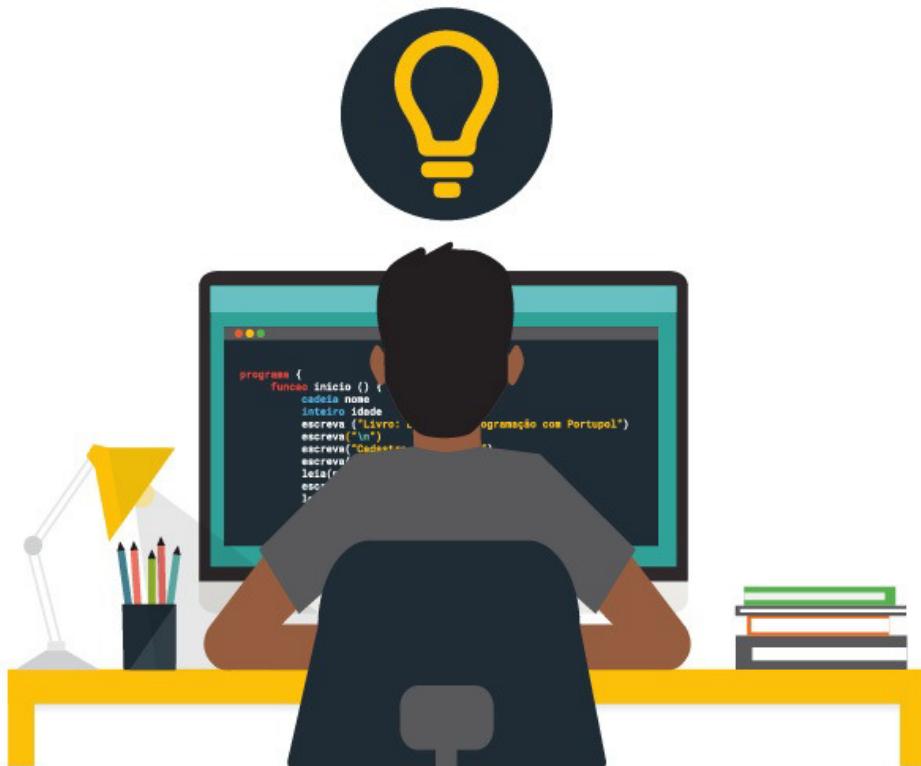


Lógica de programação com Portugol

Mais de 80 exemplos, 55 exercícios com gabarito e vídeos complementares



© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Vivian Matsui

Capa

Design Grupo Alura

[2022]

Casa do Código

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

casadocodigo.com.br



Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código é a editora do Grupo Alura, grupo que nasceu da vontade de criar uma plataforma de ensino com o objetivo de incentivar a transformação pessoal e profissional através da tecnologia.

Juntas, as empresas do Grupo constroem uma verdadeira comunidade colaborativa de aprendizado em programação, negócios, design, marketing e muito mais, oferecendo inovação na evolução dos seus alunos e alunas através de uma verdadeira experiência de encantamento.

Venha conhecer os cursos da Alura e siga-nos em nossas redes sociais.

 alura.com.br

 @casadocodigo

 @casadocodigo

ISBN

Impresso: 978-85-5519-291-3

Digital: 978-65-86110-99-9

Caso você deseje submeter alguma errata ou sugestão, acesse
<http://erratas.casadocodigo.com.br>.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Mendes, Joice Barbosa

Lógica de programação com Portugol : mais de 80 exemplos, 55 exercícios com gabarito e vídeos complementares / Joice Barbosa Mendes, Rafael da Silva Muniz. -- São Paulo : AOVS Sistemas de Informática, 2022.

ISBN 978-85-5519-291-3

1. Ciência da computação 2. Portugol (Linguagem de programação de computador) 3. Programação (Computadores) I. Muniz, Rafael da Silva. II. Título.

22-101162

CDD-005.1

Índices para catálogo sistemático:

1. Lógica de programação : Computadores : Processamento de dados 005.1

Maria Alice Ferreira - Bibliotecária - CRB-8/7964

SOBRE O LIVRO

Os sistemas informatizados e os aplicativos de celular já fazem parte do nosso dia a dia. Todos eles são programados utilizando a lógica de programação. A lógica de programação segue os mesmos conceitos da lógica do nosso dia a dia. Da hora em que acordamos até a hora em que vamos dormir executamos diversas tarefas simples que requerem uma lógica (um passo a passo da ordem do que vamos fazer e como vamos fazer).

Por exemplo, para escovar os dentes. Antes de escovarmos os dentes, precisamos colocar a pasta de dente na escova. Caso essa lógica não seja seguida, a escovação não ocorrerá corretamente. Então mesmo as tarefas mais corriqueiras seguem uma lógica, como: jogar videogame, tomar banho, almoçar, assistir televisão, usar o celular, mandar mensagem, comprar um produto em uma loja, entre outras.

Essa lógica, ao ser aplicada a um sistema ou aplicativo, passa a ser chamada de lógica de programação.

Este livro apresenta de maneira simples e objetiva os principais conceitos da lógica de programação através do Portugol. O Portugol é uma linguagem de programação escrita em português e com foco no ensino. Ele é mais simples de ser aprendido do que uma linguagem de programação profissional, já que utiliza palavras do nosso dia a dia na construção da lógica e dos algoritmos. Os algoritmos são a escrita de uma lógica de programação para resolver um mesmo problema diversas vezes.

Neste livro, serão apresentados todos os conceitos, conteúdos e

comandos necessários para a criação da lógica de programação e dos algoritmos. Os tópicos apresentados no livro são: o que é lógica, o que é lógica de programação, o que é algoritmo, tipos de algoritmos, o Portugol, a ferramenta Portugol Studio, variáveis, comando de escrita na tela, comando de leitura do teclado, operadores matemáticos, estruturas condicionais, operadores relacionais e lógicos, estruturas de repetição, vetores, matrizes e funções.

Ainda são apresentados 85 exemplos de código, 55 exercícios de fixação com gabarito e um projeto de uma loja de bicicleta com o gabarito gravado em vídeo. Todos os capítulos apresentam um resumo do que foi apresentado no livro e um vídeo complementar disponibilizado na internet.

PARA QUEM ESTE LIVRO É INDICADO

Este livro é indicado para quem está iniciando em cursos técnicos de informática, estudantes de graduação em todos os cursos da área de informática ou em outras áreas que tenham disciplinas ligadas à lógica de programação. Também é indicado para pessoas interessadas em dar o primeiro passo no mundo da programação de computadores. Como o livro foi pensado para esse público-alvo, que não tem experiência com a lógica de programação, não há nenhum pré-requisito para iniciar a leitura do livro, bastando ter um pouco de curiosidade, vontade de aprender e um celular, tablet ou computador. O material também é indicado para professores que ministram disciplinas iniciais de lógica de programação, já que ele pode ser usado como material didático em sala de aula.

Agradecimentos

Primeiramente, agradeço a Deus pelo dom da vida e pela trajetória de aprendizado e evolução que me permitiu trilhar. Gostaria de agradecer à grande paixão da minha vida, minha irmã Jéssica Mendes, que sempre me instigou a seguir em frente, me motivando a ser a melhor versão de mim mesma e me fazendo rir nos momentos mais desesperadores. Agradeço também ao professor, grande mestre e amigo Rafael Muniz, pela parceria maravilhosa que sempre tivemos ao longo dos anos trabalhando juntos; e aos amigos e parceiros do IFSP. Por último, agradeço imensamente a todos os estudantes com quem tive o prazer de ter contato, ensinando e aprendendo; vocês sempre foram a minha maior motivação em querer fazer o melhor. - *Professora Mestra Joice Barbosa Mendes*

Gostaria de agradecer primeiramente a Deus pela minha vida e pela família maravilhosa que me concedeu. Gostaria também de agradecer a minha esposa Adriana que me ajudou no processo da escrita brincando com as crianças enquanto eu me concentrava na frente do computador para escrever os capítulos e os exercícios. Agradeço também aos meus pais pela educação que me foi dada e aos meus filhos, Leo e Be, pelos ensinamentos e carinho diário. Agradeço aos meus colegas de trabalho desde a época da Petrobras, passando pelo IBGE, Casa da Moeda, UERJ e IFSP, e aos alunos que me ajudaram com suas dúvidas nas aulas me instigando a melhorar a forma de ensinar o conteúdo de lógica de programação para quem está tendo contato pela primeira vez. - *Professor Mestre Rafael da Silva Muniz*

PREFÁCIO

Cara leitora, caro leitor,

Antes de mais nada, preciso fazer um alerta: ao ler (estudar) este livro, você vai se apaixonar por lógica de programação!

Talvez eu seja suspeito para falar, pois, particularmente, sempre fui apaixonado por questões de lógica. A lógica está em tudo o que fazemos na vida, desde o simples ato de acordar e se trocar – quem é que nunca levantou com sono e colocou a camiseta do avesso? - até a realização de grandes empreendimentos, como construir um prédio, uma ponte, uma igreja etc. Muitas vezes, as tarefas mais simples ou corriqueiras são realizadas de forma automática, mas, ainda assim, há uma sequência lógica para fazê-las. Por exemplo, não é possível cozinar sem acender o fogão, não é mesmo? Por outro lado, para tarefas mais complexas, como as de construir uma casa, é necessário um planejamento e este, obrigatoriamente, deverá seguir uma lógica na sequência de execução. Ou seja, é necessário programar a construção, assim como é necessário programar uma viagem ou a compra de um carro.

Se você vai fazer uma viagem, antes de “pegar a estrada”, você programa quanto tempo vai durar a viagem – um final de semana, uma semana, um mês? -, qual o meio de transporte – vai de carro, de ônibus ou de avião? -, onde vai se hospedar – em casa de parentes ou amigos, em um hotel, em uma pousada? -, que roupas vai levar e assim por diante.

Quando decide comprar um carro, ou qualquer outra coisa,

antes de sair até uma loja e efetuar a compra, muito provavelmente, você decide o modelo do carro que quer comprar e faz uma pesquisa de preços, depois, você vai a várias revendedoras, negocia preço e condições de pagamento, antes de tomar a decisão. Então, perceba, existe uma sequência lógica para tudo que se faz.

Absolutamente tudo!

Para quase todas as atividades, podemos ter o auxílio da tecnologia – equipamentos dos mais diversos tipos e usos. E esta, por sua vez, se desenvolve cada vez mais rápido. Porém, a tecnologia não se cria sozinha, ela precisa que alguém a desenvolva e a “ensine” como funcionar. É preciso ensinar aos equipamentos tecnológicos o que fazer e qual a sequência correta do que deve ser feito. Assim, o primeiro passo para ensinar uma máquina a “pensar” é a Lógica de Programação, a qual vai indicar a sequência correta para o desenvolvimento do programa. E, para o desenvolvimento do programa, é imprescindível montar seu algoritmo.

Bem, é isso que este livro vai lhe ensinar usando uma linguagem de programação simples e acessível, por meio de exemplos claros e precisos.

Eu não poderia terminar este prefácio sem falar de seus autores, a professora Joice Mendes e o professor Rafael Muniz. Além de toda a capacidade e conhecimento em programação, quero destacar suas habilidades e a compreensão humana e construtivista do papel da educação. A professora Joice e o professor Rafael têm uma atuação muito ativa na aplicação de métodos pedagógicos de vanguarda, fazendo uso de metodologias ativas de aprendizagem, as quais têm se refletido na formação de

cidadãs e cidadãos mais autônomos, críticos e independentes, verdadeiramente capacitados a atuar de maneira ativa e consciente e, por consequência, habilitados a construir um mundo melhor.

Com este livro, você não aprenderá apenas os conceitos básicos de programação, mas certamente adquirirá a percepção lógica em uma nova dimensão de tudo que acontece à sua volta.

Aproveite!!

Pedro Fantinatti*

Campinas, janeiro de 2022.

* Professor nas áreas de Informática e Eletroeletrônica do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo.

SOBRE OS AUTORES

Professora Mestra Joice Barbosa Mendes

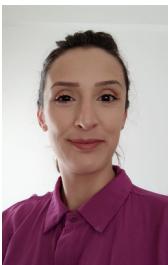


Figura 1: Professora Joice Mendes.

Experiência

Iniciou sua carreira como docente em 2012, atuando em disciplinas de programação do ensino superior para cursos de engenharia e Sistemas de Informação. Em 2016 passou a atuar em disciplinas de lógica de programação e orientação a objetos no IFSP, para cursos técnicos e de graduação.

Formação

Bacharelada em Ciência da Computação (UNIFEI-MG) em 2009. Concluiu o Mestrado em Ciência e Tecnologia da Computação (UNIFEI-MG) em 2012, com foco em Sistemas Tutores Inteligentes. Possui Formação Pedagógica para Educação Profissional de Nível Médio (IFSP-SP).

Professor Mestre Rafael da Silva Muniz



Figura 2: Professor Rafael Muniz.

Experiência

Possui mais de 10 anos de experiência na área de programação e já trabalhou como analista desenvolvedor nas empresas Petrobras, Casa da Moeda, IBGE e UERJ. Em 2016 iniciou sua carreira de professor no IFSP onde atua desde então ministrando as cadeiras relacionadas à lógica de programação e algoritmos para os cursos técnicos e de graduação. Já ministrou esse conteúdo de lógica de programação com Portugol para mais de 500 alunos.

Formação

Bacharelado em informática com ênfase em Análise de Sistemas (UNESA-RJ) em 2005. Possui três especializações na área de Engenharia de Software (SENAC-RJ), Gestão de Tecnologia da Informação e Comunicação (UCAM-RJ) e Análise e Projeto de Sistemas (PUC-RJ), e diversos cursos de curta duração na área de programação. Em 2018 concluiu o Mestrado em Educação (UNISAL-SP) para aprofundar os seus conhecimentos também na área da educação.

Sumário

Iniciando no mundo da programação	1
1 Introdução	2
1.1 Exercícios	13
2 Conhecendo as ferramentas para o Portugol	14
2.1 Estrutura padrão dos algoritmos	17
2.2 Como utilizar o Portugol Studio	19
2.3 Instalação do Portugol Studio versão Desktop	20
2.4 Como rodar nosso primeiro algoritmo	22
2.5 Exercícios	25
Comandos Portugol Studio	26
3 Comando escreva()	27
3.1 Exercícios	32
4 Variáveis	34
4.1 Declaração das variáveis	36

Sumário	Casa do Código
4.2 Tipos de variáveis	37
4.3 Nome de variáveis	38
4.4 Atribuição de valores nas variáveis	43
5 Operadores e expressões aritméticas	51
6 Comando leia()	56
6.1 Exercícios	60
Estruturas condicionais	62
7 Condicional	63
7.1 Operadores e expressões relacionais	64
7.2 Condicional simples: se()	66
7.3 Condicional composta: se senao	69
7.4 Exercícios	75
8 Condicional múltipla: se senao se	78
8.1 Exercícios	88
9 Operadores lógicos	91
9.1 TABELA VERDADE	93
9.2 Uso de parênteses	99
9.3 Exercícios	101
10 Condicional: escolha caso	104
10.1 Exercícios	107
Estruturas de repetição	110

11 Estrutura de repetição: para	111
11.1 Comando para	113
11.2 Exercícios	125
12 Estrutura de repetição: enquanto	127
12.1 Comando enquanto	128
12.2 Exercícios	134
13 Estrutura de repetição: faça enquanto	136
13.1 Comando faca enquanto	136
13.2 Exercícios	143
Vetor e Matriz	146
14 Vetor	147
14.1 Exercícios	156
15 Matriz	161
15.1 Usando biblioteca para arredondar as casas decimais	171
15.2 Exercícios	173
Função	175
16 Função	176
16.1 Função sem retorno	177
16.2 Função com retorno	183
16.3 Exercícios	185
17 Função - Passagem de parâmetro por valor	187

Sumário	Casa do Código
17.1 Função com passagem de parâmetro por valor	190
17.2 Exercícios	198
18 Função - Passagem de parâmetro por referência	200
18.1 Exercícios	208
19 Parabéns	210
Resolução dos exercícios	212
20 Resolução dos exercícios	213
20.1 Capítulo 1	213
20.2 Capítulo 2	214
20.3 Capítulo 3	214
20.4 Capítulo 4	214
20.5 Capítulo 5	215
20.6 Capítulo 6	216
20.7 Capítulo 7	217
20.8 Capítulo 8	220
20.9 Capítulo 9	222
20.10 Capítulo 10	225
20.11 Capítulo 11	227
20.12 Capítulo 12	230
20.13 Capítulo 13	232
20.14 Capítulo 14	233
20.15 Capítulo 15	235
20.16 Capítulo 16	239
20.17 Capítulo 17	241

Casa do Código	Sumário
20.18 Capítulo 18	243
21 Resolução do projeto de fixação	246
21.1 Capítulo 3	246
21.2 Capítulo 4	246
21.3 Capítulo 6	247
21.4 Capítulo 8	247
21.5 Capítulo 9	248
21.6 Capítulo 10	249
21.7 Capítulo 11	251
21.8 Capítulo 12	254
21.9 Capítulo 14	256
21.10 Capítulo 16	262
21.11 Capítulo 17	268
22 Referências	275

Versão: 26.5.18

Iniciando no mundo da programação



Figura 1: Fonte: Imagem de Gerd Altmann por Pixabay.

Nestes capítulos, estudaremos os conceitos de lógica, lógica de programação e algoritmos, e conheceremos a ferramenta Portugol Studio.

CAPÍTULO 1

INTRODUÇÃO

O mundo atual está em constante evolução e a tecnologia está cada vez mais presente nas atividades e processos realizados no cotidiano. Equipamentos como celulares, computadores, tablets e outros dispositivos eletrônicos conectam diversas pessoas ao longo dos seus dias.

Não nos surpreendemos mais ao encontrar aplicativos capazes de executar funcionalidades excepcionais, realizando operações que há alguns anos não imaginávamos. Até mesmo os eletrodomésticos estão dotados de certa inteligência, sendo possível programá-los para executar suas tarefas sem a intervenção de um humano.

Podemos ter o café pronto ao acordar, as roupas limpas em horário definido na máquina de lavar, até mesmo trancar ou destrancar as portas de casa sem ao menos estarmos por perto.



Figura 1.1: Exemplos de eletrodomésticos inteligentes. Imagem de Gerd Altmann por Pixabay.

Todas essas evoluções demonstram o poder e o alcance da tecnologia. Todos os dias surgem novos equipamentos eletrônicos, novos sistemas, novos sites, novas redes sociais e novos aplicativos de celular que facilitam a vida e, ao mesmo tempo, se tornam novas ferramentas no dia a dia das pessoas. No entanto, essas ferramentas só funcionarão a contento se estiverem programadas para realizar suas atividades de forma correta, na sequência correta e evitando possíveis erros.

A especificação dessas sequências só é possibilitada através dos *algoritmos*, que definem os passos para tratar os diversos cenários de utilização de um equipamento ou aplicativo, traduzindo para a máquina o que deve ser feito.

É nesse contexto que queremos apresentar o que são algoritmos, como construí-los e o que é necessário para desenvolvê-los com qualidade e eficiência.

Para tanto, antes de entendermos o que é o algoritmo e a lógica de programação, precisamos entender que todo computador é, basicamente, dividido em duas partes: *hardware* e *software*.

O *hardware* é a parte física do computador, ou seja, é aquilo que conseguimos pegar, tocar e/ou manusear fisicamente. Como exemplo de hardwares temos o mouse, o teclado, o gabinete, a placa de vídeo, a memória, a impressora, o *pen-drive*, entre outros. A próxima figura ilustra alguns hardwares.



Figura 1.2: Exemplos de hardwares. Fonte: pixabay.com / br.freepik.com

Já o *software* é a parte lógica do computador (programas, sistemas, apps, entre outros). Eles são responsáveis por controlar os hardwares. Alguns exemplos de softwares são o sistema operacional *Windows* ou *Linux*, o editor de texto *Word*, o navegador *Google Chrome*, o aplicativo de conversa pelo celular *WhatsApp*, entre outros. A figura a seguir ilustra alguns exemplos de softwares conhecidos atualmente.



Figura 1.3: Exemplos de software. Fonte: pixabay.com / br.freepik.com

Agora que já sabemos que o computador é dividido em duas grandes partes, estudaremos *lógica* e *lógica de programação*, que são os primeiros degraus do aprendizado dos *algoritmos*.

Lógica e lógica de programação

A *lógica* está presente no cotidiano de todas as pessoas, por mais que não tomemos conhecimento. Quando realizamos atividades corriqueiras como tomar banho, ir ao cinema, jogar videogame, acessar redes sociais, entre outras, utilizamos a lógica para pensarmos e organizarmos a *ordem do que vamos fazer e como vamos fazer*.

A lógica nos mostra a maneira correta de fazer as “coisas”.

Vamos pensar juntos: pegue um papel e anote os passos e a ordem dos passos para você: enviar um e-mail; depois para jogar on-line; e, por último, para escovar os dentes.

Repare que nos exemplos anteriores existem sequências de passos que devem ser "lógicas" para que você consiga realizar cada uma das atividades com sucesso.

Nos passos para escovar os dentes, por exemplo, primeiro você precisa colocar a pasta na escova para depois esfregar os dentes. Não vai adiantar você esfregar os dentes sem a pasta e depois passar a pasta na escova. Perceba que existe uma sequência lógica para execução dessa atividade.

O termo "lógica" nasceu como um campo da Filosofia e é conhecida desde os tempos de Aristóteles como a forma correta do pensar.

Definição formal

“A lógica está relacionada ao pensamento racional e ordenado [...]. É a arte do bem pensar.” (SOFFNER, 2017, p. 18).

“Modo pelo qual se encadeiam naturalmente as coisas e acontecimentos” (Dicionário Michaelis).

Nos computadores, o objetivo da lógica é pensar a maneira correta de escrever um algoritmo ou um programa, por isso chamamos de *Lógica de Programação*.

Definição formal

“Maneira pela qual instruções [...] são organizadas num algoritmo para viabilizar a implantação de um programa.” (Dicionário Michaelis).

Algoritmo

Agora que já sabemos o que é a lógica de programação, vamos descobrir o que são os algoritmos (termo técnico que tem se tornado familiar nos últimos anos devido ao crescimento da internet e dos aplicativos de celular).

Vamos pensar juntos: analisando a figura a seguir, o que

conseguimos entender como sendo o algoritmo?



Figura 1.4: Representação gráfica do que é algoritmo. Fonte: autores.

Primeiramente percebemos um quebra-cabeça desmontado no início da imagem (com a palavra **problema** abaixo) e no final da imagem temos o quebra-cabeça montado (com a palavra **solução** abaixo). Entre as duas imagens temos como elemento central a palavra algoritmo. Ou seja, o algoritmo está sendo utilizado como uma ponte entre o problema e a solução. Ele então descreve os passos necessários para que a solução de um problema seja encontrada.

Portanto, podemos perceber que ocorreu uma ação ou um conjunto de ações para que o quebra-cabeça saísse do estado desmontado até chegar ao estado montado. Essas ações podem ser detalhadas em pequenos passos, um seguido do outro. A esse conjunto de passos daremos o nome de **algoritmo**. Na computação, esses problemas podem ser, por exemplo, a criação de um sistema para controlar entrada e saída de carros de um estacionamento, um sistema para calcular a média dos alunos de

uma sala, um sistema para enviar mensagem via celular, um sistema de e-mail, um jogo on-line, entre outros.

Dessa maneira, definiremos um **ALGORITMO** como sendo “*sequência de passos ORDENADOS e finitos para a solução de um problema*”.

Percebiam que, na definição, surgiram as palavras *ordenados* e *finitos*. Essas palavras surgiram, pois veremos que todos os algoritmos precisam ser escritos de forma ordenada, ou seja, que obedeça a uma ordem adequada para que solucione o problema corretamente. O algoritmo precisa ser escrito dentro de uma lógica computacional. Já a palavra **finito** apareceu pois todo algoritmo precisará ter um início e um fim determinado, caso contrário, nunca terminaremos de escrevê-lo e ele nunca será executado pelo computador.

Definição formal

“Algoritmo é um conjunto de passos, passível de repetição, que resolve um problema.” (SOFFNER, 2017, p. 21).

Vamos pensar juntos: pegue um papel e anote os passos e a ordem dos passos para você trancar a porta ao sair de casa.

Uma sequência de passos poderia ser: pegar a chave correta; puxar a maçaneta para fechar a porta; inserir a chave na fechadura; girar a chave para trancá-la; retirar a chave da fechadura. Essa sequência de passos está ordenada e demonstra a sequência correta para realizarmos o trancamento. Se girarmos a chave na fechadura antes de fechar a porta, a porta não será trancada corretamente.

Repare que o algoritmo do nosso exemplo apresentou uma sequência de passos ordenados e finitos. Assim como na lógica, o nosso dia a dia é repleto de algoritmos. Da hora em que acordamos até a hora em que vamos dormir executamos diversos passos para resolvermos problemas, como escovar os dentes, almoçar, pegar um transporte público, ir para a escola, trocar mensagens pelo celular, jogar videogame, entre outros. Já os problemas computacionais podem ser acessar um e-mail, enviar uma mensagem de texto para um amigo, acessar a conta bancária pelo celular, realizar o login em um jogo on-line, entre outros.

Vamos pensar juntos: pegue um papel e anote os passos e a ordem dos passos para você mandar uma mensagem para uma outra pessoa usando um aplicativo de comunicação pelo celular.

Depois, peça para uma pessoa amiga ou familiar dizer quais são os passos necessários para que ela envie uma mensagem. Compare a quantidade de passos que você montou com a que a outra pessoa montou. Reparou alguma diferença? Os dois algoritmos foram idênticos?

Como exemplo, vamos listar dois possíveis algoritmos para enviar mensagem por aplicativo de comunicação pelo celular,

aproveite e compare com o que você escreveu.

Possíveis algoritmos para o exemplo anterior.

Algoritmo 1: detalhado	Algoritmo 2: simplificado
1. Abrir o aplicativo de comunicação no celular.	1. Procurar a pessoa que deve receber a mensagem.
2. Procurar a pessoa que deve receber a mensagem.	2. Digitar a mensagem.
3. Clicar em cima do nome dessa pessoa.	3. Enviar a mensagem.
4. Digitar a mensagem.	-
5. Clicar no ícone enviar.	-

Repare que temos duas representações dos passos necessários para realização do mesmo objetivo: mandarmos uma mensagem para uma outra pessoa usando um aplicativo de comunicação pelo celular. As duas representações resolvem o problema? Qual é a diferença entre elas? Qual é a correta?

Para respondermos às perguntas anteriores precisamos entender que um algoritmo pode ter mais de uma sequência de passos para solucionar um mesmo problema. Ou seja, para resolvermos um mesmo problema, podemos percorrer vários caminhos diferentes, só não podemos esquecer do objetivo principal que é solucionar o problema.

Conforme apresentado no algoritmo anterior, a sequência de passos pode variar devido a inúmeras particularidades. Continuando o exemplo do envio de mensagem por aplicativo, imagine que o celular estivesse sem bateria, os passos listados anteriormente resolveriam o problema? E se o celular não estivesse com conexão à rede de dados móveis ou ao wi-fi, o problema de

enviar mensagem seria resolvido com os passos listados anteriormente?

A resposta é "não". Pois, se o celular estivesse sem bateria, teríamos que conectá-lo no carregador antes, depois ligar o celular e aí sim seguir os passos listados no algoritmo. Caso ele estivesse com bateria mas sem rede de dados móveis ou wi-fi, teríamos que procurar um lugar com sinal para depois seguir os passos.

Repare que os passos do algoritmo mudam de acordo com inúmeros itens e eles ainda vão mudar dependendo da lógica de cada programador ou programadora, já que a lógica envolve o "pensar individual" e cada pessoa pensa de uma maneira particular. Dessa forma, veremos que o mesmo problema pode ter várias "soluções" possíveis.

O importante é nos concentrarmos na nossa forma de pensar para resolvermos os problemas.

Dito isso, podemos responder que os dois algoritmos no exemplo anterior resolvem o problema, só que um tem uma solução mais simplificada e o outro, mais detalhada. Essa variedade de soluções também vai ocorrer nos algoritmos computacionais, ou seja, para um mesmo problema computacional podemos ter diversos algoritmos.

Por isso, no início, não se preocupe e nem compare a sua lógica com a lógica do seu amigo ou amiga. Você perceberá que com a prática a sua lógica será "lapidada" para atender de maneira mais otimizada às soluções dos problemas.

Formas de representar um algoritmo

Nesta seção, veremos as três maneiras mais conhecidas para representarmos um algoritmo.

Três formas de representar um algoritmo. Ex: somar 2 + 6		
• Narrativo	Fluxograma	Pseudocódigo
<ul style="list-style-type: none">• Pegar o número 2• Pegar o número 6• Somar os dois números• Apresentar o resultado	<pre>graph TD; A[num1=2, num2=6] --> B[resul=num1+num2]; B --> C[“Resultado”, resul]</pre>	<pre>programa{ funcao inicio(){ inteiro num1, num2, resul num1 = 2 num2 = 6 resul = num1 + num2 escreva("Resultado",resul) } }</pre>

Figura 1.5: Três formas de representar os algoritmos. Fonte: autores.

Existem basicamente três maneiras de representarmos qualquer algoritmo. Conforme imagem anterior, a primeira representação é através de texto narrativo ou descrição narrativa. Nesse formato, os passos do algoritmo são descritos de forma textual (como nos exemplos que fizemos juntos anteriormente). A segunda é utilizando gráficos onde cada passo do algoritmo é representado através de um símbolo específico. Essa forma de representação também é chamada de fluxograma. A terceira é através das chamadas *pseudolínguagens* ou *linguagens de programação*. Na imagem anterior, temos um algoritmo escrito na linguagem **Portugol** utilizando a ferramenta **Portugol Studio**. Calma, não se preocupe, que nos próximos capítulos veremos cada linha de forma bem detalhada.

Neste livro, abordaremos apenas a terceira forma de representação dos algoritmos e, para isso, utilizaremos a pseudolínguagem chamada **Portugol**.

Resumo do capítulo: Introdução

Vimos os conceitos básicos necessários para iniciarmos no mundo da programação, diferenciando inicialmente *hardware* e *software*. Conceituamos que a lógica é a maneira de pensarmos corretamente, que algoritmo é uma sequência de passos ordenados e finitos para a solução de um problema e vimos, por último, que existem três maneiras de representarmos um algoritmo: descrição narrativa, forma gráfica e pseudolínguagem.

1.1 EXERCÍCIOS

1. (F) Monte o algoritmo narrativo (passo a passo) da sua rotina diária na parte da manhã.
2. (F) Monte um algoritmo narrativo resumido e um outro detalhado de como acessar um site na internet.

Legenda: F - fácil / M - médio / T - trabalhoso

Vídeo com o resumo do capítulo

<https://logicaportugol.com.br/cap1>



Figura 1.6: QRCode

CAPÍTULO 2

CONHECENDO AS FERRAMENTAS PARA O PORTUGOL

Conforme apresentado no capítulo anterior, todo algoritmo é uma sequência de passos **finitos** e **ordenados** descritos para a solução de um problema. Esses passos devem seguir um idioma que o computador entenda. Ou seja, para solicitarmos que o computador "*faça alguma coisa*" precisamos utilizar o idioma que ele conhece. A expressão "*que o computador faça alguma coisa*" será substituída a partir de agora pelos termos técnicos "*execute algum comando ou instrução*".

