

The image shows a close-up of a laptop screen displaying HTML code. The code includes elements like `<input type="text">`, `<input type="submit" value="Enviar" />`, and `<input type="button" value="Cancelar" />`. There are also conditional comments like `<!--#include file="header.html" -->`. A blue semi-transparent overlay is present over the screen and keyboard. In the foreground, a person's hands are visible typing on the laptop keyboard. The background is a solid blue color.

Algoritmos e Lógica de Programação

Algoritmos e lógicas de programação

Marcio Aparecido Artero
Vanessa Cadan Scheffer

© 2018 por Editora e Distribuidora Educacional S.A.
Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

Presidente

Rodrigo Galindo

Vice-Presidente Acadêmico de Graduação e de Educação Básica

Mário Ghio Júnior

Conselho Acadêmico

Ana Lucia Jankovic Barduchi

Camila Cardoso Rotella

Danielly Nunes Andrade Noé

Grasiele Aparecida Lourenço

Isabel Cristina Chagas Barbin

Lidiane Cristina Vivaldini Olo

Thatiane Cristina dos Santos de Carvalho Ribeiro

Revisão Técnica

Ruy Flávio de Oliveira

Editorial

Camila Cardoso Rotella (Diretora)

Lidiane Cristina Vivaldini Olo (Gerente)

Elmir Carvalho da Silva (Coordenador)

Leticia Bento Pieroni (Coordenadora)

Renata Jéssica Galdino (Coordenadora)

Dados Internacionais de Catalogação na Publicação (CIP)

A786a Artero, Marcio Aparecido
Algoritmos e lógicas de programação / Marcio Aparecido
Artero, Vanessa Cadan Scheffer. – Londrina : Editora e
Distribuidora Educacional S.A., 2018.
216 p.

ISBN 978-85-522-0708-5

1. Algoritmos. 2. Programação. I. Artero, Marcio
Aparecido. II. Scheffer, Vanessa Cadan. III. Título.

CDD 600

Thamiris Mantovani CRB-8/9491

2018
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 – Londrina – PR
e-mail: editora.educacional@kroton.com.br
Homepage: <http://www.kroton.com.br/>

Sumário

Unidade 1 Lógica de programação	7
Seção 1.1 - Definições de lógica	9
Seção 1.2 - Elementos fundamentais de programação	24
Seção 1.3 - Representações de algoritmos	39
Unidade 2 Elementos de algoritmos	55
Seção 2.1 - Execução sequencial e estruturas de decisão	57
Seção 2.2 - Estruturas de repetição	73
Seção 2.3 - Estrutura de dados	92
Unidade 3 Conceitos de programação	113
Seção 3.1 - Introdução à linguagem C	115
Seção 3.2 - Estruturas condicionais em linguagem C	133
Seção 3.3 - Estruturas de repetição em linguagem C	151
Unidade 4 Aplicações de programação	169
Seção 4.1 - Programação e funções com vetores	171
Seção 4.2 - Programação e funções com matrizes	188
Seção 4.3 - Recursividade	201

Palavras do autor

Caro aluno!

Seja bem-vindo à disciplina de algoritmos e lógica de programação.

É provável que você já tenha se deparado com uma linda construção, um edifício fabuloso que mais parece uma obra de arte, certo? Veja o caso do MASP (Museu de Arte de São Paulo), uma obra da arquiteta Lina Bo Bardi. Essa obra encanta pelo planejamento e audácia em que foram baseados os cálculos minuciosos. Tal obra está suspensa por quatro colunas, ligadas por uma viga que percorre todo o prédio, tem como destaque um vão de luz impressionante e um pé-direito de oito metros.

Enfim, o que essa grande obra tem a ver com algoritmos e lógica de programação? Em ambos os casos (obras sofisticadas e programação), são necessários planejamentos e um pensamento lógico para a sua construção. Assim como o projeto do MASP teve uma planta com todos os seus cálculos para que chegasse ao resultado final, um programa de computador precisa ser pensado e estruturado para alcançar o seu objetivo, usando como instrumento a lógica de programação por meio de algoritmos.

Para o sucesso da compreensão deste material, vamos dividi-lo em quatro unidades, cada uma com objetivos específicos e competências que resultam em aprender e assimilar a lógica, os algoritmos e os elementos de programação.

Na primeira unidade, você irá compreender e será capaz de realizar a lógica de programação. Lógica essa que desenvolverá em você, aluno, habilidades de encontrar soluções para os mais diversos problemas. Ainda nesta unidade, você estudará os conceitos de algoritmos e suas características.

Na segunda unidade, você irá analisar e aplicar os elementos de algoritmos; tal aprendizado o conduzirá a resolver problemas de acordo com as características da programação.

Na unidade três, você irá compreender e aplicar os conceitos de programação utilizando a linguagem C, para isso, será preciso trabalhar com a estrutura de condicionais e de repetição.

E encerrando, a unidade quatro tem como objetivo aplicar a programação utilizando: vetores, matrizes, funções e recursividade. A sua dedicação e seu potencial serão fatores decisivos para criar programas que possam solucionar diversos problemas do seu dia a dia.

É isso! Tudo muito fascinante!

Bons estudos e boa sorte!

Lógica de programação

Convite ao estudo

Caro aluno, iniciamos essa unidade apresentando a você as definições de lógica, os elementos fundamentais para programação e as representações dos algoritmos. Quando falamos em lógica, vários pensamentos nos vêm à mente, afinal de contas, o nosso dia é rodeado de lógica e procedimentos algorítmicos. Você deve estar se perguntando: como assim?

Imagine você no início do seu dia: você precisa acordar para dar início a suas tarefas, certo? Isso é uma lógica seguida de uma condição. Após estar certo de que acordou e de que o dia iniciou, o prosseguimento agora é uma sequência de algoritmos, quando para cada tarefa haverá uma rotina. Por exemplo: você irá se levantar, colocar o chinelo, ir ao banheiro, trocar de roupa, tomar café e se preparar para dar início às tarefas do dia. Percebe que um algoritmo é um conjunto de instruções que devemos fazer para realizar uma ação? Pois então, tudo isso é muito fascinante e você irá aplicar essas definições na preparação e programação de computadores.

Muito bem, a partir desta unidade você irá compreender e ser capaz de realizar a lógica de programação no seu dia a dia.

Para tanto, vamos ao nosso contexto de aprendizagem com a Kro_Engenharias, uma empresa de renome na área de engenharia civil, famosa por sua ampla área de atuação, que desenvolve desde um simples projeto até os empreendimentos mais arrojados. Os engenheiros da empresa realizam diversos cálculos matemáticos e, na maioria das vezes, fazem uso de calculadoras e planilhas eletrônicas.

Você, portador de um pensamento computacional, foi contratado para fazer parte deste time de engenheiros, porém, uma missão foi dada pelo seu gestor direto: criar um pensamento lógico para que os engenheiros futuramente

possam programar os seus diversos cálculos matemáticos.

E missão dada é missão que deverá ser cumprida!

No primeiro momento, você deverá trabalhar com seus colegas engenheiros o uso da lógica para solução de problemas, a aplicação da lógica no dia a dia e o raciocínio lógico.

Em seguida, você irá apresentar as definições e a importância dos algoritmos, assim como a utilização de variáveis, constantes e seus operadores.

Para finalizar a missão, você deixará de forma clara como funciona a linguagem natural, as características dos fluxogramas, pseudocódigos e aplicar atividades práticas para soluções de problemas de lógica.

Pois bem, atente-se aos conceitos que serão apresentados e seja sempre explorador de conhecimentos.

Boa sorte e ótimos estudos!

Seção 1.1

Definições de lógica

Diálogo aberto

Caro aluno,

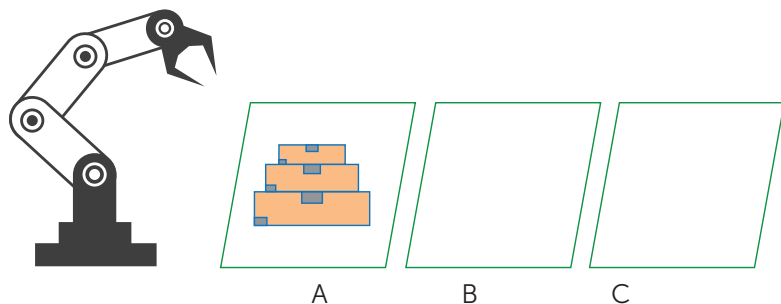
A situação-problema desta seção tem por objetivo trabalhar o raciocínio lógico desenvolvido durante o desenvolvimento do “Não pode faltar”. A Kro_Engenharias, uma empresa de renome na área da engenharia, acabou de contratá-lo, e tão breve já lhe foi dada uma missão: ajudar os seus colegas engenheiros a encontrar uma forma que possa ajudá-los na execução dos cálculos matemáticos desenvolvidos nas mais diversas áreas da engenharia. Para isso, você deverá desenvolver com os engenheiros o pensamento lógico para o desenvolvimento de *softwares* para soluções dos cálculos.

Como ponto de partida, você deverá trabalhar o pensamento lógico dos colegas engenheiros. Para tal, terá de ajudá-los a resolver o problema encontrado em um dos setores da empresa: um robô tem a função de empilhar materiais com as seguintes características:

- Os materiais mais pesados não podem ficar sobre os materiais mais leves.
- As pilhas (P1, P2 e P3) devem ter no máximo 3 fileiras de materiais.
- O galpão dispõe apenas de 3 locais (A, B e C) de movimentações dos materiais, onde devem transitar e ficar dispostos no último local (C) para o próximo processo de logística da empresa.

A proposta, então, é criar uma maneira de otimizar o trabalho do robô para que ele realize a tarefa utilizando a menor quantidade possível de passos e, conseqüentemente, um menor tempo na execução do processo.

Figura 1.1 | Robótico vetor máquina



Fonte: Freepik. Disponível em: <ahref='http://br.freepik.com/vetores-gratis/robotico-vetor-maquina_714906.htm'>Designed byFreepik>. Acesso em: 15 out. 2017.

Agora chegou o momento de conhecer as definições e aplicações da lógica para a solução deste problema e de outros que irão surgir. Preparado? Vamos lá!

Não pode faltar

Caro aluno, já imaginou que para resolver um problema há inúmeras possibilidades, que para chegar a um único lugar existem diversos caminhos? Então, pensar com lógica é ter uma ordem de raciocínio, criar critérios para chegar ao seu objetivo no menor tempo possível e com o menor esforço.

Nesta seção, vamos nos concentrar em entender os pensamentos lógicos e definições que os permeiam.

Pois bem, primeiramente vamos definir o que é lógica: “A lógica é a arte de pensar corretamente ou a lógica é um estudo dos modos corretos do pensamento” (SOARES, 2014, p. 1). Segundo Forbellone (2005, p. 1), “podemos relacionar a lógica com a correção do pensamento, pois uma de suas preocupações é determinar quais operações são válidas e quais não são, fazendo análises das formas e leis do pensamento”. Temos, ainda, que “lógica é a ciência que estuda as leis e os critérios de validade que regem o pensamento e a demonstração, ou seja, ciência dos princípios formais do raciocínio”. (ABE; SCALZITTI; SOUZA FILHO, 2001, p. 11)

Sempre que você pensa de forma ordenada e dentro da razão, está pensando de forma lógica, nem sempre você terá o mesmo pensamento que os outros, porém, o objetivo deste pensamento na sua maioria é alcançar um objetivo.

Figura 1.2 | Pensamento lógico



Fonte: <<https://pixabay.com/pt/pensador-em-uma-perda-considere-1294491/>>. Acesso em: 30 set. 2017.

Quando você pensa em **porcentagem**, vem logo no pensamento: "alguma coisa por cem" ou " $\frac{x}{100}$ " (em que "x" é um número qualquer dividido por cem). Certo? Esse raciocínio é lógico, pelo fato de que você deseja encontrar um número que será usado para resolver outro pensamento lógico, por exemplo:

Você ganha R\$ 5.000,00 por mês e terá um desconto de 11% de INSS sobre esse valor. Como deverá estruturar esse problema? Pela lógica, evidente!

Uma das soluções é resolver primeiro quanto é 11%:

$$\frac{11}{100} = 0,11$$

Em seguida, multiplicar o seu salário pelo valor encontrado:

$$5.000,00 \cdot 0,11 = 550,00$$

Valor encontrado, certo?

Agora que você encontrou o valor da porcentagem, você poderá aplicar para somar ou subtrair no seu salário. A lógica definida era encontrar o desconto de INSS, então você deverá efetuar a subtração do seu salário pelo valor da porcentagem encontrada. Veja como ficou:

$$\text{R\$ } 5.000,00 - \text{R\$ } 550,00 = \text{R\$ } 4.450,00$$

Finalizando o nosso pensamento lógico, o seu salário será de R\$ 4.450,00.

Tudo muito lógico!



Segundo Forbellone (2005), o ser humano tem a capacidade de se expressar pela escrita ou pela fala, e lógico, se baseia em um determinado idioma e gramática. Pensando assim, seja qual for o idioma o raciocínio, seguirá a mesma linha de pensamento.

Podemos dizer que isso acontece com a lógica de programação quando usamos o mesmo raciocínio para programar inúmeras linguagens de programação.

A lógica pode ser utilizada para vários outros pensamentos, veja só:

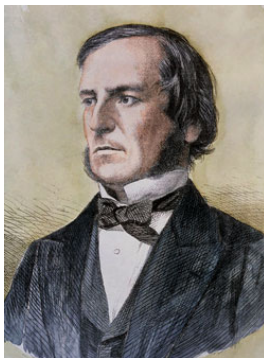
- Todo mamífero é um animal.
- Todo cachorro é um mamífero.
- Portanto, todo cachorro é um animal.

Veja que nesse exemplo específico ambos os pensamentos são válidos.

A lógica de George Boole

Segundo Alves (2014), George Boole foi um matemático e filósofo que, no século XIX, defendeu a ideia de que o raciocínio humano poderia ser expresso em termos matemáticos, por meio da lógica formal desenvolvida pelos gregos, mais precisamente pelo filósofo Aristóteles.

Figura 1.3 | George Boole (1815-1864)



Fonte: <https://upload.wikimedia.org/wikipedia/commons/c/ce/George_Boole_color.jpg>. Acesso em: 30 set. 2017.

Por meio desse raciocínio originou-se a Álgebra de Boole ou Álgebra Booleana. É sabido que esse tipo de álgebra é embasado na lógica binária. Como assim? A lógica binária possui duas representatividades, "falso" e "verdadeiro" ou "0" e "1".

Em relação aos seus operadores, são definidos AND, OR e NOT, ou seja, E, OU e NÃO, onde ("E") é a conjunção, ("OU"), a disjunção e (NÃO), a negação.

Explicando melhor, veremos que:

- Conjunção ("E") – somente se as duas representatividades forem verdadeiras, a resposta será verdadeira.

- Disjunção ("OU") – Se pelo menos uma de suas representatividades for verdadeira, a resposta será verdadeira.

- Negação (NÃO) – Quando uma representatividade for verdadeira, a resposta será falsa, e quando uma representatividade for falsa, a resposta será verdadeira.

Para entender melhor esta lógica vamos analisar a Tabela 1.1:

Tabela 1.1 | Tabela Verdade – "verdadeiro" ou "falso"

A	B	A ("E") B	A ("OU") B	NÃO A	NÃO B
Verdade	Verdade	Verdade	Verdade	Falso	Falso
Verdade	Falso	Falso	Verdade	Falso	Verdade
Falso	Verdade	Falso	Verdade	Verdade	Falso
Falso	Falso	Falso	Falso	Verdade	Verdade

Fonte: elaborada pelo autor.

Agora vamos aplicar a tabela verdade usando números binários, conforme mostra a Tabela 1.2:

Tabela 1.2 | Tabela verdade – "0" ou "1"

A	B	A ("E") B	A ("OU") B	NÃO A	NÃO B
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	0
0	0	0	0	1	1

Fonte: elaborada pelo autor.

Veremos agora algumas aplicações, como ilustra a Tabela 1.3:

Tabela 1.3 | Aplicação da tabela verdade

Função	Operador	Exemplo	Resultado
Negação	NÃO	NÃO (6>3)	Falso
Conjunção	E	(5>1) E (7>8)	Falso
Disjunção	OU	(8>9) OU (9>8)	Verdadeiro

Fonte: elaborada pelo autor.

Vamos pensar na construção de triângulo, e para tal construção não podemos usar qualquer medida, ou seja, você deve obedecer às seguintes condições:

- Na construção de um triângulo é obrigatório que a medida de qualquer um dos lados seja menor que a soma das medidas dos outros dois lados e maior que o valor absoluto da diferença entre essas medidas.

O teste lógico ficaria:

$$(a < b + c) \text{ E } (b < a + c) \text{ E } (c < a + b)$$

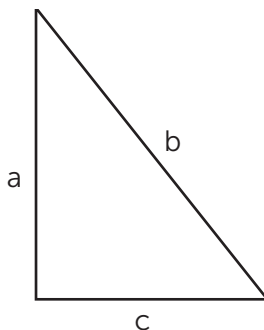
Alternativamente, o teste poderia ser:

$$(a > |b - c|) \text{ E } (b > |a - c|) \text{ E } (c > |a - b|)$$

Obs.: Foram utilizados módulos ("|...|") para que a subtração dos lados não seja um valor negativo.

Análise o triângulo da Figura 1.4:

Figura 1.4 | Triângulo retângulo



Fonte: elaborada pelo autor.

Onde:

$$a = 8 \quad b = 10 \quad c = 5$$

$$(a < b + c) \text{ E } (b < a + c) \text{ E } (c < a + b)$$

$(a < b + c)$ $(8 < 10 + 5)$ $(8 < 15)$	E	$(b < a + c)$ $(10 < 8 + 5)$ $(10 < 13)$	E	$(c < a + b)$ $(5 < 8 + 10)$ $(5 < 18)$
Verdade		Verdade		Verdade

Alternativamente:

$$(a > |b - c|) \text{ E } (b > |a - c|) \text{ E } (c > |a - b|)$$

$(a > b - c)$ $(8 > 10 - 5)$ $(8 > 5)$	E	$(b > a - c)$ $(10 > 8 - 5)$ $(8 > 3)$	E	$(c > a - b)$ $(5 > 8 - 10)$ $(8 > 2)$
Verdade		Verdade		Verdade

Analisando os resultados, você pode perceber que todas as representatividades são verdadeiras e por este motivo os valores atribuídos formam um triângulo.



Refleta

Segundo Forbellone (2005), a lógica por meio da conjunção é representada pela letra "E" e somente pode assumir um valor verdadeiro se todos os casos relacionados forem verdadeiros. Por exemplo: Se chover e relampejar, você fica em casa. Quando você fica em casa? Em qualquer uma das situações você vai ficar em casa, pois em ambas as situações a conjunção "E" implica que são verdadeiras.

E se mudamos a frase: Se chover ou relampejar, você fica em casa. Quando você fica em casa?

Que ótimo! Agora que você já conhece o relacionamento com a lógica de Boole, vamos aprofundar um pouco mais e conhecer a Linguagem proposicional.

Linguagem proposicional

Ao apresentarmos uma linguagem formal, precisamos inicialmente fornecer os componentes básicos da linguagem, chamados de alfabeto, para em seguida fornecer as regras de formação da linguagem, também chamadas de gramática. (SILVA, 2006, p. 8)



Segundo Silva (2006), usamos os seguintes elementos que representam os conectivos:

- O conectivo unário \leftarrow (negação, lê-se: NÃO).
- O conectivo binário \wedge (conjunção, lê-se: E).
- O conectivo binário \vee (disjunção, lê-se: OU).
- O conectivo binário \rightarrow (implicação, lê-se: SE...ENTÃO).
- Os elementos de pontuação são representados pelos parênteses.

Veja alguns exemplos de aplicações de fórmulas na Tabela 1.5:

Tabela 1.5 | Proposições

Tom é um gato	E	Tom caça ratos
p	\wedge	q
Jerry é um rato	ou	Jerry foge de Tom
p	\vee	q
O mar está calmo	então	vou navegar
p	\rightarrow	q
vai chover	não	vai chover
p	\leftarrow	q

Fonte: elaborada pelo autor.

Figura 1.5 | Matrizes de conectivos lógicos

$A \wedge B$	$B = 0$	$B = 1$	$A \vee B$	$B = 0$	$B = 1$
$A = 0$	0	0	$A = 0$	0	1
$A = 1$	0	1	$A = 1$	1	1
$A \rightarrow B$	$B = 0$	$B = 1$	$\leftarrow A$		
$A = 0$	1	1	$A = 0$	1	
$A = 1$	0	1	$A = 1$	0	

Fonte: Silva (2006, p. 15).

Importante:

Em nossas aplicações com proposições vamos usar em letras minúsculas p, q, r e s para compor as fórmulas da linguagem proposicional, e A, B, C e D serão a representação das fórmulas. Exemplo: $(q \rightarrow \neg q)$, neste formato podemos dizer que $(A \rightarrow B)$, ou seja, $A = q$ e $B = \neg q$.

Veja agora as precedências para resolver as proposições:

Tabela 1.6 | Precedências

Operadores	Precedências
\leftarrow	1ª
\wedge	2ª
\vee	2ª
\rightarrow	3ª

Fonte: elaborada pelo autor.

Explicando: Primeiramente será resolvida a "Negação", em segundo a "Conjunção" e "Disjunção" (que têm o mesmo nível de precedência) e por último a "Implicação".

Quando tem ambiguidade, que é o caso da conjunção e disjunção, será analisado o que estiver mais à direita, ou seja, será realizada uma associatividade à direita.

Analisando a proposição abaixo:

$$r \wedge \leftarrow s \rightarrow r \vee \leftarrow s \wedge \leftarrow r$$

Sendo que:

r = Verdade

s = Falso

1º: Isolamos as "negações" com parênteses

$$r \wedge (\leftarrow s) \rightarrow r \vee (\leftarrow s) \wedge (\leftarrow r)$$

2º: Isolamos as "Conjunções" e "Disjunções"

$$(r \wedge (\leftarrow s)) \rightarrow (r \vee ((\leftarrow s) \wedge (\leftarrow r)))$$

Resolvendo:

$$(r \wedge (\leftarrow s)) \rightarrow (r \vee ((\leftarrow s) \wedge (\leftarrow r)))$$

(verdade e (não falso)) implica (verdade ou ((não falso) e (não verdade)))

(verdade e verdade) implica (verdade ou ((verdade) e (falso)))

(verdade) implica (verdade ou (falso))

(verdade) implica (verdade)

Verdade

A resposta será **verdade**.



Pesquise mais

A videoaula a seguir demonstra de forma simples a lógica proposicional e a tabela verdade aplicada em vários exemplos e aplicações. Existem várias outras videoaulas relacionadas a esse assunto, fique à vontade para explorar e adquirir conhecimento.

Raciocínio lógico (Proposições e tabela verdade aula 3). Disponível em: <<https://www.youtube.com/watch?v=rjr-RqFM3uc>>. Acesso em: 1 out. 2017.

Pois bem, veja a seguinte aplicação usando a proposição:

$$(r \wedge s)$$

Vamos fazer uma representação da fórmula acima considerando as afirmações abaixo:

“Curitiba é capital do Paraná” é representado por “ r ”, o seu valor lógico é: $\text{Val}(r)=\text{Verdade}$

“São Paulo é maior que Curitiba” é representado por “ s ”, o seu valor lógico é: $\text{Val}(s)=\text{Verdade}$. Então:

$$\underbrace{\text{“Curitiba é capital do Paraná”}}_r \wedge \underbrace{\text{“São Paulo é maior que Curitiba”}}_s$$

Podemos concluir que:

$$(r \wedge s) \text{ equivale a } (A \wedge B)$$

A primeira fórmula (r) é verdadeira e a segunda fórmula (s) é verdadeira, portanto, a resposta final é **verdadeira**.



Veja agora alguns exemplos de proposições e não proposições:

Ela é professora.

Neste caso em específico não retrata uma proposição, pois não conseguimos atribuir nenhum valor lógico para esse tipo de declaração.

10>20

É uma proposição, pois o valor lógico é considerado falso.

Márcio é professor.

É uma proposição, pois deixa claro qual é a profissão do Márcio, também caracterizado por classificar se é verdadeiro ou falso.

Hoje vai chover?

Não caracteriza proposição. Frases interrogativas não remetem a verdadeiro ou falso.

Segundo Silva (2006), na definição de fórmula, o uso de parênteses é obrigatório quando se usam os conectivos binários, porém, na prática podemos usar abreviações que podem substituir os parênteses, veja algumas situações:

- Os parênteses que ficam mais externos de uma fórmula podem ser omitidos. Por exemplo: podemos escrever $p \wedge q \wedge \neg r \wedge \neg s$ em vez de $((p \wedge q) \wedge \neg r) \wedge \neg s$; os parênteses estão alinhados à esquerda.

Quando tem repetição do conectivo \rightarrow , podemos não usar os parênteses. Exemplo: podemos escrever $p \rightarrow q \rightarrow r$ em vez de $p \rightarrow (q \rightarrow r)$.

Na dúvida, se a fórmula ficar clara, deixe os parênteses.

Muito bem, isso é só o começo do nosso aprendizado. Agora que você já conhece o raciocínio lógico, é hora de pensar em praticar. Boa sorte e até a próxima seção!

Sem medo de errar

Chegou o momento de trazer à tona a situação-problema da seção. Deram-lhe a missão de ajudar os seus colegas engenheiros com algumas situações vivenciadas no dia a dia da empresa. Para isso, você deverá desenvolver com os engenheiros o pensamento lógico para a elaboração de algoritmos.

A situação-problema remete à utilização de um robô para organizar os materiais recebidos e distribuí-los.

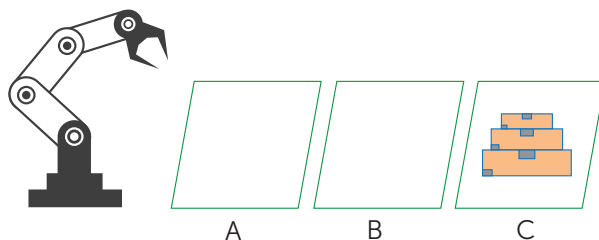
Enfim, o robô deverá empilhar os materiais recebidos, passando por três áreas específicas (A, B e C), em que A é o local onde os materiais encontram-se empilhados e C é o local onde deverão estar empilhados no final do processo.

Lembrando que os materiais mais pesados não podem ficar sobre os materiais mais leves, e que as pilhas (P1, P2 e P3) devem ter no máximo 3 fileiras de materiais.

A proposta então é criar uma maneira de otimizar o trabalho do robô para que ele realize a tarefa utilizando a menor quantidade possível de passo se, conseqüentemente, um menor tempo na execução do processo.

Muito bem, para solucionar esse problema você vai utilizar a mesma lógica aplicada no exemplo clássico da torre de Hanói.

Figura 1.6 | Robótico vetor máquina



Fonte: Freepik. Disponível em: <ahref='http://br.freepik.com/vetores-gratis/robotico-vetor-maquina_714906.htm'>Designed byFreepik. Acesso em: 15 out. 2017.

Vamos lá,

Primeiro, mover a primeira pilha (P1) do seu espaço original (A) para o terceiro espaço (C).

Segundo, mover a segunda pilha (P2) do seu espaço original (A) para o segundo espaço (B).

Terceiro, mover a primeira pilha (P1) do terceiro espaço (C) para o segundo espaço (B).

Quarto, mover a terceira pilha (P3) do seu espaço original (A) para o terceiro espaço (C).

Quinto, mover a primeira pilha (P1) do segundo espaço (B) para o espaço original (A).

Sexto, mover a segunda pilha (P2) do segundo espaço (B) para o terceiro espaço (C).

Sétimo, mover a primeira pilha (P1) do espaço original (A) para o terceiro espaço (C).

Muito bem, pratique mais, mude as ordens e se possível coloque mais pilhas.

Avançando na prática

A casa perfeita

Descrição da situação-problema

Você está à procura de uma casa nova para comprar e resolveu contratar um corretor de imóveis para lhe ajudar com essa missão, para tanto, você lhe deu algumas dicas do que seria sua casa ideal: perto do centro e com três quartos ou mais.

Seu corretor lhe trouxe duas propostas: uma casa afastada do centro, no entanto, com três quartos; e uma segunda proposta, uma casa no centro e com dois quartos.

Agora que você já tem as condições necessárias, elabore as sentenças usando proposições.

Resolução da situação-problema

Temos a seguinte situação:

p: É uma casa.

q: Está localizada no centro.

r: Precisa ter três quartos.

s: A compra é ideal.

A equação da fórmula conforme a nossa situação é:

Suas condições: $(p \wedge q \wedge r) \rightarrow s$

Condições colocadas pelo corretor:

$$(p \wedge \neg q \wedge r) \vee (p \wedge q \wedge \neg r)$$

Podemos concluir que a proposta do corretor **não é a compra ideal**.

Faça valer a pena

1. A Álgebra de Boole ou Álgebra Booleana é embasada na lógica binária, portanto, possui duas representatividades, “falso” e “verdadeiro” ou “0” e “1”. Em relação aos seus operadores são definidos AND, OR e NOT, ou seja, E, OU e NÃO, onde (“E”) é a conjunção, (“OU”), a disjunção e (NÃO), a negação.

Podemos dizer que:

- I. Conjunção (“E”) – Se pelo menos uma de suas representatividades for verdadeira, a resposta será verdadeira.
- II. Disjunção (“OU”) – Somente se as duas representatividades forem verdadeiras, a resposta será verdadeira.
- III. Negação (NÃO) – Quando uma representatividade for verdadeira, a resposta será falsa e quando uma representatividade for falsa, a resposta será verdadeira.

Assinale a alternativa correta conforme as afirmações acima:

- a) Somente a afirmação I está correta.
- b) As afirmações I, II e III estão corretas.
- c) As afirmações II e III estão corretas.
- d) Somente a afirmação III está correta.
- e) As afirmações I e III estão corretas.

2. Segundo Forbellone (2005), o ser humano tem a capacidade de se expressar pela escrita ou pela fala e, lógico, se baseia em um determinado idioma e gramática. Pensando assim, seja qual for o idioma, o raciocínio seguirá a mesma linha de pensamento.

Seguindo essa linha de raciocínio, analise as sentenças abaixo:

- I. Marcio é professor.
- II. $10 - 5$
- III. $Y > 10$

Assinale a alternativa correta que corresponde a uma proposição:

- a) Somente a afirmação I está correta.
- b) As afirmações I, II e III estão corretas.
- c) As afirmações II e III estão corretas.
- d) Somente a afirmação III está correta.
- e) As afirmações I e III estão corretas.

3. “Ao apresentarmos uma linguagem formal, precisamos inicialmente fornecer os componentes básicos da linguagem, chamados de alfabeto, para em seguida fornecer as regras de formação da linguagem, também chamadas de gramática”. (SILVA, 2006, p. 8)

Faça a leitura das afirmações abaixo e as represente de forma reduzida.

“Márcio é professor e Karina é empresária”.

Assinale a alternativa correta:

- a) $p \wedge q$
- b) $p \vee q$
- c) $\leftarrow p \vee q$
- d) $p \vee \leftarrow q$
- e) $p \vee q \wedge p$

Seção 1.2

Elementos fundamentais de programação

Diálogo aberto

Caro aluno, acredito que você já se considerou um Masterchef na cozinha, utilizando-se de receitas ousadas para criação de vários tipos de pratos. Podemos dizer que as receitas possuem os mais variados ingredientes e modos de preparo para sua elaboração. Então, observe: as receitas, com seus ingredientes e modos de preparo, podem ser consideradas um tipo de algoritmo, o objeto dessa aula. Em outras palavras: não só você, candidato a Masterchef, conhece os algoritmos, mas também sabe como eles funcionam! Pensando nesse exemplo, você vai conhecer e compreender nesta seção os conceitos e aplicações de variáveis, constantes e operadores, para a realização dos mais diferentes tipos de algoritmos.

Lembre-se que na primeira seção desta unidade trabalhamos a lógica no seu contexto de raciocínio, agora chegou o momento de você começar a estruturar esse pensamento em forma de algoritmos, certo?

Pois bem, a empresa Kro_Engenharias é uma empresa de renome na área de engenharia, famosa por sua ampla área de atuação, desenvolvendo desde um simples projeto até os empreendimentos mais arrojados.

Lembre-se que os engenheiros da empresa realizam diversos cálculos matemáticos e, na maioria das vezes, fazem uso de calculadoras e planilhas eletrônicas. Diante dessa situação, será necessário ajudá-los com o desenvolvimento e planejamento dos algoritmos para realizar os cálculos matemáticos utilizados na engenharia.

O seu desafio é colocar de forma simples uma das grandes fórmulas utilizadas pelos engenheiros, a Fórmula de Bhaskara

$\left(\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \right)$, para cálculo das raízes de uma equação de

segundo grau. Nesse contexto você deverá usar variáveis, constantes e os tipos de operadores. Tente ser o mais didático possível em sua explicação, para que seja de fácil entendimento para os seus colegas engenheiros.

Boa sorte e ótimos estudos!

Não pode faltar

Olá! você deve estar ansioso para entender e começar a desenvolver algoritmos, certo? Também, não é para menos! Serão os algoritmos que o nortearão para as soluções dos mais diversos tipos de problemas. Assim como serão a base para criar os programas de computadores.

Definição de algoritmos

Segundo Szwarcfiter e Markenzon (1994), algoritmos são definidos como o processo sistemático para a resolução de um problema. Forbellone e Eberspacher (2000) defendem que algoritmo é a sequência de passos que visam atingir um objetivo bem definido. Por fim, Saliba (1993), Berg e Figueiró (1998) descrevem algoritmo como uma sequência ordenada de passos que deve ser seguida para a realização de uma tarefa.

Seguindo essas definições para algoritmos, podemos criar as mais diversas rotinas para inúmeras situações. Por exemplo: a compra de um carro novo:

1. Analisar a real necessidade.
2. Escolher a marca.
3. Escolher o modelo.
4. Escolher a motorização.
5. Escolher a cor.
6. Negociar a forma de pagamento.
7. Pagar.
8. Retirada do carro.

Pode-se, também, criar um algoritmo para os procedimentos de fritar um bife:

1. Pegar a frigideira, a carne, o tempero e o óleo.
2. Colocar o tempero na carne.
3. Acender o fogo.
4. Colocar a frigideira no fogo.
5. Colocar o óleo na frigideira.
6. Aguardar o aquecimento do óleo.
7. Colocar a carne na frigideira.
8. Escolher o ponto da carne.
9. Retirar a carne da frigideira quando esta atingir o ponto desejado.
10. Apagar o fogo.

Perceba que não existe somente uma forma de realizar um algoritmo, você pode criar outras formas e sequências para obter o mesmo resultado.

Para descrever o funcionamento de um algoritmo, você pode usar como base a representação de funcionamento do computador, conforme mostra a Figura 1.6.

Figura 1.6 | Representação do funcionamento do computador



Fonte: elaborada pelo autor.

De acordo com esse modelo, as entradas dos algoritmos são caracterizadas pelos elementos (dados) a serem fornecidos inicialmente; o processamento será definido pela execução das ações sobre os dados de entrada e sobre dados intermediários gerados durante esta execução, e a saída será a solução do problema ou o objetivo atingido.

Caro aluno, chegou o momento de entender as características das modificações e as suas aplicações no desenvolvimento de um algoritmo.

Variáveis

Pensar em variáveis é pensar em algo que pode sofrer variações, certo? E realmente é isso mesmo! Segundo Lopes e Garcia (2002), uma

variável é considerada um local que armazena um conteúdo específico na memória principal do computador. O nome vem do fato de que esse local pode conter valores diferentes, a “gosto do freguês”, isto é, do programador, do usuário do programa, ou do programa em si. Em outras palavras, os valores podem variar no local de armazenamento, por isso esses locais receberem o nome de “variáveis”.



Pesquise mais

No site a seguir você terá a oportunidade de reforçar o seu conhecimento sobre algoritmos. PEREIRA, Ana Paula. **O que é algoritmo**. 2009. Disponível em: <<https://www.tecmundo.com.br/programacao/2082-o-que-e-algoritmo-.html>>. Acesso em: 13 out. 2017.

Para fixar o aprendizado sobre variáveis e constantes assista o vídeo disponível em: <https://www.youtube.com/watch?v=vp4jgXA_BB0>. Acesso em : 13 out. 2017

Vamos então a nossa primeira analogia sobre variáveis?

Acredito que você tenha um armário e nele há várias gavetas. Para que serve uma gaveta? Uma gaveta no seu contexto geral pode guardar diversos tipos de objetos (roupas, livros, contas, toalhas, etc.), assim como as variáveis podem armazenar vários tipos de dados. Veja abaixo os tipos de variáveis que serão os objetos de nossos estudos:

Variáveis do tipo numérico

São variáveis que armazenam dados numéricos, como: a idade de uma pessoa, o preço de um produto, o salário de um funcionário, entre outros dados caracterizados pelos números.

Ainda falando do tipo numérico, podemos dividi-lo em duas classes:

- **Inteiro:** são caracterizados pelos números inteiros, positivos ou negativos.

Exemplo: (110), (90), (-80), (-2).

- **Real:** são caracterizados por números inteiros e decimais (fracionais), sendo positivos ou negativos.

Exemplo: (10,50), (-30,22), (20).

Variáveis do tipo literal

São variáveis que armazenam letras, números e símbolos especiais. São chamados de caracteres ou, em inglês, string. Por exemplo: (Marcio, Karina38, 2017).

Variáveis do tipo lógico

São variáveis que podem assumir um valor verdadeiro ou falso, por exemplo: O Carlos tem filho? Sim ou Não.

A Carol é engenheira? Verdadeiro ou Falso.

Variáveis do tipo constante

Segundo Manzano (2000), uma constante pode ser definida por tudo aquilo que é fixo ou estável, ou seja, você atribui um valor para aquela variável e ela não sofre alteração durante certo procedimento.

Podemos citar como exemplo o valor de $\text{Pi} = 3,14$, o percentual de desconto do INSS = 11%, entre outros.

Muito bem, agora que você já conhece os tipos de variáveis, vamos dar sequência conhecendo os tipos de operadores:

Operadores aritméticos

Segundo Forbellone e Eberspacher (2000), pode-se chamar de operadores aritméticos o conjunto de símbolos que representam as operações básicas da matemática. Veja na Tabela 1.7 os operadores com as suas aplicações:

Tabela 1.7 | Operadores e aplicações

Operadores	Função	Exemplos
+	Adição	$Y + x$, $7 + 8$
-	Subtração	$10 - 8$, nota - extra
*	Multiplicação	$5 * 5$, nota *4
/	Divisão	$100/5$, valor1/valor2
pot (x,y)	Potenciação	Pot (3,3) = 3 elevado a 3 (3^3)
rad(x)	Radiciação	rad(25) = raiz quadrada de $25(\sqrt{25})$
mod	Resto da divisão	$10 \text{ mod } 3$ resulta em 1
Div	Quociente da divisão	$15 \text{ div } 5$ resulta em 3

Fonte: adaptada de: Forbellone e Eberspacher (2000).

Seguindo a mesma regra de precedências da matemática convencional, os operadores irão se comportar dentro de um algoritmo da mesma forma de prioridades, como ilustrado na Tabela 1.8:

Tabela 1.8 | Precedências entre operadores aritméticos

Prioridade	Operadores
1ª	Parênteses mais internos
2ª	pot e rad
3ª	*, /, div e mod
4ª	+ e -

Fonte: Forbellone e Eberspacher (2000, p. 20).

Quando os operadores estiverem dentro da mesma prioridade, você deverá iniciar a solução da expressão da esquerda para a direita, por exemplo:

$$10 + 4 - 8$$

$$14 - 8$$

$$6$$

Tudo bem até aqui? Agora você vai conhecer os operadores lógicos e relacionais.

Operadores lógicos

Caro aluno, na primeira seção desta unidade trabalhamos a lógica booleana. Seguindo este mesmo raciocínio, você utilizará os operadores lógicos para representar situações que não são tratáveis por operadores aritméticos; veja na Tabela 1.9 os operadores lógicos utilizados nos algoritmos.

Tabela 1.9 | Operadores lógicos

Operadores	Operadores
Não	Negação
E	Conjunção
OU	Disjunção

Fonte: adaptada de: Forbellone e Eberspacher (2000).

Agora veja a tabela verdade dos operadores lógicos:

Tabela 1.10 | Tabela verdade

A	B	A ("E") B	A ("OU") B	NÃO A	NÃO B
Verdade	Verdade	Verdade	Verdade	Falso	Falso
Verdade	Falso	Falso	Verdade	Falso	Verdade
Falso	Verdade	Falso	Verdade	Verdade	Falso
Falso	Falso	Falso	Falso	Verdade	Verdade

Fonte: elaborada pelo autor.

Veja o exemplo a seguir:

Se o terreno for plano **e** dentro de um condomínio, construo a casa.

A pergunta é: Quando vou construir a casa?

Somente se o terreno for plano **e** dentro de um condomínio.

Agora:

Se o terreno for plano **ou** dentro de um condomínio, construo a casa.

E agora? Quando você constrói a casa?

Em ambos os casos, se o terreno for plano você constrói a casa ou se for dentro de um condomínio.

Operadores relacionais

Segundo Forbellone e Eberspacher (2000), os operadores relacionais são utilizados na realização de comparação entre valores do mesmo tipo primitivo, ou seja, podem ser representados por variáveis, constantes e até mesmo em expressões aritméticas.

Veja na Tabela 1.11 os operadores relacionais.

Tabela 1.11 | Operadores relacionais

Operadores	Função	Exemplos
=	Igual	A=B
>	Maior	7>5, x>y
<	Menor	3<5, x<y
>=	Maior ou igual	x>=y
<=	Menor ou igual	x<=y
<>	Diferente	x<>y

Fonte: adaptada de: Forbellone e Eberspacher (2000)

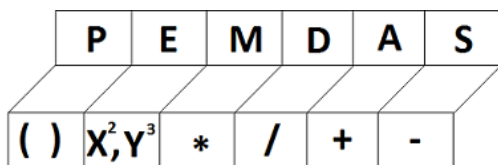


As precedências entre todos os operadores também devem ser levadas em consideração.

1. Em primeira ordem, deve-se considerar os parênteses mais internos;
2. Em segundo, os operadores aritméticos, considerando a seguinte regra:
 - 2.1. Resolver primeiro as multiplicações ou divisões, depois a adição ou a subtração.

Veja na Figura 1.7 as precedências dos operadores aritméticos:

Figura 1.7 | Precedências de operadores aritméticos



Fonte: elaborada pelo autor.

3. Em terceiro, os operadores relacionais, que obedecem aos parênteses e, na ausência desses, a operação é realizada da esquerda para a direita;
4. E em quarto, os operadores lógicos, que obedecem aos parênteses e, na ausência desses, a operação é realizada da esquerda para a direita.

Fiquem atentos para essas regras básicas, pois elas serão de grande valia para a solução dos problemas.

