

Algoritmos

Prof. Marcelo F. Siqueira
Ed. Prof. Umberto S. Costa, Richard Bonichon

17 de março de 2015

SUMÁRIO

1	Introdução	7
1.1	Algoritmos e problemas computacionais	7
1.2	Representação de algoritmos	9
1.3	Resolução de Problemas	10
1.4	Computadores	10
1.5	Exercícios propostos	12
2	Tipos de Dados e Variáveis	15
2.1	Tipos de dados	15
2.2	Variáveis	16
2.3	Exemplos	17
2.4	Nomes de variáveis	18
2.5	Exercícios Resolvidos	19
2.6	Exercícios propostos	19
3	Entrada e Saída	22
3.1	Instruções de entrada e saída	22
3.2	A estrutura de um algoritmo	23
3.3	Exercícios resolvidos	24
3.4	Exercícios propostos	25
4	Expressões Aritméticas – Parte 1	27
4.1	Operadores aritméticos	27
4.2	Precedência de operadores	27
4.3	Alteração de prioridades	28
4.4	A instrução de atribuição	31
4.5	Exercícios resolvidos	31
4.6	Exercícios propostos	32
5	Expressões Aritméticas – Parte 2	33
5.1	Operadores aritméticos sobre os reais	33
5.2	Regras semânticas	34
5.3	Um algoritmo envolvendo constantes e variáveis reais	35
5.4	Exercícios resolvidos	36
5.5	Exercícios propostos	37
6	Expressões Relacionais	38
6.1	Operadores relacionais	38

6.2	Relações e expressões aritméticas	39
6.3	Relações envolvendo tipos não inteiros	39
6.4	Exercícios resolvidos	41
6.5	Exercícios propostos	42
7	Estruturas Condicionais - Parte 1	43
7.1	Motivação	43
7.2	Comando <u>se-entao-senao-fimse</u>	44
7.3	Aninhamento de comandos condicionais	45
7.4	Comando <u>se-entao-fimse</u>	47
7.5	Exercícios resolvidos	47
7.6	Exercícios propostos	47
8	Expressões Lógicas	52
8.1	Lógica proposicional	52
8.2	Proposições compostas	53
8.3	Operadores lógicos	56
8.4	Exercícios resolvidos	58
8.5	Exercícios propostos	58
9	Estruturas Condicionais - Parte 2	60
9.1	Usando proposições compostas	60
9.2	Troca de conteúdo entre duas variáveis	61
9.3	O comando <u>escolha</u>	63
9.4	Exercícios propostos	63
10	Estruturas de Repetição - Parte 1	67
10.1	O comando <u>enquanto-faca-fimenquanto</u>	67
10.2	Exemplos	69
10.3	Exercícios propostos	70
11	Estruturas de Repetição 2	74
11.1	A seqüência de Fibonacci	74
11.2	Inversão da ordem dos dígitos de um número	75
11.3	Teste de primalidade	77
11.4	Exercícios propostos	78
12	Estruturas de Repetição 3	82
12.1	O cálculo da média aritmética	82
12.2	O maior elemento de uma seqüência	83
12.3	Os múltiplos de posição na seqüência	84
12.4	Exercícios Propostos	85
13	Estruturas de Repetição 4	90
13.1	O laço repita	90
13.2	Exemplo	90
13.3	Laço <u>repita</u> versus laço <u>enquanto</u>	92
13.4	Exercícios propostos	93

14 Vetores	97
14.1 Motivação	97
14.2 Definição e manipulação de variáveis	98
14.3 O cálculo do desvio padrão	100
14.4 O comando <u>para-faca-fimpara</u>	101
14.5 Exercícios propostos	102
15 Aninhamento de Laços	107
15.1 Laços aninhados	107
15.2 Ordenação	109
15.3 Exercícios propostos	114
16 Matrizes - Parte 1	119
16.1 Definição e Manipulação de Matrizes	119
16.2 Exemplos	121
16.2.1 Soma de duas matrizes	121
16.2.2 Cálculo de norma matricial	123
16.2.3 Cálculo da matriz transposta	123
16.3 Exercícios propostos	124
17 Matrizes 2	128
17.1 Mais exemplos	128
17.1.1 Multiplicação de duas matrizes	128
17.1.2 Quadrado mágico	130
17.2 Exercícios propostos	133
18 Modularização - Parte 1	136
18.1 O Quê e Por Quê?	136
18.2 Componentes de um módulo	137
18.3 Funções e Procedimentos	137
18.4 Chamando Funções e Procedimentos	140
18.5 Passagem de Parâmetros	141
18.6 Escopo de Dados e Código	142
18.7 Exercícios propostos	143

LISTA DE ALGORITMOS

1.1	Algoritmo para calcular a área de um quadrado.	9
1.2	Programa em linguagem C para calcular a área de um quadrado.	11
3.1	Algoritmo para ler um numero inteiro e escrever o valor lido.	24
3.2	Algoritmo para ler um numero e uma palavra e escrever ambos.	25
4.1	Cálculo de algumas expressões aritméticas com números inteiros	29
4.2	Cálculo de expressões aritméticas com variáveis inteiras	30
4.3	Cálculo do quadrado da soma de dois inteiros	32
5.1	Cálculo de expressões aritméticas com variáveis reais	34
5.2	Cálculo do volume da esfera	36
5.3	Cálculo da área de um retângulo	37
7.1	Cálculo (incompleto) da mais alta de duas pessoas.	43
7.2	Algoritmo para determinar a mais alta de duas pessoas.	45
7.3	Algoritmo para determinar a mais alta de duas pessoas.	49
7.4	Algoritmo para determinar situação escolar de aluno.	50
7.5	Algoritmo para calcular salário líquido mensal de um funcionário.	50
7.6	Soma dígitos de centena e milhar	51
7.7	Algoritmo “Misterioso”. Qual é a saída dele?	51
9.1	Cálculo da média harmônica ponderada de três notas	61
9.2	Escrita de dois números em ordem não-decrescente	65
9.3	Escrita de dois números em ordem não-decrescente v2	65
9.4	Classificar atletas por categorias de idade	66
10.1	Escrita dos inteiros de 1 a n	72
10.2	Soma dos inteiros de um dado intervalo	73
10.3	Fatorial de um inteiro não negativo	73
11.1	Cálculo n -ésimo termo da sequência de Fibonacci	80
11.2	Inverter a ordem dos dígitos de um número	80
11.3	Determinar se um número é primo ou composto	81
12.1	Cálculo da média aritmética de n números reais	83
12.2	Cálculo do maior de uma sequência de n números reais	84
12.3	Escrever os números múltiplos de uma sequência de inteiros	89
13.1	Algoritmo para escrever os inteiros de 1 a n	91

13.2	Algoritmo para calcular e escrever o resultado da série finita $\sum_{i=1}^n x^i/i$.	96
13.3	Algoritmo para somar os inteiros de um dado intervalo.	96
14.1	Algoritmo para calcular a média aritmética de cinco notas.	104
14.2	Algoritmo para calcular a média aritmética de cinco notas usando vetor.	105
14.3	Algoritmo para calcular desvio padrão.	105
14.4	Algoritmo para calcular desvio padrão com laço para.	106
15.1	Algoritmo para contar número de elementos comuns.	110
15.2	Ordenação de uma sequência de inteiros usando o método de seleção.	118
16.1	Primeira parte do algoritmo para somar duas matrizes.	122
16.2	Segunda parte do algoritmo para somar duas matrizes.	122
16.3	Terceira parte do algoritmo para somar duas matrizes.	123
16.4	Norma de soma máxima de linha	126
16.5	Cálculo da matriz transposta.	127
17.1	Multiplicação de 2 matrizes	135
18.1	Cálculo do Quadrado de um Número.	147

INTRODUÇÃO

1.1 Algoritmos e problemas computacionais

A palavra algoritmo tem origem no sobrenome do matemático, astrônomo, geólogo, geógrafo e autor persa [Mohammed ibn-Musa al-Khwarizmi](#), que viveu entre 780 e 850 d.C [2]. No século XII, sua obra sobre numerais indianos foi traduzida para o latim e apresentou a notação posicional decimal para o Mundo Ocidental. Ele também apresentou a primeira solução sistemática das equações lineares e quadráticas e é considerado um dos fundadores da Álgebra [1]. O radical de algarismo e algoritmo vem de *Algoritmi*, a forma latina do sobrenome al-Khwarizmi.

Há tantas definições diferentes para o termo algoritmo quanto autores escrevendo sobre elas. No entanto, todas essas definições concordam que um **algoritmo** é uma seqüência finita de instruções, bem definidas e não-ambíguas, para resolver um dado problema. Cada instrução de um algoritmo deve ser executada por um período de tempo finito. Em geral, a definição de algoritmo é ilustrada através de qualquer processo “mecânico”, tal como uma receita culinária ou a troca de pneu de um automóvel. No entanto, aqui, estamos interessados em **algoritmos computacionais**, ou seja, algoritmos que descrevem instruções a serem executadas por computador.

Para exemplificar o que entendemos por “algoritmo computacional”, vamos considerar o problema de se calcular a área A de um quadrado Q de lado l . No ensino médio, aprendemos que

$$A = l^2. \tag{1.1}$$

Então, dado o comprimento l dos lados do quadrado Q , uma forma de resolver o problema é usar a fórmula acima, ou seja, multiplicar o valor de l por ele próprio. Note que a fórmula em (1.1) pode ser usada para calcular a área A de *qualquer* quadrado. Tudo o que precisamos saber para utilizar a fórmula para obter a área A de qualquer quadrado é o comprimento l do lado do quadrado.

Suponha, agora, que devemos escrever uma seqüência de instruções para calcular A , a qual deve ser seguida por uma criança que sabe multiplicar dois números e sempre faz isso de forma correta. A criança deve nos solicitar o comprimento l do lado do quadrado Q e, depois de calcular A , ela deve nos informar o valor de A obtido por ela. Este valor deve ser escrito, pela criança, em uma folha de papel usando lápis ou caneta. Uma possível seqüência de instruções é

1. Solicite o comprimento l do lado do quadrado.
2. Multiplique l por l
3. Escreva o resultado da multiplicação na folha de papel.

A seqüência acima, por mais simples que seja, ilustra todos os elementos essenciais da definição

de algoritmo. A seqüência é finita (há apenas três instruções). Cada uma delas está bem definida, não deixa nenhuma dúvida sobre o que deve ser feito e pode ser realizada em um período finito de tempo. Há ainda, no nosso exemplo, dois componentes fundamentais dos algoritmos computacionais: entrada e saída. A **entrada** consiste do conjunto de dados que deve ser fornecido ao algoritmo, enquanto a **saída** é o conjunto de dados produzidos pelo algoritmo. No exemplo acima, a entrada é o comprimento l do lado do quadrado e a saída é a área A do quadrado.

O exemplo acima também ilustra uma característica importante dos algoritmos computacionais: eles são *agentes transformadores* de dados de entrada em dados de saída. Este processo de transformação é comumente denominado **processamento**. Daí, o termo **processamento de dados**. No exemplo, o valor de l foi “processado” para gerar o valor de A . O processamento, neste caso, consiste na multiplicação de l por l . É importante perceber que *um algoritmo não é a solução de um problema, mas sim um processo para se obter a solução*. Um problema que pode ser resolvido por um algoritmo computacional é denominado **problema computacional**. Isto é, um problema computacional é aquele que pode ser resolvido por um computador através de um algoritmo.



Figura 1.1: A transformação de entrada em saída por um algoritmo.

Um problema computacional possui várias ocorrências (ou instâncias). Uma **ocorrência** de um problema computacional é uma instância qualquer da entrada do problema. Por exemplo, no problema do cálculo da área do quadrado, sabemos que a entrada do problema é o comprimento l do lado do quadrado. Então, qualquer valor válido para l , ou seja, qualquer real positivo, é uma instância do problema, por exemplo $l = 2$. Um algoritmo deve sempre ser construído para resolver *todas* as possíveis ocorrências de um problema. O algoritmo do nosso exemplo contém instruções para o cálculo da área A do quadrado para qualquer valor de l dado.

Um algoritmo é dito **correto** se ele sempre termina e produz a resposta correta para *todas* as ocorrências de um dado problema. O algoritmo do nosso exemplo, portanto, é correto. Obviamente, a “criança” que executa as instruções deve saber cumpri-las de forma correta. Na nossa analogia, a criança é o computador. Os computadores que usamos na prática sempre cumprem as instruções que lhes damos de forma correta. Se houver algum erro na tentativa de solucionar um problema computacional através de um computador, este erro está nas próprias instruções que lhe demos. O computador apenas executa, fielmente, as instruções que lhe damos.

Quando começamos a construir algoritmos, umas das habilidades mais importantes é aquela de identificar qual é a entrada e qual é a saída do algoritmo *a partir da descrição (ou declaração) do problema*. Você deve procurar adquirir e aprimorar esta habilidade sem se preocupar em como resolver o problema ou escrever o algoritmo. Muitas vezes, o enunciado de um problema computacional descreve, de forma explícita, qual é a entrada e qual é a saída do algoritmo:

Escreva um algoritmo que recebe, como entrada, a base e a altura de um retângulo qualquer e produz como saída a área do retângulo.

Note que o próprio enunciado torna evidente que a entrada consiste dos comprimentos da base e altura de um retângulo e a saída, da área do retângulo. No entanto, o problema poderia ser descrito como

Dados a base e a altura de um retângulo qualquer, calcule e escreva a área do retângulo.

Note que, em ambos os casos, não é preciso saber calcular a área do retângulo ou escrever o algoritmo que faz este cálculo para determinar que a entrada do algoritmo é a base e a altura de um retângulo e a saída, ou seja, o dado que o algoritmo deve produzir como resposta, é a área do retângulo.

1.2 Representação de algoritmos

Em geral, algoritmos são descritos através de uma linguagem que se assemelha àquela que usamos para nos comunicar. O vocabulário das linguagens destinadas à descrição de algoritmos é extremamente pequeno quando comparado ao das linguagens coloquiais, mas rico o suficiente para resolvermos uma gama enorme de problemas computacionais. Aqui, descreveremos algoritmos com uma linguagem conhecida como **Portugol**, que é utilizada pela ferramenta **VISUALG**, que será utilizada na disciplina como forma de apoio ao aprendizado de algoritmos.

Por exemplo, usando a linguagem Portugol da VISUALG, o algoritmo que vimos na seção anterior para calcular a área de um quadrado a partir do comprimento de seus lados é descrito como em 1.2:

```
1 algoritmo "Area do quadrado"
2   var lado, area : real
3 inicio
4   escreva ("Entre com o comprimento dos lados do quadrado: ")
5   leia (lado)
6   area <- lado * lado
7   escreva ("A area do quadrado e: ", area)
8 fimalgoritmo
```

Algoritmo 1.1: Algoritmo para calcular a área de um quadrado.

Há uma série de detalhes sintáticos da linguagem Portugol da ferramenta VISUALG que devem ser dominados para que você possa escrever seus algoritmos usando esta linguagem. Parte do processo de aprendizado de construção de algoritmos é dedicada à familiarização com os aspectos sintáticos de alguma linguagem de descrição de algoritmos. Felizmente, as linguagens de descrição de algoritmos possuem estruturas sintáticas bastante parecidas entre si. Isto faz com que o uso de outra linguagem, após o aprendizado da primeira, seja bastante facilitado.

Uma outra forma de descrevermos algoritmos é através do uso de *fluxogramas*, que são diagramas com figuras que identificam as várias instruções do algoritmo. Nesta disciplina, também utilizaremos fluxogramas, embora a principal forma de descrição seja mesmo a linguagem Portugol.

1.3 Resolução de Problemas

Todo algoritmo está relacionado com a solução de um determinado problema computacional. Portanto, construir um algoritmo para um dado problema significa, antes de mais nada, determinar uma forma para solucionar um problema e descrevê-la como uma seqüência finita de instruções em alguma linguagem. A tarefa de encontrar a solução de um problema qualquer é, muitas vezes, realizada de forma empírica e um tanto quanto desorganizada; ocorrem vários procedimentos mentais, dos quais raramente tomamos conhecimento. A organização do processo de resolução de problemas é extremamente desejável, pois somente assim podemos verificar onde o processo não é eficiente. Identificadas as deficiências deste processo, procuramos formas de corrigi-las e, conseqüentemente, aumentamos a nossa capacidade de resolver problemas.

A capacidade para resolver problemas pode ser vista como uma habilidade a ser adquirida. Esta habilidade, como qualquer outra, pode ser obtida essencialmente pela combinação de duas partes:

- conhecimento e
- destreza.

Conhecimento é adquirido pelo estudo. Em termos de resolução de problemas, está relacionado a que táticas e estratégias usar e quando usar. Destreza é adquirida pela prática. A experiência no uso do conhecimento (prática) nos dá mais agilidade na resolução de problemas.

1.4 Computadores

O que é um computador? De acordo com o *Webster's New World Dictionary of the American Language* (segunda edição), um computador é “uma máquina eletrônica que, por meio de instruções e informações armazenadas, executa rápida e frequentemente cálculos complexos ou compila, correlaciona e seleciona dados”. Basicamente, um computador pode ser imaginado como uma máquina que manipula informação na forma de números e caracteres. A informação é denominada, de maneira geral, de **dado**. O que faz dos computadores máquinas notáveis é a extrema rapidez e precisão com que eles podem armazenar, recuperar, manipular e produzir dados.

Quando desejamos utilizar, pela primeira vez, um computador para nos auxiliar na tarefa de processamento de dados, deparamo-nos com algumas questões inerentes a este processo: “Como informamos ao computador o algoritmo que deve ser executado para obtermos o resultado desejado?”, “Como fornecemos a entrada do algoritmo?” e “Como recebemos o resultado do algoritmo?”

O ato de instruir o computador para que ele resolva um determinado problema é conhecido como **programação**. Esta tarefa nada mais é do que inserir no computador as ações do algoritmo e os dados referenciados pelas ações. Estas, quando executadas pelo computador, produzem a solução do problema. Entretanto, antes de inserir as ações e os dados no computador, devemos reescrevê-las em uma linguagem apropriada para descrever algoritmos computacionais, ou seja, em uma **linguagem de programação**. O termo **programa** é comumente empregado para designar o algoritmo em uma linguagem de programação. Entretanto, não há distinção conceitual entre algoritmo e programa¹. O Algoritmo 1.2 escrito em linguagem C é mostrado no Programa 1.4. Compare os dois!

¹Em Teoria da Computação, existe uma distinção, mas ela não será levada em conta aqui.

Cada modelo de computador possui uma linguagem de programação própria, denominada **linguagem de máquina**, e, em geral, distinta das linguagens de máquina dos demais modelos de computador. Esta é a única linguagem de programação que o computador realmente entende. No entanto, para evitar que nós tenhamos de aprender a linguagem de máquina de cada computador diferente para o qual queremos programar, muitas linguagens de programação independentes de máquina foram criadas. Se você aprende uma linguagem independente de máquina, estará apto, pelo menos em princípio, a programar qualquer computador usando essa linguagem.

As linguagens de programação independentes de máquina não são compreendidas pelos computadores. Então, para que elas possam ser úteis para nós, um programa denominado **compilador** deve estar presente no computador. Um compilador para uma determinada linguagem de programação realiza a tradução automática de um programa escrito em uma certa linguagem para um programa equivalente escrito na linguagem de máquina. Tudo que precisamos fazer para executar um programa escrito em uma linguagem de programação que não seja a linguagem de máquina do computador é *compilar* o nosso programa com um compilador específico para aquela linguagem e, em seguida, executar o programa produzido pelo compilador.

```
1  #include <stdio.h>
2  void main()
3  {
4      double lado, area ;
5      printf("Entre com o comprimento dos lados do quadrado:") ;
6      scanf("%f\n", &lado) ;
7      area = lado * lado ;
8      printf("A area do quadrado e:  %f\n", area) ;
9      return ;
10 }
```

Algoritmo 1.2: Programa em linguagem C para calcular a área de um quadrado.

Tanto o algoritmo quanto os seus dados de entrada são inseridos nos computadores por meio de equipamentos eletrônicos conhecidos como **periféricos de entrada**. O teclado e o *mouse* são exemplos de periféricos de entrada. As instruções e os dados inseridos no computador através de um periférico de entrada são armazenados em um dispositivo do computador denominado **memória** (primária ou secundária). Os dados de saída resultantes da execução do algoritmo pelo computador são apresentados também por meio de equipamentos eletrônicos denominados **periféricos de saída**. O monitor de vídeo e a impressora são exemplos de periféricos de saída.

O computador executa um determinado programa através de um dispositivo interno denominado **unidade central de processamento**, mais conhecido no mundo dos computadores pela sua abreviação em inglês: *CPU* (*Central Processing Unit*). A CPU é responsável por buscar as instruções e os dados do programa que estão armazenados na memória do computador, decodificar as instruções e executar a tarefa descrita por elas com os respectivos dados. A CPU pode ser vista como o “cérebro” do computador.

No mundo dos computadores, você ouvirá as pessoas falarem sobre *hardware* e *software*. **Hardware** se refere à máquina propriamente dita e a todos os periféricos conectados a ela. **Software** se refere aos programas que fazem a máquina realizar alguma tarefa. Muitos “pacotes” de software estão disponíveis nos dias atuais. Eles incluem processadores de texto, planilhas eletrônicas, sistemas gerenciadores de banco de dados, jogos, sistemas operacionais e compiladores. Você pode e

aprenderá a criar seus próprios softwares. Para criar software, você deverá adquirir competências e habilidades para: (1) desenvolver algoritmos para solucionar problemas computacionais e (2) usar uma linguagem de programação. Aqui, dedicamo-nos à (1).

1.5 Exercícios propostos

1. O primeiro passo no desenvolvimento de um algoritmo para um dado problema computacional é o entendimento da entrada e da saída do problema. Este entendimento deve preceder qualquer tentativa de desenvolvimento de uma solução para o problema. Neste exercício, você deve descrever qual é a entrada e qual é a saída de cada um dos problemas listados abaixo:

- a) Dado um número inteiro qualquer, calcule e escreva o antecessor e o sucessor do número dado.
- b) Dados três números reais não-negativos, calcule e escreva a média aritmética dos números dados.
- c) Dado um número real qualquer, calcule e escreva a terça parte do número dado.
- d) Dados o termo inicial e a razão de uma PA, bem como um número inteiro positivo n , calcule e escreva o valor do n -ésimo termo dessa PA.
- e) Escreva um algoritmo para ler o valor de uma temperatura em graus centrígrados e escrever a mesma temperatura em graus *Fahrenheit*. Se c é o valor da temperatura em graus centrígrados, então a temperatura, f , em *Fahrenheit* é dada por

$$f = \frac{9 \cdot c + 160}{5}.$$

- f) Chico Bento deseja calcular o saldo atual de uma de suas aplicações financeiras. Para tal, ele conhece o saldo anterior ao reajuste e sabe que este saldo foi reajustado em 1%. Escreva um algoritmo para calcular e escrever esse saldo atual.
- g) Chico Bento está preocupado com o consumo de energia de sua residência e deseja escrever um algoritmo para ajudá-lo a controlar suas despesas com energia. Chico Bento sabe que 100 quilowatts de energia equivalem a um sétimo do salário mínimo. Então, dados o valor do salário mínimo e a quantidade de quilowatts gasta na residência de Chico Bento, calcule e escreva (a) o valor em reais de cada quilowatt, (b) o valor em reais a ser pago e (c) o novo valor em reais a ser pago se Chico Bento ganhar um desconto de 10% por pagar em dia.
- h) Chico Bento possui um carro que faz, em média, 12 km com um litro de gasolina. Ele realizou uma viagem com seu carro e está interessado em saber quantos litros de combustível o carro consumiu. Para tal, ele dispõe de duas informações: o tempo gasto dirigindo e a velocidade média do carro. Escreva um algoritmo para calcular quantos litros de combustível o carro de Chico Bento consumiu.
- i) Escreva um algoritmo para ler um valor de hora, em termos de três números inteiros, *hora*, *minuto* e *segundos*, e calcular e escrever o número de segundos que se passou desde o começo do dia até o valor da hora que foi fornecido como entrada para o algoritmo.

- j) Dada uma centena, isto é, um número inteiro positivo da forma xyz tal que x é um dígito de 1 a 9 e tanto y quanto z são dígitos de 0 a 9, calcule e escreva a unidade do número dado.

Por exemplo, se o número dado é igual a 147, a solução do problema é 7. Observe que a entrada do problema consiste de um único valor, que é um número inteiro positivo representando uma centena, e não os três dígitos da centena.

2. Se você puder, escreva um algoritmo (ou seja, uma seqüência de instruções) para resolver cada um dos problemas acima. Você não precisa escrever as instruções na linguagem do VISUALG, mas sim de forma livre, como na primeira seqüência de instruções que foi dada para o problema da área do quadrado.

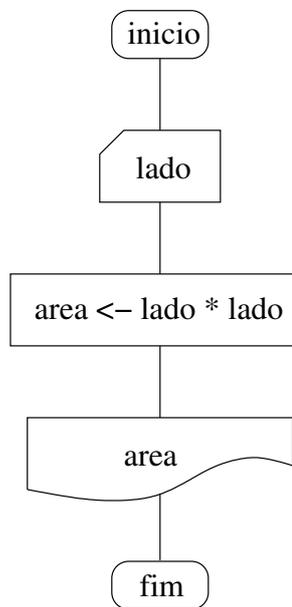


Figura 1.2: Fluxograma do algoritmo de cálculo da área do quadrado.

TIPOS DE DADOS E VARIÁVEIS

2.1 Tipos de dados

Os dados manipulados por um algoritmo podem possuir natureza distinta, isto é, podem ser números, letras, frases, etc. Dependendo da natureza de um dado, algumas operações podem ou não fazer sentido quando aplicadas a eles. Por exemplo, não faz sentido falar em somar duas letras. Para poder distinguir dados de naturezas distintas e saber quais operações podem ser realizadas com eles, os algoritmos utilizam o conceito de tipo de dados.

O **tipo de um dado** define o conjunto de valores ao qual o dado pertence, bem como o conjunto de todas as operações que podem atuar sobre qualquer valor daquele conjunto de valores. Por exemplo, como veremos mais adiante, a linguagem que utilizaremos para descrever nossos algoritmos possui o tipo de dado inteiro, que consiste no conjunto de todos os números inteiros, denotado por \mathbb{Z} , e todas as operações que podem ser aplicadas aos números inteiros (isto é, adição, subtração, multiplicação, divisão inteira e resto).

A seguir, descrevemos os tipos de dados oferecidos pela linguagem Portugol do VISUALG. Na nossa descrição, o nome de um tipo é escrito no formato tipo, assim como as demais *palavras reservadas* da linguagem Portugol. Além disso, ao definirmos um dado tipo de dados, não fornecemos uma descrição detalhada das operações que atuam sobre seus valores, pois tais operações serão objetos de estudo das próximas aulas.

- inteiro: consiste dos números inteiros e das operações de adição, subtração, multiplicação, divisão inteira e resto. Na linguagem Portugol, os números inteiros são escritos apenas como a concatenação dos dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, tal como em 5, 100 e 1678. Números negativos são representados com o sinal “-” na frente do número, tal como -23.
- real: consiste dos números reais e das operações de adição, subtração, multiplicação, divisão. Na linguagem Portugol, os números reais são caracterizados por possuírem uma *parte inteira* e uma *parte fracionária*. Por exemplo, as partes inteira e fracionária do número real 3.141596 são 3 e 141596, respectivamente. Note que um “ponto” e não uma vírgula é usado para separar as partes inteira e fracionária.

Como sabemos, os números reais incluem os números inteiros. No entanto, para evitar ambigüidades na escrita de algoritmos, assumimos que todo número escrito sem a parte fracionária é do tipo inteiro. Por exemplo, 5 e 5.0 se referem ao mesmo número (cinco), mas o primeiro é do tipo inteiro e o segundo, do tipo real. Assim como os números inteiros negativos, números reais negativos são representados com o sinal “-” na frente do número, tal como 3.141596.

caractere: consiste de um único símbolo ou de uma concatenação de símbolos do alfabeto

usado pela linguagem Portugol. Este alfabeto inclui todas as letras do alfabeto romano, todos os dígitos, 0, 1, ..., 9 e os caracteres de pontuação, tais como ?, ., ..., entre muitos outros símbolos. Os elementos do conjunto de valores do tipo caractere devem ser escritos, nos algoritmos, entre aspas duplas, como, por exemplo, "a", "Esta e uma frase formada por caracteres". Há um elemento especial, "", que é denominado de **palavra vazia**, pois não possui nenhum símbolo.

- logico: inclui apenas os valores lógicos falso e verdadeiro e as operações de negação, conjunção e disjunção. Nós estudaremos este tipo em maiores detalhes mais adiante.

2.2 Variáveis

Um algoritmo manipula dados, que podem ser dados variáveis ou constantes. Dados variáveis são representados por *variáveis*, enquanto dados constantes são representados por *constantes*¹.

Uma **variável** pode ser imaginada como um “caixa” para armazenar valores de dados. Esta caixa só pode armazenar um único valor por vez. No entanto, o valor armazenado na caixa pode mudar inúmeras vezes durante a execução do algoritmo. Em um ambiente computacional de verdade, a caixa correspondente a uma variável é uma posição da memória do computador.

Uma variável possui nome, tipo e conteúdo. O **nome de uma variável** deve ser único, isto é, identificar, de forma única, a variável no algoritmo. O **tipo de uma variável** define os valores que podem ser armazenados na variável. O **conteúdo de uma variável** é o valor que ela armazena. É importante lembrar que uma variável só pode armazenar um valor de cada vez. No entanto, ela pode assumir vários valores distintos do mesmo tipo durante a execução do algoritmo.

O ato de se criar uma variável é conhecido como **declaração de variável**.

Na linguagem Portugol, declaramos uma variável usando uma sentença da seguinte forma:

var *nome* : *tipo*

onde *nome* é o nome da variável e *tipo* é o tipo da variável.

Por exemplo, a sentença

var lado : real

declara uma variável de nome *lado* do tipo real.

Podemos declarar mais de uma variável do mesmo tipo em uma mesma linha. Por exemplo,

var lado, area : real

Note que nenhum conteúdo (isto é, valor) foi associado à variável durante a sua declaração. Esta associação é denominada **definição** e deve ser realizada após a declaração da variável usando uma **instrução de leitura** ou um **comando de atribuição**. Vamos detalhar essas duas formas.

A instrução de leitura tem a forma

¹Não discutiremos constantes neste momento.

leia (*nome*)

onde *nome* é o nome de uma variável. Por exemplo,

leia (lado)

é uma instrução de leitura que atribui um valor à variável lado. O valor atribuído pela instrução deve ser fornecido como entrada para o algoritmo durante a sua execução. Para você ter uma idéia mais concreta do que estamos falando, demonstraremos, em *sala de aula*, a execução da instrução de leitura do comprimento dos lados de um quadrado que escrevemos para o Algoritmo 1.2.

A instrução de atribuição possui a forma

nome <- *valor*

onde *nome* é o nome de uma variável e *valor* é um valor do mesmo tipo de dados da variável. Por exemplo,

lado <- 2.5

atribui o valor 2.5 à variável de nome *lado*. Para que uma instrução de atribuição faça sentido, a variável lado deve ser do tipo real e deve ter sido declarada antes da instrução de atribuição ser executada.

O símbolo <- é conhecido como **operador de atribuição**.

Muitas vezes, o valor atribuído a uma variável através de uma instrução de atribuição é resultante de uma *expressão aritmética* ou outro tipo de expressão que estudaremos mais adiante. Por exemplo,

area <- lado * lado

é uma instrução de atribuição que atribui o valor da variável *lado* ao quadrado à variável *area*. O que vemos no lado direito do operador de atribuição, lado * lado, é um exemplo de expressão aritmética.

Um valor atribuído a uma variável permanece associado a ela até que uma instrução de atribuição, que o substitua por outro, seja executada. Em qualquer dado instante de tempo durante a execução de um algoritmo, o valor armazenado em uma qualquer variável (se algum) é denominado **valor atual** (ou **valor corrente**) da variável. Enquanto não atribuirmos um valor a uma variável, a variável permanecerá com **valor desconhecido**. Finalmente, é importante lembrar que uma variável só poderá receber um valor através de uma instrução de leitura ou atribuição.

2.3 Exemplos

Seguem abaixo alguns exemplos de declaração de variáveis:

var fruta : caractere

var letra : caractere

var resultado : logico

var altura : real

var idade : inteiro

A seguir, temos exemplos de instruções de atribuição que atribuem valores a essas variáveis:

```
fruta <- "maçã"
```

```
letra <- "c"
```

```
resultado <- falso
```

```
altura <- 1.83
```

```
idade <- 5
```

As mesmas variáveis podem ter valores atribuídos a elas através de instruções de leitura como segue:

```
leia (fruta)
```

```
leia (letra)
```

```
leia (altura)
```

```
leia (idade)
```

```
leia (resultado)
```

2.4 Nomes de variáveis

Na linguagem Portugol, usamos as seguintes regras para criar um nome de variável:

1. Nomes de variáveis devem possuir como primeiro caractere uma letra ou o símbolo ''' (sublinhado). Os demais caracteres, se algum, devem ser letras, números ou sublinhado.
2. Nomes de variáveis não podem ser iguais a palavras reservadas.
3. Nomes de variáveis podem ter, no máximo, 127 caracteres.
4. Não há diferença entre letras maiúsculas e minúsculas em nomes de variáveis.

De acordo com a regra 1, nomes de variáveis não podem conter espaços em branco. De acordo com a regra 2, nomes de variáveis não podem ser palavras reservadas da linguagem Portugol. Uma **palavra reservada** é uma palavra que possui um significado especial para a linguagem Portugol. Em geral, uma palavra reservada identifica uma instrução. Neste texto, tais palavras aparecem sublinhas. O conjunto de palavras reservadas do Portugol é mostrado na Tabela 2.1.

Por exemplo,

_12234, fruta e x123

são nomes válidos para variáveis, mas

maria bonita, pi, fru?ta e 1xed

não são. O nome *maria bonita* contém um espaço em branco. O nome *pi* é uma palavra reservada. O nome *fru?ta* contém um caractere que não é letra, número nem sublinhado, ?. O nome *1xed* inicia com um número. Com exceção de *pi*, que viola a regra 2, os demais nomes violam a regra 1.

2.5 Exercícios Resolvidos

1. Escreva a declaração de uma variável do tipo real de nome *x*.

solução:

```
var x : real
```

2. Escreva a declaração de uma variável do tipo caractere de nome *carro*.

solução:

```
var carro : caractere
```

3. Escreva a instrução de atribuição que atribui o valor 2.3 à variável do problema 1.

solução:

```
x <- 2.3
```

4. Escreva a instrução de atribuição que atribui o valor "corsa" à variável do problema 2.

solução:

```
carro <- "corsa"
```

5. Quais dos seguintes nomes são válidos como nomes de variáveis?

a) xyz_2

b) _

c) _ _ _ _

d) x123

e) 123y

f) 1_2

solução:

(a), (b), (c) e (d).

2.6 Exercícios propostos

1. Escreva a declaração de uma variável do tipo caractere de nome *rua*.
2. Escreva a instrução de atribuição que atribui o nome de sua rua à variável do problema 1.
3. Escreva a declaração de uma variável do tipo logico de nome *achou*.
4. Escreva a instrução de atribuição que atribui a palavra reservada verdadeiro à variável do problema 3.