COMANDO (INSTRUÇÃO): são as palavras *do idioma* do computador, ou seja, são os termos que ele entende e que, ao receber, ele saberá o que fazer. Exemplo: O botão imprimir, que é utilizado em diversos programas, como editores de textos, planilhas, entre outros, serve para que o computador mande o conteúdo exibido na tela para a impressora configurada.

EXECUTAR: rodar; processar; montar o programa para que o usuário possa usá-lo.

Neste livro, utilizaremos como idioma, para conversarmos com o computador, a pseudolínguagem chamada de **PORTUGOL**. Trata-se de uma pseudolínguagem pois só é utilizada para fins didáticos. Não encontraremos o Portugol em empresas, indústrias e nos aplicativos que usamos no nosso dia a dia.

Mesmo sendo uma pseudolínguagem, o Portugol também possui suas regras de escrita (que são chamadas tecnicamente de *sintaxe*) e que precisam ser seguidas para que os algoritmos funcionem corretamente. Aqui usaremos as sintaxes definidas pela ferramenta chamada Portugol Studio. Ou seja, todos os nossos algoritmos deverão ter seus comandos escritos de acordo com a padronização definida por essa ferramenta (IDE).

FERRAMENTA IDE: programa de computador que entende e fornece aos programadores facilidades para desenvolverem seus algoritmos/sistemas/programas. Realiza as verificações da sintaxe da linguagem ou da pseudolínguagem. O acrônimo IDE vem do inglês "*Integrated Development Environment*", ou *Ambiente de Desenvolvimento Integrado*, em português.

Optamos por utilizar o Portugol Studio pois é “[...] um ambiente para aprender a programar, voltado para os iniciantes

em programação que falam o idioma português.” (LITE, 2020).

Apesar de especificar seus comandos com base na língua portuguesa, o Portugol Studio mantém sintaxes e formatos bem próximos das principais linguagens de programação empregadas atualmente no mundo do trabalho.

Além disso, a ferramenta é bastante estável, confiável e ainda é *open source* (possui código aberto e pode ser obtido de forma gratuita). Ela é desenvolvida e mantida pelo Laboratório de Inovação Tecnológica na Educação - LITE da Universidade do Vale do Itajaí - UNIVALI.

LINGUAGEM DE PROGRAMAÇÃO: conjunto de *palavras*, chamadas de palavras reservadas, que indicam ao computador qual comando deve ser realizado. As linguagens de programação podem ser de alto nível, quando não se preocupam com detalhes de manipulação do *hardware*; ou de baixo nível, quando manipulam elementos mais próximos do *hardware*.

SINTAXE DE UMA LINGUAGEM DE PROGRAMAÇÃO: é a forma como devemos escrever para que a linguagem entenda o que queremos fazer (VILARIM, 2017). Ou seja, é um padrão com as regras que o criador da linguagem de programação definiu. Similar à gramática empregada na língua portuguesa.

COMPILADOR: programa responsável por traduzir os comando dos nossos algoritmos para a linguagem de máquina (linguagem que o hardware entende). Os computadores só conhecem os números 0 e 1 (bits), porém nossos algoritmos são escritos usando palavras da língua portuguesa. É o compilador o responsável por fazer essa *conversão* dos comandos que escrevemos para uma sequência de 0 e 1, de modo que o computador então entenda o que ele deve fazer.

2.1 ESTRUTURA PADRÃO DOS ALGORITMOS

Agora que já sabemos o que é o Portugol e o que são os comandos, vamos começar a aprender como escrever um algoritmo seguindo a sintaxe especificada na ferramenta IDE Portugol Studio.

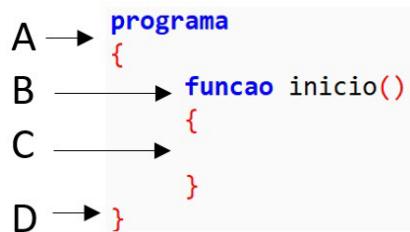


Figura 2.1: Estrutura padrão de todo algoritmo.

A ferramenta Portugol Studio define um conjunto de comandos que são necessários em todos os algoritmos que criarmos. A esse conjunto chamaremos de *estrutura padrão*. A estrutura padrão do Portugol Studio foi apresentada na figura.

Ela é composta pelos seguintes comandos:

- (A) apresenta o comando `programa {`. Esse comando serve para indicar o início da estrutura padrão.
- (B) Em seguida, aparece o comando `funcao inicio () {`. Ele representa uma função, item que veremos nos próximos capítulos e, por isso, não falaremos neste momento.
- (C) Temos o espaço entre a abertura e o fechamento das chaves do comando `funcao inicio`. É nesse espaço que escreveremos nossos algoritmos.
- (D) Fechamento das chaves.

Observações: (1) É importante percebermos que nos comandos `programa` e `funcao inicio` (item A e B) temos uma abertura de chave `{` no final da linha. Essa chave pode aparecer na mesma linha do comando ou na seguinte (funcionará da mesma maneira). (2) Sempre que abrirmos uma chave, os comandos que estiverem dentro da abertura dela até o fechamento devem ser escritos mais à direita. Repare que o comando `programa` abre uma chave e a letra `f` do comando `funcao inicio` aparece escrita mais à direita e não na mesma coluna que a letra `p` do comando `programa`. Essa regra é chamada de *indentação*, termo técnico difundido no mundo da programação.

Neste momento do aprendizado, temos que ter em mente que essa estrutura padrão é necessária para que os nossos algoritmos funcionem corretamente. Caso não seja seguida a sintaxe da estrutura padrão, o nosso algoritmo não será executado pelo compilador. Por este motivo, tenha bastante atenção ao criar um novo algoritmo.

2.2 COMO UTILIZAR O PORTUGOL STUDIO

Atualmente, pode-se utilizar a pseudolínguagem Portugol, baseada na ferramenta IDE Portugol Studio, de três maneiras diferentes: a primeira é a versão desktop, que será utilizada neste livro; a segunda é uma versão on-line chamada de Portugol WebStudio; e a terceira é uma versão mobile para aplicativos Android chamada de Portugol mobile.

1^a opção: versão desktop (versão oficial da ferramenta IDE Portugol Studio)

Desenvolvida pelo Laboratório LITE - Laboratório de Inovação Tecnológica na Educação da UNIVALI. Versão: Desktop.

- **Link de acesso:** <http://lite.acad.univali.br/portugol/>
- **Vantagem:** versão robusta, fácil de usar, opção de troca de tema, documentação muito bem detalhada, diversos exemplos que auxiliam no aprendizado, dicas.
- **Desvantagem:** necessidade de instalação (se você estiver utilizando o computador de outra pessoa ou que tenha alguma restrição na instalação de software, pode representar um problema).

2^a opção: versão on-line Portugol Webstudio

De acordo com informações obtidas no site da versão on-line, o Portugol Webstudio foi desenvolvido pelos usuários do GitHub @guiscaranse e @dgadelha, juntamente de outros colaboradores. Versão: On-line/Web.

- **Link de acesso:** <https://portugol-webstudio.cubos.io/>

- **Vantagem:** nenhum software precisa ser instalado no computador; basta ter acesso à internet para escrever os algoritmos; possui os conteúdos de ajuda, biblioteca e exemplos da ferramenta desktop.
- **Desvantagem:** necessidade de ter acesso à internet sempre disponível para executar os algoritmos.

3^a opção: Portugol mobile (versão para Android)

De acordo com informações obtidas na página do app no Google Play, o app foi inspirado na ferramenta Portugol Studio, porém ele foi escrito do zero. Ele não pertence à equipe da UNIVALI e sim ao desenvolvedor Erick Leonardo Weil. Versão: Mobile.

- **Link de acesso:** Google Play
- **Vantagem:** consegue rodar seus algoritmos em qualquer dispositivo móvel, com sistema Android, em qualquer lugar.
- **Desvantagem:** dificuldade de uso e visualização quando os códigos possuem muitas linhas, devido ao tamanho da tela do dispositivo móvel; necessita instalação.

2.3 INSTALAÇÃO DO PORTUGOL STUDIO VERSÃO DESKTOP

Neste livro, apresentaremos nossos exercícios e exemplos baseados na 1º opção (versão desktop). Para utilizá-la, basta seguir os passos a seguir:

1. Acesse o link <http://lite.acad.univali.br/portugol/> em algum

navegador.

2. Será apresentada a tela conforme figura a seguir.



Figura 2.2: Página inicial da ferramenta IDE Portugol Studio

3. Clique no botão download.
4. Escolha a pasta onde o arquivo será baixado e clique em salvar.
5. Aguarde o término do download.
6. Procure o arquivo baixado e dê um duplo clique para que seja iniciada a instalação da ferramenta em seu computador.
7. Em todas as telas da instalação, aparecerão os botões avançar e cancelar para continuar ou desistir da instalação, exceto a tela de licença de uso que precisa ser lida e ter a ciência marcada para continuar a instalação.
8. Após avançar todas as telas, a instalação é concluída.

Pronto! A ferramenta já foi instalada em seu computador.

Observação: Caso tenha dado algum erro ou você tenha ficado com alguma dúvida, entre em contato pelo e-mail livro@logicacomportugol.com.br

2.4 COMO RODAR NOSSO PRIMEIRO ALGORITMO

Para rodar nosso primeiro algoritmo precisamos dar um duplo clique no ícone da ferramenta.



Figura 2.3: Ícone para iniciar a ferramenta IDE Portugol Studio.

Ele abrirá a tela principal da ferramenta.

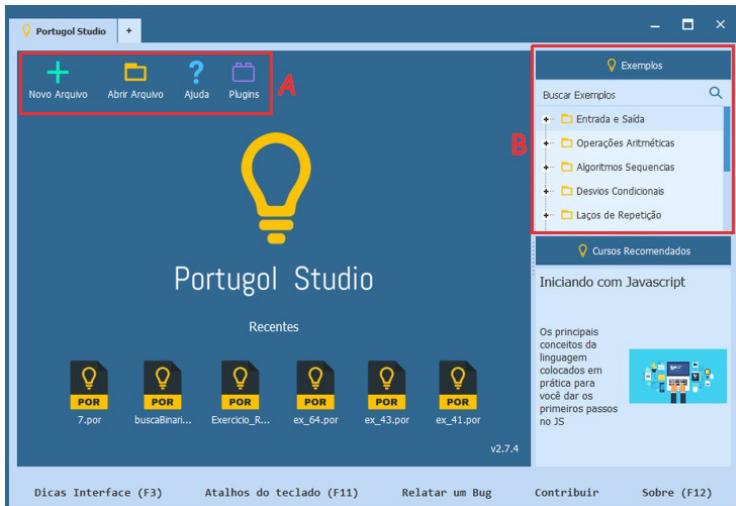


Figura 2.4: Página principal da ferramenta IDE Portugol Studio.

Na imagem anterior, temos duas partes importantes da ferramenta (A e B). Na parte (A), são exibidos os botões para criar uma nova área para digitar um algoritmo, para abrir um algoritmo já criado anteriormente, um botão de ajuda e outro para

importação de plugins. No item (B), são exibidos exemplos de algoritmos prontos para serem estudados e explorados.

Para começarmos, focaremos no primeiro botão da área (A) para criarmos uma nova área para digitarmos nosso algoritmo. Ele se chama *Novo Arquivo*. Ao clicar nele é aberta a seguinte tela:

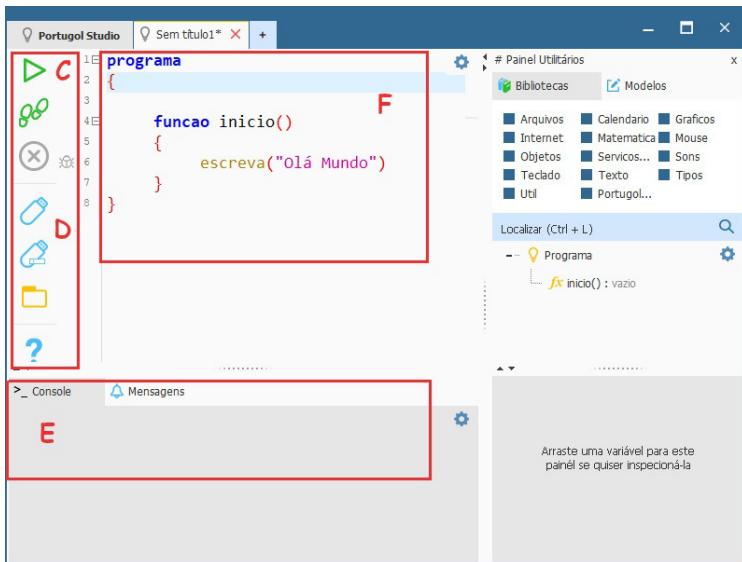


Figura 2.5: Tela para criação dos algoritmos.

No item (C), temos o botão para executarmos nossos algoritmos (rodarmos). Ao executarmos nossos algoritmos, todas as interações com o usuário serão realizadas na área chamada *Console* (E).

No item (D), temos a opção de salvar os nossos algoritmos. Na área (F), temos a tela onde escreveremos nossos algoritmos.

Podemos perceber que, ao criarmos um *novo arquivo*, a

ferramenta Portugol Studio já cria para nós a estrutura padrão e acrescenta uma linha com o comando `escreva("Olá Mundo")`. Não se preocupe neste momento, pois veremos logo no próximo capítulo como esse comando funciona.

Vamos aproveitar o algoritmo que a própria ferramenta criou para executar nosso primeiro algoritmo e verificar o que ele apresenta no console. Para isso, vamos clicar no botão de executar o algoritmo (conforme item C).

```
//Exemplo 1: algoritmo criado automaticamente pela ferramenta
programa
{
    funcao inicio()
    {
        escreva("Olá Mundo")
    }
}

//Saída no console
Olá Mundo
Programa finalizado. Tempo de execução: 22 milissegundos
```

Como dito anteriormente, toda interação do algoritmo com o usuário ocorrerá no console (item E da figura anterior). Ao rodarmos o algoritmo, percebemos que ele apresentou no console a mensagem Olá Mundo e, na linha a seguir, a mensagem Programa finalizado. Tempo de execução: 22 milissegundos .

A mensagem Olá Mundo apareceu pois está no comando `escreva` (calma!!! falaremos a seguir) e a mensagem Programa finalizado e o tempo de execução aparecerá sempre que executarmos qualquer algoritmo. Ela é uma mensagem da própria ferramenta para indicar que o programa foi executado e finalizado, e para sabermos quanto tempo foi para isso.

Esse tempo de execução varia de acordo com o tamanho do nosso algoritmo, bem como as estruturas lógicas que foram utilizadas. Nesse momento, do aprendizado não precisamos nos preocuparmos com essa informação.

Resumo do capítulo

Vimos que há três maneiras de usar o Portugol Studio (versão desktop, versão on-line e mobile). Para cada uma delas foram apresentadas as vantagens e desvantagens. Em seguida, vimos as telas principais da versão desktop (versão que será utilizada neste livro), a estrutura padrão de um programa em Portugol, os botões utilitários, as áreas de interação e executamos nosso primeiro algoritmo.

2.5 EXERCÍCIOS

3. (F) Instale a ferramenta Portugol Studio em seu computador ou utilize a forma web ou app, para executar o algoritmo com o texto *Olá mundo* (igual ao algoritmo anterior) e verifique a saída do algoritmo no console da ferramenta.

Legenda: F - fácil / M - médio / T - trabalhoso

Vídeo com o resumo do capítulo

<https://logicaportugol.com.br/cap2>



Figura 2.6: QRCode

Comandos Portugol Studio



Figura 1: Fonte: Imagem de Free-Photos por Pixabay.

Nestes capítulos, estudaremos os comandos para escrever na tela do computador, para ler os dados do teclado, para armazenar valores nos nossos algoritmos e realizar operações matemáticas.

CAPÍTULO 3

COMANDO ESCREVA()

Vamos estudar inicialmente o comando `escreva()`. Ele serve para escrevermos textos na tela do computador. No nosso caso, a tela do computador é representada pelo console da ferramenta Portugol Studio.

Vamos pensar juntos: analisando o algoritmo apresentado no exemplo 2 a seguir, com o comando `escreva()`, o que você acha que ele faz?

```
//Exemplo 2: código com o comando escreva
programa
{
    função inicio()
    {
        escreva("Casa do Código")
    }
}
```

Se lembrarmos do primeiro algoritmo que executamos, ele também tinha um comando `escreva()` e, ao executá-lo, foi exibido o seu texto no console. O mesmo acontecerá com o algoritmo acima. Como ele tem um comando `escreva()` com o texto `Casa do Código`, esse texto será exibido no console.

```
//Saída no console
```

Casa do Código

Perceba que o comando tem como sintaxe a palavra `escreva` e em seguida a abertura e fechamento dos parênteses. Dentro dos parênteses () aparece um texto dentro de um par de aspas duplas " ". Já no console é exibido apenas o texto que está dentro das aspas duplas do comando `escreva()`.

SINTAXE DO COMANDO ESCREVA

```
escreva("Texto")
```

Como é que o compilador executa o comando `escreva()`? Primeiramente ele verifica se a sintaxe do comando foi seguida e, se estiver de acordo, ele apresentará, no console, somente o texto que está dentro do par de aspas duplas ("").

O texto que será exibido variará de acordo com a necessidade do seu algoritmo. No nosso exemplo anterior, colocamos o texto *Casa do Código*, porém poderíamos colocar nosso nome, uma mensagem de bom-dia, de boa-tarde, entre outros.

Veja a diferença do comando anterior com a escrita do próximo comando e tente identificar o que está faltando na sintaxe do `escreva()`.

```
//Exemplo 3: código com erro na sintaxe do comando escreva
programa{
    funcao inicio(){
        escreva("Casa do código"
    }
}
```

Essa foi fácil, né? Ficou faltando o fechamento do parêntese depois das aspas duplas. Ao executarmos esse código não será exibido nenhum texto no console e sim uma mensagem de erro na aba **Mensagens**, conforme figura a seguir.

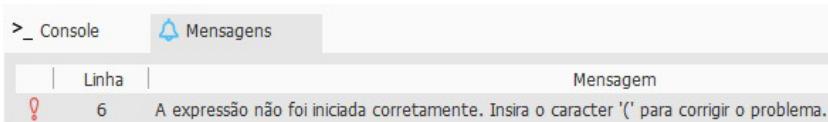


Figura 3.1: Mensagem de erro apresentada pela ferramenta.

Portanto, se os nossos algoritmos não estiverem de acordo com a sintaxe, o compilador não o executará e apresentará um erro chamado de **erro de compilação**.

Observação: o comando `escreva()` sempre deve ser escrito dentro da função `inicio()` na estrutura padrão. Porém, o exemplo anterior foi o último exemplo de algoritmo que apresentaremos com a estrutura padrão. Ocultaremos a estrutura para que os nossos algoritmos fiquem menos poluídos e assim nos concentraremos mais nos novos comandos. Logo, caso queira reproduzir os códigos apresentados a seguir, lembre-se de incluí-los na estrutura padrão.

Todos os algoritmos podem ter vários comandos `escreva()`, não existe limite para utilização do comando. Porém, existem algumas particularidades ao utilizarmos o comando.

Vamos pensar juntos: observe o código a seguir e imagine como será a saída no console.

```
//Exemplo 4: código com o comando escreva  
escreva("Tabela do Campeonato Brasileiro 2020")  
escreva("1 Flamengo 71")  
escreva("2 Internacional 70")  
escreva("3 Atlético-MG 68")
```

Agora veja como será a exibição no console.

```
//Saída no console  
Tabela do Campeonato Brasileiro 20201Flamengo 712 Internacional 7  
03 Atlético-MG 68
```

Repare que todo o texto apareceu na mesma linha. Isso acontece porque o computador não entende a quebra de linha. Para indicar que queremos que um texto seja exibido na próxima linha, mesmo estando em comandos `escreva()` diferentes, devemos indicar explicitamente nos nossos algoritmos. Essa indicação ocorre através do uso dos caracteres `\n`, incluídos dentro das aspas do comando `escreva()`. Eles devem ser inseridos nos locais em que desejamos que seja incluída uma quebra de linha. Veja o exemplo a seguir:

```
//Exemplo 5: código com o \n  
escreva("Tabela do Campeonato Brasileiro 2020 \n")  
escreva("1 Flamengo 71 \n")  
escreva("2 Internacional 70 \n")  
escreva("3 Atlético-MG 68 \n")
```

Repare o console depois da utilização do `\n`.

```
//Saída no console  
Tabela do Campeonato Brasileiro 2020  
1 Flamengo 71  
2 Internacional 70  
3 Atlético-MG 68
```

Observação: Podemos utilizar o `\n` quantas vezes for necessário em nossos algoritmos, ele só precisa estar entre as aspas duplas.

Além do `\n` temos o `\t`, que indica uma tabulação (textos alinhados). A utilização de um `\t` acrescenta apenas uma tabulação. Caso sejam necessárias mais tabulações, devemos utilizar quantos `\t` precisarmos. Veja o exemplo a seguir.

```
//Exemplo 6: código com \n e \t
escreva("Tabela do Campeonato Brasileiro 2020\n")
escreva("1 Flamengo \t\t71\n")
escreva("2 Internacional \t70\n")
escreva("3 Atlético-MG \t\t68\n")

//Saída no Console
Tabela do Campeonato Brasileiro 2020
1 Flamengo          71
2 Internacional      70
3 Atlético-MG        68
```

Rpare que as pontuações dos times aparecem alinhadas. Para cada time tivemos que adicionar um número específico de `\t` para que ficassem alinhadas. Isso ocorre pela diferença de tamanho dos nomes dos times.

Bom, dessa maneira já vimos o comando `escreva()` e já podemos utilizá-lo nos nossos algoritmos. Existem outras variações do comando que veremos nos próximos capítulos.

Vamos pensar juntos: monte um algoritmo que apresente na tela o texto $10 + 5 = 15$.

```
//Exemplo 7: código com texto fixo
escreva("10 + 5 = 15")

//Saída no console
10 + 5 = 15
```

Assim como vimos até agora, o que aparece dentro das aspas

duplas é apresentado no console. O compilador faz como se fosse um "copia" e "cola" do texto. No exemplo anterior, o texto exibido foi $10 + 5 = 15$. Mesmo representando uma operação matemática, o compilador o trata como texto. Para testarmos, vamos alterar o texto para $10 + 5 = 1$.

```
//Exemplo 8: código com erro no texto  
escreva("10 + 5 = 1")  
  
//Saída no console  
10 + 5 = 1
```

Repare que no console a soma é apresentada com erro pois, conforme visto, o texto é copiado e colado, e a operação matemática não é realizada.

Para passarmos a responsabilidade para que o compilador realize o cálculo no nosso algoritmo, precisamos nos aprofundar um pouco mais e conhecer as **variáveis**.

Resumo do capítulo

Vimos neste capítulo qual o comando que utilizamos para apresentar um texto na tela do computador. No nosso caso, os textos sempre serão exibidos no console da ferramenta. O comando que realiza essa escrita é o `escreva("")`. Vimos também que o texto que será exibido deve estar dentro do par de aspas duplas. Em seguida, vimos que para pularmos de linha utilizamos os caracteres `\n`. E, por último, vimos que podemos alinhar os textos utilizando os caracteres `\t`.

3.1 EXERCÍCIOS

4. (F) Crie um algoritmo que apresente na tela a mensagem "Sejam bem-vindos ao livro sobre Algoritmos da Casa do Código".

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Para fixação dos conteúdos apresentados, além dos exercícios de cada capítulo desenvolveremos um projeto que será incrementado a cada novo conteúdo. Para isso, imagine que o dono da loja XPTO Bikes - Venda e Manutenção de Bicicletas, chamado Pedro, contratou você para desenvolver um sistema de autoatendimento. O cliente poderá pesquisar produtos e serviços assim que chegar em sua loja.

Nessa primeira fase do projeto, Pedro pediu que o sistema mostrasse a mensagem de boas-vindas aos seus clientes. Você deve criar um algoritmo para apresentar a seguinte mensagem para os clientes: "Sejam bem-vindos à loja XPTO Bikes! Em breve teremos um sistema de autoatendimento".

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap3>



Figura 3.2: QRCode

CAPÍTULO 4

VARIÁVEIS

Agora que já sabemos como escrever na tela do computador, precisamos saber como o computador armazenará os dados que serão utilizados nos nossos algoritmos.

Vamos pensar juntos: decore o seguinte número 13.21.34 (não pode anotar no papel).

Pense no número agora.

Já que não poderia anotar no papel, você teve que armazenar o número na sua memória e em seguida voltou à memória para recuperá-lo. Assim também funciona o computador. Vamos utilizar a memória dele para armazenar números e/ou textos que são necessários nos nossos algoritmos. Cada número e/ou texto ocupará uma área da memória, dentre as milhares de áreas disponíveis. Cada uma dessas áreas é chamada de **variável** no nosso algoritmo.

Essas **variáveis** podem ser de vários tipos diferentes e podem ter o valor armazenado alterado ao longo da execução do algoritmo. É como se, no exemplo dos números que você teve que decorar, mudássemos depois de um tempo e você tivesse que

decorar o novo número no lugar do antigo.

Um item bastante importante de lembrarmos é que os algoritmos são sequências de comandos ordenados (passos) e eles são executados de forma sequencial. Todos os comandos escritos no algoritmo são executados, um após o outro, e o responsável pela execução dos algoritmos é o **compilador**. É importante relembrarmos esses conceitos pois eles serão utilizados a seguir.

Voltando às variáveis, falamos então que elas servem para armazenar valores ao longo do algoritmo e que podem ter diversos tipos. Complicado ainda, né? Vamos ver o seguinte exemplo: imagine que no nosso algoritmo precisamos armazenar a idade e o salário de um funcionário. Para isso, precisaremos de dois espaços de memória para armazenar esses dois valores. Criamos então uma variável chamada `idade` e uma outra chamada `salario`. Se repararmos na figura a seguir, perceberemos que duas caixinhas da memória do computador foram ocupadas e cada uma recebeu o nome da variável.

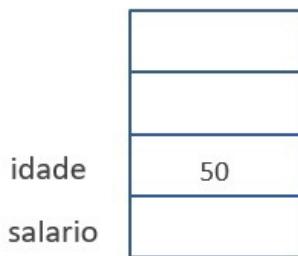


Figura 4.1: Exemplo didático de duas variáveis na memória do computador.

Uma regra bastante importante é que para utilizarmos qualquer variável nos nossos algoritmos precisamos criá-la

inicialmente. Essa criação é realizada através da **declaração das variáveis**.

4.1 DECLARAÇÃO DAS VARIÁVEIS

Antes de usarmos qualquer variável, precisamos criá-la. Essa criação é necessária para que o compilador indique ao computador que o nosso algoritmo necessita de um espaço de memória para armazenar um valor.

```
//Exemplo 9: declaração de variáveis
inteiro idade
real salario
cadeia nome
caracter letra
logico programa
inteiro quantidadeProduto
inteiro numeroDeAlunos
```

Vamos pensar juntos: olhando a declaração das variáveis no exemplo anterior, você consegue perceber alguma similaridade nas declarações?

Perceba que todas as declarações de variáveis são compostas de duas partes. A primeira parte apresenta o **tipo do dado** e a segunda parte é o **nome da variável**. O tipo da variável serve para indicar ao compilador qual é o formato do valor que será armazenado na variável. Já o nome da variável serve para diferenciar as diversas variáveis do nosso algoritmo, distinguindo umas das outras.

SINTAXE DA DECLARAÇÃO DAS VARIÁVEIS

tipo nome

Nela:

tipo: pode ser inteiro, real, caractere, cadeia ou lógico

nome: identificação da variável

4.2 TIPOS DE VARIÁVEIS

Como visto anteriormente, a declaração da variável deve especificar o tipo do dado que será armazenado no espaço reservado na memória. Para tanto, precisamos conhecer os tipos de dados disponíveis para serem utilizados no Portugol.

Na ferramenta Portugol Studio, existem cinco tipos diferentes de variáveis disponíveis: `inteiro` , `real` , `caracter` , `logico` e a `cadeia` .

Descrição	Tipo
Armazena os números inteiros. Ou seja, os números que não apresentam casas decimais. Eles podem ser negativos, neutros ou positivos. Exemplos: 5, -87, 2020, 50, 12, 0, -4, 90.	<code>inteiro</code>
Armazena os números com casas decimais. Ou seja, os números que utilizam vírgulas para separar a parte inteira do numeral e as casas decimais. Exemplos: 45.90, 3.1415, 1.68, 65.50, -34.56, -9.00.	<code>real</code>
Armazena apenas um único caractere alfanumérico ou um único caractere especial. Podem ser armazenados números, porém eles não têm valor numérico - representam apenas um símbolo e não poderão ser utilizados para operações matemáticas como soma ou subtração. Além disso, só podem possuir um algarismo. Ao utilizar o tipo	<code>caracter</code>

	caractere, o símbolo sempre ficará entre aspas simples ‘’. Exemplos: ‘a’, ‘D’, ‘γ’, ‘9’, ‘5’, ‘#’, ‘%’.
logico	Armazenam somente os valores “verdadeiro” ou “falso”.
cadeia	Armazenam textos. As cadeias diferem-se dos caracteres pela quantidade de caracteres que podem armazenar. Podemos pensar na cadeia como uma composição de vários caracteres. Os textos sempre devem ser escritos dentro de aspas duplas “ ”. Exemplos: “Lógica de Programação”, “Bom dia”, “Livro de Portugol”, “Casa do código”.

É possível usarmos diversas variáveis nos nossos algoritmos, e elas podem ser de quaisquer tipos disponíveis na linguagem. Não existe restrição relacionada à quantidade e nem ao uso dos tipos das variáveis, exceto pelo fato de que não é possível declararmos duas variáveis com o mesmo nome (mesmo que sejam de tipos diferentes).

Se declararmos, por exemplo, uma variável chamada de `idade`, não podemos mais declarar nenhuma outra variável com o mesmo nome `idade`. No exemplo anterior, se precisarmos ter mais de uma variável para armazenar `idade` no nosso algoritmo, podemos ter variáveis com nome `idade1`, `idade2`, `idadeNova`, `idadeAtual`, ou outro nome que facilite a sua identificação no nosso algoritmo.

4.3 NOME DE VARIÁVEIS

Ao definirmos o nome das nossas variáveis, devemos seguir algumas regras, são elas:

REGRAS DE DEFINIÇÃO DO NOME DE VARIÁVEIS

- Não devemos utilizar acentos no nome das variáveis;
- O nome não pode iniciar com um numeral;
- Não devemos utilizar caracteres especiais no nome das variáveis. Por exemplo, os caracteres = , + , - , * , & , entre outros.
- Nome de variáveis sempre iniciadas com letras em minúsculas;
- Se for necessário juntarmos duas ou mais palavras para identificação de uma variável, devemos juntá-las sem os espaços ou separá-las com underline (_).
Exemplo: nomepaciente ou nome_paciente .
- Existe uma “convenção” (sugestão de boas práticas) chamada *camelcase* que pode ser empregada;
- Para usar *camelcase*, em casos de variáveis com mais de uma palavra, devemos juntar as palavras e a partir da segunda palavra colocar a primeira letra em maiúscula. Caso seja uma única palavra, basta colocá-la toda em minúscula. **Exemplos:** nomePaciente , notaDoAluno , idade , pesoAtleta .