Atribuição

Atribuição pode ser utilizada em algoritmos para determinar um valor a uma variável ou até mesmo para uma expressão, e é representada por uma seta (\leftarrow).

A \leftarrow 7

Significa que **A** tem o valor **7** (ou **A** recebe o valor **7**). Importante! As variáveis devem ser do mesmo tipo do valor a ser atribuído, ou seja, uma variável do tipo inteiro só recebe um valor do tipo inteiro.

Comentários

Em alguns casos, você pode criar comentários (explicações) nas linhas de comandos dos algoritmos utilizando (`//`), lembrando que o que estiver na frente das `//` não será interpretado pelo algoritmo.

Entrada e saída

Podemos descrever os seguintes comandos de entrada e saída:

Comando **"escreva"** – utilizado para mostrar algo na tela do computador, conhecido como comando de saída.

Comando **"leia"** – utilizado para armazenar os dados de uma variável, também conhecido como comando de entrada.

Muito bem, agora que você conhece as definições e aplicações para as variáveis, constantes e operadores, vamos trabalhar alguns algoritmos na sua estrutura sequencial.

Lembrando que a estrutura sequencial de um algoritmo corresponde à forma linear de execução.

Conheça agora a estrutura do algoritmo:

Início // início do algoritmo ou parte de bloco

Var // declaração das variáveis.

....

.... // conteúdo do algoritmo.

....

....

fim. // finaliza o algoritmo.

Veja o algoritmo a seguir, que calcula a média aritmética dos alunos da disciplina de algoritmo e lógica de programação.

1. início: // início do algoritmo
2. real: nota1, nota 2, valor_da_média; // variáveis do tipo real.
3. caractere: nome; // variável do tipo caractere.
4. escreva: ("Digite o nome do aluno"); comando de saída, o conteúdo do texto sairá na tela do computador.
5. leia: (nome); // será armazenado o conteúdo da variável

"nome" na memória do computador.

6. escreva: ("Digite a Primeira Nota"); // comando de saída, o conteúdo do texto sairá na tela do computador.
7. leia: (nota 1); // será armazenado o conteúdo da variável "nota 1" na memória do computador.
8. escreva: ("Digite a Segunda Nota"); // comando de saída, o conteúdo do texto sairá na tela do computador.
9. leia: (nota 2); // será armazenado o conteúdo da variável "nota 2" na memória do computador.
10. Valor_da_média ← (nota 1 + nota 2) / 2 // neste caso é atribuído o resultado da expressão "(nota 1 + nota 2) / 2" para a variável "média"
11. Escreva: ("A média do aluno é:"); valor_da_média); // neste caso a frase "A média do aluno é:" sairá na tela do computador e o resultado armazenado na variável "média" será apresentado logo à frente da frase.
12. fim.

Veja agora como fica o algoritmo sem os comentários:

início;

real: nota 1, nota 2, valor_da_média;

caractere: nome;

escreva: ("Digite o nome do aluno");

leia: (nome);

escreva: ("Digite a Primeira Nota");

leia: (nota1);

escreva: ("Digite a Segunda Nota");

leia: (nota2);

valor_da_média (nota 1 + nota 2) / 2;

Escreva: ("A média do aluno é:"); valor_da_média);

fim.



Quando for definir uma variável, **cuidado na hora da sua identificação!** Lembre-se: você não pode iniciar uma variável com caracteres especiais (é, ç, ã, &, %, #, @, *, -, entre outros) exceto "_" (*Underline*).

Pensando nessa regra, qual a forma correta para definir uma variável em um algoritmo?

Para encerrar esta seção do livro, vamos inverter os valores de duas variáveis em um algoritmo.

Início:

```
Inteiro: A, B, INVERTE; // define as variáveis
escreva ("algoritmo inverter valores"); // mostrar a frase na
tela, comando de saída
```

```
A ← 100; // Atribui 100 para a variável A
```

```
B ← 200; // Atribui 200 para a variável B
```

```
INVERTE ← A; // atribui o valor da variável A para a variável
INVERTE
```

```
A ← B; // atribui o valor da variável B para variável A
```

```
B ← INVERTE; // atribui o valor da variável INVERTE para a
variável B
```

```
escreva ("O valor de A agora é:", A); // Mostrar na tela o novo
valor da variável A
```

```
escreva ("O valor de B agora é:",B); // Mostrar na tela o novo
valor da variável B
```

fim.



Veja agora um algoritmo que realiza o cálculo de uma saída de dados:

```
Algoritmo saída_de_dados
```

```
Var
```

```
pre_unit, pre_tot: real
```

```
quant: inteiro
```

```
Início
```

```
pre_unit := 25
quant := 5
pre_tot := pre_unit * quant
Escreva pre_tot
Fim.
```

Perceba que a linguagem usada difere da estudada até aqui, porém, o raciocínio não muda.

O que mudou, então?

O algoritmo inicia com a palavra "algoritmo" e um nome.

As variáveis são declaradas antes do início.

A forma de atribuição é realizada através ":=".

As demais sintaxes são semelhantes.

Bem, isso é apenas o começo do que você pode fazer com algoritmos. Em algumas literaturas você vai encontrar várias formas de escrever um algoritmo, mas não se assuste! A lógica será sempre a mesma. Caso tenha dúvidas, assista às vídeoaulas, realize as atividades-diagnósticos e exercite com as pré- e pós-aulas. Boa sorte e até a próxima seção!

Sem medo de errar

Chegamos ao Sem medo de errar. Vamos resgatar a nossa situação-problema do início do livro, onde você é o engenheiro responsável por contribuir com os seus colegas engenheiros ensinando algoritmos para solução de cálculos matemáticos.

O seu desafio é criar um algoritmo da Fórmula de Bhaskara

$\left(\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \right)$. Nesse contexto, você deverá trabalhar as variáveis,

constantes e os tipos de operadores. Tente ser o mais didático possível na sua explicação, para que seja de fácil entendimento a seus colegas engenheiros.

Para a solução do problema vamos entrar com algumas constantes, ok?

Nesse momento, não iremos trabalhar com condicionais, por esse motivo os valores atribuídos devem ser maiores que "0".

Você deverá mostrar as aplicações das variáveis, constantes e operadores nos comentários do algoritmo.

Vamos lá:

```
inicio // Resolver a Fórmula de Bhaskara
```

```
    real: b2, x1, x2, delta, a, b, c; // definição das variáveis e o seu  
    tipo, serão elas que armazenarão os valores das operações.
```

```
    a ← 1; // será atribuída à variável "a" o valor do tipo real "1"
```

```
    b ← -5; // será atribuída à variável "b" o valor do tipo real "-5"
```

```
    c ← 6; // será atribuída à variável "c" o valor do tipo real "6"
```

```
    escreva ("Dados os valores a = 1, b = -5 e c = 6");
```

```
    b2 ← b*b; //(é atribuído para b2 o quadrado de b)
```

```
    delta ← b2 - 4 * a * c; // (será atribuído para delta o resultado  
    das operações aritméticas.
```

```
    escreva ("Existem duas raízes")
```

```
    x1 ← (-b+rad(delta))/2*a
```

```
    x2 ← (-b-rad(delta))/2*a
```

```
    escreva ("Primeira raiz X", x1)
```

```
    escreva ("Segunda raiz X", x2)
```

```
fim.
```

Muito bem, lembrando que não usamos condições para delta menor que zero e nem para delta igual a zero. Em breve iremos trabalhar nesse contexto.

Boa sorte e ótimos estudos!

Avançando na prática

Mérito salarial

Descrição da situação-problema

Você foi responsável por um grande projeto no setor da empresa Kro_Engenharias e por esse motivo sua equipe foi gratificada com um bônus de 30% sobre seus salários. Para tal atribuição, você

precisa de um algoritmo que realize o cadastro dos colaboradores, o salário bruto e o adicional liberado para esse projeto.

Resolução da situação-problema

início

caractere: nome;

real: sal, bônus, totals;

escreva: ("Entre com o nome do funcionário");

leia: (nome);

escreva: ("Entre com o salário bruto do funcionário");

leia: (sal);

bônus \leftarrow sal*(30/100);

totals: \leftarrow sal+bônus;

escreva: ("Seu salário com bônus será de: "; totals);

fim

Faça valer a pena

1. Segundo Forbellone e Eberspacher (2000), os operadores relacionais são utilizados na realização de comparação entre valores do mesmo tipo primitivo, ou seja, eles podem ser representados por variáveis, constantes e até mesmo em expressões aritméticas.

Segundo a definição de Forbellone e Eberspacher (2000), assinale a alternativa correta:

- a) Em algoritmos usamos "<>" para referenciar a igualdade entre as variáveis.
- b) O sinal de ">" e o sinal de "<" correspondem à atribuição entre as variáveis.
- c) Em algoritmos usamos " \geq " para representar maior ou igual.
- d) O operador relacional "<>" representa a diferença entre as variáveis.
- e) Pode ser definido como igualdade em algoritmo o operador "==".

2. Analise o algoritmo seguinte:

início

real: área, base maior, base menor, altura;

escreva: ("Digite a Base maior");

leia: (basemaior);

escreva: ("Digite a Base menor");

leia: (base menor);

escreva: ("Digite a altura");

leia: (altura);

.....

.....

fim.

O algoritmo desta questão calcula a área do trapézio, sua fórmula é

$A = \frac{(B+b) \cdot h}{2}$. Assinale a alternativa que corresponde às linhas do algoritmo

que estão faltando.

- a) área $\leftarrow ((\text{base menor} + \text{base maior}))/2$
escreva ("A área do trapézio é", área)
- b) área $\leftarrow ((\text{base menor} + \text{base maior})/\text{altura})*2$
escreva ("A área do trapézio é", área)
- c) altura $\leftarrow ((\text{base menor} + \text{base maior})*\text{base})/2$
escreva ("A área do trapézio é", área)
- d) área $\leftarrow ((\text{base menor})*\text{altura})/2$
escreva ("A área do trapézio é", área)
- e) área $\leftarrow ((\text{base menor} + \text{base maior})*\text{altura})/2$
escreva ("A área do trapézio é", área)

3. O algoritmo a seguir calcula a área do círculo $A = \pi \cdot r^2$. Analise linha a linha da sua execução.

- 1. início
- 2. real: área, p1, raio;
- 3. escreva: ("digite o raio do círculo");
- 4. leia: (raio);
- 5. p1 $\leftarrow 3.14$;
- 6. área $\leftarrow (p1*(\text{pot}(\text{raio},2))$;
- 7. escreva: ("A área do círculo é: ", raio);
- 8. fim.

Em relação ao algoritmo apresentado, podemos afirmar que:

- I. Na linha 5 do algoritmo é definida uma constante.
- II. Na linha 6 do algoritmo é possível substituir por: área $\leftarrow (p1*(\text{raio} * \text{raio}))$;
- III. A linha 7 está escrita de forma errada e o correto é: escreva ("A área do círculo é:", área);

Assinale a alternativa correta.

- a) As afirmações II e III estão corretas.
- b) As afirmações I, II e III estão corretas.
- c) As afirmações I e III estão corretas.
- d) Somente a afirmação I está correta.
- e) As afirmações I e II estão corretas.

Seção 1.3

Representações de algoritmos

Diálogo aberto

Caro aluno, chegamos à terceira seção desta unidade. Nosso principal compromisso é que você conheça as características de uma linguagem natural, o desenvolvimento dos diagramas de blocos (fluxogramas) e as aplicações em pseudocódigos.

Pois bem, pense que você decidiu construir uma casa e precisou mostrar para o arquiteto o que você pensou como a casa ideal na sua concepção, e esta seria a linguagem natural. O arquiteto desenhou as características da sua casa e, nesse caso, trata-se de um fluxograma ou até mesmo de um pseudocódigo. Após esses procedimentos, a casa seria construída, e seria realizada a programação.

Resgatando o contexto de aprendizagem da seção anterior, o seu gestor está muito satisfeito com o seu desempenho perante seus colegas engenheiros, e está muito claro que o pensamento computacional permeia a equipe.

Agora é o momento de você apresentar aos seus colegas as diferentes formas de estruturar as ideias dentro do contexto computacional, ou seja, caracterizar a utilização da linguagem natural, criar e aplicar o fluxograma e os pseudocódigos e, por fim, realizar atividades para soluções de problemas lógicos.

Seguindo essa linha de pensamento, você propôs aos seus colegas engenheiros que escrevessem, em linguagem natural, fluxogramas e pseudocódigos, o cálculo da fórmula que converte a temperatura em graus centígrados para graus Fahrenheit. A fórmula para essa conversão é: $F = \frac{9 \cdot c + 160}{5}$, onde "F" é a temperatura em Fahrenheit e "c" é a temperatura em centígrados.

Seja criativo e tente desenvolver essa atividade com a maior clareza possível.

Boa sorte e bons estudos!

Não pode faltar

Caro aluno, chegamos à última seção desta unidade. Até o presente momento, você estudou a lógica de programação e os fundamentos de algoritmos, agora iremos estudar as representações das linguagens que servirão de base para futuras programações.

Na segunda seção desta unidade de ensino, você desenvolveu habilidades para a criação de algoritmos, entendeu as aplicações de variáveis, atribuições e operadores, e começou a ter um pensamento estruturado para as soluções de problemas. Pois bem, agora vamos conhecer as representações que envolvem as linguagens.

Linguagem natural

Segundo Santos (2001), a linguagem natural, na definição geral, é uma forma de comunicação entre as pessoas de diversas línguas, podendo ser falada, escrita, gesticulada, entre outras formas de comunicação. A linguagem natural tem uma grande contribuição quando vamos desenvolver uma aplicação computacional, pois pode direcionar de forma simples e eficiente as descrições dos problemas e suas soluções.

Para reforçar os conceitos de linguagem natural, vamos tomar como exemplo a operação de dois valores utilizando os quatro operadores básicos da matemática.

O problema é o seguinte: o usuário deverá entrar com dois valores, e o computador retornará o resultado com cada uma das operações aritméticas (soma, subtração, multiplicação e divisão).

Vamos lá! Existem várias formas de estruturar esse problema. Veja a seguir uma das maneiras para que ele possa ser realizado:

1. Início.
2. Entrar com o primeiro valor.
3. Entrar com o segundo valor.
4. Realizar a soma utilizando o primeiro valor mais o segundo.
5. Realizar a subtração utilizando o primeiro valor menos o segundo.
6. Realizar a multiplicação utilizando o primeiro valor vezes o segundo.

7. Realizar a divisão utilizando o primeiro valor dividido pelo segundo.
8. Mostrar na tela o resultado da soma.
9. Mostrar na tela o resultado da subtração.
10. Mostrar na tela o resultado da multiplicação.
11. Mostrar na tela o resultado da divisão.
12. Fim.

Transcrever a linguagem natural em linguagem computacional necessita de um prévio entendimento, que posteriormente poderá sofrer transformações em forma de algoritmos, diagrama de blocos e pseudocódigos.

Muito bem, agora vamos estudar o funcionamento dos diagramas de blocos.



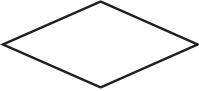




Diagrama de blocos (Fluxograma)

Segundo Manzano (2015), podemos caracterizar diagrama de blocos como um conjunto de símbolos gráficos em que cada um desses símbolos representa ações específicas a serem executadas pelo computador. Vale lembrar que o diagrama de blocos ou fluxograma determina a linha de raciocínio utilizada pelo programador para resolver problemas. Ao escrever um fluxograma, o programador deve estar ciente de que os símbolos utilizados devem estar em harmonia e ser de fácil entendimento. Para que os diagramas de blocos tenham certa coerência, os seus símbolos foram padronizados pelo ANSI (Instituto Norte-Americano de Padronização).

Veja agora a definição dos principais símbolos utilizados em um diagrama de blocos ou fluxograma:

Quadro 1.1 | Descrição e significados de símbolos no diagrama de blocos

Símbolo	Significado	Descrição
	Terminal	Representa o início ou o fim de um fluxo lógico. Em alguns casos definem as sub-rotinas.
	Entrada manual	Determina a entrada manual dos dados, geralmente através de um teclado.

Símbolo	Significado	Descrição
	Processamento	Representa a execução de ações de processamento.
	Exibição	Mostra o resultado de uma ação, geralmente através da tela de um computador.
	Decisão	Representa os desvios condicionais nas operações de tomada de decisão e laços condicionais para repetição de alguns trechos do programa.
	Preparação	Representa a execução de um laço incondicional que permite a modificação de instruções do laço.
	Processo predefinido	Define um grupo de operações relacionadas a uma sub-rotina.
	Conector	Representa pontos de conexões entre trechos de programas, que podem ser apontados para outras partes do diagrama de bloco.
	Linha	Representa os vínculos existentes entre os símbolos de um diagrama de blocos.

Fonte: Adaptado de: Manzano (2015).

A partir do momento que for utilizando os símbolos com as instruções deles, você vai aprendendo e desenvolvendo cada vez mais a lógica deles em relação aos problemas.



Refleta

Segundo Souza (2011), os profissionais da área de Exatas fazem uso de fluxogramas devido à sua familiaridade com diagramas esquemáticos, com linguagem matemática, bem como com as expressões gráficas. Isso ocorre por causa do apelo visual que os fluxogramas têm na formação desses profissionais.

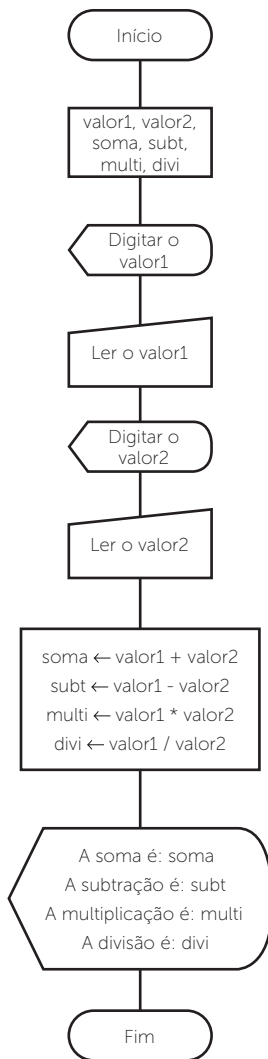
E você, como vê essa relação dos fluxogramas com a sua área de atuação?

Vamos realizar um fluxograma que resolva o cálculo de dois valores exibindo sua soma, subtração, multiplicação e divisão.

Para um melhor entendimento do fluxograma, acompanhe os comentários de execução para cada símbolo.

Diagrama 1.1 | Cálculo de dois valores

- O símbolo terminal deu início ao diagrama de blocos.
- O símbolo de processamento definiu as variáveis que serão utilizadas no diagrama de blocos: valor1, valor2, soma, subt, multi e divi.
- O símbolo exibição mostra na tela o que o usuário deve fazer.
- O símbolo de entrada manual libera para o usuário entrar com o primeiro valor.
- O símbolo exibição mostra na tela o que o usuário deve fazer.
- O símbolo de entrada manual libera para o usuário entrar com o segundo valor.
- O símbolo de processamento é realizado com as atribuições os valores calculados para suas respectivas variáveis.
- O símbolo de exibição mostra na tela o resultado de cada valor calculado.
- Finaliza o programa.



Fonte: Elaborado pelo autor.

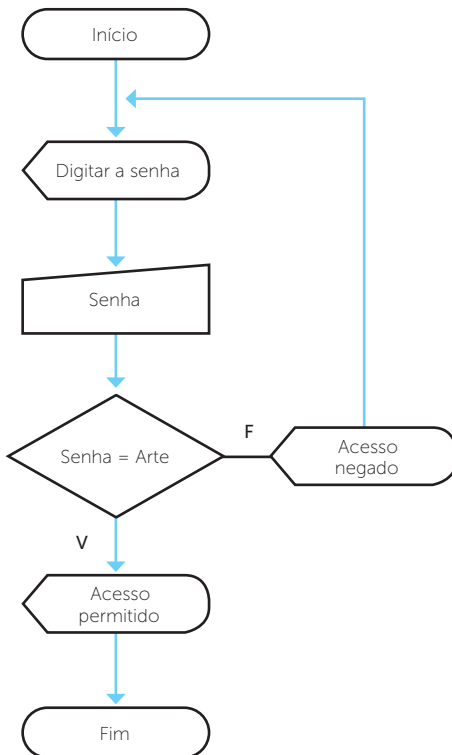


Vale observar alguns pontos de atenção para a construção de um fluxograma:

1. Estar atento aos níveis.
2. O fluxograma deve começar de cima para baixo e da esquerda para a direita.
3. Cuidado para não cruzar as linhas, principalmente as linhas de fluxos de dados.

Para finalizar os diagramas de blocos (fluxograma), vamos criar uma situação utilizando uma operação lógica em que a finalidade é entrar com uma senha no sistema.

Diagrama 1.2 | Diagrama de blocos com uso de decisão



Fonte: Elaborado pelo autor.



Existem vários softwares para criação de fluxogramas, entre eles destacamos Lucidchart, um gerador de fluxograma online e gratuito. Disponível em: <<https://www.lucidchart.com/pages/pt>> Acesso em: 23 out. 2017.

Outro software muito utilizado é o "Dia". Você pode fazer o download pelo link: disponível em: <<http://dia-installer.de/>>. Acesso em: 23 out. 2017.

Pseudocódigo

Segundo Aguilar (2011), o pseudocódigo é considerado uma ferramenta que pode auxiliar a programação. Pode ser escrito em palavras similares ao inglês ou ao português para facilitar a interpretação e o desenvolvimento de um programa.

Podemos caracterizar os algoritmos como um pseudocódigo. Existem várias formas de representar um pseudocódigo, mas o mais importante é que ele apresente a solução do problema proposto.

Veja agora o pseudocódigo (portugol) para resolver o problema das operações aritméticas citadas anteriormente:

1. calculo_operadores;
2. var;
3. valor1, valor2, soma, subt, multi, divi: real;
4. Início;
5. escreva ("Digite o valor 1");
6. leia valor1;
7. escreva ("Digite valor 2");
8. leia valor2;
9. soma ← valor1 + valor2;
10. subt ← valor1 - valor2;
11. multi ← valor1 * valor2;
12. divi ← valor1 / valor2;
13. escreva ("a soma é:", soma);
14. escreva ("a subtração é:", subt);
15. escreva ("a multiplicação é:", multi);
16. escreva ("a divisão é:", divi);
17. Fim.

Veja os comentários desse algoritmo:

Linha 1: "cálculo_operadores" esse é o nome reservado para identificar o algoritmo.

Linha 2: "Var" sinaliza o início das variáveis.

Linha 3: São os nomes dados para as variáveis (valor1, valor2, soma, subt, multi, divi); nesta linha também são definidos os tipos de variáveis (real).

Linha 4: inicia os procedimentos dos algoritmos(início).

Linha 5: "escreva" é um comando de saída e indica o que vai ser mostrado na tela do computador. Geralmente o conteúdo do texto fica entre aspas ("Digite valor 1").

Linha 6: "leia" é um comando de entrada, o valor digitado é armazenado na variável (valor1).

Linha 7: "escreva" é um comando de saída e indica o que vai ser mostrado na tela do computador. Geralmente o conteúdo do texto fica entre aspas ("Digite valor 2").

Linha 8: "leia" é um comando de entrada, o valor digitado é armazenado na variável (valor2).

Linha 9: A adição das variáveis valor1 e valor2 é atribuída para variável soma ($soma \leftarrow valor1 + valor2$).

Linha 10: A subtração das variáveis valor1 e valor2 é atribuída para variável subt ($subt \leftarrow valor1 - valor2$).

Linha 11: A multiplicação das variáveis valor1 e valor2 é atribuída para variável multi ($multi \leftarrow valor1 * valor2$).

Linha 12: A divisão das variáveis valor1 e valor2 é atribuída para variável divi ($divi \leftarrow valor1 / valor2$).

Linha 13: Escreva na tela o que está entre aspas ("a soma é:", soma) e o valor que está armazenado na variável soma. Perceba que a variável é colocada fora das aspas, para que ela seja representada pelo seu valor correspondente.

Linha 14: Escreva na tela o que está entre aspas ("a subtração é:", subt) e o valor que está armazenado na variável subt. Perceba que a variável é colocada fora das aspas, para que ela seja representada pelo seu valor correspondente.

Linha 15: Escreva na tela o que está entre aspas ("a multiplicação é:", multi) e o valor que está armazenado na variável multi.

Perceba que a variável é colocada fora das aspas, para que ela seja representada pelo seu valor correspondente.

Linha 16: Escreva na tela o que está entre aspas ("a divisão é:", divi) e o valor que está armazenado na variável divi. Perceba que a variável é colocada fora das aspas, para que ela seja representada pelo seu valor correspondente.

Linha 17: Encerre o algoritmo com a palavra "fim" e o ponto final.

Quando nos referimos ao uso de pseudocódigos em algoritmos do tipo portugol, significa que esse deve ser escrito de forma que outras pessoas possam entender de maneira mais formal, ou seja, os conceitos devem ficar claros e em uma linguagem acessível para quem irá interpretar o algoritmo.

É importante estar atento para algumas regras básicas quando utilizar pseudocódigos:

- Escolher um nome.
- Avaliar as variáveis, dar atenção aos seus tipos e às suas características.
- Descrever de forma clara o que será armazenado e se as variáveis destinadas a essa informação estão corretas.
- Verificar se as instruções fazem sentido e se têm uma sequência lógica.
- Avaliar o resultado e, quando pertinente, mostrar na tela.
- Finalizar o algoritmo.



Exemplificando

Veja um algoritmo que calcula a área do círculo usando um pseudocódigo específico para ser executado no software Visualg (software que executa os comandos dos algoritmos).

algoritmo "área do círculo"

var

área, p1, raio: real

início

 // Seção de Comandos

 Escreva ("digite o raio do círculo")

```
Leia (raio)
p1 <- 3.14
área <- (p1*(raio^2))
escreva ("A área do círculo é: ", área)
```

fimalgoritmo

Perceba que os parâmetros utilizados também são considerados um algoritmo do tipo português estruturado, ou seja, de fácil entendimento e interpretação.

Software disponível em: <<http://visualg3.com.br/>>. Acesso em: 23 out. 2017.

Parabéns! Você terminou a primeira unidade deste livro, continue estudando e praticando. Boa sorte e até a próxima unidade!

Sem medo de errar

Chegou o momento de praticar o conteúdo estudado. Lembre-se: você foi elogiado pelo seu desempenho e seu gestor espera que você coloque de forma convincente as aplicações da linguagem natural, dos diagramas de blocos (fluxogramas) e do uso de pseudocódigos.

A sua missão é demonstrar aos seus colegas engenheiros a aplicação em linguagem natural, fluxogramas e pseudocódigos, o cálculo da fórmula que converte uma temperatura em graus centígrados para graus Fahrenheit. A fórmula para essa conversão é:

$$F = \frac{9 \cdot c + 160}{5}$$
, em que "F" é a temperatura em Fahrenheit e "c" é a temperatura em centígrados.

Primeiro, você deve escrever em linguagem natural o funcionamento da fórmula que converte graus centígrados para graus Fahrenheit. Para isso, organize o seu pensamento de forma estrutural e formal, mas lembre-se, nem todos os engenheiros terão o mesmo pensamento, o importante é chegar à solução do problema.

Na construção do fluxograma vale resgatar as dicas citadas no "Não pode faltar":

1. Estar atento aos níveis no fluxograma.
2. O fluxograma deve começar de cima para baixo e da esquerda para direita.
3. Cuidado para não cruzar as linhas, principalmente as linhas de fluxos de dados.

Finalize essa atividade escrevendo a conversão de graus centígrados para graus Fahrenheit em pseudocódigo. Assim como no fluxograma, as formas de pensamento podem sofrer alterações, mas o importante é o resultado e a lógica para sua solução.

Cabem as dicas:

- Escolher um nome para o algoritmo.
- Avaliar as variáveis, dar atenção aos seus tipos e às suas características.
- Descrever de forma clara o que será armazenado e se as variáveis destinadas a essa informação estão corretas.
- Verificar se as instruções fazem sentido e se têm uma sequência lógica.
- Avaliar o resultado e, quando pertinente, mostrar na tela.
- Finalizar o algoritmo.

Boa sorte e bons estudos!

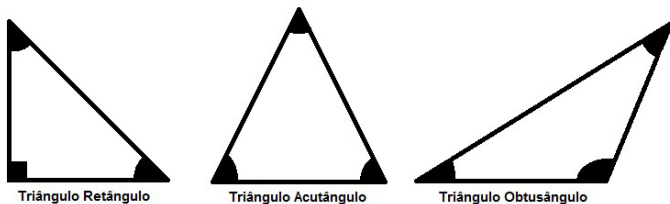
Avançando na prática

Tipos de triângulos – considerando seus ângulos internos

Descrição da situação-problema

Você foi designado para ajudar um departamento da empresa Xis&Zi a encontrar os ângulos internos dos triângulos que correspondem a alguns tipos de peças. Por meio das medidas encontradas, você deverá apontar o tipo do triângulo. A empresa precisa que você passe em forma de linguagem natural esse procedimento. Veja na Figura 1.15 os tipos de triângulos e as suas representações em relação aos ângulos.

Figura 1.15 | Tipos de triângulos classificados por ângulos internos



Fonte: elaborado pelo autor.

Resolução da situação-problema

Em linguagem natural, poderia ser escrita da seguinte forma:

1. Início.
2. Solicitar a entrada do primeiro ângulo (Variável angA).
3. Solicitar a entrada do segundo ângulo (Variável angB).
4. Solicitar a entrada do terceiro ângulo (Variável angC).
5. Verificar se as somas dos ângulos internos somam 180 graus ($\text{angA} + \text{angB} + \text{angC} = 180$).
6. Se um dos ângulos internos do triângulo for reto (90 graus) e os demais, ângulos agudos (inferiores a 90 graus), esse triângulo é classificado como sendo triângulo retângulo.
7. Agora, se todos os ângulos internos do triângulo forem agudos (inferiores a 90 graus), esse triângulo é chamado de acutângulo.
8. Enfim, se um dos ângulos internos do triângulo for obtuso (maior que 90 graus) e os demais, agudos (inferiores a 90 graus), esse triângulo é chamado obtusângulo.
9. Mostrar o tipo do triângulo na tela.
10. Fim.

Muito bem, agora é com você! Tente aplicar esse mesmo problema utilizando fluxograma e pseudocódigo.

Sucesso e até a próxima unidade!

Faça valer a pena

1. A linguagem natural na definição geral é uma forma de comunicação entre as pessoas de diversas línguas, podendo ser falada, escrita ou gesticulada, entre outras formas de comunicação. A linguagem natural precisa ser criada formalmente, ou seja, ela deverá ser de fácil entendimento ao seu interpretador.

Seguindo a premissa do conceito de linguagem natural, analise os termos a seguir:

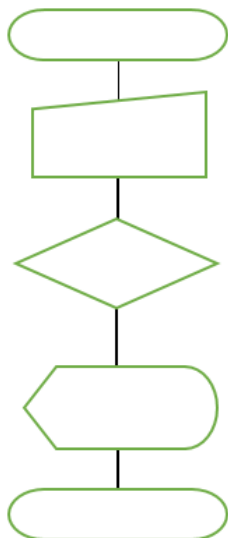
- I. Início
Somar A com B
Mostrar resultado da soma
Fim.
- II. Início
Entrar com o valor de A
Entrar com o valor de B
Somar A com B
Mostrar o resultado da soma
Fim.
- III. Início
Pi=3,14
Calcular área do círculo (área <- $p1*(raio^2)$)
Dar o resultado
Fim.

- a) As rotinas I e II estão corretas.
- b) As rotinas II e III estão corretas.
- c) A rotina II está correta.
- d) As rotinas I, II e III estão corretas.
- e) A rotina III está correta.

2. Segundo Manzano (2015), podemos caracterizar diagrama de blocos como um conjunto de símbolos gráficos, em que cada um desses símbolos representa ações específicas a serem executadas pelo computador.

O diagrama de blocos ou fluxograma determina a linha de raciocínio utilizada pelo programador para resolver problemas.

Analise o diagrama de blocos a seguir:



Assinale a alternativa correta:

- a) O diagrama apresentado representa o início da rotina, seguido por uma apresentação na tela, uma condicional (Verdadeiro ou Falso), o armazenamento de dados e o fim da rotina.
- b) O diagrama apresentado representa o início da rotina, seguido da entrada da informação, uma condicional (Verdadeiro ou Falso), o armazenamento de dados e o fim da rotina.
- c) O diagrama apresentado representa o início da rotina, seguido da entrada da informação, uma condicional (Verdadeiro ou Falso), a apresentação na tela e o fim da rotina.
- d) O diagrama apresentado representa o início da rotina, seguido por uma apresentação na tela, o procedimento, o armazenamento de dados e o fim da rotina.
- e) O diagrama apresentado representa o início da rotina, seguido da entrada da informação, o procedimento, a apresentação na tela e o fim da rotina.

3. Segundo Aguilar (2011), o pseudocódigo é considerado uma ferramenta que pode auxiliar a programação. Pode ser escrito em palavras similares ao inglês ou ao português, para facilitar a interpretação de algoritmos e assim criar possibilidades para construção de programas computacionais.

Analise a falha no trecho do pseudocódigo a seguir e assinale a alternativa que corresponde à forma correta na execução:

algoritmo "Calcular o IMC";

var;

nome: caractere;

peso, imc, altura: real;

Início

```
    escreva ("Escreva o nome");
    leia (nome);
    escreva (Escreva o peso);
    leia (peso);
    escreva ("Escreva a altura");
    leia (altura);
    imc = peso / (altura * altura);
    escreva ("O seu IMC é");
    leia(imc);
```

fim.

- a) escreva ("Escreva a altura");
leia (altura);
 $imc \leftarrow peso / (altura * altura)$;
- b) escreva ("Escreva a altura", altura);
 $imc = peso / (altura * altura)$;
escreva ("O seu IMC é");
leia ("imc");
- c) escreva ("Escreva a altura");
 $imc = peso / (altura * altura)$;
escreva ("O seu IMC é", imc);
leia (imc);
- d) escreva ("Escreva a altura");
leia (altura);
 $IMC \leftarrow peso / (altura * altura)$;
escreva ("O seu IMC é", imc);
- e) escreva ("Escreva a altura");
leia (altura);
 $IMC \leftarrow peso / (altura * altura)$;
escreva ("O seu IMC é, imc");

Referências

- ABE, Jair Minoru; SCALZITTI, Alexandre; SILVA FILHO, João Inácio. **Introdução à lógica para a ciência da computação**. São Paulo: Arte & Ciência, 2001.
- AGUILAR, Luís Joyanes. **Fundamentos de programação**: algoritmos, estruturas de dados e objetos. 3. ed. Porto Alegre: AMGH, 2011.
- ALVES, William Pereira. **Linguagem e lógica de programação**. 1. ed. São Paulo: Érica, 2014.
- BERG, Alexandre Cruz; FIGUEIRÓ, Joice Pavec. **Lógica de programação**. Rio Grande do Sul: Editora ULBRA, 1998.
- FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. **Lógica de programação**: A construção de algoritmos e estruturas de dados. São Paulo: Makron, 2000.
- FORBELLONE, André Luiz Villar. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005.
- LOPES, Anita; GARCIA, Guto. **Introdução à programação**. Rio de Janeiro: Elsevier, 2002.
- MANZANO, José Augusto N. G. **Algoritmos**: lógica para desenvolvimento de programação de computadores. 28. ed. São Paulo: Érica, 2016.
- MANZANO, José Augusto N. G. **Algoritmos**: técnicas de programação. 2. ed. São Paulo: Érica, 2015.
- MANZANO, José Augusto. **Algoritmos**: lógica para desenvolvimento de programação. 10. ed. São Paulo: Érica, 2000.
- PEREIRA, Ana Paula. **O que é algoritmo**. 2009. Disponível em: <<https://www.tecmundo.com.br/programacao/2082-o-que-e-algoritmo-htm>>. Acesso em: 13 out. 2017.
- SALIBA, Walter Luiz Caram. **Técnica de programação**: uma abordagem estruturada. São Paulo: Makron, 1993.
- SANTOS, Diana. Processamento da linguagem natural: uma apresentação através das aplicações. In: RANCHHOD, Elisabete Marques. **Tratamento das línguas por computador**. Lisboa: Caminho, 2001.
- SILVA, Flávio Soares Corrêa. **Lógica para computação**. São Paulo: Cengage Learning, 2006.
- SOARES, Edvaldo. **Fundamentos de lógica**: elementos de lógica formal e teoria da argumentação. 2. ed. São Paulo: Atlas, 2014.
- SOUZA, Marco Antônio Furlan. **Algoritmos e lógicas de programação**. 2. ed. São Paulo: Cengage Learning, 2011.
- SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. **Estruturas de dados e seus algoritmos**. Rio de Janeiro: Editora LTC, 1994.

Elementos de algoritmos

Convite ao estudo

Caro(a) estudante, bem-vindo(a) à segunda unidade do estudo dos algoritmos e da lógica de programação. Nessa etapa, iremos conhecer elementos que são essenciais para o desenvolvimento de algoritmos eficientes. Os recursos que serão vistos aqui nos permitirão alterar o fluxo sequencial do algoritmo através de tomadas de decisões e também por meio de repetições contínuas de determinadas instruções.

Na primeira unidade, você foi apresentado à multinacional Kro Engenharias, uma empresa atuante em várias frentes de projetos, que vão desde a concepção até a execução e entrega para seus clientes. Diante de tamanha demanda operacional, o sistema de gerenciamento da empresa é dividido em diversos subsistemas, que contemplam desde o gerenciamento de recursos humanos até o controle da execução e entrega dos projetos. Para que tudo funcione de maneira correta, é imprescindível que a interlocução entre cada subsistema seja feita de forma padronizada. Para isso, é necessário, em primeiro lugar, validar os diversos dados de entrada que alimentam os subsistemas e, em um segundo momento, garantir que os dados de cada subsistema estejam em um mesmo padrão nas diversas unidades e setores da empresa. A validação e a padronização dos dados garantem que todos os sistemas se comuniquem de modo correto e eficaz.

Diante dessa necessidade, após você realizar um ótimo trabalho no desenvolvimento do pensamento computacional e da lógica com os engenheiros da empresa, foi atribuída a você a tarefa de desenvolver algoritmos capazes de fazer validações e, quando necessário, conversões nos dados que alimentam alguns módulos do subsistema de cálculo.

Tais subsistemas geram relatórios para que as diversas áreas possam tomar decisões pertinentes ao seu setor, portanto, os dados utilizados devem ser verificados e padronizados.

A execução da sua tarefa será dividida em três momentos. Na primeira etapa, você fará validações nos dados de entrada correspondentes às unidades de comprimento, tempo e massa. No segundo momento, você desenvolverá algoritmos que convertam certas unidades de acordo com o padrão adotado pela empresa. E, por fim, você deverá explorar maneiras otimizadas para o armazenamento dos dados de entrada do sistema.

Tenha um ótimo trabalho nesse novo desafio! E não se esqueça: muitos resultados importantes dependem do seu trabalho com a validação e padronização de dados.

Seção 2.1

Execução sequencial e estruturas de decisão

Diálogo aberto

Olá, estudante! Iniciamos agora o estudo de certos procedimentos que são essenciais para a resolução de qualquer tipo de problema usando como ferramenta os algoritmos. Iremos aprender as rotinas que nos permitem tomar decisões durante a análise de um problema.

Tomar decisões faz parte da vida de qualquer pessoa após certa idade. Desde o momento que acordamos até a hora de dormir, decisões, mesmo que simples (*Que roupa usar? O que comer no café da manhã? Ir trabalhar de transporte privado ou público?*), são tomadas constantemente. Algumas decisões se tornam tão automáticas que nem percebemos que estamos “parando”, “pensando” e “tomando uma decisão”. Escrevemos algoritmos para obter soluções de forma estruturada, porém solucionar implica necessariamente tomar decisões.

Foi atribuída a você uma nova tarefa na Kro Engenharias: desenvolver algoritmos para validar os dados de entrada que alimentam alguns subsistemas de cálculo. Seu primeiro passo será criar fluxogramas e pseudocódigos que validem os valores de entradas de certas unidades de medidas. Esses valores têm que estar de acordo com o Sistema Internacional de Unidades (SI). Para essa tarefa, considere como entradas as unidades de comprimento, tempo e massa e faça os algoritmos necessários para as validações pertinentes a cada entrada,

Para resolver esse problema, você verá como construir estruturas condicionais simples, bem como estruturas condicionais que suportam mais de uma opção. Essas estruturas afetarão o fluxo sequencial do algoritmo, pois, dependendo da decisão, um certo caminho será escolhido em detrimento de outro. Não poderia deixar de mencionar que aprender e refletir sobre como as decisões podem afetar a resolução de problemas está além do uso restrito aos algoritmos computacionais, pois você toma decisões o tempo todo em sua vida.

Então, bom trabalho!

Não pode faltar

Na unidade anterior, você aprendeu que, para resolver determinado problema, é preciso estabelecer uma sequência de passos para alcançar tal objetivo, e nessa sequência os passos são executados um após outro, obedecendo a uma lógica. Por exemplo, para calcular o peso total de um produto (produto + embalagem), primeiro ambos os pesos precisam ser informados, para só depois serem somados, e então apresentar o peso totalizado (início-processamento-saída). Você também aprendeu que, na construção de algoritmos, podemos usar os operadores matemáticos relacionais, por exemplo, maior ($>$) e menor ($<$), para fazer comparações. Usando esses elementos, podemos comparar o peso do produto com um padrão estabelecido pela empresa e, a partir da resposta da comparação, é possível saber se o produto está apto a ir para a loja ou deve voltar à produção, ou ainda ser descartado. Como isso pode ser feito em um algoritmo? A resposta é simples: usando estruturas de decisões!

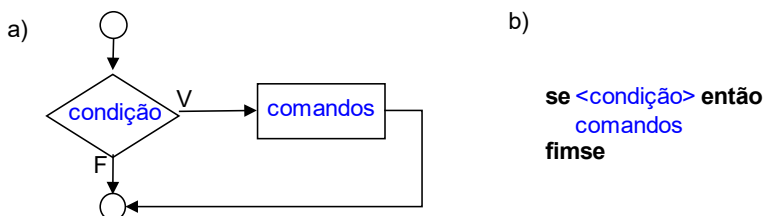
As estruturas de decisões possibilitam incluir nos algoritmos condições de controle, ou seja, através de uma decisão certos passos podem ou não ser feitos. Manzano e Oliveira (2010) chamam tais estruturas de desvio condicional, pois uma decisão sempre acarretará em um desvio no algoritmo.

Segundo Barnett (2017), no ano de 1847 George Boole (1815-1864), um matemático inglês, iniciou uma série de publicações de seus estudos sobre uma nova álgebra, fundamentada na lógica e na matemática. Embora suas primeiras publicações não fossem bem aceitas, o estudo de Boole se tornou uma ferramenta poderosa no estudo de circuitos eletrônicos e na arquitetura dos computadores. A álgebra booleana, também conhecida por álgebra binária, é a álgebra que trabalha com dois valores: zeros e uns, ou verdadeiro e falso. Quando utilizamos uma estrutura de decisão no algoritmo, estamos criando uma decisão binária, baseada na lógica binária, pois para qualquer teste só existem duas possibilidades: ser verdadeiro ou ser falso (SOFFNER, 2013). Normalmente atribui-se ao resultado verdadeiro da condição o valor de 1, e ao resultado falso o valor 0.