5. Quais dos seguintes nomes são válidos como nomes de variáveis?
- a) meucarro
 - b) salute!
 - c) x_y_1
 - d) _ _ _
 - e) 34
6. Escreva o tipo de dado ideal para se representar cada uma das seguintes informações:
- a) O nome de uma rua
 - b) A data de nascimento de uma pessoa
 - c) Se uma pessoa é diabética ou não
 - d) O saldo de uma conta bancária
 - e) O resultado de uma operação de raiz quadrada
7. Identifique o tipo de dados dos seguintes valores:
- a) "9 de agosto de 1968"
 - b) 1.3
 - c) falso
 - d) -31
 - e) "?"

aleatorio	e	grauprad	passo
abs	eco	inicio	pausa
algoritmo	enquanto	int	pi
arcos	entao	interrompa	pos
arcsen	escolha	leia	procedimento
arctan	escreva	literal	quad
arquivo	exp	log	radpgrau
asc	faca	logico	raizq
ate	falso	logn	rand
caractere	fimalgoritmo	maiusc	randi
caso	fimenquanto	mensagem	repita
compr	fimescolha	minusc	se
copia	fimfuncao	nao	sen
cos	fimpara	numerico	senao
cotan	fimprocedimento	numpcarac	timer
cronometro	fimrepita	ou	tan
debug	fimse	outrocaso	var
declare	funcao	para	verdadeiro
			xou

Tabela 2.1: Palavras reservadas da linguagem Portugol.

ENTRADA E SAÍDA

3.1 Instruções de entrada e saída

Todo algoritmo necessita de uma forma de obtenção dos dados de entrada do problema, assim como uma forma de comunicação da saída por ele produzida. Para tal, os algoritmos contam com instruções de *entrada* e *saída*. Na linguagem Portugol, a instrução de leitura é leia e a instrução de saída é escreva. Como vimos na aula anterior, a instrução de leitura requer o nome de uma variável para armazenar o dado de entrada a ser obtido.

Por exemplo,

```
leia (lado)
```

onde *lado* é o nome de uma variável. Quando a instrução acima é executada, o valor lido passa a ser o conteúdo de *lado*. Logo, para acessar o valor lido, basta usarmos o nome da variável. **É importante lembrar que a variável tem de ser declarada antes de seu uso pela instrução de leitura.**

A instrução de escrita, na linguagem Portugol, é denominada escreva. Esta instrução também requer um argumento, que corresponde ao valor a ser escrito como saída. Por exemplo, este valor pode ser o conteúdo de uma variável ou uma constante.

Assim,

```
escreva(lado)
```

escreve como saída o *valor* da variável de nome *lado*.

É interessante frisar que a instrução de escrita não é usada apenas para escrever o *resultado* do algoritmo, mas sim para escrever qualquer informação que auxilie a comunicação entre o algoritmo e o “mundo exterior”. Por exemplo, é comum escrevermos uma *mensagem* antes de uma instrução de leitura. Esta mensagem, em geral, é uma descrição muito sucinta do valor que o algoritmo espera receber do “mundo exterior” para atribuir à variável na instrução de leitura.

Por exemplo, a instrução

```
escreva("Entre com o comprimento dos lados do quadrado: ")
```

pode preceder a instrução

```
leia(lado)
```

para indicar ao “mundo exterior” que o algoritmo espera um valor correspondente ao comprimento dos lados de um quadrado. A mensagem “Entre com o comprimento dos lados do quadrado: ”

é um valor *constante*, ou seja, um valor do conjunto de valores do tipo caractere. Quando a instrução de escrita acima é executada, a saída correspondente é a sentença entre aspas duplas:

Entre com o comprimento dos lados do quadrado:

O uso da instrução de escrita para emitir mensagens é motivado pelo fato de um programa de computador, em geral, interagir com o ser humano durante sua execução; isto é, o “mundo exterior” é, em geral, um humano. Mais especificamente, é bastante comum que a entrada de um programa seja fornecida por um ser humano. Neste caso, as mensagens de saída têm como objetivo auxiliar a comunicação entre o programa e o ser humano, que é conhecido por **usuário**. Em um ambiente de programação típico, as mensagens de saída aparecem em um monitor de vídeo e os dados de entrada são fornecidos ao programa por meio de um teclado ou *mouse*.

3.2 A estrutura de um algoritmo

Um algoritmo escrito em Portugol pode ser tipicamente dividido nas seguintes partes:

- A **linha de cabeçalho**, que é a primeira linha do algoritmo, contém a palavra `algoritmo` seguida por um texto que identifica o algoritmo.
- A **declaração de variáveis**, que é a seção em que as variáveis são declaradas, é iniciada pela palavra `var` e seguida por uma ou mais declarações de variáveis.
- O **corpo do algoritmo**, que contém as instruções que, de fato, fazem o “trabalho” descrito pelo algoritmo.
O corpo do algoritmo se inicia com a palavra `inicio`
- A **linha final**, que é obrigatoriamente a última linha do algoritmo, contém a palavra `fimalgoritmo` que especifica o término do algoritmo.

Os algoritmos escritos na linguagem Portugol também possuem algumas particularidades que os tornam mais claros e mais fáceis de serem lidos e entendidos. As principais características são:

- Cada instrução do algoritmo é escrita em uma linha dedicada apenas à instrução.
- As declarações de variáveis e o conteúdo do corpo do algoritmo são *identados* com relação à linha de cabeçalho e a linha final. Além disso, pode ser que haja linhas em “branco” entre uma seqüência de linhas. Essas linhas em branco servem para separar partes do algoritmo que estão menos relacionadas.
- O algoritmo pode conter *comentários*. Comentários são linhas não “executáveis”, ou seja, elas não fazem parte do processo de produção da saída do algoritmo e servem apenas para nos auxiliar na leitura e entendimento da solução do algoritmo. As linhas de comentário se iniciam obrigatoriamente com o símbolo `//`.

O Algoritmo 3.2 lê o valor de uma variável, denominada *num*, e escreve este mesmo valor como saída.

```
1 // Este algoritmo exemplifica as instrucoes de entrada e saida
2
3 algoritmo "Leitura e escrita"
4 var
5     // Secao de declaracao de variaveis
6     num : inteiro
7 inicio
8     // Escreve uma mensagem para indicar o que deve ser lido pelo algoritmo
9     escreva ("Entre com um numero inteiro: ")
10
11    // Realiza a leitura de um numero inteiro e o associa a uma variavel
12    leia (num)
13
14    // Escreve o valor do numero lido
15    escreva ("O numero que voce forneceu foi: ", num)
16 fimalgoritmo
```

Algoritmo 3.1: Algoritmo para ler um numero inteiro e escrever o valor lido.

Alguns comentários sobre o algoritmo acima:

- O nome do algoritmo foi escrito cercado por aspas duplas:

```
algoritmo "'Leitura e escrita"
```

- Devemos declarar uma variável para cada dado de entrada. No problema computacional acima, a entrada consiste apenas de um número inteiro. Logo, declaramos apenas uma variável para a entrada do problema: *num*.
- A instrução de saída foi utilizada para emitir uma mensagem,

```
escreva ("Entre com um numero inteiro: ")
```

e para escrever a saída do algoritmo:

```
escreva ("O numero que voce forneceu foi: ", num)
```

Note que a linha acima possui dois “dados” separados por vírgula. O primeiro é a mensagem e o segundo é a variável *num*. Neste caso, os dados são escritos no dispositivo de saída em uma mesma linha, separados por um espaço e sem a vírgula.

3.3 Exercícios resolvidos

1. Considere o Algoritmo 1 para ler e escrever um número e uma palavra:

```

1 //
2 // Este algoritmo le e escreve um numero e uma palavra
3 //
4 algoritmo "Leitura e escrita de numero e palavra"
5 var
6 //
7 // Secao de declaracao de variaveis
8 //
9 num : inteiro
10 pal : caractere
11 inicio
12 // Escreve uma mensagem para solicitar um numero
13 escreva ("Entre com um numero inteiro: ")
14
15 // Realiza a leitura de um numero inteiro e o atribui a uma variavel
16 leia (num)
17
18 // Escreve uma mensagem para solicitar uma palavra
19 escreva ("Entre com uma palavra: ")
20
21 // Realiza a leitura de uma palavra e a atribui a uma variavel
22 leia (pal)
23
24 // Escreve numero e palavra lidos
25 escreva ("O numero e a palavra fornecidos foram: ", num, " e ", pal)
26 fimalgoritmo

```

Algoritmo 3.2: Algoritmo para ler um numero e uma palavra e escrever ambos.

Então, responda:

- a) Quais são os dados de entrada do algoritmo?
- b) Qual(is) é(são) a(s) variável(is) que armazena(m) o(s) valor(es) de entrada?
- c) Se fornecermos o o número inteiro 5 e a palavra “algoritmo” como entrada para o algoritmo, o que o algoritmo escreve como saída?

Solução:

1. a) Um número inteiro e uma palavra.
- b) *num* e *pal*.
- c) O numero e a palavra fornecidos foram: 5 e algoritmo

3.4 Exercícios propostos

1. Escreva um algoritmo para ler um número real e uma letra e escrever o número e a letra lidos.

2. Use a ferramenta VISUALG para executar o algoritmo que você escreveu para o problema anterior.

EXPRESSÕES ARITMÉTICAS – PARTE 1

4.1 Operadores aritméticos

Os operadores aritméticos definem as operações aritméticas que podem ser realizadas sobre os números inteiros e reais. Para os inteiros, as operações aritméticas são a adição, subtração, multiplicação e resto. Para os números reais, as operações aritméticas são a adição, subtração, multiplicação e divisão. Nesta aula, restringiremos nossa atenção aos números inteiros apenas. Na linguagem Portugol, os operadores aritméticos correspondentes às operações definidas sobre os inteiros são

- + (adição)
- – (subtração ou menos unário)
- * (multiplicação)
- \ (divisão *inteira*)
- % (resto – aplicado apenas aos valores inteiros).

Com os operadores acima, podemos escrever *expressões aritméticas* que envolvem constantes e variáveis inteiras. Por exemplo, suponha que a , b e c sejam variáveis do tipo inteiro. Então, temos que

$$a + b + c, \quad a - b * c \% 2, \quad e \quad - 5 + 3 * 8 \setminus 2$$

são todas expressões aritméticas *válidas* na linguagem Portugol.

Nos exemplos acima, tivemos o cuidado de usar operandos do *mesmo tipo*. A razão para tal é que, por definição, cada operador aritmético atua sobre valores de um mesmo tipo e o resultado da operação deve sempre ser um valor do mesmo tipo dos operandos. Logo, não faz sentido escrevermos algo como $a + 2$ quando a é uma variável ou constante do tipo real, pois existe uma ambigüidade em relação ao resultado da operação. No entanto, como veremos mais adiante, podemos definir *regras semânticas* associadas aos operadores que nos permitem interpretar, de forma única, o resultado da operação aritmética correspondente. Tais regras nos permitirão escrever expressões aritméticas envolvendo variáveis e constantes dos tipos inteiro e real.

4.2 Precedência de operadores

Qual é o valor da expressão aritmética

$$5 * 3 \% 2 \quad ?$$

Podemos dizer que o valor da expressão é igual a 1, se avaliarmos $5 * 3$ primeiro e, depois, $15 \% 2$, ou podemos dizer que é igual a 5 se avaliarmos $3 \% 2$ primeiro e, depois, $5 * 1$. As duas respostas são igualmente válidas. No entanto, como não podemos permitir ambigüidades em algoritmos, devemos definir **regras de precedência** de operadores, que são regras para definir a ordem em que os operadores aritméticos que vimos são aplicados em uma expressão aritmética qualquer.

Na linguagem Portugol, os operadores possuem **prioridades** associadas com eles. A operação associada a um operador com prioridade p é sempre executada *antes* da operação associada a um operador com prioridade q sempre que $p > q$. Quando $p = q$, a operação correspondente ao operador mais à esquerda é executado. O operador de maior prioridade é o **menos unário**, $-$. Em seguida, temos os operadores $*$, \backslash e $\%$. Finalmente, com a prioridade mais baixa, temos os operadores $+$ e $-$, onde $+$ e $-$ são os operadores de adição e subtração, respectivamente. A Tabela 4.1 resume essas prioridades.

Operador	Símbolo	Prioridade
menos unário	$-$	mais alta
multiplicação, divisão inteira e resto	$*$, \backslash e $\%$	\uparrow
adição e subtração	$+$, $-$	mais baixa

Tabela 4.1: Operadores aritméticos sobre os inteiros e suas prioridades

Por exemplo, em

$$a + b + c,$$

a operação $a + b$ é realizada e, em seguida, o resultado dela é adicionado ao valor de c , pois os operadores possuem a mesma prioridade e, portanto, as operações são realizadas da esquerda para a direita.

Na expressão aritmética

$$a - b * c \% 2,$$

a operação $b * c$ é efetuada primeiro e, em seguida, o resto da divisão de $b * c$ por 2 é calculado. Finalmente, o resto é subtraído de a . Note que a multiplicação foi efetuada antes da divisão, pois os operadores $*$ e $\%$ possuem a mesma prioridade, mas $*$ está mais à esquerda.

Uma boa forma de se familiarizar com os operadores aritméticos e as regras de precedência é escrevendo algoritmos para escrever o resultado de expressões aritméticas. O Algoritmo 4.3 calcula e escreve, usando a instrução `escreval`, o resultado de expressões envolvendo números inteiros. A instrução `escreval` faz o mesmo que a instrução `escreva`, mas gera um “salto de linha” após a escrita. Um algoritmo mais interessante, o Algoritmo 4.3, recebe, como entrada, três inteiros quaisquer e calcula e escreve o resultado de algumas expressões aritméticas envolvendo os inteiros lidos.

4.3 Alteração de prioridades

Algumas vezes é desejável alterar a ordem (imposta pelas regras de precedência) segundo a qual as operações são realizadas em uma expressão aritmética. Para tal, fazemos uso de *parênteses*. Por hipótese, todo operador possui prioridade mais baixa do que a dos parênteses. Isto garante que os operandos correspondentes ao valor das expressões entre parênteses sejam calculados *antes* de serem usados pelos demais operadores. É importante destacar que os parênteses devem

ocorrer em *pares* (um aberto e um fechado) nas expressões e podem ocorrer “aninhados” em diversos níveis.

```

1 algoritmo "Calculo de expressoes aritmeticas"
2 inicio
3   escreval("O resultado da expressao 5 * 3 % 2 e: ", 5 * 3 % 2)
4
5   escreval("O resultado da expressao -5 * 3 % 2 \ 8 e: ", -5 * 3 % 2 \ 8)
6
7   escreval("O resultado da expressao -5 - 3 - 6 * 3 e: ", -5 - 3 - 6 * 3)
8 fimalgoritmo

```

Algoritmo 4.1: Cálculo de algumas expressões aritméticas com números inteiros

Por exemplo, na expressão

$$(a - b) * (c \% 2),$$

a operação $a - b$ é realizada primeiro. Em seguida, a operação $c \% 2$ é realizada e, por último, a multiplicação dos resultados das duas operações anteriores entre os parênteses é realizada. Mas, como sabemos disso? A idéia é imaginar as expressões entre parênteses como operandos a serem “descobertos”. Com isso em mente, a expressão acima pode ser imaginada como tendo a forma

$$op_1 * op_2,$$

onde op_1 e op_2 são as expressões $(a - b)$ e $(c \% 2)$. Então, o que temos é uma simples multiplicação de dois valores, op_1 e op_2 . No entanto, para que esta multiplicação seja realizada, precisamos dos valores op_1 e op_2 . Para tal, **assumimos** que o valor, op_1 , à esquerda do operador de multiplicação, $*$, será obtido antes do valor, op_2 , à direita dele. Para calcular op_1 , avaliamos a expressão

$$a - b,$$

que se encontra dentro do parênteses. Neste momento, descobrimos que a subtração $a - b$ é a primeira operação aritmética *realizada*. Uma vez que o valor op_1 tenha sido descoberto, avaliamos

$$c \% 2,$$

que é a expressão correspondente ao valor op_2 . Neste momento, descobrimos que a operação de resto de divisão, $c \% 2$, é a segunda operação aritmética *realizada*. Neste momento, dispomos dos valores op_1 e op_2 e, portanto, podemos realizar a multiplicação $op_1 * op_2$, que passa a ser a terceira operação realizada. Logo, os operadores são aplicados na ordem $-$, $\%$ e $*$. Usamos a notação

$$(a -_1 b) *_3 (c \%_2 2)$$

para indicar este fato. Isto é, os operadores possuem índices que indicam a ordem em que são aplicados.

```

1 algoritmo "Expressoes aritmeticas envolvendo variaveis e constantes"
2 var
3   a , b , c : inteiro
4 inicio
5   escreva("Entre com o valor da variavel a: ")
6   leia(a)
7
8   escreva("Entre com o valor da variavel b: ")
9   leia(b)
10
11  escreva("Entre com o valor da variavel c: ")
12  leia(c)
13
14  escreval("O resultado da expressao a * b % c e: ", a * b % c)
15
16  escreval("O resultado da expressao -a * b % c * 8 e: ", -a * b % c * 8)
17
18  escreval("O resultado da expressao -a - b - c * 3 e: ", -a - b - c * 3)
19 fimalgoritmo

```

Algoritmo 4.2: Cálculo de expressões aritméticas com variáveis inteiras

A expressão

$$((2 + 3) - (1 + 2)) * 3 - (3 + (3 - 2))$$

é bem mais complexa do que a anterior, mas podemos determinar a ordem em que os operadores são aplicados da mesma forma que antes. O primeiro passo é substituir as expressões dentro dos parênteses por operandos a serem descobertos. Isto é feito para os parênteses mais externos:

$$op_1 * 3 - op_2.$$

Agora, vemos que se os valores entre parênteses fossem conhecidos, haveria apenas duas operações a serem realizadas: uma multiplicação e uma adição. A multiplicação possui prioridade sobre a adição e ela precisa do valor op_1 para ser realizada. Então, considere a expressão correspondente a op_1 :

$$(2 + 3) - (1 + 2).$$

Esta expressão contém outras expressões dentro de parênteses e, assim como antes, ela pode ser vista como

$$op_3 - op_4.$$

onde op_3 e op_4 correspondem às expressões $2 + 3$ e $1 + 2$, respectivamente. Para realizarmos a operação de subtração acima, precisamos dos valores op_3 e op_4 . Por estar à esquerda do operador, o valor op_3 é descoberto primeiro. Isto implica que a primeira operação realizada é a adição

$$2 + 3$$

e a próxima é a adição

$$1 + 2.$$

Em seguida, temos a subtração $op_3 - op_4$:

$$(2 + 3) - (1 + 2).$$

Depois que a subtração acima for realizada, o valor op_1 se torna conhecido e, conseqüentemente, a multiplicação $op_1 * 3$ pode ser realizada, tornando-se a quarta operação realizada. O resultado desta operação é o primeiro operando disponível da subtração em $op_1 * 3 - op_2$. Mas, esta subtração não pode ser efetuada antes do valor op_2 ser conhecido, ou seja, antes da expressão

$$3 + (3 - 2)$$

ser avaliada. Assim como fizemos antes, podemos imaginar a expressão acima tendo a forma

$$3 + op_5,$$

onde op_5 é o valor da expressão, $3 - 2$, entre parênteses. A adição na expressão acima precisa do valor op_5 para ser realizada. Isto significa que a subtração $3 - 2$ é a quinta operação realizada. Depois dela, a adição $3 + op_5$ é realizada, tornando-se a sexta operação realizada. Logo em seguida, o valor op_2 se torna conhecido, o que possibilita a realização da sétima e última operação, que é a subtração em $op_1 * 3 - op_2$. Usando a notação de subscrito, temos a seguinte ordem:

$$((2 +_1 3) -_3 (1 +_2 2)) *_4 3 -_7 (3 +_6 (3 -_5 2)).$$

4.4 A instrução de atribuição

Quando utilizamos expressões aritméticas em nossos algoritmos, necessitaremos, muito freqüentemente, armazenar o valor da expressão em uma variável. Como vimos na Aula 2, a atribuição de um valor a uma variável pode ser realizada através da instrução de leitura leia ou do operador de atribuição $<-$. A instrução leia é usada *apenas* quando o valor a ser atribuído à variável é fornecido como entrada para o algoritmo. Como o valor que queremos atribuir à variável é resultante da avaliação de uma expressão aritmética, devemos usar o operador de atribuição.

Por exemplo, suponha que o resultado da expressão

$$5 * \%2$$

deva ser atribuído a uma variável inteira de nome *resultado*. Então, a atribuição pode ser realizada da seguinte forma:

$$\text{resultado} <- 5 * \%2$$

Para um exemplo mais concreto do uso do operador de atribuição, considere o Algoritmo 4.5, que lê dois números inteiros, calcula o quadrado da soma dos dois números lidos, atribui o resultado a uma variável inteira, usando o operador $<-$, e escreve o valor da variável.

4.5 Exercícios resolvidos

1. Considere a expressão polinomial

$$5x^3 + 7x^2 - 3x - 1,$$

onde x é uma variável. Escreva a expressão acima usando a linguagem Portugol.

solução:

$$5 * x * x * x + 7 * x * x - 3 * x - 1.$$

2. Avalie a seguinte expressão aritmética de acordo com as regras de precedência da linguagem Portugol:

$$2 - 3 * 5$$

solução:

$$2 - 3 * 5 = 2 - 15 = -13.$$

```
1 algoritmo "Quadrado da soma de 2 inteiros"
2 var
3   a, b, quadrado : inteiro
4 inicio
5   escreva("Entre com o primeiro inteiro: ")
6   leia(a)
7
8   escreva("Entre com o segundo inteiro: ")
9   leia(b)
10
11  quadrado <- (a + b) * (a + b)
12
13  escreva("O quadrado da soma dos inteiros lidos e: ", quadrado)
14 fimalgoritmo
```

Algoritmo 4.3: Cálculo do quadrado da soma de dois inteiros

4.6 Exercícios propostos

1. Escreva a expressão abaixo usando a linguagem Portugol:

$$x_0 + v \cdot t$$

2. Avalie a seguinte expressão aritmética de acordo com as regras de precedência da linguagem Portugol:

$$(2 - 3) * 5$$

3. Suponha que a linha 11 do Algoritmo 4.5 seja substituída por

```
quadrado <- a + b * a + b
```

Você acha que o algoritmo continuará correto? Justifique sua resposta.

4. Suponha que a linha 11 do Algoritmo 4.5 seja substituída pelas duas seguintes linhas:

```
quadrado <- a + b
```

```
quadrado <- quadrado * quadrado
```

Você acha que o algoritmo continuará correto? Justifique sua resposta.

5. Escreva um algoritmo, usando a linguagem Portugol, para ler dois números inteiros, calcular o cubo da soma desses dois números e escrever o resultado calculado como saída.
6. Implemente o algoritmo anterior usando a ferramenta VISUALG.

EXPRESSÕES ARITMÉTICAS – PARTE 2

5.1 Operadores aritméticos sobre os reais

Como vimos na aula anterior, os operadores aritméticos definem as operações aritméticas que podem ser realizadas sobre os números inteiros e reais. Já estudamos os operadores aritméticos que atuam sobre os inteiros e, nesta aula, estudaremos aqueles que atuam sobre os reais:

- + (adição).
- – (subtração ou menos unário).
- * (multiplicação).
- / (divisão).

Observe que os três primeiros operadores são comuns aos reais e inteiros. Observe também que o operador de divisão, /, está definido apenas para os reais. Por sua vez, o operador de resto, %, está definido apenas para os inteiros. A Tabela 5.1 lista os operadores aritméticos sobre os reais e suas respectivas prioridades. Ao escrevermos expressões aritméticas, podemos alterar a prioridade desses operadores com o uso de parênteses da mesma forma que vimos antes.

Operador	Símbolo	Prioridade
menos unário	–	mais alta
multiplicação e divisão	*, /	↑
adição e subtração	+, –	mais baixa

Tabela 5.1: Operadores aritméticos sobre os reais e suas prioridades.

Por exemplo, considere a expressão

$$c - \frac{3a + 2b}{a - 1} \cdot \frac{a + b}{1 + \frac{a + b}{2c}}$$

onde a , b e c são variáveis. Na linguagem Portugol, a expressão acima pode ser escrita como segue:

$$(3.0 * a + 2.0 * b) / (c - (a - 1.0) / (1.0 + (a + b) / (2.0 * c))).$$

É importante notar que *todos* os parênteses são necessários para que a expressão, na linguagem Portugol, seja equivalente à expressão aritmética dada. Abaixo, indicamos, com índices nos

operadores, a ordem em que as operações da expressão são executadas quando a expressão é avaliada:

$$(3.0 * a + 2.0 * b) / (c - (a - 1.0) / (1.0 + (a + b) / (2.0 * c)))$$

O Algoritmo 5.1 calcula o valor da expressão acima para quaisquer valores de a , b e c fornecidos como entrada. O valor da expressão é atribuído a uma variável antes de ser escrito como saída.

```

1 algoritmo "Expressoes aritmeticas com variaveis e constantes reais"
2 var
3   a, b, c, res : real
4 inicio
5   escreva("Entre com o valor da variavel a: ")
6   leia(a)
7
8   escreva("Entre com o valor da variavel b: ")
9   leia(b)
10
11  escreva("Entre com o valor da variavel c: ")
12  leia(c)
13
14  res <- (3.0 * a + 2.0 * b) /
15          (c - (a - 1.0) / (1.0 + (a + b) /
16          (2.0 * c)))
17
18  escreva ("O resultado da expressao e: ", res)
19 fimalgoritmo

```

Algoritmo 5.1: Cálculo de expressões aritméticas com variáveis reais

5.2 Regras semânticas

Na expressão

$$(3.0 * a + 2.0 * b) / (c - (a - 1.0) / (1.0 + (a + b) / (2.0 * c))),$$

as constantes foram escritas com números reais e as variáveis são todas do tipo real. Logo, cada operador aritmético atuará sobre dois valores reais e produzirá outro valor real. No entanto, é possível, na linguagem Portugol, escrevermos expressões aritméticas que envolvam constantes e variáveis dos tipos inteiro e real. Para que tais expressões façam “sentido”, definimos a seguinte **regra semântica**: se os dois operandos de um operador binário possuem tipos distintos (um é do tipo inteiro e o outro, do tipo real), o valor do tipo inteiro é convertido para o valor do tipo real equivalente. Logo, a operação é executada sobre dois valores reais e o resultado é um valor do tipo real.

Por exemplo, na expressão

$$(a - b) * (c / 2),$$

se as variáveis a , b e c são do tipo real, o inteiro 2 é convertido (automaticamente) para o real 2.0 imediatamente antes da operação de divisão ser executada. Em outras palavras, na linguagem

Portugol, a expressão acima é equivalente à expressão:

$$(a - b) * (c / 2.0).$$

De maneira geral, o operador de divisão, /, pode ser utilizado para dividir valores inteiros. Por exemplo,

$$5 / 2$$

é igual ao valor real 2.5. Na expressão acima, não há nenhuma constante ou variável do tipo real. Mas, mesmo assim, os valores inteiros, que são operandos do operador /, são convertidos para os valores reais equivalentes antes da operação de divisão ser efetuada. Logo, nós podemos dividir dois inteiros usando /, mas o resultado da divisão é um valor do tipo real e não inteiro. Quando desejarmos realizar a divisão inteira dos dois inteiros, devemos usar o operador \.

Um outro aspecto importante das expressões aritméticas envolvendo valores inteiros e reais é a precedência de operadores. O que acontece se a expressão contiver os operadores / e %? Como sabemos, o operador % só pode ser aplicado a inteiros. Mas, nada impede que ele ocorra em uma expressão aritmética que envolva inteiros e o operador /. Por exemplo, considere a expressão

$$5 \% 2 / 2.$$

Na linguagem Portugol, o operador % possui prioridade igual a do operador /. Logo, a operação $5 \% 2$ é realizada primeiro, produzindo o inteiro 1 como resultado. Em seguida, a operação $1 / 2$ é realizada. Isto significa que os valores inteiros serão convertidos (automaticamente) para valores reais equivalentes e a divisão será executada, gerando o valor 0.5. Bem, e se a expressão fosse

$$5 / 2 \% 2 ?$$

Neste caso, a divisão $5 / 2$ é a primeira operação realizada, gerando o número real 2.5 como resultado. Em seguida, a operação $2.5 \% 2$ deve ser realizada. Mas, como o operador % não pode atuar sobre números reais, a operação $2.5 \% 2$ não pode ser realizada. Você poderia imaginar que o número 2.5 seria convertido em um número inteiro (por arredondamento ou truncamento), de modo que a operação pudesse ser efetuada. *Na linguagem Portugol, nenhum valor real pode ser automaticamente convertido em um valor inteiro.* Isto significa que a segunda expressão aritmética acima é inválida na linguagem Portugol. A ferramenta VISUALG acusará um erro se tentarmos utilizar esta expressão em um algoritmo (verifique!). Em uma aula futura, veremos uma *função* da linguagem Portugol que nos permitirá obter a parte inteira de um um valor real qualquer.

5.3 Um algoritmo envolvendo constantes e variáveis reais

Considere o problema de desenvolver um algoritmo para determinar o volume, V , de uma esfera a partir do raio, r , da esfera. Como sabemos, a relação entre os valores V e r é dada pela fórmula

$$V = \frac{4}{3} \cdot \pi \cdot r^3. \quad (5.1)$$

O nosso algoritmo deve ler o valor do raio r da esfera, calcular o valor de V e escrever este valor como saída. Para calcular o valor de V , nosso algoritmo pode se utilizar da fórmula em Eq. (5.1). Uma das particularidades da fórmula é que ela utiliza a constante π . Para lidar com situações

como essa, a linguagem Portugol possui uma palavra reservada, chamada *pi*, que representa a constante π . Logo, na linguagem Portugol, a fórmula acima pode ser escrita como segue:

$$(4 / 3) * pi * raio * raio * raio ,$$

onde assumimos que *raio* é o nome da variável que representa o raio r da esfera. Se a linguagem Portugol não nos fornecesse a constante *pi*, poderíamos escrever uma expressão para a fórmula, tal como

$$(4 / 3) * 3.141596 * raio * raio * raio ,$$

que se utiliza de um valor aproximado para o número π . No entanto, o uso da palavra *pi* é mais recomendado, pois ela faz com que a sintaxe da expressão resultante seja mais legível e significativa.

Uma vez que saibamos como escrever, na linguagem Portugol, a expressão aritmética que calcula o valor do volume V da esfera de raio r , podemos desenvolver o restante do algoritmo. A entrada do algoritmo consiste apenas do valor do raio, r , da esfera. Então, devemos declarar uma variável, digamos *raio*, para representar o valor de r . Após calcularmos o valor do volume, V , da esfera usando a fórmula em Eq. (5.1), atribuímos este valor a uma outra variável, digamos *volume*, que precisa ser declarada também. As duas variáveis do algoritmo são do tipo real. Finalmente, escreveremos o valor da variável *volume* como saída. O algoritmo resultante é o Algoritmo 5.3.

```
1 algoritmo "Volume da esfera"
2 var
3     raio, volume : real
4 inicio
5     escreva("Entre com o valor do raio da esfera: ")
6     leia(raio)
7
8     volume <- (4 / 3) * pi * raio * raio * raio
9
10    escreva ("O volume da esfera de raio ", raio, " e ", volume)
11 fimalgoritmo
```

Algoritmo 5.2: Cálculo do volume da esfera

5.4 Exercícios resolvidos

1. Escreva a seguinte expressão aritmética usando a linguagem Portugol e indique a ordem em que os operadores são aplicados na avaliação da expressão (use índices ao lado dos operadores):

$$\frac{1}{1 + \frac{1}{1 + a}} .$$

Assuma que a é uma variável do tipo real.

solução:

A expressão escrita em Portugol é a seguinte:

$$1 / (1 + 1/(1 + a))$$

Você também poderia ter escrito

$$1.0 / (1.0 + 1.0/(1.0 + a))$$

A ordem de avaliação dos operadores é indicada abaixo por índices:

$$1 /_4 (1 +_3 1 /_2 (1 +_1 a)).$$

2. Escreva um algoritmo que leia os valores correspondentes à base e altura de um retângulo, determine a área do retângulo e escreva a área como saída.

solução:

```
1 algoritmo "Area de um retangulo"
2 var
3     base, altura, area : real
4 inicio
5     escreva("Entre com o comprimento da base do retangulo: ")
6     leia(base)
7
8     escreva("Entre com o valor da altura do retangulo: ")
9     leia(altura)
10
11     area <- base * altura
12
13     escreva ("A area do retangulo e ", area)
14 fimalgoritmo
```

Algoritmo 5.3: Cálculo da área de um retângulo

5.5 Exercícios propostos

1. Escreva a seguinte expressão aritmética usando a linguagem Portugol e indique a ordem em que os operadores são aplicados na avaliação da expressão (use índices ao lado dos operadores):

$$\frac{1}{b} + \frac{5}{2 + \frac{1}{1+a}}.$$

Assuma que a e b são variáveis do tipo real.

2. Escreva um algoritmo para calcular a área de um círculo. A entrada do seu algoritmo é o valor do raio do círculo. A saída é o valor da área do círculo.
3. Implemente o algoritmo anterior usando a ferramenta VISUALG.

EXPRESSÕES RELACIONAIS

6.1 Operadores relacionais

Uma **expressão relacional**, ou simplesmente **relação**, é uma comparação entre dois valores de um mesmo tipo. Esses valores são representados na relação através de constantes, variáveis ou expressões aritméticas. A operação de comparação é realizada por um **operador relacional**. Na linguagem Portugol da ferramenta VISUALG, temos os operadores relacionais mostrados na Tabela 6.1.

Operador	Descrição
=	igual a
<>	diferente de
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

Tabela 6.1: Operadores relacionais da linguagem Portugol.

Como exemplo, suponha que a , b e c sejam variáveis do tipo inteiro. Então, temos que

$$a = 2, \quad a > b \quad \text{e} \quad c \leq b$$

são todos exemplos de expressões relacionais. Na expressão

$$a = 2,$$

estamos comparando o valor da variável a com a constante 2. O resultado desta comparação é o valor lógico verdadeiro se o valor de a é igual a 2; caso contrário, o resultado é o valor lógico falso. Na expressão

$$a > b,$$

estamos comparando os valores das variáveis a e b . O resultado desta comparação é o valor lógico verdadeiro se o valor de a é maior do que o de b . Caso contrário, o resultado é o valor lógico falso. Na expressão

$$c \leq b,$$

estamos comparando os valores das variáveis c e b . O resultado desta comparação é o valor lógico verdadeiro se o valor de c é menor ou igual ao valor de b . Caso contrário, o resultado é o valor lógico falso.

6.2 Relações e expressões aritméticas

As relações podem envolver expressões aritméticas. Por exemplo,

$$a > (b + c) \quad \text{e} \quad c \leq (5 - a),$$

onde a , b e c são variáveis do tipo inteiro, são relações válidas na linguagem Portugol. Na primeira delas, estamos comparando o valor de a com o valor resultante da expressão aritmética

$$b + c.$$

Na segunda delas, estamos comparando o valor de c com o valor resultante da expressão aritmética

$$5 - a.$$

Em particular, podemos escrever relações da forma

$$(b + c) \leq (c - 10).$$

Isto é, podemos ter expressões aritméticas nos dois “lados” de um operador relacional. Neste caso, estamos comparando o valor da expressão aritmética do lado esquerdo do operador com o valor da expressão aritmética do lado direito do operador relacional. Além disso, não precisamos sequer usar os parênteses que utilizamos nos exemplos acima. Isto é, basta-nos escrever

$$a > b + c, \quad c \leq 5 - a \quad \text{e} \quad b + c \leq c - 10,$$

que são as mesmas expressões, em Portugol, que

$$a > (b + c), \quad c \leq (5 - a) \quad \text{e} \quad (b + c) \leq (c - 10).$$

Isto porque qualquer operador aritmético possui prioridade maior do que qualquer operador relacional. Logo, a operação de comparação em uma expressão relacional é sempre realizada por último.

6.3 Relações envolvendo tipos não inteiros

É importante ressaltar que os operadores relacionais não se aplicam apenas a valores do tipo inteiro. De fato, eles também podem ser usados com valores do tipo real e caractere. Em particular, se a é uma variável do tipo real, então

$$a \leq 5.7$$

é uma expressão que compara o valor de a com o número 5.7, do tipo real. Na expressão acima, se a fosse do tipo inteiro, seu valor seria primeiro convertido para o valor real equivalente e, depois, comparado com 5.7. Analogamente, se considerarmos uma expressão relacional tal como

$$b <> 5,$$

onde b é uma variável do tipo real, o valor 5, do tipo inteiro, é primeiro convertido para o valor do tipo real equivalente, isto é, para 5.0, e, em seguida, comparado com o valor real da variável b .

