da tabela verdade para o qual, usando-se o primeiro processo para a construção de fórmulas, obtemos:

(<condição 1> e <condição 2> e <condição 3>) ou (<condição 1> e <condição 2> e não <condição 3>) ou (<condição 1> e não <condição 2> e <condição 3>) ou (não <condição 1> e <condição 2> e <condição 3>),

A qual pela equivalência notável "Leis Associativas" é o mesmo que:

(<condição 1> e <condição 2> e <condição 3>) ou (((<condição 1> e <condição 2> e não <condição 3>) ou (<condição 1> e não <condição 2> e <condição 3>)) ou (não <condição 1> e <condição 2> e <condição 3>)).

Daí, pelas equivalências notáveis "Leis idempotentes" e "Leis Distributivas" obtemos:

(((<condição 1> e <condição 2> e <condição 3>) ou (<condição 1> e <condição 2> e não <condição 3>)) ou ((<condição 1> e <condição 2> e <condição 3>) ou (<condição 1> e não <condição 2> e <condição 3>))) ou ((<condição 1> e <condição 2> e <condição 3>) ou (não <condição 1> e <condição 2> e <condição 3>))).

Pela equivalências notáveis "Leis Distributivas" e "Leis Comutativas" obtemos:

(((<condição 1> e <condição 2>) e (<condição 3> ou não <condição 3>)) ou ((<condição 1> e <condição 3>) e (<condição 2> ou não <condição 2>)) ou ((<condição 2> e <condição 3>) e (<condição 1> ou não <condição 1>))).

Agora: (<condição 3> ou não <condição 3>) eq (<condição 2> ou não <condição 2>) eq (<condição 1> ou não <condição 1>) eq V, onde V representa uma tautologia. Logo temos:

(((<condição 1> e <condição 2>) e V) ou ((<condição 1> e <condição 3>) e V) ou ((<condição 2> e <condição 3>) e V)).

Também (((<condição 1> e <condição 2>) e V) eq (<condição 1> e <condição 2>), ((<condição 1> e <condição 3>) e V) eq (<condição 1> e <condição 3>) e ((<condição 2> e <condição 3>) e V)) eq (<condição 2> e <condição 3>). Logo obtemos:

(<condição 1> e <condição 2>) ou (<condição 1> e <condição 3>) ou (<condição 2> e <condição 3>).

Veja que assim procedendo o seu algoritmo ganhará muita legibilidade, e você será um bom programador. Por isso para programar com eficiência, além de conhecimentos sobre as estruturas da lógica de programação, o domínio da Lógica Matemática também é essencial.

Recomendamos também o estudo dos Mapas de karnaugh[1] para a simplificação de fórmulas.

[1] Recomendamos para o estudo dos Diagramas de Karnaugh o livro *Fundamentos Matemáticos para Computação: Lógica e Álgebra*, deste autor, ou o livro *Lógica e Álgebra de Boole*, do autor Jacob Daghian.

5.1 Exercícios propostos

1) Em cada item abaixo, verifique se ocorre equivalência lógica entre as fórmulas dadas:

a) (p e (p ou q)) e p

b) p e (p xou q)

c) (p ou (p e q)) e p

d) (p ou (q ou r)) e ((q ou (p ou r)) ou p)

e) (p ou q) e não (não p e não q)

f) (p e q) e não (não p ou não q)

2) Um sistema de escolha está sendo desenvolvido. Em uma de suas etapas a escolha será realizada por três avaliadores e será efetivada apenas quando somente um dos avaliadores aprova-lo.

a) Construa uma tabela verdade que represente a situação.

b) Encontre uma fórmula que represente a tabela verdade do item a).

3) Um sistema de avaliação de produtos está sendo desenvolvido. Em uma de suas etapas a escolha será realizada por três avaliadores e será efetivada apenas quando somente dois dos avaliadores aprova-lo.

a) Construa uma tabela verdade que represente a situação.

b) Encontre uma fórmula que represente a tabela verdade do item a).

5.2 Gabarito dos exercícios propostos

1)

a) Sim.

b) Não.

c) Sim.

d) Sim.

e) Sim.

f) Sim.

---

2)

a)

aval 1	aval 2	aval 3	escolha
V	V	V	F
V	V	F	F
V	F	V	F
V	F	F	V
F	F	F	F
F	F	V	V
F	V	F	V
F	V	V	F

b)

(aval 1 e não aval 2 e não aval 3) ou (não aval 1 e não aval 2 e aval 3) ou (não aval 1 e aval 2 e não aval 3)

---

3)

a)

aval 1	aval 2	aval 3	escolha
V	V	V	F
V	V	F	V
V	F	V	V
V	F	F	F
F	F	F	F
F	F	V	F
F	V	F	F
F	V	V	V

b)

(aval 1 e aval 2) ou (aval 1 e aval 3) ou (aval 2 e aval 3)

BIBLIOGRAFIA RECOMENDADA

ALENCAR FILHO, E. de. **Iniciação à lógica matemática**. São Paulo: Nobel, 1995.

COPI, I. M. **Introdução à lógica**. São Paulo: Mestre Jou, l981.

CURY, M. X. **Introdução à lógica**. São Paulo: Érica, 1996.

DAGHLIAN, J. **Lógica e álgebra de Boole**. São Paulo: Atlas, 1995.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de programação***:* a construção de algoritmos e estrutura de dados. São Paulo: Pearson, 2005.

FURLAN, M. A. *et al*. **Algoritmos e lógica de programação**. São Paulo: Cengage, 2012.

MANZANO, J. A. N. G; OLIVEIRA, J. F. **Algoritmos:** lógica para desenvolvimento de programação de computadores. São Paulo: Érica, 2019.

MENDELSON, E. **Álgebra booleana e circuitos de chaveamento**. São Paulo: McGraw-Hill, 1977.

PIRES, A. de A. **Fundamentos matemáticos para computação:** lógica e álgebra. Sorocaba: Eduniso, 2017.

SALVETTI, D. D.; BARBOSA, L. M. **Algoritmos**. São Paulo: Makron Books, 1998.