Para entendermos como funciona esse controle, vamos começar estudando as estruturas condicionais simples. Você aprendeu

na unidade anterior que, na representação de algoritmos usando fluxogramas, o losango representa uma condição, um teste. Observe a Figura 2.1 – a: o fluxograma possui apenas um losango, isso significa que ele possui apenas um teste, e caso esse teste seja verdadeiro, existe uma ação a ser feita, mas, caso contrário, o bloco dos comandos é ignorado. Esse conjunto, 1 teste + ação caso verdadeiro, é o que caracteriza a condicional simples. Na Figura 2.1 – b, temos a representação em forma de pseudocódigo para a condicional simples. Veja que, para representar uma condição, utilizamos a palavra “**se**”. Para representar a sequência de comandos que deve ser executada, caso o teste seja verdadeiro, usamos “**então**”, e para marcar o final da condição, usamos “**fimse**” (PIVA JUNIOR et al., 2012).

Figura 2.1 | Estrutura condicional simples: a – Fluxograma; b – Pseudocódigo.



Fonte: elaborada pelo autor.



Assimile

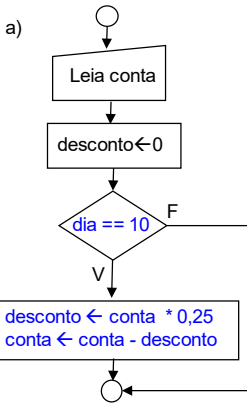
Para resolver qualquer tipo de problema, precisamos tomar decisões. Na construção de algoritmos, quando temos que fazer escolhas, utilizamos as estruturas de decisão. Tais estruturas modificam a execução sequencial do algoritmo, pois, quando uma decisão é tomada, trechos de códigos podem ou não ser executados.

Todos os algoritmos escritos até o momento são executados de forma sequencial linha por linha. A partir de agora, através das estruturas condicionais, você poderá controlar de forma diferente esse fluxo sequencial. Observe no diagrama da Figura 2.1 – a: caso o teste seja verdadeiro, um bloco de comandos será executado, mas caso seja falso, esse trecho de comandos será “pulado” (não será executado), alterando o fluxo sequencial do algoritmo.



Para melhor entendermos a condicional simples, vamos escrever um algoritmo para um estabelecimento, que todo dia 10 concede um desconto de 25% a seus clientes. Portanto, se o dia do mês for 10, o desconto será concedido, caso contrário, nenhum tratamento especial precisa ser feito para o valor da conta (Figura 2.2).

Figura 2.2 | Algoritmo para conceder desconto



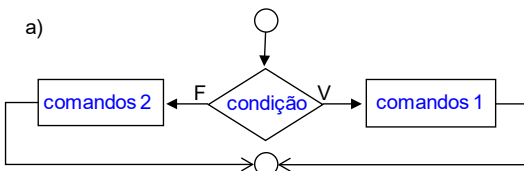
b)

1. inicio
2. var
3. conta, desconto : real
4. Leia(conta)
5. desconto ← 0
6. **se** (dia == 10) **então**
7. desconto ← conta * 0,25
8. conta ← conta - desconto
9. **fimse**

Fonte: elaborada pelo autor.

Na maioria das vezes, quando realizamos algum tipo de teste em um algoritmo, queremos que algo aconteça caso o teste seja verdadeiro, mas também podemos escrever um conjunto de ações caso o teste seja falso (MANZANO; MATOS; LOURENÇO, 2015). Observe na estrutura condicional da Figura 2.3 – a, que agora temos dois conjuntos de ações, um para verdadeiro e outro para falso. No pseudocódigo usado para representar as condicionais compostas (Figura 2.3 – b) aparece um novo elemento, o “**senão**”, que marca o início do bloco caso o teste seja falso.

Figura 2.3 | Estrutura condicional composta



b)

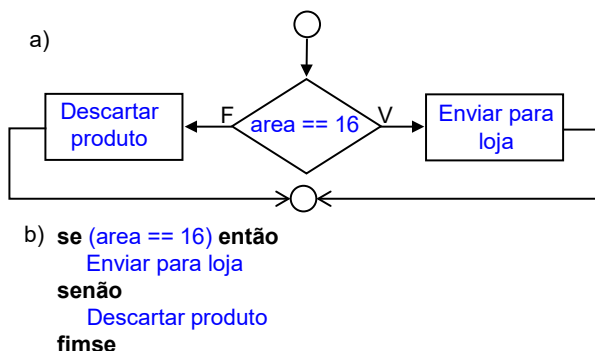
- ```
se <condição> então
 comandos 1
senão
 comandos 2
fimse
```

Fonte: elaborada pelo autor.



Vamos criar uma estrutura condicional composta para o setor de qualidade de uma empresa fabricante de processadores de computadores. A empresa padronizou o tamanho de suas peças em  $16 \text{ cm}^2$ . Portanto, cada processador, antes de ser liberado para venda, passa por um controle de qualidade, e um dos requisitos a ser verificado é o seu tamanho. Caso o processador tenha o tamanho padronizado, ele é liberado para venda, caso contrário, ele é descartado. Veja na Figura 2.4 como fica o algoritmo escrito nas duas formas de representação. Na estrutura condicional testamos a área da peça. Caso tenha  $16 \text{ cm}^2$ , ela será liberada para venda, porém se não tiver, será descartada.

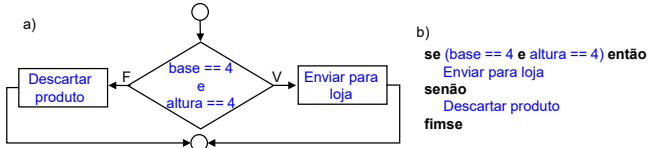
Figura 2.4 | Algoritmo para validar tamanho do processador



Fonte: elaborada pelo autor.

No algoritmo da Figura 2.4, verificamos se a área do processador é  $16 \text{ cm}^2$ , mas poderíamos fazer o teste realizando a validação através dos tamanhos da base e da altura; nesse caso devemos utilizar uma condição que envolve dois operadores: o operador de comparação ( $=$ ) e o operador lógico (E). Todos os operadores que você aprendeu na unidade anterior (relacionais matemáticos e lógicos) podem ser combinados dentro de um teste. A combinação desses recursos proporciona a construção de decisões mais complexas, pois podemos verificar diferentes valores em uma mesma variável, ou, ainda, testar valores de diferentes variáveis em uma mesma estrutura de decisão. Veja na Figura 2.5 o teste de qualidade para o tamanho do processador, comparando valores de diferentes variáveis dentro de uma mesma condição.

Figura 2.5 | Algoritmo com combinação de operadores



Fonte: elaborada pelo autor.

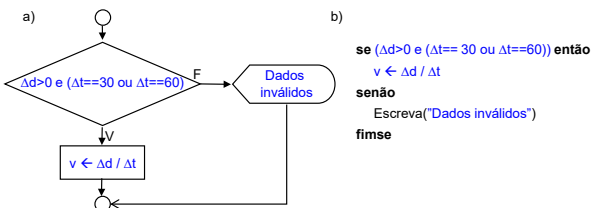


## Refleta

Agora que você já conhece a estrutura condicional composta usando operadores lógicos, qual seria o impacto na execução do algoritmo se o programador escrevesse “ou” em vez de “e” no momento de fazer a validação da base e da altura do algoritmo da Figura 2.5? Os resultados poderiam ser afetados por tal “erro”?

Vamos agora criar um algoritmo que combina os operadores lógicos “e/ou” para calcular a velocidade média de uma esteira de escoamento de produtos. A velocidade média de um objeto é uma razão entre seu deslocamento ( $\Delta x = x_2 - x_1$ ) e o tempo gasto durante esse deslocamento ( $\Delta t = t_2 - t_1$ ). Porém, antes de realizar o cálculo devemos verificar os dados de entrada do sistema. Devemos verificar se os valores de deslocamento (cm) e tempo (segundos) são positivos e, além disso, se a variação do tempo utilizado para o cálculo foi 30 ou 60 segundos (Figura 2.6). Caso as entradas sejam válidas, o cálculo é realizado, caso não sejam, é exibida uma mensagem informando que os dados fornecidos são inválidos. Para que a combinação de operadores lógicos funcionasse da maneira correta, foi necessário adicionar um par de parênteses interno para delimitar uma das comparações, pois o deslocamento obrigatoriamente tem que ser positivo, e a variação do tempo obrigatoriamente tem que ser 30 ou 60. Veja que 30 e 60 são comparados entre si, por isso usa-se o parêntese interno.

Figura 2.6 | Algoritmo para calcular velocidade média

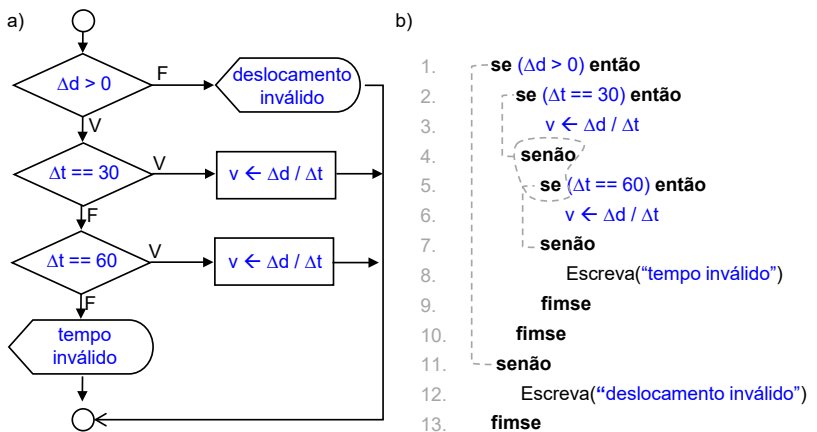


Fonte: elaborada pelo autor.



No algoritmo proposto na Figura 2.6, é possível saber qual dado informado (deslocamento ou tempo gasto) está fora dos padrões estabelecidos para o cálculo? A resposta é não. A estrutura de decisão foi construída para testar duas variáveis distintas de uma só vez, e a vantagem é que deixa o algoritmo mais legível, porém não é possível identificar qual dado informado está fora dos padrões. Para solucionar esse problema, podemos utilizar várias estruturas condicionais aninhadas. Na Figura 2.7 está representado um algoritmo para o cálculo da velocidade média da esteira, no entanto, agora usando condicionais aninhadas. Vamos analisar primeiro o algoritmo em forma de fluxograma (Figura 2.7 – a): o valor do deslocamento é testado e, caso seja um valor inválido, o usuário do sistema será avisado e o programa terminará. Mas caso o deslocamento seja válido, então o próximo passo é testar se a variação do tempo informado foi 30; caso seja verdadeiro, o cálculo é realizado e o programa termina, porém, se for inválido, então é testado se a variação do tempo informado foi 60; caso seja verdadeiro, o cálculo é realizado, mas caso seja inválido, o usuário é informado e o programa finaliza. Nessa representação, o desvio do fluxo sequencial do algoritmo fica evidente, pois caso a primeira condição seja falsa, dois blocos de códigos serão ignorados e não serão executados, funciona como se esses blocos nem mesmo existissem.

Figura 2.7 | Algoritmo com estruturas condicionais aninhadas



Fonte: elaborada pelo autor.

Na representação do algoritmo da velocidade média em pseudocódigo (Figura 2.7 – b), uma condição fica dentro da outra e, além disso, aparece uma nova combinação, o **senão se** nas linhas 4 e 5 (essa combinação pode ser usada quando um teste retorna falso e deseja-se fazer outro teste). O primeiro teste no algoritmo é feito na linha 1, e caso ele seja falso, o algoritmo dá um salto para a linha 11; mas caso ele seja verdadeiro, acontece o segundo teste (linha 2). Se o segundo teste for verdadeiro, então é realizado o cálculo, mas se o teste for falso, ele entra no bloco do **senão se**, que realiza o terceiro teste (linhas 4 e 5); caso verdadeiro, é realizado o cálculo; caso falso, o usuário é informado e o programa termina.



### Refleta

É possível reescrever o algoritmo para o cálculo da velocidade média da esteira, usando apenas duas estruturas condicionais em vez de três? Qual seria a vantagem em fazer dessa forma? E qual seria a desvantagem?

O funcionamento do pseudocódigo é idêntico ao fluxograma, porém quanto mais condicionais forem aninhadas, mais complexa ficará a interpretação do algoritmo. Por outro lado, se fôssemos gerar um relatório a partir dos resultados do algoritmo da Figura 2.7, utilizando as estruturas aninhadas, conseguiríamos informar se a velocidade média foi calculada usando a variação de tempo de 30 segundos ou 60 segundos, o que não seria possível se escrito de outra forma. Ou seja, embora o algoritmo fique mais complexo, se pensarmos na análise de dados estruturas aninhadas, podemos obter informações mais detalhadas.



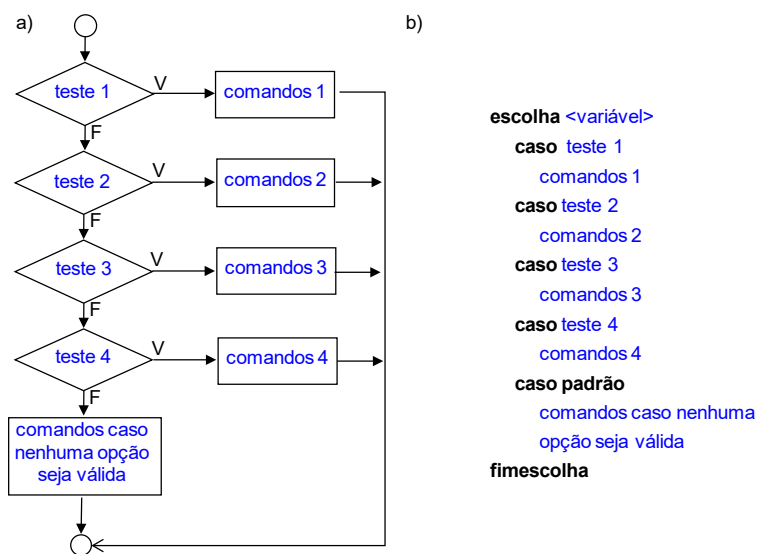
### Assimile

Não existe um limite para a quantidade de estruturas condicionais que podem ser aninhadas, mas é uma boa prática de programação utilizar até três condicionais aninhadas. A partir desse número, utiliza-se uma nova estrutura condicional, a estrutura de seleção de caso.

O terceiro tipo de estrutura condicional que vamos aprender possui uma estrutura diferente dos demais vistos até o momento. Essa estrutura é chamada de seleção de caso, pois a partir de uma

série de casos (opções), um deve ser escolhido. Alguns autores utilizam a nomenclatura múltipla escolha para essa estrutura, pois temos múltiplas opções (MANZANO; MATOS; LOURENÇO, 2015). Observe o fluxograma na Figura 2.8 – a: um teste é feito e, caso falso, faz-se o segundo teste, e assim por diante. Caso nenhum teste resulte em verdadeiro, existe um conjunto de comandos que será realizado. A utilização desse tipo de estrutura é mais interessante quando a forma de representação é o pseudocódigo (Figura 2.8 – b), o bloco de testes inicia-se escolhendo a variável que será testada (somente uma é possível), em seguida escrevem-se as possíveis opções (casos) para essa variável através do comando: “caso teste”.

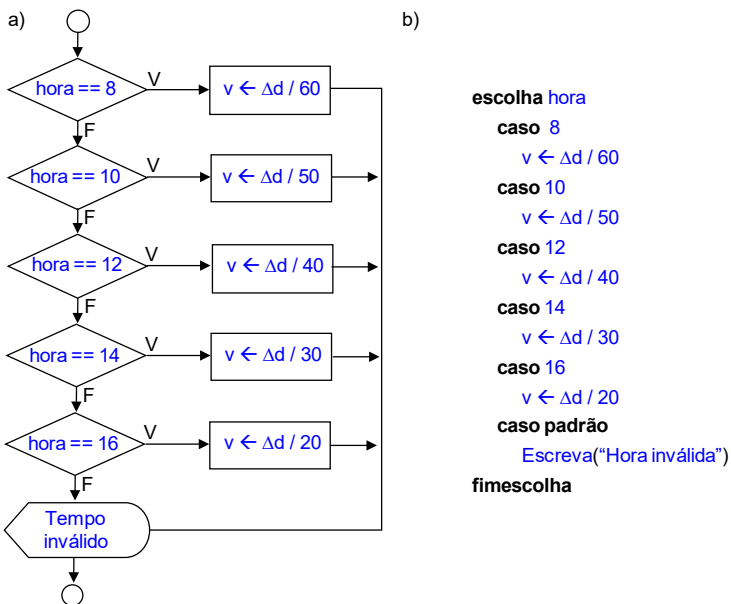
Figura 2.8 | Algoritmo seleção de casos



Fonte: elaborada pelo autor.

Para exemplificar a seleção de caso, imagine que seu chefe decidiu mapear a velocidade da esteira durante todo o dia e para isso pediu que você alterasse o algoritmo para que a velocidade fosse testada às 8h, 10h, 12h, 14h e 16h. Além disso, foi pedido que, para cada hora testada, fosse utilizada uma variação do tempo ( $\Delta t$ ) diferente: às 8h  $\Delta t = 60$ , às 10h  $\Delta t = 50$ , às 12h  $\Delta t = 40$ , às 14h  $\Delta t = 30$  e às 16h  $\Delta t = 30$ . Para atender a esse pedido, a melhor alternativa é utilizar a seleção de casos. Veja na Figura 2.9 que a variável escolhida para ser testada é “hora”, em seguida são listadas, através de cada caso, as opções.

Figura 2.9 | Cálculo da velocidade média usando a seleção de caso



Fonte: elaborada pelo autor.

Na Figura 2.9 – b, quando utilizamos a palavra-chave “caso”, estamos testando se o valor da variável, escolhida para ser testada, é igual ao valor *candidato*. Não seria possível testar se é maior ou menor, por exemplo, para isso teríamos que aninhar outras condições dentro do caso. A estrutura condicional de seleção de caso tem a vantagem de testar uma variável diversas vezes de uma maneira simples, porém tal estrutura tem um inconveniente: perceba que o único teste que conseguimos fazer é verificar a igualdade da variável com um determinado valor, ou seja, somente condições de seleção do tipo “igual a” (MANZANO; MATOS; LOURENÇO, 2015).



**Pesquise mais**

Para reforçar o conteúdo visto nesta unidade, indico os vídeos 7 e 8 da série “Curso de Lógica de Programação” do professor Gustavo Guanabara, disponível em: <<https://goo.gl/VcqtqDC>>. Acesso em: 26 out. 2017.

## Sem medo de errar

Como funcionário da Kro Engenharias, responsável por desenvolver algoritmos para validar dados de entrada para o subsistema de cálculo, foi pedido a você criar algoritmos para verificar as entradas de comprimento, tempo e massa de acordo com o Sistema Internacional de Medidas (SI). Para o comprimento, o sistema foi padronizado para aceitar a entrada em metro (m), para o tempo foi padronizado o segundo (s), e para massa foi padronizado o quilograma (kg).

O formulário do sistema da Kro Engenharias foi construído há certo tempo e foi projetado para aceitar as entradas em metro, ou centímetro, ou milímetro, conforme mostra a Figura 2.10. Porém, com a nova padronização, você terá de verificar no algoritmo se o usuário escolheu a opção metro e, caso não tenha escolhido, você terá de informá-lo que a opção escolhida não é mais válida.

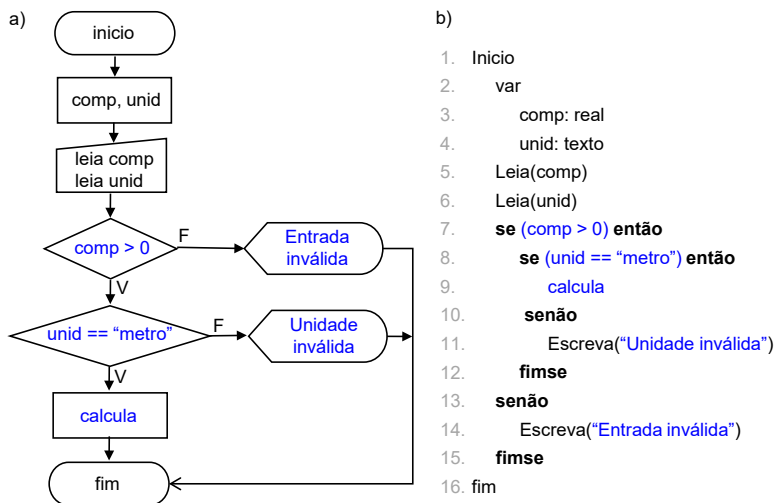
Figura 2.10 | Captura de tela de parte do formulário da Kro Engenharias

Digite o comprimento:   Metro  Centímetro  Milímetro

Fonte: elaborada pelo autor.

Para resolver o problema, a melhor opção é utilizar uma estrutura condicional composta, conforme algoritmo da Figura 2.11.

Figura 2.11 | Algoritmo para validar entrada de comprimento em metro



Fonte: elaborada pelo autor.

Vamos entender o funcionamento analisando o pseudocódigo: todo algoritmo deve ter um início e um fim (linhas 1 e 16). Após o início, é necessário declarar as variáveis que serão usadas (linhas 3 e 4) e ler os valores que são informados pelo usuário (linhas 5 e 6). A segunda parte do algoritmo consiste em realizar os testes. Iniciamos testando se o comprimento informado é um número positivo (linha 7); caso seja falso, a execução salta para o “senão” na linha 13 e avisa o usuário que a entrada não é válida. Mas caso seja verdadeiro, a segunda variável é testada, verificando se ela é igual a “metro” (linha 8). Caso seja falso, a execução salta para o “senão” na linha 10 e informa o usuário que a unidade é inválida. Mas caso seja verdadeiro, o cálculo é realizado.

Agora é com você! Baseado no algoritmo do comprimento, escreva um algoritmo para verificar se o tempo escolhido é positivo e está na unidade segundo, e outro para verificar se a massa informada é positiva e está na unidade quilograma.

## Avançando na prática

### Algoritmo para calcular o rendimento de uma máquina

#### Descrição da situação-problema

Em uma empresa é comum calcular o rendimento dos equipamentos para controle interno. O rendimento está associado ao tempo gasto para realizar um determinado trabalho. Considere duas máquinas, de diferentes fornecedores, que modelam peças a partir de desenhos tridimensionais feitos por um software. A máquina 1 modela  $x$  peças em 1 hora, e a máquina 2 modela  $2x$  peças no mesmo tempo. Nesse caso, dizemos que a máquina 1 tem uma potência menor que a segunda. A unidade de potência no SI é o joule por segundo. Essa unidade é usada com tanta frequência que recebeu um nome especial, o watt (W), em homenagem a James Watt, cuja contribuição foi fundamental para o aumento da potência das máquinas a vapor. (HALLIDAY; RESNICK, 2015). Podemos medir a potência de uma máquina ( $P$ ) usando como informações a corrente elétrica ( $i$ ) e a tensão da rede ( $U$ ), através da fórmula  $P = U \cdot i$ . No Brasil, nossa rede elétrica está configurada para trabalhar com as tensões 110 ou 220 volts. Também sabemos que a corrente elétrica é um

valor absoluto, ou seja, positivo. Escreva um algoritmo que calcula a potência de uma determinada máquina.

## Resolução da situação-problema

Como vimos, antes de realizar cálculos devemos verificar se as informações fornecidas são válidas. Então, antes de calcular a potência, o algoritmo deve verificar se os dados de entrada (tensão e corrente) são dados válidos.

Figura 2.12 | Algoritmo para calcular a potência de uma máquina

```
1. inicio
2. var
3. inteiro: p, u, i
4. Leia(u)
5. Leia(i)
6. se (u == 110 ou u == 220) então
7. se (i > 0) então
8. $p \leftarrow u * i$
9. senão
10. Escreva("Corrente elétrica inválida")
11. fimse
12. senão
13. Escreva("Tensão inválida")
14. fimse
15. fim
```

Fonte: elaborada pelo autor.

Agora é com você! Transforme o pseudocódigo em um fluxograma.

## Faça valer a pena

**1.** Os computadores são dispositivos binários, isso significa que aceitam apenas dois valores: 0 e 1. Para representar os caracteres alfabéticos e outros caracteres, como os acentos, internamente é feita uma conversão do caractere para um valor numérico, em seguida esse valor numérico é convertido em um número binário. A conversão do caractere para o valor numérico é feita de acordo com determinado padrão, e um dos padrões que foi largamente usado e ainda é encontrado hoje é a tabela ASCII (American Standard Code for Information Interchange).

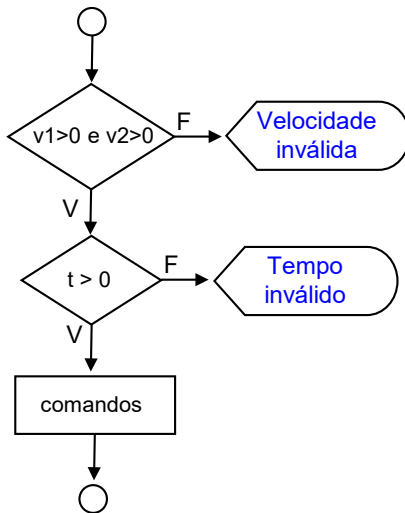
Na tabela ASCII existe um código numérico para cada letra do alfabeto, além dos dígitos numéricos e caracteres especiais. Sabendo que os dígitos de 0 a 9 são representados com os códigos de 48 a 57 na tabela, qual seria a melhor estrutura de decisão para um algoritmo que gera um relatório a partir das opções de um menu que vão de 0 a 9?

- a) Estrutura de decisão condicional.
- b) Estrutura condicional simples.
- c) Estrutura condicional aninhada.
- d) Estrutura condicional composta.
- e) Seleção de casos.

2. Um objeto em movimento pode aumentar ou diminuir sua velocidade, nesse caso, diz-se que o objeto sofreu uma aceleração ou desaceleração. Podemos calcular a aceleração média de um objetivo dividindo sua variação de velocidade por um determinado tempo.

O fluxograma da Figura 2.13 verifica se o valor das velocidades iniciais e finais e o tempo informados são válidos. Escolha qual pseudocódigo representa o fluxograma.

Figura 2.13 | Fluxograma para validar velocidade e tempo



Fonte: elaborada pelo autor.

a) Pseudocódigo 1

```

se v1 e v2 > 0 então
 se t > 0
 Comandos
 senão
 Escreva("Tempo inválido")
 fimse
fimse

```



## b) Pseudocódigo 2

```
se v1>0 e v2>0 e t>0 então
 Comandos
senão
 Escreva("Velocidade inválida")
 Escreva("Tempo inválido")
fimse
```

## c) Pseudocódigo 3

```
se v1>0 e v2>0 então
 se t > 0
 Comandos
 senão
 Escreva("Tempo inválido")
 fimse
senão
 Escreva("Velocidade inválida")
fimse
```

## d) Pseudocódigo 4

```
se v1>0 e v2>0 então
 Escreva("Velocidade inválida")
senão
 se t > 0
 Comandos
 senão
 Escreva("Tempo inválido")
 fimse
fimse
```

## e) Pseudocódigo 5

```
se v1>0 e v2>0 então
 se t > 0
 Comandos
 senão
 Escreva("Velocidade inválida")
 fimse
senão
 Escreva("Tempo inválido")
fimse
```

3. Os algoritmos são escritos com o objetivo de resolver problemas, e resolver problemas implica tomar decisões. Para a construção de algoritmos, as estruturas condicionais são utilizadas para realizar os testes. A partir de um teste, um conjunto de ações pode ser executado caso o teste resulte em verdadeiro, e um outro conjunto de ações pode ser executado caso o teste resulte em falso. Esse tipo de estrutura é chamado de estrutura condicional composta.

Considerando o algoritmo representado em pseudocódigo em cada um dos itens, e os seguintes valores para as variáveis:  $a = 5$ ,  $b = 3$ ,  $c = 5$ ,  $d = -1$ , verifique se o resultado descrito para o código é uma afirmação correta ou incorreta. Em seguida, escolha a alternativa correta.

- I. **se  $a > b$  então**  
    Escreva("Maior número é a")  
**senão**  
    Escreva("Maior número é b")  
**fimse**

O teste retorna verdadeiro e escreve "Maior número é a".

- II. **se  $a > c$  então**  
    Escreva("Maior número é a")  
**senão**  
    Escreva("Maior número é c")  
**fimse**

O teste retorna falso e escreve "Maior número é a".

- III. **se  $a <= d$  então**  
    Escreva("Menor número é a")  
**senão**  
    Escreva("Menor número é d")  
**fimse**

O teste retorna falso e escreve "Menor número é a".

- a) As três alternativas estão corretas.
- b) Somente a alternativa I está correta.
- c) Somente as alternativas II e III estão corretas.
- d) Somente a alternativa II está incorreta.
- e) Somente a alternativa III está incorreta.

## Seção 2.2

### Estruturas de repetição

#### Diálogo aberto

Caro(a) estudante, bem-vindo(a) a mais uma seção no estudo dos elementos de algoritmos. Nessa etapa, aprenderemos mais um elemento essencial para o desenvolvimento de algoritmos. Essa nova estrutura nos possibilitará escolher certos trechos do algoritmo para serem executados um determinado número de vezes e não mais só uma vez.

Os automóveis e uma grande quantidade de máquinas e equipamentos utilizados nas mais diversas áreas são fabricados em uma linha de produção com grande parte do processo automatizado. Nesse processo, máquinas realizam uma mesma tarefa várias vezes. Por exemplo, imagine uma máquina que solda certa parte de um carro. Quem determina quantas vezes ela terá de fazer o mesmo trabalho? Será que tem alguém atrás da máquina que fica apertando o botão toda vez que é para soldar? Como a máquina sabe quando parar? Todas essas questões ficarão claras com o estudo desta seção.

Para cumprir seu trabalho na Kro Engenharias, na seção anterior você desenvolveu algoritmos para validar alguns valores de entrada referentes às unidades de comprimento, tempo e massa. Agora você deverá escrever um algoritmo que faça a conversão de uma série de entradas de comprimento para a unidade quilômetro. Na Figura 2.14, temos um captura de tela de parte de um formulário da Kro Engenharias usado para cadastrar e gerar relatório acerca dos gastos de combustível de cada automóvel da empresa durante um dia. A empresa possui 15 carros que ficam à disposição de seus funcionários, e ao final do dia é feito um controle da quantidade de quilômetros que os carros percorreram naquele dia. O sistema permite que os dados sejam inseridos em quilômetros, metros ou, ainda, em milhas porém o cálculo do controle de combustível é feito usando a unidade de quilômetro. Portanto, caso seja escolhido metro ou milha, seu algoritmo deverá fazer a conversão dos valores para a unidade-padrão.

Figura 2.14 | Captura de tela do formulário da Kro Engenharias

| Carro     | Valor |
|-----------|-------|
| Carro 1:  | 123   |
| Carro 2:  | 432   |
| Carro 3:  | 222   |
| Carro 4:  | 415   |
| Carro 5:  | 360   |
| Carro 6:  | 90    |
| Carro 7:  | 0     |
| Carro 8:  | 120   |
| Carro 9:  | 110   |
| Carro 10: | 0     |
| Carro 11: | 134   |
| Carro 12: | 245   |
| Carro 13: | 234   |
| Carro 14: | 199   |
| Carro 15: | 0     |

Escolha a unidade de medida:

- Quilômetros
- Metros
- Milhas

Calcular

Fonte: elaborada pelo autor.

Para cumprir esse desafio, você aprenderá nesta seção como criar estruturas de repetição. Existem três tipos de estruturas de repetição, a escolha de qual utilizar dependerá das variáveis que alimentam a estrutura de repetição, bem como do resultado que se deseja alcançar, embora algumas vezes as diferentes estruturas possam chegar ao mesmo resultado.

Bons estudos!

## Não pode faltar

Segundo a agência americana Nasa, no final da década de 1960 o homem pisou pela primeira vez na Lua (NASA, 2017). Você consegue imaginar quantos cálculos foram feitos e refeitos inúmeras vezes para se chegar às informações necessárias para alcançar tal feito? Certamente não é possível quantificar essas repetições sem o uso de um sistema computacional. E nesse sistema vários cálculos tiveram de ser feitos sobre os mesmos conjuntos de números, e muitas vezes o mesmo conjunto de cálculos era aplicado a conjuntos diferentes. Os cálculos, em outras palavras, eram repetidos. Vamos, a partir de agora, conhecer as estruturas de repetição.

Na seção anterior criamos vários algoritmos que, a partir de uma decisão, executavam uma única vez determinada ação, porém muitas vezes precisamos que, a partir de uma decisão, uma ação seja realizada repetidas vezes. Para entendermos essa situação, vamos usar como exemplo o algoritmo que criamos na seção anterior, que verificava se o valor escolhido pelo usuário era positivo e se a unidade era metro. Caso ambas entradas fossem válidas, o algoritmo realizava um cálculo, mas se não fosse, o programa simplesmente encerrava. Certamente o usuário ficaria insatisfeito com o algoritmo se toda vez que ele digitasse um número negativo, por acidente, tivesse que abrir novamente o programa. Essa situação pode ser contornada de uma maneira simples, utilizando uma estrutura de repetição.

Estrutura de repetição é um elemento de algoritmo que permite escolher certos trechos de códigos para serem executados de forma repetida. Embora seja um recurso poderoso, deve ser usado da forma correta, senão pode causar um erro de execução, pois a estrutura pode entrar em uma repetição infinita (PIVA JUNIOR et al., 2012). A chave para evitar esse erro é realizar um teste que determinará quando a repetição deve parar.



### Assimile

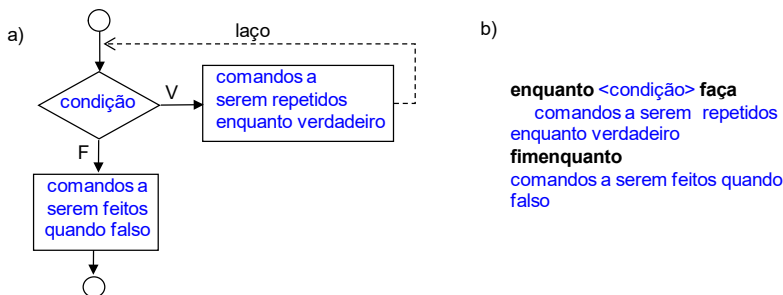
Através de uma estrutura de repetição, trechos de código podem ser executados repetidas vezes. Para evitar que o trecho seja executado infinitas vezes, utiliza-se um teste para delimitar a quantidade de repetições. Assim, um trecho será repetido enquanto o teste for verdadeiro. Quando for falso, a repetição se encerra.

Em uma estrutura de repetição sempre existirá, no mínimo, uma variável para controlar as repetições. Tal variável será usada pelo teste condicional que controla o ciclo de repetição. Essa variável poderá, em alguns casos, funcionar como um contador para a quantidade de repetições, em outros casos, será necessário adicionar uma segunda variável para fazer a contagem de repetições. Esse teste, além de determinar quando a repetição deve parar, também classifica o tipo de estrutura de repetição. Iremos agora conhecer os tipos de estruturas de repetição, todos os algoritmos de agora em diante serão identificados por um nome na primeira linha.

## Estrutura de repetição com teste no início

O primeiro tipo que vamos estudar é a estrutura de repetição com teste no início. Nesse tipo, uma condição é testada antes de o ciclo de repetição começar. Caso a condição seja verdadeira, o ciclo inicia-se e só para quando a condição não for mais verdadeira. Tal estrutura está representada em forma de fluxograma e pseudocódigo na Figura 2.15. No fluxograma, a repetição é representada por uma seta que sai do bloco "comandos a ser repetidos" e volta para a condição a ser testada, criando um laço (*loop*) de repetição. No pseudocódigo é usada a palavra "enquanto" para realizar o teste e a palavra "faça" para iniciar os comandos do laço de repetição. Podemos interpretar da seguinte forma: "Enquanto a condição for verdadeira, faça os seguintes comandos dentro do bloco". Perceba que a repetição do bloco só será interrompida quando a condição for falsa.

Figura 2.15 | Estrutura de repetição com teste no início



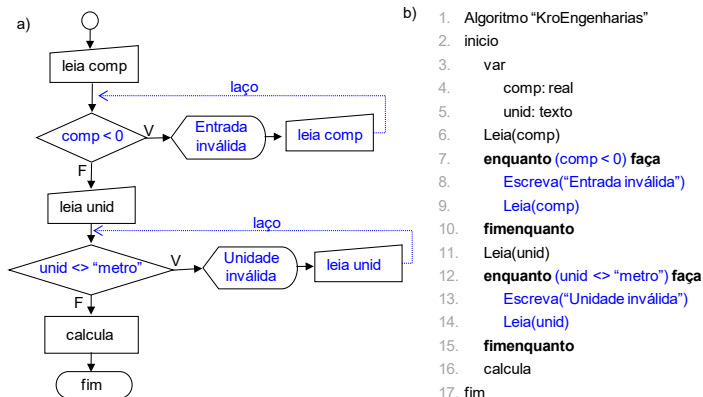
Fonte: elaborada pelo autor.



Para melhor compreendermos a estrutura de repetição com teste no início, vamos retomar um algoritmo feito na seção anterior para a Kro Engenharias. Na seção anterior, você implementou um algoritmo para validar os dados de entrada para a empresa, esse algoritmo recebe os valores referentes ao comprimento de algum objeto e tais valores devem ser positivos e estar na unidade metro, caso contrário o programa se encerra. Vamos alterar o algoritmo para que, em vez de encerrar a execução com um valor negativo ou uma unidade diferente de metro, o programa volte a pedir que o usuário digite novamente o valor. Na Figura 2.16 – a, temos o fluxograma aprimorado. Veja que só existe uma maneira de o algoritmo chegar ao fim da execução, que é após realizar o cálculo com os valores dentro dos padrões estabelecidos, pois caso os valores sejam inválidos, existem laços com comandos que leem novos valores e os testam. Nesse algoritmo, a variável que está sendo utilizada para controlar o laço de repetição é o próprio valor do comprimento e da unidade de medida.

No pseudocódigo, temos o algoritmo completo (sem conectores), o programa inicia-se e o primeiro passo é declarar as variáveis com seus respectivos tipos. Em seguida, é lido o valor do comprimento e verificada sua validade. Caso não seja válido, o usuário é informado, um novo valor é lido e posteriormente testado, pois está dentro do laço (linhas 7, 8 e 9). Após o usuário digitar um valor válido para comprimento, então o valor da unidade é lido e analisado (linhas 11 e 12). Caso seja inválido, o usuário é informado, um novo valor é lido e posteriormente testado dentro do laço (linhas 12, 13 e 14). Somente quando o usuário digitar um valor válido para unidade será feito o cálculo e o programa, encerrado.

Figura 2.16 | Algoritmo aprimorado para Kro Engenharias



Fonte:elaborada pelo autor.



O que poderia ser inserido no algoritmo da Kro Engenharias para contar quantas vezes o usuário digitou um valor inválido?

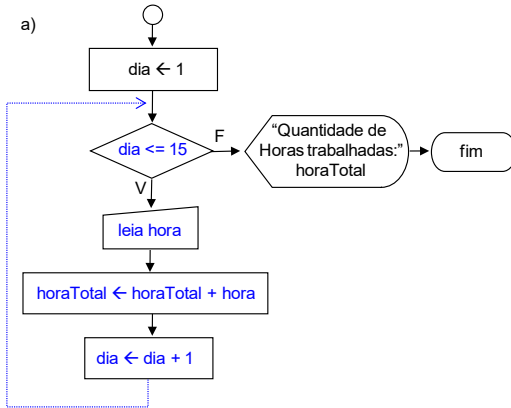
O algoritmo aprimorado criado para a Kro Engenharias (Figura 2.16) apresenta um uso clássico da estrutura de repetição com teste no início, pois não sabemos quantas vezes a repetição será executada (isso dependerá das entradas digitadas pelo usuário). Sendo assim, a repetição pode ocorrer zero vezes ou várias vezes: veja que não existe um limite inicial e final.

Embora o uso clássico da repetição com teste no início seja para os casos em que não se sabe quantas repetições serão realizadas, tal estrutura também pode ser usada para os casos em que se sabe de antemão quantas repetições serão feitas. Para entendermos esse novo mecanismo, vamos escrever um algoritmo que soma a quantidade de horas trabalhadas por um funcionário nos quinze primeiros dias do mês. Na Figura 2.17 – a, a variável *dia* inicia a execução valendo 1 (pois representa o primeiro dia do mês) e em seguida o teste ( $dia \leq 15$ ) é feito. Caso seja verdadeiro, é lido o valor da hora para aquele determinado dia e a hora total é atualizada somando o novo valor ( $horaTotal = horaTotal + hora$ ). Em seguida, o dia é atualizado e a execução volta para o teste ( $dia \leq 15$ ). Nesse exemplo, a variável que controla o laço de repetição é inicializada antes do início do teste, e para cada execução do bloco a variável de controle é atualizada. Caso isso não acontecesse, teríamos um erro de execução infinita.

O funcionamento no pseudocódigo (Figura 2.17 – b) é idêntico: a variável *dia* é inicializada com o valor 1 (linha 6), em seguida é usado o comando **"enquanto-faça"** para realizar o teste na estrutura de repetição (linha 7). Devemos ler da seguinte forma: "Enquanto o valor da variável dia for menor ou igual a 15, faça os três comandos seguintes, quando o valor da variável dia for maior que quinze, então imprima o total de horas acumuladas". O programa somente encerrará após terem sido acumuladas as horas de trabalho dos 15 primeiros dias do mês.



Figura 2.17 | Algoritmo para somar horas trabalhadas



- b)
1. Algoritmo "HorasTrabalhadas"
  2. inicio
  3. var
  4. horaTotal, hora: real
  5. dia: inteiro
  6. dia ← 1;
  7. enquanto (dia <= 15) faça
  8. Leia(hora)
  9. horaTotal ← horaTotal + hora
  10. dia ← dia + 1
  11. fimenquanto
  12. Escreva("Quantidade de horas trabalhadas: ", horaTotal)
  13. fim

Fonte: elaborada pelo autor.



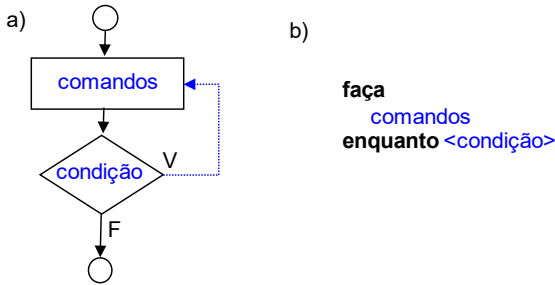
Assimile

As estruturas de repetição com teste no início também são conhecidas como laço condicional pré-teste (MANZANO, 2010). Ao utilizar essa estrutura, todos os comandos dentro do bloco que se iniciam com "enquanto-faça" e finalizam com o "fimenquanto" podem ser executadas 0 ou  $n$  vezes.