Suponha, agora, que a e b são duas variáveis do tipo caractere. Então, as expressões relacionais

$$a = b \quad \text{e} \quad a <> b$$

equivalem a perguntar se os caracteres representados por a e b são iguais ou distintos, respectivamente. Tais expressões são facilmente avaliadas com base nos valores de a e b , é claro. No entanto, como avaliamos as expressões relacionais envolvendo os demais operadores, por exemplo

$$a < b, \quad a > b, \quad a \leq b \quad \text{e} \quad a \geq b?$$

Ou seja, o que significa dizer que uma palavra é *menor do que* outra? Para responder esta pergunta, vamos primeiro considerar o caso em que os valores de a e b são um *caractere* cada. Depois, trataremos do caso em que esses valores são palavras com mais de um caractere. Quando os valores de a e b são um caractere cada, usamos o índice do caractere no *alfabeto* da linguagem Portugol da ferramenta VISUALG, denominado [ASCII](#)¹, para decidir o resultado da comparação.

Cada caractere que pode ser usado na linguagem Portugol possui um número único no alfabeto ASCII. Então, dizemos que a é menor do que b se, e somente se, o índice do caractere representado por a é menor do que o índice do caractere representado por b no alfabeto ASCII. As demais expressões relacionais (definidas pelos operadores $>$, \leq e \geq) são avaliadas de forma análoga. O alfabeto ASCII possui 256 símbolos e não cabe aqui descrever todos eles. No entanto, é bom saber que as letras maiúsculas de “A” a “Z” possuem os índices 65 a 90, respectivamente, as letras minúsculas de “a” a “z” possuem os índices 97 a 122, respectivamente, e os dígitos “0” a “9” possuem os índices 48 a 57, respectivamente. Logo, se o valor de a é “c” e o de b é “4”,

$$a < b, \quad a > b, \quad a \leq b \quad \text{e} \quad a \geq b$$

resultam nos valores lógicos falso, verdadeiro, falso e verdadeiro, respectivamente, quando são avaliadas.

O que dizer se pelo menos um de a e b contém uma palavra com mais de um caractere?

Neste caso, usamos a **ordem lexicográfica** para determinar o resultado das comparações. Para tal, suponha que a palavra representada por a possua n caracteres, a_1, a_2, \dots, a_n , nesta ordem, e a palavra representada por b possua m caracteres, b_1, b_2, \dots, b_m , nesta ordem, com $m, n \in \mathbb{Z}$ e $m, n \geq 1$. Seja $k = \min\{n, m\}$. Então, encontramos o menor valor de i em $\{1, 2, \dots, k\}$ tal que $a_i \neq b_i$. Em seguida, nós determinamos se $a_i < b_i$, $a_i > b_i$, $a_i \leq b_i$ ou $a_i \geq b_i$.

Por exemplo, se $a = \text{“abacate”}$ e $b = \text{“abacaxi”}$, então $i = 6$, pois as palavras em a e b coincidem nos cinco primeiros caracteres, isto é, ambas começam com o prefixo “abaca”. O próximo passo é comparar a_6 com b_6 . Como podemos ver, a_6 é igual a “t” e b_6 é igual a “x”. Então, $a_6 < b_6$ e $a_6 \leq b_6$. Logo, sabemos que a é menor do que b , ou seja, sabemos que as expressões

$$a < b \quad \text{e} \quad a \leq b$$

possuem valor lógico verdadeiro, enquanto as expressões

$$a > b \quad \text{e} \quad a \geq b$$

possuem valor lógico falso.

¹A abreviação de American Standard Code for Information Interchange.

É possível que $a_i = b_i$ para todos os valores de i em $\{1, 2, \dots, k\}$. Quando isto acontece, temos que a palavra em a é um prefixo da palavra em b ou a palavra em b é um prefixo da palavra em a . Por exemplo, se o valor de a é “bola” e o de b é “bolada”, então a palavra em a é um prefixo da palavra em b e, por isso, $a_i = b_i$, para todo $i \in \{1, 2, 3, 4\}$. Quando isso ocorre, a palavra de menor comprimento é considerada a menor palavra. Isto quer dizer que as expressões

$$a < b \quad \text{e} \quad a \leq b$$

possuem valor lógico verdadeiro, enquanto as expressões

$$a > b \quad \text{e} \quad a \geq b$$

possuem valor lógico falso.

Mas, e se as palavras em a e b possuírem o mesmo comprimento? Neste caso, temos que $m = n$ e, se $a_i = b_i$ para todos os valores de i em $\{1, 2, \dots, k\}$ onde $k = \min\{n, m\}$, então a e b são exatamente a mesma palavra! Por exemplo, se o valor de ambas as variáveis, a e b , é “bola”, então

$$a = b, \quad a \leq b \quad \text{e} \quad a \geq b$$

possuem valor lógico verdadeiro, enquanto

$$a <> b, \quad a > b \quad \text{e} \quad a < b$$

possuem valor lógico falso.

Para concluir, note que não faz sentido utilizar operadores relacionais com valores do tipo lógico, exceto pelos operadores $=$ e $<>$. Em outras palavras, não faz sentido dizer que o valor verdadeiro é menor, maior, menor ou igual ou maior ou igual ao valor falso. Como veremos em uma outra aula, podemos construir expressões que relacionam valores lógicos usando *operadores lógicos*.

6.4 Exercícios resolvidos

1. Avalie a relação abaixo:

$$10 \% 5 * 2 <> 5 * 2 + 1$$

solução:

$$\begin{aligned} 10 \% 5 * 2 <> 5 * 2 + 1 &\Rightarrow 0 * 2 <> 5 * 2 + 1 \\ &\Rightarrow 0 <> 5 * 2 + 1 \\ &\Rightarrow 0 <> 10 + 1 \\ &\Rightarrow 0 <> 11 \\ &\Rightarrow \text{verdadeiro} \end{aligned}$$

2. Suponha que x seja uma variável do tipo inteiro e considere a relação

$$x \% 3 > 1$$

Então, para quais valores de x a relação acima tem valor verdadeiro?

solução:

O resultado de $x \% 3$ é sempre $-2, -1, 0, 1$ ou 2 . Como a relação terá valor verdadeiro se, e somente se, $x \% 3$ for igual a 2 , temos que o valor de x deve ser $2, 5, 8, \dots$, ou seja, da forma

$$2 + 3 \cdot k,$$

para todo $k \in \mathbb{Z}_+^*$.

6.5 Exercícios propostos

1. Suponha que $a, b, nome$ e $profissao$ sejam variáveis do tipo real, inteiro, caractere e caractere, respectivamente. Então, considere as três expressões relacionais dadas a seguir:

$$a + 1 \geq b * b, \quad nome \langle \rangle \text{“ANA”} \quad \text{e} \quad profissao = \text{“medico”}$$

Qual é o valor dessas expressões quando $a, b, nome$ e $profissao$ assumem os valores abaixo?

- a) $3.0, 2, \text{“MIRIAM”}$ e “ADVOGADO”
 - b) $5.0, 2 \cdot \sqrt{2}, \text{“PEDRO”}$ e “MEDICO”
 - c) $2.5, 9, \text{“ANA”}$ e “PROFESSOR”
2. Suponha que x seja uma variável do tipo real e considere a seguinte expressão relacional:

$$x^2 - 4 > 5.$$

Então, para quais valores de x a expressão relacional acima possui valor verdadeiro?

3. Suponha que x seja uma variável do tipo inteiro. Então, escreva uma expressão relacional em Português que tenha valor verdadeiro se, e somente se, o valor de x é um número ímpar não-negativo.
4. Sejam a e b duas variáveis do tipo caractere. Então, escreva uma expressão relacional envolvendo a e b apenas que tenha resultado verdadeiro se, e somente se, a expressão relacional,

$$a < b,$$

tenha resultado falso.

ESTRUTURAS CONDICIONAIS - PARTE 1

7.1 Motivação

Até o presente momento, todos os algoritmos que vimos podem ser vistos como uma seqüência de comandos que são executados na *ordem* em que definida pela seqüência. Nesta aula, veremos um comando que nos permite “bifurcar” a seqüência de comandos de um algoritmo. Com isso, poderemos criar trechos de algoritmos que são mutuamente exclusivos com respeito à execução. Isto é, um trecho é executado se, e somente se, outro trecho não é executado.

Antes de apresentarmos o comando condicional, apresentamos um problema que justifica a existência de tal comando. Considere o problema de escrever um algoritmo para ler nome e altura de duas pessoas e produzir como saída o nome da pessoa mais alta. **Vamos assumir que as pessoas possuam alturas distintas.** Um algoritmo incompleto para o problema é mostrado em 7.1.

```
1 algoritmo "Pessoa mais alta - incompleto"
2 var
3     nome1, nome2 : caractere
4     alt1, alt2 : real
5 inicio
6     escreva("Entre com o nome da primeira pessoa: ")
7     leia(nome1)
8
9     escreva( "Entre com a altura da primeira pessoa: " )
10    leia( alt1 )
11
12    escreva( "Entre com o nome da segunda pessoa: " )
13    leia( nome2 )
14
15    escreva( "Entre com a altura da segunda pessoa: " )
16    leia( alt2 )
17
18    // Escreva o nome da pessoa de maior altura
19    // ?
20 fimalgoritmo
```

Algoritmo 7.1: Cálculo (incompleto) da mais alta de duas pessoas.

Tudo que precisamos fazer para concluir o Algoritmo 7.1 é escrever sua saída. De fato, se a

primeira pessoa for mais alta do que a segunda, então podemos usar a seguinte linha de código:

```
escreva(nome1, " é mais alto(a) do que ", nome2 )
```

Caso contrário, podemos usar a linha

```
escreva(nome2, " é mais alto(a) do que ", nome1 )
```

O problema é que não sabemos qual das duas linhas acima deve ser inserida no algoritmo *antes* de lermos e compararmos as duas alturas. Em outras palavras, o algoritmo deve determinar qual das duas linhas acima deve ser escrita durante a sua execução. Esta é uma situação de *exclusão mútua* de dois comandos, pois somente um deles pode ser escrito. No entanto, até agora, não aprendemos nenhum comando que nos permita escrever um algoritmo que execute um comando em detrimento da execução de outro. Sem tal comando, não temos como concluir o algoritmo.

7.2 Comando se-entao-senao-fimse

A linguagem Portugol possui o seguinte **comando condicional**:

```
1 se <expr> entao
2     <cmdo_1>
3     ..
4     <cmdo_n>
5 senao
6     <cmdo_1>
7     ..
8     <cmdo_m>
9 fimse
```

O comando se-entao-senao-fimse contém duas seções, cada uma das quais possui um ou mais comandos. A primeira delas é a que se situa entre o entao e o senao. A segunda é a que se situa entre o senao e o fimse. As seções são mutuamente exclusivas, o que significa que uma é executada se, e somente se, a outra não é executada. A seção a ser executada depende do valor lógico da *expressão lógica* que se situa entre o se e o então. Por enquanto, assumamos que uma expressão lógica é uma expressão relacional (veja Aula 7). Na aula seguinte, veremos expressões lógicas mais gerais. Se a expressão lógica resultar em o valor verdadeiro, a primeira seção é executada. Caso contrário, isto é, se ela avaliar para o valor falso, a segunda seção é executada.

É importante ressaltar que independente da seção de código que for executada, o fluxo de execução do algoritmo segue para o primeiro comando *após* o se-então-senão-fimse assim que o último comando da seção for executado. Isto é, o comando condicional “bifurca” o fluxo de execução, mas após o comando condicional ser executado, o programa segue com seu fluxo sequencial único que continua com o primeiro comando encontrado após o comando condicional.

O comando se-então-senão-fimse pode ser utilizado para completar o algoritmo que começamos a escrever na seção anterior. Em particular, o trecho que falta pode ser escrito como segue

```
se alt1 > alt2 entao
    escreva( nome1 , " é mais alto(a) do que " , nome2 )
senao
```

```
        escreva( nome2 , " é mais alto(a) do que " , nome1 )
    fimse
```

De fato, a expressão lógica,

```
alt1 > alt2,
```

avalia para o valor verdadeiro se, e somente se, a altura da primeira pessoa é maior do que a da segunda. Caso contrário, sabemos que a altura da primeira pessoa *não* é maior do que a da segunda. Mas, como assumimos que as alturas são distintas, se a expressão relacional resultar em falso, então saberemos que a segunda pessoa é mais alta do que a primeira. O Algoritmo 7.2 é o Algoritmo 7.1 acrescido do comando se-entao-senao-fimse que gera a saída esperada.

```
1  algoritmo "Pessoa mais alta - primeira versao"
2  var
3      nome1, nome2 : caractere
4      alt1 , alt2 : real
5  inicio
6      escreva( "Entre com o nome da primeira pessoa: " )
7      leia( nome1 )
8
9      escreva( "Entre com a altura da primeira pessoa: " )
10     leia( alt1 )
11
12     escreva( "Entre com o nome da segunda pessoa: " )
13     leia( nome2 )
14
15     escreva( "Entre com a altura da segunda pessoa: " )
16     leia( alt2 )
17
18     se alt1 > alt2 entao
19         escreva( nome1 , " é mais alto(a) do que " , nome2 )
20     senao
21         escreva( nome2 , " é mais alto(a) do que " , nome1 )
22     fimse
23 fimalgoritmo
```

Algoritmo 7.2: Algoritmo para determinar a mais alta de duas pessoas.

7.3 Aninhamento de comandos condicionais

Alguns problemas podem requerer o “aninhamento” de comandos condicionais, ou seja, que um ou mais comandos condicionais ocorram dentro da seção de comandos de outro comando condicional. Para exemplificar esta situação, vamos considerar o problema da seção anterior e assumir que as alturas das duas pessoas não necessitem ser distintas. Então, a saída do problema envolve as seguintes três (e não apenas duas) instruções escreva mutuamente exclusivas:

```
escreva( nome1 , " é mais alto(a) do que " , nome2 )
```

```
escreva( nome2 , " é mais alto(a) do que " , nome1 )
```

e

```
escreva( nome1 , " possui a mesma altura de " , nome2 )
```

Quando a expressão

```
alt1 > alt2
```

resultar em verdadeiro, a primeira instrução acima deve ser executada. No entanto, quando a expressão resultar em falso, ainda teremos que determinar qual das duas instruções restantes deve ser executada. Isto pode ser feito com uma instrução se-entao-senao-fimse dentro do bloco senao que envolve a expressão relacional acima. Em outras palavras, podemos escrever algo como:

```
se alt1 > alt2 entao
    escreva( nome1 , " é mais alto(a) do que " , nome2 )
senao
    se alt1 < alt2 entao
        escreva( nome2 , " é mais alto(a) do que " , nome1 )
    senao
        escreva( nome1 , " possui a mesma altura de " , nome2 )
    fimse
fimse
```

Note que se o fluxo de execução do algoritmo atinge o bloco senao do comando se-entao-senao-fimse mais externo, então sabemos que a segunda pessoa não é mais baixa do que a primeira, mais ainda não sabemos se ela é mais alta ou possui a mesma altura da primeira. O comando se-entao-senao-fimse mais externo determina qual desses dois casos é verdadeiro e escreve a saída esperada. O Algoritmo 7.3 contém o código final para determinar a pessoa mais alta.

Um outro problema que requer aninhamento de comandos condicionais é o seguinte: escreva um algoritmo que leia as notas (de 0 a 10) de três provas de um aluno, calcule a média aritmética das três notas do aluno e escreva o *status* dele como saída. O *status* do aluno é “aprovado” se a média das notas é igual ou maior do que 7, “exame” se a média é igual ou maior do que 3, mas menor do que 7, e “reprovado” se a média é menor do que 3. Note que os *status* são mutuamente exclusivos, isto é, para qualquer valor encontrado para a média, apenas um *status* é possível.

Note que um único comando se-entao-senao-fimse não resolve o problema, pois tal comando só pode realizar uma classificação “binária”, ou seja, em um de dois grupos. Por exemplo, podemos usar o comando condicional para saber se a média é maior ou igual a 7. Isto é suficiente para sabermos se o aluno foi “aprovado” quando a média é maior ou igual a 7. No entanto, se a média for menor do que 7, não temos como saber se o aluno está em “exame” ou foi “reprovado”.

A observação crucial é que se a média for menor do 7, podemos determinar o *status* do aluno se usarmos mais um comando se-entao-senao-fimse dentro da seção de senao do se-entao-senao-fimse que já temos. Isto é, se o fluxo de execução desviar para a seção senao é porque a média é menor do 7. Daí, para determinarmos se o *status* é “exame” ou “reprovado”, basta compararmos o valor de *media* com 3. Se este valor for maior ou igual a 3, o *status* do aluno é “exame”. Caso contrário, o *status* do aluno é “reprovado”. O algoritmo completo pode ser visto em 7.4.

7.4 Comando se-então-fimse

O comando se-então-senão-fimse possui uma forma simplificada: o comando se-então-fimse:

```
se expressão lógica então  
    comando1  
    comando2  
    ⋮  
    comandon  
fimse
```

No comando acima, há apenas uma seção de comandos entre o se e o fimse. Se a expressão lógica avalia para o valor verdadeiro, então a seção de comandos é executada. Caso contrário, a seção de comandos não é executada. Em qualquer um desses dois casos, o fluxo de execução segue com o comando imediatamente após a palavra fimse. Em outras palavras, ao invés de termos dos blocos de comandos mutuamente exclusivos, temos um bloco de comandos que pode ou não ser executado. O que ocorrerá depende unicamente do resultado da expressão lógica.

Por exemplo, considere o Algoritmo 7.5 que calcula o salário líquido mensal de um funcionário. O algoritmo recebe como entrada o valor do salário bruto mensal do funcionário e o número de filhos dele. O salário líquido é calculado como sendo 70% do salário bruto. Mas, se o funcionário possuir mais de dois filhos, ele recebe um acréscimo de R\$ 300,00 em seu salário líquido.

7.5 Exercícios resolvidos

1. Escreva a saída do Algoritmo 2 quando ele for executado nas seguintes entradas:

(a) 2

(b) 21

Solução:

(a) O algoritmo escreve como saída a sentença “Passei pelo ponto 3” quando a entrada for 2.

(b) O algoritmo escreve como saída a sentença “Passei pelo ponto 1” quando a entrada for 21.

2. Escreva um algoritmo que leia um número inteiro positivo com quatro dígitos e escreva “sim” se a soma dos algarismos da centena e milhar do número é par e “não” caso contrário.

Solução:

7.6 Exercícios propostos

1. Escreva um algoritmo que leia dois números reais e escreva “iguais” se os números forem iguais e “diferentes” se os números forem diferentes.

2. Escreva um algoritmo que leia um número real e escreva “sim” se o número está compreendido no intervalo $[20, 90]$ e “não” caso contrário.
3. Escreva um algoritmo que leia dois números reais e escreva o menor deles. Se os números forem iguais, qualquer um deles por ser escrito (pois são os mesmos).
4. Escreva um algoritmo que leia dois números reais e escreva o maior deles. Se os números forem iguais, qualquer um deles por ser escrito (pois são os mesmos).
5. Escreva um algoritmo que leia um número inteiro e escreva como saída “múltiplo de 3” se o número for múltiplo de 3 ou “não é múltiplo de 3” caso contrário. Lembre-se de que um número n é múltiplo de 3 se ele é divisível por 3, ou seja, se o resto da divisão inteira de n por 3 é zero.
6. Escreva um algoritmo que leia o salário de um funcionário e escreva o imposto de renda correspondente. Considere as alíquotas de 15% para salário de até R\$ 2.500,00 e de 20% para salário de mais de R\$ 2.500,00.
7. Escreva um algoritmo que leia o peso (Kg) e a altura de uma pessoa (m) e calcule e escreva seu índice de massa corporal (IMC). O valor do IMC é dado pelo cálculo do peso dividido pela altura elevada ao quadrado. O algoritmo deve ainda escrever "peso normal", caso o IMC seja de até 25.0 e "acima do peso" caso o IMC seja maior do que 25.0.
8. (Adaptado de KOLIVER et al., 2009) Escreva um algoritmo que leia o nome de dois clientes de uma loja e o valor (em reais) que cada um desses clientes pagou por sua compra. O algoritmo deverá escrever: (a) o valor total pago pelos dois clientes; (b) o valor médio das compras efetuadas; (c) os nomes dos clientes que efetuaram compras superiores a 20 reais.

```

1 algoritmo "Pessoa mais alta - segunda versao"
2 var
3     nome1, nome2 : caractere
4     alt1 , alt2 : real
5 inicio
6     escreva( "Entre com o nome da primeira pessoa: " )
7     leia( nome1 )
8
9     escreva( "Entre com a altura da primeira pessoa: " )
10    leia( alt1 )
11
12    escreva( "Entre com o nome da segunda pessoa: " )
13    leia( nome2 )
14
15    escreva( "Entre com a altura da segunda pessoa: " )
16    leia( alt2 )
17
18    se alt1 > alt2 entao
19        escreva( nome1 , " é mais alto(a) do que " , nome2 )
20    senao
21        se alt1 < alt2 entao
22            escreva( nome2 , " é mais alto(a) do que " , nome1 )
23        senao
24            escreva( nome1 , " possui a mesma altura de " , nome2 )
25        fimse
26    fimse
27 fimalgoritmo

```

Algoritmo 7.3: Algoritmo para determinar a mais alta de duas pessoas.

```

1 algoritmo "Situacao escolar de aluno"
2 var
3     n1, n2, n3, media : real
4 inicio
5     escreva( "Entre com a primeira nota: " )
6     leia( n1 )
7
8     escreva( "Entre com a segunda nota: " )
9     leia( n2 )
10
11    escreva( "Entre com a terceira nota: " )
12    leia( n3 )
13
14    media <- ( n1 + n2 + n3 ) / 3
15
16    se media >= 7 entao
17        escreva( "aprovado" )
18    senao
19        se media > 3 entao
20            escreva( "em exame" )
21        senao
22            escreva( "reprovado" )
23        fimse
24    fimse
25 fimalgoritmo

```

Algoritmo 7.4: Algoritmo para determinar situação escolar de aluno.

```

1 algoritmo "Salario liquido mensal de funcionario"
2 var
3     salbruto, salario : real
4     filhos : inteiro
5 inicio
6     escreva( "Entre com o salário bruto mensal: " )
7     leia ( salbruto )
8
9     escreva( "Entre com o número de filhos: " )
10    leia ( filhos )
11
12    salario <- 0.7 * salbruto
13    se filhos > 2 entao
14        salario <- salario + 300
15    fimse
16
17    escreva ( "O salário líquido é: ", salario )
18 fimalgoritmo

```

Algoritmo 7.5: Algoritmo para calcular salário líquido mensal de um funcionário.

```

1 algoritmo "Soma de dígitos de centena e milhar"
2 var
3     n, m, c : inteiro
4 inicio
5     escreva ( "Entre com um número inteiro positivo:" )
6     leia ( n )
7     m <- n \ 1000
8     c <- n \ 100 % 10
9     se ( c + m ) % 2 = 0 entao
10        escreva ( "sim" )
11     senao
12        escreva ( "não" )
13     fimse
14 fimalgoritmo

```

Algoritmo 7.6: Soma dígitos de centena e milhar

```

1 algoritmo "Misterioso"
2 var
3     n : inteiro
4 inicio
5     escreva ( "Entre com um número inteiro positivo:" )
6     leia ( n )
7     se ( n % 2 ) = 1 entao
8         se ( n % 7 ) = 0 entao
9             escreva ( "Passei pelo ponto 1" )
10        senao
11            escreva ( "Passei pelo ponto 4" )
12        fimse
13    senao
14        se ( n % 4 ) = 0 entao
15            escreva ( "Passei pelo ponto 2" )
16        senao
17            escreva ( "Passei pelo ponto 3" )
18        fimse
19    fimse
20 fimalgoritmo

```

Algoritmo 7.7: Algoritmo “Misterioso”. Qual é a saída dele?

EXPRESSÕES LÓGICAS

8.1 Lógica proposicional

Lógica é o estudo do raciocínio¹. Em particular, utilizamos lógica quando desejamos determinar se um dado raciocínio está correto. Nesta disciplina, introduzimos algumas noções básicas de *Lógica Proposicional* para que sejamos capazes de entender alguns tipos de dados e expressões utilizados nos algoritmos que desenvolveremos. No entanto, a relação entre lógica, resolução de problemas e programação de computadores é muito mais ampla, rica e complexa do que a discussão que apresentamos aqui.

A **Lógica Proposicional** consiste de uma *linguagem* e de um formalismo de *cálculo* para falar e deduzir fatos, respectivamente, sobre *proposições*. Uma **proposição** é uma sentença declarativa à qual podemos atribuir um valor *verdadeiro* ou *falso*. Há vários tipos de sentenças:

- Imperativas: “Multiplique 2 por 3.”
- Exclamativas: “Que cerveja gelada!”
- Interrogativas: “Está chovendo lá fora?”
- Declarativas: “Todo aluno da UFRN é maior de idade.”

O que distingue as sentenças declarativas das demais é o fato de que à elas podemos atribuir um valor verdadeiro ou falso, embora nem sempre sejamos capazes de saber que valor atribuir. Em lógica, assumimos que as proposições satisfazem dois princípios:

1. **Terceiro Excluído**: uma proposição ou é verdadeira ou é falsa, isto é, não existe uma terceira possibilidade.
2. **Não-Contradição**: uma proposição não pode ser, ao mesmo tempo, verdadeira e falsa.

As sentenças: “os únicos inteiros positivos que dividem 7 são 1 e o próprio 7” e “para todo inteiro positivo n , existe um primo maior do que n ” são exemplos de proposições.

Aqui, usamos letras minúsculas, tais como p , q e r , para representar proposições e adotamos a notação

$$p : 1 + 1 = 3$$

para definir p como sendo a proposição $1 + 1 = 3$.

¹Conjunto de argumentos ou idéias pensadas por alguém.

8.2 Proposições compostas

A linguagem utilizada em lógica para representar proposições e realizar cálculos sobre elas foi cuidadosamente desenvolvida para evitar ambiguidades. Este não é o caso da língua portuguesa, que nos permite facilmente construir sentenças ambíguas:

- Grandes carros e aviões.

O que é grande? Só os carros ou carros e aviões?

- José está vendo o jogo em cima das dunas.

Quem está em cima das dunas? O jogo? José? Ambos?

A linguagem que usaremos para construir algoritmos e as linguagens de programação também não são ambíguas, mas não servem para descrever argumentos, conhecimento, verdades, etc. É por isso que estudaremos uma linguagem própria para falar de proposições. Uma das características desta linguagem é o uso de *conectivos lógicos* para criar novas proposições, *proposições compostas*, a partir de proposições existentes.

Sejam p e q duas proposições. A **conjunção** de p e q , representada por $p \wedge q$, é a proposição

$$p \text{ e } q.$$

A **disjunção** de p e q , representada por $p \vee q$, é a proposição

$$p \text{ ou } q.$$

Por exemplo, se

$$p: \quad 1 + 1 = 3$$

e

$$q: \quad \text{uma década equivale a 10 anos}$$

então a conjunção de p e q é

$$p \wedge q: \quad 1 + 1 = 3 \text{ e uma década equivale a 10 anos}$$

e a disjunção de p e q é

$$p \vee q: \quad 1 + 1 = 3 \text{ ou uma década equivale a 10 anos.}$$

Os valores verdades das proposições, tais como conjunções e disjunções, podem ser descritos através de *tabelas verdades*. A **tabela verdade** de uma proposição p definida a partir das proposições p_1, \dots, p_n lista todas as possíveis combinações de valores lógicos de p_1, \dots, p_n , com T denotando verdadeiro e F denotando falso, e para cada combinação desses valores lógicos, a tabela verdade lista o valor lógico de p .

O valor lógico da proposição composta $p \wedge q$ é definido pela tabela verdade [8.1](#).

Por exemplo, se

$$p: \quad 1 + 1 = 3$$

e

$$q: \quad \text{uma década equivale a 10 anos}$$

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

Tabela 8.1: Tabela verdade da conjunção.

então p é falsa e q é verdadeira, o que implica que a conjunção

$$p \wedge q : \quad 1 + 1 = 3 \quad \text{e} \quad \text{uma década equivale a 10 anos}$$

é falsa.

O valor lógico da proposição composta $p \vee q$ é definido pela tabela verdade 8.2:

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

Tabela 8.2: Tabela verdade da disjunção.

Por exemplo, se

$$p : \quad 1 + 1 = 3$$

e

$$q : \quad \text{uma década equivale a 10 anos}$$

então p é falsa e q é verdadeira, o que implica que a disjunção

$$p \vee q : \quad 1 + 1 = 3 \quad \text{ou} \quad \text{uma década equivale a 10 anos}$$

é verdadeira.

Existe ainda uma outra proposição importante:

Seja p uma proposição qualquer. Então, a **negação** de p , denotada por \bar{p} , é a proposição

não é verdade que p .

O valor lógico da proposição \bar{p} é definido pela tabela verdade 8.3.

p	\bar{p}
V	F
F	V

Tabela 8.3: Tabela verdade da negação.

Por exemplo, se

$$p : \quad 1 + 1 = 3$$

e

q : uma década equivale a 10 anos

então p é falsa e q é verdadeira, o que implica que

\bar{p} : não é verdade que $1 + 1 = 3$

é verdadeira e

\bar{q} : não é verdade que uma década equivale a 10 anos

é falsa.

Nós podemos utilizar conjunção, disjunção e negação para construir uma nova proposição a partir de duas proposições, onde cada uma delas pode ser uma proposição composta. Quando isto acontece, usamos parênteses e regras de precedência para determinar, de forma não-ambígua, como avaliar o valor lógico da proposição resultante.

Por exemplo, se p , q e r são proposições, então

$$(p \wedge \bar{q}) \vee r$$

também é uma proposição. Como podemos avaliar o valor lógico dessa proposição? Nós supomos que o operador de negação possui precedência sobre os conectivos de conjunção e disjunção. Então, a proposição

$$p \wedge \bar{q}$$

significa a conjunção de p com \bar{q} . Isto é, o operador de negação atua sobre q antes que o conectivo de conjunção atue sobre p e q .

Finalmente, quando temos mais duas proposições conectadas por \wedge ou \vee , usamos parênteses para determinar a ordem de composição das proposições. Por exemplo,

$$(p \wedge \bar{q}) \vee r$$

significa a disjunção da proposição $p \wedge \bar{q}$ com a proposição r . Isto é, os parênteses servem para determinar que a conjunção de p com \bar{q} deve ocorrer antes da disjunção de $p \wedge \bar{q}$ com r .

Se os parênteses fossem removidos, isto é, se tivéssemos

$$p \wedge \bar{q} \vee r$$

não poderíamos determinar se a sentença acima se trata da conjunção de p com $\bar{q} \vee r$ ou da disjunção de $p \wedge \bar{q}$ com r .

Agora, se supusermos que p é **V**, q é **V** e r é **F**, o valor lógico de $(p \wedge \bar{q}) \vee r$ é

$$\begin{aligned}(p \wedge \bar{q}) \vee r &= (\mathbf{V} \wedge \bar{\mathbf{V}}) \vee \mathbf{F} \\ &= (\mathbf{V} \wedge \mathbf{F}) \vee \mathbf{F} \\ &= \mathbf{F} \vee \mathbf{F} \\ &= \mathbf{F}.\end{aligned}$$

Em alguns casos, no entanto, o uso de parênteses é desnecessário. Por exemplo,

$$(p \vee q) \wedge (\bar{p} \vee q) \wedge (p \vee \bar{q}) \wedge (\bar{p} \vee \bar{q})$$

No caso acima, não importa a ordem em que agrupamos as proposições definidas dentro dos parênteses, pois o valor lógico da proposição resultante será sempre o mesmo. Isto porque o conectivo “fora” que conecta as proposições parentizadas é o mesmo: \wedge .

8.3 Operadores lógicos

Como vimos na Aula 7, uma relação é, na verdade, uma proposição, pois ela é uma sentença declarativa (em particular, uma comparação entre dois valores) cujo valor só pode ser verdadeiro ou falso. Logo, é bastante natural nos perguntarmos se podemos *combinar* relações usando algum operador, assim como fizemos com as proposições que vimos na Seção 8.2 usando conectivos lógicos, conjunção e disjunção. A resposta é *sim*. Em particular, podemos construir **expressões lógicas**, que são expressões cujo resultado é um valor lógico. Toda relação é, portanto, uma expressão lógica. No entanto, uma expressão lógica pode consistir de mais de uma relação, as quais são combinadas através dos **operadores lógicos**. No Portugal da ferramenta VISUALG, os operadores lógicos são os mostrados na Tabela 8.4.

Operador	Descrição
<u>nao</u>	Negação
<u>e</u>	Conjunção
<u>xou</u>	Disjunção Exclusiva
<u>ou</u>	Disjunção

Tabela 8.4: Operadores lógicos da linguagem Portugal.

Suponha que a , b e c são variáveis do tipo inteiro. Então,

$$(a > b + c) \text{ e } (c \leq 5 - a)$$

é uma conjunção das relações

$$a > b + c$$

e

$$c \leq 5 - a.$$

O resultado da avaliação da expressão lógica

$$(a > b + c) \text{ e } (c \leq 5 - a)$$

(ou seja, da *conjunção* das duas relações acima) será o valor verdadeiro se, e somente se, o resultado das **duas** relações for o valor verdadeiro. Por outro lado, se a expressão lógica for

$$(a > b + c) \text{ ou } (c \leq 5 - a)$$

então o resultado da avaliação da expressão lógica (ou seja, da *disjunção*) será o valor verdadeiro se, e somente se, o resultado de **uma ou de ambas** as relações for o valor verdadeiro. Já

$$\text{nao } (a > b + c)$$

é uma negação da relação

$$a > b + c.$$

O resultado da avaliação da expressão lógica (ou seja, da *negação*) será o valor verdadeiro se, e somente se, o resultado da avaliação da relação for o valor falso.

As expressões lógicas podem combinar mais de duas relações. Por exemplo,

$$(a <> 4 + b) \text{ ou } (2 * 5 \% c = 1) \text{ e } (a \leq 5 - c)$$

e

$$\underline{\text{nao}} (c * 2 > 10) \underline{\text{ou}} (c - 3 <> 4) \underline{\text{ou}} (b > c * 4).$$

Como a utilização demasiada de parênteses pode prejudicar a legibilidade da expressão, o uso de regras de precedência de operadores é sempre útil. A Tabela 8.5 exibe a prioridade dos operadores lógicos da linguagem Portugol da ferramenta VISUALG. Estas prioridades podem ser modificadas pelo uso de parênteses, assim como fizemos com as expressões aritméticas.

Prioridade	Operador
mais alta	<u>nao</u>
↑	<u>e</u>
↑	<u>xou</u>
mais baixa	<u>ou</u>

Tabela 8.5: Prioridade de todos os operadores da linguagem Portugol vistos até o momento.