Exemplos de nomes de variáveis:

Recomendado	Não recomendado	Errado
idadepaciente	i	idade paciente
idadePaciente	id	1idade
idade_do_paciente	idp	-paciente
endereco_do_paciente	ep	endereço
enderecoDoPaciente	edp	#enddopaciente

BOAS PRÁTICAS DE PROGRAMAÇÃO

As boas práticas de programação são regras informais utilizadas por programadores para padronizar algo relacionado ao campo da programação.

Relacionado ao nome das variáveis, devemos utilizar nomes significativos para representar o valor que ela armazena. Por exemplo, se formos armazenar o valor relacionado ao salário de um funcionário, devemos declarar a variável como:

salarioFuncionario , salFuncionario , salario_do_funcionario , ou outros nomes significativos, e nunca utilizar s , sf , x , ou nomes que não nos lembrem o que essa variável está armazenando. Essa é uma boa prática, já que ela auxilia quem programa na hora da manutenção de algum algoritmo.

No entanto, o que devemos analisar na criação de variáveis?

Como determinamos o tipo de uma variável?

Vamos pensar juntos: pegue um papel e anote os tipos que você utilizaria na declaração das variáveis para cada um dos exemplos a seguir. Imagine que você foi contratado para criar um algoritmo para armazenar os seguintes dados de um paciente de um hospital, são eles: (a) idade do paciente, (b) nome, (c) altura e (d) peso.

Para ajudar na identificação do tipo de variável que será criada, devemos analisar quais os possíveis valores que ela pode assumir. Por exemplo, se quisermos representar o algoritmo de uma cafeteira que verifique se ela está ligada ou desligada, qual seria o tipo de dado mais apropriado da variável? Nesse caso, poderíamos utilizar o tipo lógico , que pode assumir os valores Verdadeiro ou Falso .

Mas, e se quisermos armazenar o valor de um salário, qual tipo de dado utilizaríamos? Nesse caso, temos dois tipos que armazenam valores numéricos: `inteiro` ou `real` . Como salários possuem casas decimais (centavos), devemos utilizar o tipo `real` . Se utilizássemos o tipo `inteiro` , os centavos do salário seriam perdidos.

E se quisermos armazenar a idade e o peso de um atleta, quais tipos de dados devemos usar? Nesse caso serão dois tipos diferentes, um para armazenar a idade do atleta e o outro para armazenar o peso do atleta. Para armazenar a idade do atleta, utilizaremos o tipo `inteiro` , já que as idades são representadas sem as casas decimais. Já o peso do atleta devemos declarar

aceitando casas decimais, nesse caso utilizaremos o tipo `real`.

Por fim, poderíamos armazenar o nome da mãe de um aluno, e aqui poderíamos ficar em dúvida entre a utilização do tipo `caracter` e o tipo `cadeia`. Qual você acha que seria melhor para a representação?

A resposta é o tipo `cadeia`, pois se utilizarmos o tipo `caracter` só armazenaríamos a primeira letra do nome. Já se ela for declarada com o tipo `cadeia`, conseguiremos armazenar o nome completo da mãe do aluno.

Vamos pensar juntos: pegue o papel em que anotou os tipos utilizados no algoritmo para armazenar os dados de um paciente de um hospital e veja quais foram os tipos que você utilizou antes dessa explicação. Você mudaria algum tipo declarado anteriormente?

A resposta correta para cada um dos exemplos é: (a) idade usar o tipo `inteiro`, (b) nome usar o tipo `cadeia`, (c) altura usar o tipo `real` e (d) peso usar o tipo `real`. Então as declarações das variáveis no nosso programa ficariam conforme exemplo a seguir.

```
//Exemplo 10: declaração das variáveis para algoritmo de hospital
inteiro idade
cadeia nome
real altura, peso
```

Perceba que, quando desejamos declarar mais de uma variável vinculada a um mesmo tipo, podemos separar os nomes identificadores das variáveis por vírgulas. Outra possibilidade de declaração das mesmas variáveis do exemplo anterior é mostrada

no exemplo a seguir.

```
//Exemplo 11: outra forma de declaração das variáveis para algoritmo de hospital
inteiro idade
cadeia nome
real altura
real peso
```

4.4 ATRIBUIÇÃO DE VALORES NAS VARIÁVEIS

Até o momento vimos que todas as variáveis precisam ser declaradas antes de serem utilizadas. Ao declararmos uma variável em um algoritmo, o compilador indica ao computador que o algoritmo precisa de uma área de memória para armazenar um valor.

Porém, a declaração serve apenas para separar a área da memória. Para utilizarmos uma variável no nosso algoritmo, precisamos realizar a operação de **atribuição de valor**. A atribuição de valor armazena um valor dentro do espaço separado na memória.

A atribuição pode ser realizada de quatro formas diferentes.

1. Através de um valor fixado diretamente no algoritmo;
2. Através do valor de outra variável;
3. Através do resultado de uma operação matemática;
4. Através da entrada de dados pelo teclado;

Calma! Veremos em detalhes como atribuímos valores a variáveis de cada uma dessas formas no livro.

Atribuição de valores fixos no código (diretamente no algoritmo)

Para atribuirmos um valor fixo a uma variável, basta colocarmos o nome da variável, declarada anteriormente, um único símbolo de igual = e o valor que se deseja atribuir (armazenar).

```
//Exemplo 12: atribuição  
idadePaciente = 40
```

Assim que o compilador executar a linha `idadePaciente = 40`, ele vai até o local de memória identificado como `idadePaciente` e armazenará o valor 40. A partir daí, ao chamarmos essa variável no algoritmo, ele nos retornará o valor 40. Essa atribuição pode ocorrer várias vezes ao longo do nosso algoritmo e também pode ocorrer na declaração de uma variável. Fique tranquilo que veremos outros exemplos para fixarmos mais esse conteúdo.

```
//Exemplo 13: atribuição de valores fixos no algoritmo  
inteiro idadeAtleta = 25  
real pesoAtleta  
  
pesoAtleta = 75.4  
idadeAtleta = 30
```

Repare que no exemplo anterior temos duas maneiras de atribuição fixa. Uma está ocorrendo na mesma linha da declaração da variável `idadeAtleta`, já a outra ocorre em um outro momento do nosso algoritmo, na linha `idadeAtleta = 30`.

As atribuições podem ocorrer diversas vezes no nosso algoritmo e em qualquer linha. O que é importante lembrarmos é que a variável só armazena o último valor atribuído, os valores

anteriores são perdidos.

Veja o exemplo da variável `idadeAtleta`. Na declaração da variável, o valor atribuído é 25. Porém, após a definição do valor de `pesoAtleta`, ocorre uma nova atribuição na variável `idadeAtleta`, que recebe o valor 30. A partir desse momento, se olharmos o espaço de memória do computador, veremos que o valor armazenado na variável `idadeAtleta` é 30 e não mais 25. Isso ocorre porque a memória não registra histórico, ou seja, ela só consegue armazenar o último valor atribuído, os demais são apagados.

Se você ainda ficou confuso, não se preocupe. Nos próximos capítulos, veremos mais detalhes sobre as operações de atribuição.

Atribuição através do valor de outra variável

A atribuição através do valor de outra variável é utilizada quando queremos que uma variável armazene o valor contido em outra variável. Para isso, devemos escrever o nome da variável que receberá o valor seguido do operador de atribuição igual `=` e o nome da variável que contém o valor desejado.

```
//Exemplo 14: atribuição através do valor de outra variável
real precoAtual, precoAntigo
precoAtual = 10.50
precoAntigo = precoAtual
precoAtual = 11.50
```

Repare no exemplo anterior que foram criadas duas variáveis: `precoAtual` e `precoAntigo`. Para não perder o conteúdo de `precoAtual`, ele é atribuído ao valor da variável `precoAntigo` (no exemplo 10.50). Em seguida, o valor do `precoAtual` é atualizado para 11.50. Assim, temos na variável `precoAtual` o

preço atualizado e, na variável `precoAntigo`, o preço anteriormente copiado da variável `precoAtual`.

Vale ressaltar que a ordem das variáveis na operação de atribuição é importante. Uma linha contendo `precoAntigo = precoAtual` é diferente de uma linha contendo `precoAtual = precoAntigo`. A variável que aparece antes do símbolo `=` é a variável que terá o valor modificado. A variável que aparece após o símbolo `=` é o local de memória de onde o valor será copiado, não sofrendo nenhuma modificação.

Atribuição através do resultado de uma operação matemática

Outra forma de atribuirmos um valor a uma variável é utilizando o resultado de uma operação matemática. Por exemplo, se tivermos o ano de nascimento de uma pessoa, podemos criar uma variável chamada `idade` para receber automaticamente sua idade. Para tanto, é necessário realizarmos uma operação aritmética - nesse caso, de subtração. O símbolo utilizado para a operação aritmética de subtração é o `-`. No próximo capítulo, veremos todos os operadores aritméticos que podemos utilizar nos nossos algoritmos.

```
//Exemplo 15: calcular a idade de uma pessoa tendo somente o ano de nascimento
```

```
    inteiro idade, anoNascimento = 1981, anoAtual = 2021  
    idade = anoAtual - anoNascimento
```

Observação: repare que para o exemplo anterior desconsideramos o mês do nascimento.

Se rodarmos o algoritmo anterior, não veremos nada no console, pois o código não contém o comando `escreva()`.

Porém, não queremos mostrar uma mensagem fixa no console. Queremos que seja apresentado o resultado da operação matemática de subtração. Podemos ficar com a seguinte dúvida: como posso apresentar o valor de uma variável no console?

Como já sabemos, o comando responsável por apresentar textos no console é `escreva()`. Porém, ele também é o comando utilizado para escrever o valor de uma variável no console. Para que isso ocorra, basta colocarmos o nome da variável desejada dentro dos parênteses. Repare que ao exibir o valor de uma variável não devemos colocá-lo dentro das aspas duplas.

```
//Exemplo 16: exibição na tela de valor de uma variável
inteiro idade, anoNascimento = 1981, anoAtual = 2021
idade = anoAtual - anoNascimento
escreva (idade)

//Saída no console
40
```

Repare que no console é exibido o valor armazenado na variável `idade`, porém sem nenhum texto prefixado. Para mesclarmos (juntarmos) textos prefixados e valores de variáveis, devemos utilizar o operador vírgula `,`. Dessa maneira, podemos criar um texto de saída para o usuário que apresente texto prefixado e valores de variáveis.

Vejamos o exemplo a seguir.

```
//Exemplo 17: exibição na tela de valor de uma variável
inteiro idade, anoNascimento = 1981, anoAtual = 2021
idade = anoAtual - anoNascimento
escreva ("Você tem ", idade, " anos.")

//Saída no console
Você tem 40 anos.
```

No comando `escreva()`, aparece um texto fixo (dentro de

aspas duplas), a variável `idade` e outro texto fixo dentro de aspas duplas. Perceba que entre os textos e a variável existe um operador vírgula `,`. Como dito anteriormente, ele serve para juntar todo o conteúdo do comando `escreva` em apenas um texto de saída.

É preciso que fiquemos atentos à utilização das aspas e das vírgulas agora. As aspas duplas devem ser usadas quando queremos incluir na saída do programa um texto fixo. Para indicar ao compilador que queremos juntar o texto fixo ao valor de uma variável, devemos usar o operador vírgula entre eles. Quando quisermos apresentar o valor de uma variável, o nome dessa variável deverá vir **fora** das aspas. Dessa forma, sinalizamos que aquilo não representa um texto a ser impresso no console e sim um valor que será buscado na memória. Veja um outro exemplo a seguir:

```
//Exemplo 18: exibição na tela de valor de uma variável  
real peso = 85.5  
real altura  
altura = 1.80  
  
escreva ("Você pesa ", peso , " e tem ", altura, " de altura")  
  
//Saída no console  
Você pesa 85.5 e tem 1.80 de altura
```

Existe outra maneira de atribuirmos valor para as variáveis que é através de dados lidos do teclado. Veremos essa forma de atribuição no capítulo 6 (comando `leia`).

Resumo do capítulo

Vimos que a **variável** é utilizada para armazenar valores no nosso algoritmo. Toda variável precisa ser criada antes de ser utilizada e chamamos essa criação de **declaração de variável**. Ao declararmos uma variável, precisamos informar o tipo da variável e o seu nome (identificador). As variáveis podem ser do tipo `inteiro` , `real` , `caracter` , `logico` e `cadeia` . Os nomes devem ser significativos. Ao declarar uma variável, o compilador separa uma área na memória do computador, identificando-o com o nome

definido e capaz de armazenar um dado do tipo especificado. Depois de declararmos uma variável, já podemos utilizá-la. Para isso, realizamos a operação de **atribuição de valores**. As atribuições podem ser realizadas através de valores fixados no algoritmo, através do valor de outra variável e através do resultado de uma operação matemática. Para realizar a atribuição de valor a uma variável, basta colocar o nome da variável seguido de um símbolo de igual = e, na sequência, o valor ou operação desejada. Por fim, vimos que é possível apresentar o valor de uma variável no console. Para isso, basta colocar o nome da variável dentro dos parênteses do comando `escreva` e sem as aspas duplas. Para intercalar textos fixos e variáveis no comando `escreva()`, devemos separá-los com o operador vírgula , .

Exercícios

5. (F) Crie um algoritmo com a declaração de três variáveis para um sistema de caixa de um supermercado.
6. (F) Crie um algoritmo que declare e atribua valores fixos para cinco variáveis de um algoritmo para o gerenciamento de uma escola. Mostre na tela os valores dessas variáveis para os usuários.
7. (M) Crie um algoritmo que declare e atribua valores fixos para três variáveis de um algoritmo para o gerenciamento de uma loja de roupas. Mostre na tela os valores das variáveis criadas.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Pedro solicitou alteração no seu sistema. Além da mensagem de boas-vindas, você deve alterar o algoritmo para que ele apresente um texto com o endereço e o número da loja.

Você deve armazenar essas informações em variáveis.

Apresente na tela a nova mensagem.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap4>



Figura 4.2: QRCode

CAPÍTULO 5

OPERADORES E EXPRESSÕES ARITMÉTICAS

Falamos anteriormente que podemos realizar operações aritméticas nos nossos algoritmos. No exemplo para calcular a idade, dado o ano atual e um ano de nascimento, usamos o operador de subtração - para obter o resultado.

```
//Exemplo 19: calcular a idade de uma pessoa tendo somente o ano de nascimento  
    inteiro idade, anoNascimento = 1981, anoAtual = 2021  
    idade = anoAtual - anoNascimento
```

Rpare que a variável `idade` recebe o resultado da operação de subtração entre as variáveis `anoAtual` e `anoNascimento`. Além da operação de subtração, temos as de adição, multiplicação e divisão. Para os números inteiros, temos também um operador que retorna o resto da divisão.

Operação	Descrição da operação
Adição: +	Realiza a soma de dois ou mais valores. Esses valores podem estar armazenados em variáveis ou não. Nesse caso, a ordem das variáveis ou valores não importa.
Subtração: -	Realiza a diferença entre dois ou mais valores. Assim como na soma, os valores podem ser de variáveis ou não. Aqui, a ordem importa.

Multiplicação: *	Calcula o produto entre dois ou mais valores, que podem estar em variáveis ou serem utilizados diretamente na operação. A ordem dos termos não importa, resultando no mesmo resultado.
Divisão: /	Retorna o valor da divisão entre dois valores. A ordem e o tipo das variáveis são levados em consideração. Se realizado com variáveis do tipo real, o resultado apresenta um valor real também. Caso seja realizado com inteiros, o resultado é do tipo inteiro, sem casas decimais.
Resto da divisão: %	Bastante similar ao da divisão, no entanto, enquanto o operador / retorna o quociente da divisão, o % retorna o resto da divisão. Perceba que o operador só faz sentido se aplicado em valores do tipo inteiro.
Precedência: ()	Indica quais operações devem ser realizadas primeiramente. Os operadores têm a mesma regra de precedência da matemática: parênteses, potências, multiplicação e divisão, adição e subtração.

Vamos pensar juntos: com base no que aprendemos até o momento, crie um algoritmo com duas variáveis. Uma deve ter o valor 20 e a outra o valor 5. Realize as operações de adição, subtração, multiplicação e divisão. Mostre o resultado das operações no console.

```
//Exemplo 20: exibição na tela de valor de uma variável
real numero1 = 20.0, numero2 = 5.0, adicao, sub, multi, divisao

adicao = numero1 + numero2
sub = numero1 - numero2
multi = numero1 * numero2
divisao = numero1 / numero2

escreva ("Adição: ", adicao, "\n")
escreva ("Subtração: ", sub, "\n")
escreva ("Multiplicacao: ", multi, "\n")
escreva ("Divisão: ", divisao, "\n")

//Saída no console
Adição: 25.0
Subtração: 15.0
```

```
Multiplicação: 100.0
Divisão: 4.0
```

Repare que o enunciado não informava o tipo das variáveis e nem que seria necessário calcular o resto da divisão, por esse motivo optamos pelas variáveis do tipo real. Porém, poderíamos montar o mesmo algoritmo utilizando as variáveis como inteiro. Como mostrado no exemplo anterior, além das variáveis com os dois valores iniciais, precisamos criar outras quatro variáveis para armazenar o resultado de cada uma das operações aritméticas.

Vamos pensar juntos: no exemplo 20 não usamos o operador %. Imagine agora que as variáveis declaradas no exemplo 20 foram alteradas para o tipo inteiro e a variável numero1 tem o valor 10 e a variável numero2 tem o valor 3. Como alterar o algoritmo anterior para que sejam exibidos o quociente e o resto da divisão?

```
//Exemplo 21: exibição na tela de valor de uma variável
inteiro numero1 = 10, numero2 = 3, adicao, sub, multi, quo, resto

adicao = numero1 + numero2
sub = numero1 - numero2
multi = numero1 * numero2
quo = numero1 / numero2
resto = numero1 % numero2

escreva ("Adição: ", adicao, "\n")
escreva ("Subtração: ", sub, "\n")
escreva ("Multiplicacao: ", multi, "\n")
escreva ("Quociente da divisão: ", quo, "\n")
escreva ("Resto da divisão: ", resto, "\n")

//Saída no console
Adição: 13
Subtração: 7
```

Multiplicação: 30
Quociente da divisão: 3
Resto da divisão: 1

Repare que criamos uma nova variável, chamada de `resto`, para receber o valor do resto da divisão, e alteramos o nome da variável de divisão para `quo` (quociente). Na variável `quo`, temos como resultado o valor 3. O 3 apareceu pois, se dividirmos 10 por 3, a resposta será 3 ($3 * 3 = 9$) e sobrará como resto 1. Repare que é o valor que aparece na variável `resto`.

Resumo do capítulo

Vimos os operadores aritméticos. Podemos utilizar os operadores `+`, `-`, `*` e `\`. Além dos operadores apresentados, ainda temos o `%` que apresenta o resto da divisão de valores inteiros. Assim como na matemática, podemos usar os parênteses para indicar a ordem das operações.

Exercícios

8. (F) Crie um algoritmo com três variáveis `num1`, `num2` e `num3` com os respectivos valores 100, 5 e 2. Calcule e apresente a seguinte expressão `(num1 + num2) * num3`.

9. (F) Crie um algoritmo com os valores das notas da primeira prova e da segunda prova de um aluno e apresente a média aritmética do aluno.

Legenda: F - fácil / M - médio / T - trabalhoso

Vídeo com o resumo do capítulo

<https://logicacomportugol.com.br/cap5>



Figura 5.1: QRCode

CAPÍTULO 6

COMANDO LEIA()

No capítulo anterior, vimos as variáveis e três maneiras de atribuirmos valores a elas. Podemos atribuir valores fixos no algoritmo, valores de outras variáveis e valores de resultados de uma operação matemática.

Vamos relembrar com o exemplo a seguir:

```
//Exemplo 22: três formas de atribuição de valores vistos no capitulo anterior

// 1) Atribuição por valores fixos
cadeia pais = "Brasil"
real mediaProva = 7.0

//2) Atribuição por valores de outra variável
inteiro qtdAlunos = 30, qtdProvasImpressas
qtdProvasImpressas = qtdAlunos

//3) Atribuição por valor do resultado de uma operação matemática

real precoGasolina = 5.20, total, qtdLitros
qtdLitros = 10.0
total = precoGasolina * qtdLitros
escreva ("Você deve pagar R$", total)
```

Vamos pensar juntos: qual é o problema dos algoritmos anteriores? Imagine que o algoritmo que controla a venda de

gasolina seja utilizado para cinco (5) clientes diferentes. Qual valor que o algoritmo mostrará para cada um dos cinco clientes?

Observando o exemplo 22, da gasolina, fixamos o valor diretamente no código, o preço da gasolina e a quantidade de litros que o cliente comprou. Com isso, o algoritmo sempre mostrará a mesma mensagem (Você deve pagar R\$ 52.00), independentemente de quantas vezes ele for executado.

```
//Saída no console  
Você deve pagar R$ 52.00
```

É como se usássemos uma calculadora que sempre apresenta o mesmo resultado. Qual o objetivo de se ter uma calculadora que só apresenta o mesmo resultado? Nenhum. No nosso algoritmo é a mesma coisa, ele não pode realizar os cálculos usando valores fixados no código, nosso algoritmo deve receber os valores a cada execução e então realizar o cálculo.

No exemplo 22, precisamos que quem estiver executando o nosso algoritmo informe o preço da gasolina e a quantidade de litros para realizar o cálculo. Para isso, precisamos utilizar o comando `leia()`. Ele é responsável por realizar a leitura dos dados através do **teclado**.

Esse processo de realizar a leitura do teclado é a quarta maneira de realizarmos a atribuição de valores. Porém, agora não utilizaremos mais o operador de atribuição igual = e sim o comando `leia()`. Dentro da abertura e do fechamento dos parênteses informamos o nome da variável que armazenará o valor lido do teclado.

//3) Exemplo 23: gasolina com a leitura dos dados pelo teclado

```
real precoGasolina, total, qtdLitros  
leia(precoGasolina)  
leia(qtdLitros)  
total = precoGasolina * qtdLitros  
escreva ("Você deve pagar R$", total)
```

Vejamos outro exemplo com a leitura dos dados sendo feita pelo teclado, através do comando `leia()`.

```
//Exemplo 24: atribuição por valor lido do teclado  
inteiro qtdAlunos  
leia(qtdAlunos)  
escreva("Quantidade de alunos presentes:", qtdAlunos)  
  
//Saída no console  
--
```

Vamos pensar juntos: execute o exemplo anterior e diga o que apareceu no console?

Se observarmos bem, o console não apresentou nenhuma mensagem, ele apenas ficou com um cursor piscando. Isso significa que ele está aguardando que você entre com alguma informação pelo teclado e aperte a tecla *enter*. Só depois que você apertar o *enter* é que o compilador continuará a executar seu algoritmo.

Uma boa prática é a utilização de mensagens de orientação para os usuários antes de qualquer entrada de dados. Para isso, basta colocarmos um `escreva()` antes do `leia()`. O comando `escreva()` deve apresentar mensagens com orientações informando o tipo de valor que o usuário deve informar. Vamos alterar o exemplo anterior incluindo um comando `escreva()` antes do comando `leia()`.

```
//Exemplo 25: atribuição por valor lido do teclado
inteiro qtdAlunos
escreva("Digite a quantidade de alunos:")
leia(qtdAlunos)
escreva("Quantidade de alunos presentes:", qtdAlunos)

//Saída no console
Digite a quantidade de alunos:_
```

Repare que, ao executarmos o algoritmo, o console apresentará a mensagem para informar a quantidade de alunos e ficará com o cursor piscando. Perceba que dessa maneira fica mais fácil para quem estiver usando seu algoritmo saber qual informação deve ser digitada. Nesse exemplo, só tínhamos uma (1) entrada, porém se pensarmos em um cadastro de um aluno, por exemplo, teremos uma tela com mais de 20 informações. Se não informarmos essa ordem ao usuário, ele provavelmente entrará com informações em locais errados.

Veja outro exemplo a seguir:

```
//Exemplo 26: comando leia()
real valorDaBarraDeChocolate
inteiro quantidadeDeBarrasDeChocolate
escreva("Digite o valor da barra de chocolate:")
leia(valorDaBarraDeChocolate)
escreva("Digite a quantidade de barras que deseja comprar:")
leia(quantidadeDeBarrasDeChocolate)

escreva("Você comprou ", quantidadeDeBarrasDeChocolate , " barra(s) de chocolate e o total a ser pago é de R$ ", valorDaBarraDeChocolate * quantidadeDeBarrasDeChocolate)

//Saída no console
Digite o valor da barra de chocolate:3.50
Digite a quantidade de barras que deseja comprar:2
Você comprou 2 barra(s) de chocolate e o total a ser pago é de R$ 7.0
```

Resumo do capítulo

Vimos o comando utilizado para realizar a atribuição de valores lidos pelo teclado. O comando é o `leia()`. Devemos colocar dentro dos parênteses o nome da variável que receberá e armazenará o valor informado. Devemos lembrar que essa variável deve ter sido declarada anteriormente. Como boa prática, sempre utilizaremos um `escreva()` na linha anterior ao comando `leia()`, indicando ao usuário do algoritmo qual dado ele deve informar.

6.1 EXERCÍCIOS

10. (F) Crie um programa que pergunte a um aluno ou aluna o seu nome, a idade e altura. Depois de obter esses valores, apresente na tela os dados solicitados.

11. (F) Crie um programa para solicitar ao usuário o seu nome e uma frase preferida. Você deve mostrar o nome e a frase no console da seguinte maneira: A frase preferida da(o) X é Y. (X é o valor do nome do usuário e Y é a frase digitada pelo usuário).

12. (F) Crie um programa para solicitar ao usuário uma variável de cada tipo. Apresente na tela os valores digitados em cada variável. Utilize sua imaginação para criar as variáveis.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Pedro não gostou da mensagem de boas-vindas e decidiu reescrever o texto. Seu algoritmo deverá fornecer agora uma mensagem personalizada aos clientes. O algoritmo deve, primeiramente, perguntar ao cliente o seu nome e em seguida apresentar a seguinte mensagem. "Prezado(a), xxx. Seja muito bem-vindo(a) à nossa loja."

Na linha seguinte deve aparecer o seguinte texto: "Oferecemos em nossa loja venda e manutenção de bicicletas. Para venda de bicicletas, procure o colaborador Junior e, para manutenção, procure o colaborador Neto. Obrigado e esperamos que tenha uma ótima experiência em nossa loja.".

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap6>



Figura 6.1: QRCode

Estruturas condicionais



Figura 1: Fonte: Imagem de Gerd Altmann Pixabay.

Nestes capítulos, estudaremos as estruturas condicionais. Elas são responsáveis por dar um certo grau de inteligência aos nossos algoritmos. A partir delas, nossos algoritmos tomarão decisões de quais linhas serão executadas ou não. Estudaremos os comandos `se` , `se senao` , `se senao se` e `escolha caso` .

CAPÍTULO 7

CONDICIONAL

Até o momento vimos que todas as linhas dos nossos algoritmos são executadas sequencialmente, uma após a outra. Neste capítulo, veremos as estruturas de decisão, também chamadas de estruturas de seleção ou condicionais. Elas são utilizadas quando queremos que os nossos algoritmos executem ou não executem um trecho de código (uma linha ou várias linhas).

Vamos pensar juntos: vamos esquecer um pouco o mundo computacional e pensar no nosso dia a dia. Imagine que você juntou dinheiro durante todo o ano para comprar um novo videogame na *black friday* deste ano. Porém, você só o comprará se ele estiver custando R\$ 1.200,00. Se ele for anunciado mais barato, exemplo, R\$ 1.000,00 você comprará? E se ele for anunciado por R\$ 2.000,00, você comprará? Se não for comprar, fará o quê?

Repare que no exemplo dado existe uma condição (comparação) sendo realizada entre o valor que você tem para comprar o videogame e o valor que ele custará. Se essa condição for verdadeira, ou seja, se o preço for R\$ 1.200,00 ou menos, você comprará. O texto não diz o que será feito caso ele seja anunciado por mais de R\$ 1.200,00. Por exemplo, juntará mais dinheiro para

comprar em outra época, ou em vez de comprar o videogame comprará uma bicicleta. O texto só se preocupa em comprar ou não comprar.

Essa mesma condição (comparação) apresentada no exemplo anterior também será utilizada nos nossos algoritmos. Porém, precisamos representar as condições de uma forma que o compilador entenda. Nesses casos, utilizaremos **expressões relacionais**.

7.1 OPERADORES E EXPRESSÕES RELACIONAIS

Para que seja possível realizarmos uma condição, precisamos comparar dois ou mais elementos. Os operadores relacionais, como o próprio nome diz, são utilizados para relacionar o conteúdo de duas variáveis ou dois valores fixos nos nossos algoritmos. Os operadores utilizados no Portugol Studio são:

Operação	Operador	Descrição da operação
maior	>	Verifica se o valor à esquerda é maior que o valor à direita.
maior ou igual	\geq	Verifica se o valor à esquerda é maior ou igual ao valor à direita.
menor	<	Verifica se o valor à esquerda é menor que o valor à direita.
menor ou igual :	\leq	Verifica se o valor à esquerda é menor ou igual ao valor à direita.
igual	\equiv	Verifica se o valor à esquerda é igual ao valor à direita.
diferente	\neq	Verifica se o valor à esquerda é diferente do valor à direita.

Os operadores relacionais são utilizados para a construção das expressões relacionais.

SINTAXE DE EXPRESSÕES RELACIONAIS

valor1 OPERADOR_RELACIONAL valor2

Aqui, `valor1` pode ser uma variável ou um valor fixo no algoritmo; `OPERADOR_RELACIONAL` é o operador relacional utilizado na comparação; `valor2` pode ser uma variável ou um valor fixo.

Observação: os tipos de dados de `valor1` e `valor2` devem ser iguais.

Exemplos: `2 > 3` , `idade == 25` , `qtdAlunos > 10` , `vagas > qtdCarros`, `previsaoTempo != "chuva"`, `letra != 'x'`

Toda expressão relacional terá como resultado os valores **verdadeiro** ou **falso**. O valor do resultado é chamado de **resultado lógico**.

Caso o exemplo do videogame fosse escrito na forma algorítmica, o resultado da expressão relacional seria verdadeiro se o preço fosse igual ou menor que R\$ 1.200,00, ou falso se o preço fosse maior. Veja como poderíamos representar o algoritmo do exemplo:

```
//Exemplo 27: algoritmo do exemplo da compra do videogame  
real precoVideoGame
```

```
escreva("Digite o preço do anúncio do videogame.")
```

```
leia(precoVideoGame)

se (precoVideoGame <= 1200.00){
    escreva("Oba! Tenho dinheiro para comprar meu videogame.")
}
```

Agora que já vimos os operadores relacionais e o uso dos operadores nas expressões relacionais, vamos conhecer a primeira estrutura condicional.

7.2 CONDICIONAL SIMPLES: SE()

Ela é chamada de simples pois o comando só se preocupa quando o resultado da condição for **verdadeiro**. Se for, o compilador executará o trecho de código escrito dentro do par de chaves do comando `se`, caso contrário, nada será feito.

Definição formal

Deitel e Deitel (1999) apresentam que o objetivo da estrutura de seleção é “[...] realizar uma ação se uma condição for verdadeira e realizar uma ação diferente se a condição for falsa”. (DEITEL; DEITEL, 1999, p. 49).