## Estrutura de repetição com teste no final

O segundo tipo de estrutura de repetição é caracterizado por fazer o teste de controle no final do bloco de comandos. Nessa estrutura tem-se um bloco que inicia com o comando "faça" e termina com o teste "enquanto" (Figura 2.18). Tal estrutura também é conhecida como laço condicional pós-teste (MANZANO, 2010).

Figura 2.18 | Estrutura de repetição com teste no final

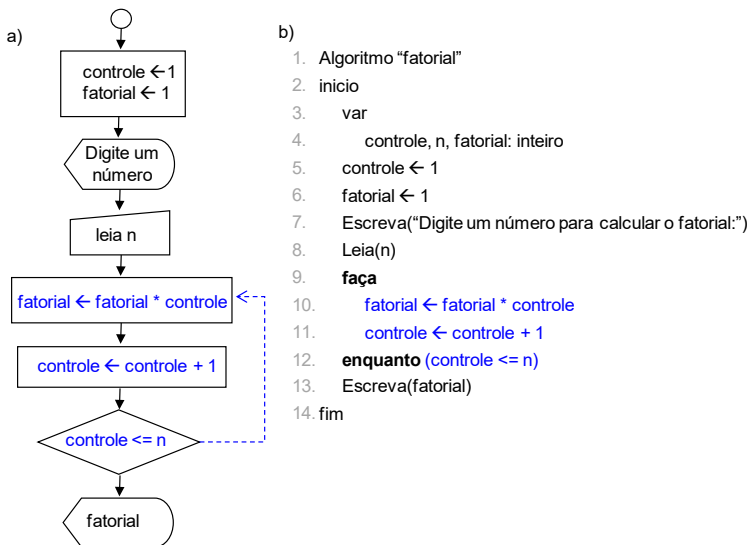


Fonte: elaborada pelo autor.

Mas, afinal, qual a diferença entre criar uma estrutura de repetição com teste no início ou teste no final? A diferença entre as estruturas com pré-teste e pós-teste é que, nas estruturas com pré-teste, o bloco de comandos pode repetir **0 ou n vezes**. Já na estrutura com pós-teste, o bloco de comandos irá repetir **1 ou n vezes**, ou seja, nessa estrutura os comandos serão executados pelo menos uma vez.

Para entendermos como funciona o laço condicional com pós-teste, vamos escrever um algoritmo que calcula o fatorial de um número. O fatorial é calculado somente para os números naturais, ou seja, números inteiros e positivos. Dado um número natural  $n$ , seu fatorial é o produto de todos os seus antecessores até 1, representamos o fatorial de um número como  $n!$ . Então, fatorial de 5! é  $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ . No algoritmo da Figura 2.19 – b, após a declaração das variáveis (linha 4), as variáveis *controle* e *fatorial* são inicializadas com o valor 1 (linhas 5 e 6). O próximo passo é pedir e guardar o valor digitado pelo usuário (linhas 7 e 8), para então entrar no bloco de repetição que começa com o comando “**faça**” (linha 9) e termina com o teste (linha 12). Veja que o bloco de comandos a ser repetido (linhas 10 e 11) vem antes do teste condicional (linha 12). E, por fim, o programa exibe o resultado do fatorial para o número escolhido. A variável *controle* é utilizada para evitar o laço infinito, e a cada iteração ela é incrementada em 1 ( $\text{controle} = \text{controle} + 1$ ). Além de controlar a repetição, a própria variável é utilizada no cálculo do fatorial, pois ela irá variar de 1 até  $n$ .

Figura 2.19 | Algoritmo para calcular o fatorial de um número



Fonte: elaborada pelo autor.



Refleta

O algoritmo da Figura 2.19 calcula o fatorial de um número natural, quantas vezes o bloco de repetições iria ser executado e qual seria o resultado, caso o número escolhido pelo usuário fosse 0?

E se, em vez de fazer o teste no final, alterássemos a estrutura de repetição para ter o teste no início, quantas vezes o bloco de repetições iria ser executado e qual seria o resultado caso o número escolhido pelo usuário fosse 0?

As estruturas de repetição com teste no início ou no final muitas vezes terão o mesmo resultado; nesse caso, cabe ao programador escolher qual estrutura usar.

### Estrutura de repetição com variável de controle

Existe ainda uma terceira estrutura de repetição com variáveis de controle, ou ainda, laço com variável de controle. Essa estrutura deve ser utilizada sempre que se conhece o início e o final das repetições, ou seja, sempre que se sabe quantas vezes o bloco irá se repetir antes de a execução acontecer.

Na estrutura de repetição com variável de controle, escolhemos uma variável para controlar as repetições (item 1, Figura 2.20), um valor para marcar o início da contagem (item 2, Figura 2. 20), outro valor para marcar o final da contagem de repetições (item 3, Figura 2.20), e ainda podemos controlar como ocorre essa contagem (passo), se será de 1 em 1, de 2 em 2, de 3 em 3, e assim por diante (item 4, Figura 2.20). Nesse tipo de estrutura, todos os valores envolvidos no controle de repetição ficam na mesma linha, no mesmo comando. Na Figura 2.20 – b podemos ver um exemplo da declaração dessa estrutura de repetição. A variável escolhida para controlar as repetições foi *x*, ela inicia a execução com valor 1 e irá executar por 5 vezes, pois o valor final de *x* será 5, e a contagem se dará de 1 em 1, pois o passo escolhido foi 1. O resultado desse algoritmo seria a impressão na tela da sequência: 1 2 3 4 5.

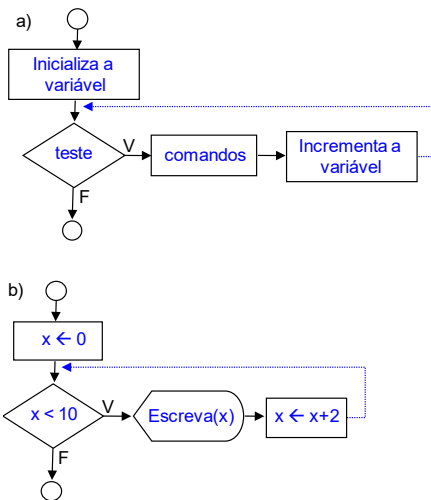
Figura 2.20 | Pseudocódigo para estrutura de repetição com variável de controle

- a)
- ①                      ②                      ③                      ④
- para(<variável> de <valor-inicial> até <valor-final> passo <incremento>) faça**  
**comandos**  
**fimpara**
- b)
- para(x de 1 até 5 passo 1) faça**  
**Escreva(x)**  
**fimpara**

Fonte: elaborada pelo autor.

Embora exista o fluxograma para representar a estrutura de repetição com variável de controle (Figura 2.21), a forma de representação mais usada para escrever tal estrutura é o pseudocódigo. Na Figura 2.21 – b, temos um exemplo do fluxograma: primeiro é atribuído o valor zero à variável *x*, em seguida é testado se o valor de *x* é menor que 10; se for verdade, o valor de *x* será escrito e *x* será incrementado em 2, e o fluxo volta ao teste de *x*. O resultado desse algoritmo seria a impressão na tela dos valores: 0, 2, 4, 6, 8. Note que o valor de *x* altera de 2 em 2, pois esse foi o critério usado para o incremento da variável de controle. Além disso, o valor 10 não é impresso, pois o teste é verdade somente para valores de *x* menores que 10 e não igual a 10.

Figura 2.21 | Fluxograma estrutura de repetição com variável de controle



Fonte: elaborada pelo autor.



## Exemplificando

Para vermos o funcionamento de um algoritmo usando a estrutura de repetição com variável de controle, vamos retomar o algoritmo que soma a quantidade de horas trabalhadas por um funcionário. Na implementação feita anteriormente com a estrutura de repetição com teste no início, era possível somar as horas trabalhadas nos 15 primeiros dias do mês. Vamos alterar o algoritmo para o funcionário escolher o intervalo de dias que ele deseja saber suas horas trabalhadas (Figura 2.22).

O primeiro passo no algoritmo é criar todas as variáveis necessárias para a execução (linhas 4 e 5), em seguida é pedido ao usuário para informar o início e o final do período para efetuar a soma das horas (linhas 6, 7, 8 e 9). De posse dessas informações, somente uma estrutura de repetição com variável de controle nos permite controlar o início e o final das repetições, de uma forma tão simples, usando apenas uma linha comando. Na linha 10 temos o comando que usa a variável *dia*, que vai do *diaInicial* até o *diaFinal*, aumentando de 1 em 1 dia (passo), e para cada dia é pedido para o usuário informar quantas horas ele trabalhou naquele determinado dia (linhas 11 e 12) e em seguida o valor do dia é acumulado com os demais dias na variável *horaTotal* (linha 13). Quando a repetição acabar, será impresso na tela o total de horas que ele trabalhou naquele determinado período.

Figura 2.22 | Algoritmo para somar horas trabalhadas

```
1. Algoritmo "horasTrabalhadas"
2. inicio
3. var
4. horaTotal, hora: real
5. dia, diaInicial, diaFinal: inteiro
6. Escreva("Digite o início do período")
7. Leia(diaInicial)
8. Escreva("Digite o final do período")
9. Leia(diaFinal)
10. para (dia de diaInicial até diaFinal passo 1) faça
11. Escreva("Digite a quantidade de horas do dia")
12. Leia(hora)
13. horaTotal ← horaTotal + hora
14. fimpara
15. Escreva("Quantidade de horas trabalhadas: ", horaTotal)
16. fim
```

Fonte: elaborada pelo autor.

Veja no algoritmo da Figura 2.22 como usamos a estrutura de repetição com variável de controle: ficou simples controlar o início e o final da repetição, além disso, conhece-se de antemão quantas vezes a repetição acontecerá. Outro detalhe importante é o controle sobre o passo, ou seja, podemos controlar como a variável de controle irá aumentar ou até mesmo diminuir. Se quiséssemos somar as horas trabalhadas nos dias pares, poderíamos iniciar a contagem no dia zero e aumentar o passo de 2 em 2; ou se quiséssemos somar os dias ímpares, bastava iniciar no dia 1, e também aumentar de 2 em 2.

As três estruturas de repetição apresentadas nessa seção compõem todas as opções que você tem para repetir um trecho do código. É possível que diferentes estruturas cheguem no mesmo resultado, então a escolha de qual usar depende do que você possui de informação antes de iniciar a repetição, e como a repetição pode ser mais bem controlada.



Os laços de repetição com teste no início ou no fim (enquanto... faça / faça...enquanto) devem ser utilizados quando não se conhece de antemão o número de vezes que determinado bloco de comandos deve ser executado. Já as estruturas de repetição com variável de controle devem ser usadas sempre que se sabe quantas repetições serão feitas, em outras palavras, somente quando se conhece o início e o final das repetições.



A habilidade de criar soluções através de algoritmos só é aprimorada com o exercício de inúmeros algoritmos. Portanto, faça os exemplos deste livro e outros em endereços na internet. Nesses endereços, você encontrará explicações e exemplos para:

Estrutura de repetição com teste no início: <<http://www.dicasdeprogramacao.com.br/estrutura-de-repeticao-enquanto>>; Acesso em: 11 nov. 2017.

Com teste no final: <<http://www.dicasdeprogramacao.com.br/estrutura-de-repeticao-repita-ate>>; Acesso em: 11 nov. 2017.

Com variável de controle: <<http://www.dicasdeprogramacao.com.br/estrutura-de-repeticao-para>>. Acesso em: 11 nov. 2017.

## Sem medo de errar

Como funcionário da Kro Engenharia, você foi designado a desenvolver um algoritmo que, antes de realizar cálculos que envolvam comprimento, garanta que os dados estejam na unidade de metro. Esse algoritmo é essencial para a padronização da comunicação entre os subsistemas e também para garantir que todos os resultados estejam em uma mesma unidade de medida. O algoritmo deverá ler a quantidade de quilômetros percorridos por cada um dos 15 carros da empresa durante um dia e ao final informar a quantidade total percorrida na unidade de quilômetro. Como o formulário da Kro Engenharia permite ao funcionário informar os dados em quilômetros, metros ou milhas, o algoritmo deve verificar qual foi a unidade escolhida e fazer as conversões necessárias.

No algoritmo da Figura 2.23 está implementado o código para controle da quilometragem diária dos carros da Kro Engenharias.

1. Da linha 1 até 6, o programa é iniciado e as variáveis são declaradas.

2. Para ler a distância percorrida de 15 carros, foi usada uma estrutura de repetição com variável de controle, pois é conhecida a quantidade de repetições antes da execução do programa. O início da estrutura de repetição ocorre na linha 7, a variável de controle é *carro* que inicia com valor 1 e vai até 15, aumentando de 1 em 1 (passo).

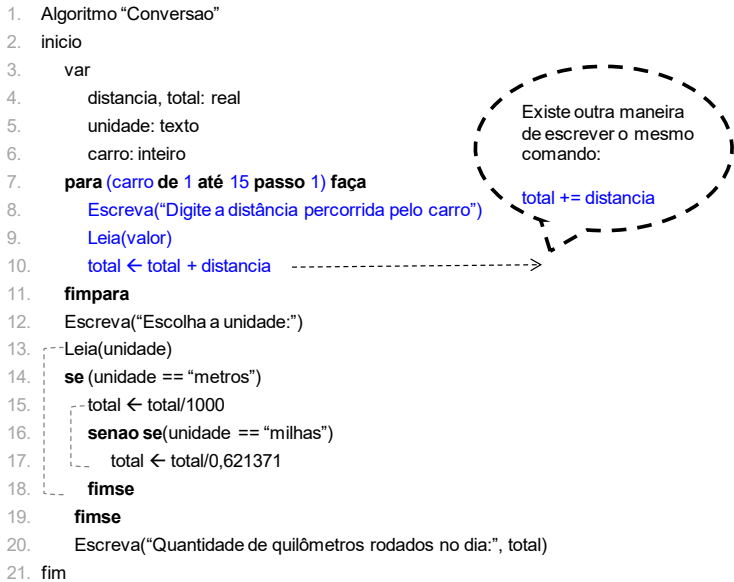
3. O bloco de repetição vai da linha 7 até 11, contendo 3 comandos; o primeiro comando (linha 8) apenas exibe uma mensagem para o usuário; no segundo comando (linha 9), o algoritmo lê o valor digitado pelo usuário e armazena na variável *valor*; e o terceiro comando (linha 10) é um acumulador para as distâncias informadas a cada dia. No balão há uma dica: toda vez que se pretende acumular valores em uma variável, a sintaxe pode ser feita utilizando um operador reduzido conforme dica.

4. Após serem inseridas as distâncias percorridas pelos 15 veículos, é necessário ler o tipo de entrada que o usuário escolheu (linhas 12 e 13). Após ler é necessário testar. No formulário existem três possibilidades para a unidade (quilômetros, metros ou milhas), então no algoritmo existem dois testes, pois caso os dois sejam falsos, o terceiro será verdadeiro. Na linha 14 é feito o teste, e se a unidade escolhida foi "metros", a variável *total* é dividida por 1000 para fazer a conversão para quilômetros, mas caso não tenha sido escolhido "metros", então testa-se se a unidade informada foi "milhas" (linha 16). Caso tenha sido, a variável *total* é dividida por 0,621371 para fazer a conversão para quilômetros, mas caso não tenha sido, os testes se encerram (linhas 18 e 19). E agora? Caso não tenha sido escolhido metro ou milha, isso quer dizer que foi escolhido quilômetro, então não é necessário nenhuma conversão.

5. Por fim, na linha 20 é impresso na tela a quantidade de quilômetros rodados pelos 15 carros naquele dia.



Figura 2.23 | Algoritmo para controle de quilometragem



Fonte: elaborada pelo autor.

O algoritmo escrito totaliza a quantidade de quilômetros rodados pelos 15 carros durante um dia. E se fosse necessário armazenar a quantidade de cada carro? Nesse caso, precisaríamos ter 15 variáveis, uma para cada carro. Na próxima seção você conhecerá um novo elemento de algoritmo que permitirá armazenar diversos valores em uma mesma variável, deixando seus algoritmos cada vez mais sofisticados.

## Avançando na prática

### Automatização de máquina para jogo de boliche

#### Descrição da situação-problema

As máquinas de boliche dispõem os pinos de forma automática da seguinte forma: fila 1 = 1 pino; fila 2 = 2 pinos; fila 3 = 3 pinos e fila 4 = 4 pinos, ou seja, trabalha com 10 pinos. Porém, um novo fabricante decide inovar e permitir que o jogador escolha até 28 pinos para jogar, e você foi contratado para escrever o algoritmo

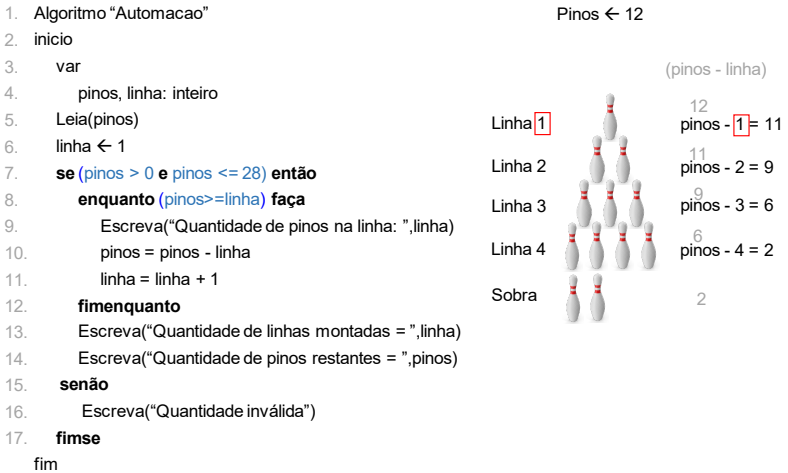
que automatizará o processo de disposição dos pinos. Seu algoritmo, além de organizar a disposição dos pinos, tem de validar a quantidade de pinos escolhida pelo jogador, pois números menores que 0 e maiores que 28 não são permitidos. Outro cuidado que você deve tomar é se o usuário escolher uma quantidade que não seja suficiente para fechar uma fila, por exemplo, se escolher 12 pinos, serão utilizados 10 dispostos em 4 filas e sobrarão 2 pinos. Nesse caso, você deverá exibir uma mensagem dizendo que 2 pinos não puderam ser enfileirados.

## **Resolução da situação-problema**

Para resolver esse problema, vamos usar a estrutura de repetição com teste no início. Antes de iniciar a repetição, devemos verificar se o jogador digitou um valor válido (entre 0 e 28); esse teste é feito na linha 7. Somente se o valor for válido, a execução prossegue na linha 8, entrando em uma estrutura de repetição com teste no início. Essa repetição acontecerá enquanto a quantidade de pinos for maior ou igual ao número da linha. Dentro da repetição tem-se três comandos: o primeiro na linha 9 escreve quantos pinos foram usados naquela linha; o segundo comando na linha 10 atualiza a quantidade de pinos de acordo com que eles vão sendo dispostos; e o terceiro, na linha 11, atualiza a linha em que os pinos estão sendo dispostos. Quando a condição do laço de repetição se tornar falso, a repetição se encerra e são impressas quantas linhas foram montadas e quantos pinos restaram.

A condição dentro da estrutura de repetição (pinos  $\geq$  linha) foi montada porque existe uma relação direta entre o número da linha e a quantidade de pinos naquela linha. Vamos considerar que o jogador escolheu 12 pinos para jogar, na linha 1 é disposto 1 pino, portanto, restam 11 para serem dispostos ( $12 - 1$ ). Na linha 2 são dispostos 2 pinos, então agora restam 9 pinos; na terceira linha são dispostos 3, restando 6 pinos; na quarta linha são dispostos 4 pinos, restando apenas 2 pinos. Nesse momento, estamos na linha 5 e temos somente 2 pinos, portanto, pinos é menor que linhas, encerrando assim a repetição com uma sobra de dois pinos.

Figura 2.24 | Algoritmo para automatizar máquina de boliche



Fonte: elaborada pelo autor.

## Faça valer a pena

1. Considere um sensor de temperatura instalado em uma máquina para monitorar sua temperatura no decorrer do dia. Tal sensor faz a leitura e envia os dados para um programa de computador. O algoritmo foi escrito para fazer a leitura de 10 temperaturas ao decorrer do dia e imprimir a média. A temperatura média é a soma das temperaturas dividida pela quantidade de temperaturas medidas.

Sabendo que o algoritmo deve ter uma estrutura de repetição para efetuar a leitura das temperaturas e calcular a média, analise as seguintes afirmações e escolha a opção correta.

- I. O desenvolver do algoritmo pode usar qualquer estrutura de repetição, pois embora funcionem de diferentes maneiras, todas trarão o mesmo resultado.
- II. A melhor estrutura de repetição para esse caso é a estrutura de repetição com variável de controle, pois sabe-se de antemão quantas repetições em um trecho do algoritmo serão executadas.
- III. A melhor estrutura de repetição para esse caso é a estrutura de repetição com teste no início, pois sabe-se de antemão quantas repetições em um trecho do algoritmo serão executadas.

- a) Todas as afirmações são verdadeiras.
- b) Somente a alternativa I está correta.
- c) Somente as alternativas I e II estão corretas.
- d) Somente as alternativas II e III estão corretas.
- e) Somente as alternativas I e III estão corretas.

**2.** Em uma estrutura de repetição com variável de controle são informados a variável de controle, o valor inicial e o valor final dessa variável e o incremento dessa variável (passo), podendo ser de 1 em 1, 2 em 2, etc. Esse é o tipo de estrutura mais utilizado quando se sabe de antemão quantas vezes a repetição será feita, abrindo um gama de opções para os algoritmos.

Considerando o algoritmo a seguir, escolha qual opção apresenta o resultado do algoritmo.

- 1. início
- 2. var
- 3. soma, i: inteiro
- 4. soma ← 0
- 5. **para**(i de 1 até 20 passo 2) **faça**
- 6. soma ← soma + i
- 7. **fimpara**
- 8. Escreva("Resultado da soma:")
- 9. Escreva(soma)
- 10. fim

- a) Resultado da soma: 100.
- b) Resultado da soma: 0.
- c) Resultado da soma: 20.
- d) Resultado da soma: 40.
- e) Resultado da soma: 50.

**3.** A empresa X decidiu fazer um programa para traçar o perfil de seus clientes. Para isso, pediu à equipe de TI que escrevesse um algoritmo que fizesse a leitura dos dados de todos os clientes em um banco de dados e os classificasse de acordo com a idade de cada cliente, separando-os por faixa etária.

Escolha o algoritmo que representa a estrutura de repetição necessária para fazer a leitura dos dados no banco de dados.

- a) **para**(cliente de 1 até fim passo 1) faça  
    Leia(cliente)  
**fimpara**
  
- b) **faça**  
    Leia(cliente)  
**enquanto**(cliente<100)
  
- c) **enquanto**(cliente<200) faça  
    Leia(cliente)  
**fimenquanto**
  
- d) **para**(cliente de 1 até 100 passo 1) faça  
    Leia(cliente)  
**fimpara**
  
- e) **enquanto**(cliente<>fim) faça  
    Leia(cliente)  
**fimenquanto**

## Seção 2.3

### Estrutura de dados

#### Diálogo aberto

Caro estudante, chegamos à última seção da Unidade 2, que foi toda destinada a apresentar os elementos para construção de algoritmos. Nas seções anteriores, você desenvolveu algoritmos capazes de fazer validações e conversões de unidades de medidas. Para guardar os valores de entradas, você utilizou diversas variáveis. Será que existe uma maneira de otimizar o armazenamento e a utilização de tais variáveis?

Nesta seção você deverá desenvolver para a Kro Engenharias um novo algoritmo que agrupe todas as entradas do mesmo assunto em uma única variável, a fim de otimizar o desempenho do sistema. Para tal missão, considere o problema do empilhamento de produtos feito pelo robô na unidade 1 do seu livro. Como uma caixa com maior peso não pode ficar sobre uma com menor peso, o robô tem que armazenar o peso de cada caixa que está no estoque para fazer a comparação. Escreva um algoritmo que armazene o peso, a largura e a altura de 200 caixas em uma estrutura de dados. Sabendo que a altura do galpão do estoque onde o robô trabalha é de 40 metros, complemente o algoritmo para que o robô saiba quando deve parar de empilhar.

Para completar sua missão na Kro Engenharias, nesta unidade você aprenderá como criar variáveis compostas com uma ou duas dimensões. Você também verá a diferença entre variáveis compostas homogêneas e heterogêneas, sendo que estudaremos primeiro as homogêneas para depois entender o mecanismo de funcionamento das heterogêneas. Para finalizar a seção, você aprenderá um poderoso recurso de programação que lhe permitirá criar rotinas específicas para melhor organizar e otimizar seu código.

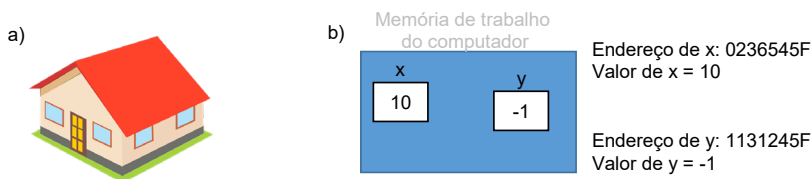
Bons estudos!

## Não pode faltar

Na seção anterior, você criou um algoritmo para a Kro Engenharias que somava a quilometragem gasta pelos quinze carros da companhia em um dia. Tal algoritmo não armazenava individualmente a quilometragem de cada carro, pois seriam necessárias quinze variáveis diferentes, uma para cada carro. Aprenderemos agora um tipo especial de variável que permite armazenar diversos valores em seu endereço. Tais variáveis são conhecidas como variáveis compostas.

Para compreendermos esse tipo de variável, vamos fazer uma analogia entre as variáveis na memória do computador e as residências em ruas. Na Figura 2.25 - a está representada uma residência que naturalmente possui um endereço composto por: nome da rua, número, cidade, estado e CEP, ou seja, ela possui um endereço que a torna única e identificável, portanto qualquer encomenda enviada para essa casa chegará por intermédio de seu endereço. Nessa residência existe espaço suficiente apenas para uma família morar (1 endereço  $\rightarrow$  1 família). Assim são as variáveis primitivas (do tipo inteiro, real, booleana) na memória, para cada variável existe um endereço que a torna única e identificável, e cada uma dessas variáveis só pode armazenar um valor de cada vez (1 variável  $\rightarrow$  1 valor) (Figura 2.25 - b).

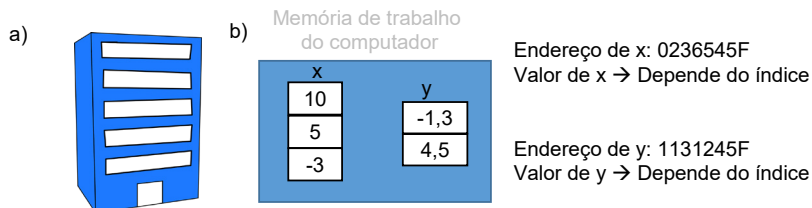
Figura 2.25 | Analogia entre uma residência e variáveis primitivas



Fonte: Adaptada de <<https://pixabay.com/pt/casa-casa-3d-s%C3%ADmbolo-casa-2263353>> Acesso em: 20 nov. 2017.

Já na Figura 2.26 - a, temos um edifício com cinco andares, que, da mesma forma que a casa, possui um endereço único. Em cada andar do edifício foi construído um único apartamento capaz de abrigar uma família, portanto temos 1 endereço  $\rightarrow$  5 famílias. As variáveis compostas são como os edifícios: em apenas um endereço vários valores podem ser armazenados simultaneamente. Na Figura 2.26 - b a variável x possui espaço para armazenar três valores inteiros, já a variável y possui espaço para armazenar dois valores reais.

Figura 2.26 | Analogia entre um edifício e variáveis compostas



Fonte: Adaptada de <<https://pixabay.com/pt/edif%C3%ADcio-blue-567929>> Acesso em: 20 nov. 2017.

Mas como essas variáveis são criadas? Como seus valores serão identificados se eles estão dentro da mesma variável? Na declaração da variável será adicionado um elemento que informará quantos “espaços” existirão para armazenamento de valores. Funciona como se uma porção inteira de memória fosse dividida em subporções, ou subespaços, e em cada um desses subespaços é possível inserir um valor. Mas se os vários valores têm o mesmo endereço, como diferenciar um do outro? Assim como os apartamentos em um prédio possuem números para diferenciá-los, as variáveis compostas possuem **índices** que as diferenciam. Portanto, uma variável composta possui um endereço na memória e índices para identificar seus subespaços. Existem autores que usam a nomenclatura “variável indexada” para se referir às variáveis compostas, pois sempre existirão índices (index) para os dados armazenados em cada espaço da variável composta (PIVA JUNIOR et al., 2012).

Existem diversas maneiras de se organizar os dados em um computador, cada uma das opções disponíveis é chamada de **estrutura de dados**. Dentre as estruturas, uma das mais utilizadas é a variável composta, pois em uma única estrutura é possível armazenar (organizar) diversos dados, de forma rápida e eficiente.



### Assimile

Variável composta é uma das maneiras de se organizar diversos dados em uma única estrutura na memória, por isso também é chamada de estrutura de dados. Embora existam outras estruturas de dados, a variável composta é a mais rápida que existe em um sistema computacional, pois cada dado é alocado sequencialmente na memória, tornando seu acesso extremamente rápido.



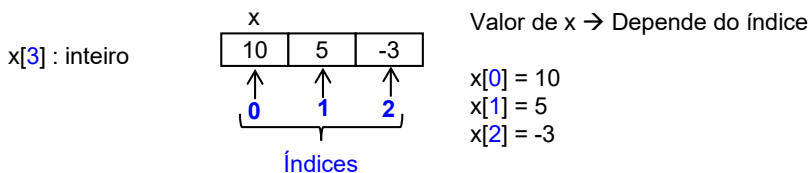
## Variável composta unidimensional

Para essa estrutura de dado, o nome **vetor** é o mais utilizado entre os profissionais, portanto vamos adotar essa nomenclatura. Um vetor é uma variável composta que se divide em apenas uma dimensão, ou seja, é como se fosse uma tabela de uma linha e N colunas (O resultado é o mesmo se pensarmos em um vetor como uma tabela de N linhas e uma coluna). Na hora de criarmos o vetor devemos informar apenas a quantidade de colunas que desejamos, além disso, precisamos de apenas um índice para identificar os elementos do vetor.

Como declarar um vetor:

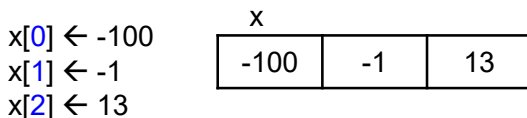
variável[N] : tipo  $\rightarrow$  N é a quantidade de colunas (ou linhas)

Exemplo: Vamos criar um vetor do tipo inteiro com espaço para armazenar três valores (N = 3):



Cada elemento do vetor x é localizado pelo seu índice, que **sempre** começará pelo valor zero. Portanto, o primeiro valor de qualquer vetor sempre estará no índice zero. Além disso, o vetor é do tipo inteiro, isso significa que só poderão ser inseridos valores desse tipo, essa restrição caracteriza a variável composta como sendo **homogênea**.

Para adicionarmos valores dentro de uma variável composta unidimensional, basta informarmos a variável e qual índice desejamos preencher. Exemplo:





Vamos utilizar uma variável composta unidimensional para aprimorar o algoritmo criado para a Kro Engenharias na seção anterior, permitindo armazenar a quilometragem de cada carro em um índice do vetor.

Na linha 4 é declarada a variável composta *distancia* que irá armazenar a quilometragem de cada um dos 15 carros. Na linha 7 é criada uma estrutura de repetição para controlar o registro da quilometragem de cada carro. A variável *carro* é utilizada para controlar a quantidade de carros e as posições (índices) da variável composta, por isso ela inicia em 0 e termina em 14. Então, na primeira execução do laço de repetição, *carro* vale 0, é pedido para o usuário digitar a distância percorrida pelo carro e é feita a leitura do valor digitado (linhas 8 e 9). Em seguida, o valor lido é guardado dentro do vetor *distancia* no índice *carro*; como *carro* nesse momento vale 0, então o valor digitado é armazenado em *distancia[0]* (linha 10), em seguida é acumulado o valor digitado na variável *total* (linha 11). Na segunda iteração do laço de repetição, *carro* vale 1, portanto o valor digitado será armazenado em *distancia[1]*, e assim sucessivamente até o último valor ser armazenado em *distancia[14]*. Após todos os 15 valores terem sido informados, a primeira estrutura de repetição finaliza na linha 12, o total de quilômetros andado naquele dia é impresso e um novo laço de repetição se inicia, agora escrevendo o valor que cada carro percorreu (linhas 14 a 16).

Figura 2.27 | Algoritmo usando vetor

```
1. Algoritmo "ControleCarro"
2. inicio
3. var
4. distancia[15] : real
5. total, valor: real
6. carro: inteiro
7. -- para (carro de 0 até 14 passo 1) faça
8. Escreva("Digite a distância percorrida pelo carro")
9. Leia(valor)
10. distancia[carro] ← valor
11. total ← total + distancia[carro]
12. -- fimpara
13. Escreva("Quantidade de quilômetros rodados no dia:", total)
14. -- para (carro de 0 até 14 passo 1) faça
15. Escreva("Distância percorrida pelo carro:", distancia[carro])
16. -- fimpara
17. fim
```

Fonte: elaborada pelo autor.

## Variável composta multidimensional

Para essa estrutura de dados, o nome **matriz** é o mais utilizado dentre os programadores, portanto vamos adotar essa nomenclatura. Uma matriz é uma variável composta que se divide em duas dimensões, ou seja, é "uma tabela" de  $M$  linhas e  $N$  colunas (imagine uma planilha do Excel, ela possui  $M$  linhas e  $N$  colunas). Na hora de criarmos uma matriz, devemos informar a quantidade de linhas e de colunas que desejamos, conseqüentemente, precisamos de dois índices para identificar os elementos da matriz.



### Assimile

Para gravar ou acessar dados em uma variável composta, temos que usar os índices. Em qualquer tipo de variável composta o índice sempre iniciará em zero, conseqüentemente, o maior índice em uma variável composta será sempre um a menos que a quantidade de elementos. Se tivermos 15 elementos, o maior índice será 14; se tivermos 10 elementos, o maior índice será 9, portanto sempre teremos índice =  $N - 1$ .

Além disso, para iterar pelos elementos de um vetor, usam-se estruturas de repetição, sendo a variável de controle do laço de repetição a variável usada como índice da variável composta.

Como declarar uma matriz:

variável[M][N]: tipo

→  $M$  é a quantidade de linhas  
→  $N$  é a quantidade de colunas

Exemplo: Vamos criar uma matriz do tipo inteiro com três linhas e duas colunas ( $M = 3$  e  $N = 2$ ).

|           | coluna 0 | coluna 1 |
|-----------|----------|----------|
| linha 0 → | 10       | 5        |
| linha 1 → | -3       | 2        |
| linha 2 → | 1        | 100      |

x[3][2] : inteiro

Valor de x → Depende de dois índices

x[0][0] = 10      x[0][1] = 5  
x[1][0] = -3     x[1][1] = 2  
x[2][0] = 1      x[2][1] = 100

Quantos elementos conseguimos colocar na matriz 3x2? Para saber a capacidade de uma matriz (quantidade de elementos)

basta multiplicarmos a quantidade de linhas pela quantidade de colunas ( $3 \times 2 = 6$  elementos). Cada elemento da matriz é localizado por dois índices, o primeiro refere-se à linha, e o segundo à coluna, ambos índices **sempre** começarão pelo valor zero. Portanto, o primeiro valor de qualquer matriz sempre estará no índice (0,0). Como a estrutura foi "tipada" como inteiro, só valores inteiros poderão ser adicionados a essa matriz, caracterizando-a como homogênea.

Para adicionarmos valores dentro de uma variável composta multidimensional, basta informarmos a variável e qual o índice da linha e da coluna que desejamos preencher. Exemplo:

$x[0][0] = -100$      $x[0][1] = -1$   
 $x[1][0] = 13$       $x[1][1] = -15$   
 $x[2][0] = -5$       $x[2][1] = 1050$

|      |      |
|------|------|
| -100 | -1   |
| 13   | -15  |
| -5   | 1050 |

É de extrema importância compreender a ordem em que os dados são inseridos em uma matriz.

- O primeiro valor a ser usado refere-se à linha, portanto é escolhida a linha 0;
- Após selecionar a linha, é a vez de percorrer todas as colunas, começando pela 0, então em uma matriz  $3 \times 3$  teremos a seguinte ordem de execução:

|                        |                        |                        |
|------------------------|------------------------|------------------------|
| 1° - Linha 0, Coluna 0 | 2° - Linha 0, Coluna 1 | 3° - Linha 0, Coluna 2 |
| 4° - Linha 1, Coluna 0 | 5° - Linha 1, Coluna 1 | 6° - Linha 1, Coluna 2 |
| 7° - Linha 2, Coluna 0 | 8° - Linha 2, Coluna 1 | 9° - Linha 2, Coluna 2 |

Para realizar o preenchimento de dados em uma matriz, utilizamos estruturas de repetição aninhadas. A primeira estrutura será referente às linhas, e a segunda às colunas.



### Exemplificando

Vamos alterar o algoritmo da Kro Engenharias (Figura 2.27) para que seja armazenada a quilometragem percorrida em cada dia do mês pelos 15 carros. Agora temos dois índices, "dia do mês" e "carro", usaremos o dia do mês como índice para as linhas, então teremos 31 linhas, e os

carros continuarão sendo as colunas, portanto teremos uma matriz 31 x 15. O algoritmo está representado na Figura 2.28 - a. Na linha 4 está declarada a variável composta multidimensional *distância* com capacidade para armazenar 465 valores (31 dias e 15 carros); nas linhas 5 e 6 estão declaradas as variáveis de controle do laço de repetição e variáveis para armazenar os demais valores. O preenchimento dos dados inicia com a estrutura de repetição na linha 7, atribuindo o valor 0 à variável de controle *dia*, em seguida, na linha 8, inicia a segunda estrutura de repetição que controlará cada um dos 15 carros. Portanto, na primeira execução a variável *distância* irá armazenar o valor digitado no índice (0,0): `distancia[0][0]`; na segunda execução o laço volta para a linha 8, agora com a variável *carro* valendo 1, então o valor digitado será armazenado em `distancia[0][1]`, veja que a variável *dia* não muda, isso porque ela está sendo usada para marcar as linhas da matriz, e a linha só muda quando todas as colunas foram preenchidas. Para cada valor digitado, a variável *total* acumula o resultado (linha 12) e, após todos os 465 valores serem preenchidos, os laços de repetição se encerram (linha 13 e 14) e a quantidade de quilômetros percorridas no mês é informada.

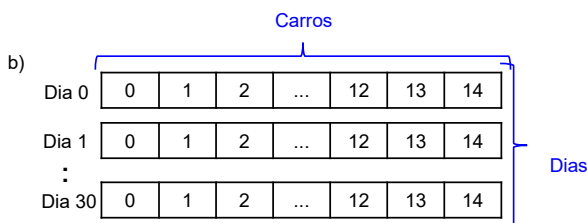
Na Figura 2.28 - b há um esquema representando a estrutura de dados `distancia[31][15]`. Perceba que uma matriz é na verdade a união de vários vetores. Para cada dia temos um vetor com 15 colunas representando os carros.

Figura 2.28 | Algoritmo usando matriz

```

a) 1. Algoritmo "ControleCarro
 2. inicio
 3. var
 4. distancia[31][15] : real
 5. dia, carro : inteiro
 6. total, valor : real
 7. -- para (dia de 0 até 30 passo 1) faça
 8. -- para (carro de 0 até 14 passo 1) faça
 9. Escreva("Digite a distância percorrida pelo carro")
 10. Leia(valor)
 11. distancia[dia][carro] ← valor
 12. total ← total + distancia[dia][carro]
 13. -- fimpara
 14. -- fimpara
 15. Escreva("Quantidade de quilômetros percorridos no mês:", total)
 16. fim

```



Fonte: elaborada pelo autor.

## Variável composta heterogênea

A utilização das variáveis compostas aprendidas até o momento permitem otimizar o armazenamento de valores do mesmo tipo (só inteiros – só reais etc.), porém existe um tipo de variável composta que permite armazenar valores de diferentes tipos dentro da mesma estrutura, tal estrutura é conhecida como registro (MANZANO; MATOS; LOURENÇO, 2015; PIVA JUNIOR et al., 2012; SOFFNER, 2013).

Antes de declararmos uma variável composta heterogênea é preciso criar sua estrutura, ou seja, os campos que farão parte dessa variável. Esse conjunto de campos é o que determina o tipo da variável composta e dependerá do objetivo de cada aplicação.

Veja como se declara uma variável composta heterogênea: Utilizamos a palavra “**tipo**” para deixar explícito que estamos criando um novo tipo de variável que é um registro (linha 1). Esse novo tipo precisa de um nome para identificá-lo (identificador) e um conjunto de campos que começa a ser declarado pelo comando “**registro**” (linha 2) e finaliza com “**fimregistro**” (linha 4). Entre o início e o final do registro podemos declarar quantas variáveis (campos) forem necessárias. Após ser finalizado o novo tipo, basta escolher uma variável e atribuir o nome identificador do tipo criado à variável (linha 6).

1. **tipo**
2.     <identificador> = **registro**
3.                     <campos com seus tipos>
4.                     **fimregistro**
5. **var**
6.     <variável> : identificador

Para melhor entendermos, vamos criar uma variável composta heterogênea para armazenar os dados de uma ordem de serviço de

manutenção. Para cada pedido de manutenção feito na empresa é gerado um código, a data, o valor, o equipamento, o departamento e a descrição da manutenção requerida (Figura 2.29 - a). Todos esses elementos podem ser agrupados em uma única variável composta heterogênea, veja o código na Figura 2.29 - b, o algoritmo para criar esse registro. Na linha 2 é criado um registro com nome "Ficha\_Manutencao", nessa estrutura existem 6 campos (variáveis) com seus respectivos tipos. O tipo especificado não ocupa espaço na memória (PIVA JUNIOR et al., 2012), portanto não pode ser acessado diretamente. Para utilizarmos o registro criado, precisamos associá-lo a uma variável, é o que faz o comando na linha 11, atribui à variável *ordemServico* o tipo *Ficha\_Manutencao*.