De acordo com essas precedências, o valor da expressão lógica

$$(2 > 3) \underline{\text{e}} (3 < 2) \underline{\text{ou}} (2 < 3)$$

é verdadeiro, pois

$$\begin{aligned} (2 > 3) \underline{\text{e}} (3 < 2) \underline{\text{ou}} (2 < 3) &\Rightarrow \underline{\text{falso}} \underline{\text{e}} (3 < 2) \underline{\text{ou}} (2 < 3) \\ &\Rightarrow \underline{\text{falso}} \underline{\text{e}} \underline{\text{falso}} \underline{\text{ou}} (2 < 3) \\ &\Rightarrow \underline{\text{falso}} \underline{\text{ou}} (2 < 3) \\ &\Rightarrow \underline{\text{falso}} \underline{\text{ou}} \underline{\text{verdadeiro}} \\ &\Rightarrow \underline{\text{verdadeiro}}. \end{aligned}$$

Por outro lado, se o operador ou tivesse maior prioridade do que o operador e, então

$$(2 > 3) \underline{\text{e}} (3 < 2) \underline{\text{ou}} (2 < 3)$$

avaliaria para falso, pois

$$\begin{aligned} (2 > 3) \underline{\text{e}} (3 < 2) \underline{\text{ou}} (2 < 3) &\Rightarrow (2 > 3) \underline{\text{e}} \underline{\text{falso}} \underline{\text{ou}} (2 < 3) \\ &\Rightarrow (2 > 3) \underline{\text{e}} \underline{\text{falso}} \underline{\text{ou}} \underline{\text{verdadeiro}} \\ &\Rightarrow (2 > 3) \underline{\text{e}} \underline{\text{verdadeiro}} \\ &\Rightarrow \underline{\text{falso}} \underline{\text{e}} \underline{\text{verdadeiro}} \\ &\Rightarrow \underline{\text{falso}}. \end{aligned}$$

No exemplo acima, as relações foram “cercadas” com parênteses. Na Lógica Proposicional, esses parênteses seriam desnecessários, pois *qualquer operador relacional possui prioridade sobre todo operador lógico*. No entanto, **a ferramenta VISUALG exige que as relações que compõem uma expressão lógica sejam cercadas por parênteses**. Logo, usando a linguagem Portugol desta ferramenta, as expressões lógicas acima *não* podem ser reescritas como mostrado a seguir:

$$2 > 3 \underline{\text{e}} 3 < 2 \underline{\text{ou}} 2 < 3.$$

8.4 Exercícios resolvidos

1. Avalie as seguintes expressões lógicas:

a) falso ou ($10 \% 5 * 2 <> 5 * 2 + 1$)

b) nao falso e ($3 * 3 \setminus 3 < 15 - 5 \% 7$)

solução:

a)

$$\begin{aligned} \text{falso ou} (10 \% 5 * 2 <> 5 * 2 + 1) &\Rightarrow \text{falso ou} (0 * 2 <> 5 * 2 + 1) \\ &\Rightarrow \text{falso ou} (0 <> 5 * 2 + 1) \\ &\Rightarrow \text{falso ou} (0 <> 10 + 1) \\ &\Rightarrow \text{falso ou} (0 <> 11) \\ &\Rightarrow \text{falso ou verdadeiro} \\ &\Rightarrow \text{verdadeiro} . \end{aligned}$$

b)

$$\begin{aligned} \text{nao falso e} (3 * 3 \setminus 3 < 15 - 5 \% 7) &\Rightarrow \text{verdadeiro e} (3 * 3 \setminus 3 < 15 - 5 \% 7) \\ &\Rightarrow \text{verdadeiro e} (9 \setminus 3 < 15 - 5 \% 7) \\ &\Rightarrow \text{verdadeiro e} (3 < 15 - 5 \% 7) \\ &\Rightarrow \text{verdadeiro e} (3 < 15 - 5) \\ &\Rightarrow \text{verdadeiro e} (3 < 10) \\ &\Rightarrow \text{verdadeiro e verdadeiro} \\ &\Rightarrow \text{verdadeiro} . \end{aligned}$$

2. Suponha que x seja uma variável do tipo inteiro e considere a seguinte expressão lógica:

$$(x \% 3 = 0) \text{ e } (x \% 7 = 0)$$

Então, para quais valores de x a expressão lógica acima avalia para o valor verdadeiro?

solução:

Para todo valor inteiro que seja um múltiplo comum de 3 e de 7. Por exemplo, -21 e 21 .

3. Suponha que x seja uma variável do tipo inteiro. Então, escreva uma expressão lógica envolvendo x que avalie para o valor verdadeiro se, e somente se, o valor de x for par e não for maior do que 11.

solução:

$$(x \% 2 = 0) \text{ e } (x \leq 11)$$

8.5 Exercícios propostos

1. Avalie o valor da expressão $p \text{ e } (q \text{ ou } r)$ quando sabe-se que:

- a) p é verdade, q é falso e r é falso
 b) p é verdade, q é verdade e r é falso
 c) p é falso, q é falso e r é verdade
2. Resolva as expressões lógicas:
- a) nao ($2 > 3$)
 b) ($6 < 8$) ou ($3 > 7$)
 c) nao ($2 <> 2.0$)
 d) ($5 >= 6$) ou ($6 < 7$) ou nao ($a + 5 - 6 = 8$), onde $a = 5$
 e) ($(34 < 9) \text{ e } (5 + u = 34)$) ou ($(5 = 15/3) \text{ e } (8 > 12)$), onde $u = 29$
3. Avalie as seguintes expressões lógicas:
- a) nao ($7 <> 15 \setminus 2$) ou verdadeiro e ($4 - 6 > 4 - 20$)
 b) ($2 * 5 > 3$) ou ($5 + 1 < 2$) e ($2 < 7 - 2$)
4. Suponha que x seja uma variável do tipo real e considere a seguinte expressão lógica:

$$x * x - 4 > 5$$

Então, para quais valores de x a expressão lógica acima avalia para o valor falso?

5. Suponha que x seja uma variável do tipo logico. Então, escreva uma expressão lógica envolvendo x que avalie para o valor falso se, e somente se, o valor de x for verdadeiro.
6. Suponha que x seja uma variável do tipo logico. Então, escreva uma expressão lógica envolvendo x que avalie para o valor verdadeiro se, e somente se, o valor de x for falso.
7. Suponha que x seja uma variável do tipo inteiro. Então, escreva uma expressão lógica envolvendo x que avalie para o valor verdadeiro se, e somente se, o valor de x for par ou não for maior do que 11, mas não ambos.
8. Escreva um algoritmo para determinar se um aluno está aprovado ou reprovado em uma disciplina baseando-se em sua *porcentagem de faltas*, *média parcial* e *nota do exame final*. Um aluno para ser aprovado precisa cumprir as seguintes condições:
- Sua porcentagem de faltas não deve ultrapassar 25%;
 - Sua *média parcial* deve ser igual ou maior que 7.0, ou a soma de sua *média parcial* e de sua *nota do exame final* deve ser igual ou maior que 10.0.

Os valores de *porcentagem de faltas*, *média parcial* e *nota do exame final* do aluno devem ser lidos pelo algoritmo. A saída do algoritmo deve ser "*aluno aprovado*" ou "*aluno reprovado*".

ESTRUTURAS CONDICIONAIS - PARTE 2

9.1 Usando proposições compostas

Considere o problema de se calcular a média harmônica ponderada de três notas de prova. As notas variam de 0 a 10 e os pesos são 1, 2 e 3 para a primeira, segunda e terceira notas, respectivamente. A fórmula da média harmônica ponderada, digamos mh , para os pesos dados acima é

$$mh = \begin{cases} \frac{6}{\frac{1}{n_1} + \frac{2}{n_2} + \frac{3}{n_3}} & \text{se } n_1, n_2 \text{ e } n_3 \text{ são todas diferentes de } 0 \\ 0 & \text{caso contrário.} \end{cases}$$

onde n_1 , n_2 e n_3 são os valores da primeira, segunda e terceira notas, respectivamente. Note que a fórmula para se calcular mh é condicional. Em outras palavras, se todas as notas são distintas de zero, então o valor de mh , é inversamente proporcional a uma soma de frações. Caso contrário, isto é, quando pelo menos uma das notas é igual a zero, o valor de mh , é igual a zero.

Agora, vamos escrever um algoritmo para calcular mh . A entrada do algoritmo consiste dos valores das três notas, n_1 , n_2 e n_3 . A saída do algoritmo é o valor de mh . Para obter este valor, podemos utilizar a fórmula acima. No entanto, como a fórmula possui duas “partes”, nosso algoritmo precisará de um comando condicional. Este comando deve testar se todas as notas são diferentes de zero. Como este teste pode ser escrito? Pelo que vimos na aula anterior, o teste desejado pode ser escrito como uma composição de três relações usando o conectivo de conjunção:

$$(nota1 \neq 0) \text{ e } (nota2 \neq 0) \text{ e } (nota3 \neq 0)$$

onde $nota1$, $nota2$ e $nota3$ são os nomes das variáveis que armazenam os valores de n_1 , n_2 e n_3 , respectivamente. Se a expressão lógica acima avaliar para verdadeiro, então o valor de mh é calculado por:

$$mh = \frac{6}{\frac{1}{n_1} + \frac{2}{n_2} + \frac{3}{n_3}}.$$

Caso contrário, o valor de mh deve ser igual zero. Um comando se-então-senão-fimse com a expressão lógica acima é tudo que precisamos para fazer o cálculo de mh . O algoritmo completo está em 9.1.

É interessante notar que o comando se-então-senão-fimse poderia ser substituído por um comando se-então-fimse se removermos a seção senao e inicializarmos o valor da variável *media* com 0 imediatamente acima do comando condicional, como mostrado no seguinte trecho de código:

```

media <- 0
se ( nota1 <> 0 ) e ( nota2 <> 0 ) e ( nota3 <> 0 ) entao
    media <- 6 / ( 1 / nota1 + 2 / nota2 + 3 / nota3 )
fimse

```

```

1 algoritmo "Media harmonica ponderada"
2 var
3     nota1, nota2, nota3, media : real
4 inicio
5     escreva( "Entre com a primeira nota: " )
6     leia( nota1 )
7     escreva( "Entre com a segunda nota: " )
8     leia( nota2 )
9     escreva( "Entre com a terceira nota: " )
10    leia( nota3 )
11    se ( nota1 <> 0 ) e ( nota2 <> 0 ) e ( nota3 <> 0 ) entao
12        media <- 6 / ( 1 / nota1 + 2 / nota2 + 3 / nota3 )
13    senao
14        media <- 0
15    fimse
16    escreva( "A media harmonica ponderada das tres notas e: " , media )
17 fimalgoritmo

```

Algoritmo 9.1: Cálculo da média harmônica ponderada de três notas

9.2 Troca de conteúdo entre duas variáveis

Quando escrevemos algoritmos mais complexos é comum nos depararmos com a tarefa de trocar os conteúdos de duas variáveis. Isto é, se x e y são duas variáveis do mesmo tipo, trocar os conteúdos de x e y significa atribuir o valor de x a y e atribuir o valor de y a x . Muitos iniciantes no estudo de algoritmos podem pensar em cumprir esta tarefa escrevendo o trecho de código

```

y <- x
x <- y

```

ou

```

x <- y
y <- x

```

que *não* estão corretos. Para entender o porquê dos trechos acima não estarem corretos, vamos supor que x e y sejam variáveis do tipo inteiro e que possuam os valores 4 e 7, respectivamente, antes de qualquer um dos trechos de código ser executado. Como qualquer variável só pode armazenar um único valor de seu tipo em um mesmo instante de tempo, a execução do trecho de código

```

y <- x
x <- y

```

resulta em x e y com o valor 4. De fato, quando a instrução $y \leftarrow x$ é executada, o valor de x , que é 4 antes da atribuição, é atribuído à variável y , que continha o valor 7 antes da atribuição. Desta forma, o antigo valor de y é substituído por 4 e, após a instrução, tanto x quanto y possuem 4 como conteúdo. A próxima instrução, $x \leftarrow y$, simplesmente atribui o valor atual de y , que agora é 4, à variável x , que já possui o valor 4. Logo, ao final das duas instruções, as duas variáveis possuem valor 4. O mesmo tipo de análise mostra que outro trecho de código,

```
x ← y
y ← x
```

deixa tanto x quanto y com valor 7. Como podemos realizar esta “simples” tarefa de forma correta?

A forma correta de trocar o valor de duas variáveis se utiliza de uma variável extra, comumente denominada de *variável temporária* ou *variável auxiliar*. Esta variável serve para armazenar, temporariamente, o valor que será sobrescrito por uma instrução de atribuição. Por exemplo, se queremos executar a instrução $y \leftarrow x$, então devemos guardar o valor de y , que será sobrescrito pela instrução, em algum local “seguro” de modo que ele possa ser usado quando necessitarmos. Este local seguro é a tal variável temporária. De forma concreta, suponha que z seja a variável temporária. Então, antes de executar a instrução $y \leftarrow x$, executamos $z \leftarrow y$. Isto faz com que o valor de y seja guardado em z antes de y receber o valor que está em x . Finalmente, atribuímos o antigo valor de y , que agora está em z , a x através da instrução $x \leftarrow z$.

O trecho de código correto é, portanto,

```
z ← y
y ← x
x ← z
```

Note que se x e y possuem os valores 4 e 7 antes do trecho acima ser executado, então $z \leftarrow y$ faz com que z receba o valor 7, $y \leftarrow x$ faz com que y receba o valor 4 e $x \leftarrow z$ faz com que x receba o valor 7. Logo, ao final da execução do código acima, os valores em x e y são 7 e 4, respectivamente.

Para ilustrar a ocorrência de uma troca de valores de variáveis em um algoritmo, vamos considerar um problema extremamente simples: escrever em ordem não-decrescente dois números fornecidos como entrada para o problema. Um algoritmo para este problema deve ler os dois números, determinar qual deles é o menor e qual deles é o maior e escrever o menor seguido pelo maior. Quando os números forem iguais, a ordem de escrita dos números é indiferente.

Um algoritmo para o problema acima é dado em 9.2. Note que a idéia é trocar os valores das duas variáveis que recebem os números de entrada sempre que o primeiro número for maior do que o segundo. Se este não for o caso, a troca não é feita. Desta forma, podemos escrever um comando de saída que sempre escreve o valor da primeira variável seguido pelo valor da segunda. Como o valor da primeira variável é sempre menor ou igual ao valor da segunda variável imediatamente antes da instrução de escrita ser executada, o algoritmo sempre fornecerá a resposta esperada. Um outro algoritmo, que não usa troca de variáveis, é dado em 9.3.

9.3 O comando escolha

O uso do comando se-então-senão-fimse de forma aninhada e com vários níveis de aninhamento pode dificultar a leitura do algoritmo. Em tais situações, o melhor é usar um outro comando de estrutura condicional fornecido pela linguagem Portugol da ferramenta VISUALG: o comando escolha. Este comando permite que o resultado de uma única expressão, denominada *expressão de seleção*, seja comparado com os resultados de várias expressões. Essas expressões se encontram agrupadas e cada grupo está associado a uma seqüência de comandos. Se o valor de qualquer uma das expressões de um grupo é igual ao valor da expressão de seleção, todos os comandos associados ao grupo são executados. A sintaxe do comando escolha é mostrada abaixo:

```
escolha expressão de seleção  
    caso expressão 1, expressão 2, ..., expressão n1  
        seqüência de comandos  
    caso expressão 1, expressão 2, ..., expressão n2  
        seqüência de comandos  
    ⋮  
    outrocaso  
        seqüência de comandos  
fimescolha
```

Em geral, a expressão de seleção e as expressões de cada grupo são expressões aritméticas ou simplesmente constantes, tais como números, letras e frases. A palavra reservada caso representa um grupo de expressões (as quais estão descritas à direita da palavra caso) e um grupo de comandos (que está logo abaixo da palavra caso). A palavra reservada outrocaso representa o grupo de comandos executados quando o valor da expressão de seleção não é igual ao valor de nenhuma expressão dos grupos anteriores. É possível que grupos distintos contêmam expressões que avaliem para o mesmo valor. Se este mesmo valor é igual ao valor da expressão de seleção, apenas os comandos do grupo mais próximo da expressão de seleção são executados.

Para ilustrar o comando escolha, considere o problema de classificar os atletas de um clube de futebol por categorias que se distinguem pela idade do atleta: Infantil (de 5 a 10 anos), Juvenil (de 11 a 15 anos), Junior (de 16 a 20 anos) e Profissional (de 21 a 25 anos). O nosso objetivo é construir um algoritmo que lê o nome e a idade de um atleta e escreve como saída o nome da categoria à qual ele pertence (“Infantil”, “Juvenil”, “Junior” ou “Profissional”). Se o atleta não pertence a nenhuma das categorias acima, o algoritmo deve escrever “Nenhuma categoria”.

O problema pode ser resolvido naturalmente com o uso do comando escolha, pois cada categoria está relacionada a um grupo de valores (as idades). Um algoritmo que utiliza o comando escolha para resolver o problema é dado em 9.3. Como exercício, reescreva o mesmo algoritmo usando comandos se-então-senão-fimse aninhados e compare seu algoritmo com o Algoritmo 9.3.

9.4 Exercícios propostos

1. Escreva um algoritmo que leia um número inteiro e escreva como saída “divisível por 3 e 7” se o número for divisível por 3 e por 7 ou “não é divisível por 3 e 7” caso contrário.

2. Crie um algoritmo que solicite o nome de um time de futebol ao usuário. Se o nome do time informado for "Flamengo", "Fluminense", "Vasco" ou "Botafogo", escreva "É um time carioca". Se o nome do time informado for "São Paulo", "Palmeiras", "Santos" ou "Corinthians", escreva "É um time paulista". Se o nome do time não for nenhum dos citados anteriormente, escreva "Time desconhecido".
3. A Prefeitura de Natal abriu uma linha de crédito para os funcionários estatutários. O valor máximo da prestação de um empréstimo não pode ultrapassar 30% do salário bruto do funcionário. Escreva um algoritmo que leia o nome de um funcionário, seu salário bruto e o valor da prestação do empréstimo que ele solicitou e, em seguida, escreva como saída o nome do funcionário seguido da mensagem "teve o crédito concedido" se o empréstimo solicitado puder ser concedido ou seguido da mensagem "crédito negado" caso contrário.
4. Escreva um algoritmo que leia um número real, n , e escreva "menor que 20", "igual a 20" ou "maior que 20" se $n < 20$, $n = 20$ ou $n > 20$, respectivamente.
5. Escreva um algoritmo que leia um número inteiro positivo com três dígitos e escreva como saída "par" se o dígito da centena é par e "ímpar" caso contrário.
6. Escreva um algoritmo que leia um número inteiro positivo com quatro dígitos e escreva como saída "sim" se a soma dos dígitos da unidade e da centena são múltiplos de 4 e "não" caso contrário.
7. Escreva um algoritmo que leia nome, sexo e idade de uma pessoa e escreva o nome e a mensagem "aceita" se a pessoa for do sexo feminino e tiver mais de 25 anos ou for do sexo masculino e tiver mais do que 30 anos. Caso contrário, o algoritmo deve escrever o nome e a mensagem "não aceita". O dado sexo deve ser informado com a letra M ou a letra F (maiúsculas).
8. Um comerciante compra um determinado produto para revender. O valor de revenda é calculado da seguinte forma: se o comerciante pagar menos de R\$ 20,00 pelo produto, o valor de revenda é tal que o comerciante obtenha um lucro de 45%; se o valor de compra é maior ou igual a R\$ 20,00, o valor de revenda é tal que o comerciante obtenha um lucro de 30%. Então, escreva um algoritmo que leia o valor de compra de um produto e calcule e escreva o valor de revenda do produto.
9. Escreva um algoritmo que leia dois números reais e os escreva em ordem não-decrescente.
10. Escreva um algoritmo que leia dois números reais e os escreva em ordem não-crescente.
11. Escreva um algoritmo que leia três números reais distintos e os escreva em ordem não-decrescente.
12. Escreva um algoritmo que leia cinco números reais distintos e escreva o maior e o menor deles.
13. Escreva um algoritmo que leia três números reais positivos e escreva "sim" se os três números podem ser as medidas dos lados de um triângulo e "não" caso contrário. Lembre-se de que três números podem ser as medidas dos lados de um triângulo se, e somente se, cada um deles é menor do que a soma dos outros dois.
14. Escreva um algoritmo que leia três números reais e escreva "equilátero" se eles formam os lados de um triângulo equilátero, "isósceles" se eles formam os lados de um triângulo isósceles, "escaleno" se eles formam os lados de um triângulo escaleno e "não formam os lados de um triângulo" se eles não formam os lados de um triângulo.

```

1 algoritmo "Numeros em ordem nao-decrescente - primeira versao"
2 var
3     num1, num2, num3 : real
4 inicio
5     escreva( "Entre com o primeiro numero: " )
6     leia( num1 )
7     escreva( "Entre com o segundo numero: " )
8     leia( num2 )
9     se num1 > num2 entao
10        num3 <- num1
11        num1 <- num2
12        num2 <- num3
13     fimse
14     escreva( "Os numeros em ordem nao-decrescente: " , num1 , " e " , num2 )
15 fimalgoritmo

```

Algoritmo 9.2: Escrita de dois números em ordem não-decrescente

```

1 algoritmo "Numeros em ordem nao-decrescente - segunda versao"
2 var
3     num1, num2 : real
4 inicio
5     escreva( "Entre com o primeiro numero: " )
6     leia( num1 )
7     escreva( "Entre com o segundo numero: " )
8     leia( num2 )
9     se num1 > num2 entao
10        escreva( "Os numeros em ordem nao-decrescente: " , num2 , " e " , num1 )
11     senao
12        escreva( "Os numeros em ordem nao-decrescente: " , num1 , " e " , num2 )
13     fimse
14 fimalgoritmo

```

Algoritmo 9.3: Escrita de dois números em ordem não-decrescente v2

```

1 algoritmo "Atletas por categorias de idade"
2 var
3     nome, categoria : caractere
4     idade : inteiro
5 inicio
6     escreva( "Entre com o nome do atleta: " )
7     leia( nome )
8     escreva( "Entre com a idade do atleta: " )
9     leia( idade )
10    escolha idade
11        caso 5, 6, 7, 8, 9, 10
12            categoria <- "Infantil"
13        caso 11, 12, 13, 14, 15
14            categoria <- "Juvenil"
15        caso 16, 17, 18, 19, 20
16            categoria <- "Junior"
17        caso 21, 22, 23, 24, 25
18            categoria <- "Profissional"
19        outrocaso
20            categoria <- "Nenhuma categoria"
21    fimescolha
22    escreva( "O atleta ", nome, " pertence a categoria ", categoria )
23 fimalgoritmo

```

Algoritmo 9.4: Classificar atletas por categorias de idade

ESTRUTURAS DE REPETIÇÃO - PARTE 1

10.1 O comando enquanto-faca-fimenquanto

Considere o problema de escrever um algoritmo para ler um número inteiro positivo, n , e escrever todos os números inteiros de 1 a n como saída. Por mais simples que este problema possa parecer, não temos como resolvê-lo com os comandos que aprendemos até aqui. A razão é que o valor de n não é conhecido antes de escrevermos o algoritmo. Logo, não temos como escrever um algoritmo com n comandos do tipo escreva(i), onde i é um inteiro de 1 a n . No entanto, se tivéssemos uma forma de “repetir” a execução do comando escreva um número variável de vezes, poderíamos facilmente resolver nosso problema. De fato, o que gostaríamos é de poder escrever algo como:

```

i <- 1
execute n vezes as duas linhas abaixo:
  escreva(i)
  i <- i + 1

```

A linguagem Portugol da ferramenta VISUALG contém um comando, o enquanto-faca-fimenquanto, que faz exatamente o que queremos. Este comando possui a sintaxe descrita abaixo:

```

enquanto expressão lógica faca
  comando1
  comando2
  :
  comandon
fimenquanto

```

O comando enquanto-faca-fimenquanto, mais conhecido como **laço enquanto**, funciona da seguinte forma: a expressão lógica, denominada **condição** do laço, é primeiramente avaliada. Se o valor da expressão for verdadeiro, a seqüência de comandos do **corpo do laço**, isto é, a seção de comandos delimitada pelas palavras reservadas faca e fimenquanto, é executada e a expressão é novamente avaliada. Caso contrário, o primeiro comando após a palavra fimenquanto é executado. Note que se a expressão lógica que define a condição do laço avaliar para falso durante a primeira avaliação, a seqüência de comandos do corpo do laço não é executada nenhuma vez.

Usando o comando enquanto-faca-fimenquanto, podemos facilmente escrever o trecho de código que realiza a escrita dos números de 1 a n . Para tal, precisamos definir a condição do laço. Como

queremos executar o corpo do laço n vezes, uma escolha natural é a condição que testa se o valor de i é menor ou igual ao valor de n , $i \leq n$. Enquanto esta condição for verdadeira, o comando de escrita é executado e o valor de i é incrementado em uma unidade. Em particular, temos:

```
 $i \leftarrow 1$   
enquanto  $i \leq n$  faca  
    escreva( $i$ )  
     $i \leftarrow i + 1$   
fimenquanto
```

A variável i realiza dois papéis importantes no trecho de código acima. Ela serve para enumerar os valores que serão escritos pelo algoritmo e para contar o número de vezes em que o corpo do laço é executado. Devido a este último papel, a variável recebe o nome de variável **contadora**. Para compreender como o trecho de algoritmo acima é executado, vamos supor que o valor de n seja 4. Se este for o caso, então os seguintes passos são executados pelo trecho acima:

1. atribui o valor 1 à variável i
2. compara o valor de i com 4
3. escreve o valor de i
4. incrementa o valor de i em uma unidade
5. compara o valor de i com 4
6. escreve o valor de i
7. incrementa o valor de i em uma unidade
8. compara o valor de i com 4
9. escreve o valor de i
10. incrementa o valor de i em uma unidade
11. compara o valor de i com 4
12. escreve o valor de i
13. incrementa o valor de i em uma unidade
14. compara o valor de i com 4

A Tabela 10.1 mostra o valor de i antes e depois de cada um dos passos acima. Note que o corpo do laço é executado exatamente 4 vezes (passos 3, 4, 6, 7, 9, 10, 12 e 13). Após a execução do passo 13, o valor da variável contadora, i , passa a ser 5, o que faz com que a expressão lógica que define a condição do laço (avaliada nos passos 2, 5, 8, 11 e 14) avalie para falso pela primeira vez.

O algoritmo completo para o problema de escrever os número de 1 a n é mostra no Algoritmo 10.1.

10.2 Exemplos

O laço enquanto nos permite resolver alguns problemas bem mais complexos do que o que vimos na Seção 10.1. Como exemplo, considere o problema de escrever um algoritmo para ler dois números inteiros, a e b , com $b > a$, e calcular e escrever a soma de todos os números entre a e b , incluindo os próprios extremos a e b na soma. O problema acima pode ser resolvido sem o uso de um laço enquanto, mas a solução requer o uso de uma fórmula. Por outro lado, podemos usar o laço enquanto para “enumerar” todos os números inteiros de a a b e calcular a soma desejada de forma “incremental”. Esta solução consiste de duas etapas que são discutidas a seguir.

A enumeração de todos os números entre a e b , inclusive, pode ser feita como segue:

```
 $i \leftarrow a$   
enquanto  $i \leq b$  faça  
     $i \leftarrow i + 1$   
fimenquanto
```

Note que o corpo do laço acima é executado $b - a + 1$ vezes e, em cada execução, a variável i possui um valor distinto no intervalo $[a, b] \subset \mathbb{Z}$. O próximo passo é acumular, de forma incremental, o valor de i em uma variável. Para tal, fazemos uso de uma variável **acumuladora** como segue:

```
 $soma \leftarrow 0$   
 $i \leftarrow a$   
enquanto  $i \leq b$  faça  
     $soma \leftarrow soma + i$   
     $i \leftarrow i + 1$   
fimenquanto
```

O nome variável acumuladora vem do fato que, ao final da j -ésima execução do corpo do laço, para $j \in \{1, \dots, b - a + 1\}$, o valor de $soma$ é igual à soma dos valores $a, a + 1, a + 2, \dots, a + j - 1$. Isto significa que a variável acumula a soma dos valores que já foram enumerados pelo laço. Ao final da última iteração, ou seja, quando j assumir o valor $b - a + 1$, a variável acumuladora será igual à soma dos números $a, a + 1, a + 2, \dots, b$, que é o valor desejado.

O algoritmo completo é mostrado no Algoritmo 10.2.

Variáveis acumuladoras podem acumular valores de somas, subtrações, multiplicações e divisões. Um algoritmo no qual usamos uma variável acumuladora para acumular o valor de multiplicações é o que calcula o fatorial de um número inteiro. Por definição, o fatorial, $n!$, de n é dado por

$$n! = \begin{cases} 1 & \text{se } n = 0 \text{ ou } n = 1 \\ n \cdot (n - 1)! & \text{se } n > 1 \end{cases}.$$

A definição acima é *recursiva*, pois o fatorial de n é definido em termos do fatorial de $(n - 1)$. No entanto, nós aprendemos que a definição recursiva equivale à definição não-recursiva dada a seguir:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n - 1) \cdots 2 \cdot 1 & \text{se } n > 0 \end{cases}.$$

Usando a definição não-recursiva e uma variável acumuladora, temos o algoritmo dado em 10.2. Note que a variável acumuladora “acumula” uma multiplicação e não uma soma. Logo, ela deve ser inicializada com 1 e não com zero, que foi o caso da variável *soma* do Algoritmo 10.2. Note também que a variável contadora é inicializada com o número de cujo fatorial queremos calcular e que ela é decrementada de uma unidade a cada execução do corpo do laço.

10.3 Exercícios propostos

1. Escreva um algoritmo que leia um número inteiro positivo, n , e escreva os n primeiros números pares positivos. Por exemplo, dado $n = 4$, o algoritmo deveria escrever como saída os números 2, 4, 6 e 8.
2. Escreva um algoritmo que leia um número inteiro positivo, n , e escreva os n primeiros inteiros positivos ímpares. Por exemplo, dado $n = 4$, o algoritmo deveria escrever como saída os números 1, 3, 5 e 7.
3. Escreva um algoritmo que leia um número inteiro positivo, n , e escreva o quadrado dos n primeiros inteiros positivos.
4. Escreva um algoritmo que leia dois números inteiros positivos, a e b , com $a < b$, e calcule e escreva o quadrado de todos os números ímpares no intervalo $[a, b]$.
5. Escreva um algoritmo que leia dois números inteiros positivos, a e b , com $a < b$, e calcule e escreva a média aritmética de todos os números pares compreendidos no intervalo $[a, b]$.
6. Dizemos que um número inteiro positivo, n , é *perfeito* se n for igual à soma de seus divisores positivos diferentes de n . Por exemplo, $n = 28$ é perfeito, pois $1 + 2 + 4 + 7 + 14 = 28$. Escreva um algoritmo que leia um número inteiro positivo, n , verifique se n é perfeito e escreva “é perfeito” em caso afirmativo e “não é perfeito” caso contrário.
7. Qualquer número inteiro positivo de quatro algarismos pode ser dividido em duas dezenas formadas pelos seus dois primeiros e dois últimos dígitos. Por exemplo,

1297 : 12 e 97.
5314 : 53 e 14.

Escreva um algoritmo que imprima todos os números de quatro algarismos cuja raiz quadrada seja a soma das dezenas formadas pela divisão acima. Por exemplo,

$$\sqrt{9801} = 99 = 98 + 01.$$

Portanto, 9801 é um dos números a serem escritos pelo algoritmo. Note que este algoritmo não possui nenhum dado de entrada!

8. Dado um inteiro positivo, n , o *número harmônico*, H_n , é definido pela soma

$$H_n = \sum_{k=1}^n \frac{1}{k}.$$

Escreva um algoritmo que leia n e escreva como saída o valor de H_n .

9. Escreva um algoritmo que leia um número inteiro n e escreva os números recíprocos (inversos multiplicativos) dos n primeiros inteiros positivos. O recíproco de um número i é dado por $1/i$.

10. Escreva um algoritmo que leia o preço e a quantidade de cinco produtos diferentes de uma loja e escreva o valor total da compra.
11. Escreva um algoritmo que leia uma quantidade indeterminada de números inteiros positivos e escreva:
 - a) Quantos deles estão no intervalo [0..25]
 - b) Quantos estão no intervalo [26..50]
 - c) Quantos são maiores do que 50.

O algoritmo deve parar quando o usuário digitar um número negativo.

12. Escreva um algoritmo que leia um número inteiro e escreva o quadrado desse número enquanto ele for positivo. Se um inteiro negativo for digitado, o algoritmo deve parar e escrever uma mensagem informando que um número negativo foi digitado.
13. Escreva um algoritmo que leia um número inteiro positivo n e escreva quantos divisores positivos ele possui.

Passo	Valor de i antes	Valor de i depois
1	<i>desconhecido</i>	1
2	1	1
3	1	1
4	1	2
5	2	2
6	2	2
7	2	3
8	3	3
9	3	3
10	3	4
11	4	4
12	4	4
13	4	5
14	5	5

Tabela 10.1: O valor i antes e depois de cada comando do trecho de algoritmo que ilustra o laço enquanto.

```

1 algoritmo "Inteiros de 1 a n"
2 var
3     num, i : inteiro
4 inicio
5     escreva( "Entre com um numero inteiro positivo: " )
6     leia( num )
7     i <- 1
8     enquanto ( i <= num ) faca
9         escreva( i , " " )
10        i <- i + 1
11    fimenquanto
12 fimalgoritmo

```

Algoritmo 10.1: Escrita dos inteiros de 1 a n

```

1 algoritmo "Soma de inteiros em um intervalo"
2 var
3   a, b, i, soma : inteiro
4 inicio
5   escreva( "Entre com o menor inteiro do intervalo: " )
6   leia( a )
7   escreva( "Entre com o maior inteiro do intervalo: " )
8   leia( b )
9   soma <- 0
10  i <- a
11  enquanto ( i <= b ) faça
12    soma <- soma + i
13    i <- i + 1
14  fimenquanto
15  escreva( "A soma dos numeros do intervalo e: ", soma )
16 fimalgoritmo

```

Algoritmo 10.2: Soma dos inteiros de um dado intervalo

```

1 algoritmo "Fatorial de inteiro nao-negativo"
2 var
3   num, i , fat : inteiro
4 inicio
5   escreva( "Entre com um numero inteiro nao-negativo: " )
6   leia( num )
7   fat <- 1
8   i <- num
9   enquanto ( i > 1 ) faça
10    fat <- fat * i
11    i <- i - 1
12  fimenquanto
13  escreva( "O fatorial de ", num , " e: ", fat )
14 fimalgoritmo

```

Algoritmo 10.3: Fatorial de um inteiro não negativo

ESTRUTURAS DE REPETIÇÃO 2

11.1 A seqüência de Fibonacci

Um problema parecido, mas ligeiramente mais complicado do que o do cálculo do fatorial (veja as notas da Aula 14), é o do cálculo do n -ésimo termo da seqüência de Fibonacci, f_n , onde:

$$f_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ f_{n-1} + f_{n-2} & \text{se } n > 1 \end{cases}$$

Em outras palavras, os dois primeiros termos da seqüência, f_0 e f_1 , são iguais a 0 e 1, respectivamente. A partir do terceiro termo, o valor de qualquer termo, f_i , com $i \geq 2$, é igual a soma dos dois termos anteriores:

$$f_i = f_{i-1} + f_{i-2}.$$

Vamos escrever um algoritmo que leia um inteiro positivo, n , e calcule e escreva o n -ésimo termo, f_n , da seqüência de Fibonacci. A idéia por trás deste algoritmo é calcular f_n de forma incremental, isto é, o algoritmo começa com a inicialização de duas variáveis, a e b , com os valores $f_0 = 0$ e $f_1 = 1$. Em seguida, ele calcula o valor de f_2 como sendo $a + b$. O valor resultante é armazenado em uma variável temporária, t . Antes de calcular f_3 , o algoritmo copia o valor de b para a e o valor de t para b . Desta forma, temos que $a = f_2$ (o antigo valor de b) e $b = f_3$ (o atual valor de t). Em seguida, o valor de f_4 é calculado como $a + b$ novamente, pois $f_4 = f_3 + f_2$. Se observarmos bem, a soma $a + b$ foi repetida no cálculo de f_3 e f_4 . O que acabamos de descrever pode ser expresso como

```

a ← 0
b ← 1
i ← 1
enquanto i ≤ n faça
    t ← a + b
    a ← b
    b ← t
    i ← i + 1
fimenquanto

```

Se supusermos que $n = 3$, o laço acima calcula o valor de f_3 da seguinte forma:

1. a recebe o valor 0.
2. b recebe o valor 1.
3. i recebe o valor 1.
4. O valor de $i = 1$ é comparado com o valor $n = 3$. Como $i = 1 \leq n = 3$, o corpo do laço é executado pela primeira vez, para calcular f_1 .
5. t recebe o valor de $f_2 = a + b = 1$.
6. a recebe o valor $f_1 = b = 1$.
7. b recebe o valor de $t = f_2 = 1$.
8. O valor de i é incrementado e se torna 2.
9. O valor de $i = 2$ é comparado com o valor $n = 3$. Como $i = 2 \leq n = 3$, o corpo do laço é executado pela segunda vez, para calcular f_2 .
10. t recebe o valor de $f_3 = a + b = 2$.
11. a recebe o valor $f_2 = b = 1$.
12. b recebe o valor de $t = f_3 = 2$.
13. O valor de i é incrementado e se torna 3.
14. O valor de $i = 3$ é comparado com o valor $n = 3$. Como $i = 3 \leq n = 3$, o corpo do laço é executado pela terceira vez, para calcular f_3 .
15. t recebe o valor de $f_4 = a + b = 3$.
16. a recebe o valor $f_3 = b = 2$.
17. b recebe o valor de $t = f_4 = 3$.
18. O valor de i é incrementado e se torna 4.
19. O valor de $i = 4$ é comparado com o valor $n = 3$. Como $i = 4 > n = 3$, o corpo do laço não é mais executado. Note que a contém o valor de f_3 .