Para Souza et al (2019), as estruturas de decisão “são estruturas que permitem a tomada de uma decisão sobre qual o caminho a ser escolhido, de acordo com o resultado de uma expressão lógica.” (SOUZA et al, 2019, p. 151).

Vamos pensar juntos: de acordo com a definição apresentada,

analise os dois algoritmos, suas saídas e veja a diferença entre eles.

SINTAXE DA ESTRUTURA CONDICIONAL SIMPLES SE()

```
se (condicao){  
}
```

Nessa sintaxe, condicao será uma expressão relacional. Caso a expressão apresente como resultado o valor **verdadeiro**, os comandos escritos dentro da abertura e do fechamento da chave do se serão executados. Caso a expressão apresente como resultado o valor **falso**, o compilador pulará até a primeira linha após o fechamento da chave e não executará o que está dentro do par de chaves do comando se .

```
//Exemplo 28: condicional simples para verificar se a pessoa tem  
altura mínima para entrar no brinquedo.  
real altura = 1.65  
  
se (altura > 1.60){  
    escreva("Altura permitida")  
}  
  
escreva("\nContinuação do algoritmo")  
  
//Saída no console  
Altura permitida  
Continuação do algoritmo  
  
//Exemplo 29: condicional simples para verificar se a pessoa tem  
altura mínima para entrar no brinquedo.  
real altura = 1.40
```

```
se (altura > 1.60){  
    escreva("Altura permitida")  
}  
  
escreva ("\\nContinuação do algoritmo")  
  
//Saída no console  
Continuação do algoritmo
```

Repare que os dois algoritmos são idênticos, exceto pelo valor atribuído à variável `altura`. No primeiro exemplo, 28, ela começa com 1.65 e no segundo, (29), com 1.40. O algoritmo utiliza uma estrutura condicional com uma expressão relacional verificando se o valor da variável `altura` é maior que um valor fixo, no caso 1.60. Se for, a saída da expressão será verdadeira e o compilador entrará no par de chaves, executando o trecho de código. Caso o resultado seja falso, o compilador não entrará no par de chaves e com isso não executará os comandos ali contidos. É por isso que, no exemplo 28, ele apresenta no console a mensagem "Altura permitida" e no segundo ele não mostra.

Vamos pensar juntos: Analise o algoritmo a seguir e pense no que será exibido no console.

```
//Exemplo 30: condicional simples para verificar se uma pessoa é  
maior de idade.  
inteiro idade  
  
escreva("Digite sua idade:")  
leia(idade)  
  
se (idade >= 18){  
    escreva("Maior de idade")  
}
```

Repare que o algoritmo solicita que o usuário digite no teclado

o valor da idade. Se a idade digitada for maior ou igual a 18 anos, será exibida a mensagem "Maior de idade".

Vamos pensar juntos: se quiséssemos que o algoritmo apresentasse, além da mensagem "Maior de idade", a mensagem "Menor de idade" quando a idade digitada fosse menor que 18 anos, o que teríamos que alterar no nosso algoritmo?

```
//Exemplo 31: condicional simples para verificar se a pessoa é maior de idade ou menor de idade.
```

```
inteiro idade
```

```
escreva("Digite sua idade:")
leia(idade)

se (idade >= 18){
    escreva("Maior de idade")
}

se (idade < 18){
    escreva("Menor de idade")
}
```

Repare que dessa maneira temos duas estruturas condicionais, cada uma com sua mensagem específica. Dependendo da idade, ou ela será verdadeira na primeira comparação, ou na segunda. Porém, existe uma forma mais otimizada de escrevermos esse algoritmo.

7.3 CONDICIONAL COMPOSTA: SE SENAO

Como dito anteriormente, toda expressão relacional tem uma única saída que pode ser **verdadeira** ou **falsa**. Na estrutura condicional simples, vimos que o algoritmo só se preocupa quando

a saída for verdadeira. A estrutura condicional composta já se preocupará com as duas possíveis saídas. Assim teremos trechos de códigos para serem executados quando a saída for verdadeira e outro trecho de código quando a saída for falsa.

SINTAXE DA ESTRUTURA CONDICIONAL COMPOSTA SE SENAO

```
se (condicao){  
} senao {  
}
```

condicao será uma expressão relacional. Caso a expressão apresente como resultado o valor **verdadeiro**, os comandos escritos dentro da abertura e do fechamento da chave do se serão executados. Caso a expressão resulte no valor **falso**, o compilador pulará até o comando senao e executará o trecho de código dentro do par de chaves do comando senao .

Vamos pensar juntos: como podemos alterar o algoritmo do exemplo 31 para usar o condicional composto se senao .

```
//Exemplo 32: condicional composta para verificar se a pessoa é maior de idade ou menor de idade.  
inteiro idade  
  
escreva("Digite sua idade:")  
leia(idade)  
  
se (idade >= 18){
```

```
    escreva("Maior de idade")
}senao {
    escreva("Menor de idade")
}
```

Repare que no exemplo 31 tínhamos duas expressões relacionais. Porém, ao utilizarmos a estrutura condicional composta, não precisamos mais de duas comparações. Apenas uma é suficiente, já que o resultado da expressão relacional terá como saída duas possibilidades, verdadeira ou falsa. A verdadeira é representada pelo conteúdo contido no par de chaves do `se` e a falsa é representada pelo valor contido no `senao`.

Dessa maneira, otimizamos o tempo de execução do nosso algoritmo, já que temos uma comparação a menos a ser realizada.

Vamos analisar outros exemplos.

```
//Exemplo 33: condicional composto
//Sistema para indicar qual documento deve ser solicitado de acordo com a nacionalidade.
cadeia paisDeNascimento

escreva("Digite seu país de nascimento:")
leia(paisDeNascimento)

se (paisDeNascimento == "Brasil"){
    escreva("Solicitar número de CPF.")
}senao {
    escreva("Solicitar número do passaporte.")
}
```

Vamos pensar juntos: analise o algoritmo anterior e veja que ele foi construído usando um operador relacional específico. Seria possível reescrever esse algoritmo usando outro operador relacional? Qual?

```

//Exemplo 34: condicional composto
//Sistema para indicar qual documento deve ser solicitado de acordo com a nacionalidade.
cadeia paisDeNascimento

escreva("Digite seu país de nascimento:")
leia(paisDeNascimento)

se (paisDeNascimento != "Brasil"){
    escreva("Solicitar número do passaporte.")
}senao {
    escreva("Solicitar número de CPF.")
}

```

Repare que os dois exemplos anteriores, 33 e 34, foram construídos para apresentar o documento que deve ser solicitado, de acordo com a nacionalidade. Ao usarmos o operador igual == temos que verificar se o que foi digitado no teclado é igual ao texto "Brasil". Com a igualdade, teremos duas possibilidades, ou o paisDeNascimento será igual ou diferente de Brasil . Se for igual, ele apresentará uma mensagem e se for falso, apresentará outra mensagem.

Podemos mudar a forma da comparação: ao invés de usarmos o igual, usaremos o diferente != (exemplo 34). Veja que, além de mudar o operador, precisamos mudar a ordem das mensagens dentro de cada par de chaves.

```

//Exemplo 35: condicional composta
//Sistema para indicar a sala do cinema de acordo com filme selecionado.
inteiro filme

escreva("XPTO Cinema\n")
escreva("Digite 1 para assistir ao filme Matrix e 2 para assistir ao filme Jogo da Imitação.")
leia(filme)

se (filme == 1){

```

```
    escreva("Sala A no final do corredor")
}senao {
    escreva("Sala B ao lado do banheiro")
}
```

Vamos pensar juntos: construa um algoritmo para ler dois números do teclado e apresente qual é o maior entre eles.

```
//Exemplo 36: condicional composto
//Sistema para verificar qual o maior entre dois números informados pelo teclado
inteiro numero1, numero2

escreva("Digite o número 1: ")
leia(numero1)
escreva("Digite o número 2: ")
leia(numero2)

se (numero1 > numero2){
    escreva("Número 1 maior")
}senao {
    escreva("Número 2 maior")
}
```

Repare que o algoritmo foi construído verificando se o valor armazenado na variável numero1 é maior do que o valor armazenado na variável numero2 , porém ele poderia ter sido escrito de diversas outras maneiras. Poderíamos colocar o comando `se(numero1 < numero2)` ou `se(numero2 > numero1)` . Em todos os casos, devemos verificar quando o resultado será verdadeiro ou falso, e então colocar o trecho de código que será executado no par de chaves correspondente.

Vamos pensar juntos: analise com mais atenção o algoritmo do exemplo 36. Se entrarmos com o valor para variável numero1

igual ao valor da variável `numero2` o que vai acontecer? Como podemos resolver esse problema?

No caso de entrarmos com valores iguais, o algoritmo verificará se `numero1` é maior que o `numero2` e a expressão terá como resultado `falso`. Logo, ele vai para o `senao` e mostrará a mensagem "Número 2 maior".

Para resolvemos esse problema, podemos alterar o algoritmo da seguinte maneira.

```
//Exemplo 37: condicional composto
//Sistema para verificar qual o maior entre dois números informados pelo teclado.
// Entrando com valores iguais pelo teclado
inteiro numero1, numero2

escreva("Digite o número 1: ")
leia(numero1)
escreva("Digite o número 2: ")
leia(numero2)

se(numero1 != numero2){
    se (numero1 > numero2){
        escreva("Número 1 maior")
    }senao {
        escreva("Número 2 maior")
    }
}senao{
    escreva("Números iguais")
}
```

O exemplo anterior, 37, resolve o problema apresentado. Porém, aprofundaremos um pouco mais nas estruturas condicionais no próximo capítulo e veremos que esse algoritmo poderia ser escrito de maneira otimizada.

Resumo do capítulo

Vimos inicialmente que as estruturas condicionais servem para mostrar ou não mostrar um determinado trecho de código. Se o trecho de código será ou não executado dependerá do resultado lógico da expressão relacional. Toda expressão relacional só pode ter como resultado os valores verdadeiro ou falso. Os operadores relacionais são utilizados nas expressões relacionais para realização das comparações. Vimos que podemos utilizar os seguintes operadores nas nossas expressões: > (maior), >= (maior ou igual), < (menor), <= (menor ou igual), == (igual) ou != (diferente). Em seguida, vimos o comando condicional simples `se`. O condicional simples só se preocupa quando o resultado lógico da expressão tiver valor verdadeiro, nesse caso ele executará o trecho de código que estiver dentro do par de chaves do comando `se`. Caso o resultado seja falso, o compilador pulará o par de chaves do `se` e executará a primeira linha abaixo do fechamento da chave. Para finalizar o capítulo, vimos o condicional composto `senao`. No condicional composto, já é previsto ter trecho de código quando o resultado da expressão relacional for verdadeiro e falso. Quando for verdadeiro, o compilador executará somente o trecho de código que estiver dentro do par de chaves do `se`, ignorando o trecho de código contido no par de chaves do `senao`. Se o resultado da operação for falso, o compilador pulará direto para a abertura do par de chaves do `senao` e executará o trecho de código escrito dentro do par de chaves.

7.4 EXERCÍCIOS

13. (F) Crie um programa que solicite ao usuário um número entre 1 e 10. Se o número informado for 5, apresente a mensagem "Você acertou!". Caso contrário, o programa não apresenta nenhuma mensagem.

14. (F) Crie um programa que solicite a velocidade de um carro e apresente a mensagem "MULTADO", caso a velocidade seja maior que 80.

15. (F) Crie um programa que solicite o nome, a nota 1 e a nota 2 de um(a) aluno(a). Em seguida, o programa deve calcular a média aritmética e, se o(a) aluno(a) ficar com nota maior ou igual a 6.0, o programa deve mostrar a mensagem "APROVADO(A)". Se a nota for menor que 6.0, o programa deve apresentar a mensagem

"EM RECUPERAÇÃO".

16. (M) Altere o exercício 15 para que seja solicitada a nota da recuperação, somente se o(a) aluno(a) tiver ficado em recuperação. Em seguida, o programa deve verificar se essa nota da recuperação é maior ou igual a 5.0. Se for, o programa deve mostrar a mensagem "APROVADO(A)", caso contrário deve mostrar a mensagem "REPROVADO(A)".

17. (F) Crie um programa para uma loja de sucos. O preço de cada suco é R\$ 5.50, porém, se o cliente comprar mais de 10 sucos, o preço individual passa para R\$ 4.50. O programa deve solicitar a quantidade de sucos desejados pelo cliente e apresentar o preço final a ser pago.

18. (F) Crie um programa para gerenciar uma fila de atendimento. O programa deve perguntar se a pessoa precisa de atendimento prioritário ou não. Se for respondido "sim", o programa deve mostrar a mensagem "Vá para os caixas 1, 2 e 3". Caso contrário, o programa deve mostrar a mensagem "Vá para qualquer caixa, exceto os 1, 2 e 3, que são prioritários."

19. (F) Crie um programa para calcular e informar se compensa mais abastecer um automóvel com gasolina ou com etanol. O programa deve solicitar ao usuário o preço da gasolina e, em seguida, o preço do etanol. Depois efetuar a divisão do preço do etanol pelo preço da gasolina. Se o resultado for maior ou igual a 0.7, o programa deve apresentar a mensagem "Compensa abastecer com gasolina". Caso contrário, o programa deve apresentar a mensagem "Compensa abastecer com etanol."

20. (F) Crie um programa que solicite um número inteiro e

apresente se ele é positivo ou negativo.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Como melhoria no sistema, deve-se apresentar agora um menu com as seguintes opções: 1 - Ver ofertas de bicicletas usadas e 2 - Ver ofertas de bicicletas novas. Essa exibição do menu deve aparecer depois de apresentar as mensagens já solicitadas no capítulo anterior. Ao selecionar o item 1 do menu, o sistema deve exibir: "Bicicleta usada na cor azul, aro 26, com 18 marchas e com o valor promocional de R\$ 400,00". Ao selecionar o item 2 do menu, o sistema deve exibir: "Bicicleta nova na cor amarela, aro 26, com 18 marchas e na promoção pelo preço de R\$ 999,99".

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap7>



Figura 7.1: QRCode

CAPÍTULO 8

CONDICIONAL MÚLTIPLA: SE SENAO SE

Neste capítulo, aprofundaremos um pouco mais nas estruturas condicionais. Vamos aprender sobre as condições múltiplas `se` `senao` `se`.

No capítulo anterior, vimos que a estrutura condicional composta possui dois trechos de códigos que podem ser executados. Um trecho de código é executado quando o resultado lógico da expressão é verdadeiro e o outro, quando o resultado lógico da expressão é falso. Dessa maneira, temos um trecho de código para cada um dos possíveis resultados da operação lógica.

Porém, em alguns casos, necessitamos realizar mais de duas comparações.

Vamos relembrar o exemplo anterior que verificava entre dois números qual era o maior. Vimos que, além de um número ser maior que o outro, temos o caso em que os dois números são iguais. Repare que agora temos três possíveis resultados (maior, menor ou igual).

Resolvemos esse exemplo no capítulo anterior verificando se os números `numero1` e `numero2` são diferentes e, se forem,

entramos no `se`. Dentro do `se`, temos uma nova verificação em que comparamos se o `numero1` é maior que o `numero2`. Se o resultado da condição lógica for verdadeiro, sabemos que o `numero1` é o maior entre os dois e, se o resultado for falso, sabemos que o `numero2` é o maior. Dessa maneira resolvemos o problema utilizando duas estruturas de condição composta (um `se` dentro de outro `se`).

Com o uso das estruturas condicionais múltiplas, resolveremos esse exemplo de uma forma mais simples.

SINTAXE DA ESTRUTURA CONDICIONAL MÚLTIPLA SE SENAO SE

```
se (condicao){  
} senao se ( condicao2 ){  
} senao se ( condicao3 ){  
} senao se ( condicaoN ){  
} senao {  
}
```

Nessa sintaxe, condicao será uma expressão relacional. Caso a expressão apresente como resultado o valor **verdadeiro**, os comandos escritos dentro da abertura e do fechamento da chave do se serão executados. Caso a expressão apresente como resultado o valor **falso**, o compilador pulará até o comando senao se . O comando senao agora tem um se .

Esse se é uma nova condição que será executada como foi a primeira. Se o resultado for **verdadeiro**, o compilador entrará no par de chaves da condição. Caso contrário, o compilador vai para o próximo senao se . E esse passo a passo se repetirá enquanto existir senao se na estrutura. É possível ter N senao se . Ao finalizar todos os senao se , existe um comando senao que só será executado se todos os resultados das condições anteriores forem falsos.

```

//Exemplo 38: condicional múltipla
//Sistema para verificar qual o maior entre dois números informados pelo teclado.
// Entrando com valores iguais pelo teclado
inteiro numero1, numero2

escreva("Digite o número 1: ")
leia(numero1)
escreva("Digite o número 2: ")
leia(numero2)

se (numero1 > numero2){
    escreva("Número 1 maior")
}senao se (numero1 < numero2){
    escreva ("Número 2 maior")
}senao{
    escreva("Números iguais")
}

```

Analisando o trecho de código, percebemos que o comando `senao` aparece para realizar uma nova comparação necessária no algoritmo. Primeiramente é verificado se o `numero1` é maior que o `numero2`; se for falso, ainda podemos ter que o `numero2` é maior que o `numero1`, ou então, que eles são iguais. Então temos que realizar pelo menos mais uma condição para deixar somente uma para o `senao`. No exemplo, verificamos se o `numero2` é maior que o `numero1`, porém poderíamos verificar se os dois eram iguais, que o nosso algoritmo funcionaria perfeitamente.

Se o resultado lógico da condição for falso novamente, o algoritmo vai para o próximo `senao` que, no exemplo 38, não tem mais um `se`, e ele então executará o conteúdo dentro do par de chaves.

Sempre que o resultado de uma condição for verdadeiro no `se` sozinho ou então em alguma `senao se`, o algoritmo entrará no respectivo par de chaves e depois sairá da estrutura condicional,

mesmo que ainda tenham diversos senao se abaixo. Isso acontece pois o algoritmo já encontrou um resultado verdadeiro e por isso não precisa verificar as demais condições. Perceba que as estruturas condicionais estão relacionadas umas com as outras.

É possível ter diversas estruturas condicionais no decorrer dos nossos códigos. Cada uma terá um objetivo e seguirá a mesma regra definida anteriormente.

Repare então que agora podemos realizar várias comparações de acordo com a necessidade do algoritmo utilizando todas as estruturas condicionais.

Agora que vimos como funciona a estrutura senao se , vamos pensar no exemplo a seguir.

Vamos pensar juntos: vamos imaginar o sistema de votação atual no Brasil. Ele é realizado de acordo com a idade das pessoas. As regras de votação são: a) Menores de 16 anos não podem votar; b) De 16 anos até 17 anos, o voto é facultativo; c) De 18 anos até 69 anos, o voto é obrigatório; d) Maiores que 70 anos, o voto é facultativo;

Escreva esse algoritmo somente com condicionais compostas e, em seguida, reescreva o mesmo algoritmo usando condicionais múltiplas.

```
//Exemplo 39: condicional composta  
//Sistema de votação
```

```
inteiro idade
```

```

escreva("Digite uma idade: ")
leia(idade)

se (idade<16){
    escreva("Não pode votar")
}senao{
    se(idade < 18){
        escreva("Voto opcional")
    }senao{
        se(idade < 70){
            escreva("Voto obrigatório")
        }senao{
            escreva("Voto opcional")
        }
    }
}

//Exemplo 40: condicional múltipla
//Sistema de votação

inteiro idade

escreva("Digite uma idade: ")
leia(idade)

se (idade<16){
    escreva("Não pode votar")
}senao se(idade < 18){
    escreva("Voto opcional")
}senao se(idade < 70){
    escreva("Voto obrigatório")
}senao{
    escreva("Voto opcional")
}

```

Comparando as resoluções dos exemplos 39 e 40, verificamos que os dois servem para resolver o problema do sistema de votação, porém cada um de uma maneira. O exemplo 39 utiliza seis comandos e o 40 utiliza quatro. Se imaginarmos um exemplo real, no qual podem existir dezenas de verificações, a diminuição de código que ganhamos com isso facilita o entendimento, a manutenibilidade e reduz o tempo de processamento.

O exemplo 40 começa verificando se a idade é menor que 16; caso ela seja igual ou maior, o resultado será falso e o algoritmo irá para o senao . O senao é seguido de uma nova comparação verificando se a idade é menor que 18.

Se o algoritmo chegou a essa segunda comparação, significa que o resultado da primeira foi falso, logo a idade então é maior ou igual a 16. Como na segunda comparação temos que a idade é menor que 18, ele só entrará no par de chaves dessa condição se a idade for 16 ou 17.

Se o resultado da segunda condição for falso, ele vai para o próximo senao , que também tem uma nova condição se . Dessa vez, é verificado se a idade é menor que 70. Como o algoritmo falhou na primeira e na segunda condição, significa que, até o momento, a idade informada é maior que 18. Como existe uma outra restrição de idade maior ou igual a 70, essa nova condição verifica se é menor que 70.

Se ela for verdadeira, significa que a idade digitada está no intervalo de 18 e 69. Caso contrário, significa que a idade é maior que 70 e, como não existe outra condição, ela é colocada em um senao sozinho sem nenhuma comparação.

As estruturas condicionais múltiplas também podem ser utilizadas para direcionar o algoritmo de acordo com uma opção escolhida em um menu de opções. Vamos ver como usar as estruturas condicionais múltiplas na montagem de um menu de opções.

```
//Exemplo 41: condicional múltipla  
//Sistema de opção de escolha de menu
```

```
inteiro opcao
```

```

escreva("Opções do menu: ")
escreva("1 - contratar serviço de internet \n")
escreva("2 - cancelar serviço de internet \n")
escreva("3 - alterar dados do usuário \n")
escreva("4 - emitir 2ª via do boleto \n")
escreva("5 - falar com um dos nossos atendentes \n")

leia(opcao)

se (opcao == 1){
    escreva("Você escolheu a opção 1 - Contratar serviço de internet")
    escreva("\n Atualmente temos dois planos de dados. Mega 75Mb por R$ 79,90 e Super fibra 100Mb por R$ 99,99.")
    escreva("\n Informe seu CPF, nome completo, CEP, telefone de contato e plano escolhido, e aguarde alguns instantes.")
}senao se (opcao == 2){
    escreva("Você escolheu a opção 2 - Cancelar serviço de internet")
    escreva("\n Informe seu CPF e o motivo do cancelamento.")
}senao se (opcao == 3){
    escreva("Você escolheu a opção 3 - Alterar dados do usuário.")
    escreva("\n Informe seu CPF e o novo dado a ser atualizado.")
}senao se (opcao == 4){
    escreva("Você escolheu a opção 4 - Emitir 2ª via do boleto.")
    escreva("\n Informe seu CPF e o mês desejado para emissão da 2ª via.")
}senao se (opcao == 5){
    escreva("Você escolheu a opção 5 - Falar com um dos nossos atendentes.")
    escreva("\n Todos os nossos atendentes estão ocupados no momento. Aguarde até 5 minutos que você será atendido.")
}senao{
    escreva("Opção inválida")
}

escreva ("Obrigado pelo contato.")

```

No exemplo 41, o algoritmo é utilizado para montar um sistema de escolhas de um menu. As opções disponíveis são

apresentadas e em seguida é realizada a leitura da opção desejada pelo usuário. Cada um dos itens de menu corresponde a uma condição na estrutura `se senao se`.

Veja que inicialmente é verificado, com o `se` sozinho, se a opção digitada foi 1. Se for, o programa entrará no par de chaves do `se` e, ao encontrar o fechamento do par, o programa sairá e apresentará a mensagem "Obrigado pelo contato", que não está vinculada a nenhuma condição. Caso não seja digitada a opção 1, o programa verificará se a opção digitada foi 2, nesse caso já será no `senao se`. Se for, será executado o conteúdo do par de chaves. Caso não seja, o programa verificará se a próxima opção do menu foi digitada.

Isso ocorrerá até o programa chegar ao último `senao`. O último `senao` está sozinho, ou seja, ele não tem uma condição `se` junto dele e então ele entrará diretamente já que não tem que realizar nenhuma comparação. O programa então executará o conteúdo dentro do par de chaves e em seguida sairá da estrutura condicional. Ele continuará executando o algoritmo e o próximo comando é apresentar a mensagem "Obrigado pelo contato".

Compare o algoritmo do exemplo 41 com o algoritmo apresentado a seguir.

```
//Exemplo 42: condicional simples e composta  
//Sistema de opção de escolha de menu
```

```
inteiro opcao
```

```
escreva("Opções do menu: ")  
escreva("1 - contratar serviço de internet \n")  
escreva("2 - cancelar serviço de internet \n")  
escreva("3 - alterar dados do usuário \n")  
escreva("4 - emitir 2ª via do boleto \n")
```

```

escreva("5 - falar com um dos nossos atendentes \n")

leia(opcao)

se (opcao == 1){
    escreva("Você escolheu a opção 1 - Contratar serviço de internet")
    escreva("\n Atualmente temos dois planos de dados. Mega 75Mb por R$ 79,90 e Super fibra 100Mb por R$ 99,99.")
    escreva("\n Informe seu CPF, nome completo, CEP, telefone de contato e plano escolhido, e aguarde alguns instantes.")
}
se (opcao == 2){
    escreva("Você escolheu a opção 2 - Cancelar serviço de internet")
    escreva("\n Informe seu CPF e o motivo do cancelamento.")
}
se (opcao == 3){
    escreva("Você escolheu a opção 3 - Alterar dados do usuário.")
    escreva("\n Informe seu CPF e o novo dado a ser atualizado.")
}
se (opcao == 4){
    escreva("Você escolheu a opção 4 - Emitir 2ª via do boleto.")
    escreva("\n Informe seu CPF e o mês desejado para emissão da 2ª via.")
}
se (opcao == 5){
    escreva("Você escolheu a opção 5 - Falar com um dos nossos atendentes.")
    escreva("\n Todos os nossos atendentes estão ocupados no momento. Aguarde até 5 minutos que você será atendido.")
}senao{
    escreva("Opção inválida")
}

escreva ("Obrigado pelo contato.")

```

Diferentemente do algoritmo do exemplo 41, que utiliza condicional múltipla, o exemplo 42 utiliza quatro (4) estruturas se simples e, ao final, uma estrutura se senão . Qual seria o

resultado da execução desse exemplo?

Aqui, como as estruturas não estão relacionadas, elas não são excludentes. Ou seja, mesmo que a opção inserida no teclado tenha sido 1, todas as condições serão verificadas. Além disso, o único senão utilizado está associado apenas à condição `opcao == 5`. Dessa forma, para todos os valores de opção diferentes de 5, a mensagem `Opção inválida` será apresentada.

A utilização de vários `se` sequenciais pode ser necessária em casos onde as variáveis que estão sendo analisadas não possuem relação. Ou seja, quando o resultado de uma condição não influenciar na análise de outras condições.

Resumo do capítulo

Vimos que as estruturas condicionais múltiplas `se` `senao` `se` são utilizadas quando necessitamos realizar mais de duas comparações em uma mesma estrutura. Pode-se ter na estrutura apenas um único `se` no início, seguido de vários `senao` `se` e um único `senao` no final. O funcionamento de cada condição funciona exatamente como visto nos capítulos anteriores, ou seja, só é possível obter os resultados verdadeiro ou falso em cada uma das condições. Ao encontrar um resultado verdadeiro, o algoritmo executa o trecho de código dentro do par de chaves da condição verdadeira e, em seguida, sai da estrutura. Caso não encontre nenhum verdadeiro, o algoritmo entra no `senao` sozinho, se existir, e depois sai da estrutura.

8.1 EXERCÍCIOS

21. (F) Crie um programa que solicite ao usuário um número e apresente na tela qual é o dia da semana do respectivo número. Considere que os números fornecidos devem estar no intervalo entre 1 e 7. Considere que 1 é domingo, 2 é segunda e assim por diante.

22. (F) Crie um programa para uma loja de sucos no qual são oferecidos os seguintes sucos: L - Laranja, M - Morango, A -

Acerola e U - Uva. O usuário deve informar uma letra e o sistema apresentará o nome do suco e qual a principal vitamina que o suco fornece, são elas: laranja vitamina C, morango vitamina A, acerola vitamina C e uva vitamina E.

23. (F) Crie um programa que solicite ao usuário a estação do ano desejada, e o sistema deve apresentar o dia que começa a estação, são elas: outono - 20 de março, inverno - 21 junho, primavera - 22 setembro e verão - 21 de dezembro.

24. (F) Crie um programa que solicite ao usuário uma vogal e apresente palavras de acordo com a vogal informada.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Com o crescimento da loja, o sistema deve apresentar no menu outras duas opções que são os itens 3 - Ver ofertas de acessórios e 4 - Ver novos serviços.

Ao selecionar o item 3 do menu, o sistema deve exibir: "Acessório em oferta - Capacete de proteção por R\$59,99" e, ao selecionar o item 4 do menu, o sistema deve exibir: "Novos serviços oferecidos: Lavagem completa da sua bicicleta por R\$ 12,99 | Manutenção dos freios por R\$ 10,99 | Troca de pneus por R\$ 55,99".

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap8>



Figura 8.1: QRCode

CAPÍTULO 9

OPERADORES LÓGICOS

Até o momento, todas as condições construídas utilizaram apenas uma única comparação. No capítulo anterior, verificamos se uma idade era maior que 18 ou se um número era igual a 0. Porém, em alguns problemas necessitamos realizar duas ou mais comparações em uma mesma condição. Por exemplo, precisamos verificar se uma idade é maior que 18 e, ao mesmo tempo, se também é menor que 25. Repare que temos duas comparações para serem realizadas.

Para esse tipo de comparação, utilizaremos um novo tipo de operador, chamado de **operador lógico**.

Vamos pensar juntos: monte um algoritmo para verificar se uma pessoa está apta para realizar a prova da carteira de habilitação para motos. O(A) candidato(a) precisa ter 18 anos e ter assistido a 45 horas de aula teórica e 20 horas de aula prática.

Esse algoritmo pode ser escrito usando o encadeamento de comandos `se .`. Primeiramente verificamos a idade do(a) candidato(a); em seguida, se ele(a) cumpriu as 45 horas de aula teórica e, por fim, se cumpriu as 20 horas de aula prática. Veja a

seguir como poderia ser construído.

```
//Exemplo 43:  
//Verificando se uma pessoa está apta para prova para carteira de  
//habilitação para motos.  
  
inteiro idade  
real horasTeoricas, horasPraticas  
  
escreva("Digite a idade do(a) candidato(a): ")  
leia(idade)  
  
escreva("Digite a quantidade de horas teóricas assistida:")  
leia(horasTeoricas)  
  
escreva("Digite a quantidade de horas práticas assistida:")  
leia(horasPraticas)  
  
se (idade>=18){  
    se( horasTeoricas >= 45.0){  
        se(horasPraticas >= 20.0){  
            escreva("Apto(a)")  
        }senao{  
            escreva("Não apto(a)")  
        }  
    }senao{  
        escreva("Não apto(a)")  
    }  
}senao{  
    escreva("Não apto(a)")  
}  
}
```

Repare que conseguimos construir o algoritmo, porém ele ficou muito encadeado. Isso é um problema para manutenção do código quando tivermos que realizar diversas comparações. Podemos reescrever esse algoritmo, escrito em 13 linhas, em apenas 4 linhas e, para isso, teremos que usar os operadores lógicos.