Figura 2.29 | Formulário para pedido de manutenção

1. **tipo**
2. **Ficha\_Manutencao = registro**
3. **codigo : inteiro**
4. **equipamento: texto**
5. **data: data**
6. **departamento: texto**
7. **descricao: texto**
8. **valor: real**
9. **fimregistro**
10. **var**
11. **ordemServico : Ficha\_Manutencao**

Fonte: elaborada pelo autor



**Reflita**

Em uma variável composta heterogênea podemos armazenar valores de diferentes tipos, criando uma tabela com diversas colunas que se deseja armazenar. Qual ou quais critérios devem ser usados para determinar quais campos agrupar em uma estrutura composta heterogênea?

A leitura e a escrita de valores dentro de uma variável composta são feitas utilizando o nome da variável *ponto*, campo a ser lido ou atribuído: **variavel.campo**.

No nosso exemplo da Figura 2.29 para armazenar dados na estrutura criada, devemos utilizar:

```
ordemServico.codigo ← 1281
ordemServico.equipamento ← "Ar condicionado"
ordemServico.data: 20/11/2017
```

E para ler os dados utilizamos: `Leia(ordemServico.codigo)` e assim sucessivamente.

A variável que foi atribuída à estrutura *Ficha\_Manutencao* é capaz de armazenar apenas uma ordem de serviço, porém ela poderia ser declarada como um vetor, e assim armazenaria **N** ordens de serviços.

```
var
 ordemServico[N]: Ficha_Manutencao
```



**Pesquise mais**

É muito importante que você leia o capítulo 7 de MANZANO, José Augusto N. G.; MATOS, Ecívaldo; LOURENÇO, André Evandro. **Algoritmos: Técnicas de Programação**. 2. ed. São Paulo: Érica, 2015, para maiores detalhes a respeito da utilização de variáveis compostas heterogêneas.

## Rotinas de algoritmos – Funções

Todos os algoritmos desenvolvidos até o momento são compostos por um único bloco delimitado por um início e um final. Construir algoritmos dessa forma pode tornar a codificação complexa à medida que a quantidade de comandos aumenta, além de acarretar em repetição de comandos em alguns casos. Para contornar esse problema, podemos dividir os comandos de um algoritmo em sub-rotinas (funções), assim, além de organizarmos melhor o algoritmo, dividimos os comandos em funções específicas e evitamos repetir códigos.

Para construir as sub-rotinas precisamos acrescentar alguns elementos em nossos algoritmos. Na Figura 2.30 o *inicio* utilizado na linha 7 marca a rotina principal (quando não usamos sub-rotinas



essa é a única), veja que antes dessa rotina principal existe um novo bloco de comandos (linhas 2 a 6). Após atribuir nome do algoritmo "linha 1" precisamos declarar as variáveis que serão utilizadas tanto nas sub-rotinas como na rotina principal (linha 2). O próximo passo é escrever as sub-rotinas. Na Figura 2.30 existe apenas uma sub-rotina declarada (linhas 3 a 6), veja que a declaração é feita com três comandos: (i) "**Sub-rotina**" para indicar que ali será declarada uma função; (ii) um nome para essa sub-rotina; (iii) "**(Parâmetros)**", significa que uma sub-rotina pode receber variáveis externas a ela. Somente depois de declarar as variáveis e as sub-rotinas é que podemos iniciar a rotina principal (linha 7). Todo algoritmo, independentemente do número de sub-rotinas, sempre iniciará a execução pela rotina principal.

Figura 2.30 | Estrutura para criar sub-rotinas

```
1. Algoritmo "Nome_algoritmo"
2. -var
3. Sub-rotina "Nome_subRotina1"(Parâmetros)
4. inicio
5. var
6. fimSub-Rotina
7. inicio ←
8. var
9. fimAlgoritmo
```

Fonte: elaborada pelo autor.

Para entendermos a utilização de sub-rotinas, vamos criar um algoritmo que converte uma certa temperatura digitada em graus Celsius para graus Kelvin. Veja na solução da Figura 2.31 que o algoritmo possui dois blocos, linhas 2 a 8 e linhas 9 a 13. No primeiro bloco estão sendo declaradas variáveis que serão utilizadas tanto na sub-rotina quanto na rotina principal (linha 3). Na linha 4 tem-se a declaração da sub-rotina, que foi nomeada como *ConverterTemperatura* e essa função recebe um valor externo do tipo real. A função usa esse valor externo para fazer a conversão na linha 6, em seguida é escrita a temperatura convertida (linha 7). Na rotina principal (linhas 9 a 13), é solicitado ao usuário digitar uma temperatura em graus Celsius (linha 10), em seguida esse valor é lido e guardado dentro da variável *temperaturaCelsius* (linhas 11 e 12) e por fim a sub-rotina é invocada (linha 12), ou seja, somente nesse ponto é que a função é executada.

Figura 2.31 | Algoritmo para converter temperatura usando sub-rotina

```
1. Algoritmo "Temperatura"
2. var
3. temperaturaCelsius, temperaturaKelvin : real
4. - Sub-rotina ConverterTemperatura (temperaturaCelsius : real)
5. - inicio
6. temperaturaKelvin ← temperaturaCelsius + 273
7. Escreva("Temperatura convertida:", temperaturaKelvin)
8. - fimSub-Rotina
9. - inicio
10. Escreva("Digite uma temperatura em graus Celsius")
11. Leia(temperaturaCelsius)
12. ConverterTemperatura(temperaturaCelsius)
13. - fim
```

Fonte: elaborada pelo autor.



## Pesquise mais

As sub-rotinas são extremamente usadas na construção de algoritmos. Aprenda mais sobre sua utilização lendo:

- O capítulo 12 do livro MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. **Estudo dirigido de algoritmos**. 13. ed. São Paulo: Érica, 2010.
- O capítulo 8 do livro MANZANO, José Augusto N. G.; MATOS, Eivaldo; LOURENÇO, André Evandro. **Algoritmos: Técnicas de Programação**. 2. ed. São Paulo: Érica, 2015.
- Os vídeos 12 e 13 da série "Curso de Lógica de Programação" do professor Gustavo Guanabara, disponível em <<https://goo.gl/dbFxzX>>. Acesso em: 22 nov. 2017.

Ambos os livros estão disponíveis na biblioteca virtual <<https://biblioteca-virtual.com>> Acesso em: 22 nov. 2017.

## Sem medo de errar

Caro(a) estudante: foi designado a você desenvolver um algoritmo para a Kro Engenharias com o objetivo de ajudar o robô da empresa a organizar as caixas no estoque, pois existe uma regra que uma caixa com maior peso não pode ficar sobre uma mais leve, portanto o robô tem que "saber" qual o peso de cada caixa. Além

disso, o galpão do estoque possui 40 metros de altura, para que o robô saiba quando parar de empilhar, ele deve “saber” a altura de cada caixa. A memória do robô é limitada, possuindo capacidade para armazenar informações de 200 caixas.

Com o conhecimento que adquiriu até aqui, você pode propor uma solução como o algoritmo da Figura 2.32. O algoritmo possui uma sub-rotina para realizar o empilhamento que recebe um valor externo do tipo inteiro “caixa”, ou seja, recebe qual caixa deverá empilhar (linhas 2 a 5). Na linha 6 inicia-se a rotina principal, o primeiro passo é criar uma estrutura de variável composta heterogênea para guardar as informações sobre a caixa (linhas 7 a 12). Na linha 14 é declarado um vetor com capacidade para armazenar 200 informações do tipo *Dados\_Caixa*, ou seja, 200 informações da variável heterogênea criada. Das linhas 17 até 22 temos uma estrutura de repetição para guardar 200 valores dentro de cada campo da variável heterogênea *informacao*. Veja que usamos *informacao[caixa]*, pois essa variável é um vetor, então temos que usar seus índices para armazenar os dados. Na linha 23 a variável *caixa* é novamente zerada, pois ela está sendo utilizada para acessar os índices do vetor. Das linhas 24 a 28 temos outra estrutura de repetição, agora com teste no final, para empilhar as caixas até que a altura da pilha seja menor ou igual a 40 metros, que é a altura do galpão. Na linha 25 a sub-rotina criada é chamada, na linha 26 a altura da pilha é atualizada, na linha 27 o índice da caixa é atualizado e na linha 28 o teste da altura é realizado.

Figura 2.32 | Algoritmo para armazenar dados para o robô.

```

1. Algoritmo "ProgramarRobo"
2. Sub-rotina Empilhar_Caixa(caixa : inteiro)
3. inicio
4. comandos para empilhar
5. fimSub-Rotina
6. inicio
7. tipo
8. Dados_Caixa = registro
9. peso: real
10. altura: real
11. largura: real
12. fimregistro
13. var
14. informacao[200] : Dados_Caixa
15. caixa : inteira
16. dado1, dado2, dado3, alturaPilha : real
17. para (caixa de 0 até 100 passo 1) facer

```

Declarando uma sub-rotina

Declarando uma variável composta heterogênea

```

17. para (caixa de 0 ate 199 passo 1) faça
18. Leia (dado1, dado2, dado3)
19. informacao[caixa].peso ← dado1
20. informacao[caixa].altura ← dado2
21. informacao[caixa].largura ← dado3
22. fimpara
23. caixa ← 0
24. faça
25. Empilhar_Caixa(caixa)
26. alturaPilha ← alturaPilha + informacao[caixa].altura
27. caixa ← caixa + 1
28. enquanto (alturaPilha <= 40)
29. fim

```

Usando uma estrutura de repetição para armazenar 200 valores

"Chamando" a sub-rotina criada

Usando uma estrutura de Repetição para o robô saber quando parar de empilhar

Fonte: elaborada pelo autor.

Para solidar seu conhecimento em algoritmo, é imprescindível praticar, portanto, reescreva o algoritmo alterando o nome das variáveis (simples e composta) e da sub-rotina.

## Avançando na prática

### Ordenação de valores em um vetor

#### Descrição da situação-problema

Uma das estruturas de dados mais utilizadas para armazenamento de informações do mesmo tipo é a variável composta homogênea. Porém, muitas vezes, além de armazenar os dados é preciso ordená-los em ordem crescente ou decrescente. A empresa X lhe contratou para escrever um algoritmo para ordenar os registros da quantidade de calor medido por um sensor a cada dia durante todo o mês. A partir desses dados, a empresa conseguirá mapear se houve alguma anomalia durante o mês.

#### Resolução da situação-problema

Para resolver o problema da empresa X, você deverá usar um vetor com 31 posições para armazenar a temperatura diária. Após conhecer o valor de cada dia, você deverá usar uma sub-rotina para ordenar o algoritmo.

Iniciamos o algoritmo criando um vetor com capacidade para armazenar 31 valores do tipo real *medidas[31]* (linha 3). Essa variável está fora da rotina principal e fora da sub-rotina, portanto dizemos que ela é uma variável global, pois é acessível em qualquer parte do algoritmo. Vamos agora ver o que acontece na rotina principal (linhas 23 a 31): nas linhas 25 e 26 são criadas duas variáveis, como

elas estão dentro de uma rotina, são ditas variáveis locais, pois outras sub-rotinas não possuem acesso ao valor dessas variáveis. Das linhas 27 a 30 tem-se uma estrutura de repetição para armazenar os 31 valores dentro do vetor. E na linha 31 é chamada a sub-rotina para ordenar os valores armazenados, passando como parâmetro o vetor "medidas". Na sub-rotina temos toda a lógica necessária para ordenar um vetor (linhas 4 a 22).

Nesse algoritmo, a ordenação é feita comparando 1 valor, com todos  $n$  valores,  $n$  vezes. Como temos 31 valores, teremos  $31 \times 30$ , ou seja, 930 comparações ( $N \times (N-1)$ ). Essas repetições são realizadas usando duas estruturas de repetições. A variável *controle* é usada na estrutura de repetição externa para fazer com que os 31 valores sejam verificados, já a variável *dia*, na estrutura interna, é usada para comparar o valor do dia com os demais valores. Caso o valor do dia seja maior que a do próximo dia (linha 11), esses valores são trocados usando uma variável auxiliar para armazenar o dado temporariamente (linhas 12 a 14), sendo esse processo repetido até que todos os dias tenham sido comparados entre si.

Esse algoritmo é conhecido como *BubbleSort*; assista a esse vídeo para ajudar a entender seu funcionamento <<https://www.youtube.com/watch?v=llX2SpDkQDc>> Acesso em: 22 nov. 2017.

Figura 2.33 | Algoritmo para ordenação de vetor Bubblesort

```

1. Algoritmo "OrdenacaoVetor"
2. var
3. medidas[31] : real → Variável acessível em todo algoritmo (variável global)
4. Sub-rotina Ordenar_Vetor(medidas[31] : real)
5. inicio
6. var
7. dia, controle : inteiro } Variáveis acessíveis somente na
8. aux : real } rotina Ordenar_Vetor(variável local)
9. para (controle de 0 até 30 passo 1) faça
10. para (dia de 0 até 30 passo 1) faça
11. se (medidas[dia] > medidas[dia + 1])
12. aux = medidas[dia];
13. medidas[dia] = medidas[dia + 1]
14. medidas[dia + 1] = aux;
15. fimse
16. fimpara
17. fimpara
18. Escreva("Vetor ordenado")
19. para (dia de 0 até 30 passo 1) faça
20. Escreva(medidas[dia])
21. fimpara
22. fimSub-Rotina

```

```

23. inicio
24. var
25. valor : real
26. dia: inteiro } Variáveis acessíveis somente na rotina principal (variável local)
27. para (dia de 0 até 30 passo 1) faça
28. Leia (valor)
29. medidas[dia] ← valor
30. fimpara } Usando uma estrutura de repetição
31. Ordenar_Vetor(medidas) → "Chamando" a sub-rotina criada
32. fim

```

Fonte: elaborada pelo autor.

## Faça valer a pena

**1.** “A primeira ação de um projetista de software deve ser a definição dos dados que serão armazenados e processados pelo programa a ser criado. Com base nessa informação será possível especificar os tipos de dados (inteiro, real, caractere e lógico) e as estruturas de armazenamento a serem utilizadas.” (MANZANO; MATOS; LOURENÇO, 2015, p. 65.)

Dentre as estruturas de dados disponíveis, as variáveis compostas homogêneas e heterogêneas são amplamente utilizadas. A respeito de tais estruturas de dados, analise as afirmações abaixo:

- I. Uma variável composta unidimensional precisa ter sua capacidade de armazenamento especificada no momento em que é declarada. Seus dados serão acessados para leitura e escrita através de um índice que sempre começará em zero.
  - II. Matriz é uma estrutura de dados cujos dados são organizados em linhas e colunas. Para acessar os dados dessa estrutura precisamos usar um índice que sempre começará em zero.
  - III. A utilização de variáveis compostas heterogêneas é similar à utilização das variáveis primitivas (inteiro, real, caractere e lógico), pois é necessário especificar o tipo que tal variável terá.
- a) Somente as alternativas I e III estão corretas.
  - b) Somente as alternativas I e II estão corretas.
  - c) Somente a alternativa I está correta.
  - d) Somente a alternativa II está correta.
  - e) Somente a alternativa III está correta.

**2.** A utilização de vetores nos algoritmos se torna eficiente à medida que armazenamos diversos valores em uma mesma estrutura de dados e utilizamos estruturas de repetição para iterar (acessar) sobre os elementos desse vetor. A fábrica Y possui um sistema de controle de energia que armazena o consumo diário em kW dentro de uma estrutura de dados

do tipo vetor. Considerando um mês com 30 dias, a fábrica deseja emitir relatórios sobre o consumo do mês.

Selecione o algoritmo que calcula o consumo mensal.

a)

1. Algoritmo "ExercicioVetor"
2. inicio
3. **var**
4.     consumo[30] : real
5.     diaAtual : inteira
6.     consumoTotal : real
7.     **para** (diaAtual de 0 até 30 passo 1) **faça**
8.         consumoTotal  $\leftarrow$  consumoTotal + consumo[diaAtual]
9.     **fimpara**
10.     Escreva(consumoTotal);
11. fim

b)

1. Algoritmo "ExercicioVetor"
2. inicio
3. **var**
4.     consumo[30] : real
5.     diaAtual : inteira
6.     consumoTotal : real
7.     **para** (diaAtual de 1 até 31 passo 1) **faça**
8.         consumoTotal  $\leftarrow$  consumoTotal + consumo[diaAtual]
9.     **fimpara**
10.     Escreva(consumoTotal);
11. fim

c)

1. Algoritmo "ExercicioVetor"
2. inicio
3. **var**
4.     consumo[30] : real
5.     diaAtual : inteira
6.     consumoTotal : real
7.     diaAtual  $\leftarrow$  0;
8.     **faça**
9.         consumoTotal  $\leftarrow$  consumoTotal + consumo[diaAtual]
10.         diaAtual  $\leftarrow$  diaAtual + 1
11.     **enquanto** (diaAtual < 29)
12.     Escreva(consumoTotal);
13. fim

d)

1. Algoritmo "ExercicioVetor"
2. inicio
3. **var**
4.     consumo[30] : real
5.     diaAtual : inteira
6.     consumoTotal : real
7.     **para** (diaAtual de 0 até 29 passo 1) **faça**
8.         consumoTotal  $\leftarrow$  consumoTotal + consumo[diaAtual]
9.     **fimpara**
10.     Escreva(consumoTotal);
11. fim

- e)
1. Algoritmo "ExercicioVetor"
  2. inicio
  3. **var**
  4.     **consumo**[30] : real
  5.     **diaAtual** : inteira
  6.     **consumoTotal** : real
  7.     **diaAtual** ← 0;
  8.     **faça**
  9.         **consumoTotal** ← **consumoTotal** + **consumo**[**diaAtual**]
  10.        **diaAtual** ← **diaAtual** + 1
  11.     **enquanto** (**diaAtual** <= 30)
  12.         **Escreva**(**consumoTotal**);
  13. **fim**

3. Matrizes são estruturas bidimensionais utilizadas para armazenar dados de um mesmo tipo, ou seja, são estruturas de dados fortemente tipadas. Uma matriz é na verdade a união de vários vetores de mesmo tamanho, portanto se em um vetor basta de 1 índice para identificar e acessar seus elementos, em uma matriz são necessários dois índices, um para as linhas e outro para as colunas.

Considerando o algoritmo e a matriz dados, escolha a opção que representa as informações que serão escritas pelo comando da linha 8.

1. Algoritmo "Exercicio"
2. inicio
3. **var**
4.     **m**, **n**, **valor**[4][4] : inteiro
5.     **para** (**m** de 0 até 3 passo 1) **faça**
6.         **para** (**n** de 0 até 3 passo 1) **faça**
7.             **if** (**m** == **n**) **então**
8.                 **Escreva**(**valor**[**m**][**n**])
9.             **fimse**
10.         **fimpara**
11.     **fimpara**
12. **fim**

|    |     |     |    |
|----|-----|-----|----|
| 10 | 5   | -1  | 3  |
| -4 | -57 | 10  | 5  |
| 30 | -40 | 100 | 6  |
| 25 | 1   | 17  | 30 |

- a) 10, 5, -1, 3
- b) 10, -4, 30, 25
- c) 10, -57, 100, 30
- d) 25, 1, 17, 30
- e) 25, 30, -4, 10



# Referências

BARNETT, Janet Heine. **Origins of boolean algebra in the logic of classes:** George Boole, John Venn and C. S. Peirce. Disponível em: <<https://www.maa.org/press/periodicals/convergence/origins-of-boolean-algebra-in-the-logic-of-classes-george-boole-john-venn-and-c-s-peirce>>. Acesso em: 18 nov. 2017.

HALLIDAY, David; RESNICK, Robert. **Fundamentos de Física.** Rio de Janeiro: LTC, 2015.

MANZANO, José Augusto N. G.; MATOS, Ecivaldo; LOURENÇO, André Evandro. **Algoritmos:** técnicas de programação. 2. ed. São Paulo: Érica, 2015.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. **Estudo dirigido de algoritmos.** 13. ed. São Paulo: Érica, 2010.

MANZANO, José Augusto N. G. **Estudo dirigido linguagem C.** 13. ed. São Paulo: Érica, 2010.

NASA. **The first person on the moon.** Disponível em: <<https://www.nasa.gov/audience/forstudents/k-4/stories/first-person-on-moon.html>>. Acesso em: 02 nov. 2017.

PIVA JUNIOR, Dilermando (et al.). **Algoritmos e programação de Computadores.** Rio de Janeiro: Elsevier, 2012.

SOFFNER, Renato. **Algoritmos e programação em linguagem C.** São Paulo: Saraiva, 2013.



# Conceitos de programação

## Convite ao estudo

Caro aluno, chegou o momento de passar para prática todas as técnicas estudadas em algoritmos. Lembrando que um algoritmo é um conjunto de instruções para resolver uma tarefa ou solucionar um problema.

Nesta unidade vamos estudar os conceitos de programação e, para isso, vamos fazer a seguinte analogia:

Certa vez, um entusiasta se preparava para construir uma casa, rascunhou a planta da casa uma vez, outra e mais outra, até que chegou ao desenho definitivo. Tudo certo! Pensou no tamanho do terreno, posição do sol, tipo de material a ser utilizado e todas as outras variáveis pertinentes ao seu sonho. Se preparou, poupou dinheiro, até que chegou o grande dia da construção. Podemos comparar a construção da casa deste entusiasta com uma linguagem de programação. A linguagem de programação terá a mesma finalidade da construção, colocar em prática tudo aquilo que foi pensado e calculado para realização de uma tarefa.

Retomando a contextualização do início do livro, a multinacional Kro Engenharia está muito satisfeita com o seu desempenho de transmitir o pensamento computacional para os engenheiros da empresa, por este motivo, o seu gerente lhe designou a missão de trabalhar a linguagem de programação C, onde você deverá auxiliar seus colegas a compreender e aplicar os conceitos da linguagem.

No primeiro momento, você deverá prender a atenção dos engenheiros para a estrutura da programação C, depois, mostrar alguns dos tipos de bibliotecas mais utilizadas na programação C. Ainda, deverá manter o foco nos tipos de

variáveis, constantes e outras formas de identificação dos dados. Para finalizar, deverá colocar de forma prática as expressões em linguagem C.

Os desafios são grandes, porém, prazerosos ao ponto de encorajar o seu raciocínio crítico para a solução de problemas utilizando a Linguagem de programação C.

# Seção 3.1

## Introdução à linguagem C

### Diálogo aberto

Caro aluno, chegou o momento de transcrever os conceitos e técnicas utilizadas em lógica, algoritmos e fluxogramas na linguagem de programação. Vamos citar como exemplo a língua brasileira de sinais para surdos, também conhecida como libras, imagine um programa de televisão, uma palestra ou um curso onde alguns de seus espectadores são surdos, um interprete é acionado para realizar a tradução simultânea de tudo que está acontecendo e sendo falado. Pois bem, assim funciona com as linguagens de programação, onde você desenvolve um pensamento lógico, desenvolve um algoritmo ou um fluxograma e converte em uma linguagem de programação para que o mesmo seja interpretado e executado.

Nesta seção, você, aluno, vai estudar as estruturas de um programa em linguagem C, as bibliotecas, como definir variáveis, constantes e com expressões em linguagem C.

Neste primeiro momento é fundamental o entendimento de todas as técnicas para iniciar a programação em Linguagem C, sendo ela a linguagem norteadora dos programas a serem desenvolvidos pelos engenheiros da Kro Engenharias. A sua missão é desenvolver um programa para realizar a conversão da temperatura de graus centígrados (Celsius) para graus Fahrenheit. A fórmula para essa conversão é:  $F = \frac{9.c + 160}{5}$ , onde "F" é a temperatura em Fahrenheit e "c" é a temperatura em centígrados. Você deverá elaborar a rotina de programação em linguagem C, inserir comentários nas linhas de programação e em seguida compilar o programa e entregar para o(a) professor(a) um relatório das rotinas do programa.

Boa sorte e ótimos estudos.

Caro aluno, a história da linguagem C é descrita por várias literaturas e por vários autores. Para contar esta história, vamos citar Manzano (2013), que escreve no seu livro o seguinte relato: em 1972 em um dos laboratórios da empresa Bell Telephone Labs. Inc. (atual Alcatel - Lucent) foi criada a linguagem de programação de computadores C por Dennis M. Ritchie, que desenvolveu a linguagem para auxiliar na segunda versão do sistema operacional UNIX, projeto liderado por Ken Thompson. Somente mais tarde Brian W. Kernighan juntou-se ao projeto de ampliação da linguagem C.

Aprendemos nas seções anteriores deste livro que para executar um programa precisamos de um sequenciamento do gerenciamento de dados caracterizado por 3 níveis:

- Entrada de dados: realiza as coletas de dados;
- Processamento: os dados são transformados em informação;
- Saída: onde todas as informações geradas pelo processamento de dados são apresentadas em um periférico.

Segundo Damas (2016), um programa é uma sequência de código organizada de tal forma que permita resolver um determinado problema. Um programa pode ser desenvolvido em módulos distintos e/ou em subprogramas. Dessa forma terá que existir um critério ou um formato de escrita bem definido que indique ao compilador todo o conjunto de código escrito pelo programador, onde se define em qual instrução ou local ele começará a executar.

### Compilador

Para executar um programa em linguagem C é necessário um compilador cuja função, segundo Schildt (2005), é traduzir os códigos em linguagem de alto nível para linguagem de baixo nível. O compilador traduz os códigos do programa e verifica se eles não possuem erros na sua execução.



Lembrando que existe diferença entre linguagem C e C++, a linguagem C é uma linguagem estruturada que será estudada neste livro. Já a linguagem C++, derivada da linguagem C, é uma linguagem orientada a objeto.

Segundo Manzano (2015), a primeira etapa do processo de compilação é o pré-processamento onde se identificam as bibliotecas e as primeiras sintaxes (comandos) do programa. Em seguida, o código-fonte é transformado em código objeto, o qual, através do linker, tem a função de vincular todos os arquivos gerados em um único arquivo executável.

Segundo Soffner (2013), a Linguagem C possui um total de 32 palavras reservadas, conforme definido pelo padrão ANSI:

Quadro 3.1 | Palavras reservadas em linguagem C

|          |        |          |          |
|----------|--------|----------|----------|
| auto     | double | int      | struct   |
| break    | else   | long     | switch   |
| case     | enum   | register | typedef  |
| char     | extern | return   | union    |
| const    | float  | short    | unsigned |
| continue | for    | signed   | void     |
| default  | goto   | sizeof   | volatile |
| do       | if     | static   | while    |

Fonte: Soffner (2013, p. 36).

É importante falar que, ao programar em C, você deve estar atento à forma de escrever, pois o programa diferencia a letra maiúscula da letra minúscula.

Agora, vamos entender como funcionam as bibliotecas da linguagem C.

## Bibliotecas em Linguagem C

Segundo Manzano (2015), as primeiras linhas de programação devem conter menções às bibliotecas, também conhecidas como

arquivos de cabeçalho. Para inserir as bibliotecas no programa é necessário colocar `#include` (inclusão de um arquivo no programa fonte) e, em seguida, entre os símbolos de menor "<" e maior ">" (quando se usa < e >, o arquivo é procurado na pasta include) o nome da biblioteca. Vale salientar que as bibliotecas auxiliam nas construções dos códigos e funções, sem a necessidade de declarar função por função, deixando a programação bem mais prática. Veja a seguir as principais bibliotecas utilizadas na linguagem C:

- **stdio** – essa biblioteca é responsável pelas funções de entradas e saídas, como é o caso da função `printf` e `scanf` que vamos aprender mais à frente.

Exemplo: `#include <stdio.h>`

- **stdlib** – essa biblioteca transforma as strings (vetores de caracteres) em números.

Exemplo: `#include <stdlib.h>`

- **string** – biblioteca responsável pela manipulação de strings.

Exemplo: `#include <string.h>`

- **time** – biblioteca utilizada para manipulação de horas e datas.

Exemplo: `#include <time.h>`

- **math** – biblioteca utilizada para operações matemáticas.

Exemplo: `#include <math.h>`

- **ctype** – biblioteca utilizada para classificação e transformação de caracteres.

Exemplo: `#include <ctype.h>`.

Antes de começar a programação em linguagem C, vamos conhecer como funciona a manipulação de dados e os tipos de operadores.

## Variáveis

As variáveis são locais reservados na memória para armazenamento dos dados. Podemos considerar como sendo as variáveis mais usadas as do tipo:

- **inteiro**: armazena os números inteiros (negativos ou positivos). Em linguagem C é definida por "int", veja algumas colocações na Tabela 3.1:



Tabela 3.1 | Dados reais

| Tipo de dado inteiro                              | Faixa de abrangência              | Tamanho |
|---------------------------------------------------|-----------------------------------|---------|
| unsigned short int                                | de 0 a 65.535                     | 16 bits |
| short / short int / signed int / signed short int | de -32.768 a 32.767               | 16 bits |
| unsigned int / unsigned long int                  | de 0 a 4.292.967.295              | 32 bits |
| int / long / long int / signed long int           | de -2.147.483.648 a 2.147.483.647 | 32 bits |

Fonte: Manzano (2013, p. 36).

- **real**: permite armazenar valores de pontos flutuantes e com frações. Em linguagem C é definido por "float", e quando precisa do dobro de dados numéricos é utilizado o tipo "double" ou "long double", veja na tabela 3.2.

Tabela 3.2 | Dados reais

| Tipo de dado real | Faixa de abrangência            | Tamanho |
|-------------------|---------------------------------|---------|
| float             | de $-3,4^{38}$ a $3,4^{38}$     | 32 bits |
| double            | de $-1,7^{308}$ a $1,7^{308}$   | 64 bits |
| long double       | de $-3,4^{4932}$ a $1,1^{4932}$ | 96 bits |

Fonte: Manzano (2013, p. 37).

- **Caractere**: caracteriza os caracteres, números e símbolos especiais, são delimitadas por aspas simples ('). Em linguagem C é definida por "char", veja a representatividade na tabela 3.3:

Tabela 3.3 | Dados reais

| Tipo de dado real  | Faixa de abrangência                                                                    | Tamanho |
|--------------------|-----------------------------------------------------------------------------------------|---------|
| char / signed char | de -128 a 127                                                                           | 8 bits  |
| unsigned char      | de 0 até 255                                                                            | 8 bits  |
| char               | Pode ser considerado <i>signed char</i> ou <i>unsigned char</i> , dependendo do sistema | 8 bits  |

Fonte: Manzano (2013, p. 37).

Podemos utilizar as variáveis do tipo de dado "void", são variáveis que não retornam um tipo definido, ou seja, não retorna um valor específico.

## Constantes

Segundo Schildt (2005), as constantes em linguagem C são consideradas modificadores de tipo de acesso, ou seja, não podem ser alteradas. Elas podem ser representadas pelo comando "const".

Exemplo:

```
const int art=100;
```

As constantes também podem ser caracterizadas por quaisquer tipos de dados básicos, por exemplo: as constantes do tipo texto são envolvidas por aspas simples (') ou aspas duplas. As aspas simples representam um único caractere, por exemplo, 'a' ou '100', e as aspas duplas caracterizam um conjunto de caracteres, por exemplo, "A conversão da temperatura de graus centígrados para graus Fahrenheit e".

As constantes inteiras são representadas por números inteiros negativos ou positivos, exemplo: -150 e 1500 são constantes inteiras (int).

Nas constantes do tipo flutuante são usados os comandos float e double, por exemplo: 10.235 é um número em ponto flutuante. Vale lembrar que os formatos em decimais dos números em linguagem C usam o padrão americano, onde no lugar da vírgula para as casas decimais usa-se o ponto.



Pesquise mais

Tipo de variáveis e constantes. No artigo abaixo, você terá a oportunidade de conhecer todas as particularidades dos tipos de variáveis e constantes. MC-102 Algoritmos e Programação de computadores. Disponível em: <<http://www.ic.unicamp.br/~ducatte/mc102/aula03.pdf>>. Acesso em: 18 mar. 2018

## Operadores em Linguagem C

### Operadores de atribuição

Para atribuir um valor a uma variável utilizamos o sinal de igual "=".

Exemplo:  $y = x + 100$

### Operadores aritméticos

Os operadores aritméticos em Linguagem C são representados por operadores **binários** e **unários**.

## Operadores Binários

+ soma

- subtração

\* multiplicação

/ divisão

% resto de divisão

Por exponenciação – lembre-se de que para esse operador a biblioteca `math.h` deve ser inicializada no programa.

## Operador Unário

Uma das representações unárias é caracterizada pela utilização do resto da divisão, representada pela porcentagem "%", que indica o resto dos operadores binários. Por exemplo:

`20%6` /\*O resultado é 2\*/

Segundo Mizrahi (2008), o operador unário pode ser usado para representar a troca de sinais de uma determinada variável, por exemplo:

`y = -10;`

`y = -y;`

Após essa operação, o valor de Y assume o valor de 10 positivo, lembrando que em linguagem C não existe a representação `y = +10`.

A raiz quadrada também é considerada um operador unário, representado pelo `"sqrt"`. Neste tipo de operador, a biblioteca `math.h` também deve ser inicializada no programa.

Importante lembrar que devemos obedecer às precedências entre os operadores, multiplicação (\*), divisão (/) e % e depois sobre a Adição (+) e subtração (-) que sempre serão executados da esquerda para direita. Vale salientar que as expressões dentro dos parênteses são as primeiras a serem executadas.

## Operadores de Incremento e Decremento

Quando você precisar adicionar um "1" à variável, você fará uso do "++" (incremento), e quando quiser tirar um "1" da variável, você utiliza "--" (decremento), essas operações são caracterizadas unárias. Veja os exemplos abaixo:

$y = y + 1$ ; utilizando incremento ficaria desta maneira:  $++y$

$y = y - 1$ ; utilizando o decremento ficaria desta maneira:  $--y$

Vale lembrar que o posicionamento dos "++" nas variáveis podem sofrer alterações nos seus valores quando representados, veja no exemplo abaixo:

$p = 10$ ;

$q = ++p$ ;

A saída para essa instrução será:

$p = 11$  e  $q = 11$

Agora, se usado "++" na frente da variável, ficará da seguinte maneira:

$p = 10$ ;

$q = p++$ ;

Utilizando o incremento na frente da variável significa que é adicionado "1" depois da sua utilização:

$p = 11$  e  $q = 10$

## Operadores Relacionais

Os operadores relacionais podem ser classificados da seguinte maneira na Linguagem C:

Quadro 3.4 | Operadores Relacionais

| Operador | Descrição      |
|----------|----------------|
| >        | Maior          |
| <        | Menor          |
| >=       | Maior ou igual |
| <=       | Menor ou igual |
| ==       | Igual          |
| !=       | Diferente      |

Fonte: elaborado pelo autor.

## Operadores lógicos

Quadro 3.5 | Operadores lógicos

| Operador | Descrição |
|----------|-----------|
| &&       | E         |
|          | OU        |
| !        | NÃO       |

Fonte: elaborado pelo autor.

Segundo Mizrahi (2008), o operador lógico “!” é considerado um operador unário e os operadores lógicos “&&” e “||” são binários.



### Pesquise mais

Existem vários programas de compilação que você pode utilizar para compilar o seu programa em Linguagem C, entre eles podemos destacar GCC, Visual C++ e DEV C++.

Você pode utilizar para o seu treinamento o software DEV C++ desenvolvido pela Bloodshed, no link abaixo é possível fazer o download e também conhecer as etapas de instalação e configuração:

Softonic. Ambiente completo para a programação nas linguagens C/ C++. Disponível em: <<https://dev-c.softonic.com.br/>>. Acesso em: 18 mar. 2018.

### Função main()

A função main() é reservada para iniciar um programa em C, sendo a primeira a ser executada.

```
main()
{
}
```

Quando utilizamos a “{” (chave aberta) indica o início de uma função em C, e quando usamos a “}” (chave fechada) indica o término das funções e do programa.

Quando usamos a int antes de main () significa que retornará um número do tipo inteiro.

```
int main ()
{
}
```

Também pode ser utilizada a palavra-chave `void`, esta é uma função sem retorno, ou seja, não recebe nenhum argumento.

```
void main ()
{
}
```

## Função `printf()`

A função `printf ()` é um comando de saída onde possui um vínculo com a biblioteca `stdio.h`. É utilizada quando se pretende obter uma resposta na tela do computador.

A sua síntese é definida por:

`printf ("expressão de controle", listas de argumentos);`

Existem algumas formatações na utilização da função `printf()` conforme mostra o quadro 3.6:

Quadro 3.6 | Código de formatação para função `printf()`

| Código            | Especificação                                                                                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>%c</code>   | Permite a escrita de apenas um caractere.                                                                                                                                                              |
| <code>%d</code>   | Permite fazer a leitura de números inteiros decimais.                                                                                                                                                  |
| <code>%e</code>   | Realiza-se a escrita de números em notação científica.                                                                                                                                                 |
| <code>%f</code>   | É feita a escrita de números reais (ponto flutuante).                                                                                                                                                  |
| <code>%g</code>   | Permite a escrita de <code>%e</code> ou <code>%f</code> no formato mais curto.                                                                                                                         |
| <code>%o</code>   | Permite que números octais sejam escritos.                                                                                                                                                             |
| <code>%s</code>   | Efetua-se a escrita de uma série de caracteres.                                                                                                                                                        |
| <code>%u</code>   | Escreve-se um número decimal sem sinal.                                                                                                                                                                |
| <code>%x</code>   | Permite a escrita de um número hexadecimal.                                                                                                                                                            |
| <code>%n[]</code> | Permite determinar entre colchetes quais caracteres podem ser ou não aceitos na entrada de uma sequência de caracteres, sendo "n" um valor opcional que determina o tamanho da sequência de caracteres |

Fonte: Manzano (2013, p. 38).

Veja no exemplo abaixo a aplicação de formatação da função `printf()`:  
`printf ("O valor encontrado foi %d", vl);`

Perceba que o valor da variável "v" foi posicionado no local do "%d", lembrando que "%d" é uma formatação para um dado do tipo inteiro.

Outras particularidades:

```
printf (" \n Resposta: a = %.2f e b = %.2f \n", a,b);
```

Neste exemplo, antes de apresentar a frase, o programa pulou uma linha "\n", o "%" é utilizado quando os dados numéricos são flutuantes, ou seja, valores fracionados, quando usamos %.2f significa que o valor será arredondado em duas casas decimais, ex: 2,45.

## Função scanf()

A função scanf() é um comando de entrada, ou seja, são informações que possibilitam a entrada de dados pelo teclado, assim, a informação será armazenada em um determinado espaço da memória. A sintaxe é definida por uma expressão de controle (sempre entre aspas duplas) e pela lista de argumento.

A sintaxe da função scanf() é definida por:

```
scanf("expressão de controle", lista de argumentos);
```

A função scanf() faz uso de alguns códigos de formação, veja a quadro 3.7:

Quadro 3.7 | Código de formatação para função scanf()

| Código    | Especificação                                               |
|-----------|-------------------------------------------------------------|
| %c        | Permite que seja efetuada a leitura de apenas um caractere. |
| %d        | Permite fazer a leitura de números inteiros decimais.       |
| %e        | Permite a leitura de números em notação científica.         |
| %f        | É feita a leitura de números reais (ponto flutuante).       |
| %l        | Realiza-se a leitura de um número inteiro longo.            |
| %o        | Permite a leitura de números octais.                        |
| %s        | Permite a leitura de uma série de caracteres.               |
| %u        | Efetua-se a leitura de um número decimal sem sinal.         |
| %x        | Permite que seja feita a leitura de um número hexadecimal.  |
| %[código] | Permite que seja feita uma entrada formatada pelo código.   |

Fonte: Manzano (2013, p. 38).

Veja a sintaxe abaixo:

```
scanf ("%d", &v);
```

Neste exemplo o computador entrará com um valor decimal e retornará o valor da variável "vl" .

O "&" é utilizado na função scanf() na lista de argumentos, sua função é retornar o conteúdo da variável, ou seja, retorna o endereço do operando.

```
main()
{
int num;
printf("Digite um número: ");
scanf("%d",&num);
printf("\n o número é %d",num);
printf("\no endereço e %u",&num);
}
```



### Refleta

Observe que no programa abaixo foi utilizada uma atribuição para variável. Qual o valor da variável na tela do computador?

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int v;
printf("\n Valor inicial da variavel v= %d", v);
v = 5;
printf("\n O valor da variavel e = %d \n", v*5);
return 0;
}
```

Neste caso em específico, não foi utilizado armazenamento em memória da variável.

Cabe ressaltar que você pode fazer comentários em qualquer lugar do seu programa, basta utilizar barras duplas "//".

Exemplo:

```
#include <stdio.h> // biblioteca para entrada e saída de dados
int main() // comando de início e o mais importante do programa
{ // início de uma função
```



```
printf("Meu primeiro programa"); // comando para saída de dados
na tela
return 0; // indica que o processo esta voltando para o Sistema
Operacional
} // fim de uma função ou de um programa
```

Segundo Manzano (2015), a instrução retorna zero "return 0" indica que o programa está sendo encerrando e que o processo que estava sendo executado vai retornar para o sistema operacional.

A instrução system("pause") tem a função de pausar a execução do programa, para que o resultado seja visualizado.



### Exemplificando

Muito bem! Agora vamos ver um exemplo de um programa realizado em Linguagem C, onde mostrará a idade de uma pessoa.

```
include <stdio .h>
int main ()
{
 int id ;
 printf ("Quantos anos voce tem?: ") ;
 scanf ("%d" , &id) ;
 printf ("%d? Nossa, voce parece que tem %d anos !\n" , id , id*2);
 system ("pause");
 return 0 ;
}
```

Maravilha, esse foi o seu primeiro passo para programação em linguagem C. É muito importante a sua dedicação e treinamento. Boa sorte e bons estudos!

## Sem medo de errar

Chegou o momento de resolver a situação problema proposta pela seção, a Kro engenharia precisa que todos os engenheiros da empresa aprendam a programação em linguagem C. O seu desafio é, junto com os engenheiros, desenvolver um programa em linguagem C que faça a conversão da temperatura de graus

centígrados (Celsius) para graus Fahrenheit. A fórmula para essa conversão é:  $F = \frac{9.c + 160}{5}$ , onde "F" é a temperatura em Fahrenheit e "c" é a temperatura em centígrados.

Lembrando que você junto aos engenheiros deverão apresentar as rotinas utilizadas no programa, os comentários e a compilação do programa.

Muito bem, para resolver esse problema, você deverá realizar a programação. Veja o modelo apresentado abaixo:

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main()
{
 float fahrenheit;
 float celsius;
 printf("Conversão de graus centígrados para graus Fahrenheit
 \n");
 printf("Digite o valor em graus Celsius: \n");
 scanf("%f", &celsius);
 fahrenheit = (9*celsius+160)/5;
 printf("O valor em Fahrenheit e: %.2f \n", fahrenheit);
 return 0;
}
```

Já com a programação pronta, é o momento de colocar os comentários, para isso, no final de cada linha utilizar o "//" para iniciar os comentários.

Utilizando um computador com o DEV C++ instalado, faça a compilação do programa.

Tente outras maneiras de criar o programa, faça testes e pratique o máximo possível.