Um algoritmo para calcular o n -ésimo termo da seqüência de Fibonacci é dado em 11.2. Note que o algoritmo também fornece a resposta correta quando $n = 0$. De fato, como o valor da variável i é 1 na primeira vez em que a condição do laço é avaliada, o corpo do laço não é executado para $n = 0$. Logo, o valor de a escrito pelo algoritmo é igual ao valor que fn .

11.2 Inversão da ordem dos dígitos de um número

Suponha que desejemos desenvolver um algoritmo para ler um número inteiro não-negativo, n , e escrever como saída o número inteiro correspondente a n , quando n é lido da direita para a esquerda. Isto é, queremos calcular um número que correspondente ao número obtido quando invertemos a ordem dos dígitos de n . Por exemplo, se $n = 43$, então o algoritmo deveria escrever o número 34; se $n = 120$, então o algoritmo deveria escrever o número 21; se $n = 1304$, então o algoritmo deveria escrever o número 4031; e, se $n = 5$, então o algoritmo deveria escrever o número 5. Qual estratégia devemos usar para construir o algoritmo desejado?

Uma possível estratégia é a seguinte: se os algarismos de n são d_1, d_2, \dots, d_k , com $k \geq 1$

quando enumerados da direita para a esquerda, então o número que queremos calcular é igual a

$$d_1 \times 10^{(k-1)} + d_2 \times 10^{(k-2)} + \dots + d_k \times 10^{(0)} = d_1 \times 10^{(k-1)} + d_2 \times 10^{(k-2)} + \dots + d_k.$$

Por exemplo, se $n = 43$ então o número a ser escrito é igual a

$$3 \times 10^1 + 4 \times 10^0 = 30 + 4 = 34;$$

se $n = 120$ então o número a ser escrito é igual a

$$0 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 0 + 20 + 1 = 21;$$

se $n = 1304$ então o número a ser escrito é igual a

$$4 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 = 4000 + 0 + 30 + 1 = 4031;$$

e se $n = 5$ então o número a ser escrito é igual a

$$5 \times 10^0 = 5 \times 1 = 5.$$

O problema com a estratégia acima é que nós não temos os algarismos individuais, d_1, d_2, \dots, d_k , de n . Mas, com o que aprendemos até o presente momento, somos capazes de calculá-los.

O algarismo da unidade pode ser obtido com a operação *resto*, $n \% 10$, da divisão de n por 10. Os demais algarismos podem ser obtidos por uma seqüência de operações de divisão inteira por 10 seguida por uma operação resto de divisão por 10. Por exemplo, se $n = 43$ então o algarismo da unidade é igual a $n \% 10 = 43 \% 10 = 3$ e o algarismo da dezena é igual a $(n \setminus 10) \% 10 = (43 \setminus 10) \% 10 = 4 \% 10 = 4$. Por sua vez, se $n = 120$ então o algarismo da unidade é igual a $n \% 10 = 120 \% 10 = 0$ e os algarismos da dezena e centena são obtidos como segue:

$$(n \setminus 10) \% 10 = (120 \setminus 10) \% 10 = 12 \% 10 = 2$$

e

$$((n \setminus 10) \setminus 10) \% 10 = ((120 \setminus 10) \setminus 10) \% 10 = (12 \setminus 10) \% 10 = 1 \% 10 = 1.$$

Em geral, para $k > 1$, o k -ésimo algarismo da direita para a esquerda, pode ser obtido por uma seqüência de $(k - 1)$ operações de divisão (inteira) por 10 seguida por uma operação de resto de divisão por 10. Isto implica também que se o número n possui exatamente j dígitos, uma seqüência de j divisões inteiras por 10 resultará no valor zero. Por exemplo, se $n = 5$ então

$$n \setminus 10 = 5 \setminus 10 = 0;$$

se $n = 43$ então

$$(n \setminus 10) \setminus 10 = (43 \setminus 10) \setminus 10 = 4 \setminus 10 = 0;$$

e se $n = 120$ então

$$((n \setminus 10) \setminus 10) \setminus 10 = ((120 \setminus 10) \setminus 10) \setminus 10 = (12 \setminus 10) \setminus 10 = 1 \setminus 10 = 0.$$

Uma vez que saibamos como calcular cada algarismo do número n , como podemos calcular o número que desejamos? Uma outra observação nos levará a uma solução bastante elegante. Sejam d_1 e d_2 dois dígitos quaisquer. Se quisermos obter o número $d_1 d_2$ a partir de d_1 e d_2 , basta multiplicarmos d_1 por 10 e somar o resultado a d_2 : $d_1 \times 10 + d_2$. Por exemplo, se $d_1 = 4$ e $d_2 = 3$,

o número $d_1d_2 = 43$ é igual a $d_1 \times 10 + d_2 = 4 \times 10 + 3$. De forma análoga, se quisermos obter o número $d_1d_2d_3$ a partir de d_1d_2 e d_3 , onde d_3 é outro dígito qualquer, basta multiplicarmos d_1d_2 por 10 e somar o resultado a d_3 : $d_1d_2 \times 10 + d_3$. Por exemplo, se $d_1d_2 = 43$ e $d_3 = 7$, o número $d_1d_2d_3 = 437$ é igual a $d_1d_2 \times 10 + d_3 = 43 \times 10 + 7 = 437$. Agora, considere o seguinte laço enquanto:

```

m <- n % 10
enquanto (n \ 10) <> 0 faça
    n <- n \ 10
    m <- m * 10 + (n % 10)
fimenquanto

```

Note que as duas linhas repetidas serão executadas $j - 1$, onde j é o número de algarismos de n . Em cada execução, o atual valor de m é multiplicado por 10 e somado com o “próximo” dígito de n . O algarismo d_1 é calculado antes do laço ser executado; isto é, a variável m recebe o valor $n \% 10$, que é igual a d_1 . Na primeira execução do corpo do laço, o valor de m se torna

$$(d_1 \times 10) + d_2.$$

Na próxima execução do corpo do laço, o valor de m se torna

$$((d_1 \times 10) + d_2) \times 10 + d_3,$$

e assim por diante até que, na última execução do laço, o valor de m se torna

$$d_1 \times 10^{(j-1)} + d_2 \times 10^{(j-2)} + \dots + d_j,$$

que é o número desejado. O Algoritmo 11.2 ilustra a solução completa para o problema que estudamos. Note que o algoritmo utiliza uma variável chamada q ao invés de n no cálculo de m . Isto é uma “boa” prática, que evita mudança de valor e dois usos distintos da variável de entrada.

11.3 Teste de primalidade

Um número **primo**, n , é um número inteiro maior do que 1 que só é divisível por 1 e por ele próprio. Um número inteiro maior do que 1 que não é primo é dito **composto**. Suponha que desejemos escrever um algoritmo para determinar se um dado número inteiro, n , com $n \geq 2$, é primo ou composto. A entrada do algoritmo é o número n e a saída é a sentença “é primo” se n é primo e “é composto” caso contrário.

A estratégia de solução que utilizaremos consiste em tentar dividir n por $2, 3, \dots, n - 1$. Se sucedermos então o número n não é primo. Caso contrário, ele é. As divisões podem ser realizadas em um laço, pois elas consistem da mesma operação. No entanto, como nem sempre realizaremos todas as divisões, não temos como saber quantas divisões serão executadas. Esta última observação é um forte indício de que precisamos de um laço enquanto. Mais especificamente, testaremos se n é divisível por i , para $i = 2, 3, \dots, n - 1$. No entanto, assim que determinarmos que n é divisível por i , não precisaremos continuar com os testes, pois já saberemos que n é composto.

Em outras palavras, podemos construir um laço tal como

```

i <- 2
enquanto (n % i) <> 0 faca
  i <- i + 1
fimenquanto

```

O laço acima terminará se o valor de i atingir n ou se n for divisível por i , para algum valor de i no intervalo $[2, n - 1] \subset \mathbb{Z}$. Mas, como saberemos se o número é primo ou composto depois que o laço for encerrado? A resposta é simples: basta notar que se i atingir o valor de n , então i foi incrementada de 2 até n , o que só é possível se o corpo do laço tiver sido repetido para os valores de i iguais a $2, 3, \dots, n - 1$. Mas, por sua vez, todas essas repetições só podem ocorrer se o teste $(n \% i) \neq 0$ resultar em verdadeiro para todos esses valores de i . Bom, mas quando isto ocorre, sabemos que n é primo. Por outro lado, se o valor de i não atingir n , então o teste $(n \% i) \neq 0$ resultou em falso para algum valor de i no intervalo $[2, n - 1]$. Caso contrário, o laço só se encerraria quando i atingisse o valor n . Então, podemos concluir que se $i < n$ depois do laço, o número n é composto, pois n é divisível por i e pelo quociente, q , da divisão inteira de n por i . Como $i > 1$, o valor de q também é maior do que 1. Logo, n possui dois divisores maiores do que 1 (i e q), o que implica que n deve ser um número composto. Então, o comando condicional

```

se i < n entao
  escreva( "e composto" )
senao
  escreva( "e primo" )
fimse

```

pode ser escrito após o laço enquanto para determinar se n é primo ou composto com base no valor da variável i quando o laço encerrar. A solução do teste de primalidade é mostrada no Algoritmo 11.3.

11.4 Exercícios propostos

1. Escreva um algoritmo que leia três números inteiros, a , b e n , com $n > 0$, e escreva como saída o n -ésimo termo, f_n , da seqüência, $(f_i)_{i=1}^{\infty}$, tal que

$$f_i = \begin{cases} a & \text{se } i = 1 \\ b & \text{se } i = 2 \\ f_{i-1} + f_{i-2} & \text{se } i > 2 \end{cases} .$$

2. Escreva um algoritmo que leia três números inteiros, a , b e n , com $n > 0$, e escreva como saída o n -ésimo termo, f_n , da seqüência, $(f_i)_{i=1}^{\infty}$, tal que

$$f_i = \begin{cases} a & \text{se } i = 1 \\ b & \text{se } i = 2 \\ f_{i-1} + f_{i-2} & \text{se } i > 2 \text{ e } i \text{ é ímpar} \\ f_{i-1} - f_{i-2} & \text{se } i > 2 \text{ e } i \text{ é par} \end{cases} .$$

3. Dizemos que um número inteiro positivo, n , tal que n possui pelo menos 2 dígitos e n não possui nenhum dígito 0, é *palíndromo* se, e somente se, o primeiro dígito de n é igual ao seu

último dígito, o segundo dígito de n é igual ao seu penúltimo dígito e assim sucessivamente. Por exemplo, 567765 é palíndromo, 32423 é palíndromo, mas 567675 não é palíndromo. Escreva um algoritmo que leia um número inteiro positivo, n , verifique se n é palíndromo e escreva “é palíndromo” em caso afirmativo e “não é palíndromo” caso contrário.

4. Uma maneira de calcular o valor do número π é utilizar a seguinte série

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

Escreva um algoritmo que leia um número inteiro positivo, n , e calcule e escreva o valor de π através da série acima, levando em conta apenas os números com precisão de pelo menos n casas decimais. Isto é, adicione apenas os termos cujo valor absoluto seja maior ou igual a 10^{-n} .

5. Escreva um algoritmo que leia um número real x e um número inteiro positivo, n , e calcule e escreva uma aproximação para $\cos x$ usando os n primeiros termos da seguinte série

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^k \frac{x^{2k}}{(2k)!} + \dots$$

6. Escreva um algoritmo que leia dois números inteiros positivos e escreva como saída o Mínimo Múltiplo Comum (MMC) desses dois números. Lembre-se de que o MMC de dois números, digamos a e b , é o menor número inteiro positivo que é um múltiplo tanto de a quanto de b .
7. Escreva um algoritmo que leia dois números inteiros positivos e escreva como saída o Máximo Divisor Comum (MDC) desses dois números. Lembre-se de que o MDC de dois números, digamos a e b , é o maior número inteiro positivo que é um divisor tanto de a quanto de b .
8. Dizemos que um número inteiro positivo, n , é *triangular* se ele é o produto de três números naturais consecutivos. Por exemplo, 120 é triangular, pois $4 \times 5 \times 6 = 120$. Escreva um algoritmo que leia um número inteiro positivo, n , verifique se ele é triangular e escreva “é triangular” em caso afirmativo e “não é triangular” caso contrário.
9. Escreva um algoritmo que leia três números inteiros positivos, n , a e b , e escreva, em ordem crescente, os n primeiros inteiros positivos que são múltiplos de a ou b ou ambos. Por exemplo, se $n = 6$, $a = 2$ e $b = 3$, o algoritmo deve escrever como saída os números 2, 3, 4, 6, 8 e 9.
10. Uma pessoa aplicou seu capital, C , a juros e deseja saber, trimestralmente, a posição de seu investimento inicial. Chamando i a taxa de juros do trimestre e n o número de trimestres do investimento, sabe-se que o valor atual, M_n , do investimento após n trimestres é dado pela fórmula

$$M_n = C \cdot (1 + i)^n.$$

Escreva um algoritmo que receba como entrada o capital inicial, C , a taxa de juros, i , e o número, X , de *anos* completos em que o capital foi investido, e produza como saída o valor M_n , onde n é o número de trimestres dos X anos. Não utilize o operador de potenciação.

```

1 algoritmo "N-esimo termo da sequencia de Fibonacci"
2 var
3     a, b, i, t, n : inteiro
4 inicio
5     escreva ("Informe um valor inteiro não-negativo: " )
6     leia (n)
7     a <- 0
8     b <- 1
9     i <- 1
10    enquanto i <= n faça
11        t <- a + b
12        a <- b
13        b <- t
14        i <- i + 1
15    fimenquanto
16    escreva( "O ", n , "-ésimo termo da sequência de Fibonacci é: ", a)
17 fimalgoritmo

```

Algoritmo 11.1: Cálculo n -ésimo termo da sequência de Fibonacci

```

1 algoritmo "Numero da direita para a esquerda"
2 var
3     n, m, q, r : inteiro
4 inicio
5     escreva( "Entre com um inteiro nao-negativo: " )
6     leia( n )
7     m <- n % 10
8     q <- n \ 10
9     enquanto q <> 0 faça
10        r <- q % 10
11        m <- m * 10 + r
12        q <- q \ 10
13    fimenquanto
14    escreva( "O numero ", n , " da direita para a esquerda e: ", m )
15 fimalgoritmo

```

Algoritmo 11.2: Inverter a ordem dos dígitos de um número

```

1 algoritmo "Primo ou composto"
2 var
3   n, i: inteiro
4 inicio
5   escreva( "Entre com um numero inteiro maior que 1: " )
6   leia( n )
7   i <- 2
8   enquanto ( n % i ) <> 0 faca
9     i <- i + 1
10  fimenquanto
11  se i < n entao
12    escreva( "O numero ", n , " e composto")
13  senao
14    escreva( "O numero ", n , " e primo")
15  fimse
16 fimalgoritmo

```

Algoritmo 11.3: Determinar se um número é primo ou composto

ESTRUTURAS DE REPETIÇÃO 3

12.1 O cálculo da média aritmética

Considere o seguinte problema: dados um número inteiro positivo, n , e uma sequência, x_1, x_2, \dots, x_n , com n números reais, calcule e escreva a média aritmética dos n números da sequência. Como sabemos, a média aritmética desses n números pode ser obtida através da fórmula

$$\frac{1}{n} \cdot \sum_{i=1}^n x_i.$$

No entanto, o que torna o problema acima um pouco complicado é o fato de não sabermos o valor de n *antes de escrevermos o algoritmo*. Conseqüentemente, não temos como saber quantas variáveis deveremos declarar no algoritmo para armazenar os valores da sequência, $(x_i)_{i=1}^n$, de entrada. Mas, note que precisamos dos valores da sequência apenas para calcular a soma da fórmula:

$$\sum_{i=1}^n x_i.$$

Como a soma acima pode ser calculada de forma iterativa através de um laço, de uma variável acumuladora e à medida que os valores da sequência forem lidos, não precisamos definir uma variável para armazenar cada valor. De fato, o seguinte trecho de algoritmo ilustra o cálculo da soma:

```
soma ← 0
i ← 1
enquanto i ≤ n faça
    leia( x )
    soma ← soma + x
    i ← i + 1
fimenquanto
```

No trecho acima, o corpo do laço é executado n vezes e, para cada execução, um valor da entrada é lido e armazenado na variável x . Obviamente, quando um valor é lido da entrada e armazenado em x , o valor que estava em x é “perdido”, pois x só pode armazenar um valor por vez. Mas, isso pouco importa, pois queremos apenas calcular a soma dos n números da sequência.

O algoritmo completo é dado em [12.1](#).

```

1 algoritmo "Media aritmetica de n numeros reais"
2 var
3     i, n : inteiro
4     soma, x, media : real
5 inicio
6     escreva( "Entre com a quantidade de numeros: " )
7     leia( n )
8     soma <- 0
9     i <- 1
10    enquanto i <= n faca
11        escreva( "Entre com o ", i , "-esimo numero: " )
12        leia (x)
13        soma <- soma + x
14        i <- i + 1
15    fimenquanto
16    media <- soma / n
17    escreva( "A media aritmetica dos ", n , " numeros e ", media )
18 fimalgoritmo

```

Algoritmo 12.1: Cálculo da média aritmética de n números reais

12.2 O maior elemento de uma sequência

Suponha que desejemos escrever um algoritmo para ler um número inteiro positivo, n , seguido por uma sequência de n números reais, e escrever o maior de todos os números da sequência. Mais uma vez, temos uma situação em que a entrada do problema possui um tamanho variável, pois não sabemos quantos números podem fazer parte dela *antes de escrever o algoritmo*.

Em princípio, podemos fazer a leitura da entrada da mesma forma que fizemos para o problema da Seção 12.1. Mas, e quanto à estratégia de solução do problema? A solução do problema também pode ser encontrada à medida que a entrada está sendo lida. De fato, podemos definir uma variável, digamos *maior*, para guardar o maior valor de todos os números lidos da entrada. Inicialmente, esta variável recebe o valor do primeiro número lido. Em seguida, para cada número, x , lido, comparamos o valor de x com o valor de *maior*. Se aquele for maior do que este, atribuímos o valor de x a *maior*. Caso contrário, o valor de *maior* permanece o mesmo. Isto é,

```

leia(maior)
i <- 2
enquanto i <= n faca
    leia(x)
    se x > maior entao
        maior <- x
    fimse
    i <- i + 1
finenquanto

```

Note que o primeiro número é lido antes do laço ser encontrado. Como o valor de n é supostamente positivo, podemos assumir que há pelo menos um valor a ser lido. Este valor é lido antes do laço ser encontrado para que a variável *maior* seja inicializada com o primeiro valor lido. Em

seguida, no corpo do laço, os demais valores são lidos. O corpo do laço se repete $n - 1$ vezes, que é exatamente o número de valores restantes a serem lidos. Cada valor lido no corpo do laço é comparado com o maior valor lido antes dele, que está armazenado na variável *maior*. Se o valor em *maior* for menor do que o valor lido, *maior* recebe o valor lido. Logo, após a execução do laço, *maior* conterá o maior valor lido e tudo que precisamos fazer é escrever este valor.

O algoritmo completo é dado em 12.2.

```

1 algoritmo "Maior numero de uma sequencia"
2 var
3     i, n : inteiro
4     maior, x : real
5 inicio
6     escreva( "Entre com a quantidade de numeros da sequencia: " )
7     leia( n )
8     escreva( "Entre com o primeiro numero da sequencia: " )
9     leia( maior )
10    i <- 2
11    enquanto i <= n faca
12        escreva( "Entre com o proximo numero da sequencia: " )
13        leia( x )
14        se x > maior entao
15            maior <- x
16        fimse
17        i <- i + 1
18    fimenquanto
19    escreva( "O maior numero da sequencia e: ", maior )
20 fimalgoritmo

```

Algoritmo 12.2: Cálculo do maior de uma sequência de n números reais

12.3 Os múltiplos de posição na sequência

Suponha que desejemos escrever um algoritmo para ler uma sequência de números inteiros seguida pelo número zero e escrever os números da sequência que forem múltiplos de suas respectivas posições, exceto o zero. Por exemplo, se a sequência for 3, 7, 8, 16 e 0, a saída deve ser 3 e 16, pois 3 é múltiplo de 1 (a posição do 3 na sequência), 7 não é múltiplo de 2 (a posição do 7 na sequência), 8 não é múltiplo de 3 (a posição do 8 na sequência) e 16 é múltiplo de 4 (a posição do 16 na sequência). Novamente, temos uma situação em que o tamanho da entrada (a quantidade de números da sequência) não é conhecido no momento em que escrevemos o algoritmo. Então, não podemos declarar uma variável para cada elemento da sequência. Além disso, o tamanho da entrada também não é informado na própria entrada, como nos exemplos anteriores.

A própria entrada possui uma propriedade que nos permite saber qual é o último número da sequência: o número que o sucede é igual a zero. Logo, a entrada pode ser lida com o seguinte laço:

```

leia (x)
enquanto x <> 0 faca

```

```

    leia (x)
fimenquanto

```

O trecho algorítmico acima lerá números até que um zero seja lido. Agora, note que o problema nos pede para escrever os números da entrada que são múltiplos de suas respectivas posições. Esta operação de escrita pode ser feita à medida que os números sejam lidos, como segue:

```

leia (x)
enquanto  $x \neq 0$  faca
    se  $x$  é múltiplo de sua posição na sequência entao
        escreva (x)
    fimse
leia (x)
fimenquanto

```

Mas, como podemos determinar se x é múltiplo de sua posição na sequência se não temos essas posições? Mais uma vez, uma observação mais cuidadosa da entrada nos conduz à solução: os números são lidos na ordem em que eles ocorrem na sequência. Esta observação nos diz que podemos definir uma variável para contar quantos números foram lidos até o momento. Com essa variável, podemos verificar se x é múltiplo de sua posição comparando o resto da divisão de x pelo valor da variável com zero. O trecho de algoritmo a seguir utiliza essa estratégia:

```

leia (x)
i <- 1
enquanto  $x \neq 0$  faca
    se  $(x \% i) = 0$  entao
        escreva (x)
    fimse
leia (x)
    i <- i + 1
fimenquanto

```

O algoritmo completo é mostrado em 3.

12.4 Exercícios Propostos

1. Escreva um algoritmo que leia um número inteiro positivo, n , e uma sequência, x_1, \dots, x_n , de n números inteiros, calcule e escreva o triplo, $3 \cdot x_1, \dots, 3 \cdot x_n$, de cada um dos n números da sequência.
2. Escreva um algoritmo que leia um número inteiro positivo, n , e uma sequência, x_1, \dots, x_n , de n números inteiros, calcule e escreva o menor dentre todos os n números, x_1, \dots, x_n , da sequência.
3. Escreva um algoritmo que leia cem números inteiros e escreva a quantidade de números de entrada que são maiores do que 30 e menores do que 50.
4. Escreva um algoritmo que leia vinte números inteiros, calcule e escreva a *soma* dos quadrados menores ou iguais a 225 dos vinte números dados como entrada.

5. Escreva um algoritmo que leia duzentos números inteiros e escreva a quantidade de números de entrada que são pares e a quantidade de números de entrada que são ímpares.
6. Escreva um algoritmo que leia a idade e o peso de 20 pessoas, calcule e escreva a média dos pesos das pessoas da mesma faixa etária. Os dados de entrada estão dispostos na forma $idade_1, peso_1, idade_2, peso_2, \dots, idade_{20}, peso_{20}$, onde $idade_i$ e $peso_i$ são a idade e o peso da i -ésima pessoa, para $i = 1, \dots, 20$. A idade é um número inteiro positivo e o peso é um número real. As faixas etárias são de 1 a 10 anos, 11 a 20 anos, 21 a 30 anos e maiores de 30 anos.
7. No dia da estreia do filme “Senhor dos Anéis”, uma grande emissora de TV realizou uma pesquisa logo após o encerramento do filme. Cada espectador respondeu a um questionário no qual constava sua idade e a sua opinião em relação ao filme: excelente – 3, bom – 2, regular – 1. Escreva um algoritmo que leia a idade e a opinião de 20 espectadores, calcule e escreva
 - a média das idades das pessoas que responderam excelente;
 - a quantidade de pessoas que responderam regular;
 - a percentagem de pessoas que responderam bom entre todos os espectadores entrevistados.

Os dados de entrada estão dispostos na forma

$$idade_1, \quad opinião_1, \quad \dots, \quad idade_{20}, \quad opinião_{20},$$

onde $idade_i$ e $opinião_i$ são a idade e a opinião da i -ésima pessoa, para $i = 1, \dots, 20$. A idade é um número inteiro positivo, enquanto a opinião é um dos números inteiros: 1, 2 e 3.

8. Escreva um algoritmo que leia uma sequência de números reais terminada pelo número zero e calcule e escreva a quantidade de números lidos que estão no intervalo $[100, 200]$.
9. Escreva um algoritmo que leia o sexo de uma certa quantidade de pessoas e escreva a quantidade de pessoas que são do sexo masculino e a quantidade de pessoas que são do sexo feminino. A entrada é dada como uma sequência de caracteres formada *apenas* pelas letras “F”, “M”, “f” ou “m” e seguida da letra “X”. As letras “F” e “f” representam pessoas do sexo feminino, enquanto as letras “M” e “m” representam pessoas do sexo masculino.
10. Uma empresa de fornecimento de energia elétrica faz a leitura mensal dos medidores de consumo. Para cada consumidor são fornecidos os seguintes dados:
 - número do consumidor;
 - quantidade de kWh (quilowatts por hora) consumida durante o mês;
 - tipo do consumidor.

O número do consumidor é um número inteiro positivo que identifica unicamente o consumidor. A quantidade de kWh consumida durante o mês é um número real não-negativo e o tipo do consumidor é um dos números 1, 2 e 3, onde 1 significa consumidor residencial, 2 significa consumidor comercial e 3 significa consumidor industrial. Os valores em R\$ pagos por 1 kWh são R\$ 0,30, R\$ 0,50 e R\$ 0,70 para os consumidores dos tipos 1, 2 e 3, respectivamente. Escreva um algoritmo que leia os dados da leitura mensal dos medidores de consumo, calcule e escreva

- o custo total do consumo de cada consumidor;
- o total de consumo de energia de cada tipo de consumidor;
- a média de consumo dos consumidores dos tipos 1 e 2.

Os dados devem ser lidos como uma sequência de triplas da forma *número do consumidor*, *quantidade de kWh consumida* e *tipo do consumidor*. A sequência de entrada deve terminar com um número zero.

11. Uma empresa realizou uma pesquisa com 1000 habitantes de uma região para coletar sexo, idade e altura deles. A empresa deseja calcular as seguintes informações:
- a média de idade dos habitantes da região;
 - a média de altura das mulheres da região com mais de 21 anos;
 - a maior altura entre os homens e
 - o percentual de habitantes com idade entre 18 e 30 anos.

Então, escreva um algoritmo para ler os dados de entrada da pesquisa e calcular e escrever as informações que a empresa deseja encontrar. A entrada se dará como uma sequência de triplas, *sexo*, *idade* e *altura*, onde *sexo* é um dos caracteres 'F', 'f', 'M' e 'm', *idade* é um inteiro positivo e *altura* é um número real positivo.

12. A Prefeitura de Natal fez uma pesquisa entre os habitantes assalariados de Natal para coletar dados sobre o salário e número de filhos deles. A Prefeitura deseja saber a média salarial dos assalariados, o número médio de filhos, o maior salário e o percentual de assalariados com salário até R\$ 800,00. Escreva um algoritmo para resolver o problema da Prefeitura. O algoritmo deve ler uma sequência de pares, *salário* e *número de filhos*, onde *salário* é um real positivo e *número de filhos* é um inteiro positivo. Esta sequência é seguida pelo número zero para indicar o seu término. A saída do algoritmo consiste da média salarial dos assalariados, número médio de filhos, maior salário e percentual de assalariados com salário até R\$ 800,00.
13. Uma das maneiras de se conseguir calcular a raiz quadrada de um número inteiro positivo é através da subtração, do número, de ímpares consecutivos a partir de 1 até se atingir o número zero. O número de subtrações realizadas é igual a raiz do número. Por exemplo, se número for 16, então temos

$$16 - 1 = 15$$

$$15 - 3 = 12$$

$$12 - 5 = 7$$

$$7 - 7 = 0$$

Logo, realizamos 4 subtrações, o que está de acordo com a raiz de 16. Se, por acaso, o resultado de uma subtração for negativo, o número não possui raiz quadrada exata e o processo de cálculo deve ser abortado. Por exemplo, se o número for 14, então temos

$$14 - 1 = 13$$

$$13 - 3 = 10$$

$$10 - 5 = 5$$

$$5 - 7 = -2$$

Escreva um algoritmo que leia um número inteiro positivo, n , e aplique o processo acima para determinar se a raiz do número. Se o número admitir uma raiz inteira, o algoritmo deve escrever esta raiz. Caso contrário, o algoritmo deve escrever a mensagem “O número não possui raiz inteira”.

```

1 algoritmo "Múltiplos de posição de uma sequência"
2 var
3     i, x : inteiro
4 inicio
5     escreva( "Entre com o primeiro número da sequência ou zero: " )
6     leia( x )
7     i <- 1
8     enquanto x <> 0 faça
9         se ( x % i ) = 0 então
10            escreva( x , " " )
11        fimse
12        escreva( "Entre com o próximo número da sequência ou zero: " )
13        leia( x )
14        i <- i + 1
15    fimenquanto
16 fimalgoritmo

```

Algoritmo 12.3: Escrever os números múltiplos de uma sequência de inteiros

ESTRUTURAS DE REPETIÇÃO 4

13.1 O laço repita

A linguagem Portugol da ferramenta VISUALG dispõe de outro comando de repetição que executa uma sequência de comandos repetidamente até que uma dada condição se torne verdadeira:

```

repita
  comando1
  comando2
  :
  comandon
ate expressão lógica

```

O comando repita-ate é semelhante ao comando enquanto-faca-fimenquanto, pois ele também repete uma sequência de comandos com base no valor de uma expressão lógica. No entanto, há uma diferença sutil entre eles: o comando repita-ate *executa a sequência de comandos pelo menos uma vez* e, só depois da primeira execução, é que a expressão lógica é avaliada. Se ela avaliar para falso, a sequência de comandos no corpo do laço é repetida. Caso contrário, o laço é finalizado.

O uso mais corriqueiro do comando repita-ate é na consistência da leitura de um valor de entrada. O que queremos dizer com isso? Vimos vários exemplos de algoritmos em que a entrada é restrita a um subconjunto dos valores possíveis de serem assumidos por uma variável. Por exemplo, algoritmos em que a entrada deve ser um número inteiro *positivo* e que usamos uma variável inteira para armazenar o valor lido. Tal variável pode armazenar valores não-positivos, tais como 0 e -13 . Para esses algoritmos, assumimos que o “usuário” iria sempre entrar um valor que respeitasse a restrição, mas o fato é que não podemos garantir que isso vá acontecer. Se envolvermos o comando de leitura de n por um comando repita-ate, garantiremos que o restante do algoritmo só será executado quando um valor positivo for atribuído a n (veja o Algoritmo 13.1).

13.2 Exemplo

Há outras situações em que o comando repita-ate é mais naturalmente utilizado do que o comando enquanto-faca. Como exemplo, considere o problema de escrever um algoritmo para ler um número inteiro positivo, n , e um número real, x , e calcular e escrever o resultado da série

$$x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots + \frac{x^n}{n}.$$

```

1 algoritmo "Inteiros de 1 a n"
2 var
3   num, i : inteiro
4 inicio
5   repita
6     escreva( "Entre com um numero inteiro positivo: " )
7     leia( num )
8   ate num > 0
9   i <- 1
10  enquanto ( i <= num ) faca
11    escreva( i , " " )
12    i <- i + 1
13  fimenquanto
14 fimalgoritmo

```

Algoritmo 13.1: Algoritmo para escrever os inteiros de 1 a n .

A série possui n termos. Note que, para todo $i = 1, \dots, n$, o i -ésimo termo da série é da forma

$$\frac{x^i}{i}.$$

Logo, podemos escrever um algoritmo para calcular a série com base na mesma estratégia usada para soma os n primeiros números inteiros. A única diferença é que acumularemos os termos da série ao invés dos índices deles. Isto é, podemos utilizar um laço como

```

soma <- 0
i <- 1
enquanto i <= n faca
  soma <- soma + i-ésimo termo
  i <- i + 1
fimenquanto

```

O problema do laço acima é o cálculo do i -ésimo termo. Embora saibamos qual é o termo, ele possui uma potência e esta operação deve ser implementada com o uso de outro laço (já que não estudamos o operador de potenciação ainda). No entanto, uma observação cuidadosa dos termos da série nos leva a uma solução mais simples. Em particular, note que, para todo $i = 1, \dots, n-1$, o $(i+1)$ -ésimo termo da série, $x^{(i+1)}/(i+1)$, pode ser obtido a partir do i -ésimo termo, x^i/i , multiplicando o numerador por x e incrementando o denominador em uma unidade. Esta observação sugere que a soma pode ser realizada através do seguinte laço:

```

soma <- 0
i <- 1
num <- x
den <- 1
enquanto i <= n faca
  soma <- soma + num / den
  num <- num * x
  den <- den + 1
  i <- i + 1
fimenquanto

```

Há duas outras observações importantes sobre o laço acima. A primeira delas diz respeito à redundância da variável *den*; isto é, esta variável possui o mesmo valor que *i* quando realizamos a acumulação dos termos da soma. Logo, podemos substituir a linha *soma <- soma + num / den* pela linha *soma <- soma + num / i* e remover a variável *den* e as demais linhas que envolvem esta variável. A segunda observação importante é que o corpo do laço *enquanto* será executado pelo menos uma vez, pois $n \geq 1$ por hipótese. Isto sugere a utilização do laço *repita*, como abaixo:

```

soma <- 0
i <- 1
num <- x
repita
  soma <- soma + num / i
  num <- num * x
  i <- i + 1
ate i > n

```

Embora ambos os laços possam ser utilizados no trecho de algoritmo acima, note que a utilização do laço *repita* é bem mais “natural”, pois o corpo do laço é executado pelo menos uma vez sem a necessidade de realizar um teste que sempre resultará no valor verdadeiro. Uma solução completa para o problema que acabamos de discutir acima está ilustrado em Algoritmo 13.2.

13.3 Laço repita versus laço enquanto

Os comandos repita-ate e enquanto-faca-fimenquanto são, de fato, equivalentes, pois tudo o que um deles faz, o outro também pode fazer. Mais especificamente, o laço *repita* com a forma

```

repita
  comando1
  comando2
  ⋮
  comandon
ate expressão lógica

```

pode ser transformado no laço *enquanto*

```

comando1
comando2
⋮
comandon
enquanto nao expressão lógica faca
  comando1
  comando2
  ⋮
  comandon
fimenquanto

```

Em outras palavras, escrevemos os comandos no corpo do laço repita antes do laço enquanto para garantir que eles sejam executados pelo menos uma vez. Também negamos a expressão lógica, pois os comandos devem ser repetidos enquanto a expressão original avaliar para falso.