Operadores lógicos são utilizados para juntar duas expressões lógicas. Seu objetivo é juntar dois valores lógicos e apresentar

apenas um resultado lógico de saída.

Para isso, precisamos conhecer a **TABELA VERDADE** dos operadores lógicos. O resultado lógico de saída dependerá exclusivamente do tipo de operador lógico utilizado e dos valores lógicos de entrada.

9.1 TABELA VERDADE

A **TABELA VERDADE** apresenta o resultado final da comparação lógica entre duas entradas lógicas. Vimos que a saída de qualquer condição lógica só pode ser **verdadeiro** ou **falso**. Essa regra continua valendo. O que verificaremos agora será qual o resultado lógico final quando temos a combinação de duas entradas lógicas.

No exemplo 43, podemos ter um candidato com mais de 18 anos e que ainda não cumpriu as 45 horas de aula teórica. Ou seja, temos um resultado **verdadeiro** (idade maior que 18) e um resultado **falso** (não cumpriu as 45 horas de aula teórica). Nesse caso, teremos que consultar a tabela verdade para saber qual o resultado final quando tivermos uma entrada **verdadeiro** e uma **falso**.

Neste livro, abordaremos apenas os dois principais operadores lógicos, o **e** e o **ou**. Cada um deles é utilizado em uma determinada situação.

Operador e

O operador **e** é utilizado quando necessitamos que as duas entradas sejam **verdadeiras** para que a saída final também seja

`verdadeira` . Se uma das duas entradas for `falsa` , a saída também será `falsa` .

Esse é o caso do exemplo 43, já que para realizar a prova é necessário que o candidato tenha mais que 18 anos (`verdadeiro`) e tenha cumprido as 45 horas de aula teórica (também `verdadeiro`). No exemplo, o candidato ainda tem que cumprir 20 horas de aula prática. Como temos três comparações, o algoritmo separará as duas primeiras entradas e a saída dessa comparação será uma das entradas para a próxima entrada. Veja o passo a passo:

- Primeiro: o candidato tem mais de 18 e cumpriu as 45 horas de aula teórica.
- Segundo: o resultado lógico da primeira comparação e o candidato cumpriu as 20 horas práticas.

Vamos simular com os seguintes valores:

a) Candidato com 25 anos, cumpriu 46 horas teóricas e cumpriu 15 horas práticas, logo temos os seguintes valores lógicos: `verdadeiro` , `verdadeiro` e `falso` .

Primeiro: o candidato tem mais de 18 e cumpriu as 45 horas de aula teórica. `verdadeiro` e `verdadeiro` terá saída `verdadeiro` .

Segundo: o resultado lógico da primeira comparação e o candidato cumpriu as 20 horas práticas. `verdadeiro` e `falso` terá como saída `falso` .

b) Candidato com 32 anos, cumpriu 25 horas teóricas e cumpriu 10 horas práticas, logo temos os seguintes valores lógicos:

verdadeiro , falso e falso .

Primeiro: o candidato tem mais de 18 e cumpriu as 45 horas de aula teórica. **verdadeiro e falso** terá saída **falso** .

Segundo: o resultado lógico da primeira comparação e o candidato cumpriu as 20 horas práticas. **falso e falso** terá como saída **falso** .

c) Candidato com 15 anos, cumpriu 30 horas teóricas e cumpriu 18 horas práticas, logo temos os seguintes valores lógicos: **falso , falso e falso** .

Primeiro: o candidato tem mais de 18 e cumpriu as 45 horas de aula teórica. **falso e falso** terá saída **falso** .

Segundo: o resultado lógico da primeira comparação e o candidato cumpriu as 20 horas práticas. **falso e falso** terá como saída **falso** .

Vamos imaginar agora a seguinte situação: se no final de semana fizer sol e você não tiver prova, você vai à praia. Perceba que para você ir à praia é necessário que as duas condições sejam verdadeiras (ter sol e não ter prova). Se uma das duas falhar, ou seja, tiver **falso** como resultado, você não vai à praia, mesmo que o outro seja **verdadeiro** . Esse é o operador **e** .

entrada 1	entrada 2	entrada1 e entrada2
V	V	V
V	F	F
F	V	F
F	F	F

Essa tabela pode ser estendida para quaisquer quantidades de entradas. De uma forma geral, quando utilizamos o operador `e` , o resultado lógico final só será verdadeiro se todas as condições unitárias forem verdadeiras. Se, dentre o conjunto de entradas, pelo menos uma for falsa, o resultado lógico será falso.

Operador ou

Continuando no exemplo da praia, com o operador `ou` , para que você vá à praia, basta que uma das duas condições seja verdadeira . Ou você vai à praia pois está sol, independentemente de você ter prova ou não; ou você vai à praia pois não tem prova, independentemente de estar sol ou não. Ou seja, você só não vai à praia se não fizer sol e ao mesmo tempo você tiver prova.

No `ou` , o resultado final só será `falso` quando as duas entradas forem `falso` .

No exemplo da prova da carteira de motorista, basta que uma das condições seja verdadeira para que o candidato tenha direito a fazer a prova. Portanto, se a condição para tirar a carteira fosse `ou` em vez de `e` , teríamos a seguinte situação.

Vamos simular com os seguintes valores:

a) Candidato com 25 anos, cumpriu 46 horas teóricas e cumpriu 15 horas práticas, logo temos os seguintes valores lógicos: `verdadeiro` , `verdadeiro` e `falso` .

Primeiro: o candidato tem mais de 18 ou cumpriu as 45 horas de aula teórica. `verdadeiro` ou `verdadeiro` terá saída `verdadeiro` .

Segundo: o resultado lógico da primeira comparação ou o candidato cumpriu as 20 horas práticas. verdadeiro ou falso terá como saída verdadeiro .

b) Candidato com 32 anos, cumpriu 25 horas teóricas e cumpriu 10 horas práticas, logo temos os seguintes valores lógicos: verdadeiro , falso e falso .

Primeiro: o candidato tem mais de 18 ou cumpriu as 45 horas de aula teórica. verdadeiro ou falso terá saída verdadeiro .

Segundo: o resultado lógico da primeira comparação ou o candidato cumpriu as 20 horas práticas. verdadeiro ou falso terá como saída verdadeiro .

c) Candidato com 15 anos, cumpriu 30 horas teóricas e cumpriu 18 horas práticas, logo temos os seguintes valores lógicos: falso , falso e falso .

Primeiro: o candidato tem mais de 18 ou cumpriu as 45 horas de aula teórica. falso ou falso terá saída falso .

Segundo: o resultado lógico da primeira comparação ou o candidato cumpriu as 20 horas práticas. falso ou falso terá como saída falso .

entrada 1	entrada 2	entrada1 ou entrada2
V	V	V
V	F	V
F	V	V
F	F	F

SINTAXE DE UMA EXPRESSÃO LÓGICA

Operador e

(condicao1 e condicao2)

Operador ou

(condicao1 ou condicao2)

Nessa sintaxe, condicao1 deve ser uma expressão relacional e condicao2 deve ser outra expressão relacional. O compilador executará primeiramente cada uma das condições de forma separada. Em seguida, já com os resultados das condições, o compilador verificará o resultado final na tabela verdade do operador lógico.

Por exemplo, imagine que a condicao1 teve como resultado o valor lógico verdadeiro . A condicao2 também teve como saída o valor lógico verdadeiro . O compilador então montará uma expressão lógica entre verdadeiro e verdadeiro . De acordo com a tabela verdade, o operador lógico e resulta em verdadeiro quando tem como entrada dois verdadeiros. Já se as entradas tivessem sido verdadeiro e falso , o resultado final da expressão lógica seria falso (de acordo com a tabela verdade do e).

Agora que já vimos os dois operadores lógicos, vamos alterar o exercício 43 usando o operador lógico e . Usaremos o e porque no enunciado fica claro que, para realizar a prova, o candidato necessita que as três condições sejam verdadeiras (ter idade maior

ou igual a 18, ter cumprido, no mínimo, 45 aulas teóricas e, no mínimo, 20 aulas práticas).

```
//Exemplo 44:  
//Verificando se uma pessoa está apta para prova para carteira de  
//habilitação para motos, utilizando operadores lógicos.  
  
inteiro idade  
real horasTeoricas, horasPraticas  
  
escreva("Digite a idade do(a) candidato(a): ")  
leia(idade)  
  
escreva("Digite a quantidade de horas teóricas assistida:")  
leia(horasTeoricas)  
  
escreva("Digite a quantidade de horas práticas assistida:")  
leia(horasPraticas)  
  
se ((idade>=18 e horasTeoricas >= 45.0) e horasPraticas >= 20.0))  
{  
    escreva("Apto(a)")  
}senao{  
    escreva("Não apto(a)")  
}
```

Repare que, ao utilizarmos os operadores lógicos, a quantidade de linhas diminuiu drasticamente. Essa diminuição simplifica o entendimento da lógica, além de facilitar a manutenção do código.

9.2 USO DE PARÊNTESES

O uso dos parênteses nas expressões lógicas serve para indicar a ordem de precedência da operação. A regra do seu uso é a mesma utilizada na matemática, com os operadores matemáticos. Ou seja, primeiro o compilador executará o que estiver dentro dos parênteses mais internos até chegar aos externos.

O uso dos parênteses é importante pois, dependendo do tipo do operador lógico usado, o resultado da expressão pode ser totalmente diferente.

A seguir veremos como ficaria o exemplo da praia usando o operador lógico e .

```
//Exemplo 45:  
//Você vai à praia se o tempo estiver com sol e você não tiver prova no final de semana.  
  
cadeia previsaoTempo  
caracter temProva  
  
escreva("Digite a previsão do tempo para o final de semana (sol ou chuva): ")  
leia(previsaoTempo)  
  
escreva("Digite s ou n para indicar se terá prova no final de semana:")  
leia(temProva)  
  
se (previsaoTempo == "sol" e temProva == 'n'){  
    escreva("Ólá, você vai à praia no final de semana.")  
}senao{  
    escreva("Infelizmente você não vai à praia no final de semana.")  
}
```

Agora, se para você ir à praia bastasse que uma das duas entradas fosse verdadeira , utilizariamos o operador lógico ou e o programa ficaria conforme a seguir.

```
//Exemplo 46:  
//Você vai à praia se o tempo estiver com sol ou você não tiver prova no final de semana.  
  
cadeia previsaoTempo  
caracter temProva  
  
escreva("Digite a previsão do tempo para o final de semana (sol o
```

```

u chuva): ")
leia(previsaoTempo)

escreva("Digite s ou n para indicar se terá prova no final de sem
ana:")
leia(temProva)

se (previsaoTempo == "sol" ou temProva == 'n'){
    escreva("Obá, você vai à praia no final de semana.")
}senao{
    escreva("Infelizmente você não vai à praia no final de semana
.")
}

```

Repare que a única diferença entre os exemplos 45 e 46 é o operador `ou` e o `e`. Porém, o que vai mudar mesmo é o resultado da operação, de acordo com os valores fornecidos pelo usuário. No caso do operador `e`, para que o usuário vá à praia é necessário que as duas entradas sejam verdadeiras (tem que fazer sol e não pode ter prova). Se estiver usando o operador `ou`, para que o usuário vá à praia é necessário que apenas uma das duas entradas seja verdadeira (ou fará sol ou o usuário não terá prova).

Resumo do capítulo

Vimos os operadores lógicos e as expressões lógicas, que são utilizadas quando temos que realizar duas ou mais comparações em uma mesma condição. Em seguida, vimos que é através da **TABELA VERDADE** que sabemos qual a saída lógica para cada par de entradas lógicas. O operador `e` só tem como saída o resultado lógico `verdadeiro` quando as duas entradas são `verdadeiras`. Em todas as outras possíveis entradas, a saída sempre será `falso`. Já no operador `ou` basta termos uma condição de entrada `verdadeira` que a saída sempre será `verdadeira`. O `ou` só tem a saída `falso` quando as duas entradas forem `falso` também.

9.3 EXERCÍCIOS

25. (F) Crie um programa que solicite ao usuário um peso e uma altura, e apresente na saída o valor do IMC e um dos

seguintes indicadores, são eles: IMC menor que 18.5 - magreza -, IMC entre 18.5 e 24.9 - normal, IMC entre 24.9 e 30 - sobre peso e IMC maior que 30 - obesidade. A fórmula para o cálculo é IMC = peso / (altura * altura).

26. (M) Crie um programa que verifique se um candidato está apto a tirar a carteira de motorista do tipo D. Os requisitos são: ter idade maior que 21 anos; estar habilitado pelo menos dois anos com a carteira B ou um ano com a carteira C; não ter nenhuma infração nos últimos doze meses.

27. (T) Crie um programa para calcular o desconto de acordo com os itens comprados em uma padaria. Se o cliente comprar 10 pães e mais um queijo, ele ganha 10% de desconto. Se o cliente comprar uma bisnaga ou um pão de forma, ele tem um desconto de 15%. Agora se o cliente comprar leite e pão doce ou suspiro, ele ganha 5% de desconto. Os preços dos produtos devem ser definidos por você. O desconto não é acumulativo e será aplicado o maior percentual, de acordo com as regras, uma única vez no final da compra.

28. (M) Crie um programa para calcular a média aritmética de um aluno em um bimestre. Seu programa deve pedir a nota do teste, a nota da prova e a quantidade de faltas do aluno. Se o aluno tiver a média maior ou igual a 7.0 e menos que 10 faltas, ele estará aprovado. Se o aluno tiver média entre 5.0 e 6.9 e menos que 10 faltas, ele estará em recuperação. Se o aluno tiver média menor que 5.0 ou mais que 10 faltas, ele estará reprovado.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Com o crescimento da loja, o sistema deve apresentar no menu outras três opções que são os itens 5 - Promoção I 10% de desconto e 6 - Promoção II 20% de desconto.

Descrição da promoção I: Lave sua bicicleta (R\$ 12,99) e realize manutenção no freio (R\$ 10,99) com desconto de 10% no total do pagamento. Descrição da promoção II: Troque um pneu da bicicleta (R\$ 55,99) e realize a manutenção nos freios (R\$ 10,99) com 20% de desconto no total do pagamento.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap9>



Figura 9.1: QRCode

CAPÍTULO 10

CONDICIONAL: ESCOLHA CASO

Agora veremos a última estrutura condicional que podemos utilizar nos nossos programas. Ela tem três características principais para o seu uso: (a) só pode ser usada com os tipos `inteiro` ou `caracter`; (b) só pode ser utilizada para verificar igualdade; (c) não pode ser utilizada com os operadores relacionais e lógicos;

Dessa maneira, a estrutura escolha caso acaba tendo sua utilização bem restrita. Normalmente ela é utilizada para verificar escolhas de itens de menu. Vejamos a sintaxe da estrutura.

SINTAXE DO ESCOLHA CASO

```
escolha (opcao){  
    caso 1:  
  
        pare  
  
    caso 2:  
  
        pare  
  
    caso 50:
```

```
    pare  
  
    caso contrario:  
}
```

Aqui, opcao é a variável que será utilizada na comparação. Os itens que serão comparados fazem parte dos casos . No exemplo, temos a comparação com os valores 1, 2 e o 50. Inicialmente o compilador verificará se o valor que está na variável opcao é igual a 1. Se for, ele executará os comandos que aparecem depois dos dois pontos do caso até o comando pare e em seguida ele sairá do escolha caso . Caso a opção não seja 1, o compilador comparará com o próximo caso , que no exemplo é 2. Se for igual, ele executará os comandos até encontrar o pare e depois sairá do escolha caso . Na sequência, verificará os próximos casos. Se o valor for diferente de 1, 2 e 50, o compilador executará os comandos dentro caso contrario até encontrar o fechamento da chave do escolha caso . Ou seja, o caso contrario funciona como o senao das estruturas condicionais se .

A comparação realizada pelo compilador na estrutura condicional escolha caso é igual às realizadas nas estruturas anteriores, isto é, o compilador realiza a comparação entre dois valores e o resultado sempre será verdadeiro ou falso .

```
//Exemplo 47:  
// Programa para escolha do sabor do sorvete, utilizando tipo de  
variável inteiro.  
  
inteiro tipoSorvete
```

```

escreva("Sabores dos sorvetes de frutas\n")
escreva("Código 1 - Sabor: Uva \n")
escreva("Código 2 - Sabor: Morango \n")
escreva("Código 3 - Sabor: Manga \n")
escreva("Código 4 - Sabor: Amora \n")

escreva("Digite o código do sabor (1 até 4):")
leia(tipoSorvete)

escolha(tipoSorvete){
    caso 1:
        escreva ("Sorvete de Uva - 70 calorias")
    pare
    caso 2:
        escreva ("Sorvete de Morango - 70 calorias")
    pare
    caso 3:
        escreva ("Sorvete de Manga - 71 calorias")
    pare
    caso 4:
        escreva ("Sorvete de Amora - 54 calorias")
    pare
    caso contrario:
        escreva("Código inválido")
}

```

No exemplo 47 foi utilizado o tipo `inteiro` para realizar a comparação dos códigos. Porém, como dito anteriormente, além do tipo `inteiro`, o comando `escolha` caso aceita o tipo `caracter`. Veja como o exemplo 47 poderia ser escrito utilizando o tipo `caracter` em vez do tipo `inteiro`.

```
//Exemplo 48:
// Programa para escolha do sabor do sorvete, utilizando tipo de
variável caracter.
```

```

caracter tipoSorvete

escreva("Sabores dos sorvetes de frutas\n")
escreva("Código A - Sabor: Uva \n")
escreva("Código B - Sabor: Morango \n")
escreva("Código C - Sabor: Manga \n")

```

```

escreva("Código D - Sabor: Amora \n")
escreva("Digite o código do sabor (A até D):")

leia(tipoSorvete)

escolha(tipoSorvete){
    caso 'A':
        escreva ("Sorvete de Uva - 70 calorias")
    pare
    caso 'B':
        escreva ("Sorvete de Morango - 70 calorias")
    pare
    caso 'C':
        escreva ("Sorvete de Manga - 71 calorias")
    pare
    caso 'D':
        escreva ("Sorvete de Amora - 54 calorias")
    pare
    caso contrario:
        escreva("Código inválido")
}

```

Comparando os exemplos 47 e 48, percebe-se que o funcionamento do comando é igual, independentemente do tipo de variável que é utilizada. Perceba que o valor especificado no caso deve vir entre aspas simples, por se tratar de comparação de caractere.

Resumo do capítulo

Vimos a última estrutura condicional existente chamada de `escolha caso`. Diferente das anteriores, o `escolha caso` só pode ser usado com os tipos de variáveis `inteiro` e `caracter`. Além disso, ela só pode ser utilizada para realização de comparação de igualdade e não pode usar nenhum operador lógico.

10.1 EXERCÍCIOS

29. (F) Crie um programa que solicite ao usuário um número entre 1 e 12 e apresente na tela o mês correspondente.

30. (M) Crie um programa que solicite uma letra ao usuário e diga se é uma vogal ou não vogal.

31. (M) Crie um programa que solicite o tamanho de uma blusa (P, M e G) e apresente o tamanho da blusa solicitada. (P: 0.46 X 0.55 - M: 0.51 X 0.56 - G: 0.52 X 0.58)

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Com o crescimento da loja, Pedro distribuiu diversos computadores para que os clientes pudessem consultar o sistema de qualquer local da loja. Além disso, ele solicitou que o sistema fosse alterado de forma que os funcionários pudessem gerar uma ordem de serviço (OS) de qualquer computador. Inicialmente Pedro solicitou que o sistema perguntasse na tela principal se o acesso seria como cliente ou funcionário. Se o usuário escolher cliente, o sistema deve funcionar como funcionava até essa nova versão. Se o usuário escolher funcionário, o sistema deve solicitar o código de acesso.

Para não ter uma pergunta a mais para o cliente, o sistema foi desenvolvido da seguinte maneira: se no nome do cliente for digitado `xptorestrito`, o sistema abre o módulo para abertura de OS. Caso o texto seja diferente, ou seja, qualquer outro nome, o sistema funciona apresentando o menu como anteriormente.

O sistema deve solicitar ao funcionário as seguintes

informações:

- a) O cliente lavou a bicicleta? Digite S ou N.
- b) O cliente trocou o pneu da bicicleta? Digite S ou N.
- c) O cliente realizou manutenção nos freios? Digite S ou N.

Ao final, o sistema deve apresentar o valor com os descontos, se for o caso, da ordem de serviço.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap10>



Figura 10.1: QRCode

Estruturas de repetição

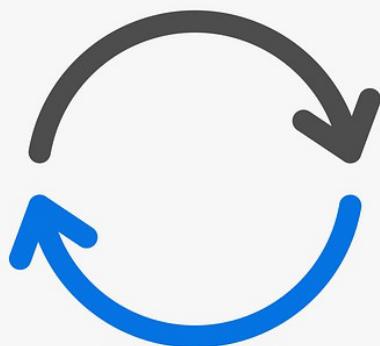


Figura 1: Fonte: Imagem de inspire-studio no Pixabay.

Nestes capítulos, estudaremos as estruturas de repetição. Elas são responsáveis por repetir um determinado trecho de código enquanto uma condição for verdadeira . Estudaremos os comandos para , enquanto e faca enquanto .

CAPÍTULO 11

ESTRUTURA DE REPETIÇÃO: PARA

As estruturas de repetição são utilizadas quando necessitamos repetir várias vezes um mesmo trecho de código no nosso algoritmo. Em vez de montarmos o algoritmo com um trecho de código repetindo diversas vezes, colocamos um único trecho de código dentro de uma estrutura de repetição e indicamos quantas vezes ele deve ser repetido.

O próprio algoritmo então passa a controlar quantas vezes o trecho de código deve ser repetido durante a execução do programa. Ficou meio confuso? Espere mais um pouco que veremos na prática esse conceito.

Vejamos o exemplo a seguir para entendermos o uso da estrutura de repetição no site da Casa do Código.

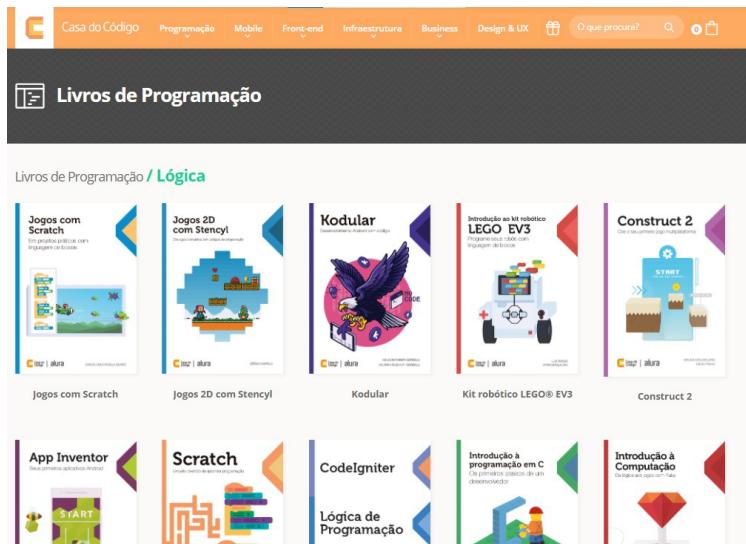


Figura 11.1: Fonte: Imagem de inspire-studio no Pixabay.

Repare na imagem do site da Casa do Código em que são exibidos diversos livros no item Lógica, dentro do tópico Programação. Todos aparecem com a imagem do livro e, logo abaixo da imagem, o nome do livro. São exibidos 5 livros em cada linha e, em seguida, começa-se uma nova linha novamente com 5 livros. Navegue nos outros tópicos do site e veja que o mesmo padrão (imagem do livro e nome abaixo) se repete.

Provavelmente o programador não precisou criar seu algoritmo para listar 50 ou 100 livros, ele criou o algoritmo para mostrar apenas um livro e em seguida mandou o algoritmo rodar a quantidade de vezes necessária em cada tópico. Esse algoritmo, provavelmente, ainda tem uma outra lógica para dizer que, depois de mostrar 5 livros em uma linha, ele deve apresentar mais 5 livros na linha seguinte.

Essa lógica de exibição de N livros em cada tópico só foi possível de ser construída com o uso das estruturas de repetição. Esse mesmo funcionamento pode ser observado em outros sites da internet, por exemplo: sites de compras, sites de buscas, redes sociais, entre outros.

11.1 COMANDO PARA

A primeira estrutura de repetição que estudaremos será o comando `para`. Ele é utilizado sempre que soubermos a quantidade de vezes que devemos repetir um certo trecho de código.

Por exemplo, desenvolver um programa para ler a idade dos 40 alunos de uma sala, criar um algoritmo para verificar se as 82 vagas de um estacionamento estão ocupadas, desenvolver um algoritmo para imprimir a tabuada do número 5, entre outros. Observe que em todos os exemplos apresentados temos um número predefinido no enunciado e esse número será utilizado na estrutura de repetição.

Vamos pensar juntos: analise o exemplo a seguir, juntamente à saída no console, e veja se consegue entender como a estrutura de repetição `para` funciona.

```
//Exemplo 49: algoritmo com a estrutura de repetição para
íntero contador
escreva("Início do programa")
```

```
para (contador = 1; contador <= 5; contador++){
    escreva ("\\nBom dia")
}

//Saída no console
Início do programa
Bom dia
Bom dia
Bom dia
Bom dia
Bom dia
```

Podemos observar que antes de entrar na estrutura de repetição para temos a declaração de uma variável contador e, em seguida, um comando escreva com o texto "Início do programa". Veja que o texto é exibido no console. Depois, temos a estrutura de repetição para com 3 parâmetros entre os parênteses (contador = 1 ; contador <= 5 ; contador++).

Dentro do par de chaves do comando para , temos um comando escreva com o texto "Bom dia". Veja que o comando escreva com o texto "Bom dia" só está escrito apenas uma única vez no algoritmo.

Ao analisar a saída do programa no console, vemos que, após o texto "Início do programa", aparecem 5 vezes o texto "Bom dia". Ou seja, mesmo só tendo escrito apenas uma única vez o comando escreva("\\nBom dia") , o algoritmo o repetiu 5 vezes na hora da execução do código. Esse é o funcionamento da estrutura de repetição, repetir N vezes o trecho de código que estiver escrito dentro do par de chaves da estrutura.

E por que o texto "Bom dia" apareceu 5 vezes e não 10 ou 100? A resposta é que a quantidade de repetições é definida pelo **contador** que é configurado nos 3 parâmetros do comando para .

A explicação de cada um dos parâmetros é realizada na sintaxe do comando `para`.

SINTAXE DO `PARA`

```
para (inicialização ; condição ; atualização){  
}
```

Nela, os três parâmetros dentro do par de parênteses são: a inicialização, a condição e a atualização. Esses parâmetros são configurados com uma variável chamada de **contador**. O contador é responsável por controlar a quantidade de repetições que o comando deve repetir.

Na inicialização, indicamos com qual valor o contador será iniciado. Imagine que utilizaremos uma variável contadora chamada `cont`. Veja alguns exemplos do parâmetro de inicialização: `cont= 0`, `cont= 35`, `cont = -50`.

Em seguida, temos a condição. Assim como nas estruturas condicionais, vistas nos capítulos anteriores, a condição serve para verificar se a repetição deve continuar ou não. O resultado da condição sempre será `verdadeiro` ou `falso`. Se for `verdadeiro`, a estrutura continuará repetindo e se for `falso`, ela sairá do comando e continuará a execução do algoritmo na primeira linha abaixo do fechamento da chave "`}`" do `para`.

Para testar as condições, serão utilizados os operadores relacionais, vistos nos capítulos anteriores. Exemplos de condição: `cont < 20` , `cont >= 18`, `cont <= 0`.

O último parâmetro é a atualização. Esse parâmetro vai indicar como o contador será atualizado após a execução de um ciclo. Ele pode ter seu valor subindo ou diminuindo. Exemplos: cont = cont +1 , cont = cont + 50, cont = cont - 3.

Quando queremos adicionar apenas um item ao valor já existente do contador, podemos utilizar a escrita simplificada. Em vez de escrevermos cont = cont + 1 , podemos escrever cont++ . Essa simplificação só pode ser usada para adicionar uma unidade ao valor da variável contadora. Se for necessário adicionarmos mais de uma unidade, devemos usar a forma extensa cont = cont + 2 , por exemplo.

A soma de apenas um elemento é chamada de **incremento** (cont++) e a subtração de apenas um elemento é chamado de **decremento** (cont--). O trecho de código que será repetido deve ser escrito dentro da abertura da chave e fechamento da chave do comando para {}.

Vejamos mais um exemplo.

```
//Exemplo 50: Algoritmo para imprimir a tabuada de 3

inteiro contador

para (contador = 1; contador <= 10; contador++){
    escreva ( contador * 3)
    escreva ( "\n")
}

//Saída no console
3
6
9
12
```

15
18
21
24
27
30

Vamos pensar juntos: altere o exemplo 50 para que o algoritmo apresente qual valor está sendo multiplicado em cada linha do resultado. Exemplo:

```
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
...
3 x 10 = 30
```

```
//Exemplo 51: Algoritmo para imprimir a tabuada de 3 com a saída
formatada
```

```
inteiro contador

para (contador = 1; contador <= 10; contador++){
    escreva ( " 3 x ", contador, " = ", contador * 3)
    escreva ("\n")
}

//Saída no console
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Veja que, para realizar a alteração na mensagem de saída do

algoritmo, formatamos apenas o texto do comando `escreva`. Os cálculos não precisaram ser alterados.

Conforme apresentado na sintaxe, os três parâmetros do comando `para` são controlados por um contador. Esse contador pode ser configurado de acordo com a lógica desejada no algoritmo. Além disso, o contador pode ser de diversos tipos de dados.

Vamos pensar juntos: quais alterações seriam necessárias nos parâmetros para que o exemplo 51 apresentasse a tabuada do número 3 de forma decrescente?

Exemplo:

```
3 x 10 = 30
3 x 9 = 27
3 x 8 = 24
3 x 7 = 21
...
3 x 1 = 3
```

//Exemplo 52: Apresentar a tabuada de 3 de forma decrescente.

```
inteiro contador

para (contador = 10; contador >= 1 ; contador--){
    escreva ( " 3 x ", contador, " = ", contador * 3)
    escreva ("\n")
}

//Saída no console
3 x 10 = 30
3 x 9 = 27
3 x 8 = 24
3 x 7 = 21
3 x 6 = 18
```

$3 \times 5 = 15$
 $3 \times 4 = 12$
 $3 \times 3 = 9$
 $3 \times 2 = 6$
 $3 \times 1 = 3$

Veja que realmente só precisamos alterar os parâmetros do comando para para ter a saída decrescente. No exemplo 51, o contador inicia do 1 e vai incrementando (aumentando de um em um) até chegar ao 10.

No exemplo 52, queríamos que o algoritmo escrevesse essa mesma tabuada só que de forma decrescente. O contador então foi alterado para iniciar do valor 10 e a condição foi alterada para `contador >= 1`. Também foi necessário realizar a alteração na condição, pois é ela que delimita quando a repetição deve parar.

No exemplo 51, o algoritmo deveria apresentar a multiplicação de 1 até o valor 10. Agora o algoritmo tem como nova condição de parada ir até a multiplicação do valor 1. Por último, temos a alteração na atualização. Anteriormente o contador era atualizado somando 1 a cada repetição. Agora ele continuará sendo atualizado a cada repetição, porém subtraindo 1.

Vamos pensar juntos: imagine que você foi contratado para desenvolver um sistema para uma academia de ginástica. Para cada aluno novo da academia, o sistema deve solicitar o seu nome e gerar automaticamente uma nova matrícula. A matrícula deve constar inicialmente dos dois últimos dígitos do ano do cadastro (exemplos: 21 quando for cadastrado no ano 2021, 22 quando for no ano de 2022, assim por diante) e, em seguida, um número sequencial.

Imagine que o sistema vai funcionar apenas para 15 novas matrículas. Como você poderia gerar essas matrículas de forma automática? Exemplo de matrículas em 2021: a primeira matrícula seria 2101, 2102, 2103, até 2115.