## Avançando na prática

### A força

#### Descrição da situação-problema

Em um debate com seus amigos engenheiros, você foi desafiado a resolver através do cálculo da força e depois programar em Linguagem C a seguinte situação:

Imagine uma placa de cimento sobre uma plataforma plana onde será necessário calcular a força normal uma na outra. A ideia do cálculo é não deixar que o objeto sólido seja atravessado por outros. Veja que a força da gravidade empurra a placa para baixo, ao mesmo tempo em que a força normal vai contra ela.

Para realizar esses cálculos, você poderá usar a seguinte fórmula:

$$N = m \cdot g$$

Onde:

N: força normal

m: massa do objeto

g: gravidade

Depois que você entendeu a lógica da proposta, faça o programa em linguagem C.

#### Resolução da situação-problema

Para resolver essa situação, a sugestão de programação é a seguinte:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
 float n, m, g;
 printf("Digite a massa do objeto:\n");
```

```
scanf("%f", &m);
printf ("Digite a gravidade:\n");
scanf("%f", &g);
n=(m*g);
printf("O calculo da forca e: %f \n", n);
return 0;
}
```

Você ainda pode melhorar e diferenciar a programação.  
Boa sorte e bons estudos!



**Pesquise mais**

Caro aluno! Os assuntos abordados nesta jornada se aplicam às linguagens mais modernas de hoje. Você aprenderá o que é programação, quais linguagens de programação estão disponíveis e quais ferramentas de desenvolvedor usar. Aproveite esta jornada e bons estudos!!++.

Introdução à programação - Learn | Microsoft Docs: <https://docs.microsoft.com/pt-br/learn/modules/web-development-101-introduction-programming/>



## Faça valer a pena

**1.** Para executar um programa em linguagem C é necessário um compilador onde sua função segundo Schildt (2005) é processar os códigos em linguagem de alto nível para linguagem baixo nível. O compilador processa os códigos do programa e verifica se estes não possuem erros na sua execução.

Qual o procedimento que tem a função de vincular todos os arquivos gerados em um único arquivo executável? Assinale a alternativa correta:

- a) Compilação.
- b) Linker.
- c) Processamento.
- d) Leitura.
- e) Escrita.

**2.** Quando você precisar adicionar um "1" à variável, você fará uso do "++" (incremento), e quando quiser tirar um "1" da variável, você utiliza "- -" (decremento), essas operações são caracterizadas unárias, porém, o posicionamento dos "++" nas variáveis podem sofrer alterações nos seus valores quando representados.

Analise o programa abaixo e responda a alternativa que corresponde ao valor para p e q.

```
int main()
{
 int p,q;
 p=1;
 q=1;
 printf("Valor de p = %d\n", p);
 printf("Valor de q = %d\n", q);
 printf("\n Criando o Incremento: p++\n");
 p++;
 printf("Criando um Decremento: q--\n");
 q--;
 printf("\n O novo valor para p = %d\n", p);
 printf(" O novo valor para q = %d\n",q);
 return 0;
}
```

- a) O novo valor para "p" é 1 e o novo valor de "q" é 0.
- b) O novo valor para "p" é 0 e o novo valor de "q" é 1.
- c) O novo valor para "p" é 1 e o novo valor de "q" é 1.
- d) O novo valor para "p" é 0 e o novo valor de "q" é 2.
- e) O novo valor para "p" é 2 e o novo valor de "q" é 0.

**3.** Analise o programa abaixo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(void)
4 {
5 float nota1,nota2,media;
6 printf("\n\nDigite a primeira nota: ");
7 scanf("%f",¬a1);
8 printf("Digite a segunda nota: ");
9 scanf("%d",¬a2);
10 media = (nota1 + nota2)/2;
11 printf("Media do aluno = %d\n",media);
12 return 0;
13 }
```

Considerando as afirmações abaixo, responda a alternativa correta.

I- A linha 3 do programa está errada, pois dentro do comando "main" não pode ser usado o tipo "void".

II- Se mantiver a linha 9 e 12 com "%d", o programa será compilado mesmo com o tipo de variável diferente da declarada, porém, a resposta para a média do aluno será "0".

III- Na linha 6 do programa, a instrução \n\n faz com que o programa pule duas linhas antes do início da frase.

- a) Somente a alternativa I está correta.
- b) As alternativas I e II estão corretas.
- c) Somente a alternativa II está correta.
- d) As alternativas II e III estão corretas.
- e) Somente a alternativa III está correta.

## Seção 3.2

### Estruturas condicionais em linguagem C

#### Diálogo aberto

Caro aluno, na seção anterior, você teve a oportunidade de conhecer o funcionamento da linguagem C, aprendeu a definir as suas variáveis, constantes e operadores, assim como criar expressões. Agora é o momento de deixar tudo mais interessante.

Vamos lá!

Nesta seção, você será levado a conhecer a execução sequencial, as estruturas condicionais simples e composta, assim como as estruturas condicionais encadeadas e de seleção de casos.

Não tenha receio, a partir de agora, você terá a oportunidade de conhecer as mais diversas formas de programação, a fim de chegar a um mesmo objetivo. Pense que você é um chef de cozinha e que para criar uma receita existem diferentes possibilidades.

Pois bem! A multinacional Kro Engenharias precisa que você auxilie os engenheiros a desenvolverem uma programação em linguagem C que calcule as seguintes fórmulas: fórmula do movimento Retilíneo Uniforme, onde  $S = S_0 + V \cdot t$  (fórmula para medir o tempo, espaço e Velocidade), e a fórmula do Movimento Retilíneo Uniformemente Variado, onde  $S = S_0 + V_0 \cdot t + \frac{1}{2} a \cdot t^2$  (Fórmula para medir o tempo, espaço e velocidade no MRUV). Lembrando que o usuário deverá ter a opção de escolher entre uma das fórmulas. Você deverá apresentar os códigos do programa ao seu professor após a sua compilação, recordando que você poderá usar a estrutura condicional encadeada e a seleção de casos.

Boa sorte e ótimos estudos.

## Não pode faltar

Caro aluno, na primeira seção desta unidade, você teve a oportunidade de trabalhar os conceitos e estruturas da Linguagem C, entre elas: a estrutura de um programa em Linguagem C, os tipos de bibliotecas, as variáveis e constantes e finalizando com algumas funções em Linguagem C.

Agora, chegou o momento de conhecer algumas das formas de estruturas condicionais em linguagem C. Vamos voltar um pouco na execução sequencial de um programa e logo em seguida trabalhar as estruturas condicionais simples, compostas, encadeadas e de seleção de casos.

Então, vamos em frente!

### Execução sequencial

Pois bem, seguimos falando de execução sequencial, que na verdade nada mais é do que um comando sendo executado, um após o outro. Podemos dizer que é a sequência em que foram escritos.

Veja o programa abaixo que executa o raio e perímetro de um círculo de forma sequencial:

```
#include <stdio.h>

int main() {
 float raio, area, perimetro, pi;
 printf("Digite o raio: ");
 scanf("%f", &raio);
 pi = 3.141592;
 area = pi*(raio * raio);
 perimetro = 2.0 * pi * raio;
 printf(" \n Raio: %.2f \n", raio);
 printf(" \n Area: %.2f \n", area);
 printf(" \n Perimetro: %.2f \n", perimetro);
 return 0;
}
```

Neste exemplo, podemos observar que o programa foi realizado usando um sequenciamento de comandos até a sua conclusão.



Antes de dar início à estrutura condicional, é importante que você, aluno, recorde de alguns operadores relacionais usados para comparação, por exemplo:

Expressão "<" verifica se um valor é menor que o outro.

Expressão ">" verifica se um valor é maior que o outro.

Expressão "<=" verifica se um valor é menor ou igual que o outro.

Expressão ">=" verifica se um valor é maior ou igual que o outro.

Expressão "==" verifica se um valor é igual ao outro.

Expressão "!=" verifica se um valor é diferente do outro.



## Assimile

Para não perder nada pelo caminho, vale assimilar os operadores lógicos, assim como a tabela verdade:

Tabela 3.4 | Operadores lógicos

| Operadores | Função                   |
|------------|--------------------------|
| !          | Negação - NOT            |
| &&         | Conjunção - AND          |
|            | Disjunção Inclusiva - OR |

Tabela Verdade

| A       | B       | A && B  | A    B  | ! A     | ! B     |
|---------|---------|---------|---------|---------|---------|
| Verdade | Verdade | Verdade | Verdade | Falso   | Falso   |
| Verdade | Falso   | Falso   | Verdade | Falso   | Verdade |
| Falso   | Verdade | Falso   | Verdade | Verdade | Falso   |
| Falso   | Falso   | Falso   | Falso   | Verdade | Verdade |

Muito bem, agora podemos prosseguir.

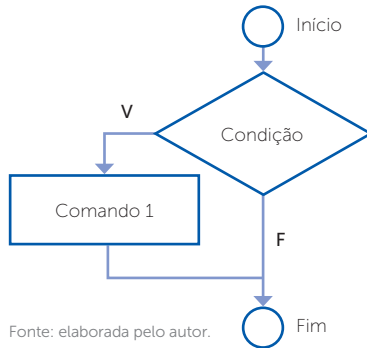
## Estrutura condicional simples

Segundo Manzano (2013), para a solução de um problema, podemos utilizar a instrução "if", em português "se", onde sua função é tomar uma decisão e criar um desvio dentro do programa, onde, desta forma, podemos chegar a uma condição que pode ser verdadeira ou falsa. Lembrando que a instrução pode receber

valores em ambos os casos. Para determinar o início e fim de uma instrução, iremos utilizar os símbolos de chaves “{” e “}”.

Veja como fica a estrutura condicional simples no fluxograma da figura 3.7.

Figura 3.7 | Fluxograma representando a função “if”



Na sequência, veja a sintaxe da instrução “if” (se) utilizada na linguagem C:

```
if <(condição)>
{
 <conjunto de comandos>;
}
```

Muito bem, nada melhor do que visualizar uma aplicação deste tipo de condição senão aplicando-a na prática. No exemplo abaixo, foi criada uma condição para que o programa execute o valor do projeto e se este for maior que R\$ 20.000,00, deverá aparecer uma mensagem na tela mostrando que o projeto não é viável.

```
#include <stdio.h>
main()
{
 char projeto[20];
 float orcamento, imposto, total;
 printf("\n Digite o nome do projeto \n");
```

```

scanf("%s", &projeto);
printf("\n Digite o valor do orcamento\n ");
scanf("%f", &orcamento);
printf("\n digite o valor dos impostos \n");
scanf("%f", &imposto);
total = orcamento + imposto;
printf ("\nO Valor do projeto e = %.2f\n", total);
if(total > 20000)
{
 printf("\n \n O %s Nao e viavel\n", projeto);
}
return(0);
}

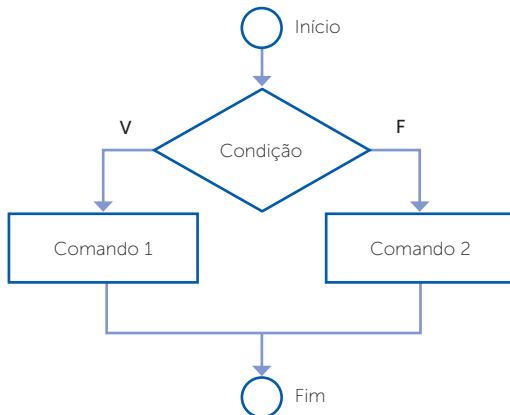
```

Muito bom, veremos agora a estrutura condicional composta.

### Estrutura condicional composta

A estrutura condicional composta é bem parecida com a estrutura condicional simples, a diferença é: quando uma condição não é satisfeita, um outro comando será executado. Neste caso, vamos usar o comando "else", que significa "então". Veja como fica esta estrutura no fluxograma da figura 3.8:

Figura 3.8 | Fluxograma representando a função "if" e "else".



Agora, veja a representação da sintaxe:

```
if <(condição)>
{
<conjunto de comandos>;
}
else
{
<conjunto de comandos>;
}
```

Muito bom, agora veja como fica a aplicação do comando "else" utilizando o programa abaixo:

```
#include <stdio.h>
int main() {
float orcamento;
char projeto[20];
printf("Digite o nome do projeto \n");
scanf("%s", &projeto);
printf("Digite o valor do orcamento do projeto \n");
scanf("%f", &orcamento);
if (orcamento <=20000)
{
printf("\nO projeto %s podera ser executado, seu valor foi inferior ou igual a R$20000", projeto);
}
else
{
printf("\nO projeto %s nao podera ser executado, seu valor foi superior a R$20000 ", projeto, orcamento);
}
return 0;
}
```

E aí, melhorou o pensamento? Ficou bem mais estruturado e o comando "else" possibilitou um retorno à condicional "if".



## Exemplificando

Veja no exemplo abaixo a aplicação de um programa em linguagem C que retorna se o valor de um número digitado é positivo ou negativo, representando uma estrutura condicional composta:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int num;
 printf("Digite um numero: ");
 scanf ("%d",&num);
 if (num>=0)
 {
 printf ("\n\nO numero e positivo\n");
 }
 else
 {
 printf ("O numero e negativo");
 }
 return 0;
}
```

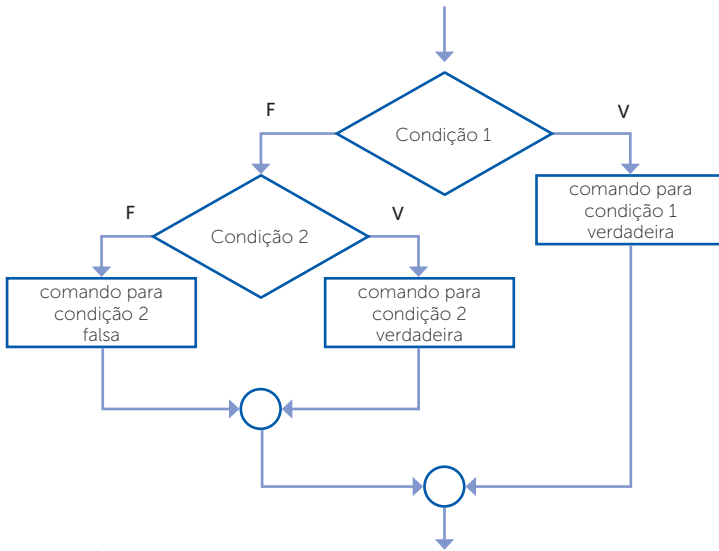
Muito bem! Agora vamos trabalhar a estrutura condicional encadeada

### Estrutura condicional encadeada

A estrutura condicional encadeada é também conhecida como ifs aninhados, que, segundo Schildt (1997), é um comando if que é o objeto de outros if e else. Em resumo, sempre um comando else estará ligado ao comando if de seu nível de aninhamento.

Veja na figura 3.9 um dos tipos de fluxogramas que representa uma estrutura condicional encadeada.

Figura 3.9 | Fluxograma estrutura condicional encadeada



Fonte: elaborada pelo autor.

Podemos caracterizar a sintaxe de uma estrutura condicional encadeada da seguinte forma:

```
if (condição) comando;
else
 if (condição) comando;
 else(condição) comando;
 .
 .
 .
else comando;
```

Muito bem! Agora veja no programa abaixo uma estrutura condicional encadeada, onde será calculado o intervalo entre dois períodos de tempo.

```

#include <stdio.h>
int main() {
 int h1, min1, h2, min2, h, min;
 printf("\nDigite o instante inicial (horas e minutos)\n");
 scanf("%d %d", &h1, &min1);
 printf("\nDigite o instante final\n");
 scanf("%d %d", &h2, &min2);
 h = h2 - h1;
 min = min2 - min1;
 if ((h < 0) || ((h == 0) && (min < 0)))
 printf("\nDados invalidos! O segundo instante e anterior ao
primeiro\n");
 else
 {
 if (min < 0)
 {
 h = h - 1;
 min = min + 60;
 }
 printf("\nEntre os periodos %dh %dmin e %dh %dmin passaram-
se %dh %dmin", h1, min1, h2,
min2, h, min);
 }
 return 0;
}

```



**Pesquise mais**

Assista à videoaula desenvolvida pelo Bóson Treinamentos que fala sobre as estruturas condicionais encadeadas. 13 – Programação em Linguagem C. Bóson Treinamentos. Youtube. 27 fev. 2015. Disponível em: <<https://www.youtube.com/watch?v=7ZL8tHLTTfs>>. Acesso em: 18 mar. 2018.

## Estrutura condicional de seleção de casos

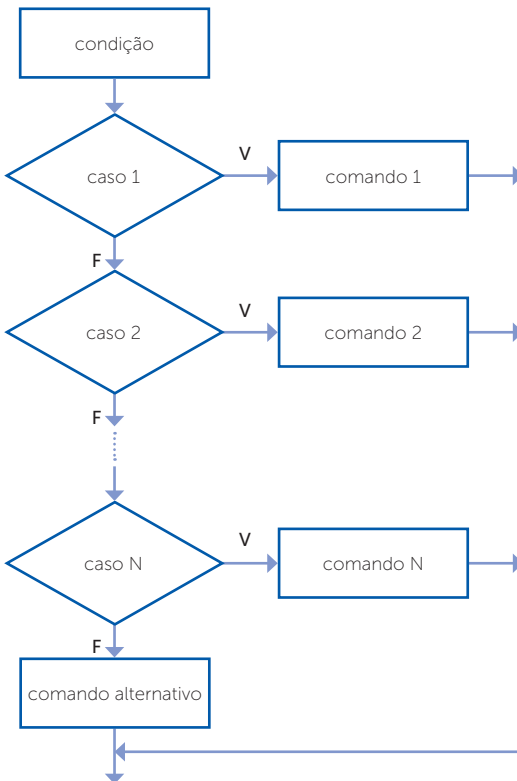
A Estrutura Condicional de seleção de casos “switch-case” é aplicada quando usamos uma variável do tipo inteiro para comparar ou testar determinados valores. Quando os valores são avaliados, o comando é executado. Em alguns casos é uma ótima opção para substituir a programação de estrutura condicional encadeada.

Caso nenhum dos valores seja encontrado, o comando default será executado.

Lembrando que os comandos são executados até o ponto que o comando break for localizado.

Veja na figura 3.10 o fluxograma representando a estrutura switch-case:

Figura 3.10 | Fluxograma switch-case



Fonte: elaborada pelo autor.



Agora, veja a sintaxe:

```
switch (variável)
{
case constante1:
<comandos>
break;
case constante2:
<comandos>
break;
default: <comandos>
}
```

Veja que o comando `break` é utilizado para forçar a saída do laço de repetição, ou seja, ele sai do comando sem executar as próximas instruções.

Bom, para fixar o que está sendo estudado, vamos aplicar um exemplo, onde a finalidade é ter como opção a área e o perímetro de um círculo, a partir do valor do raio:

```
#include <stdio.h>

int main() {
float area, raio, pi, perimetro;
int tipo;
pi = 3.141592;
printf("\nDigite o valor para a raio\n");
scanf("%f", &raio);
printf("Digite o que se deseja sobre circulo");
printf("\n 1 para area do circulo\n");
printf("\n 2 para perimetro\n");
printf("\n");
scanf("%d",&tipo);
switch (tipo)
{
```

```

case 1:
printf("O valor para a area do circulo e: %.2f\n", area=pi*(raio*raio));
break;
case 2: printf("O valor para o perimetro do circulo e: %.2f",
perimetro = 2.0 * pi * raio);
break;
default: printf("Tipo invalido");
}
return 0;
}

```



### Refleta

Pense nas possibilidades que você pode ter usando as estruturas de tomadas de decisão "if-else", if-else-if e "switch-case". Lembre-se de que cada caso poderá ter uma particularidade diferente em desenvolver um programa. Imagine se você tivesse que criar um programa que calculasse os prêmios dos engenheiros da Kro Engenharia, de acordo com a produtividade de cada um, como você faria?

Muito bem! Encerramos mais uma seção do nosso livro, lembre-se de que a prática levará à perfeição. Bons estudos e até a próxima seção!

## Sem medo de errar

Chegou o momento de colocar em prática o conteúdo estudado, recordando que o seu desafio é auxiliar os engenheiros da multinacional Kro Engenharia para que desenvolvam uma programação em linguagem C que calcule as seguintes fórmulas: fórmula do movimento Retilíneo Uniforme, onde  $S = S_0 + V \cdot t$  (fórmula para medir o tempo, espaço e Velocidade), e a fórmula do Movimento Retilíneo Uniformemente Variado, onde  $S = S_0 + V_0 \cdot t + 1/2 a \cdot t^2$  (Fórmula para medir o tempo, espaço e velocidade no MRUV).

No desenvolvimento do programa, você deverá colocar duas opções para acessar as formulas. Em seguida encaminhar para o seu professor o resultado do código.

Vamos lá!

Na solução desta atividade, você poderá utilizar duas formas de programa, a estrutura condicional encadeada e a seleção de casos.

Caso sua opção seja a estrutura encadeada ("if- else-if"), você pode usar a seguinte sintaxe:

```
if (condição) comando;
else
 if (condição) comando;
 else(condição) comando;
.
.
.
else comando;
```

E se for usar a estrutura de seleção, poderá usar "switch-case" com a seguinte sintaxe:

```
switch (variável)
{
case constante1:
<comandos>
break;
case constante2:
<comandos>
break;
default: <comandos>
}
```

Lembre-se, sua missão neste momento é encontrar a solução da situação problema. Aconselho a realizar a programação utilizando tanto a estrutura condicional encadeada quanto a de seleção de casos.

Boa sorte e ótimos estudos!

### Tipos de Triângulos

#### Descrição da situação-problema

Você foi designado a resolver um problema no departamento de artes do instituto Arte Forma, eles precisam de um programa em linguagem C que identifique algumas figuras geométrica, mais especificamente, alguns tipos de triângulos, e, para isso, você deve se atentar às seguintes condições:

1. Você solicitará ao usuário entrar com as medidas das figuras.
2. O programa deverá verificar se as medidas correspondem a um tipo de triângulo.
3. Se a condição for satisfatória, ou seja, verdadeira, você deverá mostrar se essas medidas correspondem a um triangulo equilátero, isósceles ou escaleno.
4. Não atendendo à condição verdadeira, o seu programa deverá retornar informando que as medidas não correspondem a um triângulo.

Existem algumas formas para solução deste problema, use a sua criatividade e diversidade para essa programação.

#### Resolução da situação-problema

Podemos usar a seguinte programação para soluções deste problema:

```
#include <stdio.h>
int main() {
 int a, b, c;
 printf("Classificacao do triangulo: informe a medida dos lados:\n");
 scanf("%d %d %d", &a, &b, &c);
 if (a < b + c && b < a + c && c < a + b)
 {
 printf("\n\n Dadas as medidas: %d, %d, %d, temos um
 triangulo", a, b, c);
 if(a == b && a == c)
```

```

 {
 printf("Este e um triangulo EQUILATERO! \n");
 }
 else
 if (a==b | a == c | b ==c)
 {
 printf("Este e um triangulo ISOSCELES!\n");
 }
 else
 printf("Este e um triangulo ESCALENO! \n");
 }
 else
 printf("\n\n As medidas fornecidas, %d,%d,%d nao formam
 um triangulo", a, b, c);
return 0;
}

```

Tente diversificar o máximo possível a sua programação, você pode utilizar o programa sugerido e aplicar a estrutura de seleção de caso "switch-case". Boa sorte e bons estudos!

## Faça valer a pena

**1.** Analise a seguinte programação abaixo:

```

Int main() {
 char x;
 printf("1. inclusao\n");
 printf("2. alteracao\n");
 printf("3. exclusao\n");
 printf(" Digite sua opcao:");
 x=getchar();
 switch(x)
 {
 case '1':
 printf("escolheu inclusao\n");
 break;

```

```

case '2':
 printf("escolheu alteracao\n");
 break;
case '3':
 printf("escolheu exclusao\n");
 break;
default:
 printf("opcao invalida\n");
}
return 0;
}.

```

Após analisar o programa, assinale a alternativa que corresponde a mensagem de retorno se o usuário digitar a opção "4":

- a) Escolheu exclusão.
- b) Opção inválida.
- c) "0".
- d) Escolheu inclusão.
- e) Tela em branco sem nenhum retorno.

**2.** Quando usamos a estrutura condicional encadeada, sabemos que várias opções deverão ser analisadas, entre elas, se a declaração de um "if" está sendo declarado dentro de um outro "if" externo.

Analise a afirmação acima e assinale a alternativa que melhor se ajusta na estrutura condicional encadeada:

- a) Deverá ser analisado se o "if" tem ligação com um "else".
- b) Podem ser criados vários "if", independentemente dos "else".
- c) Só é possível criar "if-else-if" se tiver ligação com outro "if" interno.
- d) Não é necessário a utilização do "else" na estrutura condicional encadeada.
- e) Obrigatoriamente exige um "if" após outro "if", sem uso do "else".

**3.** Analise o programa abaixo:

```

#include <stdio.h>
int main ()
{

```

```

int num;
printf ("Digite um numero: ");
scanf ("%d",&num);
if (num==99)
{
 printf ("\n\nQue sorte! Voce acertou!\n");
 printf ("Realmente o numero escolhido foi 99\n");
}
.....
.....
return(0);
}

```

Assinale a alternativa que melhor se encaixa no programa acima:

a) else

```

{
 if (num>99)
 {
 printf ("O numero que voce digitou e acima do escolhido.");
 }
}

```

b) else

```

{
 if (num>99)
 {
 printf ("O numero que voce digitou e acima do escolhido.");
 }
 if
 {
 printf ("O numero e menor que o escolhido");
 }
}

```

c) if

```

{
 if (num>99)
 {
 printf ("O numero que voce digitou e acima do escolhido.");
 }
 else
 {
 printf ("O numero e menor que o escolhido");
 }
}

```

```
d)else
{
 if (num>99)
 {
 printf ("O numero que voce digitou e acima do escolhido.");
 }
 else
 {
 printf ("O numero e menor que o escolhido");
 }
}

e) else
{
 if (num>99)
 printf ("O numero que voce digitou e acima do escolhido.");
 else
 {
 printf ("O numero e menor que o escolhido");
 }
}
```



## Seção 3.3

### Estruturas de repetição em linguagem C

#### Diálogo aberto

Muito bem! Caro aluno, na seção anterior, você foi levado a conhecer a execução sequencial, as estruturas condicionais simples e composta, assim como as estruturas condicionais encadeadas e de seleção de casos. Neste momento, você vai desenvolver técnicas para trabalhar as estruturas de repetição.

Pois então, o que quer dizer estrutura de repetição?

Estrutura de repetição são funções utilizadas para realizar instruções mais de uma vez, podendo criar *loops* com um número limitado de vezes, onde será trabalhada a repetição com teste no início, no final e com variáveis de controle.

A empresa Kro Engenharia incumbiu você em orientar a equipe de engenheiros a realizar um programa que calcule o fatorial de um número natural  $n$ . ( $n! = n \cdot (n - 1)! = n \cdot (n - 1) \cdot (n - 2)! = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \dots \cdot 1!$ ). Você deverá fazer a demonstração utilizando repetição com teste no início e repetição com variáveis de controle, compilar o programa e entregar a rotina de programação para o professor.

Perfeito! Agora, vamos desenvolver algumas situações que remetem à estrutura de repetição.

Boa sorte e bons estudos!

## Não pode faltar

Caro aluno, na seção anterior, você estudou a estrutura condicional simples, composta, encadeada e de seleção de casos, agora, é o momento de trabalhar a repetição com teste no início, no final e com variáveis de controle.

Lembrando que, segundo Manzano (2013), para solução de um problema, é possível utilizar a instrução "if" para tomada de decisão e também criar desvios dentro de um programa para uma condição verdadeira ou falsa.

### Repetição com teste no início - while

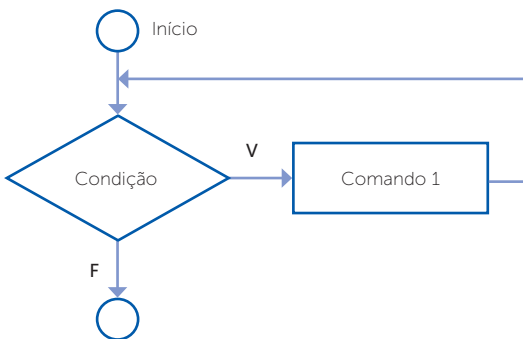
Para entender o que é uma repetição com teste no início, você precisa estar ciente de que algo será repetidamente executado enquanto uma condição verdadeira for verificada, somente após a sua negativa essa condição será interrompida.

Segundo Soffner (2013, p. 64), o programa "não executará nenhuma repetição (e as ações que ali dentro estiverem programadas) sem antes testar uma condição".

Para realizar a repetição com teste no início, você usará o comando iterativo "while", que significa em português "enquanto".

Podemos utilizar o seguinte fluxograma para trabalhar o teste no início:

Figura 3.11 | Fluxograma do comando while



Fonte: elaborada pelo autor.

Assim como caracterizado no fluxograma, podemos transcrever a repetição com teste no início, com a seguinte sintaxe:

```
while (<condição>
{
Comando 1;
Comando 2;
Comando n;
}
```

Quando trabalhamos com teste no início, precisamos estar atentos para que não ocorra um *loop* infinito, por este motivo, você poderá fazer o uso de:

**Contador** – é utilizado para controlar as repetições, quando esta é determinada.

**Incremento e decremento** – trabalha o número do contador, seja ele aumentado ou diminuído.

**Acumulador** – que, segundo Soffner (2013), irá somar as entradas de dados de cada iteração da repetição, gerando um somatório a ser utilizado quando da saída da repetição.

**Condição de parada** – utilizada para determinar o momento de parar quando não se tem um valor exato desta repetição.



Refleta

Podemos usar dois tipos de contadores em linguagem de programação "C", um é definido pela variável seguida de dois sinais de adição, por exemplo: `x++`. Qual a outra forma que podemos utilizar para trabalhar com contador?

Vamos ver alguns exemplos específicos de repetição com teste no início:

Exemplo 1: Determina que o valor inicial será contado até o final da condição (while)

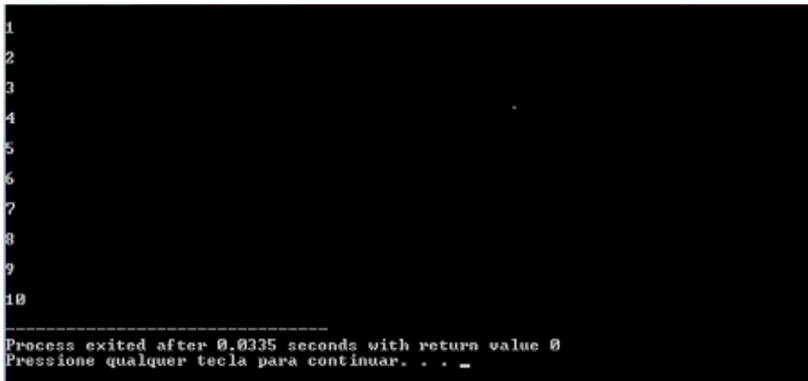
```

#include <stdio.h>
main()
{
int contador = 1; //iniciando a variavel
 while (contador <= 10) // Teste de inicio
 {
 printf("\n%d\n", contador); //Executa o comando
 contador++; // controle do contador
 }
return 0;
}

```

Veja que no exemplo 1 o contador teve o seu valor iniciado em um "1", portanto, pode sofrer alterações de acordo com a condição exposta pelo programador.

Figura 3.12 | Resultado do código usando o comando while.



```

1
2
3
4
5
6
7
8
9
10

Process exited after 0.0335 seconds with return value 0
Pressione qualquer tecla para continuar. . . _

```

Fonte: elaborada pelo autor.

No exemplo 2, podemos notar que para calcular uma determinada tabuada o programador determinou que o usuário entrasse com o valor inicial e final.

Exemplo 2:

```

#include <stdio.h>
main() {
int final, y, inicial;
printf("T A B U A D A \n\n");

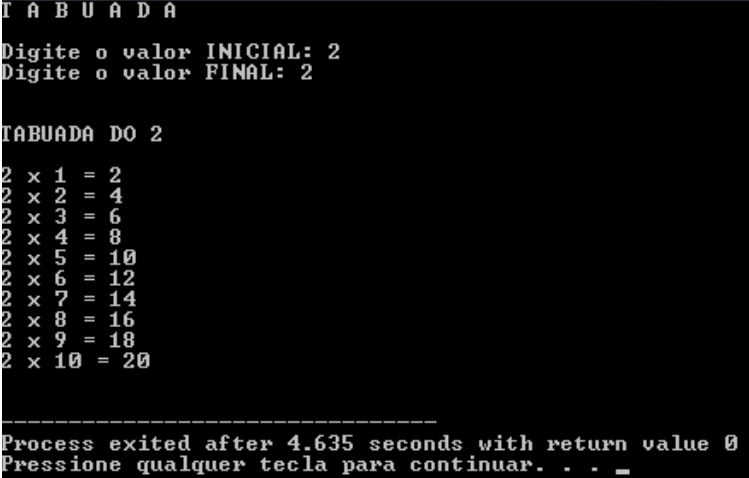
```

```

printf("Digite o valor INICIAL: ");
scanf("%d", &inicial);
printf("Digite o valor FINAL: ");
scanf("%d", &final);
printf("\n");
while(inicial <= final)
{
printf("\nTABUADA DO %d\n\n", inicial);
y = 1;
while(y <= 10)
{
printf("%d x %d = %d\n", inicial, y, inicial*y);
y = y + 1;
}
inicial = inicial + 1;
}
printf("\n");
return 0;
}

```

Figura 3.13 | Resultado do código com entradas de valores pelo usuário



```

T A B U A D A
Digite o valor INICIAL: 2
Digite o valor FINAL: 2

TABUADA DO 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

Process exited after 4.635 seconds with return value 0
Pressione qualquer tecla para continuar. . . _

```

Fonte: elaborada pelo autor.

## Repetição com teste no final – do-while

Segundo Schildt (1997), o laço “do-while” analisa a condição ao final do laço, ou seja, os comandos são executados antes do teste de condição.

O interessante deste comando é que o usuário tem a possibilidade de digitar novamente uma nova informação.

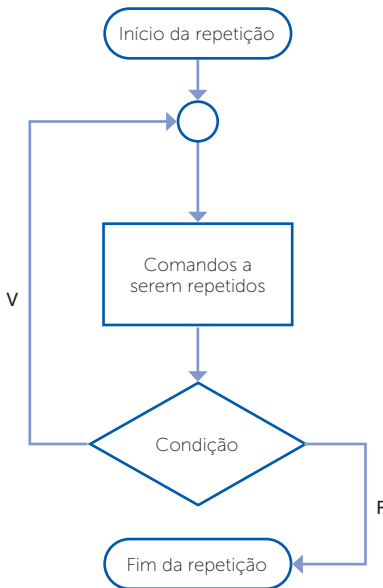


Pesquise mais

O comando do-while pode ter várias aplicações, veja o vídeo no YouTube do canal “De aluno para aluno” sobre o tema “Programar em C - Como Utilizar ‘do while’ - Aula 13”. 24 out, 2012. Disponível em: <<https://www.youtube.com/watch?v=bjMD1QSVV-s>>. Acesso em: 18 mar. 2018.

Veja como fica o fluxograma utilizando o teste de repetição no final:

Figura 3.14 | Fluxograma com teste de repetição no final



Fonte: elaborada pelo autor.

Agora, veja a sintaxe para realização da repetição com teste no final:

```
do
{
comandos;
}
while (condição);
```

Exemplo 1: Programa que calcula a metragem quadrada de um terreno usando o teste no final para criar a opção de digitar novos valores sem sair do programa.

```
#include <stdio.h>
main() {
float metragem1,metragem2,resultado;
int resp;
metragem1 = 0;
metragem2 = 0;
resultado = 0;
do
{
printf("CALCULO DE METROS QUADRADOS");
printf("\n \n Digite a primeira metragem do terreno: \n");
scanf("%f",&metragem1);
printf("\n Digite a segunda metragem do terreno: \n");
scanf("%f",&metragem2);
resultado = (metragem1 * metragem2);
printf("\n \n O Terreno tem = %.2f M2 \n",resultado);
printf("Digite 1 para continuar ou 2 para sair\n");
scanf("%d", &resp);
}while (resp==1);
return 0;
}
```

Figura 3.15 | Resultado de código com teste do final

```
C A L C U L O D E M E T R O S Q U A D R A D O S
Digite a primeira metragem do terreno:
25
Digite a segunda metragem do terreno:
15
O Terreno tem = 375.00 M2
Digite 1 para continuar ou 2 para sair
1
C A L C U L O D E M E T R O S Q U A D R A D O S
Digite a primeira metragem do terreno:
20
Digite a segunda metragem do terreno:
10
O Terreno tem = 200.00 M2
Digite 1 para continuar ou 2 para sair
2

Process exited after 51.7 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Fonte: elaborada pelo autor.

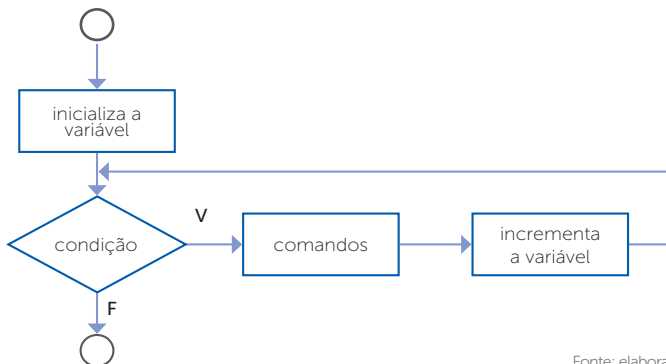
Segundo Schildt (2005), uma das aplicações para o laço do-while é a implementação de rotinas de menu, onde a sua função desejada é executada pelo menos uma vez dentro do laço.

Muito bem, já trabalhamos a repetição com teste no início e com teste no final, agora chegou o momento de aprender como utilizar a repetição com variáveis de controle.

### Repetição com variáveis de controle – Laço “for”.

O comando iterativo “for”, que em português significa “para”, segundo Mizrahi (2008), é geralmente usado para repetir uma informação por um número fixo de vezes, ou seja, podemos determinar quantas vezes acontecerá a repetição.

Figura 3.16 | Fluxograma com repetição com variáveis de controle



Fonte: elaborada pelo autor.



A sintaxe fica da seguinte maneira:

```
for(inicialização; condição final; incremento)
{
 comandos;
}
```

Vamos entender um pouco da sintaxe do “for” antes de partir para o exemplo. Você pode notar que temos três expressões separadas por ponto e vírgula, certo? Vamos entender um pouco de cada uma.

**Inicialização** – é neste momento que iremos colocar a instrução de atribuição, a inicialização é executada uma única vez antes de começar o laço.

**Condição final** – é realizado um teste onde é determinado se a condição é verdadeira ou falsa, enquanto for verdadeira permanece no laço e quando for falsa, encerra o laço e passa para a próxima instrução.

**Incremento** – parte das nossas explicações anteriores, onde é possível incrementar uma repetição de acordo com um contador específico, lembrando que o incremento é executado depois dos comandos.



## Assimile

Podemos usar o comando iterativo “for” em outras situações, como por exemplo:

Realizar testes em mais de uma variável:

```
for (i=1, x=0; (i + x) < 10; i++, x++);
```

Utilizar o comando “for” para representações de caracteres, por exemplo:

```
for(ch='d'; ch < 'm'; ch++).
```

Para ficar um pouco mais claro, vamos ao nosso exemplo:

Exemplo 4: Mostra a sequência de números, onde x vai de 10 a 0 e y vai de 0 a 10.

```
main()
```

```

{
 int x,y;
 for(x = 10,y = 0; x >= 0, y <= 10; x--,y++)
 {
 printf("%2d %2d\n",x,y);
 }
 return 0;
}

```

Figura 3.17 | Resultado do código com repetição de variáveis de controle

```

10 0
 9 1
 8 2
 7 3
 6 4
 5 5
 4 6
 3 7
 2 8
 1 9
 0 10

Process exited after 0.3545 seconds with return value 0
Pressione qualquer tecla para continuar. . .

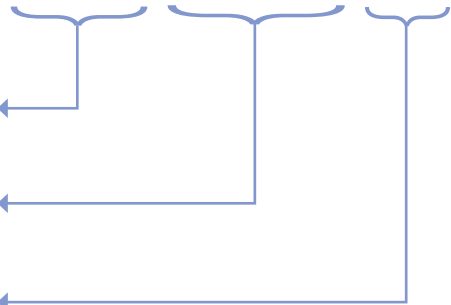
```

Fonte: elaborada pelo autor.

Perceba que na linha onde foi utilizado o laço “for” tivemos as seguintes expressões:

for(x = 10,y = 0; x >= 0, y <= 10; x--,y++)

- Na primeira expressão, “x” tem o seu valor iniciado em “10” e “y” iniciado em “0”.
- Na segunda expressão, “x” está condicionada a ser maior ou igual a “0” e “y” ser menor ou igual “10”.
- Para “x” será realizado o decremento 1 até atingir o seu valor final que é “0”, e “y” incrementado 1 para ao seu valor final que é “10”.





Você pode usar o comando “break” dentro de um laço “for” para uma determinada condição, focando assim, o término do laço, veja o exemplo abaixo:

```
#include <stdio.h>
main()
{
int w;
for (w = 1; w <= 15; w++)
{
if (w == 8)
{
break;
}
printf ("%d ", w);
}
printf("\n \n Parar a condicao de repeticao w = %d \n",
w);
return 0;
}
```

Existem várias maneiras de aplicar a estrutura de repetição dentro de um programa, cabe a você, agora, realizar todos os exemplos trabalhados nesta seção e nas situações problemas e praticar.

Boa sorte e bons estudos!

## Sem medo de errar

A empresa Kro Engenharia incumbiu você em orientar a equipe de engenheiros a realizar um programa que calcule o fatorial de um número natural  $n$ .  $(n! = n \cdot (n - 1)! = n \cdot (n - 1) \cdot (n - 2)! = n$

. (n - 1) . (n - 2) . (n - 3) . ... . 1!) você deverá fazer a demonstração utilizando repetição com teste no início e repetição com variáveis de controle.

Para solução deste desafio, você poderá usar as seguintes programações:

Repetição com teste no início

```
while (<condição>
{
Comando 1;
Comando 2;
Comando n;
}
```

```
#include <stdio.h>
#int main()
{
 int n, contador, fatorial;
 printf("Digite um numero para calcular o fatorial: ");
 scanf("%d", &n);
 fatorial = 1;
 contador = 1;
 while (contador <= n) {
 fatorial = fatorial * contador;
 contador=contador+1;
 }
 return 0;
}
```

Repetição com variáveis de controle:

```
for(inicialização; condição final; incremento)
{
comandos;
}
```

```
#include <stdio.h>
#int main ()
{
 int n, contador, fatorial;
 printf("Digite um numero para calcular o fatorial: ");
 scanf("%d", &n);
 fatorial = 1;
 for (contador = 1; contador <= n; contador++)
 fatorial = fatorial * contador;
 return 0;
}
```

Faça as modificações pertinentes nos programas dados como sugestão, compile e entregue ao seu professor. Boa sorte e ótimos estudos!