Por outro lado, um laço enquanto da forma

```

enquanto expressão lógica faca
  comando1
  comando2
  ⋮
  comandon
fimenquanto

```

pode ser transformado em um laço repita da forma

```

se expressão lógica entao
  repita
    comando1
    comando2
    ⋮
    comandon
  ate nao expressão lógica
fimse

```

Em outras palavras, escrevemos o laço repita dentro de uma estrutura condicional para garantir que o corpo do laço seja executado uma vez apenas se a expressão lógica for verdadeira. Também negamos a expressão lógica para garantir que o corpo do laço enquanto seja repetido sempre que a expressão avaliar para o valor verdadeiro, que é exatamente o que ocorre com o laço enquanto.

Por exemplo, em uma aula anterior usamos o laço enquanto em um algoritmo para calcular a soma de todos os números de um intervalo fechado, $[a, b]$, onde a e b são números inteiros, com $a < b$, dados como entrada para o algoritmo. Um algoritmo equivalente escrito com o laço repita é dado em 13.3. Note que simplesmente aplicamos a transformação discutida acima.

13.4 Exercícios propostos

1. Resolva todos os exercícios da aula anterior usando o laço repita. Tente desenvolver os algoritmos sem se preocupar em usar a transformação vista nesta aula; isto é, utilize o laço repita mais “naturalmente”.
2. Numa fábrica trabalham homens e mulheres divididos em três classes:
 - A Os que fazem até 30 peças por mês;
 - B Os que fazem de 31 a 35 peças por mês;
 - C Os que fazem mais de 35 peças por mês.

Os trabalhadores da classe A recebem salário mínimo. Os trabalhadores da classe B recebem salário mínimo e mais 3% do salário mínimo por peça, acima das 30 iniciais. Os

trabalhadores da classe C recebem salário mínimo e mais 5% do salário mínimo por peça, acima das 30 iniciais. Escreva um algoritmo que leia o valor do salário mínimo e uma sequência com o seguinte “trio” de dados: o número do operário (um inteiro positivo e único para cada operário), o número de peças fabricadas por mês (um inteiro não-negativo) e o sexo do operário (a letra "F" ou "M"). A sequência de entrada é seguida pelo número 0. Em seguida, o algoritmo deve calcular e escrever o salário total de cada operário, o total da folha mensal de pagamento da fábrica, o número total de peças fabricadas por mês, a média de peças fabricadas pelos homens de cada classe, a média de peças fabricadas pelas mulheres de cada classe e o número do operário (ou operária) de maior salário (se houver empate, deve ser escrito o menor número).

3. Escreva um algoritmo que leia um inteiro positivo, n , e um valor real, x , e calcule e escreva o somatório

$$\frac{x}{n} - \frac{x^2}{n-1} + \frac{x^3}{n-2} - \dots + (-1)^{n+1} \cdot \frac{x^n}{1}.$$

4. Escreva um algoritmo que calcule e escreva a soma dos 50 primeiros termos da série

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

5. Considere a equação

$$x^3 - 3x^2 + 1 = 0.$$

Qualquer raiz real da equação acima pode ser encontrada através de aproximações sucessivas obtidas com a utilização da fórmula

$$x_{n+1} = x_n - \frac{x_n^3 - 3x_n^2 + 1}{2x_n^2 - 6x_n}.$$

Escreva um algoritmo que receba como entrada uma estimativa inicial, x_1 , para uma raiz e calcule e escreva a trigésima aproximação. Usando a ferramenta VISUALG, execute o seu algoritmo usando como entrada $x_1 = 1.5$.

6. O número 3025 goza da seguinte propriedade

$$\begin{cases} 30 + 25 = 55 \\ 55^2 = 3025 \end{cases}$$

Escreva um algoritmo determine e escreva todos os números de quatro dígitos que possuem a propriedade acima. Note que este algoritmo não possui nenhum dado de entrada.

7. Numa universidade, cada aluno possui os seguintes dados:

- Renda pessoal
- Renda familiar
- Total gasto com alimentação
- Total gasto com outras despesas

Escreva um algoritmo que calcule e escreva a porcentagem dos alunos que gasta acima de R\$ 200,00 com outras despesas, o número de alunos com renda pessoal maior que renda familiar e a porcentagem gasta com alimentação e outras despesas em relação às rendas pessoal e familiar.

A entrada do algoritmo é uma sequência com quatro números reais positivos (renda pessoal, renda familiar, total gasto com alimentação e total gasto com outras despesas) para cada aluno, seguida pelo número zero.

8. Um número inteiro positivo, n , é dito *triangular* se, e somente se, ele é o resultado do produto de três números inteiros positivos e consecutivos. Por exemplo, 24 é triangular, pois $24 = 2 \times 3 \times 4$. Agora, escreva um algoritmo que leia um número inteiro positivo, n , e escreva como saída “é triangular” se n for triangular e “não é triangular” caso contrário.
9. Escreva um algoritmo para ler um número inteiro positivo, n , e escrever os dígitos de n , da esquerda para a direita, separados por um espaço. Por exemplo, se $n = 2439$, então a saída do algoritmo deveria ser 2 4 3 9.
10. Escreva um algoritmo que imprima a tabela de equivalência de graus Fahrenheit para Celsius (centígrados). Os limites são de 50 a 70 graus Fahrenheit com intervalo de 1 grau. A fórmula para conversão de Fahrenheit (F) para Celsius (C) é

$$C = \frac{F - 32}{1,8} .$$

```

1 algoritmo "Soma de termos de serie finita"
2 var
3     n, i : inteiro
4     soma, num, x: real
5 inicio
6     repita
7         escreva( "Entre com um numero inteiro positivo: " )
8         leia( n )
9     ate n > 0
10    escreva( "Entre com o primeiro termo da serie: " )
11    leia( x )
12    soma <- 0
13    i <- 1
14    num <- x
15    repita
16        soma <- soma + num / i
17        num <- num * x
18        i <- i + 1
19    ate i > n
20    escreva( "A soma de x^i / i para i de 1 a ", n , " é igual a " , soma )
21 fimalgoritmo

```

Algoritmo 13.2: Algoritmo para calcular e escrever o resultado da série finita $\sum_{i=1}^n x^i/i$.

```

1 algoritmo "Somar inteiros de um intervalo"
2 var
3     a, b, i, soma : inteiro
4 inicio
5     escreva( "Entre com o menor inteiro do intervalo: " )
6     leia( a )
7     escreva( "Entre com o maior inteiro do intervalo: " )
8     leia( b )
9     soma <- 0
10    i <- a
11    se i <= b entao
12        repita
13            soma <- soma + i
14            i <- i + 1
15        ate i > b
16    fimse
17    escreva( "A soma dos numeros do intervalo e: ", soma )
18 fimalgoritmo

```

Algoritmo 13.3: Algoritmo para somar os inteiros de um dado intervalo.

VETORES

14.1 Motivação

Considere o problema de calcular a média aritmética das notas de 5 alunos de uma disciplina e determinar e escrever o número de alunos que obtiveram nota superior à média calculada. Como sabemos, o cálculo da média aritmética das notas de 5 alunos de uma disciplina pode ser feito com o uso de um laço enquanto como o que mostramos no trecho de código abaixo:

```
soma ← 0
i ← 1
enquanto i ≤ 5 faça
    leia( nota )
    soma ← soma + nota
    i ← i + 1
fimenquanto
media ← soma / 5
```

Mas, se seguirmos com este trecho de algoritmo, como determinaremos o número de alunos com nota superior à média calculada? Isto porque não guardamos as notas de cada um dos 5 alunos em variáveis, o que nos impede de comparar o valor da média com o das notas lidas depois que o trecho acima for executado. Logo, devemos optar por outro caminho. Um desses caminhos está ilustrado no Algoritmo 14.1, que utiliza cinco variáveis para guardar as notas lidas da entrada.

O Algoritmo 14.1 não utiliza uma estrutura de repetição para ler as notas. Ao invés disso, ele utiliza cinco instruções de leitura. Para determinar quantas notas são superiores à média, o algoritmo compara cada nota lida com a média. Se a nota é superior à média, o algoritmo incrementa um contador, que é inicializado com o valor zero. Note que o trecho do algoritmo que compara e, se necessário, incrementa o contador é o mesmo para cada uma das notas (o que muda é o nome da variável comparada e incrementada) e consiste de uma estrutura condicional.

Se o código é o “mesmo”, por que não podemos utilizar uma estrutura de repetição? O problema aqui é que não temos como “trocar” o nome de uma variável a cada iteração de um laço. No problema que temos em mãos, há apenas 5 variáveis e a redundância que temos não chega a ser um “fardo”. No entanto, se tivéssemos 100, 1000, ou mesmo 1000000 de notas, esta solução seria inviável, uma vez que teríamos de escrever, respectivamente, 100, 1000 ou 1000000 estruturas condicionais semelhantes, uma para cada nota. Felizmente, a linguagem Portugol possui uma forma eficaz de solução que utiliza uma estrutura de dados denominada *vetor*.

14.2 Definição e manipulação de variáveis

A estrutura de dados **vetor** é uma estrutura de dados linear utilizada para armazenar uma seqüência de valores do *mesmo tipo*. Um dado vetor é definido como tendo algum número *fixo* de *células* idênticas. Cada célula armazena um, e somente um, dos valores de dados do vetor. Cada uma das células de um vetor possui um *índice*, através do qual podemos referenciá-la de forma única.

Ao definirmos uma variável do tipo vetor, estamos, na verdade, especificando uma variável e um novo tipo de dados, que é o tipo vetor da variável. Isto porque o tipo da variável não está “pronto para uso”, como é o caso dos tipos inteiro, real, caractere e logico. Para sermos mais específicos, vamos supor que queremos definir uma variável, denominada *notas*, como um vetor de 5 células do tipo inteiro. Na linguagem Portugol, esta definição segue a seguinte sintaxe:

nome : vetor [*tamanho*] de *tipo*

onde *nome* é o nome da variável do tipo vetor, *tamanho* é uma faixa de valores, que consiste do primeiro e do último índice, e *tipo* é o tipo dos valores das células do vetor. Então, definimos *notas* como

notas : vetor [1..5] de real

A declaração acima corresponde à declaração de cinco variáveis do tipo real. Essas cinco variáveis são as cinco células do vetor. Nós podemos manipular cada uma das células individualmente, usando o nome da variável e o índice da célula. Mais especificamente, temos as células

notas[1], *notas*[2], *notas*[3], *notas*[4], *notas*[5],

que correspondem, respectivamente, a primeira, segunda, terceira, quarta e quinta células do vetor *notas*. Cada uma das células acima corresponde a uma variável do tipo real. Tudo que fazemos com uma variável do tipo real pode ser feito com as células de *notas*. Por exemplo, o comando

leia(*notas*[1])

realiza a leitura de um valor do tipo real e faz com que este valor seja o conteúdo da célula *notas*[1]. Já

escreva(*notas*[1])

escreve o conteúdo da célula *notas*[1]. De forma geral, podemos usar *notas*[1] em qualquer instrução que manipule um valor real. O seguinte trecho de algoritmo ilustra diversas manipulações:

```
leia(notas[1])  
i <- 3  
leia(notas[i])  
notas[i - 1] <- ( notas[1] + notas[i] ) / 2
```

Note que, ao escrevermos $notas[i]$, estamos nos referindo à célula de índice i do vetor $notas$, ou seja, o conteúdo de i nos dá o índice da célula. É justamente esta flexibilidade que nos permite manipular vetores de forma bastante compacta. Para você ter uma idéia clara do que estamos falando, considere o trecho de algoritmo a seguir que faz a leitura de valores para as células de $notas$:

```

i ← 1
enquanto i ≤ 5 faca
    escreva( "Entre com a nota ", i , ": " )
    leia( notas[i] )
    i ← i + 1
fimenquanto

```

Podemos, facilmente, modificar o trecho acima para que ele calcule a média das notas também:

```

i ← 1
soma ← 0
enquanto i ≤ 5 faca
    escreva( "Entre com a nota ", i , ": " )
    leia( notas[i] )
    soma ← soma + notas[i]
    i ← i + 1
fimenquanto
media ← soma / 5

```

Finalmente, podemos escrever um trecho de algoritmo bastante compacto para contar quantas das notas lidas são maiores do que a média das notas. Este trecho de algoritmo é mostrado a seguir:

```

i ← 1
maiores ← 0
enquanto i ≤ 5 faca
    se notas[i] > media entao
        maiores ← maiores + 1
    fimse
    i ← i + 1
fimenquanto

```

Note que, ao combinarmos os dois trechos acima, temos um algoritmo muito mais compacto do que o Algoritmo 14.1 para calcular a média aritmética de cinco notas e o número de notas acima da média. Mas, muito mais importante do que isso é que o mesmo algoritmo pode ser modificado, muito facilmente, para lidar com 100, 1000 e 1000000 de notas. De fato, considere o Algoritmo 14.2. Só precisamos trocar o número 5 por 100, 1000 ou 1000000 nas linhas 3, 9, 15 e 18 do algoritmo para obter um novo algoritmo que lida com 100, 1000 ou 1000000 notas, respectivamente.

Variáveis do tipo vetor são comumente chamadas *variáveis compostas homogêneas*. O termo “composta” se deve ao fato da variável ser formada por uma coleção de células. O termo “homogênea” se deve ao fato de todas as células da variável vetor serem de um mesmo tipo de dados. Um outro termo bastante comum, em Computação, para designar vetor é *arranjo unidimensional*.

14.3 O cálculo do desvio padrão

O desvio padrão é uma medida de resumo que nos dá uma idéia de quão disperso estão os valores de um conjunto em relação ao valor esperado dos valores. Se os valores são representados por

$$v_1, \dots, v_n,$$

então o desvio padrão desses valores com relação à média aritmética deles é dado pela fórmula

$$\sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^n (v_i - \bar{v})^2},$$

onde \bar{v} é a média aritmética.

Vamos escrever um algoritmo para calcular o desvio padrão de 10 notas de prova em relação à média aritméticas dessas notas. Cada nota é um número real de 0 a 10. O algoritmo deve ler as dez notas, calcular a média e o desvio padrão e produzir, como saída, o valor da média e o desvio padrão.

Usaremos a fórmula que vimos anteriormente para cálculo do desvio padrão. Esta fórmula depende do cálculo da raiz quadrada de um número real. Para realizar este cálculo, faremos uso do operador de potenciação, \wedge . Este operador calcula o valor de expressões do tipo

$$x^y$$

onde x e y são números reais. Para calcular a raiz quadrada de x usando o operador \wedge , escrevemos

$$x \wedge 0.5$$

Assim como fizemos antes, definiremos um vetor chamado *notas* para armazenar as 10 notas que serão lidas da entrada. A leitura e cálculo da média das 10 notas podem ser feitos como segue:

```

i ← - 1
soma ← - 0
enquanto i ≤ 10 faça
    escreva( "Entre com a nota ", i, ": " )
    leia( notas[i] )
    soma ← soma + notas[i]
    i ← i + 1
fimenquanto
media ← soma / 10

```

Para calcular o desvio padrão, usamos o seguinte laço:

```

i <- 1
desvio <- 0
enquanto i <= 10 faca
    desvio <- desvio + ( notas[i] - media ) * ( notas[i] - media )
    i <- i + 1
fimenquanto
desvio <- ( desvio / 9 )^0.5

```

O algoritmo completo é mostrado em [14.3](#).

14.4 O comando para-faca-fimpara

Como você já deve ter notado, a manipulação de variáveis do tipo vetor é sempre realizada de forma indexada e através de laços. Usamos laços para ler, escrever e fazer cálculos com as variáveis do tipo vetor. Nos exemplos que vimos antes, usamos laços enquanto. De forma geral, o número de iterações do laço é controlado por uma variável que também serve para indexar o vetor. Esta variável é sempre incrementada em uma unidade e comparada com o tamanho do vetor ao final de cada iteração. Como este tipo de laço é tão freqüente quando usamos variáveis do tipo vetor, um laço mais apropriado para o uso com vetores, o **laço para**, foi definido.

O laço para é implementado pelo comando para-faca-fimpara, que tem a sintaxe dada a seguir:

```

para variável de controle de valor inicial ate valor final faca
    comando1
    comando2
    :
    comandon
fimpara

```

A *variável de controle* assume o *valor inicial*, que passa a ser seu valor atual. Em seguida, o valor atual da variável de controle é comparado com o *valor final*. Se o valor atual for menor ou igual ao valor final, o corpo do laço é executado. Caso contrário, o laço termina. Se o corpo do laço é executado, então o valor atual da variável de controle é incrementado em uma unidade depois da execução do último comando do corpo do laço (sem que tenhamos de escrever código para isso) e comparado novamente com o valor final. Se o valor atual for menor ou igual ao valor final, o corpo do laço é executado novamente e assim por diante. Por exemplo, o laço para,

```

para i de 1 ate 10 faca
    escreva( i , " " )
fimpara

```

faz com que os números 1, 2, ..., 10 sejam escritos. Note que a variável de controle, *i*, não é incrementada de forma explícita por nenhum comando de atribuição e soma. O incremento da variável é parte do comando do laço. O Algoritmo [14.4](#) é o Algoritmo [14.3](#) com o uso do laço para.

14.5 Exercícios propostos

1. Escreva um algoritmo que defina um vetor de elementos inteiros de tamanho 100, leia valores de entrada para este vetor e escreva a soma dos elementos que ocupam as posições pares do vetor seguida pelo valor da soma dos elementos que ocupam as suas posições ímpares.
2. Escreva um algoritmo que defina um vetor v de elementos inteiros de tamanho 100, leia valores de entrada para este vetor e escreva a soma, $v_i + v_{101-i}$, de cada par de elementos, v_i e v_{101-i} , que ocupam as posições i e $101 - i$ do vetor, para todo $i \in \mathbb{Z}$ variando de 1 a 50. Isto é, a saída do algoritmo consiste dos valores das somas $v_1 + v_{100}, v_2 + v_{99}, \dots, v_{50} + v_{51}$.
3. Escreva um algoritmo que defina um vetor de elementos inteiros de tamanho 100, leia valores de entrada para este vetor, troque os valores dos elementos v_i e v_{101-i} , para todo $i \in \mathbb{Z}$ variando de 1 a 50, e escreva os elementos do vetor resultante em ordem crescente de posição (isto é, v_1, v_2, \dots, v_{100}).
4. Escreva um algoritmo que defina um vetor de elementos inteiros de tamanho 100, leia um número inteiro, n , com $n > 0$ e $n \leq 10$, e 100 valores de entrada para o vetor, troque os valores dos elementos v_i e v_j , para todo $i \in \mathbb{Z}$ variando de 1 a 100 e $j = ((i+n) \% 100) + 1$, e escreva os elementos do vetor resultante em ordem crescente de posição (isto é, v_1, v_2, \dots, v_{100}).
5. Escreva um algoritmo que leia um número inteiro positivo, n , com $n \leq 100$, e uma seqüência de n números reais e escreva a seqüência de n números em ordem inversa à de leitura.
6. Escreva um algoritmo que leia o preço de compra e o preço e venda de 100 mercadorias e calcule e escreva a quantidade de mercadorias proporcionam (1) um lucro menor que 10%, (2) um lucro maior ou igual a 10% e menor ou igual a 20% e (3) um lucro maior do que 20%.
7. Escreva um algoritmo para calcular o produto escalar de dois vetores. A entrada do algoritmo consiste de um número n , com $n \leq 100$, e de $2 \cdot n$ números reais, x_1, x_2, \dots, x_n e y_1, y_2, \dots, y_n . A saída do algoritmo é o produto escalar, $\langle x, y \rangle$, dos vetores x e y definidos como

$$x = (x_1, x_2, \dots, x_n) \quad \text{e} \quad y = (y_1, y_2, \dots, y_n),$$

isto é,

$$\langle x, y \rangle = \sum_{i=1}^n x_i \cdot y_i.$$

8. Escreva um algoritmo que leia valores para um vetor de 100 inteiros e calcule e escreva o maior e o menor elemento lido, o percentual de números pares, a média dos elementos do vetor e o número de elementos do vetor que são menores do que a média.
9. Tentando descobrir se um dado de seis faces era viciado, um dono de cassino o lançou n vezes, onde $n \leq 100$. Escreva um algoritmo que leia os n resultados dos lançamentos (cada resultado é um número inteiro de 1 a 6), determine e escreva o número de ocorrências de cada face.
10. Um jogador viciado em jogos de cassino deseja fazer um levantamento estatístico simples sobre uma roleta. Para isso, ele fez n , com $n \leq 100$, lançamentos nesta roleta. Sabendo

que uma roleta contém 37 números (de 0 a 36), escreva um algoritmo que leia o resultado dos n lançamentos e calcule e escreva a frequência de cada um dos 37 números da roleta.

```

1 algoritmo "Calcula media de notas e numero de notas superiores a media"
2 var
3     n1, n2, n3, n4, n5, media : real
4     maiores : inteiro
5 inicio
6     escreva( "Entre com a primeira nota: " )
7     leia( n1 )
8     escreva( "Entre com a segunda nota: " )
9     leia( n2 )
10    escreva( "Entre com a terceira nota: " )
11    leia( n3 )
12    escreva( "Entre com a quarta nota: " )
13    leia( n4 )
14    escreva( "Entre com a quinta nota: " )
15    leia( n5 )
16    media <- ( n1 + n2 + n3 + n4 + n5 ) / 5
17    maiores <- 0
18    se n1 > media entao
19        maiores <- maiores + 1
20    fimse
21    se n2 > media entao
22        maiores <- maiores + 1
23    fimse
24    se n3 > media entao
25        maiores <- maiores + 1
26    fimse
27    se n4 > media entao
28        maiores <- maiores + 1
29    fimse
30    se n5 > media entao
31        maiores <- maiores + 1
32    fimse
33    escreva( "A media das notas e: " , media )
34    escreva( "O numero de notas maiores do que a media e: " , maiores )
35 fimalgoritmo

```

Algoritmo 14.1: Algoritmo para calcular a média aritmética de cinco notas.

```

1 algoritmo "Calcula media de notas e numero de notas superiores a media"
2 var
3     n1, n2, n3, n4, n5, media : real
4     maiores : inteiro
5 inicio
6     escreva( "Entre com a primeira nota: " )
7     leia( n1 )
8     escreva( "Entre com a segunda nota: " )
9     leia( n2 )
10    escreva( "Entre com a terceira nota: " )
11    leia( n3 )
12    escreva( "Entre com a quarta nota: " )
13    leia( n4 )
14    escreva( "Entre com a quinta nota: " )
15    leia( n5 )
16    media <- ( n1 + n2 + n3 + n4 + n5 ) / 5
17    maiores <- 0
18    se n1 > media entao
19        maiores <- maiores + 1
20    fimse
21    se n2 > media entao
22        maiores <- maiores + 1
23    fimse
24    se n3 > media entao
25        maiores <- maiores + 1
26    fimse
27    se n4 > media entao
28        maiores <- maiores + 1
29    fimse
30    se n5 > media entao
31        maiores <- maiores + 1
32    fimse
33    escreva( "A media das notas e: " , media )
34    escreva( "O numero de notas maiores do que a media e: " , maiores )
35 fimalgoritmo

```

Algoritmo 14.2: Algoritmo para calcular a média aritmética de cinco notas usando vetor.

```

algoritmo "Calcula desvio padrao"
var notas : vetor [ 1..10 ] de real
soma, media, desvio : real
i : inteiro
inicio
i <- 1
soma <- 0
enquanto i <= 10 faca
    escreva( "Entre com a nota ", i , ": " )
    leia( notas[ i ] )
    soma <- soma + notas[ i ]
    i <- i + 1
fimenquanto
media <- soma / 10
i <- 1
desvio <- 0
enquanto i <= 10 faca
    desvio <- desvio + ( notas[ i ] - media ) * ( notas[ i ] - media )
    i <- i + 1
fimenquanto
desvio <- raizq( desvio / 9 )
escreva( "A media das notas e: " , media )
escreva( "O desvio padrao e: " , desvio )
fimalgoritmo

```

Algoritmo 14.3: Algoritmo para calcular desvio padrão.

```

1 algoritmo "Calcula desvio padrao com laco para"
2 var
3   notas : vetor [ 1..10 ] de real
4   soma, media, desvio : real
5   i : inteiro
6 inicio
7   soma <- 0
8   para i de 1 ate 10 faca
9     escreva( "Entre com a nota ", i , ": " )
10    leia( notas[ i ] )
11    soma <- soma + notas[ i ]
12 fimpara
13 media <- soma / 10
14 desvio <- 0
15 para i de 1 ate 10 faca
16   desvio <- desvio + ( notas[ i ] - media ) * ( notas[ i ] - media )
17 fimpara
18 desvio <- raizq(desvio / 9)
19 escreva( "A media das notas e: " , media )
20 escreva( "O desvio padrao e: " , desvio )
21 fimalgoritmo

```

Algoritmo 14.4: Algoritmo para calcular desvio padrão com laço para.

ANINHAMENTO DE LAÇOS

15.1 Laços aninhados

Em princípio, qualquer comando pode fazer parte do corpo de um laço, inclusive um outro laço. Quando isto acontece, dizemos que os laços estão *aninhados*. Por exemplo, o trecho de algoritmo a seguir escreve os pares ordenados na forma (i, j) , onde $i \in [1, 4] \subset \mathbb{N}$ e $j \in [5, 9] \subset \mathbb{N}$:

```

para  $i$  de 1 ate 4 faca
  para  $j$  de 5 ate 9 faca
    escreva( "(" ,  $i$  , "," ,  $j$  , ")" )
  fimpara
fimpara

```

Para cada iteração do corpo do laço mais externo,

```

para  $i$  de 1 ate 4 faca
  :
fimpara

```

o laço mais interno é executado por completo. Com isso, a saída do trecho algorítmico acima é

```

( 1 , 5 )
( 1 , 6 )
( 1 , 7 )
( 1 , 8 )
( 1 , 9 )

( 2 , 5 )
( 2 , 6 )
( 2 , 7 )
( 2 , 8 )
( 2 , 9 )

( 3 , 5 )
( 3 , 6 )
( 3 , 7 )
( 3 , 8 )
( 3 , 9 )

```

```
( 4 , 5 )
( 4 , 6 )
( 4 , 7 )
( 4 , 8 )
( 4 , 9 )
```

Alguns problemas requerem o aninhamento de dois ou mais laços. Quando estudarmos *matrizes*, teremos oportunidade de resolver problemas com o aninhamento de três ou mais laços. Por enquanto, vamos discutir a solução de um problema que pode ser facilmente descrita com o aninhamento de dois laços. O problema consiste em contar o número de elementos comuns a dois subconjuntos, A e B , de números inteiros. O subconjunto A possui 10 elementos, enquanto o subconjunto B possui 5 elementos. Suponha que A e B não possuam elementos repetidos.

Para solucionar o problema acima, usaremos dois vetores, a e b , para representar os conjuntos A e B , respectivamente. Para contar quantos elementos são comuns a a e b , basta utilizar um contador e *procurar* cada elemento do vetor a no vetor b . Toda vez que um elemento do vetor a estiver no vetor b , incrementamos o contador de elementos. Mas, como “procuramos” um elemento no vetor b ? A ideia é comparar o elemento procurado, digamos $a[i]$, com os elementos do vetor b até que uma igualdade seja verificada ou todos os elementos do vetor b tenham sido comparados. Esta operação de *busca* pode ser efetuada, de forma natural, com o seguinte laço enquanto:

```
achou <- falso
j <- 1
enquanto ( nao achou ) e ( j <= 5 ) faca
  se a[i] = b[j] entao
    achou <- verdadeiro
  senao
    j <- j + 1
fimse
fimenquanto
```

Note que o laço enquanto acima termina por causa de uma de duas condições mutuamente exclusivas: (1) a variável *achou* recebeu o valor verdadeiro ou (2) o valor do contador j é maior do que 5. Quando o laço termina porque a condição (2) é verdadeira, o valor da variável *achou* é falso. Logo, para saber se encontramos o elemento do vetor a que procuramos no vetor b , basta verificarmos o valor que a variável *achou* possui após o término do laço. Se o valor for da variável for verdadeiro, então devemos incrementar o contador de achados para contabilizar o fato do elemento $a[i]$ fazer parte do vetor b . Por exemplo, se *comuns* é a variável contadora, o código

```
se achou entao
  comuns <- comuns + 1
fimse
```

pode ser escrito logo após o laço, incrementando a variável *comuns* se o valor de *achou* for verdadeiro.

O laço enquanto e a estrutura condicional acima nos permitem procurar *um* elemento do vetor a no vetor b . Mas, como fazemos para procurar *todos* os elementos do vetor a da mesma forma que fizemos para apenas um deles? Ora, basta notar que o código acima não depende do elemento do vetor a a ser procurado. O código, na verdade, considera o elemento $a[i]$, mas o índice i é “desconhecido” do código. Este índice pode assumir qualquer valor de 1 a 10. Logo, podemos criar um novo laço que tem como corpo o código acima. Tudo que o novo laço fará é variar o índice i de $a[i]$ de forma que todos os elementos do vetor a sejam procurados no vetor b :

```

comuns ← 0
para i de 1 ate 10 faça
    achou ← falso
    j ← 1
    enquanto ( não achou ) e ( j ≤ 5 ) faça
        se a[i] = b[j] entao
            achou ← verdadeiro
        senão
            j ← j + 1
    fimse
fimenquanto
se achou entao
    comuns ← comuns + 1
fimse
fimpara

```

Note que temos um laço enquanto dentro de um laço para. O papel do laço para é “escolher” um elemento do vetor a por vez para ser procurado no vetor b . O papel do laço enquanto é procurar o elemento escolhido pelo laço para. Se o elemento escolhido for encontrado, então o contador $comuns$ é incrementado. Assim, quando o laço para terminar de executar, o valor do contador $comuns$ será igual ao número de elementos do vetor a que também são elementos do vetor b .

15.2 Ordenação

Em Computação, o termo *ordenação* se refere à tarefa de rearranjar uma sequência de valores para torná-la uma sequência crescente, decrescente, não-crescente ou não-decrescente. Por exemplo,

5 1 10 100 0

é uma sequência com 5 números inteiros. Esta sequência não está ordenada, pois ela nem é crescente, decrescente, não-crescente ou não-decrescente. Entretanto, se permutarmos a posição de alguns elementos, então podemos obter uma sequência crescente com os mesmos 5 inteiros:

0 1 5 10 100

De forma análoga,

100 10 5 1 0

é uma sequência decrescente obtida com outra permutação da posição dos elementos da sequência original. O *problema da ordenação* consiste em encontrar uma permutação da posição dos

elementos da sequência original que gere uma sequência crescente, decrescente, não-crescente ou não-decrescente. Obviamente, para falarmos de sequências crescentes, decrescentes, não-crescentes ou não-decrescentes, assumimos que qualquer valor da sequência original deve ser “comparável” com todos os demais para se determinar se um é maior, menor ou igual ao outro.

```
1 algoritmo "Numero de elementos comuns em dois vetores"
2 var
3   a : vetor[ 1..10 ] de inteiro
4   b : vetor[ 1..5 ] de inteiro
5   i , j , comuns : inteiro
6   achou : logico
7 inicio
8   para i de 1 ate 10 faca
9     escreva( "Entre com o elemento ", i , " do vetor a: " )
10    leia( a[ i ] )
11  fimpara
12  para i de 1 ate 5 faca
13    escreva( "Entre com o elemento ", i , " do vetor b: " )
14    leia( b[ i ] )
15  fimpara
16  comuns <- 0
17  para i de 1 ate 10 faca
18    achou <- falso
19    j <- 1
20    enquanto ( nao achou ) e ( j <= 5 ) faca
21      se a[ i ] = b[ j ] entao
22        achou <- verdadeiro
23      senao
24        j <- j + 1
25      fimse
26    fimenquanto
27    se achou entao
28      comuns <- comuns + 1
29    fimse
30  fimpara
31  escreva( "O numero de elementos comuns e: ", comuns )
32 fimalgoritmo
```

Algoritmo 15.1: Algoritmo para contar número de elementos comuns.

O problema da ordenação é extremamente importante, pois é muito comum precisarmos ordenar um conjunto grande de dados antes de manipulá-lo. Um exemplo disso é um catálogo telefônico. Quando procuramos o telefone de alguém usando um catálogo telefônico, usamos o sobrenome e o nome desse alguém, pois sabemos que os números de telefone estão listados no catálogo em ordem lexicográfica crescente de sobrenome e nome dos assinantes. O fato dos sobrenomes e nomes aparecerem dessa forma faz com que a busca pelo número de um assinante seja feita de forma rápida. Você já imaginou uma busca sem que os sobrenomes e nomes estejam dispostos em ordem lexicográfica crescente? É por isso que *alguém* realizou a tarefa de ordenar os números dos assinantes da forma como conhecemos em um catálogo telefônico.

Veremos, agora, um algoritmo capaz de ordenar um conjunto de valores quaisquer (mas, comparáveis uns com os outros, é claro). Este algoritmo assume que o conjunto de valores a ser ordenado está armazenado em um vetor. O algoritmo pode ordenar os valores em ordem não-decrescente ou não-crescente. Para simplificar nossa exposição, vamos assumir que os valores sejam números inteiros distintos e que devam ser ordenados em ordem crescente. O algoritmo utiliza um método de ordenação bastante conhecido em Computação e denominado *ordenação por seleção*.

Para ilustrar o método de ordenação por seleção, vamos supor que o conjunto de valores a ser ordenado esteja armazenado no vetor a e que consista de: 23, -9, 12, 25, 7, 0, -1, 2, 35 e -4, nesta ordem:

1	2	3	4	5	6	7	8	9	10
23	-9	12	25	7	0	-1	2	35	-4

Se o vetor possui n elementos, o método realiza a ordenação em $n - 1$ passos. No primeiro passo, o menor valor do vetor é encontrado e colocado em sua primeira posição. No segundo passo, o segundo menor valor do vetor é encontrado e colocado em sua segunda posição. De forma geral, no i -ésimo passo, para $1 \leq i \leq n - 1$, o i -ésimo menor elemento do vetor é encontrado e colocado na i -ésima posição do vetor. Após esses $n - 1$ passos, o vetor estará ordenado!

De fato, considere o vetor a , que possui $n = 10$ elementos. No passo $i = 1$ do método de ordenação por seleção, encontramos o menor elemento de a e o colocamos na posição 1 de a . O menor elemento de a é -9, que está na posição 2; isto é, $a[2] = -9$. Queremos colocá-lo na posição 1. Para tal, *trocamos* os elementos $a[1] = 23$ e $a[2] = -9$ de posição, obtendo o seguinte vetor a :

1	2	3	4	5	6	7	8	9	10
-9	23	12	25	7	0	-1	2	35	-4

No passo $i = 2$, encontramos o segundo menor elemento de a , que é o elemento $a[10] = -4$, e o colocamos na posição 2. Para tal, trocamos os elementos $a[2] = 23$ e $a[10] = -4$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	12	25	7	0	-1	2	35	23

No passo $i = 3$, encontramos o terceiro menor elemento de a , que é o elemento $a[7] = -1$, e o colocamos na posição 3. Para tal, trocamos os elementos $a[3] = 12$ e $a[7] = -1$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	25	7	0	12	2	35	23

No passo $i = 4$, encontramos o quarto menor elemento de a , que é o elemento $a[6] = 0$, e o colocamos na posição 4. Para tal, trocamos os elementos $a[4] = 25$ e $a[6] = 0$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	0	7	25	12	2	35	23

No passo $i = 5$, encontramos o quinto menor elemento de a , que é o elemento $a[8] = 2$, e o colocamos na posição 5. Para tal, trocamos os elementos $a[5] = 7$ e $a[8] = 2$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	0	2	25	12	7	35	23

No passo $i = 6$, encontramos o sexto menor elemento de a , que é o elemento $a[8] = 7$, e o colocamos na posição 6. Para tal, trocamos os elementos $a[6] = 25$ e $a[8] = 7$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	0	2	7	12	25	35	23

No passo $i = 7$, encontramos o sétimo menor elemento de a , que é o elemento $a[7] = 12$, e o colocamos na posição 7. Mas, este elemento já está na sétima posição de a . Então, o vetor permanece inalterado neste passo:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	0	2	7	12	25	35	23

No passo $i = 8$, encontramos o oitavo menor elemento de a , que é o elemento $a[10] = 23$, e o colocamos na posição 8. Para tal, trocamos os elementos $a[8] = 25$ e $a[10] = 23$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	0	2	7	12	23	35	25

No passo $i = 9$, encontramos o nono menor elemento de a , que é o elemento $a[10] = 25$, e o colocamos na posição 9. Para tal, trocamos os elementos $a[9] = 35$ e $a[10] = 25$ de posição, obtendo o vetor:

1	2	3	4	5	6	7	8	9	10
-9	-4	-1	0	2	7	12	23	25	35

Como podemos verificar, o vetor a está ordenado em ordem não-decrescente após os 9 passos que executamos do método de ordenação por seleção. Este método sempre funciona? Por quê? O método de ordenação por seleção coloca o i -ésimo menor elemento da sequência de entrada na i -ésima posição do vetor, para todo i variando de 1 a $n - 1$. Mas, isto implica que, ao final de $n - 1$ passos, o maior elemento do vetor estará na posição n , pois os $n - 1$ menores do que ele estão nas posições $1, \dots, n - 1$. Logo, o vetor a estará ordenado após os $n - 1$ passos do método.