```
//Exemplo 53: Algoritmo para gerar número de matrícula de novos alunos de uma academia.
```

```
inteiro matricula
cadeia nomeCliente

para (matricula = 2101; matricula <= 2115; matricula++){
    escreva ( "\n\nDigite o nome da(o) cliente:")
    leia (nomeCliente)

    escreva ("A matrícula da(o) ", nomeCliente, " é ", matricula)
}

//Saída no console
Digite o nome da(o) cliente: Renato
A matrícula da(o) Renato é 2101

Digite o nome da(o) cliente: Maria
A matrícula da(o) Maria é 2102

Digite o nome da(o) cliente: Paula
A matrícula da(o) Paula é 2103
....
Digite o nome da(o) cliente: Priscila
A matrícula da(o) Priscila é 2115
```

Observe que, para resolver o problema de gerar uma matrícula automaticamente, iniciamos o contador com o número da primeira matrícula (2101). Colocamos a condição para ir até o último número de matrícula prevista, no caso no enunciado foi informado 2115. A partir da primeira matrícula todas as demais serão geradas pela própria estrutura de repetição. Dessa forma,

temos como automatizar a geração das matrículas somente utilizando a própria estrutura de repetição.

Consolidando dados lidos na repetição

Um assunto que surge com a utilização das estruturas de repetição é relacionado à geração de relatórios ao final das repetições. Por exemplo, em sistemas escolares, nos quais são armazenadas as notas dos alunos de uma turma, pode-se querer saber qual foi a maior nota tirada, qual a menor nota e qual a média de todas as notas. Em um sistema que controle um estacionamento, pode-se querer saber quantos carros entraram no estacionamento, qual a média de tempo que os carros ocupam uma vaga, qual carro ficou mais tempo ocupando uma vaga, entre outros. Todas essas informações podem ser retiradas do algoritmo, sendo necessário apenas criar algumas lógicas para obtenção dessas informações.

Vamos pensar juntos: imagine que você foi contratado para desenvolver um sistema para um caixa eletrônico de dinheiro. Esse caixa só serve para sacar dinheiro e só funciona para 10 clientes por dia. Ao final de um dia, o sistema deve emitir um relatório apresentando: qual foi o maior valor de dinheiro sacado, qual a média do dinheiro sacado e qual foi o total de dinheiro sacado por todos os clientes naquele dia.

//Exemplo 54: Algoritmo do caixa eletrônico.

```
real valorSacado, maiorValorSacado=0.0, mediaValorSacado, totalValorSacado=0.0
inteiro cont
```

```

para (cont = 1 ; cont <=10 ; cont++){
    escreva("***** Bem-vindo ao Caixa eletrônico *****")
    escreva("\n ")
    escreva ( "Qual valor deseja sacar: ")
    leia (valorSacado)

    // Verificar maior valor
    se( cont == 1){ // Entra na condição apenas quando for o primeiro saque
        maiorValorSacado = valorSacado
    }senao{ // Demais saque
        se (maiorValorSacado < valorSacado){ // Verifica se o novo valor lido é o maior
            maiorValorSacado = valorSacado
        }
    }

    // Controla a soma de todos os valores sacados
    totalValorSacado = totalValorSacado + valorSacado
}

escreva("Relatório do dia")
escreva("\n ")
escreva("Maior valor sacado: ", maiorValorSacado)
escreva("\n ")
escreva("Média do valor sacado: ", totalValorSacado / 10)
escreva("\n ")
escreva("Total de valor sacado: ", totalValorSacado )

//Saída no console - Exemplo com apenas 3 valores lidos
***** Bem-vindo ao Caixa eletrônico *****
Qual valor deseja sacar: 50
***** Bem-vindo ao Caixa eletrônico *****
Qual valor deseja sacar: 30
***** Bem-vindo ao Caixa eletrônico *****
Qual valor deseja sacar: 20
.....
Relatório do dia
Maior valor sacado: 50.0
Média do valor sacado: 33.33
Total de valor sacado: 100.0

```

Repare que, para realizarmos as três operações de descobrir o maior valor sacado, a média dos saques e o total sacado, foi necessário criar mais três variáveis do tipo real (`maiorValorSacado`, `mediaValorSacado` e `totalValorSacado`).

A variável utilizada para armazenar todos os valores lidos é chamada de **totalizadora** (nesse caso foi a variável `totalValorSacado`). Todas as variáveis totalizadoras precisam ser inicializadas antes de serem utilizadas, já que elas terão os valores contidos nela somados com os novos valores lidos. Porém, ao iniciarmos um novo programa, essa variável ainda não terá nenhum valor e, por isso, ela precisa começar do zero, ou outro valor, conforme feito na declaração do exemplo 54.

Depois dos comandos `escreva` e `leia`, temos a linha `// Verificar maior valor`. Essa linha é chamada de **comentário**, pois serve para que a pessoa programadora comente ou documente o que ele fez ou porque ele criou aquela lógica. Os comentários são muito utilizados por programadores em linhas em que existe um grau de complexidade a ser explicado. Comandos básicos ou triviais não são documentados para não poluir tanto o algoritmo. Então é importante comentar os itens complexos dos seus algoritmos. Para o compilador, essa linha não tem função, já que ela não será executada ao rodar o algoritmo.

Continuando no exemplo 54, temos uma estrutura condicional incluída inicialmente, que serve para verificar qual é o maior valor lido. A lógica utilizada é que, ao ler o primeiro valor sacado, ele já será o maior valor, pois ainda não existe outro para realizar a comparação. Por isso, verificamos quando a variável contadora é

igual a 1 para ter a certeza de que é a primeira vez que o ciclo do algoritmo é executado (se(cont == 1)).

Caso não seja a primeira repetição, o algoritmo entrará no senao e, como a variável maiorValorSacado já possui um valor que foi lido na primeira repetição, será necessário realizar uma nova condição. Essa nova condição verificará se o valor lido é maior que o valor que está na variável maiorValorSacado (com a expressão se (maiorValorSacado < valorSacado)). Se for, significa que o valor lido é o maior até o momento e por isso a variável maiorValorSacado precisa ser atualizada. Essa lógica funcionará a cada novo valor lido.

Em seguida tem-se a totalização de todos os valores lidos até o momento, por isso a variável recebe o valor que já está nela e é somando o novo valor sacado (totalValorSacado = totalValorSacado + valorSacado). Essa variável funciona como acumuladora dos valores sacados.

Ao sair da estrutura de repetição, teremos em uma variável o maior valor sacado e qual o total do valor sacado. Como utilizamos o para e sabemos quantas vezes a estrutura de repetição repetiu, basta dividirmos o total do valor sacado (totalValorSacado) pela quantidade de repetições, assim obteremos a média dos saques (totalValorSacado / 10).

Essa mesma lógica pode ser utilizada para todo e qualquer sistema que necessite saber o maior, menor, total ou média.

Nos próximos capítulos, estudaremos as outras duas estruturas de repetição, que são: o enquanto e o faca...enquanto .

Resumo do capítulo

Vimos neste capítulo estrutura de repetição. Elas são utilizadas quando queremos que um trecho de código seja repetido N vezes nos nossos algoritmos. Iniciamos pelo comando **para**, que é utilizado quando sabemos de antemão quantas vezes o trecho de código deve ser repetido. O comando é composto de 3 (três) parâmetros: inicialização, condição e atualização. Para o funcionamento do comando, utilizamos uma variável chamada **contador** que está presente nos três parâmetros. O primeiro parâmetro serve para indicar com qual valor o contador será iniciado. O segundo é a condição de parada da repetição, ela que indicará se a repetição deve continuar ou não. Na condição de parada utilizam-se os operadores relacionais juntamente com o contador. O último parâmetro é a atualização. Nesse parâmetro é definido como o contador será atualizado (adicionado ou subtraindo um valor).

11.2 EXERCÍCIOS

32. (F) Crie um programa que solicite 5 números e apresente na tela a soma de todos os números.

33. (F) Crie um programa para ler 10 números e no final da leitura de todos os números apresente quantos números lidos foram maiores que 50.

34. (F) Crie um programa para ler a altura de 12 atletas de basquete. Apresente no final quantos têm mais de 1.90.

35. (M) Crie um programa para ler a nota de 25 alunos de uma turma de lógica de programação. Apresente no final da leitura a maior nota, a menor nota e a média das notas.

36. (T) Crie um programa que solicite ao usuário o número de bolinhas de gude que estão em um pote de vidro. Se o número digitado for igual a 82, apresente a mensagem "Parabéns, você acertou". Se o número digitado for menor que 82, apresente a mensagem "Você errou! Existem mais bolinhas do que você

digitou". Se o número digitado for maior que 82, apresente a mensagem "Você errou! Existem menos bolinhas do que você digitou". O programa deve dar 5 oportunidades para que o usuário tente acertar a quantidade correta de bolinhas de gude.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Pedro solicitou uma nova alteração no sistema. Ele percebeu que em média ele atende por dia 18 clientes. O sistema deve continuar exibindo o menu e as informações normalmente, porém ele deve rodar 18 vezes e não apenas uma, como é hoje em dia.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap11>



Figura 11.2: QRCode

CAPÍTULO 12

ESTRUTURA DE REPETIÇÃO: ENQUANTO

Neste capítulo, estudaremos a segunda estrutura de repetição, o comando `enquanto`. Assim como o comando `para`, visto no capítulo anterior, o `enquanto` é utilizado para repetir N vezes um determinado trecho de código.

Vimos que usamos o comando `para` quando já sabemos de antemão (quando estamos escrevendo o algoritmo) quantas vezes o trecho de código deve ser repetido. Por exemplo: algoritmo para ler a velocidade de 10 carros em uma rodovia, algoritmo para imprimir o peso de 20 produtos de uma cesta básica, entre outros.

Porém, em muitos outros casos não saberemos quantas vezes o programa precisará repetir. Essa informação só é conhecida durante a execução do algoritmo. Como exemplos, temos um algoritmo para ler a velocidade de todos os veículos que passarem em uma rodovia, algoritmo para imprimir o peso de todos os produtos do carrinho de compras de um cliente em um hortifrutti, algoritmo para ler a idade de todos os clientes que comprarem ingresso para a sessão do cinema hoje de noite, entre outros.

Veja que em todos os exemplos apresentados no último

parágrafo não sabemos ao certo quantas vezes o trecho de código precisará ser repetido, já que depende de fatores externos.

Nesses casos, em que não sabemos o número de repetição necessário no momento da escrita do algoritmo, é recomendável utilizarmos os comandos enquanto e/ou faca enquanto . Porém, assim como o para , o enquanto e o faca enquanto também podem ser utilizados quando sabemos de antemão quantas vezes o código deve repetir.

Imagine o seguinte programa: você foi contratada para desenvolver um sistema de caixa de um hortifrutti. Os clientes selecionam seus produtos e colocam em seus carrinhos e, no final da compra, seu sistema deve registrar todos os produtos selecionados e calcular o valor que o cliente pagará.

Repare que a quantidade de vezes que o algoritmo terá que registar os produtos de cada cliente vai variar, já que cada cliente tem a liberdade de escolher a quantidade de produtos que deseja levar.

12.1 COMANDO ENQUANTO

Assim como o comando para , o enquanto depende da configuração dos três itens para funcionar corretamente (inicialização, condição e atualização). Porém, no comando para , esses três itens fazem parte do parâmetro do comando.

No enquanto , somente a condição está presente no comando. Os demais itens, inicialização e atualização, necessitam ser implementados pela pessoa programadora no local correto, caso contrário, o algoritmo não funcionará corretamente.

SINTAXE DO ENQUANTO

```
*local da inicialização*
enquanto (*condição*){

*local da atualização*
}
```

Nessa sintaxe, no `enquanto`, somente a condição é um parâmetro do comando. Assim como no comando `para`, a condição serve para verificar se a repetição deve continuar ou não. O resultado da condição sempre será verdadeiro ou falso. Se for verdadeiro, a estrutura continuará repetindo e se for falso, o algoritmo sairá da estrutura de repetição e continuará a execução na primeira linha abaixo do fechamento da chave "}".

Na condição, podem ser utilizados os operadores relacionais e/ou os operadores lógicos. Exemplos de condição: `idade >= 18` , `qtdVagas == 0`, (`peso < 80.00` ou `altura < 1.90`).

A inicialização não faz parte do comando e deve ser lembrada de ser implementada pelo programador. Ela deve ser escrita antes da linha do comando `enquanto`. Caso contrário não ocorrerá nenhuma repetição, já que o resultado da condição será falso. Alguns exemplos da inicialização de uma variável: `idade=28`, `leia(altura)` , `qtdVagas = 10`.

A atualização, que serve para atualizar o valor da variável utilizada na condição, deve estar dentro do par de chaves do comando `enquanto`. Normalmente, utiliza-se o comando `leia` na atualização quando não se sabe quantas vezes o trecho de código deve ser repetido. Recomenda-se que ele seja

o último comando antes do fechamento do par de chave.
Exemplos: leia(idade) , leia(peso) , leia(salario) ,
leia(continuar) .

O trecho de código que será repetido deve ser escrito dentro da abertura da chave e fechamento da chave do comando enquanto {}.

Vamos pensar juntos: imagine que você foi contratado para desenvolver um sistema de uma clínica de vacina. Todos os dias, a clínica recebe apenas 30 vacinas BCG. Monte um programa utilizando o comando enquanto que solicite quantos dias de nascimento tem o bebê que receberá a vacina BCG. Ao atingir o limite de 30 vacinas, o programa deve mostrar a média de dias dos bebês vacinados.

//Exemplo 55: Algoritmo de uma clínica para vacinação de 30 (trinta) recém-nascidos com a vacina BCG.

```
inteiro totalDiasNascimento =0, diasNascimento, contador
contador = 1
enquanto(contador <= 30){
    escreva("Digite a quantidade de dias do recém-nascido: ")
    leia(diasNascimento)
    totalDiasNascimento = totalDiasNascimento + diasNascimento
    contador++
}
escreva("\nMédia dos bebês vacinados no dia foi de : ", totalDiasNascimento/30 , " dias." )

//Saída no console - exemplo com apenas 3 repetições
Digite a quantidade de dias do recém-nascido: 12
Digite a quantidade de dias do recém-nascido: 8
```

```
Digite a quantidade de dias do recém-nascido: 7
```

```
Média dos bebês vacinados no dia foi de : 9 dias.
```

Rpare que o enunciado apresentava a quantidade de vezes que o algoritmo precisava repetir. Nesse caso, poderíamos utilizar o para ou o enquanto . Ou seja, quando soubermos no momento da escrita do algoritmo a quantidade de vezes que o algoritmo deve repetir, podemos utilizar tanto o para como o enquanto . Porém, quando não soubermos a quantidade de vezes que o algoritmo deve repetir, devemos usar o enquanto e/ou o faça enquanto .

Vamos pensar juntos: qual alteração necessária para que o exemplo 55 repita N vezes e não somente 30?

```
//Exemplo 56: Algoritmo de uma clínica para vacinação de N recém-nascidos com a vacina BCG.
```

```
inteiro totalDiasNascimento = 0, diasNascimento, qtdVacinas = 0
cadeia continuar = "sim"
enquanto(continuar == "sim"){
    escreva("Digite a quantidade de dias do recém-nascido: ")
    leia(diasNascimento)
    totalDiasNascimento = totalDiasNascimento + diasNascimento
    qtdVacinas++
    escreva("Digite sim para continuar e não para sair do programa: ")
    leia(continuar)
}
escreva("\nMédias dos bebês vacinados no dia foi de : ",totalDiasNascimento/qtdVacinas )

//Saída no console - exemplo com apenas 2 repetições
Digite a quantidade de dias do recém-nascido: 2
```

```
Digite sim para continuar e não para sair do programa: sim  
Digite a quantidade de dias do recém-nascido: 20  
Digite sim para continuar e não para sair do programa: não
```

Médias dos bebês vacinados no dia foi de : 11

Repare que no exemplo 56 foram criadas duas novas variáveis (`continuar` e `qtdVacinas`) e uma foi excluída (`contador`). Agora, em vez de usarmos uma variável contadora para controlar a quantidade de repetições, utilizamos uma variável que tem seu valor atualizado via teclado. Essa atualização é informada pelo usuário indicando se ele deseja continuar ou não a repetição. Porém, a variável precisa ser iniciada com o valor "sim" para que o algoritmo execute, pelo menos uma vez, o comando `enquanto` .

Depois que o comando entrar uma vez no `enquanto` e antes de ele finalizar a repetição, é necessário solicitar ao usuário se ele deseja continuar ou não. Para continuar a repetição, o usuário deve digitar no teclado a palavra "sim" ou então digitar "não" para sair. Com essa atualização sendo informada pelo usuário, o algoritmo pode repetir quantas vezes for necessário.

Vejamos outro exemplo.

Vamos pensar juntos: escreva um algoritmo que simule um radar em uma rodovia. A entrada das velocidades deve ser informada via teclado. Os veículos que passarem com a velocidade acima de 120 km/h devem ser multados. No final do algoritmo, deve-se apresentar a quantidade de veículos com a velocidade medida e a quantidade de veículos multados.

```
//Exemplo 57: Algoritmo para ler a velocidade de N veículos em um
```

a estrada e multá-los quando a velocidade for maior que 120 (cento e vinte) km/h. No final do algoritmo deve-se apresentar a quantidade de veículos com velocidade medida e a quantidade de veículo s multados.

```
real velocidadeVeiculo
cadeia continuarRepetindo = "sim"
inteiro qtdVelocidadeMedida = 0 ,qtdMultas = 0
enquanto(continuarRepetindo == "sim"){
    escreva("Digite a velocidade: ")
    leia(velocidadeVeiculo)
    se(velocidadeVeiculo > 120.00){
        qtdMultas++
    }
    qtdVelocidadeMedida++
    escreva("Digite sim para ler outra velocidade ou não para sair: ")
    leia(continuarRepetindo)
}
escreva("Total de veículo(s): ", qtdVelocidadeMedida , ".\nVeículo(s) multado(s): ", qtdMultas, ".")
```

//Saída no console
Digite a velocidade: 90
Digite sim para ler outra velocidade ou não para sair: sim
Digite a velocidade: 130
Digite sim para ler outra velocidade ou não para sair: sim
Digite a velocidade: 80
Digite sim para ler outra velocidade ou não para sair: não
Total de veículo(s): 3.
Veículo(s) multado(s): 1.

Repare que no exemplo anterior não sabíamos, na escrita do algoritmo, quantos carros o programa deveria ler. Nesse caso, utilizamos novamente o comando enquanto , já que com ele podemos repetir o trecho de código N vezes.

A quantidade de vezes que o trecho de código será repetido é definida pelo valor da variável chamada continuarRepetindo . Essa variável tem seu valor informado através do uso do teclado e é o usuário que define quando quer parar de repetir ou não.

No próximo capítulo, estudaremos a última estrutura de repetição, chamada de `faca...enquanto`.

Resumo do capítulo

Vimos neste capítulo a segunda estrutura de repetição, o comando `enquanto`. Ele pode ser utilizado quando fixamos no código a quantidade de vezes que o trecho de código deve ser repetido ou quando não soubermos de antemão (durante a escrita do código) quantas vezes o algoritmo será repetido. O comando necessita dos 3 itens para funcionar corretamente, são eles: inicialização, condição e atualização. Porém, somente a condição faz parte do comando, os outros dois itens precisam ser especificados pela programadora ao longo do algoritmo. A condição é usada para definir se o trecho de código deve continuar repetindo ou não. Nela, podemos utilizar os operadores relacionais e/ou lógicos. A inicialização serve para indicar o valor inicial da variável que é utilizada na condição. Ela deve ser escrita antes da linha do comando `enquanto` e precisa estar de acordo com a condição para que o algoritmo entre na estrutura de repetição pelo menos uma vez, caso seja necessário. A atualização serve para atualizar o valor da variável utilizada na condição. Normalmente, utiliza a leitura do teclado para realizar a atualização da variável.

12.2 EXERCÍCIOS

37. (F) Altere o exercício 32 para que utilize a estrutura de repetição `enquanto`.

38. (M) Crie um programa para ler N números até que a soma dos números seja maior ou igual a 100. Apresente a quantidade de números necessários para alcançar a soma maior ou igual a 100.

39. (T) Crie um programa para ler o nome e a velocidade da volta de N pilotos em uma pista de kart. Ao final do programa, você deve apresentar o nome do piloto com a volta mais rápida, o nome do piloto com a volta mais lenta e a média das voltas de todos os pilotos.

40. (T) Altere o exercício 36 para que sejam dadas N oportunidades para que o usuário acerte o número de bolinhas. Assim que o programa mostrar a mensagem de acerto ou erro, ele

deve perguntar se o usuário deseja continuar. Assim que encerrar o programa, o algoritmo deve mostrar o relatório com a quantidade de tentativas realizadas e qual erro mais comum (números informados abaixo ou acima).

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Você deve alterar o sistema para que ele funcione para N clientes e não mais apenas 18 vezes. Ao final da consulta de um cliente, o sistema deve apresentar os itens de menu acrescido da opção 10 – Sair. Ao digitar a opção 10, o sistema deve sair do programa e exibir quantos clientes usaram o sistema naquele dia.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap12>



Figura 12.1: QRCode

CAPÍTULO 13

ESTRUTURA DE REPETIÇÃO: FAÇA ENQUANTO

Neste capítulo, estudaremos a terceira estrutura de repetição, o comando `faca enquanto`. Assim como o comando `enquanto`, visto no capítulo anterior, o `faca enquanto` pode ser utilizado quando sabemos e também quando não sabemos o número de repetições que o algoritmo deve executar.

13.1 COMANDO FACÀ ENQUANTO

Esse comando é muito parecido com o comando `enquanto`, exceto pelo local onde está situada a condição.

No `enquanto`, a condição aparece logo na primeira linha do comando, e ele termina com o fechamento do par de chave. No comando `faca enquanto`, a condição aparece no final do comando. Ele vem junto com o fechamento da chave do comando.

Essa mudança de localização da condição altera a lógica do comando. Veja a sintaxe do `faca enquanto` a seguir.

SINTAXE DO ENQUANTO

```
faca{  
    *local da atualização*  
}enquanto (*condição*)
```

Nessa sintaxe, no comando `faca enquanto`, temos um único parâmetro fazendo parte do comando, que no caso é a condição. Assim como no comando `para` e `enquanto`, visto nos capítulos anteriores, a condição serve para verificar se a repetição deve continuar ou não. O resultado da condição sempre será `verdadeiro` ou `falso`. Se for `verdadeiro`, a estrutura continuará repetindo e se for `falso`, o algoritmo sairá da estrutura de repetição e continuará a execução na primeira linha abaixo do fechamento da chave "`}`".

Para testar as condições, serão utilizados os operadores relacionais e/ou os operadores lógicos. Exemplos de condição: `idade >= 18`, `qtdVagas == 0`, `peso < 80.00`.

Assim como o comando `enquanto`, somente a condição faz parte do parâmetro do comando. A inicialização não é necessária já que o comando garante que sempre ocorrerá pelo menos uma repetição.

Já a atualização, que serve para atualizar o valor da variável utilizada na condição, deve estar dentro do par de chaves do comando `faca enquanto`. Normalmente, utiliza-se o comando `leia` para realizar a leitura do valor do teclado e

ele costuma ser o último comando antes do fechamento do par de chaves. Exemplos: `leia(velocidade)` , `leia(tamanho)` , `leia(sair)` .

O trecho de código que será repetido deve ser escrito dentro da abertura da chave e fechamento da chave do comando `faca enquanto {}`.

Como apresentado na sintaxe do comando, a inicialização não se torna obrigatória já que o algoritmo dentro do ciclo sempre será executado pelo menos uma vez. Isso acontece pois a condição aparece no final do comando, ou seja, de qualquer maneira o algoritmo precisa passar pelo trecho que pode/deve ser repetido pelo menos uma vez até chegar à condição.

Essa é a principal diferença entre o comando `enquanto` e `faca enquanto` . Ou seja, o `faca enquanto` sempre executará pelo menos uma vez a repetição e o comando `enquanto` pode ou não repetir uma vez (isso vai depender do valor da inicialização da variável utilizada na condição). A própria localização da condição nos comandos `enquanto` e `faca enquanto` indica a diferenciação entre ambos: no primeiro, o teste é realizado no início, no segundo o teste é realizado apenas no final.

Vamos pensar juntos: altere o exemplo 57 para que seja executado utilizando o comando `faca enquanto` .

//Exemplo 58: Alteração no Exemplo 57 para utilizar o comando `faca enquanto`

```

real velocidadeVeiculo
cadeia continuarRepetindo
inteiro qtdVelocidadeMedida = 0 ,qtdMultas = 0
faca{
    escreva("Digite a velocidade: ")
    leia(velocidadeVeiculo)
    se(velocidadeVeiculo > 120.00){
        qtdMultas++
    }
    qtdVelocidadeMedida++
    escreva("Digite sim para ler outra velocidade ou não p
ara sair: ")
    leia(continuarRepetindo)
}enquanto(continuarRepetindo == "sim")
    escreva("Velocidade medida de ", qtdVelocidadeMedida , " veíc
ulo(s) e foram multado(s) ", qtdMultas , " veículo(s).")

//Saída no console
Digite a velocidade: 90
Digite sim para ler outra velocidade ou não para sair: sim
Digite a velocidade: 130
Digite sim para ler outra velocidade ou não para sair: sim
Digite a velocidade: 80
Digite sim para ler outra velocidade ou não para sair: não
Velocidade medida de 3 veículo(s) e foram multado(s) 1 veículo(s)
.

```

Repare que foram realizadas apenas 3 alterações: a retirada da inicialização da variável `continuarRepetindo` (poderia ficar com a inicialização que não mudaria em nada, pois antes de chegar à condição ocorre a atualização), alteração do comando `enquanto` para o comando `faca`, e após o fechamento da chave do comando `faca` foi incluído o comando `enquanto(continuarRepetindo == "sim")`.

Vamos pensar juntos: já que vimos a sintaxe dos três comandos, podemos levantar a seguinte dúvida. Vimos que devemos utilizar o comando `para` quando sabemos previamente

a quantidade de vezes que o trecho de código deve repetir (lembrando que nesses casos também podemos usar os comandos `enquanto` e `faca enquanto`). E quando não soubermos, qual devemos utilizar? O `enquanto` ou `faca enquanto`?

Basicamente, devemos analisar se o nosso algoritmo deve repetir pelo menos uma vez a estrutura de repetição. Se a resposta for "sim", podemos utilizar qualquer uma das duas, desde que a inicialização da variável seja preenchida corretamente no comando `enquanto`. Agora se a resposta for "não", só devemos utilizar o `enquanto`.

Porém, existem duas situações em que normalmente utilizamos o comando `faca enquanto`, são elas: apresentação de menu e validação dos dados de entrada.

Apresentando menu com *faca enquanto*

Na apresentação de um menu de opções, temos que garantir que pelo menos uma vez o menu apareça para o usuário. Nesses casos, normalmente utilizamos o `faca enquanto`. Veja o exemplo a seguir.

//Exemplo 59: Exemplo de utilização do comando `faca enquanto` para exibição de menu

```
inteiro opcao
faca{
    escreva("***** MENU DE PRODUTOS *****\n\n")
    escreva("1 - Cadastrar novo produto\n")
    escreva("2 - Listar produtos\n")
    escreva("3 - Alterar produto\n")
    escreva("4 - Excluir produto\n")
    escreva("5 - sair\n")
    escreva ("Digite a opção desejada: ")
```

```

leia(opcao)

    // condicional para cada um dos itens do menu

}enquanto(opcao != 5)

//Saída no console
***** MENU DE PRODUTOS *****

1 - Cadastrar novo produto
2 - Listar produtos
3 - Alterar produto
4 - Excluir produto
5 - sair
Digite a opção desejada:

```

Repare que no menu de opções existem várias opções e uma delas é a sair (5). Utilizando o `faca enquanto`, garantimos que o algoritmo será executado pelo menos uma vez e que o usuário deixará o programa repetindo até que ele informe a opção `sair`.

Validando dados de entrada com *faca enquanto*

A outra situação, na qual sempre utilizamos o `faca enquanto`, é na validação dos dados de entrada. Até o momento os nossos algoritmos não validavam os dados de entrada. Ou seja, mesmo que informássemos no enunciado que o usuário só poderia entrar com valores entre 100 e 200, por exemplo, não fazíamos essa validação. Logo, se o usuário digitasse um número fora desse intervalo, o programa continuava executando normalmente.

Com a validação dos dados de entrada, criaremos uma barreira para que o programa só continue a ser executado se a informação digitada pelo usuário estiver correta. Vejamos o exemplo a seguir.

```
//Exemplo 60: Exemplo de validação dos dados de entrada
```

```
inteiro idade
```

```

        escreva("***** Sistema para agendamento da prova CNH ****
*\n ")

        faca{
            escreva("Atenção! todos os candidatos devem ser m
aiores de 18(anos). \n ")
            escreva("Digite a idade do candidato:")
            leia(idade)
        }enquanto(idade<18)

//Saída no console
***** Sistema para agendamento da prova CNH *****
Atenção! todos os candidatos devem ser maiores de 18(anos).
Digite a idade do candidato:16
Atenção! todos os candidatos devem ser maiores de 18(anos).
Digite a idade do candidato:17
Atenção! todos os candidatos devem ser maiores de 18(anos).
Digite a idade do candidato:18

Programa finalizado.

```

Repare que o `faca enquanto` é utilizado para validar se a idade digitada no teclado é maior ou igual a 18 (dezoito). Se ela não for, como apresentado no exemplo, o algoritmo continuará repetindo o trecho de código definido até que seja digitado o valor correto. Esse tipo de validação deve ser usado sempre que for necessário.

Vejamos mais um exemplo.