## Avançando na prática

### Opção de Menu

#### Descrição da situação-problema

A empresa Do Whi realiza vários cálculos e muitos deles são utilizados com frequência pelos seus funcionários, onde os que mais se destacam são o cálculo da força, da área, perímetro e raio de um círculo. Você já trabalha a algum tempo na empresa e sugeriu para os gerentes a criação de um programa em linguagem C, onde fizesse a utilização de um menu para facilitar o acesso aos cálculos.

#### Resolução da situação-problema

Para solução deste problema, você poderá usar a programação de repetição com teste no final:

```
do
{
 comandos;
}
while (condição);
```

```
#include <stdio.h>
#include <stdlib.h>
```

```

main()
{
 char material[60];
 float n, m, a;
 float raio, area, perimetro, pi;
 int opcao;
 do
 {
 printf("\t\t\n OPCOES DE CALCULOS \n");
 printf("\n 0. SAIR DO MENU \n");
 printf("\n 1. CALCULAR A FORCA \n");
 printf("\n 2. CALCULAR RAO, AREA E PERIMETRO DO
 CIRCULO \n");
 printf("\n 3. RETORNAR AO MENU \n");
 printf("\n Opcao: ");
 scanf("%d", &opcao);
 switch(opcao)
 {
 case 0:
 printf("SAIR...\n");
 break;
 case 1:
 printf ("\n Digite a massa do objeto: ");
 scanf("%f", &m);
 printf ("\n Digite a aceleracao: ");
 scanf("%f", &a);
 n=(m*a);
 printf("\n O calculo da forca e: %.2f \n", n);
 break;
 case 2:
 printf("Digite o raio: ");
 scanf("%f", &raio);
 pi = 3.141592;
 area = pi*(raio * raio);

```

```

 perimetro = 2.0 * pi * raio;
 printf(" \n Raio: %.2f \n", raio);
 printf(" \n Area: %.2f \n", area);
 printf(" \n Perimetro: %.2f \n", perimetro);
 break;
 case 3: system("cls");
 break;
 default:
 printf("OPÇÃO INVALIDA \n");
 }
} while(opcao);
return 0;
}

```

Fique à vontade para colocar novas rotinas que contenham outros cálculos.

Pratique muito e sucesso!

## Faça valer a pena

**1.** Analise o programa abaixo, que realiza a soma dos números positivos usando repetição com teste no final e observe a parte que está faltando.

```

#include<stdio.h>
int main()
{
 int n;
 int soma = 0;
 do
 {
 printf("Digite um número positivo para ser somado ou negativo para sair:
");
 scanf("%d", &n);
 .
 .
 .
 printf("A soma eh %d\n", soma);
 return 0;
 }
}

```

Assinale a alternativa que corresponde à parte que melhor se identifica com o código do programa.

a) 

```
if(n >= 0)
soma = soma + n;
}
while(n >= 0);
```

b) 

```
if(n >= 0)
soma = soma + n;
while(n >= 0);
}
```

c) 

```
if(n <= 0)
soma = soma + n;
}
while(n >= 0);
```

d) 

```
if(n >= 0)
soma = soma + n;
}
while(n <= 0);
```

e) 

```
if(n >= 0)
soma = ++ n;
while(n >= 0);
}
```

**2.** Segundo a programação com teste no início, Soffner (2013, p. 64) coloca que um programa “não executará nenhuma repetição (e as ações que ali dentro estiverem programadas) sem antes testar uma condição”. Para realizar a repetição com teste no início, você usará o comando iterativo “while”, que significa em português “enquanto”.

Levando em consideração que precisamos estar atentos para que não ocorra um loop infinito, analise as afirmações abaixo e responda a alternativa correta:

**I. Contador** – é utilizada para controlar as repetições, quando esta é determinada.

**II. Incremento e decremento** – trabalha o número do contador, somente quando for positivo.

**III. Acumulador** – que segundo Soffner (2013) irá somar as entradas de dados de cada iteração da repetição, gerando um somatório a ser utilizado quando da saída da repetição.

**IV. Condição de parada** – utilizada para determinar o momento de parar quando não se tem um valor exato desta repetição.

- a) Somente a afirmação I está correta;
- b) As afirmações I, III e IV estão corretas;
- c) As afirmações II e III estão corretas;
- d) Somente a afirmação IV está correta;
- e) As afirmações I, II, III e IV estão corretas.



**3.** Analise o código do programa abaixo, onde foi utilizada a estrutura de repetição com variável de controle

```
#include <stdio.h>
main()
{
 int contador; //variável de controle do loop
 for(contador = 1; contador <= 10; contador++)
 {
 printf("%d ", contador);
 }
 return(0);
}
```

Analizando o programa acima, qual a leitura podemos fazer da linha:

For (contador = 1; contador <= 10; contador++), onde a primeira expressão é: contador = 1, a segunda expressão é: contador <= 10 e a terceira expressão é: contador++

- a) Na primeira expressão, "contador" tem o seu valor iniciado em "1".  
Na segunda expressão, "contador" está condicionada a ser igual a "10".  
Na terceira expressão, "contador" será realizado o incrementado de 1 para ao seu valor.
- b) Na primeira expressão, "contador" tem o seu valor iniciado em "1".  
Na segunda expressão, "contador" está condicionada a ser menor a "10".  
Na terceira expressão, "contador" será realizado o incrementado de 1 para ao seu valor.
- c) Na primeira expressão, "contador" tem o seu valor iniciado em "1".  
Na segunda expressão, "contador" está condicionada a ser maior ou igual a "10".  
Na terceira expressão, "contador" será realizado o decremento de 1 para ao seu valor.
- d) Na primeira expressão, "contador" tem o seu valor iniciado em "0".  
Na segunda expressão, "contador" está condicionada a ser menor ou igual a "10".  
Na terceira expressão, "contador" será realizado o incrementado de 2 para ao seu valor.
- e) Na primeira expressão, "contador" tem o seu valor iniciado em "1".  
Na segunda expressão, "contador" está condicionada a ser menor ou igual a "10".  
Na terceira expressão, "contador" será realizado o incrementado de 1 para ao seu valor.

# Referências

ESPERIDIÃO, Hélio. 02 - Exercício - Estruturas de repetição em C. 2017. Youtube. 10 mar. 2017. Disponível em: <<https://www.youtube.com/watch?v=PNeYeq4-Tt0>> Acesso em: 27 jan. 2018.

DAMAS, Luís. **Linguagem C**. Tradução: João Araújo Ribeiro; Orlando Bernardo Filho. 10 ed. [Reimpr.]. Rio de Janeiro: LTC, 2016.

MANZANO, José Augusto Navarro Garcia. **Estudo Dirigido de Linguagem C**. 17 ed. rev. São Paulo: Érica, 2013.

\_\_\_\_\_. **Linguagem C**: acompanhada de uma xícara de café. São Paulo: Érica, 2015.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. 2 ed. São Paulo: Pearson Prentice Hall, 2008.

SCHILD, Herbert. **C: completo e total**. Tradução: Roberto Carlos Mayer. 3 ed. São Paulo: Pearson Prentice Hall, 2005.

\_\_\_\_\_. **C: completo e total**. Tradução: Roberto Carlos Mayer. 3 ed. São Paulo: Pearson; Informática edition, 1997.

SOFFNER, Renato. **Algoritmos e Programação em Linguagem C**. 1 ed. São Paulo: Saraiva, 2013.

# Aplicações de programação

## Convite ao estudo

Caro estudante, chegamos à última unidade no estudo dos algoritmos e lógica de programação. Ao longo do livro você teve oportunidade de conhecer diversos elementos e uma série de técnicas que são utilizadas para criar soluções computacionais para os mais diversos problemas. Na primeira unidade você viu o que é a lógica de programação, como desenvolver o raciocínio lógico e as formas de representação de um algoritmo. Na segunda unidade você aumentou seu repertório aprendendo quatro grandes elementos de algoritmos: as estruturas de decisão, as estruturas de repetição, as variáveis compostas e as sub-rotinas. Na terceira unidade, você foi apresentado à linguagem de programação "C", uma das mais importantes na história da computação.

Nessa unidade você terá a oportunidade de continuar explorando e aprimorar a programação na linguagem "C" através de novos desafios. Dando continuidade ao seu trabalho na Kro Engenharias, chegou o momento de você implementar na linguagem "C" um dos algoritmos que você propôs para seu chefe. Sua primeira missão nessa nova unidade será implementar o programa que faz a conversão da medida da quilometragem percorrida por um carro em um determinado dia. Na segunda missão você deverá entregar um programa que faça o controle da temperatura dos braços mecânicos da linha de soldagem de peças automotivas. Para finalizar seu trabalho na Kro Engenharias, você enfrentará um novo desafio, e deverá implementar uma solução numérica para o cálculo de raízes quadradas.

Para cumprir sua jornada, você aprenderá nessa unidade como trabalhar com os vetores e matrizes na linguagem "C", vendo os detalhes da sintaxe, atribuição de valores, iteração por seus elementos. Usaremos muitas sub-rotinas para que você se acostume com essa organização, e veremos como funciona uma sub-rotina recursiva.

Bons estudos!

# Seção 4.1

## Programação e funções com vetores

### Diálogo aberto

Caro estudante, entramos na reta final da disciplina de algoritmos e lógica de programação, e essa primeira seção é destinada a lhe aproximar ainda mais da linguagem C consolidando várias técnicas que você viu durante o livro. Iniciamos com o estudo dos vetores na linguagem C, tal estrutura de dados é largamente utilizada para o armazenamento interno em um programa computacional. Imagine que você precise armazenar o salário de 1000 funcionários, ou então que você desenvolva um sistema para monitorar a pressão em uma caldeira a cada hora do dia e esses dados precisam ser armazenados para relatório.

Como você já viu, usar 1000 variáveis é algo inconcebível, já que podemos realizar essa tarefa usando apenas uma variável composta.

Após o estudo dessa seção você conseguirá implementar um dos algoritmos criados para a Kro Engenharias. Lembra do sistema que controlava o gasto de combustível diário dos quinze automóveis da empresa disponibilizados para seus funcionários? Portanto, você deverá implementar um programa que armazena o combustível gasto por cada um dos quinze automóveis durante um dia. Lembrando que o formulário da Kro Engenharias permite a entrada nas unidades quilômetro, metro ou milha, e você deverá garantir que o dado que será guardado estará na unidade quilômetro. Apresente o programa funcionando para seu gerente e entregue o código fonte para que ele possa analisar com tranquilidade.

Para completar essa primeira missão você verá nessa seção a sintaxe de vetores na linguagem C, como é feita a atribuição de valores nos vetores e como é feita a leitura dos elementos nos vetores. Você também aprenderá como criar e utilizar sub-rotina em C.

Preparado? Bons estudos!

### Definição, características, sintaxe e atribuição de valores em vetores

Em programação, vetor (*array*) é um tipo especial de variável capaz de armazenar diversos valores “ao mesmo tempo” usando um mesmo endereço na memória. Por armazenar diversos valores também é chamado de variável composta, ou ainda estrutura de dado matricial (MANZANO, 2010).

Existem algumas características a respeito dessa variável que devem ficar claras:

- A capacidade do vetor, ou seja, quantos elementos ele consegue guardar, deve ser determinada no momento em que se declara essa variável especial.
- O vetor é uma variável composta unidimensional, ou seja, o acesso a seus dados tanto para leitura como para escrita precisa apenas de um índice.
- Um vetor pode ser homogêneo, ou seja, armazenar apenas um tipo de valor (real, inteiro, caracter etc.), ou pode ser heterogêneo e armazenar valores de diferentes tipos. Caso ele seja heterogêneo o nome *struct* (estrutura) é mais adequado para identificar esse tipo.
- Independentemente de ele ser homogêneo ou heterogêneo, a velocidade do acesso aos seus dados é a mesma. Isso acontece porque os dados são alocados sequencialmente na memória, a fim de otimizar o desempenho do sistema.

Todas essas características ficarão mais claras no decorrer do texto, vamos ver agora como funciona na prática o uso de vetores na linguagem “C”.

Quando estamos programando em C, alguns tipos de dados primitivos são usados com mais frequência: *int* (inteiro), *float* (real de precisão simples), *double* (real de precisão dupla) e *char* (caracteres). Vamos criar vetores usando esses tipos primitivos mais comumente utilizados.

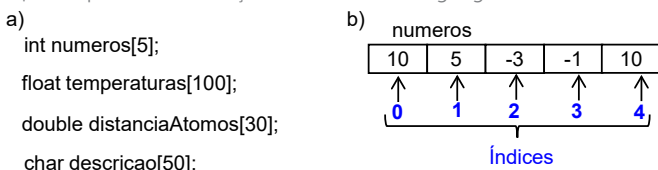
Sintaxe de vetores homogêneos na linguagem C: Para criar um vetor homogêneo basta escrever o tipo da variável seguido do nome do vetor seguido de colchetes e a quantidade de

elementos. Devemos ainda colocar um ponto e vírgula ao final da linha de comando.

Sintaxe: `tipo variavel[N];`  *N* é a quantidade de colunas (ou linhas)

Confira alguns exemplos de declaração de vetores na Figura 4.1 – a, e uma ilustração de um dos vetores com seus respectivos índices:

Figura 4.1 | Exemplos de declaração de vetores na linguagem C



Fonte: elaborada pelo autor.

Por meio dos exemplos é possível observar as 3 primeiras características dos vetores. Veja que independentemente do tipo de variável primitiva usada, em todos os casos a capacidade do vetor já foi informada. Para todos os casos usamos apenas um índice, pois um vetor é uma variável composta unidimensional (alguns autores costumam chamar o vetor de matriz unidimensional) (MANZANO, 2015). Todos os vetores usados no exemplo 1 são homogêneos, portanto, no vetor “numeros” só será possível armazenar valores inteiros, já no vetor “temperaturas” será possível armazenar valores de  $3,4 \cdot 10^{-38}$  até  $3,4 \cdot 10^{38}$ , no vetor “distanciaAtomos” será possível armazenar valores de  $1,7 \cdot 10^{-308}$  até  $3,4 \cdot 10^{308}$ , e por fim no vetor “descricao” será possível armazenar até 50 caracteres (mais adiante falaremos melhor sobre isso, pois na verdade temos que deixar um espaço para um caractere especial quando usamos o tipo *char*). Veja que o tipo da variável primitiva é o que determina **quais** valores poderão ser inseridos nas estruturas de dados, e o tamanho determina **quantos** valores poderão ser inseridos.



Refleta

Agora você já sabe que para utilizar um vetor é preciso no momento da declaração informar seu tamanho. Qual consequência poderia trazer se declarássemos um vetor com uma capacidade muito pequena para uma determinada aplicação? Ou se declarássemos ele com uma capacidade muito grande? Quais seriam as vantagens e desvantagens em cada caso?

## Atribuição de valores a vetores homogêneos

Para armazenarmos um valor dentro de um vetor usamos seu índice. Você já viu que a atribuição de valores em C é feita com o sinal de igualdade "=", portanto basta informar o nome do vetor e entre colchetes o índice em que você deseja guardar um valor e usando a atribuição escolher o valor.

`variavel[N] = valor;`  N é o índice no qual deseja guardar

Exemplos:

```
numeros[0] = -100;
temperatura[10] = 37.5;
distanciaAtomos[5] = 0.02345;
```

Alguns pontos sobre a atribuição de valores têm que ser fixados:

- Em qualquer vetor o primeiro índice é sempre 0, portanto se declaramos que o vetor tem capacidade para 5 elementos, seu índice irá variar de 0 a 4 (de  $n$  até  $n - 1$ ).
- Não é obrigatório que todas as posições sejam ocupadas, você pode declarar um vetor com 10 posições e usar somente 5.
- A atribuição em vetores do tipo char é feita de maneira diferente, usando a função `strcpy(destino, "texto")`, disponível na biblioteca `<string.h>`. Exemplo: `strcpy(produto, "Processador i7")`, guarda no vetor `produto` o texto `Processador i7`.

Você pode estar se perguntando nesse momento: "e se eu quiser guardar um valor digitado pelo usuário no vetor?" Vejamos como é feito no programa da Figura 4.2.

Figura 4.2 | Programa para armazenar valores em vetores

```
1. #include <stdio.h>
2. void main(){
3. int numeros[5];
4. printf("Digite um numero: ");
5. scanf("%d",&numeros[0]);
6. printf("Voce digitou o valor: %d",numeros[0]*2);
7. getchar();
8. }
```

Fonte: elaborada pelo autor.

Na Figura 4.2 na linha 3 o vetor "numeros" é declarado com capacidade para armazenar 5 valores inteiros.



Na linha 4 a função `printf()` é utilizada para exibir uma mensagem para o usuário, e na linha 5 a função `scanf()` é utilizada para ler o que o usuário digitou e guardar no vetor. O comando dessa linha merece uma explicação a mais, veja que é utilizado o marcador `%d`, isso “avisa” o compilador que queremos guardar um número inteiro, depois é utilizada a sequência: `&numeros[0]`. Nessa sequência, o caractere `&` traz dentro dele o endereço da variável “numeros” e o zero dentro do colchete é a posição em que será armazenado o valor digitado. Na linha 6 exibimos para o usuário o dobro do valor que ele mesmo digitou, veja que estamos multiplicando por 2 o valor que está guardado no vetor “numeros” índice 0. Na linha 7 usamos um comando para fazer uma parada no sistema, sem esse comando a janela de execução abre e fecha e não conseguimos ver o programa funcionando.

## O vetor de caracteres

Vamos agora entender um pouco melhor o vetor de caracteres. O tipo `char` é primitivo em qualquer linguagem e uma variável desse tipo tem capacidade para armazenar um único caractere. Porém, o mais comum é armazenarmos palavras ou frases, estes nada mais são do que um conjunto de caracteres. Portanto, para representar tal conjunto precisamos de uma variável que armazene uma cadeia de caracteres (MANZANO, 2015), ou seja, um vetor de caracteres. Esse tipo de dado é chamado de *string*, e não é considerado um tipo primitivo.

Para guardar valores que o usuário digitou é preciso usar a função `scanf("%s",&destino)` ou `fgets(destino,tamanho,fluxo)`, ambas pertencentes à biblioteca `<stdio.h>`. Para entender quando usar um ou outro vamos criar um programa para armazenar um equipamento e o departamento a que ele pertence (Figura 4.3).

Figura 4.3 | Armazenamento de *strings*

```
1. #include <stdio.h>
2. void main(){
3. char equipamento[20];
4. char departamento[20];
5. printf("Digite o equipamento: ");
6. scanf("%s",equipamento);
7. printf("Digite o departamento: ");
8. fflush(stdin);
9. fgets(departamento,20,stdin);
10. getchar();
11. }
```

Fonte: elaborada pelo autor.

Na Figura 4.3 criamos dois vetores de caracteres nas linhas 3 e 4. Na linha 6 utilizamos a função `scanf()` para guardar o equipamento digitado pelo usuário, veja que usamos o código `%s` para “informar” a função que vamos guardar uma *string*. Já na linha 9 usamos a função `fgets()` para guardar o departamento, que possui um tamanho máximo de 20 caracteres e seu fluxo de dados está vindo do `“stdin”` ou seja, da entrada padrão que é o teclado. Temos dois pontos importantes a analisar aqui.

- Diferença entre as funções: a função `scanf(“%s”)` armazena apenas palavras simples, ou seja, ela interrompe o armazenamento quando encontra um “espaço em branco”. Portanto, se caso você tentar guardar um equipamento com nome composto, por exemplo, “Braço mecânico”, essa função não é adequada. Já a função `fgets()` tem outro comportamento, ela armazena qualquer caractere, inclusive espaços em branco, até que seu tamanho supere.
- Características da função `fgets()`: o último parâmetro dessa função se refere ao fluxo de origem da *string*, de onde está vindo, e pode ser do teclado (`stdin`), de arquivo de texto, ou até de banco de dados. Para que o comando funcione corretamente adicionamos a função `fflush(stdin)` antes de capturar a entrada, pois essa função limpa qualquer “lixo” que esteja na memória (*buffer*) do fluxo de entrada. Por fim, “é importante levar em conta que uma sequência de caracteres é considerada terminada quando nela há o código de finalização definido por meio do símbolo `“\0”`, que é colocado ao ser acionada a tecla `<Enter>`.” (MANZANO, 2015, p. 394). Esse código é colocado pelo compilador e não por nós, e é usado pelo comando para “saber” que a *string* acabou. Como consequência perdemos uma posição no vetor, portanto considerando nosso exemplo da Figura 4.3, o usuário poderá digitar uma frase de apenas 19 caracteres.



**Pesquise mais**

Os vetores de caracteres, *strings*, são amplamente utilizados e suas possibilidades vão muito além do que foi apresentado aqui. Aprenda mais sobre essas estruturas e sobre as funções da biblioteca `<string.h>` acessando os endereços:

<[https://www.youtube.com/watch?v=5mJZh\\_ikDaQ&list=PL8iN9FQ7\\_jt4DJbeQqv--jpTy-2gTA3Cp&index=31](https://www.youtube.com/watch?v=5mJZh_ikDaQ&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=31)>. Acesso em: 16 dez. 2017.

<<http://linguagemc.com.br/a-biblioteca-string-h>>. Acesso em: 16 dez. 2017.

## Sintaxe de vetores heterogêneos na linguagem C

Um recurso muito utilizado em programação é armazenar um conjunto de dados dentro de uma mesma variável, formando assim uma estrutura heterogênea ou, como é conhecido na linguagem C, **struct** (DEITEL; DEITEL, 2011). É importante ressaltar que uma *struct* funciona como um “tipo de dado”, e para que seja possível utilizá-la é necessário atribuir uma (ou mais) variáveis à estrutura criada. Vejamos como declarar uma *struct* na Figura 4.4.

Figura 4.4 | Declaração de vetores heterogêneos (*struct*)

|                                                                                                                                  |                 |                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Sintaxe:</b>                                                                                                                  | <b>Exemplo:</b> | <pre>1. struct ficha_manutencao { 2.     int codigo; 3.     char equipamento[20]; 4.     char departamento[20]; 5.     char descricao[100]; 6.     float valor; 7. }; 8. struct ficha_manutencao ordemServico;</pre> |
| <pre>struct &lt;identificador&gt; { &lt;listagem dos tipos e membros&gt;; } struct &lt;identificador&gt; &lt;variavel&gt;;</pre> |                 |                                                                                                                                                                                                                      |

Fonte: elaborada pelo autor.

Na Figura 4.4, na linha 1 é usado o comando *struct* para iniciar o bloco que cria a variável composta seguido de um nome para identificar tal estrutura, no nosso caso o nome escolhido foi “ficha\_manutencao” (lembrando que não pode ter espaço e nem acento). Das linhas 2 a 6 é feita a declaração dos campos que farão parte dessa estrutura. O bloco deve estar entre chaves e a chave que fecha o bloco tem que vir acompanhada por ponto e vírgula (linha 7). Na linha 8 temos a variável “ordemServico” sendo declarada com seu tipo “struct ficha\_manutencao”, pois é através dessa variável que acessaremos os campos para leitura e escrita.

## Atribuição de valores a vetores heterogêneos

Para atribuímos valores aos campos da *struct*, devemos usar o nome da variável que recebeu como tipo a estrutura com um ponto (.) e o nome do campo: variavel.campo. Veja um exemplo na Figura 4.5.

Figura 4.5 | Atribuição de valores a vetores heterogêneos

```
1. #include <stdio.h>
2. void main(){
3. struct ficha_manutencao {
4. int codigo;
5. char equipamento[20];
6. float valor;
7. };
8. struct ficha_manutencao ordemServico;
9. printf("Digite o código da ordem de serviço: \n");
10. scanf("%d",&ordemServico.codigo);
11. printf("Digite o equipamento: \n");
12. fflush(stdin);
13. fgets(ordemServico.equipamento,20,stdin);
14. printf("Digite o valor: \n");
15. scanf("%f",&ordemServico.valor);
16. getchar();
17. }
```

Fonte: elaborada pelo autor.

Na Figura 4.5, das linhas 3 a 7 a estrutura é definida, e na linha 8 é atribuída uma variável à estrutura. Na linha 9 é informado ao usuário para que digite um valor e na linha 10 o valor digitado é guardado dentro no campo "codigo" da variável "ordemServico" que é do tipo *struct*. Para guardar o equipamento é mais adequado usar a função *fgets()* (linha 13), pois os equipamentos podem ter nomes compostos, por exemplo, "tritador de para-brisa". Mas sabemos que antes de usar o *fgets()* devemos limpar o buffer de entrada com a função *fflush()* (linha 12). E na linha 15 é guardado o valor dentro do seu respectivo campo na variável composta.



### Assimile

Quando utilizamos campos que não são do tipo char dentro da *struct*, podemos atribuir valores direto sem a função *scanf()* da seguinte forma:

```
ordemServico.codigo = 12345;
ordemServico.valor = 2130;
```

Para os campos de caracteres essa atribuição não é permitida na linguagem C. Se quisermos fazer essa atribuição direta sem uso do *scanf()*, devemos utilizar a função *strcpy(destino, "texto")* que pertence à biblioteca `<string.h>`. Então usando essa função podemos atribuir caracteres da seguinte forma:

```
strcpy(ordemServico.maquina, "Braço mecânico");
strcpy(ordemServico.departamento, "Produção");
```

## Iteração sobre os elementos de vetores

Não faz sentido criar um vetor e guardar um único valor. A grande utilidade dos vetores é guardar vários valores a fim de otimizar o armazenamento e acesso aos dados. Para que os vetores cumpram sua tarefa é necessário utilizá-los juntamente com as estruturas de repetição, principalmente a estrutura com variável de controle.

A variável de controle da estrutura de repetição será usada como o índice para a leitura/armazenamento de dados no vetor, portanto a sintaxe será:

```
for(i=0; i<10; i++){
 numeros[i];
}
```

Veja que a variável "i" está sendo usada tanto para controlar a estrutura de repetição como também índice do vetor. Olhando para esse código podemos concluir que o vetor "numeros" irá armazenar 10 valores, que ocuparão os índices de 0 a 9, lembrando que  $i < 10$  é diferente de  $i \leq 10$ .

## Problemas utilizando função com vetores

Sub-rotinas, ou como são mais comumente chamadas, funções, são utilizadas para organizar o programa e evitar a repetição de comandos. Estamos usando várias funções que já estão prontas dentro das bibliotecas, porém essas funções não atendem todas as necessidades, afinal cada problema demanda uma solução e se não existe a solução pronta, então temos que criar nossas próprias funções.



## Pesquise mais

Funções são extremamente úteis na estruturação de um programa. Seu uso permite organizar e reutilizar códigos. Leia o capítulo 10 do livro: MANZANO, José Augusto N. G. *Linguagem C: acompanhada de uma xícara de café*. São Paulo: Érica, 2015. 480 p., e aprenda mais sobre essa poderosa ferramenta.

A sintaxe da declaração de uma função pode ser feita antes ou depois da função da principal (*main*) e é feita da seguinte forma:

```
<tipo> nome(<tipo de parâmetros> <parâmetros>) {
 <Variáveis locais>;
 <Comandos>;
 <return> <valor>;
}
```

**<tipo>** tipo de dado que a sub-rotina retorna. Uma função pode retornar qualquer valor (inteiro, ponto flutuante etc.), inclusive nenhum valor, nesse caso será usada a palavra reservada *void*.

**<nome>** nome da função. Não pode ter espaço nem acento.

**<tipo de parâmetros>** quando existir parâmetros é preciso definir seu tipo.

**<parâmetros>** é opcional passar um valor para a função.

**<Variáveis locais>** são variáveis que somente podem ser usadas dentro da função (não existem fora dela).

**<Comandos>** o bloco de comandos da função.

**<return>** usado somente quando a função não retornar *void*.

**<valor>** o valor que a função retornará quando não for *void*, tem que ser do mesmo tipo que o **<tipo>** da função.



## Exemplificando

Vamos utilizar uma estrutura de repetição para preencher um vetor inteiro que armazena valores de temperatura medido por um sensor durante o dia. O programa deverá armazenar 10 valores e depois

retornar qual foi a média da temperatura no dia. O cálculo da média será feito em uma sub-rotina e será "chamado" dentro da função principal. Na Figura 4.6 temos a solução desse problema.

Figura 4.6 | Programa para calcular média de temperatura

```
1. #include <stdio.h>
2. void CalcularMedia(float temperatura){
3. float media;
4. media = temperatura / 10;
5. printf("A média diária da temperatura foi: %.2f",media);
6. }
7. void main(){
8. float temperaturas[10];
9. float total;
10. int i;
11. for(i=0;i<3;i++){
12. printf("Informe a temperatura %d: ",i);
13. scanf("%f",&temperaturas[i]);
14. total = total + temperaturas[i];
15. }
16. CalcularMedia(total);
17. getchar();
18. }
```

Fonte: elaborada pelo autor.

Na Figura 4.6, logo após a inserção da biblioteca `<stdio.h>` e declarada a função `void CalcularMedia(float temperatura)`, podemos dividir esse comando em três partes:

- O primeiro item, que no nosso caso é `void`, é o tipo de dado que a sub-rotina dará retorno. Portanto, nossa função não retornará nada (`void`), ela fará seu trabalho e fim, não trará nada para a função principal (`main`).
- O segundo item é o próprio nome da sub-rotina `CalcularMedia`.
- O terceiro item é o que está entre parênteses após o nome da função. Dentro dos parênteses especificamos as variáveis que a função receberá quando for chamada. No nosso caso, quando chamarmos a função `CalcularMedia()` deverá ser colocado dentro do parênteses uma variável do tipo `float`.

Feita a declaração da sub-rotina, dentro do bloco devem ir os comandos pertinentes ao que queremos que seja realizado. Para isso a variável "media" é declarada na linha 3, na linha 4 é feito o cálculo da média, e na linha 5 é impresso na tela o resultado. Na Linha 6 a chave marca o final da função `CalcularMedia()`. Agora é hora de construir a função principal (main). Nas linhas 8 a 10 são declaradas as variáveis. Na linha 11 é criada a estrutura de repetição com a variável de controle "i" indo de 0 a 9 (veja que é somente menor que 10 e não menor-igual), com o passo de incremento 1 (i++). Na linha 12 é impressa uma mensagem pedindo para digitar a temperatura, apenas para fins de aparência foi adicionada a variável i nessa linha. Na linha 13 a função `scanf()` armazena um valor do tipo *float* no vetor `temperaturas` na posição i (`temperaturas[i]`). Na linha 14 a variável "total" vai acumulando as temperaturas informadas. Após o término das 10 repetições na linha 11, a função `CalcularMedia` é chamada na linha 12 e é passado como parâmetro o valor da temperatura acumulada através da variável "total". A partir desse ponto quem entra em execução é a sub-rotina, ela faz seu procedimento e como seu tipo de retorno é void, quando termina o programa está completo.

Caso queira fazer a impressão das 10 temperaturas guardadas basta colocar a função `printf()` dentro de uma nova estrutura de repetição, após a linha 15, da seguinte forma:

```
printf("Temperatura %d : %f", i, temperaturas[i]);
```

Chegamos ao final dessa seção que foi dedicada a explorar a utilização de vetores na linguagem C. Aproveite o conhecimento apresentado e aplique a novas situações, pois somente através da prática você dominará as técnicas e linguagens de programação.

## Sem medo de errar

Chegou o momento de você implementar o programa que armazena o combustível gasto por cada um dos quinze automóveis da Kro Engenharias durante um dia. Lembrando que o formulário da Kro Engenharias (Figura 4.7) permite a entrada nas unidades: quilômetro, metro ou milha, e você deve garantir que o dado que será guardado estará na unidade quilômetro. A conversão da unidade deve ser feita em uma sub-rotina.



Figura 4.7 | Captura de tela do formulário da Kro Engenharia

| Controle de combustível |                                  |           |                                  |           |                                  |                                                                                                                                                        |
|-------------------------|----------------------------------|-----------|----------------------------------|-----------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Carro 1:                | <input type="text" value="123"/> | Carro 6:  | <input type="text" value="90"/>  | Carro 11: | <input type="text" value="134"/> | Escolha a unidade de medida:<br><input type="text" value="Quilômetros"/><br><input type="text" value="Metros"/><br><input type="text" value="Milhas"/> |
| Carro 2:                | <input type="text" value="432"/> | Carro 7:  | <input type="text" value="0"/>   | Carro 12: | <input type="text" value="245"/> |                                                                                                                                                        |
| Carro 3:                | <input type="text" value="222"/> | Carro 8:  | <input type="text" value="120"/> | Carro 13: | <input type="text" value="234"/> |                                                                                                                                                        |
| Carro 4:                | <input type="text" value="415"/> | Carro 9:  | <input type="text" value="110"/> | Carro 14: | <input type="text" value="199"/> |                                                                                                                                                        |
| Carro 5:                | <input type="text" value="360"/> | Carro 10: | <input type="text" value="0"/>   | Carro 15: | <input type="text" value="0"/>   |                                                                                                                                                        |
|                         |                                  |           |                                  |           |                                  | <input type="button" value="Calcular"/>                                                                                                                |

Fonte: elaborada pelo autor.

Para criar esse programa precisamos das bibliotecas `<stdio.h>` e `<string.h>` (linhas 1 e 2). Das linhas 3 a 11 está a função que verifica a unidade escolhida e faz a conversão. Na linha 3 a função é declarada, com as seguintes características:

- tipo de retorno: *float*
- nome: TestarUnidade
- 2 parâmetros: 1 vetor de caracteres – variável “unidade” e um *float* – “totalConvertido”. Portanto, quando essa função for chamada terão que ser passados dois parâmetros, senão o programa não executa.

Dentro da sub-rotina é usada a função `strcmp()` da biblioteca `<string.h>` que compara o valor de duas strings, e quando elas são iguais retorna zero. Por isso as condições das linhas 4, 6 e 8 comparam o resultado da função com zero (`if(strcmp(unidade, "km") == 0)`). Caso o usuário tenha digitado uma unidade inválida, será retornado -1. A função principal inicia-se na linha 12, seguida da criação de todas as variáveis necessárias ao programa (linhas 13 a 16). Da linha 17 até 21 é feito o laço que lê e armazena o valor dos 15 carros, acumulando a cada iteração na variável total. Na linha 23 é lida a unidade que o usuário escolheu e guardada na variável “unidadeUsada”. E na linha 24 a sub-rotina TestarUnidade é chamada e seu resultado é guardado dentro de uma variável chamada “totalConvertidoKM”, lembrando que o resultado da sub-rotina será um valor do tipo *float*. E por fim, é impresso na tela o valor acumulado e na unidade quilômetro.

Figura 4.8 | Programa para calcular quilometragem

```
1. #include <stdio.h>
2. #include <string.h>
3. float TestarUnidade(char unidade[20],float totalConvertido){
4. if(strcmp(unidade,"km") == 0){
5. return totalConvertido;
6. }else if(strcmp(unidade,"m") == 0){
7. return totalConvertido/1000;
8. }else if(strcmp(unidade,"milha") == 0){
9. return totalConvertido/0.621371;
10. }else return -1;
11. }
12. void main(){
13. float distancia[15];
14. float char unidadeUsada[20];
15. float total, totalConvertidoKM;
16. int i;
17. for(i=0;i<15;i++){
18. printf("Digite o valor percorrido pelo carro %d: ",i);
19. scanf("%f",&distancia[i]);
20. total = total + distancia[i];
21. }
22. printf("Digite a unidade utilizada (km, m ou milha): ");
23. scanf("%s",&unidadeUsada);
24. totalConvertidoKM = TestarUnidade(unidadeUsada,total);
25. printf("A quantidade total percorrida em quilômetros foi: %f",totalConvertidoKM);
26. getchar();
27. }
```

Fonte: elaborada pelo autor.

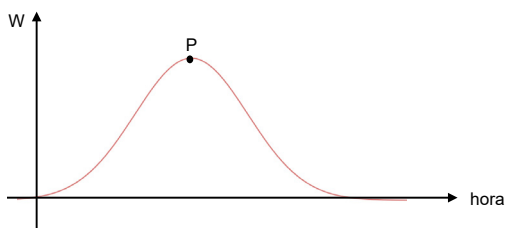
É importante que você entenda todos os passos utilizados. Para isso, altere o nome das variáveis, da sub-rotina, dos parâmetros e implemente o programa fazendo as alterações necessárias para que o formulário também aceite a entrada da unidade jarda. Outra modificação interessante seria adaptar o programa para que a unidade possa ser digitada tanto com letras maiúsculas como minúsculas.

### Procurar o maior valor em um vetor

#### Descrição da situação-problema

O estudo da física permite entender o comportamento de vários sistemas, sendo a potência uma das grandezas explicada por essa área do conhecimento. Essa grandeza é representada no sistema internacional pela unidade Watt (W), e tem como objetivo determinar a quantidade de energia concedida por uma fonte a cada unidade de tempo. Na Figura 4.9 temos o gráfico de um sistema que monitora o funcionamento de uma máquina durante o dia. Em uma determinada hora do dia a máquina atinge sua potência máxima, indicada pelo ponto P na figura. Implemente um programa em C, que leia e armazene 20 valores e retorne qual foi o valor máximo atingido naquele determinado dia.

Figura 4.9 | Comportamento da potência de uma máquina



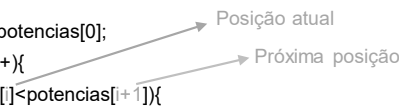
Fonte: elaborada pelo autor.

#### Resolução da situação-problema

Para procurarmos o maior valor em um vetor temos que comparar o valor do índice atual com o valor do próximo índice e guardar o maior valor em uma variável. Veja o programa na Figura 4.10. Das linhas 6 a 9 o programa lê e armazena os 20 valores no vetor. Na linha 10 foi atribuída à variável "maiorValor" o valor armazenado na primeira posição do vetor (Esse passo é importante caso os valores sejam todos iguais). Na linha 11 inicia-se uma nova estrutura de repetição para encontrar o maior valor, na linha 12 existe uma estrutura condicional que compara o valor da posição atual (i) com o valor da próxima posição (i+1) e armazena o maior valor na variável "maiorValor". Após terem sido feitas todas as comparações na linha 16 é impresso o maior valor encontrado.

Figura 4.10 | Programa para retornar maior valor em um vetor

```
1. #include <stdio.h>
2. void main(){
3. float potencias[20];
4. float maiorValor;
5. int i;
6. for(i=0;i<20;i++){
7. printf("Digite o valor da potência 1: ");
8. scanf("%f",&potencias[i]);
9. }
10. maiorValor = potencias[0];
11. for(i=0;i<20;i++){
12. if(potencias[i]<potencias[i+1]){
13. maiorValor = potencias[i+1];
14. }
15. }
16. printf("A maior potência atingida pela máquina foi: %f",maiorValor);
17. getch();
18. }
```



Fonte: elaborada pelo autor.

Altere o programa para que ele também armazene o menor valor.

## Faça valer a pena

**1.** Vetor é uma das estruturas que existe para o armazenamento de dados. Sua utilização é vasta devido à facilidade em ler e escrever dados e à velocidade com que as operações são realizadas, pois cada valor no vetor é armazenado sequencialmente na memória (MANZANO, 2015).

Considere as afirmações sobre os vetores e escolha a opção correta.

I – O vetor é uma estrutura de dados dinâmica, ou seja, seu tamanho pode ser redimensionado em tempo de execução.

II – Como o vetor é uma estrutura unidimensional, para acessar seus dados é preciso somente um índice.

III – Uma das características dos vetores é que eles podem armazenar dados de qualquer tipo.

- Somente a alternativa I está correta.
- Somente a alternativa II está correta.
- Somente a alternativa III está correta.
- Somente as alternativas I e II estão corretas.
- Somente as alternativas II e III estão corretas.

**2.** Usando vetores do tipo *char* é possível construir cadeias de caracteres mais comumente chamado de *string*. A biblioteca `<string.h>` fornece várias funções para manipulação de *strings*, como por exemplo, função que conta número de caracteres, que guarda novas *strings*, que compara o tamanho de duas *strings* etc.

A função `strcmp(var1,var2)` compara o conteúdo de duas variáveis. Considerando que `var1` e `var2` possuem como conteúdo a mesma *string*: "engenharia", escolha a opção que contém o resultado da função.

- a) engenharia
- b) 1
- c) 0
- d) var2
- e) var1

**3.** No desenvolvimento de uma solução que utiliza vetores, estes são utilizados juntamente com estruturas de repetição, sendo a com variável de controle a preferida para esse caso, pois a variável que controla o laço de repetição pode ser utilizada como índice para o vetor.

Considere o programa abaixo e escolha a opção que contém o que será impresso na linha 8.

```
1. #include<stdio.h>
2. void main(){
3. int resultado[5];
4. int i;
5. for(i=0;i<5;i++){
6. resultado[i] = i+2;
7. }
8. printf("Resultado = %d",resultado[2]);
9. getchar();
10. }
```

- a) Resultado = 0
- b) Resultado = 2
- c) Resultado = 3
- d) Resultado = 4
- e) Resultado = 5

## Seção 4.2

### Programação e funções com matrizes

#### Diálogo aberto

Caro estudante, bem-vindo a mais uma seção! Continuaremos nosso estudo a respeito de variáveis compostas na linguagem C, agora explorando as variáveis bidimensionais. Essa estrutura nos permite armazenar valores não somente em linhas, como acontece no vetor, mas também em colunas. Através dessa estrutura é possível criar tabelas na memória do computador, otimizando o armazenamento de dados e uso de variáveis.

Nessa seção você completará sua segunda missão da Kro Engenharias, você deverá entregar um programa que faça o controle da temperatura dos braços mecânicos da linha de soldagem de peças automotivas. Perceba que devemos armazenar no mínimo dois valores: qual o braço e qual a temperatura. Para isso você recebeu uma planilha com o código de 10 equipamentos de uma linha de produção:

Tabela 4.1 | Código dos braços mecânicos da linha de soldagem

| Braço | Código | Setor    |
|-------|--------|----------|
| 1     | 1010   | Soldagem |
| 2     | 1020   | Soldagem |
| 3     | 1030   | Soldagem |
| 4     | 1040   | Soldagem |
| 5     | 1050   | Soldagem |
| 6     | 1060   | Soldagem |
| 7     | 1070   | Soldagem |
| 8     | 1080   | Soldagem |
| 9     | 1090   | Soldagem |
| 10    | 1100   | Soldagem |

Fonte: elaborada pelo autor.

Portanto, em seu programa cada um dos braços deve ser identificado pelo seu código. Além disso, para cada equipamento

serão armazenadas 3 medidas de temperatura, e ainda a média dessa temperatura.

Para completar essa missão você verá nessa seção a sintaxe de matrizes na linguagem C, como é feita a atribuição de valores em uma matriz e como é feita a leitura dos elementos nessa estrutura de dados.

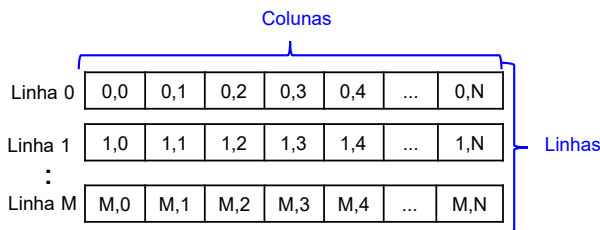
Preparado? Bons estudos!