Vejamos, agora, o algoritmo que implementa o método de ordenação por seleção acima. Uma operação chave deste algoritmo é aquela que encontra o i -ésimo menor elemento da sequência original de entrada. Para entender esta operação é preciso notar que, ao iniciarmos a busca pelo i -ésimo menor elemento da sequência, o vetor a está parcialmente ordenado, pois

$$a[1], a[2], \dots, a[i - 1]$$

já armazenam os $i - 1$ menores elementos do vetor. Isto significa que o i -ésimo menor está entre os elementos

$$a[i], a[i + 1], \dots, a[n].$$

Logo, a busca pelo i -ésimo menor elemento deve se restringir aos elementos acima. Esta busca pode ser feita por um laço do tipo para como mostrado pelo trecho de algoritmo escrito logo abaixo:

```

menor ← i
para j de i + 1 ate n faca
    se a[j] < a[menor] entao
        menor ← j
    fimse
fimpara
se i <> menor entao
    temp ← a[menor]
    a[menor] ← a[j]
    a[j] ← temp
fimse

```

No trecho de algoritmo acima, a variável *menor* tem como finalidade guardar o índice do menor elemento visto até então entre os elementos $a[i], a[i + 1], \dots, a[n]$. Inicialmente, *menor* recebe o valor de i , que é a posição onde queremos colocar o i -ésimo menor elemento da sequência de entrada. Em seguida, o laço compara o elemento da posição *menor* com os elementos $a[i + 1], \dots, a[n]$. Toda vez que um elemento menor do que $a[menor]$ é encontrado, o valor de *menor* passa a ser a posição deste elemento. Quando o laço termina, *menor* contém a posição do menor elemento entre os elementos $a[i], a[i + 1], \dots, a[n]$. Se esta posição não é i , trocamos os elementos $a[i]$ e $a[menor]$ de posição. Desta forma, o i -ésimo menor elemento sempre acabará na posição i .

O trecho algorítmico que vimos acima encontra o i -ésimo menor elemento da sequência de entrada e o coloca na posição i do vetor a . Para utilizar este trecho de algoritmo para colocar *todos* os elementos da sequência em suas devidas posições, basta executar o trecho para todos os valores de i variando de 1 a $n - 1$. Mas, isto pode ser feito através do uso de um outro laço para, que terá como corpo o trecho de algoritmo acima. Mais especificamente, temos o seguinte:

```

para i de 1 ate n - 1 faca
    menor ← i
    para j de i + 1 ate n faca
        se a[j] < a[menor] entao
            menor ← j
        fimse
    fimpara
    se i <> menor entao
        temp ← a[menor]
        a[menor] ← a[i]
        a[i] ← temp
    fimse
fimpara

```

O algoritmo completo é mostrado em Algoritmo 15.2.

Há um detalhe importante no Algoritmo 15.2. O vetor a é definido como um vetor com 100 elementos, mas o algoritmo solicita como entrada o número de elementos a serem armazenados no vetor, que deve ser um número menor ou igual a 100. Logo, embora o vetor tenha tamanho 100, pode ser que um número bem menor de células seja efetivamente utilizado durante uma execução do algoritmo. Este artifício é bastante comum na prática, pois, em geral, não sabemos *a priori* quantos elementos receberemos como entrada. Então, utilizamos um vetor de tamanho “grande” o suficiente para armazenar qualquer entrada que achamos ser possível de ser dada ao algoritmo.

Na descrição do método de ordenação por seleção, assumimos que os números dados como entrada são distintos. O que acontece se houver números repetidos? O algoritmo ainda funciona corretamente? Execute o algoritmo em uma entrada de tamanho pequeno com alguns elementos repetidos e verifique que ele produz a saída correta. Em seguida, analise o algoritmo e descreva o porquê do algoritmo estar correto para entradas com números repetidos também.

15.3 Exercícios propostos

1. Escreva um algoritmo que leia um número inteiro positivo, n , e escreva a tabuada até 10 dos inteiros de 1 a n .
2. Escreva um algoritmo que leia um número inteiro positivo, n , e uma sequência de n números inteiros, determine e escreva a quantidade de segmentos de números iguais consecutivos que compõem essa sequência. Por exemplo, para $n=9$ e a sequência 5, 2, 2, 4, 4, 4, 4, 1, 1, então a saída deve ser o número 4, pois

5, 2, 2, 4, 4, 4, 4, 1, 1

possui quatro segmentos de números iguais consecutivos:

$\underbrace{5}, \underbrace{2, 2}, \underbrace{4, 4, 4, 4}, \underbrace{1, 1}.$

Observe que há segmentos com um único elemento, tal como $\underbrace{5}$ no exemplo acima.

3. Escreva um algoritmo que leia um inteiro positivo, n , e uma sequência com n números inteiros e calcule e escreva o comprimento do maior segmento crescente da sequência. Por exemplo, se a entrada for

9

5, 10, 3, 2, 4, 7, 9, 8, 5

então a saída é 4, pois

2, 4, 7, 9

é um segmento crescente com comprimento máximo e igual a 4 da sequência

5, 10, 3, $\underbrace{2, 4, 7, 9}$, 8, 5

Se a entrada for

5

10, 8, 7, 5, 2

então a saída é 1, pois todos os segmentos crescentes de tamanho máximo da sequência

10, 8, 7, 5, 2

possuem tamanho 1 e há exatamente cinco segmentos crescentes de tamanho máximo:

$\underbrace{10}, \underbrace{8}, \underbrace{7}, \underbrace{5}, \underbrace{2}$.

4. Sabe-se que um número inteiro positivo da forma n^3 é igual à soma de n números ímpares consecutivos, isto é,

$$1^3 = 1$$

$$2^3 = 3 + 5$$

$$3^3 = 7 + 9 + 11$$

$$4^3 = 13 + 15 + 17 + 19$$

\vdots

Escreva um algoritmo que leia um número inteiro positivo, m , e, em seguida, calcule e escreva todos os ímpares consecutivos cuja soma é igual a n^3 para n variando de 1 a m .

5. Escreva um algoritmo que leia um número inteiro positivo, $n > 1$, e determine e escreva a sua decomposição em fatores primos, exibindo a multiplicidade de cada fator. Por exemplo, se a entrada for $n = 18$, então a saída deve ser da forma

2 com multiplicidade 1

3 com multiplicidade 2

pois $2 \cdot 3 \cdot 3 = 18$ e 2 e 3 são números primos.

6. Escreva um algoritmo que leia dois números inteiros positivos, n e m , e duas sequências ordenadas, em ordem não-decrescente, com n e m números inteiros, respectivamente. Em seguida, o algoritmo deve criar e escrever como saída uma única sequência ordenada em ordem não-decrescente com todos os $m+n$ números das duas sequências de entrada. Assuma que $n, m \leq 100$.
7. Dadas duas sequências com n números inteiros entre 0 e 9, podemos interpretá-las como dois números inteiros de n algarismos para, em seguida, calcular a sequência de números que representa a soma dos dois inteiros. Por exemplo, se $n = 8$ e 8, 2, 4, 3, 4, 2, 5, 1 e 3, 3, 7, 5, 2, 3, 3, 7 forem as duas sequências, então a soma dos dois “números” dados é igual a

1 ^a sequência		8	2	4	3	4	2	5	1
2 ^a sequência	+	3	3	7	5	2	3	3	7
		1	1	6	1	8	6	5	8

Escreva um algoritmo que leia um número inteiro positivo, n , com $n \leq 100$, e duas sequências com n números inteiros entre 0 e 9 e calcule e escreva a sequência que representa o número resultante da soma dos dois “números” dados pelas duas sequências de entrada (como acima).

8. Escreva um algoritmo que leia um número inteiro positivo, n , com $n \leq 100$, e uma sequência,

$$x_1, x_2, \dots, x_n,$$

com n números inteiros, determine um segmento de soma máxima e escreva a soma dos elementos deste segmento. Por exemplo, se n for igual a 12 e a sequência dada como entrada for

$$5, 2, -2, -7, \underline{3, 14, 10, -3, 9}, -6, 4, 1$$

o segmento de soma máxima é 3, 14, 10, -3, 9 e a soma dos inteiros deste segmento é igual a 33. Observe que um segmento é um subsequência de elementos consecutivos da sequência.

9. Escreva um algoritmo que leia o nome dos alunos de uma turma de tamanho indefinido (mas não superior a 60) e sua nota em uma prova (0 a 10: o algoritmo deve verificar se a nota fornecida é válida. O algoritmo para de ler a entrada quando o nome do aluno fornecido for vazio (""). Para cada aluno, o algoritmo deve escrever seu nome e sua nota *normalizada*, dada pela fórmula

$$\frac{n_i}{n_{\max}},$$

onde n_i é a nota que o aluno tirou na prova e n_{\max} é a maior nota obtida dentre todos os alunos da turma.

10. Deseja-se escrever um algoritmo para fazer a emissão da folha de pagamento de uma empresa. Para cada um dos n funcionários, as seguintes informações são fornecidas, por funcionário, em sequência:

Código	Descrição
NOME	Nome do funcionário
SAL	Salário do funcionário
HED	Horas extras diurnas
HEN	Horas extras noturnas
ND	Número de dependentes
FAL	Faltas em horas
DE	Descontos eventuais
REF	Gastos com refeições feitas na empresa
VAL	Vales retirados durante o mês

Para cada funcionário, o algoritmo deve escrever as seguintes informações:

Nome,
 Salário,
 $\text{Horas Extras} = \text{HED} * \text{SAL}/160 + \text{HEN} * 1,2 * \text{SAL}/160,$
 $\text{Salário Família} = \text{ND} * 0,05 * \text{Salário Mínimo vigente},$
 $\text{Salário Bruto} = \text{Salário} + \text{Horas Extras} + \text{Salário Família}.$

e os descontos efetuados:

$\text{INSS} = 0,08 * \text{SAL},$
 $\text{Faltas} = \text{FAL} * \text{SAL}/160,$
 Refeições,
 Vales,
 Descontos Eventuais,
 $\text{Imposto de Renda} = 0,08 * \text{Salário Bruto}.$

e finalmente o seu Salário Líquido:

$$\text{Salário Líquido} = \text{Salário Bruto} - \text{Descontos.}$$

11. Reescreva o Algoritmo 15.2 para que ele ordene números reais.
12. Reescreva o Algoritmo 15.2 para que ele ordene palavras.
13. Use a ferramenta VISUALG para testar os algoritmos que você escreveu para os problemas 13 e 14.

```

1 algoritmo "Ordenacao de inteiros usando o metodo de selecao"
2 var
3   a : vetor[ 1..100 ] de inteiro
4   i , j , n , temp, menor : inteiro
5 inicio
6   escreva( "Entre com o numero de elementos do vetor (<= 100): " )
7   repita
8     leia( n )
9   ate n <= 100
10  para i de 1 ate n faca
11    escreva( "Entre com o elemento ", i , " do vetor a: " )
12    leia( a[ i ] )
13  fimpara
14  para i de 1 ate n - 1 faca
15    menor <- i
16    para j de i + 1 ate n faca
17      se a[ j ] < a[ menor ] entao
18        menor <- j
19      fimse
20    fimpara
21    se i <> menor entao
22      temp <- a[ menor ]
23      a[ menor ] <- a[ i ]
24      a[ i ] <- temp
25    fimse
26  fimpara
27  escreva( "A sequencia ordenada e: " )
28  para i de 1 ate n faca
29    escreva( a[ i ] , " " )
30  fimpara
31 fimalgoritmo

```

Algoritmo 15.2: Ordenação de uma sequência de inteiros usando o método de seleção.

MATRIZES - PARTE 1

16.1 Definição e Manipulação de Matrizes

Sabemos como definir variáveis de um novo tipo de dados, denominado vetor, que representam sequências de valores de um mesmo tipo. Por possuírem uma estrutura sequencial, vetores são denominados arranjos unidimensionais. Nesta aula, generalizaremos a noção de arranjo para duas dimensões. Mais especificamente, definiremos variáveis de um novo tipo de dados, conhecido como *matriz*, que representa tabelas, com m linhas e n colunas, de valores de um mesmo tipo. Cada uma das m linhas de uma matriz pode ser vista como um vetor de tamanho n . É por isso que matrizes são também denominadas de *arranjos bidimensionais*.

Uma **matriz** é uma coleção de valores de um mesmo tipo de dados dispostos na forma de uma tabela com m linhas e n colunas, onde m e n são constantes inteiras positivas. Cada elemento ou *célula* de uma matriz armazena um único valor e todos os valores de uma matriz são de um mesmo tipo. Cada célula de uma matriz pode ser identificada de forma única por dois índices, digamos i e j , com $i, j \in \mathbb{Z}$, tais que $1 \leq i \leq m$ e $1 \leq j \leq n$. O índice i identifica a linha da tabela em que o elemento se encontra, enquanto o índice j identifica a coluna (veja Figura 16.1).

	1	2	3	4	5	6	7	8
1								
2								
3								
4			c					
5								
6								
7								
8								

↘ elemento (4,3)

Figura 16.1: Uma matriz com 8 linhas e 8 colunas.

Cada linha de uma matriz com m linhas e n colunas, ou simplesmente uma *matriz m por n* , pode ser considerada como sendo um vetor contendo n elementos. Na linguagem Portugol da ferramenta VISUALG, uma variável do tipo matriz m por n é declarada através da seguinte sintaxe:

nome : vetor [*tamanho1* , *tamanho2*] de *tipo*

onde *nome* é o nome da variável do tipo matriz, *tamanho1* e *tamanho2* são faixas de valores consistindo do primeiro e último índices das linhas e colunas da matriz, respectivamente, e *tipo* é o tipo dos valores das células do vetor. Se a matriz possui *m* linhas e *n* colunas, então *tamanho1* e *tamanho2* devem ser escritos como $m_1..m_2$ e $n_1..n_2$, respectivamente, onde $m_2 - m_1 = m - 1$ e $n_2 - n_1 = n - 1$. Comumente, temos $m_1 = n_1 = 1$, $m_2 = m$ e $n_2 = n$. No entanto, nada impede que usemos outros valores para m_1 , m_2 , n_1 e n_2 . Por exemplo, $m_1 = -1$, $m_2 = m - 2$, $n_1 = 0$ e $n_2 = n - 1$.

Por exemplo, o comando abaixo declara uma variável matriz 5 por 3 de valores inteiros:

```
tabela : vetor [ 1..5 , 1..3 ] de inteiro
```

A declaração acima corresponde à declaração de $15 = 5 \times 3$ variáveis do tipo inteiro. Essas quinze variáveis são as quinze células da matriz. Nós podemos manipular cada uma das células individualmente, usando o nome da variável e os índices da célula. As células da matriz *tabela* são:

<i>tabela</i> [1, 1]	<i>tabela</i> [1, 2]	<i>tabela</i> [1, 3]
<i>tabela</i> [2, 1]	<i>tabela</i> [2, 2]	<i>tabela</i> [2, 3]
<i>tabela</i> [3, 1]	<i>tabela</i> [3, 2]	<i>tabela</i> [3, 3]
<i>tabela</i> [4, 1]	<i>tabela</i> [4, 2]	<i>tabela</i> [4, 3]
<i>tabela</i> [5, 1]	<i>tabela</i> [5, 2]	<i>tabela</i> [5, 3]

Cada uma das células acima corresponde a uma variável do tipo inteiro. Tudo que fazemos com uma variável do tipo inteiro pode ser feito com as células de *tabela*. Por exemplo, o comando

```
leia(tabela[1, 2])
```

realiza a leitura de um valor do tipo inteiro e faz com que este valor seja o conteúdo da célula *tabela*[1, 2]. Já

```
escreva(tabela[1, 2])
```

escreve o conteúdo da célula *tabela*[1, 2]. De forma geral, podemos usar *tabela*[1, 2] em qualquer instrução que manipule um valor inteiro. O seguinte trecho de algoritmo ilustra diversas manipulações:

```
i <- 2
j <- 3
leia(tabela[i, j])
tabela[i, j] <- tabela[i, j - 1] + tabela[i - 1, j]
```

Note que, ao escrevermos *tabela*[*i*, *j*], estamos nos referindo à célula da linha *i* e coluna *j* da matriz *tabela*, ou seja, o conteúdo de *i* nos dá a linha e o conteúdo de *j* nos dá a coluna da célula. Esta flexibilidade nos permitiu escrever algoritmos envolvendo vetores de forma bastante compacta. A mesma flexibilidade se estende para matrizes. Por exemplo, o trecho de algoritmo

```
para i de 1 ate 5 faca
  para j de 1 ate 3 faca
    tabela[i, j] <- i + j
  fimpara
fimpara
```

atribui o valor $i + j$ para o elemento da i -ésima linha e j -ésima coluna de *tabela*. Já o trecho de algoritmo

```

para  $i$  de 1 ate 5 faca
  para  $j$  de 1 ate 3 faca
    escreva( "Entre com o elemento [",  $i$  , ",",  $j$  , ": " )
    leia( tabela[ $i$ ,  $j$ ] )
  fimpara
fimpara

```

só difere do anterior pelo corpo do laço mais interno, que faz a leitura de cada elemento da matriz *tabela* ao invés de uma atribuição de valor. De forma geral, a manipulação de todos os elementos de uma matriz, um de cada vez, é feita com dois laços aninhados, como nos exemplos acima. De fato, podemos facilmente modificar qualquer um dos dois trechos de algoritmo acima para escrever, ao invés de ler, o conteúdo de todos os elementos da matriz *tabela*:

```

para  $i$  de 1 ate 5 faca
  para  $j$  de 1 ate 3 faca
    escreva( "tabela [",  $i$  , ",",  $j$  , " ] = " , tabela[ $i$ ,  $j$ ] )
  fimpara
fimpara

```

16.2 Exemplos

Em geral, o uso de matrizes é mais frequente na resolução de problemas computacionais de Engenharia, os quais envolvem álgebra linear numérica. Neste contexto, variáveis do tipo matriz possuem uma relação direta com a noção de matriz em Matemática. No entanto, vários programas de computador utilizam matrizes para outras finalidades, que variam desde a representação de uma simples planilha a grades de interfaces gráficas. Os exemplos que veremos a seguir têm por finalidade familiarizar o leitor com a manipulação de matrizes. Por isso, eles são aplicações demasiadamente simples e que estão relacionadas à noção de matriz em Matemática.

16.2.1 Soma de duas matrizes

Aprendemos no Ensino Médio que se A e B forem matrizes de dimensão $m \times n$, então $C = A + B$ também será $m \times n$ e tal que $c_{ij} = a_{ij} + b_{ij}$, para todo $i \in \{1, \dots, m\}$ e todo $j \in \{1, \dots, n\}$. Por exemplo, se

$$A = \begin{bmatrix} 8 & 6 \\ 1 & 4 \\ 5 & 7 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 1 & 2 \\ 0 & 3 \\ 2 & 1 \end{bmatrix},$$

então

$$C = A + B = \begin{bmatrix} 9 & 8 \\ 1 & 7 \\ 7 & 8 \end{bmatrix}.$$

Com a definição acima em mente, podemos construir um algoritmo para ler duas matrizes, A e B , de mesma dimensão, calcular a matriz soma, C , e escrever seus elementos. Vamos

representar as matrizes A , B e C por variáveis a , b e c do tipo matriz de elementos do tipo real. Assim como fizemos com vetores, vamos declarar a , b e c como tendo “muitas” linhas e colunas. Em seguida, solicitamos ao usuário que nos forneça o número, m , de linhas e o número, n , de colunas das matrizes. A próxima etapa é a leitura dos $m \times n$ elementos das matrizes A e B (veja Algoritmo 16.1).

```

1 algoritmo "Soma de duas matrizes"
2 var
3   a, b, c : vetor [ 1..100 , 1..100 ] de real
4   i, j, m, n : inteiro
5 inicio
6   escreva( "Entre com o numero de linhas das matrizes (<= 100): ")
7   repita
8     leia( m )
9   ate m <= 100
10  escreva( "Entre com o numero de colunas das matrizes (<= 100): ")
11  repita
12    leia( n )
13  ate n <= 100
14  para i de 1 ate m faca
15    para j de 1 ate n faca
16      escreva( "Entre com o elemento a[ ", i , "," , j , "]: " )
17      leia( a[ i , j ] )
18      escreva( "Entre com o elemento b[ ", i , "," , j , "]: " )
19      leia( b[ i , j ] )
20    fimpara
21  fimpara

```

Algoritmo 16.1: Primeira parte do algoritmo para somar duas matrizes.

Uma vez que tenhamos os elementos de A e B , precisamos calcular a matriz soma $C = A + B$. Por definição, sabemos que $c_{ij} = a_{ij} + b_{ij}$, para todo $i \in \{1, \dots, m\}$ e todo $j \in \{1, \dots, n\}$. Isto é,

$$c[i , j] \leftarrow a[i , j] + b[i , j]$$

para todos os valores que as variáveis i e j assumem nos intervalos de 1 a m e 1 a n , respectivamente. A soma e atribuição acima podem ser feitas com dois laços aninhados (veja Algoritmo 16.2).

```

1   para i de 1 ate m faca
2     para j de 1 ate n faca
3       c[ i , j ] <- a[ i , j ] + b[ i , j ]
4     fimpara
5  fimpara

```

Algoritmo 16.2: Segunda parte do algoritmo para somar duas matrizes.

Finalmente, os elementos da matriz C são escritos como mostrado em Algoritmo 16.3.

```

1  para i de 1 ate m faca
2      para j de 1 ate n faca
3          escreva( "c[ ", i , ", " , j , "]" = " , c[ i , j ] , " " )
4      fimpara
5  escreval( "" )
6  fimpara
7  fimalgoritmo

```

Algoritmo 16.3: Terceira parte do algoritmo para somar duas matrizes.

16.2.2 Cálculo de norma matricial

Em Álgebra Linear, aprendemos que o modo usual de expressarmos a “magnitude” de um vetor ou matriz é através de um escalar denominado *norma*. Uma norma matricial é uma função, $\|\cdot\| : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$, que associa um número real a cada matriz e que satisfaz às seguintes condições:

- $\|A\| \geq 0$ e $\|A\| = 0$ se, e somente se, todo elemento de A é igual a zero,
- $\|A + B\| \leq \|A\| + \|B\|$ e
- $\|k \cdot A\| = |k| \cdot \|A\|$,

onde $A, B \in \mathbb{R}^{m \times n}$ são matrizes de mesma dimensão e $k \in \mathbb{R}$ é um escalar. Uma norma matricial bastante conhecida e utilizada em cálculos numéricos é a *norma de soma máxima de linha*:

$$\|A\|_{\infty} = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|.$$

Basicamente, a norma soma máxima de linha é igual a maior soma que obtemos quando somamos o valor absoluto dos elementos de uma linha da matriz. Por exemplo, se a matriz A é igual a

$$\begin{bmatrix} 2 & -1 \\ 3 & 5 \end{bmatrix},$$

então

$$\|A\|_{\infty} = \max\{|2| + |-1|, |3| + |5|\} = \max\{3, 8\} = 8.$$

O Algoritmo 16.2.2 lê uma matriz e calcula e escreve a norma de soma máxima de linha da matriz.

16.2.3 Cálculo da matriz transposta

A transposta de uma matriz A , denotada por A^T , é uma matriz obtida trocando-se as linhas de A pelas colunas, de modo que a linha i se torne a coluna i e a coluna j se torne a linha j . Por exemplo, se

$$A = \begin{bmatrix} 5 & 2 & 0 \\ 6 & 9 & 1 \\ 3 & 4 & 2 \\ 7 & 0 & 8 \\ 1 & 5 & 6 \end{bmatrix},$$

então

$$A^T = \begin{bmatrix} 5 & 6 & 3 & 7 & 1 \\ 2 & 9 & 4 & 0 & 5 \\ 0 & 1 & 2 & 8 & 6 \end{bmatrix}.$$

Em geral, temos que se A é uma matriz m por n , então A^T é uma matriz n por m tal que $a_{ij}^T = a_{ji}$, para todo $i \in \{1, \dots, n\}$ e todo $j \in \{1, \dots, m\}$. Usando esta definição, o Algoritmo 16.5 calcula e escreve os elementos da transposta de uma matriz de reais dada como entrada do algoritmo. Preste bastante atenção no uso correto dos índices nos laços aninhados do algoritmo.

16.3 Exercícios propostos

1. Escreva, usando a linguagem Portugol da ferramenta VISUALG, três declarações distintas de uma mesma variável do tipo matriz com 10 por 20 elementos do tipo caractere. A diferença entre as declarações deve se dar, apenas, nas faixas de valores que definem a dimensão da matriz.
2. Escreva um algoritmo que leia valores para cada elemento de uma matriz B de 100 linhas e 200 colunas de números reais, calcule a soma dos elementos da linha de índice 40 e da coluna de índice 30 e escreva o resultado dessas duas somas.
3. Escreva um algoritmo para calcular a matriz resultante da multiplicação de um escalar por uma dada matriz. A entrada do algoritmo é composta por dois inteiros, m e n , um número real, λ , e pelos elementos de uma matriz, A , m por n de reais. Assuma que $m \leq 100$ e $n \leq 50$. A saída do algoritmo é composta pelos elementos da matriz $\lambda \cdot A$.
4. Escreva um algoritmo para calcular o vetor resultante da multiplicação de um dado vetor por uma dada matriz. A entrada do algoritmo é composta por dois inteiros m e n , pelos elementos de um vetor V de tamanho m de números reais e pelos elementos de uma matriz A , m por n de reais. Assuma que $m \leq 100$ e $n \leq 50$. A saída do algoritmo é composta pelos elementos do vetor $V \cdot A$ de tamanho n de números reais.
5. Escreva um algoritmo para verificar se existem elementos repetidos em uma dada matriz. A entrada do algoritmo é composta por dois inteiros, m e n , e pelos elementos de uma matriz, A , m por n de inteiros. Assuma que $m \leq 100$ e $n \leq 50$. A saída do algoritmo é a mensagem “sim” se A contiver pelo menos um elemento repetido e “não” caso contrário.
6. Deseja-se escrever um algoritmo para atualizar as contas correntes dos clientes de uma agência bancária. É dado o cadastro de n clientes contendo, para cada cliente, o número de sua conta e o seu saldo; o cadastro está ordenado pelo número da conta. Em seguida, é dado o número m de operações efetuadas no dia e, para cada operação, o número da conta, uma letra **C** ou **D** indicando se a operação é de crédito ou débito e o valor da operação.
A entrada do algoritmo é um inteiro positivo, n , uma sequência com n pares, (número da conta, saldo), de cada um dos n clientes, um número inteiro não-negativo m e uma sequência de m pares, (número da conta, letra C ou D), com m transações bancárias. A saída do algoritmo é uma sequência de n pares, (número da conta, saldo), onde o saldo da conta é o saldo após as m transações terem sido efetuadas. Assuma que $n \leq 40$ e $m \leq 1000$.
7. Escreva um algoritmo para ler um número inteiro positivo, n , e escrever as n primeiras

linhas do triângulo de Pascal¹.

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 4 10 10 5 1
⋮
```

<

¹Descoberto em 1654 pelo matemático francês Blaise Pascal.

```

1 algoritmo "Norma da soma maxima de linha"
2 var
3   a : vetor [ 1..100 , 1..100 ] de real
4   soma, norma : real
5   i, j, m, n : inteiro
6 inicio
7   escreva( "Entre com o numero de linhas da matriz (<= 100): ")
8   repita
9     leia( m )
10  ate m <= 100
11  escreva( "Entre com o numero de colunas da matriz (<= 100): ")
12  repita
13    leia( n )
14  ate n <= 100
15  para i de 1 ate m faca
16    para j de 1 ate n faca
17      escreva( "Entre com o elemento a[ ", i , ", " , j , "]: " )
18      leia( a[ i , j ] )
19    fimpara
20  fimpara
21  norma <- 0
22  para i de 1 ate m faca
23    soma <- 0
24    para j de 1 ate n faca
25      se a[ i , j ] < 0 entao
26        soma <- soma - a[ i , j ]
27      senao
28        soma <- soma + a[ i , j ]
29      fimse
30    fimpara
31    se soma > norma entao
32      norma <- soma
33    fimse
34  fimpara
35  escreva( "A norma da soma maxima de linha da matriz e: ", norma )
36 fimalgoritmo

```

Algoritmo 16.4: Norma de soma máxima de linha

```

1 algoritmo "Transposta de uma matriz"
2 var
3   a, b : vetor [ 1..100 , 1..100 ] de real
4   i, j, m, n : inteiro
5 inicio
6   escreva( "Entre com o numero de linhas da matriz (<= 100): ")
7   repita
8     leia( m )
9   ate m <= 100
10  escreva( "Entre com o numero de colunas da matriz (<= 100): ")
11  repita
12    leia( n )
13  ate n <= 100
14  para i de 1 ate m faca
15    para j de 1 ate n faca
16      escreva( "Entre com o elemento a[ ", i , "," , j , "]: " )
17      leia( a[ i , j ] )
18      b[ j , i ] <- a[ i , j ]
19    fimpara
20  fimpara
21  para i de 1 ate n faca
22    para j de 1 ate m faca
23      escreva( "b[ ", i , "," , j , "] = " , b[ i , j ] , " " )
24    fimpara
25    escreval( "" )
26  fimpara
27 fimalgoritmo

```

Algoritmo 16.5: Cálculo da matriz transposta.

MATRIZES 2

17.1 Mais exemplos

Nesta aula, veremos mais dois algoritmos envolvendo matrizes. O primeiro deles calcula a matriz resultante da multiplicação de duas matrizes e utiliza três laços do tipo para aninhados. O segundo verifica se uma dada matriz é ou não é um quadrado mágico e utiliza uma combinação de laços do tipo para e enquanto. Os dois algoritmos são mais elaborados do que os que vimos antes.

17.1.1 Multiplicação de duas matrizes

Sejam A e B duas matrizes com dimensões $m \times p$ e $p \times n$, respectivamente. Então, a multiplicação

$$A \times B$$

da matriz A pela matriz B resulta em uma matriz, C , de dimensão $m \times n$ tal que o elemento c_{ij} é igual a

$$\sum_{k=1}^p a_{ik} \cdot b_{kj},$$

para $i \in \{1, \dots, m\}$ e $j \in \{1, \dots, n\}$. Em outras palavras, cada elemento c_{ij} é igual ao produto

$$\begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{ip} \end{bmatrix} \cdot \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{pj} \end{bmatrix}$$

da i -ésima linha de A pela j -ésima coluna de B . Por exemplo, se

$$A = \begin{bmatrix} 8 & 6 \\ 1 & 4 \\ 5 & 7 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 2 & 1 \end{bmatrix},$$

então

$$C = A \times B = A \cdot B = \begin{bmatrix} 8 & 6 \\ 1 & 4 \\ 5 & 7 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 0 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 26 & 28 & 6 \\ 13 & 10 & 4 \\ 26 & 24 & 7 \end{bmatrix},$$

pois

$$c_{11} = [8 \ 6] \cdot \begin{bmatrix} 1 \\ 3 \end{bmatrix} = 8 \cdot 1 + 6 \cdot 3 = 8 + 18 = 26,$$

$$c_{12} = [8 \ 6] \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 8 \cdot 2 + 6 \cdot 2 = 16 + 12 = 28,$$

$$c_{13} = [8 \ 6] \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 8 \cdot 0 + 6 \cdot 1 = 0 + 6 = 6,$$

$$c_{21} = [1 \ 4] \cdot \begin{bmatrix} 1 \\ 3 \end{bmatrix} = 1 \cdot 1 + 4 \cdot 3 = 1 + 12 = 13,$$

$$c_{22} = [1 \ 4] \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 1 \cdot 2 + 4 \cdot 2 = 2 + 8 = 10,$$

$$c_{23} = [1 \ 4] \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 \cdot 0 + 4 \cdot 1 = 0 + 4 = 4,$$

$$c_{31} = [5 \ 7] \cdot \begin{bmatrix} 1 \\ 3 \end{bmatrix} = 5 \cdot 1 + 7 \cdot 3 = 5 + 21 = 26,$$

$$c_{32} = [5 \ 7] \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 5 \cdot 2 + 7 \cdot 2 = 10 + 14 = 24,$$

$$c_{33} = [5 \ 7] \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 5 \cdot 0 + 7 \cdot 1 = 0 + 7 = 7.$$

O que queremos, agora, é desenvolver um algoritmo para ler duas matrizes, A e B , com dimensões $m \times p$ e $p \times n$, respectivamente, e calcular e escrever a matriz $C = A \times B$ de dimensão $m \times n$. A entrada deste algoritmo é composta das dimensões m , n e p e dos elementos de A e B . A saída é composta pelos elementos da matriz C . É importante notar que o número de colunas de A e o número de colunas de B devem ser iguais. Como esses números são representados por p , não precisamos de quatro valores de entrada (mas sim três) para representar as dimensões de A e B .

O passo crucial do algoritmo que queremos construir é o que calcula os elementos de C , pois os demais passos se referem à entrada e saída de dados, que são realizadas da mesma forma que vimos em outros exemplos. Os elementos c_{ij} podem ser calculados com o seguinte código:

```
para  $i$  de 1 ate  $m$  faca  
  para  $j$  de 1 ate  $n$  faca  
    // calcule o elemento  $c_{ij}$  da matriz  $C = A \times B$   
  fimpara  
fimpara
```

Como sabemos,

$$c_{ij} = \sum_{k=1}^p a_{ik} \cdot b_{kj}.$$

A soma acima pode ser feita com outro laço do tipo para e uma variável acumuladora, que pode ser a própria célula da variável matriz que representa C . De fato, se as matrizes A , B e C são

representadas pelas variáveis a , b e c , respectivamente, então cada c_{ij} pode ser obtido com o código

```

c[i, j] <- 0
para k de 1 ate p faca
    c[i, j] <- c[i, j] + a[i, k] * b[k, j]
fimpara

```

Note que os valores de i e j são “fixos” no laço acima. Apenas o valor da variável contadora k varia.

Substituindo o código acima para calcular *um* único c_{ij} na linha de comentário dos laços aninhados que vimos antes, obtemos um trecho de código que calcula *todos* os elementos c_{ij} da matriz C :

```

para i de 1 ate m faca
    para j de 1 ate n faca
        c[i, j] <- 0
        para k de 1 ate p faca
            c[i, j] <- c[i, j] + a[i, k] * b[k, j]
        fimpara
    fimpara
fimpara

```

O algoritmo completo para a multiplicação de duas matrizes está em Algoritmo [17.1.1](#).

17.1.2 Quadrado mágico

A matriz

$$\begin{bmatrix} 8 & 0 & 7 \\ 4 & 5 & 6 \\ 3 & 10 & 2 \end{bmatrix}$$

possui a seguinte propriedade: a soma dos elementos de qualquer uma de suas linhas, colunas, diagonal principal ou diagonal secundária é igual a 15. Quando isto acontece, dizemos que a matriz é um *quadrado mágico*. Um problema computacional interessante é o de verificar se uma dada matriz quadrada é ou não é um quadrado mágico. A entrada deste problema é um inteiro n , com $n \geq 2$, e uma matriz quadrada, digamos A , de ordem n . A saída do problema é a mensagem “A matriz não é um quadrado mágico” ou a mensagem “A matriz é um quadrado mágico”.

O que queremos aqui é construir um algoritmo para resolver o problema acima. Basicamente, o algoritmo que queremos é um verificador de uma propriedade. Ele deve *decidir* se a entrada satisfaz uma dada propriedade. A saída do algoritmo deve indicar se a propriedade é ou não é satisfeita pela entrada. Algoritmos desta natureza são conhecidos como *algoritmos de decisão*.

Para construir nosso algoritmo, valeremo-nos de uma ideia simples. Suponha que recebemos m números reais, x_1, x_2, \dots, x_m , e queremos decidir se esses números são todos iguais. Para fazer isso de forma eficiente, consideramos a seguinte propriedade: *se a, b e c são três números tais que $a = b$ e $b = c$, então $a = c$* . Usando esta propriedade, escolhemos qualquer um dos números que recebemos, digamos x_i , e comparamos este número com todos os números x_j , tais que $i \neq j$

e $j \in \{1, \dots, m\}$. O que podemos dizer se todas as comparações resultam em igualdade? A propriedade acima nos garante que todos os números são iguais. Por outro lado, se alguma comparação resultar em desigualdade, sabemos que x_1, x_2, \dots, x_m não são todos iguais. Mais ainda, quando x_1, x_2, \dots, x_m não forem todos iguais, alguma comparação de x_i com outro número resultará em desigualdade. Caso contrário, todos os x_1, x_2, \dots, x_m seriam iguais¹.

Mas, o que a ideia acima tem a ver com o nosso problema? Bem, se considerarmos que cada x_i é a soma dos elementos de uma linha, coluna, diagonal principal ou diagonal secundária da matriz, o problema que queremos resolver é “idêntico” ao problema que acabamos de discutir. Isto sugere que devemos calcular a soma dos elementos de cada linha, coluna, diagonal principal e diagonal secundária da matriz, separadamente, e comparar os valores obtidos com a estratégia acima. Esta é a estratégia de solução que utilizaremos, mas para fazer esta estratégia funcionar de forma eficiente, faremos as comparações, uma de cada vez, e enquanto obtivermos igualdade. Assim que uma desigualdade for descoberta, não precisamos mais continuar comparando, pois já sabemos que *todos* os valores não são iguais. O primeiro passo da nossa estratégia é o cálculo da soma dos elementos da primeira linha da matriz. Isto equivale a escolher o elemento x_i , que pode ser a soma dos elementos de qualquer linha, coluna, diagonal principal ou diagonal secundária da matriz. Arbitrariamente, escolhemos a primeira linha da matriz. O trecho de código dado a seguir calcula a soma dos elementos da primeira linha da matriz:

```
s1 <- 0
para j de 1 ate n faca
    s1 <- s1 + a[1, j]
fimpara
```

Assumimos que a matriz está representada pela variável a e acumula a soma na variável $s1$.

Uma vez que saibamos o valor da soma dos elementos de *uma* linha, devemos comparar este valor com o valor da soma dos elementos das demais linhas, colunas, diagonal principal e diagonal secundária da matriz. Mas, não precisamos realizar *todas* as comparações se isto não for necessário. Faremos uma comparação por vez. Enquanto as comparações resultarem em igualdade, continuamos comparando. Mas, se uma desigualdade ocorrer, paramos de comparar.

O trecho de código dado a seguir calcula a soma dos elementos de uma linha da matriz que não seja a primeira. Em seguida, compara o valor desta soma com $s1$. Se a comparação resultar em igualdade, outra linha da matriz é considerada para cálculo da soma de seus elementos e comparação com $s1$. Se a comparação resultar em desigualdade, o cálculo da soma é abortado. Para que isto seja possível, usamos uma variável lógica de nome *igual* e um laço do tipo enquanto:

```
igual <- verdadeiro
i <- 2
enquanto igual e (i <= n) faca
    s2 <- 0
    para j de 1 ate n faca
        s2 <- s2 + a[i, j]
    fimpara
    se s1 = s2 entao
        i <- i + 1
```

¹Concluimos isso por um argumento baseado em contradição!