```

//Exemplo 61: Exemplo de validação dos dados de entrada

    real nota
    escreva("***** Sistema para registro de notas dos alunos
*****\n ")

    faca{
        escreva("Atenção! Nenhum aluno pode ter nota nega
tiva ou maior que 10. \n ")
        escreva("Digite a nota do aluno:")
        leia(nota)
    }enquanto(nota<0.0 ou nota > 10.0)

```

```
//Saída no console
***** Sistema para registro de notas dos alunos *****
Atenção! Nenhum aluno pode ter nota negativa ou maior que 10.
Digite a nota do aluno:15.2
Atenção! Nenhum aluno pode ter nota negativa ou maior que 10.
Digite a nota do aluno:8.6

Programa finalizado.
```

Veja que no exemplo anterior o algoritmo ficará repetindo a validação de entrada enquanto a nota digitada não estiver entre 0 e 10. Como os números que não podem ser digitados estão em dois intervalos diferentes, foi usado o operador lógico ou para compor a condição.

No próximo capítulo, estudaremos a estrutura de dados chamada de vetor.

Resumo do capítulo

Vimos neste capítulo a terceira estrutura de repetição, o comando `faca enquanto`. Ele pode ser utilizado quando fixamos no código a quantidade de vezes que o trecho de código deve ser repetido ou quando não soubermos de antemão (quando estamos escrevendo o código) quantas vezes o algoritmo será repetido. Assim como o comando `enquanto`, o comando `faca enquanto` só utiliza a condição como parâmetro. A inicialização no `faca enquanto` é necessária já que o comando garante que sempre ocorrerá pelo menos uma repetição. A atualização precisa ser especificada pelo programador antes da condição. A condição é usada para definir se o trecho de código deve continuar repetindo ou não. Na condição, podemos utilizar os operadores relacionais e/ou lógicos. A atualização serve para atualizar o valor da variável utilizada na condição. Normalmente, utiliza-se a leitura do teclado para realizar a atualização da variável. A diferença entre o `enquanto` e o `faca enquanto` é a localização da condição (no `faca enquanto` ela aparece na última linha do comando e no `enquanto` na primeira linha). Com essa mudança de local, utiliza-se o `faca enquanto` para exibição de menus e para validação dos dados de entrada.

13.2 EXERCÍCIOS

41. (M) Altere o exercício 38 para que só sejam aceitos números maiores que 0 na entrada dos dados.

42. (F) Você foi contratado para desenvolver um sistema de emissão de boletos. O cliente deve informar qual o melhor dia para pagamento do boleto. Os dias disponíveis são 2, 5 ou 10. O sistema deve validar o dia informado pelo cliente e apresentar a mensagem boleto registrado caso o dia seja válido. Se o dia for inválido, o sistema deve solicitar um novo dia até que ele seja digitado corretamente.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Pedro solicitou que algumas funcionalidades fossem retiradas do sistema e algumas novas fossem implementadas, porém o programador responsável considerou que eram mudanças complexas. Com isso, foi sugerido ao dono da loja de bicicleta o desenvolvimento de uma nova versão (2.0) planejada para atender todas as funcionalidades e começando do zero. Pedro aceitou e ficou de levantar todas as funcionalidades previstas para serem lançadas na versão 2.0 no próximo capítulo.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap13>



Figura 13.1: QRCode

Vetor e Matriz



Figura 1: Fonte: Imagem do Pixabay.

Nestes capítulos, estudaremos os vetores e as matrizes. Essas estruturas são utilizadas para armazenar vários dados do mesmo tipo e que tenham o mesmo objetivo.

CAPÍTULO 14

VETOR

Neste capítulo, estudaremos as estruturas homogêneas unidimensionais chamadas de vetor. O vetor é utilizado para armazenar vários dados de um mesmo tipo. Ele funciona como se fosse uma variável que em vez de armazenar um único valor pode armazenar N valores.

Vamos pensar juntos: imagine que você foi contratado para desenvolver um sistema para uma escola de natação. O sistema precisa armazenar as idades dos 10 alunos de uma turma. Com o que você aprendeu até agora, como é que você armazenaria essas idades?

//Exemplo 62: Declaração das variáveis para armazenar 10 idades

```
inteiro idade1, idade2, idade3, idade4, idade5  
inteiro idade6, idade7, idade8, idade9, idade10
```

De acordo com o que aprendemos até agora, criariíamos o algoritmo com 10 variáveis (idade1, idade2, idade3, idade4,..., idade10).

Agora imagine que a escola de natação fez um convênio com um colégio e que todos os seus 680 alunos farão aula de natação. O

que seria necessário alterar no sistema para armazenar agora a idade dos 680 novos alunos?

Repare que, ao utilizarmos as variáveis para armazenar os valores das idades, encontramos dificuldade quando o número de variáveis começa a aumentar. Imagine agora um sistema de folha de pagamento de uma indústria que tem que armazenar o salário de mais de 20.000 funcionários, seria inviável trabalhar com tantas variáveis nesse sistema.

Nesses casos, em que temos que armazenar muitos valores, utilizaremos os vetores em vez das variáveis. Normalmente, recomenda-se que utilize o vetor quando for necessário criar 3 ou mais variáveis para armazenar a mesma informação (exemplo da natação: se na escola só tivessem 2 alunos poderia usar variáveis, porém se tivesse 3 ou mais alunos recomenda-se a utilização do vetor).

Com o uso do vetor, melhoramos o algoritmo da escola de natação, pois, em vez de termos 10 variáveis, passamos a ter apenas um vetor com 10 posições.

//Exemplo 63: Declaração de um vetor com 10 posições

```
inteiro idade[10]
```

Utilizaremos os vetores quando os tipos dos dados forem iguais e tiverem o mesmo objetivo. Mas o que quer dizer "mesmo objetivo"? Imagine que você tivesse que criar um programa para armazenar a idade e a respectiva matrícula de 10 alunos (exemplo de matrícula: 2123, 2145, 2134, entre outros). Veja que os tipos de dados da idade e da matrícula são iguais (os dois são do tipo inteiro), porém cada um tem um objetivo bem definido (um é para

armazenar a idade e o outro a matrícula). Nesses casos, mesmo sendo de tipos iguais, devem ser criados vetores diferentes, já que cada um tem o objetivo de armazenar uma informação diferente do outro. Essa análise é bastante importante de ser feita na declaração dos vetores.

Veja, didaticamente, a diferença entre a utilização de 10 variáveis e a utilização de 1 vetor com 10 posições na memória do computador.

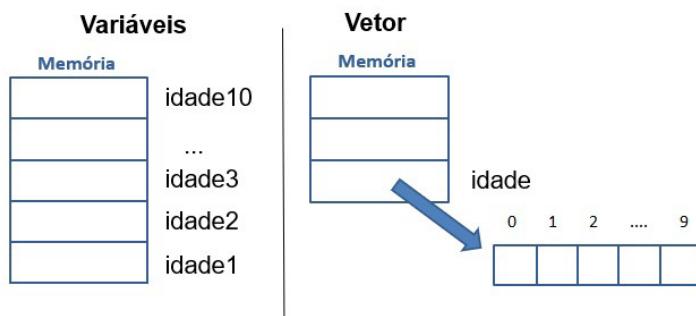


Figura 14.1: Comparação, didática, entre vetor e variável.

Repare que o vetor funciona como se pegássemos uma variável e dividíssemos o seu espaço na memória em N espaços menores. O processo real é um pouco mais complicado que o apresentado na imagem, porém não é necessário nos aprofundarmos nesse assunto agora.

Continuando na imagem, vemos que uma única posição armazenará mais de um valor. Como é que faremos para identificar cada um dos valores? Repare que na parte de cima de cada posição do vetor temos um índice (número sequencial iniciado no 0 que representa cada posição do vetor).

O índice do vetor, ou posição do vetor, será utilizado para identificar qual é a posição em que o valor deve ser armazenado ou lido. Para facilitar os passos para a utilização do vetor, dividiremos a sintaxe em duas partes: declaração e atribuição/leitura.

Declaração de um vetor

Assim como fazemos para as variáveis, os vetores também precisam ser declarados antes do seu uso. A declaração seguirá a mesma lógica da declaração das variáveis, com uma exceção. Veja a seguir a sintaxe da declaração do vetor.

SINTAXE DA DECLARAÇÃO DO VETOR

```
tipo_vetor nome_vetor[tamanho_vetor]
```

Nela, `tipo_vetor` deve ser preenchido com o tipo do dado que será armazenada no vetor. Os tipos utilizados na declaração dos vetores são os mesmos utilizados nas variáveis (`inteiro`, `real`, `cadeia`, `logico` e `caracter`). O `nome_vetor` é o nome utilizado para identificação do vetor no algoritmo. O nome do vetor deve seguir o mesmo padrão do nome das variáveis. O tamanho do vetor é a quantidade de espaços que o vetor terá para armazenar os dados.

Vamos pensar juntos: agora que já vimos a sintaxe da declaração do vetor, declare um vetor para armazenar o modelo de 5 carros.

//Exemplo 64: Declaração de um vetor para armazenar o modelo de 5 carros

```
cadeia carro[5]
```

Veja que a declaração do vetor é muito parecida com a declaração de uma variável, exceto pela quantidade de posições necessárias no vetor.

Vamos pensar juntos: como é que vamos acessar cada um dos índices (posições) do vetor?

Para irmos diretamente para uma posição do vetor, basta chamarmos o nome do vetor e passarmos dentro dos colchetes o índice desejado. No exemplo `carro[2]` , estamos pedindo que o compilador accesse a área de memória em que se encontra o vetor chamado `carro` e que vá até o índice 2 desse vetor. Essa mesma lógica vale para todos os índices do vetor.

Atenção: o índice de todo vetor começa sempre do 0 e é importante observar que o último índice de todo vetor sempre será o tamanho declarado menos 1. Exemplo: em `real salario[8]` , o último índice do vetor `salario` será o 7 (`salario[7]`). Se colocar no algoritmo, por exemplo, uma atribuição para o índice 8, o programa apresentará um erro de execução.

Atribuição e leitura do vetor

Para atribuirmos um valor ao vetor, basta utilizarmos o operador `=` ou o comando `leia` , exatamente como fazíamos com as variáveis, porém temos que informar em qual índice esse

valor será atribuído ou lido.

Vamos pensar juntos: agora que já vimos a sintaxe da declaração do vetor e como acessar cada índice, construa um programa para armazenar os seguintes modelos de carros: Uno, HB20, Argo, Fit e Onix.

```
//Exemplo 65: Algoritmo para ler e armazenar os seguintes modelos de carro: Uno, HB20, Argo, Fit e Onix
```

```
cadeia carro[5]

carro[0] = "Uno"
carro[1] = "HB20"
carro[2] = "Argo"
carro[3] = "Fit"
carro[4] = "Onix"

escreva("O modelo do carro armazenado no índice 2 é ", carro[2])
//Saída no console
O modelo do carro armazenado no índice 2 é Argo
```

Repare que no exemplo anterior atribuímos modelos de carros para todas as posições do vetor e apresentamos o modelo de apenas uma posição do vetor na tela para o usuário.

Porém, perceba que tivemos que fazer essa atribuição e a escrita na forma manual, posição por posição. Nesse exemplo, foi fácil pois tínhamos apenas 5 posições. Imagine que tenhamos que criar um vetor com 1.000 posições ou com 100.000 posições, fazer de forma manual se torna impossível.

Vamos pensar juntos: qual estrutura podemos utilizar para automatizar a atribuição e a leitura de um vetor?

Se você se lembrou das estruturas de repetição está de parabéns. Lembre-se de que as estruturas de repetição servem para repetir um trecho de código quantas vezes nosso algoritmo necessitar. Como sempre sabemos o tamanho do vetor, já que sempre temos que declará-lo, podemos utilizar uma estrutura de repetição para fazer o trabalho de ler e escrever os dados de um vetor. Vejamos o exemplo a seguir.

//Exemplo 66: Algoritmo para armazenar os modelos de carro Uno, HB20, Argo, Fit e Onix, e em seguida exibi-los na tela.

```
cadeia carro[5]

// estrutura de repetição para ler os modelos dos carros do teclado e armazenar no vetor
para(inteiro cont = 0; cont <= 4; cont++){
    escreva("Digite o modelo do ", cont, "º carro: ")
    leia(carro[cont])

}
escreva("\n\n")
// estrutura de repetição para escrever na tela os modelos dos 5 carros armazenados no vetor
para(inteiro cont = 0; cont <= 4; cont++){
    escreva("O ", cont, "º carro armazenado foi o ", carro[cont], ". \n")
}

//Saída no console
Digite o modelo do 0º carro: Uno
Digite o modelo do 1º carro: HB20
Digite o modelo do 2º carro: Argo
Digite o modelo do 3º carro: Fit
Digite o modelo do 4º carro: Onix

O 0º carro armazenado foi o Uno.
```

```
0 1 ° carro armazenado foi o HB20.  
0 2 ° carro armazenado foi o Argo.  
0 3 ° carro armazenado foi o Fit.  
0 4 ° carro armazenado foi o Onix.
```

Rpare que na saída do console ficou estranho exibir a posição 0°. Podemos resolver essa saída adicionando 1 ao local em que o cont é exibido. Veja o exemplo a seguir e compare com o anterior.

```
//Exemplo 67: Algoritmo para armazenar os modelos de carro Uno, H  
B20, Argo, Fit e Onix, e em seguida exibi-los na tela.  
//Saída alterada para não apresentar o 0°
```

```
cadeia carro[5]  
  
// estrutura de repetição para ler os modelos dos carros do t  
eclado e armazenar no vetor  
para(inteiro cont = 0; cont <= 4; cont++){  
    escreva("Digite o modelo do ", cont, " ° carro: ")  
    leia(carro[cont])  
  
}  
escreva("\n\n")  
// estrutura de repetição para escrever na tela os modelos do  
s carros armazenados no vetor  
para(inteiro cont = 0; cont <= 4; cont++){  
    escreva("0 ", cont + 1, " ° carro armazenado foi o ", ca  
rro[cont], ". \n")  
}  
  
//Saída no console  
Digite o modelo do 0 ° carro: Uno  
Digite o modelo do 1 ° carro: HB20  
Digite o modelo do 2 ° carro: Argo  
Digite o modelo do 3 ° carro: Fit  
Digite o modelo do 4 ° carro: Onix  
  
0 1 ° carro armazenado foi o Uno.  
0 2 ° carro armazenado foi o HB20.  
0 3 ° carro armazenado foi o Argo.  
0 4 ° carro armazenado foi o Fit.
```

O 5º carro armazenado foi o Onix.

Vejamos mais um exemplo da automatização da atribuição e leitura de um vetor.

```
//Exemplo 68: Algoritmo para armazenar o salário e o número da camisa de um time principal de futebol. No final o algoritmo deve apresentar os salários que estão acima da média dos salários.

real salario[11], totalSalario = 0.0 , mediaSalario
inteiro numeroCamisa[11]

// estrutura de repetição para atribuir valores no vetor
para(inteiro cont = 0; cont < 11; cont++){
    escreva("Digite o salário do ", cont + 1 , "º jogador: ")
    leia(salario[cont])
    escreva("Digite a camisa que o ", cont + 1, "º jogador usa em campo: ")
    leia(numeroCamisa[cont])
    totalSalario = totalSalario + salario[cont]
}
escreva("\n")
escreva("-----")
mediaSalario = totalSalario/11
escreva("Média dos salários: ", mediaSalario)
escreva("-----")
escreva("\n")

// estrutura de repetição para ler os salários no vetor e apresentar somente os maiores que a média
para(inteiro cont = 0; cont < 11; cont++){
    se( salario[cont] > mediaSalario ){
        escreva("\nSalário acima da média: ", salario[cont],
" - Camisa: ", numeroCamisa[cont])
    }
}

//Saída no console
Digite o salário do 1º jogador: 25000
Digite a camisa que o 1º jogador usa em campo: 4
Digite o salário do 2º jogador: 50000
Digite a camisa que o 2º jogador usa em campo: 8
Digite o salário do 3º jogador: 75000
Digite a camisa que o 3º jogador usa em campo: 9
```

```
Digite o salário do 4º jogador: 10000
Digite a camisa que o 4º jogador usa em campo: 2
```

```
-----Média dos salários: 40000.0-----
```

```
Salário acima da média: 50000.0 - Camisa: 8
Salário acima da média: 75000.0 - Camisa: 9
```

No próximo capítulo, estudaremos a estrutura homogênea multidimensional chamada de matriz.

Resumo do capítulo

Vimos neste capítulo os vetores. Eles funcionam como uma variável que armazena mais de um valor. Para utilização dos vetores é necessário que o tipo do dado seja o mesmo e o objetivo também. A declaração de um vetor é igual à de uma variável, exceto pela necessidade de indicar dentro do par de colchetes a quantidade de posições do vetor. Para identificar uma posição, utilizam-se os índices, que sempre se iniciam em 0. A leitura/escrita pode ser automatizada, utilizando as estruturas de repetição para percorrer todos os índices do vetor.

14.1 EXERCÍCIOS

43. (M) Crie um programa para armazenar a velocidade de 6 voltas de um piloto em uma pista de kart. Depois de ter armazenado as velocidades, seu programa deve apresentar as velocidades na ordem contrária da lida (a última velocidade lida será a primeira a ser exibida, e assim sucessivamente).

44. (F) Crie um programa para fidelização de clientes de um restaurante. A cada pagamento no restaurante, o valor é armazenado na cartela de fidelização. Assim que o cliente completar as 10 posições da cartela, o sistema deve apresentar a seguinte mensagem: "Hoje o seu almoço é uma cortesia da casa, Parabéns!".

45. (T) Crie um programa para armazenar os 6 caracteres da

senha do usuário. A senha só pode ter as vogais (a, e, i, o e u). Depois de armazenar cada vogal em uma posição, seu programa deve realizar a criptografia da senha. A lógica da criptografia é: cada letra 'a' deve ser substituída pelo caractere 'z', letra 'e' pelo caractere '3', letra 'i' pelo caractere 'l', letra 'o' pelo caractere '0' e letra 'u' pelo caractere \$. Após criptografar a senha, o programa deve apresentar a senha digitada e a senha criptografada.

46. (T) Crie um programa para armazenar o nome do jogador e a quantidade de gols de 11 jogadores de um time. Ao finalizar a leitura dos jogadores, o algoritmo deve apresentar na tela o nome e a quantidade de gols do artilheiro do time.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Com o crescimento de sua loja, Pedro solicitou novas funcionalidades para o sistema. Os desenvolvedores sugeriram criar do zero uma versão 2.0 e manter somente as funcionalidades mais utilizadas da versão anterior. O sistema deve funcionar da seguinte maneira:

1. Na tela principal, será apresentada a mensagem “Bem-vindo ao autoatendimento da biciceraria XPTO Bikes.” Abaixo da mensagem deve ser exibida a mensagem solicitando o nome do cliente.
2. Ao digitar seu nome, o sistema deve apresentar o seguinte menu:

- 2.1. Opção 1 – Ver promoções.
 - 2.2. Opção 2 – Solicitar serviço de manutenção.
 - 2.3. Opção 3 – Listar carrinho de compra.
 - 2.3. Opção 4 - Finalizar carrinho de compra.
 - 2.5. Opção 0 - Sair.
3. Abaixo deve ser exibida a mensagem “Digite sua opção desejada:”.
4. Se o cliente digitar a opção 1, o sistema deve apresentar os seguintes itens:
- 4.1. Código 101 - Bicicleta nova na cor amarela, aro 26, com 18 marchas e na promoção pelo preço de R\$ 999,99.
 - 4.2. Código 102 - Bicicleta usada na cor azul, aro 26, com 18 marchas e com o valor promocional de R\$ 400,00.
 - 4.3. Código 103 - Capacete de proteção por R\$59,99.
 - 4.4. Código 104 - Freio a disco por R\$ 89,99.
 - 4.5. 8 – Adicionar ao carrinho de compras.
 - 4.6. 0 – Voltar.
5. Se o cliente digitar a opção 2, o sistema deve apresentar os serviços disponíveis:
- 5.1. Código 201 - Troca de pneu - R\$ 55,99.
 - 5.2. Código 202 - Lavagem completa -R\$ 12,99.
 - 5.3. Código 203 - Freio - R\$ 10,99.
 - 5.4. 8 – Adicionar ao carrinho de compras.
 - 5.5. 0 – Voltar.

6. Se o cliente digitar a opção 3, o sistema deve listar o nome e o valor dos produtos incluídos no carrinho de compras. Se o cliente digitar a opção 4, o sistema deve perguntar a forma de pagamento (dinheiro ou cartão). No dinheiro, o sistema deve calcular 10% de desconto do valor final.
7. Observação: no autoatendimento, o cliente consegue adicionar no máximo 3 produtos/serviços em seu carrinho de compras. Se ele atingir esse número, o sistema deve apresentar a mensagem informando que o seu carrinho de compras já está cheio.

Acesso restrito

8. Ao digitar o código `xptorestrito` no lugar do nome do cliente, o sistema deve abrir a área restrita dos funcionários.
9. O sistema deve perguntar o nome do cliente e o valor gasto na loja. Nesses casos, o próprio funcionário acompanhou o cliente e sabe exatamente os itens comprados ou serviços adquiridos.
10. O funcionário deve entrar com a forma de pagamento (D - dinheiro ou C - cartão), e o sistema deve apresentar o valor final a ser pago. Se o pagamento for no dinheiro, o sistema deve calcular 10% de desconto no valor final.

Funcionamento geral do sistema

11. O sistema deve validar todas as entradas do teclado,

- quando possível;
12. O sistema deve ficar rodando até que seja a opção 0 no menu principal.
 13. O sistema deve apresentar a quantidade de vendas e o valor total das vendas do dia ao ser finalizado.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap14>



Figura 14.2: QRCode

CAPÍTULO 15

MATRIZ

Neste capítulo, estudaremos as estruturas homogêneas multidimensionais chamadas de matriz. A matriz é utilizada para armazenar vários dados de um mesmo tipo, igual ao vetor, porém com mais de uma dimensão. No vetor, as posições são distribuídas em colunas, uma do lado da outra, e sempre em uma única linha. Já na matriz, além da utilização das colunas, as posições também são distribuídas em mais linhas. Veja a imagem a seguir.

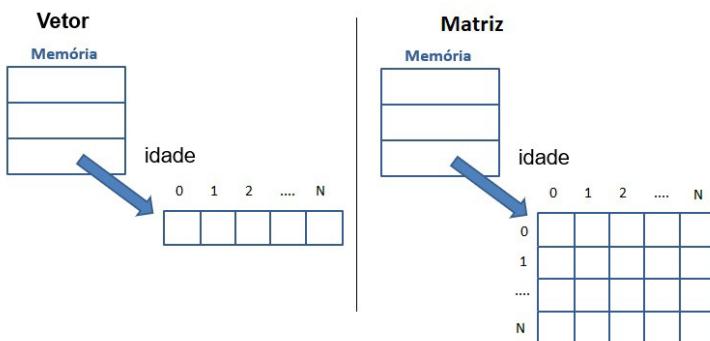


Figura 15.1: Comparação, didática, entre vetor e matriz.

Vamos pensar juntos: imagine que você foi contratada para desenvolver um sistema para controlar a venda de ingressos de um pequeno teatro. Atualmente o teatro conta com 5 fileiras e cada

fileira tem 6 poltronas. Seu sistema deve apenas exibir o mapa do teatro para o cliente escolher qual fileira e poltrona deseja comprar.

Esse sistema poderia ser desenvolvido usando vetores, ou seja, um vetor para cada fileira, por exemplo. Porém, a manipulação de leitura e escrita desses vetores tornaria o algoritmo um pouco complexo. Com a utilização de matrizes, esse problema de complexidade é resolvido, já que em vez de usarmos cinco vetores usariamos apenas uma matriz. Vejamos a sintaxe da matriz e o exemplo da venda de ingressos.

SINTAXE DA MATRIZ

```
tipo_matriz nome_matriz[quantidade_linhas][quantidade_colunas]
```

`tipo_matriz` deve ser preenchido com o tipo do dado que será armazenado na matriz. Os tipos utilizados na declaração das matrizes são os mesmos utilizados nos vetores (`inteiro`, `real`, `cadeia`, `logico` e `caracter`). O `nome_matriz` é o nome utilizado para identificação da matriz no algoritmo. O nome da matriz deve seguir o mesmo padrão do nome dos vetores. A `quantidade_linhas` indica a quantidade de linhas que a matriz terá e a `quantidade_colunas` é a quantidade de colunas que cada linha da matriz terá. O tamanho da matriz, quantidade de posições, é o resultado da multiplicação das linhas pelas colunas.

//Exemplo 69: Algoritmo para venda de ingressos de um pequeno teatro com 5 fileiras e 6 poltronas em cada fileira.

```
inteiro poltrona[5][6]
    // atribuindo valor inicial 0 para cada poltrona
para(inteiro linha = 0; linha < 5; linha++){
    para (inteiro coluna = 0; coluna < 6; coluna++){
        poltrona[linha][coluna]=0
    }
}
escreva("Mapa do teatro\n")
escreva("\n_____
_____\n")
escreva("                # PALCO #
\n")
para(inteiro linha = 0; linha < 5; linha++){
    escreva("\n")
    para (inteiro coluna = 0; coluna < 6; coluna++){
        escreva ( "P(",linha, coluna, "): ", poltrona[linha][coluna], " | ")
    }
}
escreva("\n_____FINAL DO TEATRO_____
_____\n")
}

//Saída no console
Mapa do teatro
```

```
# PALCO #

P(00): 0 | P(01): 0 | P(02): 0 | P(03): 0 | P(04): 0 | P(05): 0 |
P(10): 0 | P(11): 0 | P(12): 0 | P(13): 0 | P(14): 0 | P(15): 0 |
P(20): 0 | P(21): 0 | P(22): 0 | P(23): 0 | P(24): 0 | P(25): 0 |
P(30): 0 | P(31): 0 | P(32): 0 | P(33): 0 | P(34): 0 | P(35): 0 |
P(40): 0 | P(41): 0 | P(42): 0 | P(43): 0 | P(44): 0 | P(45): 0 |
```

```
FINAL DO TEATRO
```

Repare que declaramos a matriz com duas dimensões, 5 linhas e 6 colunas (`inteiro poltrona[5][6]`). Como no vetor tínhamos apenas uma dimensão e utilizávamos um único para na sua manipulação, teremos que utilizar dois comandos para na manipulação da matriz, já que agora são duas dimensões.

Se analisarmos com calma a figura da matriz na memória, podemos observar que todas as colunas estão sempre vinculadas a uma linha. Dessa forma, temos que criar dois comandos para , um dentro do outro, já que ao iniciarmos, por exemplo, na linha 1 temos que percorrer todas as suas colunas (poltronas) para então ir para a próxima linha. Essa lógica ocorre em todas as linhas da matriz.

Usamos então o `para de fora` para percorrer as linhas e o `para de dentro`, mais interno, para percorrer todas as colunas.

Repare também que a saída do console está formatada para apresentar como se fosse o desenho do teatro. Temos a apresentação do local do palco e em seguida todas as poltronas disponíveis no teatro. Ao lado do P, temos a informação da linha e coluna que representam a respectiva poltrona. Veja que, para a saída ficar nessa formatação, foi necessário usar diversas quebras de linhas e espaços em pontos específicos.

Vamos pensar juntos: continuando o exemplo da venda de ingresso, agora o seu sistema deve solicitar ao usuário qual poltrona ele deseja comprar. O sistema então deve ocultar a poltrona comprada. O sistema só deve realizar uma compra por enquanto, como ficaria seu algoritmo?

```

//Exemplo 70: Algoritmo para venda de ingressos de um pequeno teatro com 5 fileiras e 6 poltronas em cada fileira.

inteiro poltrona[5][6]=0, fileiraComprada, colunaComprada
para(inteiro linha = 0; linha < 5; linha++){
    para (inteiro coluna = 0; coluna < 6; coluna++){
        poltrona[linha][coluna]=0
    }
}
escreva("Mapa do teatro\n")
escreva("\n_____
_____ \n")
escreva("                                # PALCO #
\n")
para(inteiro linha = 0; linha < 5; linha++){
    escreva("\n")
    para (inteiro coluna = 0; coluna < 6; coluna++){
        escreva ( "P(",linha, coluna, "): ", poltrona[linha][coluna], " | ")
    }
}
escreva("\n_____ FINAL DO TEATRO _____
\n\n")
escreva("Digite o número da fileira em que deseja comprar a poltrona:")
leia(fileiraComprada)
escreva("Digite o número da coluna em que deseja comprar a poltrona:")
leia(colunaComprada)
poltrona[fileiraComprada][colunaComprada] = -1
escreva("\n_____
\n")
escreva("                                # PALCO #
\n")
para(inteiro linha = 0; linha < 5; linha++){
    escreva("\n")
    para (inteiro coluna = 0; coluna < 6; coluna++){
        se(poltrona[linha][coluna] == -1){
            escreva ( "      ***   | ")
        }senao{
            escreva ( "P(",linha, coluna, "): ", poltrona[linha][coluna], " | ")
        }
    }
}

```

```
        }
    }
    escreva("\n_____FINAL DO TEATRO_____
    \n\n")
```

//Saída no console
Mapa do teatro

PALCO

P(00): 0 | P(01): 0 | P(02): 0 | P(03): 0 | P(04): 0 | P(05): 0 |
P(10): 0 | P(11): 0 | P(12): 0 | P(13): 0 | P(14): 0 | P(15): 0 |
P(20): 0 | P(21): 0 | P(22): 0 | P(23): 0 | P(24): 0 | P(25): 0 |
P(30): 0 | P(31): 0 | P(32): 0 | P(33): 0 | P(34): 0 | P(35): 0 |
P(40): 0 | P(41): 0 | P(42): 0 | P(43): 0 | P(44): 0 | P(45): 0 |

FINAL DO TEATRO

Digite o número da fileira em que deseja comprar a poltrona:2
Digite o número da coluna em que deseja comprar a poltrona:3

PALCO

P(00): 0 | P(01): 0 | P(02): 0 | P(03): 0 | P(04): 0 | P(05): 0 |
P(10): 0 | P(11): 0 | P(12): 0 | P(13): 0 | P(14): 0 | P(15): 0 |
P(20): 0 | P(21): 0 | P(22): 0 | **** | P(24): 0 | P(25): 0 |
P(30): 0 | P(31): 0 | P(32): 0 | P(33): 0 | P(34): 0 | P(35): 0 |
P(40): 0 | P(41): 0 | P(42): 0 | P(43): 0 | P(44): 0 | P(45): 0 |

FINAL DO TEATRO

Repare que são criadas duas variáveis para que o usuário informe a fileira e a poltrona que deseja comprar. Em seguida, são

apresentados o mapa das poltronas e a mensagem para que o cliente informe a fileira e a poltrona desejada. Após a escolha da poltrona, é atribuído o valor -1 para a fileira e coluna desejada indicando que ela está ocupada. As poltronas vazias são indicadas pelo valor 0, inicializadas logo após a declaração da matriz.

Veja que na parte do algoritmo que exibe cada poltrona na tela é verificado se o valor da poltrona é igual a -1. Se for, em vez de mostrar a fileira e a coluna, o algoritmo apresenta 4 asteriscos.

Além do uso para representar mapa de locais, outro uso muito recomendado da matriz é para armazenar mais de um dado no formato de tabela. Veja o exemplo a seguir.

Vamos pensar juntos: você foi contratada(o) para desenvolver um sistema de exibição do boletim digital de um curso de inglês. Seu sistema deve armazenar e apresentar a informação de 15 alunos. Cada aluno passa por 6 avaliações em cada semestre (prova oral, prova escrita e conversação por bimestre). Além da nota dessas avaliações, seu sistema ainda deve registrar a média final do semestre de cada aluno.

De acordo com o enunciado, poderíamos montar o sistema utilizando 15 vetores com tamanho 7 cada um. Porém, como explicado no exemplo anterior, a manipulação dos dados ficaria complicada. Podemos simplificar essa complexidade utilizando apenas uma matriz com 15 linhas e 7 colunas cada. Veja o exemplo a seguir.

//Exemplo 71: Algoritmo para armazenar 6 notas de 15 alunos e mais a média final do aluno.