## Não pode faltar

### Definição, características e sintaxe de matrizes

Manzano (2015) utiliza apenas uma nomenclatura para definir variáveis compostas: **matrizes**, e as divide em matrizes unidimensionais e bidimensionais. A nomenclatura adotada pelo autor, embora não seja a mais comum, faz sentido, pois o que difere ambas variáveis compostas, do ponto de vista do programador, é apenas a quantidade de índices que ambas possuem: vetor  $\rightarrow$  1 dimensão  $\rightarrow$  1 índice; matriz  $\rightarrow$  2 dimensões  $\rightarrow$  2 índices. Do ponto de vista estrutural, uma matriz é um agrupamento de vetores, formando uma tabela na memória de trabalho do computador (Figura 4.11).

Figura 4.11 | Organização estrutural de uma matriz



Fonte: elaborada pelo autor.

Existem algumas características a respeito dessa variável que devem ficar claras:

- A capacidade da matriz, ou seja, a quantidade de linhas e colunas tem que ser determinada no momento em que se declara essa variável especial. A quantidade de elementos é a multiplicação da quantidade de linhas por colunas ( $M \times N$ ).
- A matriz é uma variável composta bidimensional, ou seja, o acesso a seus dados tanto para leitura como para escrita precisa de dois índices. Veja na Figura 4.11, que dentro de cada espaço está o número da linha seguido do número da coluna.

- A matriz tem que ter o tipo de dado que irá armazenar no momento da declaração.

Vejamos como trabalhar com essa estrutura, primeiro aprendendo sua sintaxe:

Sintaxe: Tipo variável[M][N]



$M$  é a quantidade de linhas  
 $N$  é a quantidade de colunas (ou linhas)

Exemplos:

```
int numeros[5][3];
float temperaturas[100][3];
double distanciaAtomos[30][2];
```

Nos exemplos a matriz "numeros" poderá armazenar até 15 valores inteiros, a matriz "temperaturas" poderá armazenar até 300 valores com casas decimais, e a matriz "distanciaAtomos" poderá armazenar até 60 valores com casas decimais.



Refleta

Agora que você está conhecendo mais a linguagem C, você sabe a diferença entre uma variável do tipo *float* e uma do tipo *double*? No dia a dia de um engenheiro ou cientista qual tipo é mais utilizado? Esses dois tipos são suficientes para representar todos os números que existem?

## Atribuição de valores a matrizes

Para armazenarmos um valor dentro de uma matriz usamos o índice da linha e da coluna, portanto, basta informar a variável e entre o primeiro colchete o índice da linha e entre o segundo colchete o índice da coluna e usando a atribuição escolher o valor.

variável[M][N] = valor;

Exemplos:

```
numeros[0][1] = -100;
Temperatura[50][0] = 37.5;
distanciaAtomos[5][1] = 0.02345;
```

Não se esqueça:

- Em qualquer variável composta o índice começa por zero, então em uma matriz o primeiro espaço para armazenamento é sempre (0,0), ou seja, índice 0 tanto para linha como para coluna.



- Não é obrigatório que todas as posições sejam ocupadas, você pode declarar uma matriz com 10 linhas (ou colunas) e usar somente 1.

Para armazenar valores em uma matriz digitados pelo usuário, utilizamos a função `scanf()` informando os dois índices. Veja um exemplo na Figura 4.12.



## Exemplificando

Vamos escrever um programa que armazena um valor inteiro digitado pelo usuário na primeira posição de uma matriz, ou seja, no índice (0,0). Na linha 5 do programa a função `scanf()` irá guardar o valor digitado na variável "numeros" índices: 0 e 0. Da mesma forma que os valores podem ser escritos na matriz, também podem ser lidos, veja na linha 6 que multiplicamos o valor guardado em "numeros[0][0]" por dois.

Figura 4.12 | Programa para armazenar valores em matrizes

```
1. #include <stdio.h>
2. void main(){
3. int numeros[5][3];
4. printf("Digite um numero: ");
5. scanf("%d",&numeros[0][0]);
6. printf("O dobro do que você digitou é: %d",numeros[0][0]*2);
7. getchar();
8. }
```

Fonte: elaborada pelo autor.

## Iteração sobre os elementos de matrizes

Vimos que vetores são usados juntamente com estruturas de repetição para que seus valores sejam acessados de forma mais eficiente. Com matrizes também utilizamos esse recurso, porém com uma diferença: precisamos de estruturas de repetições aninhadas. A quantidade de estruturas que precisamos está diretamente associada à quantidade de índices da variável composta, portanto se temos dois índices, usaremos duas estruturas de repetição. A primeira estrutura de repetição controla o índice das linhas, e a segunda controla o índice das colunas, portanto sua sintaxe será:

```
for(i=0; i<2; i++){
 for(j=0; j<3; j++){
 numeros[i][j];
 }
}
```

O nome da variável usada para controle no laço é opcional, porém é comum usar *i* ou *m* para linha e *j* ou *n* para coluna. Nessa estrutura a matriz possui duas linhas e três colunas, a inserção inicia na linha 0 (*for* externo), e preenche todas as colunas daquela linha (*for* interno), terminadas as colunas daquela linha, a variável *i* é incrementada e a linha 1 passa a ter suas colunas preenchidas. Veja na Tabela 4.2 a ordem de inserção dos dados.

Tabela 4.2 | Ordem de inserção em uma matriz 2 x 3

|    |    |    |
|----|----|----|
| 1ª | 2ª | 3ª |
| 4ª | 5ª | 6ª |

Fonte: elaborada pelo autor.



## Exemplificando

Vamos escrever um programa que armazena 6 valores digitados pelo usuário em uma matriz. Como já vimos os valores serão armazenados em sequência nas linhas e colunas. Como o *printf()* está dentro dos dois *for()*, serão impressas 6 mensagens pedindo que o usuário digite um valor, e cada um será guardado na sua respectiva posição conforme iteração das variáveis *i* - e - *j*.

```
1. #include<stdio.h>
2. void main(){
3. float matriz[2][3];
4. int i, j;
5. for(i=0;i<5;i++){
6. for(j=0;j<5;j++){
7. printf("Digite o valor[%d][%d]: ",i,j);
8. scanf("%f",&matriz[i][j]);
9. }
10. }
11. }
```



## Assimile

Para iterar sobre os elementos de uma matriz são necessárias duas estruturas de repetição com variável de controle, em C utiliza-se

o **for**. O laço externo irá controlar a iteração sobre as linhas, e o laço interno controlará a iteração sobre as colunas. Conseqüentemente, a ordem de inserção em uma matriz é feita da seguinte forma:

- Inicia-se pela linha 0, coluna 0.
- Em seguida, através do “for” interno, todas as colunas da linha 0 são preenchidas.
- Acabadas as colunas da linha 0, o laço externo é incrementado e a linha passa a valer 1.
- Em seguida, todas as colunas da linha 1 são preenchidas.
- E assim sucessivamente.

## Problemas utilizando funções com matrizes

Matrizes são amplamente utilizadas para armazenar valores temporariamente a fim de efetuar diversos cálculos. Uma das grandes aplicações de matrizes é na computação gráfica, pois cada pixel de uma imagem é representada com um valor na matriz. Outro recurso matemático muito utilizado para resolver problemas em engenharia é a diagonalização de matrizes. Através da diagonalização encontram-se elementos chamados autovalores e autovetores que são utilizados para resolver diversos problemas. Segundo Colares (2011), o mecanismo de busca do Google utiliza esse recurso para ranquear as páginas de resultado.

Vamos implementar um programa usando matriz que soma os elementos da diagonal principal. Lembrando que a matriz precisa ser quadrada, ou seja, o número de linhas e colunas deve ser o mesmo. Veja na Tabela 4.3 uma matriz 3 x 3, na qual os elementos da diagonal principal estão destacados.

Tabela 4.3 | Matriz 3 x 3 com diagonal principal destacada

|     |     |     |
|-----|-----|-----|
| 30  | 20  | -12 |
| -1  | 100 | 9   |
| -33 | 90  | 1   |

Fonte: elaborada pelo autor.



Até o momento, em todos os programas que você criou não funcionava a acentuação do teclado. Caso tenha usado palavras com acento na função `printf()` o resultado no `prompt` certamente não foi o esperado. Para utilizarmos acentos devemos usar a função `setlocale(LC_ALL, "Portuguese");` que faz parte da biblioteca `<locale.h>`.

Vamos utilizar a função para acertar a acentuação no programa da Figura 4.13, que soma os elementos da diagonal principal de uma matriz 5 x 5. Tal função está sendo usada logo após a declaração das variáveis na função principal na linha 18. Das linhas 3 até 14 está a função que faz o processo de verificação e soma. Veja que a função retorna `void`, ou seja, faz seu processo e não retorna nenhum valor para a função que a chamou, e recebe como parâmetro uma variável matricial. Quando o elemento pertence à diagonal principal seus índices de linha e coluna são iguais, por isso na linha 8 fazemos essa verificação. Caso sejam iguais, o valor dessa posição é acumulado na variável "soma". Das linhas 15 até 26 está a função principal que armazena os valores na variável "matriz" e na linha 25 é utilizada a função auxiliar para somar a diagonal principal, passando como parâmetro a matriz.

Figura 4.13 | Programa para somar elementos da diagonal principal

```
1. #include<stdio.h>
2. #include<locale.h>
3. void SomarDiagonal(float valores[5][5]){
4. float soma = 0;
5. int i, j;
6. for(i=0;i<5;i++){
7. for(j=0;j<5;j++){
8. if(i == j){
9. soma = soma + valores[i][j];
10. }
11. }
12. }
13. printf("A soma da diagonal principal é = %.2f",soma);
14. }
15. void main(){
```

```
16. float matriz[5][5];
17. int i, j;
18. setlocale(LC_ALL, "Portuguese");
19. for(i=0; i<5; i++){
20. for(j=0; j<5; j++){
21. printf("Digite o valor[%d][%d]: ", i, j);
22. scanf("%f", &matriz[i][j]);
23. }
24. }
25. SomarDiagonal(matriz);
26. }
```

Fonte: elaborada pelo autor.



## Pesquise mais

Para uma melhor compreensão e domínio da estrutura matricial leia o capítulo 7 sobre matrizes do livro: MANZANO, José Augusto N. G. **Linguagem C: acompanhada de uma xícara de café.** São Paulo: Érica, 2015. 480 p. e o capítulo 6 do livro: MANZANO, José Augusto N. G. **Estudo dirigido de linguagem C.** São Paulo: Érica, 2013.

Ambos os livros estão disponíveis na biblioteca virtual <<https://biblioteca-virtual.com>>. Acesso em: 25 dez. 2017.

Parabéns por ter concluído mais uma importante etapa do caminho para a construção de programas computacionais. Tivemos a oportunidade de ver nessa seção a definição, características e sintaxe de matrizes na linguagem C, além da atribuição de valores e iteração sobre os elementos de matrizes. Persista nos estudos e em breve terá concluído todo o livro.

## Sem medo de errar

Após ter estudado o uso da estrutura matricial na linguagem C, você já tem condições de resolver o problema do controle da temperatura dos braços mecânicos da linha de soldagem de peças automotivas. Foi lhe passado uma planilha com o código dos 10 braços mecânicos utilizados nessa linha, cada código é utilizado para identificar uma máquina. Para cumprir o

desafio você deve ter em mãos os dados da planilha. Além do código, você deve armazenar 3 temperaturas para cada equipamento além da média.

Siga os seguintes passos:

- Crie uma matriz com 10 linhas e 5 colunas. As colunas vão armazenar: código, temp1, temp2, temp3, média.
- Guarde os 10 códigos na coluna 0 da matriz.
- Preencha as outras 3 colunas com a temperatura de cada máquina.
- Calcule a média e guarda na coluna 4 da matriz.
- Exiba na tela a média de cada um dos 10 braços mecânicos.

Na Figura 4.14 propomos uma possível solução para esse problema usando 3 funções além da principal, dessa forma o código fica mais organizado e de fácil manutenção. Tal solução usa variáveis globais e locais, caso ainda tenha dúvida consulte seu professor.

Figura 4.14 | Algoritmo para calcular a média da temperatura dos braços mecânicos

```
1. #include<stdio.h>
2. #include<locale.h>
3. float braco[10][5];
4. int i, j;
5. void ArmazenarCodigo(){ //função para guardar o código dos braços
6. int codigo = 1000; //variável local
7. for(i=0;i<10;i++){
8. codigo = codigo + 10;
9. braco[i][0] = codigo;
10. }
11. }
12. void ArmazenarTemperaturas(){ //função para guardar os demais dados
13. ArmazenarCodigo(); //chamando a função para atribuir os códigos.
14. for(i=0;i<10;i++){
15. printf("Digite as 3 temperaturas do braço %.0f: \n",braco[i][0]);
16. scanf("%f %f %f",&braco[i][1],&braco[i][2],&braco[i][3]);
17. braco[i][4] = (braco[i][1] + braco[i][2] + braco[i][3])/3;
18. }
19. }
20. void ExibirDados(){ //função para exibir os médias calculadas
21. ArmazenarTemperaturas(); //chamando a função para atribuir valores
```

```

22. for(i=0;i<10;i++){
23. printf("\n Média do braço %.0f: %.2f",braco[i][0],braco[i][4]);
24. }
25. }
26. void main(){
27. setlocale(LC_ALL,"Portuguese");
28. ExibirDados(); //chamando a função para exibir os dados
29. }

```

Fonte: elaborada pelo autor.

Essa não é a única forma de solucionar o problema proposto, caso tenha implementado de outra forma, ótimo trabalho! Caso ainda não tenha feito, tente apresentar uma nova solução.

## Avançando na prática

### Calcular o determinante de uma matriz 2 x 2

#### Descrição da situação-problema

Programas para engenharia são implementados visando realizar uma série de cálculos. Uma das formas mais eficientes de se armazenar dados para processamento é a forma matricial. Tal estrutura permite uma série de operações, dentre elas o cálculo do determinante. Calcular o determinante de uma matriz significa transformar uma matriz quadrada ( $M = N$ ) em escalar. Tal operação permite saber se a matriz possui inversa ou não, pois caso o determinante seja zero, a matriz não possui inversa.

#### Resolução da situação-problema

O determinante de uma matriz 2 x 2 é feito pela diferença da multiplicação dos elementos da diagonal principal pela multiplicação dos elementos da diagonal secundária. A identificação dos elementos da diagonal principal é feita verificando se os índices da linha e da coluna são iguais ( $i == j$ ), já para identificar os elementos da diagonal secundária verificamos se o índice da linha é igual ao maior índice de colunas menos o índice da coluna atual ( $i == \text{maiorIndiceColuna} - j$ ). Veja na Figura 4.15 a solução do determinante.

Figura 4.15 | Programa para calcular determinante 2 x 2

```
1. #include<stdio.h>
2. #include<locale.h>
3. void CalcularDeterminante(float valores[2][2]){
4. int i,j;
5. float diagPrinc = 1, diagSec = 1, determinante;
6. for(i=0;i<2;i++){
7. for(j=0;j<2;j++){
8. if(i == j){
9. diagPrinc = diagPrinc * valores[i][j];
10. }else if(i == 1 - j){
11. diagSec = diagSec * valores[i][j];
12. }
13. }
14. }
15. determinante = diagPrinc - diagSec;
16. printf("\n Determinante = %.2f",determinante);
17. }
18. void main(){
19. float matriz[2][2];
20. int i,j;
21. setlocale(LC_ALL,"Portuguese");
22. for(i=0;i<2;i++){
23. for(j=0;j<2;j++){
24. printf("\n Digite o elemento[%d][%d]: ",i,j);
25. scanf("%f",&matriz[i][j]);
26. }
27. }
28. CalcularDeterminante(matriz);
29. }
```

Fonte: elaborada pelo autor.

## Faça valer a pena

**1.** Matriz é uma das estruturas que existe para o armazenamento de dados e é formada a partir de um vetor. Na verdade, alguns autores tratam o vetor como um caso particular de uma matriz (MANZANO, 2013). Sua utilização é vasta devido à facilidade em ler e escrever dados e à velocidade com que as operações são realizadas.



Analise as asserções a seguir e a relação proposta entre elas.

I – Matriz é uma estrutura de dados utilizada para armazenar dados em forma de tabela, ou seja, distribuídos em linhas e colunas, e uma das características dessa estrutura de dados é ser dinâmica.

PORQUE

II – A capacidade da matriz precisa ser informada no momento da declaração da variável, sendo necessário informar dois valores, um para representar o índice da linha e outro para representar o índice da coluna.

- a) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- c) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- d) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e) As asserções I e II são proposições falsas.

**2.** Quando utiliza-se uma solução com matrizes é preciso ter clareza sobre a ordem de inserção dos dados. Tanto o índice da linha quanto da coluna iniciam-se em zero, portanto a primeira posição a ser ocupada é linha 0 – coluna 0.

Considere o programa abaixo e escolha a opção que contém o que será impresso na linha 11.

```
1. #include<stdio.h>
2. void main(){
3. float matriz[2][2];
4. int i,j,resultado = 0;
5. for(i=0;i<2;i++){
6. for(j=0;j<2;j++){
7. matriz[i][j] = i + j;
8. resultado = resultado + matriz[i][j];
9. }
10. }
11. printf("\n Resultado = %d",resultado);
12. }
```

- a) Resultado = 0
- b) Resultado = 1
- c) Resultado = 2
- d) Resultado = 3
- e) Resultado = 4

3. No desenvolvimento de uma solução que utiliza matriz, normalmente são utilizadas duas estruturas de repetição, pois a primeira estrutura é utilizada para controlar o índice das linhas e a segunda estrutura é utilizada para controlar o índice das colunas.

Considerando a matriz dada, escolha a alternativa que contém o que será impresso na linha 14.

|     |     |    |
|-----|-----|----|
| 100 | 10  | 20 |
| -1  | 40  | 30 |
| 55  | -25 | 10 |

```
1. #include<stdio.h>
2. #include<locale.h>
3. void main(){
4. float matriz[3][3];
5. int i,j,resultado = 0;
6. setlocale(LC_ALL,"Portuguese");
7. for(i=0;i<3;i++){
8. for(j=0;j<3;j++){
9. if(i == 2 - j){
10. resultado = resultado + matriz[i][j];
11. }
12. }
13. }
14. printf("\n Resultado = %d",resultado);
15. }
```

- a) Resultado = 0
- b) Resultado = 150
- c) Resultado = 130
- d) Resultado = 69
- e) Resultado = 115

## Seção 4.3

### Recursividade

#### Diálogo aberto

Caro estudante, bem-vindo à última seção do livro de algoritmos e lógica de programação. Muitas informações que contribuem com sua formação foram apresentadas ao longo do livro. Nessa unidade, primeiro você viu como trabalhar com vetores na linguagem C, em seguida você aprendeu como trabalhar com matrizes nessa linguagem, em ambos os casos as funções estiveram presentes para tornar os programas mais organizados e reutilizáveis, evitando a repetição de códigos.

Nessa seção iremos aprofundar um pouco mais nosso conhecimento a respeito das funções, conhecendo o que é uma função recursiva e como e quando utilizá-la. Para desenvolver mais essa competência você irá criar um programa para a Kro Engenharias que faz o cálculo da raiz quadrada de um número usando o método de Newton (aproximações sucessivas de Newton). Esse procedimento é um clássico no estudo de recursividade e você terá a oportunidade de apresentá-lo à empresa para finalizar seu trabalho e mostrar sua evolução.

Uma das formas de se calcular a raiz de um número é utilizar um método iterativo chamado de Método de Newton, o qual utiliza a definição da reta tangente e possui a seguinte fórmula:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$
 (PIRES, 2015). Reduzindo essa fórmula geral para o caso do cálculo de uma raiz quadrada, tem-se:  $x_n = \frac{x_{n-1}^2 + n}{2x_{n-1}}$ , onde

$x_n$  é a raiz quadrada,  $x_{n-1}$  é a raiz anterior calculada pelo método e  $n$  é o número cuja raiz se deseja calcular. Para calcular a raiz quadrada usando o método iterativo você precisará informar: (i) o número que pretende calcular, (ii) um valor inicial para a raiz e (iii) um critério de parada, pois a partir do valor inicial o método fará os cálculos até que o critério de parada seja atingido. Métodos iterativos e funções

recursivas estão fortemente ligados, pois a partir de um resultado o método iterativo calcula o próximo resultado, o que remete à ideia de função recursiva.

Prontos para começar? Bons estudos!

## Não pode faltar

Você já aprendeu que utilizar funções melhora a organização do código, permite a reutilização de trechos de código e evita a repetição de comandos. Vimos que para criar uma função temos que seguir uma determinada estrutura com os seguintes passos: (i) determinar o tipo de retorno da função (lembrando que quando a função não retorna nada usamos *void*); (ii) escolher o nome da função (lembrando que a condição para o nome é o mesmo da nomenclatura para variáveis); (iii) entre parênteses especificar quantos e quais tipos de parâmetros a função receberá; (iv) especificar os comandos da função, incluindo as variáveis locais (variáveis que pertencem somente a essa função); (v) se o tipo de retorno for diferente de *void*, então a função deverá ter o comando ***return valor***. Os itens (iv) e (v) sempre estarão entre chaves que marcam o início e o final da função.

```
<tipo> nome(<tipo de parâmetros> <parâmetros>) {
 <Variáveis locais>;
 <Comandos>;
 <return> <valor>;
}
```

Para relembrar o uso do *return* vamos ver um exemplo de uma função que soma ao valor de uma conta uma taxa de juros de 15% (Figura 4.16). Desse exemplo quatro pontos devem ficar claros para que possamos prosseguir nosso estudo sobre funções:

- A função será executada quando a função principal (*main*) alcançar a linha 9. Todo programa sempre começará a ser executado pela função *main*.
- A função será executada e retornará um valor do tipo *float*, pois foi definido dessa forma na linha 2.
- Como a função retorna um valor o uso do comando *return* é obrigatório, por isso na linha 2 usamos tal comando já com o cálculo do acréscimo de 15% sobre o valor da conta.

- Quando deseja-se guardar o resultado de uma função dentro de uma variável, esta deve ter o mesmo tipo do retorno da função. Por isso, na linha 6 declaramos a variável resultado como *float*.

Figura 4.16 | Função que adiciona juros a um valor

```

1. #include<stdio.h>
2. float AdicionarJuros(float valor){
3. return valor * 1.15;
4. }
5. void main(){
6. float conta, resultado;
7. printf("Digite o valor da conta: ");
8. scanf("%f",&conta);
9. resultado = AdicionarJuros(conta);
10. printf("\n Valor final da conta = %.2f",resultado);
11. getchar();
12. }
```

Fonte: elaborada pelo autor.

Entendidos os pontos vitais sobre funções, estamos preparados para avançar em seu estudo aprendendo o recurso de recursividade.

## Definição e caracterização de recursividade

Recursividade significa recorrer a uma determinada situação (AULETE, 2017). Em programação uma função recursiva é uma função que chama a ela própria ou ainda nas palavras de Soffener (2017): "Recursividade é a possibilidade de uma função chamar a si mesma." (SOFFENER, 2017, p. 107). Vejamos a sintaxe de uma função recursiva na Figura 4.17:

Figura 4.17 | Sintaxe de função recursiva

```

<tipo> funcaoRecursiva(){
 //comandos
 funcaoRecursiva(); ← Chamando a si próprio
 //comandos
}
void main(){
 //comandos
 funcaoRecursiva();
 //comandos
}
```

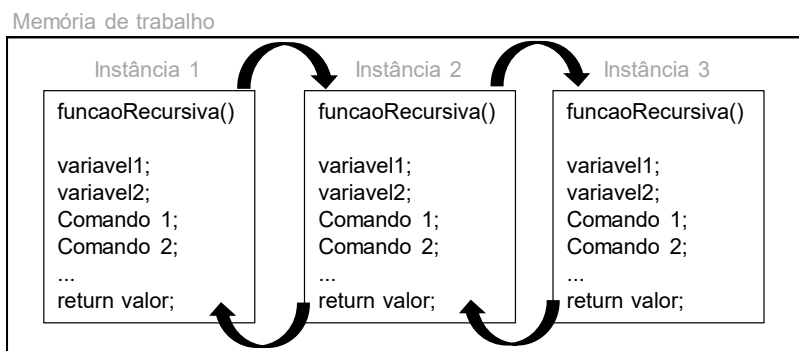
Fonte: elaborada pelo autor.

Portanto, para criar uma função recursiva basta fazermos uma chamada da função dentro da própria função. Embora a sintaxe seja simples, precisamos entender seu funcionamento e quando usar essa técnica, pois, se mal estruturada, a função pode entrar em um laço de repetição infinito.

Primeiro, vamos entender como funciona uma função recursiva:

- Por definição a função chama a ela mesmo, portanto é preciso estabelecer quando parar esse laço, para isso uma estrutura condicional pode ser usada.
- Para cada chamada da função é criada uma nova ocorrência da função na memória (instância) com os comandos e variáveis alocados em outro local. Embora as variáveis possuam o mesmo nome, elas são independentes justamente por possuírem endereços diferentes. Veja na Figura 4.18 um esquema que representa essa ideia.

Figura 4.18 | Esquema do funcionamento de função recursiva



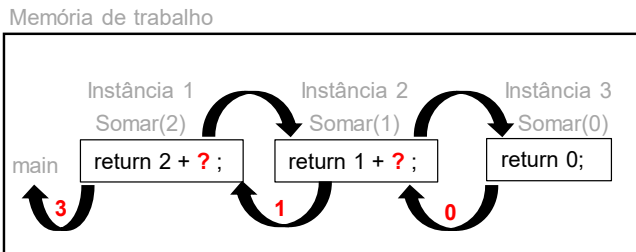
Fonte: elaborada pelo autor.

- Cada instância da função deve retornar um valor para a instância anterior, ou seja, a que lhe deu origem, portanto a instância 3 retornará para a instância 2, e a 2 para a 1. Podemos resumir que a instância  $N$  sempre retornará um resultado para a instância  $N - 1$ , caso isso não aconteça, tem-se um erro.
- Toda função recursiva tem um caso (uma instância) chamado de *caso base*. Esse é o caso mais simples, que interromperá o laço, pois sendo o mais simples será resolvido sem precisar chamar novamente a função.



Vejam como funciona a função recursiva e seu caso base em um exemplo que, a partir de um valor inteiro, informado pelo usuário, todos os seus antecessores maiores que zero são somados (Figura 4.19). Na linha 16 a função recursiva é chamada passando como parâmetro o valor informado pelo usuário. A partir daí, a função *Somar(int valor)* entra em ação. Se o usuário digitar zero, a condicional da linha 4 retorna falso e a função retorna zero, mas caso o valor seja diferente de zero o programa entra na linha 5, onde está escrito a recursão. Vamos fazer uma simulação supondo que o usuário digitou 2. A primeira instância da função é criada com parâmetro 2; quando a função chega na linha 5, o *return* chama novamente a função *Somar()*, criando uma nova instância na memória, mas agora passando *n-1*, ou seja, 1; novamente na linha 5 o *return* chama novamente a função *Somar()*, criando uma nova instância na memória, mas agora passando *n-1*, ou seja, 0. Nesse momento a condicional na linha 4 não é falsa, e a função não gera uma nova instância, retornando 0. A partir desse retorno as demais instâncias podem ser completadas e retornam o resultado final para a chamada inicial que está dentro da função *main*. A instância 3 do exemplo não gerou uma nova instância porque ela continha o *caso base*, e a função pode ser resolvida.

Figura 4.19 | Função recursiva para soma



1. `#include<stdio.h>`
2. `#include<locale.h>`
3. **`int Somar(int valor){`**
4. `if(valor != 0){ //condição que interromperá o laço`
5. `return valor + Somar(valor-1); //chamada recursiva`
6. `}`
7. `else{`
8. `return valor;`

```

9. }
10. }
11. void main(){
12. setlocale(LC_ALL,"Portuguese");
13. int n, resultado;
14. printf("\n Digite um número inteiro: ");
15. scanf("%d",&n);
16. resultado = Somar(n);
17. printf("\n Resultado da soma = %d",resultado);
18. getchar();
19. }

```

Fonte: elaborada pelo autor.

## Situações de programação adequadas para a recursividade

O problema da soma de números proposto na Figura 4.19 poderia ser resolvido usando uma estrutura de repetição, por exemplo, `for(i=0;i<=n;i++)`? A resposta é sim. A função recursiva substitui estruturas de repetição e tem como objetivo tornar o código mais simples e elegante, consequentemente o entendimento e manutenção se tornam mais fáceis.

Segundo Feofiloff (2017), professor da Universidade de São Paulo (USP):



Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema. Dizemos que esses problemas têm estrutura recursiva. Para resolver um tal problema, podemos aplicar o seguinte método:

se a instância em questão for pequena, resolva-a diretamente (use força bruta se necessário);

senão, reduza-a a uma instância menor do mesmo problema, aplique o método à instância menor, volte à instância original.

A aplicação desse método produz um algoritmo recursivo. (FEOFILOFF, 2017, p. 1).



Portanto, a recursão deve ser usada sempre que um problema maior puder ser dividido em partes menores do mesmo problema.



### Assimile

Recursividade é uma técnica sofisticada em programação na qual uma função chama a si mesma criando várias instâncias (chamadas recursivas). Embora seja uma técnica que proporciona um código mais limpo e facilita a manutenção, seu uso deve levar em consideração a quantidade de memória necessária para a execução do programa.

Além de analisar a essência do problema, outros critérios devem ser considerados na escolha de um algoritmo recursivo: (i) a cada chamada recursiva aloca recursos na memória para a função, se a função for muito grande poderá ocorrer um estouro de memória. (ii) é preciso avaliar o custo-benefício em se ter um código mais sofisticado em detrimento de uma estrutura de repetição, pois a segunda opção gasta menos memória.



### Refleta

Muitos problemas podem ser resolvidos tanto com recursividade quanto com estruturas de controle. Existe um critério bem definido para embasar a escolha de qual técnica utilizar? Em qual situação você optaria por uma função recursiva?

## Comparação de procedimentos recursivos com procedimentos não recursivos

Faremos agora duas implementações, uma usando estrutura de repetição e outra usando recursividade de um problema clássico no mundo da programação: o cálculo do fatorial. O fatorial de um número é a multiplicação dele por seus antecessores, portanto fatorial de 5 é 120, pois:  $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ . Representamos o fatorial de um número usando exclamação (N!).

Na primeira solução (Figura 4.20 – a), foram necessárias 3 variáveis para realizar o cálculo (*valor, i, fatorial*), caso o programa

necessitasse de manutenção, teríamos que nos atentar aos pontos que ocorrem tais variáveis. Considerando o programa todo foram usadas 5 variáveis ocupando um total de 20 bytes na memória de trabalho, independente do número escolhido para calcular o fatorial (Cada variável inteira ocupa 4 bytes, temos 5 variáveis no total).

Na segunda solução foi necessário somente a variável "valor" para fazer o cálculo do fatorial e considerando o programa todo foram usadas 3 variáveis (*n*, *resultado*, *valor*). Embora a solução com recursividade seja mais elegante e mais fácil de fazer manutenção, o tamanho total que essa solução ocupa na memória varia conforme o número escolhido para calcular o fatorial, por exemplo, se o valor escolhido for 5!, a solução ocupará 8 bytes da função *main* + 20 bytes das chamadas recursivas, totalizando 28 bytes.

Figura 4.20 | Fatorial não recursivo (a) x recursivo (b)

a) solução não recursiva

```
1. #include<stdio.h>
2. #include<locale.h>
3. int CalcularFatorial(int valor){
4. int i,fatorial=1;
5. for(i=1;i<=valor;i++){
6. fatorial = fatorial * i;
7. }
8. return fatorial;
9. }
10. void main(){
11. setlocale(LC_ALL,"Portuguese");
12. int n, resultado;
13. printf("\n Digite um número inteiro: ");
14. scanf("%d",&n);
15. resultado = CalcularFatorial(n);
16. printf("\n Resultado do fatorial = %d",resultado);
17. getchar();
18. }
```

b) solução recursiva

```
1. #include<stdio.h>
2. #include<locale.h>
3. int CalcularFatorial(int valor){
4. if(valor != 1){
5. return valor * CalcularFatorial(valor-1);
6. } else {
7. return valor;
8. }
9. }
10. void main(){
11. setlocale(LC_ALL,"Portuguese");
12. int n, resultado;
13. printf("\n Digite um número inteiro: ");
14. scanf("%d",&n);
15. resultado = CalcularFatorial(n);
16. printf("\n Resultado do fatorial = %d",resultado);
17. getchar();
18. }
```

Fonte: elaborada pelo autor.



**Pesquise mais**

Acesse a página do professor Paulo Feofiloff, disponível em: <<https://www.ime.usp.br/~pf/algoritmos/aulas/recu.html>>. Acesso em: 2 jan. 2018, e faça os exercícios propostos para exercitar o uso de funções recursivas.

Visite a página <<http://linguagemc.com.br/recursividade-em-c>>. Acesso em: 2 jan. 2018, veja uma outra implementação do algoritmo para fatorial.

## Sem medo de errar

Como último desafio para a Kro Engenharias foi proposto a você implementar um algoritmo recursivo para cálculo da raiz quadrada

de um número usando o método de aproximações sucessivas de Newton. A função recursiva deverá conter o seguinte cálculo:

$$x_n = \frac{x_{n-1}^2 + n}{2x_{n-1}}, \text{ onde } x_n \text{ é a raiz quadrada de um número } n, x_{n-1} \text{ é}$$

a raiz anterior calculada pelo método e  $n$  é o número cuja raiz se deseja calcular. Para que o algoritmo iterativo funcione, a função deve conhecer três elementos:

- O número que se pretende calcular.
- Um valor inicial para a raiz (é um chute).
- Um critério de parada.

A resolução do problema está na Figura 4.21, vamos analisá-lo:

A função principal (*main*) tem como objetivo pedir um número para o usuário, chamar a função "CalcularRaiz" e exibir o resultado da raiz. Quando a chamada da função é feita (na linha 14) são passados dois parâmetros, o número cuja raiz se deseja calcular e um valor inicial para a raiz (o chute), o qual passamos metade do número escolhido pelo usuário ( $\text{numero}/2$ ).

Todo o cálculo acontece na função recursiva: na linha 5 foi implementada a fórmula para o cálculo da raiz quadrada segundo o método de Newton. Na linha 6 está localizado o critério de parada ( $\text{raiz} - \text{raizAnt} < 0.001$ ), ou seja, se a diferença entre a iteração atual e a anterior for inferior ao valor 0.001 significa que já tem-se uma boa aproximação do valor da raiz e o algoritmo pode ser finalizado. O critério de parada está dentro da função *fabs()*, que tem como objetivo retornar o valor absoluto de um número de ponto flutuante. Enquanto o critério não é atingido a função chama a si própria (linha 8) passando como parâmetro o mesmo valor  $n$ , porém um novo valor para a raiz, o que caracteriza tanto o método recursivo quanto o iterativo.

Figura 4.21 | Solução recursiva para o método de Newton

```
1. #include<stdio.h>
2. #include<math.h>
3. float CalcularRaiz(float n, float raizAnt)
4. {
5. float raiz = (pow(raizAnt, 2) + n)/(2 * raizAnt);
6. if (fabs(raiz - raizAnt) < 0.001)
7. return raiz;
```

```

8. return CalcularRaiz(n, raiz);
9. }
10. void main(){
11. float numero, raiz;
12. printf("\n Digite um número para calcular a raiz: ");
13. scanf("%f",&numero);
14. raiz = CalcularRaiz(numero,numero/2);
15. printf("\n Raiz quadrada funcao = %f",raiz);
16. }

```

Fonte: elaborada pelo autor.

Altere o algoritmo acima para que o resultado obtido pela função *CalcularRaiz()* seja comparado com a função *sqrt()*. Crie um novo programa para o cálculo iterativo da raiz quadrada, porém, agora sem recursividade, compare o tempo de execução de ambos.

## Avançando na prática

### Determinação do máximo divisor comum (MDC)

#### Descrição da situação-problema

No ensino fundamental aprendemos a calcular o máximo divisor comum (MDC), que é o maior número inteiro que divide outros números inteiros ao mesmo tempo. Por exemplo, se considerarmos os números 16 e 24, o MDC será 8, pois é o maior número que divide ambos ao mesmo tempo. O cálculo do MDC pode ser utilizado para localizar o valor máximo entre vários valores informados. Agora, vamos criar um programa para automatizar o cálculo do MDC entre dois números.

#### Resolução da situação-problema

Vamos implementar a resolução do MDC usando o algoritmo das divisões sucessivas. Para entender o processo, vamos considerar os valores 16, 24.

1º) Dividimos o primeiro número pelo segundo e guardamos o resto:

$$16 / 24 = 1 \text{ (com resto 16);}$$

2º) Dividimos o divisor da divisão anterior pelo resto da divisão:

$$24 / 16 = 1 \text{ (com resto 8);}$$

3º) Repetimos o segundo passo até que o resto seja zero:

$$16 / 8 = 2 \text{ (com resto 0).}$$

Quando o resto é zero, o divisor é o MDC.

Na Figura 4.22 temos o programa que faz o cálculo do MDC entre dois números. Na linha 3 usamos o operador "%" que faz a divisão entre dois números inteiros e armazena o resto. Na linha 4 temos o critério de parada ou caso base, no qual compara o resto da divisão com zero e na linha 6 temos o comando responsável pela recursão.

Figura 4.22 | Função recursiva para calcular o MDC entre dois números

```
1. #include<stdio.h>
2. int CalcularMDC(int a, int b) {
3. int r = a % b;
4. if(r == 0)
5. return b;
6. return CalcularMDC(b,(a % b));
7. }
8. void main(){
9. int n1, n2, resultado;
10. printf("\n Digite dois números inteiros positivos: ");
11. scanf("%d %d",&n1,&n2);
12. resultado = CalcularMDC(n1,n2);
13. printf("\n MDC = %d",resultado);
14. }
```

Fonte: elaborada pelo autor.

## Faça valer a pena

**1.** A recursividade é uma técnica de programação na qual uma função chama a si própria tornando o código mais limpo e elegante, o que facilita a manutenção e reutilização de trechos de códigos e funções. Toda chamada recursiva deve retornar um valor à função que "fez o chamado". Analise as asserções a seguir e a relação proposta entre elas.

I - Funções recursivas sempre podem ser utilizadas para substituir estruturas de repetição.

PORQUE

II - Toda função recursiva é composta pelo caso base e pelas chamadas recursivas funcionando como um laço de repetição.

- a) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- c) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- d) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e) As asserções I e II são proposições falsas.

**2.** Vetores são variáveis compostas unidimensionais, sendo assim, podem armazenar diversos valores ao mesmo tempo. Funções podem receber como parâmetro um vetor de qualquer tipo, portanto é possível utilizar tal estrutura de dados juntamente com funções recursivas.

Considerando o programa abaixo que utiliza um vetor em uma função recursiva, escolha a opção que representa o que será impresso na linha 18.

```
1. int funcao(int x[], int n){
2. int resultado = 0;
3. if(n >= 0){
4. resultado = funcao(x,n-1);
5. if(x[n] == 100) return resultado;
6. else return resultado = resultado + x[n];
7. }
8. else return resultado;
9. }
10. void main(){
11. int numeros[5], resultado = 0;
12. numeros[0] = 10;
13. numeros[1] = 30;
14. numeros[2] = 100;
15. numeros[3] = 5;
16. numeros[4] = 50;
17. resultado = funcao(numeros,5);
18. printf("\n Resultado = %d",resultado);
19. }
```

- a) Resultado = 195
- b) Resultado = 100
- c) Resultado = 0
- d) Resultado = 5
- e) Resultado = 95

**3.** Os profissionais especializados em desenvolvimento de software trabalham com diversos tipos de estruturas de dados, dentre elas destacamos o uso das listas. Nessa estrutura os dados vão sendo alocados sequencialmente e a ordem de chegada e saída desses dados cria alguns tipos especiais de listas, como as pilhas e as filas. As estruturas podem ser alocadas dinamicamente ou de modo estático, e para esse último caso temos os vetores. A combinação de estruturas de dados e funções possibilita a criação de importantes soluções para as mais diversas áreas.

Considerando o programa abaixo que utiliza um vetor em uma função recursiva, escolha a opção que representa o que será impresso na linha 20.

```
1. #include<stdio.h>
2. int funcao(int n, int v[]){
3. if (n == 1)
4. return v[0];
5. else {
6. int x;
7. x = funcao(n-1,v);
8. if (x > v[n-1]) return x;
9. else return v[n-1];
10. }
11. }
12. void main(){
13. int numeros[5], resultado = 0;
14. numeros[0] = 10;
15. numeros[1] = 30;
16. numeros[2] = 100;
17. numeros[3] = 5;
18. numeros[4] = 50;
19. resultado = funcao(5,numeros);
20. printf("\n Resultado = %d",resultado);
21. }
```

- a) Resultado = 0
- b) Resultado = 10
- c) Resultado = 30
- d) Resultado = 100
- e) Resultado = 5



# Referências

AULETE. O dicionário da língua portuguesa na internet. 2017. Disponível em: <<http://www.aulete.com.br/recursão>>. Acesso em: 2 jan. 2018.

COLARES, Gilderléia Bezerra. **Autovalores e Autovetores e Aplicações**. 2011. 42 f. Monografia (Especialização) - Curso de Especialização em Matemática, Universidade Federal de Santa Catarina, Foz do Iguaçu, 2011. Disponível em: <<https://repositorio.ufsc.br/bitstream/handle/123456789/107651/302312.pdf?sequence=1>>. Acesso em: 24 dez. 2017.

DEITEL, Paul; DEITEL, Harvey. **C Como Programar**. 6. ed. São Paulo: Pearson, 2011. 846 p.

FEOFILOFF, Paulo. Recursão e algoritmos recursivos. 2017. Disponível em: <<https://www.ime.usp.br/~pf/algoritmos/aulas/recu.html>>. Acesso em: 2 jan. 2018.

MANZANO, José Augusto Navarro Garcia. **Estudo Dirigido de Linguagem C**. 17. ed. rev. São Paulo: Érica, 2013.

\_\_\_\_\_. **Estudo Dirigido de Linguagem C**. São Paulo: Érica, 2010.

\_\_\_\_\_. **Linguagem C: acompanhada de uma xícara de café**. São Paulo: Érica, 2015.

PEREIRA, Silvio do Lago. Linguagem C. Disponível em: <<https://www.ime.usp.br/~slago/slago-C.pdf>>. Acesso em: 19 dez. 2017.

PIRES, Augusto de Abreu. **Cálculo numérico: prática com algoritmos e planilhas**. São Paulo: Atlas, 2015.

SOFFNER, Renato. **Algoritmos e Programação em Linguagem C**. 1. ed. São Paulo: Saraiva, 2013.







ISBN 978-85-522-0708-5



9 788552 207085 >