```

    senao
         $igual \leftarrow \text{falso}$ 
    fimse
finenquanto

```

O laço que acabamos de ver implementa o *teste das linhas*. Se a variável *igual* possuir o valor verdadeiro após o término deste laço, sabemos que a soma dos elementos de qualquer uma das linhas da matriz é igual à soma dos elementos de qualquer outra linha da matriz. Mas, ainda não podemos afirmar que a matriz *a* é um quadrado mágico. Para isso, devemos realizar o *teste das colunas* e o *teste das diagonais*. Por outro lado, se a variável *igual* possuir o valor falso após o término do teste das linhas, já podemos afirmar que a matriz *a* **não** é um quadrado mágico e não precisamos realizar mais nenhum teste. O trecho de algoritmo a seguir mostra como realizar o teste das colunas se, e somente se, a variável *igual* possuir o valor verdadeiro após o término do teste das linhas. Este trecho é muito parecido com o que implementa o teste das linhas:

```

     $j \leftarrow 1$ 
    enquanto  $igual$  e  $(j \leq n)$  faca
         $s2 \leftarrow 0$ 
        para  $i$  de 1 ate  $n$  faca
             $s2 \leftarrow s2 + a[i, j]$ 
        fimpara
        se  $s1 = s2$  entao
             $j \leftarrow j + 1$ 
        senao
             $igual \leftarrow \text{falso}$ 
        fimse
    finenquanto

```

É importante observar que tanto o teste das linhas quanto o teste das colunas será abortado assim que uma desigualdade for encontrada, pois isto faz com que a variável *igual* receba o valor falso na estrutura condicional que verifica se *s1* e *s2* possuem o mesmo valor. Após o teste das colunas, realizamos o teste da diagonal principal, que é mostrado no seguinte trecho de algoritmo:

```

se  $igual$  entao
     $s2 \leftarrow 0$ 
    para  $i$  de 1 ate  $n$  faca
         $s2 \leftarrow s2 + a[i, i]$ 
    fimpara
     $igual \leftarrow (s1 = s2)$ 
fimse

```

O teste da diagonal secundária é quase idêntico e realizado logo a seguir:

```

se  $igual$  entao
     $s2 \leftarrow 0$ 
    para  $i$  de 1 ate  $n$  faca
         $s2 \leftarrow s2 + a[i, n - i + 1]$ 
    fimpara

```

```

    igual <- (s1 = s2)
fimse

```

É importante observar também que um teste, seja ele de coluna ou diagonal, só será executado se a matriz passar no teste anterior. Finalmente, após todos os testes acima, devemos verificar o valor de *igual* para saber se a matriz passou ou não por *todos* os testes. Note que a matriz passa por todos os testes se, e somente se, o valor da variável *igual* é verdadeiro. Então, o último trecho do algoritmo que verifica se a matriz é ou não é um quadrado mágico é como segue:

```

se igual entao
    escreva( "A matriz e um quadrado magico" )
senao
    escreva( "A matriz nao e um quadrado magico" )
fimse

```

O algoritmo completo é deixado como exercício para o aluno. Note apenas que deixamos de mostrar, apenas, a seção de declaração de variáveis e o trecho que realiza a leitura da entrada de dados, que consiste da ordem n da matriz e dos elementos da matriz. Esta tarefa deve ser trivial².

17.2 Exercícios propostos

1. Dizemos que uma matriz de inteiros, A , n por n , é uma *matriz de permutação* se em cada linha e em cada coluna houver $n - 1$ elementos nulos e um único elemento 1. Por exemplo,

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

é uma matriz de permutação, mas

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

não é de permutação. Escreva um algoritmo para determinar se uma dada matriz quadrada de inteiros é de permutação. A entrada do algoritmo consiste de um inteiro positivo, n , e dos números inteiros que compõem a matriz. A saída do algoritmo é a mensagem “é de permutação” se a matriz é de permutação e a mensagem “não é de permutação” caso contrário. Assuma que $n \leq 1000$.

2. Escreva um algoritmo para ler um número inteiro positivo, n , e escrever as n primeiras linhas do triângulo de Pascal³.

²Pelo menos para aqueles que fizeram os exercícios da aula anterior.

³Descoberto em 1654 pelo matemático francês Blaise Pascal.

$$\begin{array}{cccccc}
1 & & & & & \\
1 & 1 & & & & \\
1 & 2 & 1 & & & \\
1 & 3 & 3 & 1 & & \\
1 & 4 & 6 & 4 & 1 & \\
1 & 4 & 10 & 10 & 5 & 1 \\
\vdots & & & & &
\end{array}$$

3. Uma matriz $D_{8 \times 8}$ pode representar a posição atual de um jogo de damas, sendo que 0 indica uma casa vazia, 1 indica uma casa ocupada por uma peça branca e -1 indica uma casa ocupada por uma peça preta. Supondo que as peças pretas estão se movendo no sentido crescente das linhas da matriz D , escreva um algoritmo para determinar as posições das peças pretas que:

- a) podem tomar peças brancas;
- b) podem mover-se sem tomar peças;
- c) não podem se mover.

A entrada do algoritmo consiste dos elementos da matriz $D_{8 \times 8}$ e a saída consiste dos índices das posições que satisfazem (a), seguidos pelos índices das posições que satisfazem (b), seguidos pelos índices das posições que satisfazem (c).

4. Suponha que os elementos a_{ij} de uma matriz inteira $A_{n \times n}$ representem os custos de transporte da cidade i para a cidade j . Então, dado um itinerário com k cidades, podemos calcular o custo total de cada itinerário. Por exemplo, se

$$A = \begin{bmatrix} 4 & 1 & 2 & 3 \\ 5 & 2 & 1 & 400 \\ 2 & 1 & 3 & 8 \\ 7 & 1 & 2 & 5 \end{bmatrix},$$

então o custo do itinerário 1 4 2 4 4 3 2 1 com $k = 8$ cidades (não necessariamente distintas) é igual a

$$a_{14} + a_{42} + a_{24} + a_{44} + a_{43} + a_{32} + a_{21} = 3 + 1 + 400 + 5 + 2 + 1 + 5 = 417.$$

Escreva um algoritmo que leia um inteiro positivo n , os elementos de uma matriz A , de n por n inteiros, um inteiro positivo $m \leq n * (n - 1)$, um inteiro positivo k e uma sequência de m itinerários, cada qual com k cidades, e que calcule e escreva o custo total de cada itinerário.

```

1 algoritmo "Multiplicacao de duas matrizes"
2 var
3   a, b, c : vetor [ 1..100 , 1..100 ] de real
4   i, j, m, p, n, k : inteiro
5 inicio
6   escreva( "Entre com o numero de linhas da primeira matriz (<= 100): ")
7   repita
8     leia( m )
9   ate m <= 100
10  escreva( "Entre com o numero de colunas da primeira matriz (<= 100): ")
11  repita
12    leia( p )
13  ate p <= 100
14  escreva( "Entre com o numero de colunas da segunda matriz (<= 100): ")
15  repita
16    leia( n )
17  ate n <= 100
18  para i de 1 ate m faca
19    para j de 1 ate p faca
20      escreva( "Entre com o elemento [ ", i , ", " , j , "]" da matriz A: " )
21      leia( a[ i , j ] )
22    fimpara
23  fimpara
24  para i de 1 ate p faca
25    para j de 1 ate n faca
26      escreva( "Entre com o elemento [ ", i , ", " , j , "]" da matriz B: " )
27      leia( b[ i , j ] )
28    fimpara
29  fimpara
30  para i de 1 ate m faca
31    para j de 1 ate n faca
32      c[ i , j ] <- 0
33      para k de 1 ate p faca
34        c[ i , j ] <- c[ i , j ] + a[ i , k ] * b[ k , j ]
35      fimpara
36    fimpara
37  fimpara
38  para i de 1 ate m faca
39    para j de 1 ate n faca
40      escreval( "c[ ", i , ", " , j , "]" = " , c[ i , j ] )
41    fimpara
42  fimpara
43 fimalgoritmo

```

Algoritmo 17.1: Multiplicação de 2 matrizes

MODULARIZAÇÃO - PARTE 1

Os algoritmos que temos construído até então são muito simples, pois resolvem problemas simples e apresentam apenas os componentes mais elementares dos algoritmos: constantes, variáveis, expressões condicionais e estruturas de controle. Entretanto, a maioria dos algoritmos resolve problemas complicados, cuja solução pode ser vista como formada de várias subtarefas ou *módulos*, cada qual resolvendo uma parte específica do problema.

Neste tópico, veremos como escrever um algoritmo constituído de vários módulos e como estes módulos trabalham em conjunto para resolver um determinado problema algorítmico.

18.1 O Quê e Por Quê?

Um **módulo** nada mais é do que um grupo de comandos que constitui um trecho de algoritmo com uma função bem definida e o mais independente possível das demais partes do algoritmo. Cada módulo, durante a execução do algoritmo, realiza uma tarefa específica da solução do problema e, para tal, pode contar com o auxílio de outros módulos do algoritmo. Desta forma, a execução de um algoritmo contendo vários módulos pode ser vista como um processo cooperativo.

A construção de algoritmos compostos por módulos, ou seja, a construção de algoritmos através de **modularização** possui uma série de vantagens:

- **Torna o algoritmo mais fácil de escrever.** O desenvolvedor pode focalizar pequenas partes de um problema complicado e escrever a solução para estas partes, uma de cada vez, ao invés de tentar resolver o problema como um todo de uma só vez.
- **Torna o algoritmo mais fácil de ler.** O fato do algoritmo estar dividido em módulos permite que alguém, que não seja o seu autor, possa entender o algoritmo mais rapidamente por tentar entender os seus módulos separadamente, pois como cada módulo é menor e mais simples do que o algoritmo monolítico correspondente, entender cada um deles separadamente é menos complicado do que tentar entender o algoritmo como um todo de uma só vez.
- **Eleva o nível de abstração.** É possível entender o que um algoritmo faz por saber apenas o *que* os seus módulos fazem, sem que haja a necessidade de entender os detalhes internos aos módulos. Além disso, se os detalhes nos interessam, sabemos exatamente onde examiná-los.
- **Economia de tempo, espaço e esforço.** Frequentemente, necessitamos executar uma mesma tarefa em vários lugares de um mesmo algoritmo. Uma vez que um módulo foi escrito, ele pode, como veremos mais adiante, ser “chamado” quantas vezes quisermos e de

onde quisermos no algoritmo, evitando que escrevamos a mesma tarefa mais de uma vez no algoritmo, o que nos poupará tempo, espaço e esforço.

- **Estende a linguagem.** Uma vez que um módulo foi criado, podemos utilizá-lo em outros algoritmos sem que eles sejam modificados, da mesma forma que usamos os operadores leia e escreva em nossos algoritmos.

A maneira mais intuitiva de proceder à modularização de problemas é realizada através da definição de um módulo principal, que organiza e coordena o trabalho dos demais módulos, e de módulos específicos para cada uma das subtarefas do algoritmo. O módulo principal solicita a execução dos vários módulos em uma dada ordem, os quais, antes de iniciar a execução, recebem dados do módulo principal e, ao final da execução, devolvem o resultado de suas computações.

18.2 Componentes de um módulo

Os módulos que estudaremos daqui em diante possuem dois componentes: corpo e interface. O **corpo** de um módulo é o grupo de comandos que compõe o trecho de algoritmo correspondente ao módulo. Já a **interface** de um módulo pode ser vista como a descrição dos dados de entrada e de saída do módulo. O conhecimento da interface de um módulo é tudo o que é necessário para a utilização correta do módulo em um algoritmo.

A interface de um módulo é definida em termos de parâmetros. Um **parâmetro** é um tipo especial de variável utilizado pelos módulos para receber ou comunicar valores de dados a outras partes do algoritmo. Existem três categorias de parâmetros:

- **parâmetros de entrada**, que permitem que valores sejam passados *para* um módulo a partir do algoritmo que solicitou sua execução;
- **parâmetros de saída**, que permitem que valores sejam passados *do* módulo para o algoritmo que solicitou sua execução; e
- **parâmetros de entrada e saída**, que permitem tanto a passagem de valores *para* o módulo quanto a passagem de valores *do* módulo para o algoritmo que solicitou sua execução.

Quando criamos um módulo, especificamos o número e os tipos das variáveis correspondentes aos parâmetros que ele necessita, isto determina a interface do módulo. Qualquer algoritmo que use um módulo deve utilizar os parâmetros que são especificados na interface do módulo para se comunicar com ele.

18.3 Funções e Procedimentos

Em Portugal, temos duas formas de definir um módulo:

- **função:** uma função é um módulo que produz um único valor de saída. Ela pode ser vista como uma expressão que é avaliada para um único valor, sua saída, assim como uma função em Matemática.
- **procedimento:** um procedimento é um tipo de módulo usado para várias tarefas, não produzindo valores de saída.

As diferenças entre as definições de função e procedimento permitem determinar se um módulo para uma dada tarefa deve ser implementado como uma função ou procedimento. Nas definições acima, estamos considerando que a produção de um valor de saída por uma função é diferente da utilização de parâmetros de saída ou de entrada e saída. Veremos mais adiante que os parâmetros de saída e de entrada e saída modificam o valor de uma variável do algoritmo que a chamou, diferentemente do valor de saída produzido por uma função que será um valor a ser usado em uma atribuição ou envolvido em alguma expressão.

Como exemplo da diferenciação do uso de funções ou procedimentos, considere, por exemplo, que um módulo para determinar o menor de dois números é necessário. Neste caso, o módulo deve ser implementado como uma função, pois ele vai produzir um valor de saída. Por outro lado, se um módulo para determinar o maior e o menor valor de uma seqüência de números é requerido, ele deverá produzir seus resultados via os parâmetros de saída e será implementado como um procedimento, pois ele vai produzir dois valores de saída.

A fim de escrevermos uma função ou procedimento, precisamos construir as seguintes partes: interface e corpo. Como dito antes, a interface é uma descrição dos parâmetros do módulo, enquanto corpo é a seqüência de instruções do módulo. A interface de uma função tem a seguinte forma geral:

funcao <nome> (<lista de parâmetros formais>) :<tipo de retorno>

onde

- *nome* é um identificador único que representa o nome da função;
- *lista de parâmetros formais* é uma lista dos parâmetros da função;
- *tipo de retorno* é o tipo do valor de retorno da função.

Por exemplo:

funcao *quadrado* (real *r*) : real

Logo após a descrição da *interface* podemos descrever o corpo do algoritmo. Para delimitar os comandos que fazem parte da função, utilizamos fimfunção.

O corpo de uma função é uma seqüência de comandos que compõe a função e que sempre finaliza com um comando especial denominado **comando de retorno**. O comando de retorno é da seguinte forma

retorne <valor de retorno>

onde *valor de retorno* é o valor de saída produzido pela função. Por exemplo:

retorne *x*

Vejamos um exemplo de uma função. Suponha que desejemos construir uma função para calcular o quadrado de um número. O número deve ser passado para a função, que calculará o seu quadrado e o devolverá para o algoritmo que a solicitou. Vamos denominar esta função de *quadrado*, como mostrado a seguir:

```

// função para calcular o quadrado de um número
função quadrado (real r) : real
var
    // declaração de variáveis
    x : real
inicio
    // calcula o quadrado
    x ← r * r

    // retorna o quadrado calculado
    retorne x
fimfuncao

```

A interface de um procedimento tem a seguinte forma geral:

procedimento <nome> (<lista de parâmetros formais>)

onde

- *nome* é um identificador único que representa o nome do procedimento;
- *lista de parâmetros formais* é uma lista dos parâmetros do procedimento.

Por exemplo,

procedimento *ler_número* (var *val* : real)

O corpo de um procedimento é uma seqüência de comandos que não possui comando de retorno, pois apenas funções possuem tal tipo de comando. Para delimitar os comandos que fazem parte do procedimento, utilizamos fimprocedimento.

Vejam os um exemplo de um procedimento. Suponha que desejemos construir um procedimento para ler um número e passar o número lido para o algoritmo que solicitou a execução do algoritmo através de um parâmetro de saída. Denominemos este procedimento de *ler_número*, como mostrado a seguir:

```

// procedimento para ler um número
procedimento ler_número (var val : real )
inicio
    // Solicita a entrada do número
    escreva ("Entre com um número:" )
    leia(val)

fimprocedimento

```

Aqui, *val* é o parâmetro de saída que conterà o valor de entrada que será passado para o algoritmo que solicitar a execução do procedimento *ler_número*. A palavra-chave var, que antecede o nome da variável na lista de parâmetros formais do procedimento, é utilizada para definir *val* como parâmetro de saída ou entrada e saída.

18.4 Chamando Funções e Procedimentos

Funções e procedimentos não são diferentes apenas na forma como são implementados, mas também na forma como a solicitação da execução deles, ou simplesmente **chamada**, deve ser realizada. A chamada de uma função é usada como um valor constante que deve ser atribuído a uma variável ou como parte de uma expressão, enquanto a chamada de um procedimento é realizada como um comando a parte.

Como exemplo, considere um algoritmo para ler um número e exibir o seu quadrado. Este algoritmo deve utilizar o procedimento *ler_número* e a função *quadrado*, vistos antes, para ler o número e obter o seu quadrado, respectivamente. O algoritmo segue abaixo:

```

// algoritmo para calcular e exibir o quadrado de um número dado como entrada
algoritmo "calcula_quadrado"
var
    // declaração de variáveis
    num, x : real

inicio
    // lê um número
    ler_número(num)

    // calcula o quadrado do número lido
     $x \leftarrow \text{quadrado}(num)$ 

    // escreve o quadrado
    escreva ("O quadrado do número é: ", x)

finalgoritmo

```

Note a diferença da chamada do procedimento *ler_número* para a chamada da função *quadrado*. Na chamada do procedimento, temos uma instrução por si só. Por outro lado, a chamada da função ocupa o lugar de um valor ou expressão em uma instrução de atribuição. Isto porque a chamada

quadrado(num)

é substituída pelo valor de retorno da função.

Tanto na chamada do procedimento *ler_número* quanto na chamada da função *quadrado*, temos um parâmetro: a variável *num*. Na chamada do procedimento *ler_número*, *num* é um parâmetro de saída, como definido na interface do procedimento, e, portanto, após a execução do procedimento, *num* conterá o valor lido dentro do procedimento. Já na chamada da função *quadrado*, *num* é um parâmetro de entrada, como definido na interface da função, e, assim sendo, o valor de *num* é passado para a função.

A variável *num* é denominada **parâmetro real**. Um parâmetro real especifica um valor passado para o módulo pelo algoritmo que o chamou ou do módulo para o algoritmo que o chamou. Os parâmetros formais são aqueles da declaração do módulo. A lista de parâmetros reais deve concordar em número, ordem e tipo com a lista de parâmetros formais.

18.5 Passagem de Parâmetros

Os valores dos parâmetros reais de entrada são passados por um mecanismo denominado **cópia**, enquanto os valores dos parâmetros reais de saída e entrada e saída são passados por um mecanismo denominado **referência**.

O mecanismo de passagem por cópia funciona da seguinte forma. O valor do parâmetro real (uma constante ou o valor de uma variável ou expressão) de entrada é atribuído ao parâmetro formal quando da chamada do procedimento/função. Por exemplo, na chamada

$$x \leftarrow \text{quadrado}(\text{num})$$

o valor da variável *num* é atribuído ao parâmetro formal *r* da função *quadrado*.

No mecanismo de passagem por referência, quando da chamada do procedimento/função, o parâmetro formal passa a compartilhar a mesma área de armazenamento de parâmetro real, assim, qualquer alteração no valor do parâmetro formal feita pelo procedimento/função acarretará uma modificação no parâmetro real quando do término do procedimento/função. Sendo assim, quando a chamada

$$\text{ler_número}(\text{num})$$

é executada, a variável real *num* e a variável formal *val* (que está precedida da palavra chave ref) compartilham uma mesma área de armazenamento, assim, *num* e *val* contêm o mesmo valor. Quando é feita a leitura do dado e armazenada em *val*, esta atualização afetará o valor de *num*. Ao término do procedimento a variável *num* conterá o valor atualizado.

Devemos observar que os parâmetros reais de saída e de entrada e saída devem *obrigatoriamente* ser variáveis, uma vez que não faz sentido modificar o valor de uma constante ou de uma expressão.

18.6 Escopo de Dados e Código

O **escopo** de um módulo (ou variável) de um algoritmo é a parte ou partes do algoritmo em que o módulo (ou variável) pode ser referenciado. Quando iniciamos o estudo de modularização é natural nos perguntarmos qual é o escopo de um dado módulo e das constantes ou variáveis nele presentes. Em particular, o escopo de um módulo determina quais são os demais módulos do algoritmo que podem chamá-lo e quais ele pode chamar.

Os módulos de um algoritmo são organizados por níveis. No primeiro nível, temos apenas o algoritmo principal. Aqueles módulos que devem ser acessados pelo algoritmo principal devem ser escritos dentro dele e, nesta condição, são ditos pertencerem ao segundo nível. Os módulos escritos dentro de módulos de segundo nível são ditos módulos de terceiro nível. E assim sucessivamente. Por exemplo, veja o algoritmo 18.1.

No algoritmo 18.1, desejávamos que a função *quadrado* e o procedimento *ler_numero* fossem chamados pelo módulo principal e, por isso, escrevemos tais módulos dentro do módulo principal, no segundo nível da hierarquia modular do algoritmo.

A regra para determinar o escopo de um módulo é bastante simples: um módulo *X* escrito dentro de um módulo *A* qualquer pode ser acessado apenas pelo módulo *A* ou por qualquer módulo escrito dentro de *A* ou descendente (direto ou não) de algum módulo dentro de *A*.

Como exemplo, considere um algoritmo no qual o módulo principal contém outros quatro módulos, *S1*, *S2*, *F1*, *F2*. Por sua vez, *S1* contém mais dois módulos, *S3* e *F3*; e *F2* contém os módulos *S4* e *F4*. De acordo com as regras de escopo descritas anteriormente, o módulo *F3* só pode ser chamado por *S1* e *S3*, enquanto o módulo *F1* pode ser acessado por todos os módulos.

Variáveis podem ser locais ou globais. Uma variável (constante) é dita **local** a um módulo se ela é declarada naquele módulo. Por exemplo, a variável *x* da função *quadrado* é uma variável local a esta função. Uma variável é dita **global** a um módulo quando ela não está declarada

no módulo, mas pode ser referenciada a partir dele. Neste curso, consideraremos que variáveis são sempre locais, isto é, nenhum módulo poderá referenciar uma variável declarada em outro módulo.

18.7 Exercícios propostos

- Para se determinar o número de lâmpadas necessárias para cada cômodo de uma residência, existem normas que fornecem o mínimo de potência de iluminação exigida por metro quadrado (m^2) conforme a utilização deste cômodo. Suponha que só serão usadas lâmpadas de 60W. Seja a seguinte tabela:

Utilização	Classe	Potência/ m^2
quarto	1	15
sala de TV	1	15
salas	2	18
cozinha	2	18
varandas	2	18
escritório	3	20
banheiro	3	20

- Faça um módulo (função ou um procedimento) que recebe a classe de iluminação de um cômodo e suas duas dimensões (largura e comprimento) e devolve o número de lâmpadas necessárias para o cômodo.
 - Faça um algoritmo que leia um número indeterminado de informações, contendo cada uma o nome de um cômodo, sua classe de iluminação e as suas duas dimensões e, com base no módulo anterior, imprima a área de cada cômodo, sua potência de iluminação e o número total de lâmpadas necessárias. Além disso, seu algoritmo deve calcular o total de lâmpadas necessárias e a potência total para a residência.
- A comissão organizadora de um *rallye* automobilístico decidiu apurar os resultados da competição através de um processamento eletrônico. Um dos programas necessários para a classificação das equipes é o que emite uma listagem geral do desempenho das equipes, atribuindo pontos segundo determinadas normas.
 - Escreva um módulo (função ou procedimento) que calcula os pontos de cada equipe em cada uma das etapas do *rallye*, seguindo o seguinte critério. Seja Δ o valor absoluto da diferença entre o tempo-padrão e o tempo despendido pela equipe numa etapa (fornecidos como parâmetros):

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

- Faça um algoritmo que leia os tempos-padrão (em minutos) para as três etapas da competição e o número de equipes participantes. Para cada equipe, leia seu número de inscrição e os tempos (em minutos) despendidos para cumprir cada um das três etapas da competição, e, utilizando o módulo anterior, calcule: os pontos obtidos por cada equipe em cada etapa, a soma total de pontos por equipe e a equipe vencedora.
- Faça uma função *quantosdias* que recebe o dia, o mês e o ano de uma data e retorna um

valor que contém o número de dias do ano até a data fornecida. Em seguida, faça um algoritmo que recebe n pares de datas, onde cada par deve ser fornecido no formato $dia1$, $mês1$, $ano1$, $dia2$, $mês2$, $ano2$, verifica se as datas estão corretas e mostra a diferença, em dias, entre essas duas datas. Utilize a função *quantosdias*.

4. Escreva uma função que recebe dois números inteiros positivos e determina o produto dos mesmos, utilizando o seguinte método de multiplicação:

- (i) dividir, sucessivamente, o primeiro número por 2, até obter 1 como quociente;
- (ii) paralelamente, dobrar, sucessivamente, o segundo número;
- (iii) somar os números da segunda coluna que tenham um número ímpar na primeira coluna. O total obtido é o produto procurado. Exemplo para 9×6 :

$$\begin{array}{r}
 9 \quad 6 \quad \rightarrow \quad 6 \\
 4 \quad 12 \\
 2 \quad 24 \\
 1 \quad 48 \quad \rightarrow \quad 48 \\
 \hline
 \qquad \qquad \qquad 54
 \end{array}$$

Em seguida, escreva um programa que leia n pares de números e calcule os respectivos produtos, utilizando a função anterior.

5. Um número a é dito ser *permutação* de um número b se os dígitos de a formam uma permutação dos dígitos de b . Exemplo:

5412434 é uma permutação de 4321445, mas não é uma permutação de 4312455.

Observação: considere que o dígito 0 (zero) não aparece nos números.

- a) Faça uma função *contadígitos* que, dados um inteiro n e um inteiro d , $0 < d \leq 9$, devolve quantas vezes o dígito d aparece em n ;
 - b) Utilizando a função do item anterior, faça um algoritmo que leia dois números a e b e responda se a é permutação de b .
6. Um número b é dito ser *sufixo* de um número a se o número formado pelos últimos dígitos de a são iguais a b . Exemplo:

$$\begin{array}{r}
 \begin{array}{cc} a & b \\ \hline
 567890 & 890 \quad \rightarrow \quad \text{sufixo} \\
 1234 & 1234 \quad \rightarrow \quad \text{sufixo} \\
 2457 & 245 \quad \rightarrow \quad \text{não é sufixo} \\
 457 & 2457 \quad \rightarrow \quad \text{não é sufixo}
 \end{array}
 \end{array}$$

- a) Construa uma função *sufixo* que dados dois números inteiros a e b verifica se b é um sufixo de a .
- b) Utilizando a função do item anterior, escreva um algoritmo que leia dois números inteiros a e b e verifica se o menor deles é subsequência do outro. Exemplo:

$$\begin{array}{r}
 \begin{array}{ccc} a & b & \\ \hline
 567890 & 678 & \rightarrow \quad b \text{ é subsequência de } a \\
 1234 & 2212345 & \rightarrow \quad a \text{ é subsequência de } b \\
 235 & 236 & \rightarrow \quad \text{Um não é subsequência do outro}
 \end{array}
 \end{array}$$

7. Escreva uma função que dados um vetor real A com n elementos e um vetor B com m elementos, ambos representando conjuntos (estes vetores devem ser representados por variáveis globais no Portugol do VisuAlg), verifica se A está contido em B ($A \subset B$). Em seguida, utilizando a função anterior, verifique se dois conjuntos são iguais ($A = B$ se e somente se $A \subset B$ e $B \subset A$).
8. Escreva uma função que troca o conteúdo de duas variáveis. Em seguida, escreva uma função que recebe dois inteiros, i e j , e, utilizando o acesso a uma matriz real $A_{m \times n}$ representada como uma variável global, troca a linha i pela linha j .
9. Dizemos que uma matriz $A_{n \times n}$ é um *quadrado latino* de ordem n se em cada linha e em cada coluna aparecem todos os inteiros $1, 2, 3, \dots, n$ (ou seja, cada linha e coluna é permutação dos inteiros $1, 2, \dots, n$). Exemplo:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 4 & 1 & 2 & 3 \\ 3 & 4 & 1 & 2 \end{pmatrix}$$

A matriz acima é um quadrado latino de ordem 4. Considerando que A é o nome de uma variável global que representa uma matriz inteira $n \times n$, pede-se:

- a) Escreva uma função que recebe um índice i e verifica se na linha i de A , ocorrem todos os inteiros de 1 a n .
 - b) Escreva uma função que recebe um índice j e verifica se na coluna j de A , ocorrem todos os inteiros de 1 a n .
 - c) Utilizando as funções acima, verifique se uma dada matriz inteira $A_{n \times n}$ é um quadrado latino de ordem n .
10. Um conjunto pode ser representado por um vetor da seguinte forma: $V[0]$ é o tamanho do conjunto; $V[1], V[2], \dots$ são os elementos do conjunto (sem repetições).
 - a) Faça uma função *intersecção* que dados dois conjuntos de números inteiros A e B , constrói um terceiro conjunto C que é a intersecção de A e B . Lembre-se de que em $C[0]$ a sua função deve colocar o tamanho da intersecção.
 - b) Faça um algoritmo que leia um inteiro $n \geq 2$ e uma seqüência de n conjuntos de números inteiros (cada um com no máximo 100 elementos) e construa e imprima o vetor INTER que representa a intersecção dos n conjuntos.

Não é preciso ler todos os conjuntos de uma só vez. Você pode ler os dois primeiros conjuntos e calcular a primeira intersecção. Depois, leia o próximo conjunto e calcule uma nova intersecção entre esse conjunto lido e o conjunto da intersecção anterior, e assim por diante.

Bibliografia

O texto deste capítulo foi elaborado a partir dos livros abaixo relacionados:

1. Farrer, H. et. al. *Algoritmos Estruturados*. Editora Guanabara, 1989.
2. Shackelford, R.L. *Introduction to Computing and Listings*. Addison-Wesley Longman, Inc,

1998.

```

1 algoritmo "Calcula_Quadrado"
2
3 // módulo para cálculo do quadrado de um número
4 funcao quadrado (r : real) : real
5 var
6 // declaração de variáveis
7 x : real
8 inicio
9 // calcula o quadrado
10 x <- r * r
11 // passa para o algoritmo chamador o valor obtido
12 retorne x
13 fimfuncao
14
15 // módulo para ler um número
16 procedimento ler_numero (var val : real)
17 inicio
18 // solicita um número ao usuário
19 escreva ("Entre com um número: ")
20 leia (val)
21 fimprocedimento
22
23 // declaração de constantes e variáveis
24 var
25 num, x : real
26
27 inicio
28
29 // lê um número
30 ler_numero(num)
31
32 // calcula o quadrado
33 x <- quadrado(num)
34
35 // escreve o quadrado
36 escreva ("O quadrado do número é: ", x)
37 fimalgoritmo

```

Algoritmo 18.1: Cálculo do Quadrado de um Número.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] S. Gandz. The origin of the term "algebra". *The American Mathematical Monthly*, 33(9):437–440, 1926. [1.1](#)
- [2] D. J. Struik. *A Concise History of Mathematics*. Dover Publications, Inc., fourth edition, 1987. [1.1](#)