```

real notasAlunos[15][7], totalNotaAluno
inteiro linha, coluna

para(linha = 0; linha < 3; linha++){
    escreva("\n")
    totalNotaAluno = 0.0
    para(coluna = 0; coluna < 6; coluna++){
        escreva("Digite a ", coluna + 1, "ª nota do aluno ",
    linha +1, ":" )
        leia(notasAlunos[linha][coluna])
        totalNotaAluno = totalNotaAluno + notasAlunos[linha][
    coluna]
    }
    notasAlunos[linha][7] = totalNotaAluno/6
}

para(linha = 0; linha < 3; linha++){
    escreva("\n")
    para(coluna = 0; coluna < 7; coluna++){
        escreva(notasAlunos[linha][coluna], " - ")
    }
}

```

Observe que o `para` que realiza a leitura da nota dos alunos vai até o índice 5. O último índice, o 6, usamos para armazenar a média das notas do aluno. Normalmente quando conseguimos obter um dado a partir do cálculo entre outros dados, nesse caso, a média, não realizamos o armazenamento desse dado. Deixamos para realizar o cálculo todas as vezes que for necessário obter o dado e evitamos erros de falta de atualização do dado gerado. Porém, criamos nesse exemplo para explorarmos a utilização da matriz com dados tabulados.

Vamos pensar juntos: como podemos alterar o exemplo anterior para que, além das notas, tenhamos o nome do respectivo aluno.

//Exemplo 72: Algoritmo para armazenar 6 notas de 15 alunos e m
as a média final do aluno. O algoritmo também armazena o respectiv
o nome do aluno daquelas notas.

```
real notaAluno[15][7], totalNotaAluno
inteiro linha, coluna
cadeia nomeAluno[15]

para(linha = 0; linha < 2; linha++){
    escreva("\n")
    totalNotaAluno = 0.0
    escreva("Digite o nome do ", linha +1, "º aluno: ")
    leia(nomeAluno[linha])
    para(coluna = 0; coluna < 6; coluna++){
        escreva("Digite a ", coluna + 1, "ª nota do aluno ",
        nomeAluno[linha]," : ")
        leia(notaAluno[linha][coluna])
        totalNotaAluno = totalNotaAluno + notaAluno[linha][co
luna]
    }
    notaAluno[linha][6] = totalNotaAluno/6
}

para(linha = 0; linha < 2; linha++){
    escreva("\n\n")
    escreva("Aluno: \n", nomeAluno[linha], " - Notas: ")
    para(coluna = 0; coluna < 7; coluna++){
        se (coluna == 6){
            escreva(" Média final: ", notaAluno[linha][6])
            se(notaAluno[linha][6] >= 6){
                escreva(" - Aprovado(a)")
            }senao{
                escreva(" - Reprovado(a) - Reavaliação")
            }
        }senao{
            escreva(notaAluno[linha][coluna], " - ")
        }
    }
}

//Saída no console. Exemplo com apenas dois alunos.
```

Digite o nome do 1º aluno: João
Digite a 1ª nota do aluno João: 5.6
Digite a 2ª nota do aluno João: 9.2
Digite a 3ª nota do aluno João: 8.3
Digite a 4ª nota do aluno João: 4.5
Digite a 5ª nota do aluno João: 7.2
Digite a 6ª nota do aluno João: 8.8

Digite o nome do 2º aluno: Pedro
Digite a 1ª nota do aluno Pedro: 5.8
Digite a 2ª nota do aluno Pedro: 6.1
Digite a 3ª nota do aluno Pedro: 1.3
Digite a 4ª nota do aluno Pedro: 2.1
Digite a 5ª nota do aluno Pedro: 3
Digite a 6ª nota do aluno Pedro: 0.9

Aluno:

João - Notas: 5.6 - 9.2 - 8.3 - 4.5 - 7.2 - 8.8 - Média final: 7
.26666666666668 - Aprovado(a)

Aluno:

Pedro - Notas: 5.8 - 6.1 - 1.3 - 2.1 - 3.0 - 0.9 - Média final:
3.199999999999993 - Reprovado(a) - Reavaliação

Diferentemente da média, em que criamos apenas mais uma coluna, não podemos usar mais uma coluna para armazenar o nome, pois são tipos de dados diferentes. Para armazenar as notas, utilizamos real e, para armazenar o nome, utilizamos o tipo cadeia.

Uma maneira para atribuirmos o nome do aluno é criarmos um vetor do tipo cadeia e associarmos os índices da linha da matriz com o índice do vetor. Ou seja, utilizamos o índice do vetor 0, por exemplo, para armazenar o nome do primeiro aluno. As notas que devem ser cadastradas na linha 0 da matriz devem ser também do primeiro aluno. Essa mesma associação deve acontecer em todos os índices do vetor e da linha da matriz.

Veja que na saída do console é apresentado o nome associado de cada aluno.

Um problema que aparece na saída do console é que a média final aparece com muitas casas decimais. Para formatarmos a quantidade de casas decimais, precisamos usar uma biblioteca disponibilizada no Portugol Studio.

15.1 USANDO BIBLIOTECA PARA ARREDONDAR AS CASAS DECIMAIAS

As bibliotecas são conjuntos de comandos, que algum programador criou, disponibilizadas para uso de todos. No Portugol Studio, existem atualmente 9 bibliotecas que podemos utilizar, são elas: arquivo, gráficos, matemática, mouse, sons, teclado, texto, tipos e util. Em cada uma delas existem N comandos que podemos utilizar nos nossos algoritmos.

Para utilizarmos qualquer uma dessas 9 bibliotecas, precisamos, antes de tudo, incluí-las nos nossos algoritmos. Veja a sintaxe para incluir uma biblioteca a seguir.

SINTAXE DA DECLARAÇÃO DA BIBLIOTECA

```
programa{  
    inclua biblioteca nome_biblioteca --> apelido_biblioteca
```

O comando `inclua biblioteca` deve ser escrito dentro do par de chaves do `programa` e antes da função `inicio(){}`. O `nome_biblioteca` é o nome da biblioteca definido pela ferramenta Portugol Studio. O `apelido_biblioteca` é o apelido dado à biblioteca para ser utilizado dentro do nosso algoritmo.

Para usarmos a biblioteca, depois da declaração, basta chamarmos o apelido criado seguido de um ponto (.) e o nome do comando. Os comandos podem solicitar parâmetros, assim como acontece, por exemplo, com o comando `escreva` e `leia`.

No caso do comando para arredondar as casas decimais devemos passar dois parâmetros: o primeiro é a variável com o número que desejamos arredondar, e o segundo parâmetro é a quantidade de casas decimais que queremos que seja exibida. Veja o exemplo a seguir.

```
//Exemplo 73: Algoritmo para armazenar 6 notas de 15 alunos e mai  
s a média final do aluno. O algoritmo também armazena o respectiv  
o nome do aluno daquelas notas.
```

```
incluir biblioteca Matematica --> mat  
funcao inicio() {  
    real pi = 3.14159265358979323846, numeroArredondado  
    escreva ("PI: ", mat.arredondar(pi, 2))  
    escreva ("\n")  
    escreva ("PI arredondado: ", mat.arredondar(pi, 4))
```

```
//Saída no console.  
PI: 3.14  
PI arredondado: 3.1416
```

Veja que o comando para arredondar foi usado duas vezes, uma para exibir duas casas decimais e a outra para exibir quatro. Voltando ao exemplo da média dos alunos do curso de inglês, basta usar essa biblioteca e esse comando para resolver o problema das casas decimais na média final de cada aluno.

Para saber mais das bibliotecas disponíveis do Portugol Studio, acesse o *Ajuda* da ferramenta e clique no item *Bibliotecas*.

Nos próximos capítulos, estudaremos as funções.

Resumo do capítulo

Vimos neste capítulo as matrizes. Elas são estruturas homogêneas, iguais aos vetores, porém elas têm mais de uma dimensão. No vetor trabalhamos com apenas uma linha contendo várias colunas e na matriz trabalhamos com várias linhas e em cada linha temos várias colunas. Assim como nos vetores, os dados armazenados na matriz devem ser sempre do mesmo tipo. A declaração da matriz é igual à do vetor, exceto pelo acréscimo do colchete da quantidade de linhas que serão utilizadas. Essa informação da quantidade de linhas deve vir antes da informação da quantidade de colunas. Na manipulação da matriz deve-se utilizar dois comandos `para`, um dentro do outro. O mais de fora é responsável pela repetição das linhas e o mais de dentro pela repetição das colunas.

15.2 EXERCÍCIOS

47. (T) Crie um programa para armazenar as poltronas vendidas de ônibus interestadual com 42 poltronas. O sistema deve solicitar ao usuário a poltrona desejada e em seguida desabilitar a exibição da poltrona para venda.

48. (T) Você foi contratado para desenvolver um sistema para um estacionamento. O processo de parar o carro ocorre quando o cliente deixa o carro na porta do estacionamento para o

manobrista realizar a parada. Ao estacionar o carro, o manobrista reserva a vaga no sistema informando a placa do veículo. Ao chegar para buscar o carro, o cliente informa a placa do veículo ao manobrista que realiza a consulta no sistema. O manobrista descobre em qual vaga (índice da matriz) está o carro e torna a vaga livre para outro cliente. Atualmente o estacionamento conta com 18 vagas, sendo 9 em cada lado. Monte o sistema com um menu para exibir as vagas do estacionamento, cadastrar um veículo em uma vaga e desocupar uma vaga.

Legenda: F - fácil / M - médio / T - trabalhoso

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap15>



Figura 15.2: QRCode

Função



Figura 1: Fonte: Imagem do Pixabay.

Nestes capítulos, estudaremos as funções. Elas são utilizadas para modularizar os nossos códigos, diminuindo o conteúdo do programa principal e reutilizando trechos de código.

CAPÍTULO 16

FUNÇÃO

Neste capítulo, estudaremos as funções. Funções são pequenos programas (trechos de código), separados do programa principal, que realizam algum processamento específico e/ou que são utilizados várias vezes no algoritmo. Em vez de repetirmos um trecho de código várias vezes no programa principal, nós o separamos em uma função e no programa principal fazemos apenas a chamada para a função, em todos os locais desejados.

Até o momento, nós já tínhamos trabalhado com algumas funções criadas por outros programadores, por exemplo, o comando `leia` e o comando `escreva`.

Vamos pensar juntos: imagine que você tenha um trecho de código que realiza um cálculo complexo. Para a realização desse cálculo são necessárias 10 linhas. Esse cálculo é utilizado 3 vezes no seu algoritmo, logo você precisa de 30 linhas para que o cálculo seja executado 3 vezes no seu algoritmo. Agora imagine que você transforme essas 10 linhas do cálculo em uma função. Com isso, nos 3 locais do seu algoritmo que teriam o cálculo, você precisará apenas chamar a função. Para chamarmos uma função basta uma única linha. Ou seja, nesse exemplo hipotético, teríamos a necessidade de escrever 30 linhas para a realização do cálculo sem

função ou 3 linhas utilizando função. Uma redução de 90% na quantidade de linhas escritas. Isso, quando pensamos em algoritmos com centenas ou milhares de linhas, faz muita diferença.

Agora, como é que escrevemos essa tal de função então? As funções são compostas por diversas configurações. Começaremos pela função mais simples, que é a sem retorno e sem passagem de parâmetros.

16.1 FUNÇÃO SEM RETORNO

A função sem retorno é uma função que, ao finalizar a sua execução, não retornará nenhum valor para quem a chamou.

Quando chamamos (usamos) uma função, o programa principal para de ser executado e o compilador redireciona a sua execução para a função desejada. Ao finalizar a execução da função, o compilador retorna para o programa principal na linha da chamada da função. O retorno serve para indicar ao compilador se a função retornará ou não algum valor para o programa que realizou a chamada.

Veja a figura a seguir que apresenta a execução de uma chamada para uma função.

Programa principal

```
programa {  
    funcao inicio(){  
        comando  
        comando  
        comando  
        comando para função  
        comando  
        comando  
    }  
}
```

Função

```
funcao xxxx(){  
    comando  
    comando  
}
```

Figura 16.1: Chamada para uma função.

SINTAXE DA DECLARAÇÃO E CHAMADA DE UMA FUNÇÃO SEM RETORNO

```
programa{  
  
    //declaração da função  
    funcao nomeFuncao(){  
        // algoritmo da função  
    }  
  
    funcao inicio(){  
        // chamada para função  
        nomeFuncao()  
    }  
}
```

Na sintaxe, nomeFuncao é o nome da nova função criada. O nome da função deve seguir o mesmo padrão de criação de nomes das variáveis, vetores e matrizes. O algoritmo da função deve ser escrito dentro do par de chaves da função. Repare que a função deve ser declarada fora da funcao inicio e dentro do programa{ .

Após a declaração de uma função, ela já pode ser usada no programa principal (funcao inicio) ou em outras funções criadas. Para isso, basta realizar a **chamada** para a função. Essa chamada é composta do nome da função juntamente da abertura e do fechamento das chaves.

```
//Exemplo 74: Exemplo de função sem retorno

programa {
    funcao calcularPorcentagem(){
        escreva("Dentro da função calcular porcentagem\n")
        real numero = 30.0, porcentagem = 20.0, resultado
        resultado = (numero * (porcentagem/100))
        escreva ("O resultado da porcentagem é ", resultado)
    }
    funcao inicio() {
        escreva("Início do programa principal")
        escreva("\n")
        escreva("Final do programa principal")
    }
}

//Saída no console.
Início do programa principal
Final do programa principal
```

Veja que no exemplo 74 criamos uma função chamada `calcularPorcentagem`, com o procedimento para realização do cálculo da porcentagem e, no final, um `escreva` para apresentar o resultado na tela. Porém, ao analisarmos o console, não encontramos os resultados dos comandos `escreva` de dentro da função. Isso ocorre pois nós não fizemos a chamada para a função no programa principal.

Veja agora o exemplo 75 a seguir.

```
//Exemplo 75: Exemplo de função sem retorno

programa {
    funcao calcularPorcentagem(){
        escreva("Dentro da função calcular porcentagem\n")
        real numero = 30.0, porcentagem = 20.0, resultado
        resultado = (numero * (porcentagem/100))
        escreva ("O resultado da porcentagem é ", resultado)
    }

    funcao inicio() {
        escreva("Início do programa principal")
        escreva("\n")
        calcularPorcentagem()
        escreva("\n")
        escreva("Final do programa principal")
    }
}

//Saída no console.
Início do programa principal
Dentro da função calcular porcentagem
O resultado da porcentagem é 6.0
Final do programa principal
```

Veja que no exemplo 75 as mensagens dos comandos `escreva` de dentro da função aparecem no console.

Atenção, nos exemplos anteriores, os valores das variáveis estão fixos no código para que o foco do aprendizado seja exclusivamente no uso das funções. Após esse aprendizado, os algoritmos das funções devem ser escritos com todos os comandos que vimos até o momento, ou seja, podemos ter declaração de variáveis, condicionais, estruturas de repetição, vetores e matrizes.

Como dito anteriormente, ao encontrar uma chamada de uma função, o compilador para a execução do programa principal e direciona a execução para a função chamada. Isso pode ser observado pela ordem das mensagens que aparecem na saída no console do exemplo 75.

A primeira mensagem exibida é a "Início do programa principal" e, em seguida, já é apresentada a mensagem "Dentro da função calcular porcentagem". Isso ocorre pois, depois de mostrar a mensagem de início e pular linha, o compilador encontra uma chamada para função `calcularPorcentagem()`. Nesse exato momento, ele para de executar o programa principal e começa a executar a função. Assim que ele entra na função, o programa executa o `escreva`, declara as variáveis, realiza os cálculos e apresenta o resultado na tela. Quando o compilador encontra o fechamento da chave da função, ele finaliza a execução naquela função e retorna para o programa principal (exatamente na linha da chamada da função).

Vejamos outro exemplo.

```
//Exemplo 76: Exemplo de sistema para um nutricionista calcular a
dieta diária de seus pacientes. Inicialmente o sistema precisa calcular a
quantidade de água ideal por dia de acordo com peso.
```

```
programa {
    funcao calcularQtdAguaDiaria(){
```

```

        real peso, resultado
caracter praticaAtividadeFisica
faca{
    escreva ("Digite seu peso: ")
    leia(peso)
}enquanto (peso <=0)
faca{
    escreva("Digite S se você pratica atividade física
a ou N caso não pratique: ")
    leia(praticaAtividadeFisica)
}enquanto (praticaAtividadeFisica !='S' e practicaAtividadeFisica != 'N')
se(praticaAtividadeFisica == 'S'){
    resultado = peso * 0.04
}senao{
    resultado = peso * 0.035
}
escreva ("Você deve beber ", resultado, " litros de água por dia.")
}

funcao inicio() {
    escreva("Início do programa principal")
    escreva("\n")
    calcularQtdAguaDiaria()
    escreva("\n")
    escreva("Final do programa principal")
}
}

//Saída no console.
Início do programa principal
Digite seu peso: 62
Digite S se você pratica atividade física ou N caso não pratique:
S
Você deve beber 2.48 litros de água por dia.

```

Veja que o valor da quantidade de água foi apresentado na tela diretamente. Agora imagine que o seu algoritmo necessitasse do resultado do cálculo da água diária para ser usado em uma outra função ou em outra parte do programa. Como poderíamos resolver esse problema?

Da maneira como está implementado, utilizando função sem retorno, não teria como. A solução seria termos o código da função dentro do programa principal. Então já não faria sentido a utilização da função.

Porém, as funções podem retornar valores para o programa principal. Dessa maneira, resolveríamos o problema do resultado do cálculo e continuariamos utilizando funções.

16.2 FUNÇÃO COM RETORNO

As funções também podem retornar um valor para quem a chamou. Porém, cada função só pode retornar um único valor.

SINTAXE DA DECLARAÇÃO E CHAMADA DE UMA FUNÇÃO COM RETORNO

```
programa{  
  
    //declaração da função  
    funcao tipoRetorno nomeFuncao(){  
        // algoritmo da função  
  
        retorne retornoFuncao  
    }  
  
    funcao inicio(){  
        // declaração da variável do tipo retorno da função  
        tipoRetornoFuncao nomeVariavel  
        nomeVariavel = nomeFuncao()  
    }  
}
```

`tipoRetorno` é o tipo de dado que será retornado pela função. O retorno é realizado através do uso da palavra reservada `retorne`. A chamada da função passa a ter agora uma atribuição. Antes, na linha da chamada, só tínhamos o nome da função e os parênteses; agora, antes da chamada da função, teremos uma variável do mesmo tipo declarado no retorno da função. Complicado o conceito? Veja o exemplo a seguir que ficará mais fácil.

//Exemplo 77: Exemplo de função da porcentagem COM retorno da função. Compare com o exemplo 75 para entender as mudanças necessárias para a função calcularPorcentagem retornar um valor.

```
programa {
    funcao real calcularPorcentagem(){
        real numero = 30.0, porcentagem = 20.0, resultado
        resultado = (numero * (porcentagem/100))
        retorno resultado
    }

    funcao inicio() {
        real porcentagemCalculada
        porcentagemCalculada = calcularPorcentagem()
        escreva("A porcentagem calculada foi: ", porcentagemC
        alculada)
    }
}

//Saída no console.
A porcentagem calculada foi: 6.0
```

Perceba que a mensagem de saída agora está no programa principal e o cálculo continua sendo realizado dentro da função `calcularPorcentagem()`.

Agora se precisássemos que fosse retornada a porcentagem utilizada no cálculo e o resultado, não teria como fazer isso, pois, como dito anteriormente, toda função só pode retornar um único valor.

Outro item importante de se levantar é que os valores das variáveis da função continuam fixos. Como poderíamos alterá-los de tal maneira que a função recebesse esses valores do programa principal e em seguida realizasse o cálculo? Isso veremos no próximo capítulo com a passagem de parâmetros para função.

Resumo do capítulo

Vimos neste capítulo as funções. Elas são pequenos trechos de código utilizados para evitar a reescrita de código e/ou quando queremos criar códigos com responsabilidades específicas, como uma função responsável por realizar um cálculo complexo, função para determinar a porcentagem, função para calcular o desconto de uma compra, entre outras. As funções podem ser declaradas com ou sem valor de retorno. Caso a função tenha um valor de retorno, ao finalizar a sua execução, o compilador retorna o valor da função para quem a chamou. Esse valor precisa ser atribuído a uma variável no programa principal.

16.3 EXERCÍCIOS

49. (F) Crie um programa que solicite uma letra e no final diga se ela é vogal ou não. Esse programa deve utilizar uma função que retorne o resultado para o programa principal. A mensagem final deve ser exibida então no programa principal.

50. (M) Crie um programa para uma calculadora. Cada uma das operações deve ser uma função específica. O resultado da operação deve ser exibido dentro da função.

51. (T) Altere o programa da calculadora para que ele realize as 4 operações sempre. No final do programa, o algoritmo deve apresentar a soma do valor de retorno das 4 operações.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Transforme a exibição do menu principal em uma função sem retorno e sem parâmetros.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap16>



Figura 16.2: QRCode

CAPÍTULO 17

FUNÇÃO - PASSAGEM DE PARÂMETRO POR VALOR

No capítulo anterior, estudamos as funções com retorno ou sem retorno. Neste capítulo, veremos as funções com passagem de parâmetros.

Assim como o retorno é opcional, a passagem de parâmetro também é. A definição sobre se uma função terá retorno e/ou parâmetro dependerá da necessidade e da lógica criada para a função. Não existe uma regra para ser usada em todas as funções.

Parâmetros

Os parâmetros são valores informados dentro dos parênteses de alguma chamada de função. Precisamos utilizar os parâmetros pois, ao definirmos uma nova função, o compilador cria um novo escopo de variáveis.

Escopo de variáveis locais e globais

O escopo de uma variável é o local em que a variável existe. Vimos que as variáveis ocupam espaços na memória e que o compilador é o responsável por esse controle. Porém, quando

deixamos de usar uma variável (saímos do seu escopo), o compilador esquece esse espaço de memória, que fica livre para ser usado por outro programa.

O escopo das variáveis é classificado em **local** e **global**.

O escopo de uma **variável local** é a função ou o comando em que a variável foi criada. Ou seja, podemos definir que o **escopo de uma variável local** é a abertura e o fechamento da chave onde a variável foi criada.

Por exemplo, quando declaramos uma variável dentro do programa principal (`funcao inicio()`), como fizemos até agora nos nossos algoritmos, a variável é visualizada em todo o nosso programa principal, já que o escopo dela é o programa principal. Nesse caso, ela é uma variável local para a `funcao inicio()` . Se tentarmos acessar essa mesma variável em uma outra função, por exemplo, o compilador não a encontrará, já que cada função tem um escopo local de variáveis.

Vamos pensar juntos: analise o exemplo 78 e a saída no console e diga qual o escopo da variável `novaIdadeLocal` .

```
//Exemplo 78: Exemplo de variável local criada dentro de um escopo do comando se
    inteiro idade = 18
    se ( idade == 18){
        inteiro novaIdadeLocal = 20
    }
    escreva ("Nova idade é:", novaIdadeLocal)

//Saída no console.
ERRO: A variável "novaIdadeLocal" não foi declarada neste escopo.
. Linha: 9, Coluna: 28
```

Veja que, ao tentar executar o programa, o compilador apresenta a mensagem de erro indicando que a variável novaIdadeLocal não faz parte do escopo. Isso acontece pois a variável foi criada dentro do escopo do comando se . O escreva com a variável novaIdadeLocal está depois do fechamento da chave do se . Como ele está fora do escopo da criação da variável, ocorrerá um erro na execução do algoritmo.

Essa regra vale para todos os comandos e também para as funções que criaremos.

Porém, em alguns casos, precisamos que uma variável seja visualizada por várias funções ao mesmo tempo. Nesses casos, temos que usar as variáveis com **escopo global**.

A declaração das variáveis globais deve ser realizada fora de todas as funções e dentro do programa{ . Recomenda-se a utilização das variáveis globais em casos muito específicos, pois, como as variáveis ficam o tempo todo na memória, elas podem deixar o programa mais pesado e lento.

```
//Exemplo 79: Exemplo de variável global
programa {
    inteiro novaIdadeLocal
    funcao inicio() {
        inteiro idade = 18
        se ( idade == 18){
            novaIdadeLocal = 20
        }
        imprimirNovaIdade()
    }
    funcao imprimirNovaIdade(){
        escreva ("Nova idade é:", novaIdadeLocal)
    }
}

//Saída no console.
```

```
Nova idade é:20
```

Repare no exemplo 79 que a variável `novaIdadeLocal` foi declarada, agora, fora do função `inicio()`. Dessa maneira, a variável passou a ter seu escopo global e toda e qualquer função terá acesso ao valor dela. Repare que no console já não aparece mais a mensagem de erro e sim a mensagem com a variável escrita na função `imprimirNovaIdade()`.

Voltando aos parâmetros, quando passamos um valor dentro de parênteses, estamos atribuindo valores de um escopo (função) para um outro escopo (outra função). Existem dois tipos de passagem de parâmetros para as funções, o **por valor** e o **por referência**. Neste capítulo, estudaremos a passagem de parâmetros por valor.

17.1 FUNÇÃO COM PASSAGEM DE PARÂMETRO POR VALOR

Na passagem de parâmetro por valor, o valor que é passado para a função é copiado para dentro da função. Ou seja, é realizada uma cópia do valor em uma nova variável na memória. Com isso, se o valor for alterado dentro da função, ele não alterará o valor da variável de origem.

Vejamos o exemplo a seguir.

```
//Exemplo 80: Exemplo de função com passagem de parâmetro por valor.
```

```
programa {
    funcao real calcularPorcentagem(real numero, real porcentagem){
        real resultado
        resultado = (numero * (porcentagem/100))
```

```

        retorne resultado
    }

    funcao inicio() {
        real porcentagemCalculada
        porcentagemCalculada = calcularPorcentagem(30.0, 2
0.0)
        escreva("A porcentagem calculada foi: ", porcentagemC
alculada)
    }
}

//Saída no console.
A porcentagem calculada foi: 6.0

```

Rpare que, na declaração da função, foram declarados dois parâmetros: `real numero`, `real porcentagem`. Essas mesmas variáveis foram retiradas de dentro da função (exemplo 77). Isso ocorre pois a declaração de um parâmetro é na verdade a declaração de uma variável, só que dentro dos parênteses da função. As regras de nome e o funcionamento na memória são idênticas às variáveis declaradas até o momento.

Veja na imagem a seguir como o compilador separa duas variáveis para armazenar o valor copiado no parâmetro da função.

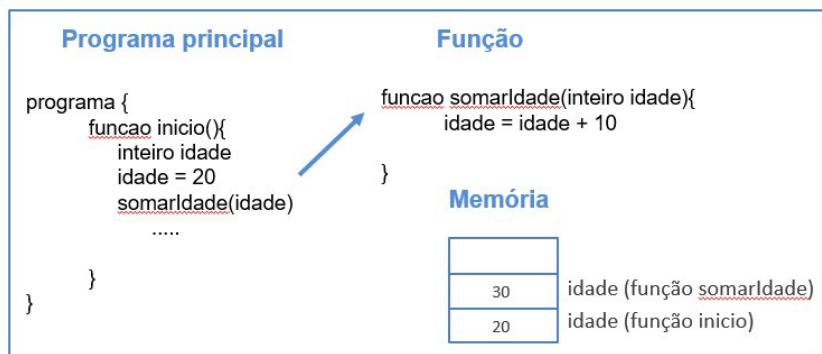


Figura 17.1: Memória na passagem de parâmetros por valor.

No exemplo da figura, foi utilizado o mesmo nome para duas variáveis diferentes (`idade`), porém não ocorrerá nenhum erro, já que cada variável foi criada dentro de um escopo diferente.

Voltando ao exemplo 80, é possível verificar que, além da declaração dos dois parâmetros na função, a chamada para a função no programa principal também foi alterada (`calcularPorcentagem(30.0, 20.0)`). Repare que agora são passados os valores dos parâmetros nos parênteses separados por vírgula.

Ao encontrar uma chamada para uma função, o compilador para de executar o programa atual e vai até a função. Como no exemplo a função tem em sua declaração dois parâmetros por valor, o compilador cria duas variáveis na memória e copia o valor passado nos parâmetros para as variáveis criadas (seguindo a ordem da declaração dos parâmetros). Em seguida, ele executa o código da função.

```
//Exemplo 81: Exemplo de função com passagem de parâmetro por valor.
programa {
    funcao real calcularPorcentagem(real numero, real porcentagem){
        real resultado
        resultado = (numero * (porcentagem/100))
        retorno resultado
    }
    funcao inicio() {
        real porcentagemCalculada, numeroDesejado, porcentagem
    m
        escreva ("Digite o número que deseja calcular a porce
ntagem: ")
        leia(numeroDesejado)
        escreva("Digite a porcentagem:")
        leia(porcentagem)
        porcentagemCalculada = calcularPorcentagem(numeroD
```

```

        desejado, porcentagem)
            escreva("A porcentagem calculada foi: ", porcentagemC
alculada)
        }
    }

//Saída no console.
Digite o número que deseja calcular a porcentagem: 30
Digite a porcentagem:20
A porcentagem calculada foi: 6.0

```

No exemplo 81, alteramos o algoritmo para que os valores passados para a função sejam informados pelo usuário através do teclado. Para isso, precisamos criar duas variáveis no programa principal (`numeroDesejado` e `porcentagem`).

Agora, em vez de passarmos dois valores fixos como parâmetro, estamos passando o valor de duas variáveis. Perceba que o nome de uma das variáveis é igual no programa principal e no parâmetro da função (variável `porcentagem`). Novamente, não ocorrerá nenhum erro pois cada uma das variáveis chamada `porcentagem` foi declarada em um escopo diferente.

O compilador separará na memória duas áreas chamadas de `porcentagem` , porém uma estará vinculada ao programa principal e a outra, à função `calcularPorcentagem` . Elas funcionam independentemente, mesmo que o valor da `porcentagem` seja alterado dentro da função `calcularPorcentagem` ela não alterará o valor da variável `porcentagem` do programa principal.

Veja outro exemplo de função com parâmetro por valor.

```

//Exemplo 82: Exemplo de função com passagem de parâmetro por val
or.
programa {
    funcao real calcularPrecoBiscoito(inteiro qtdBiscoito

```

```

, real valorBiscoito ){
    real resultado
    se (qtdBiscoito > 10){
        valorBiscoito = valorBiscoito * 0.9
        escreva("Desconto de 10% devido a quantidade
da compra. Valor R$", valorBiscoito)
    }
    resultado = valorBiscoito * qtdBiscoito
    retorno resultado
}
funcao inicio() {
    inteiro quantidadeBiscoito
    real valorBiscoito, valorPagar
    escreva ("Digite a quantidade de biscoitos deseja
da: ")
    leia(quantidadeBiscoito)
    escreva("Digite o valor do biscoito desejado:")
    leia(valorBiscoito)
    valorPagar = calcularPrecoBiscoito(quantidadeBisc
oito, valorBiscoito)
    escreva("\n")
    escreva("O biscoito sem desconto R$", valorBiscoi
to)
    escreva("\n")
    escreva("Valor total a ser pago é de R$", valorPa
gar)
}
}

//Saída no console.
Digite a quantidade de biscoitos desejada: 12
Digite o valor do biscoito desejado:5
Desconto de 10% devido a quantidade da compra. Valor R$4.5
0 biscoito custa sem desconto R$5.0
Valor total a ser pago é de R$54.0

```

Repare no exemplo 82 que são passados dois parâmetros para a função e cada um deles tem um tipo. A quantidade de parâmetros e os seus tipos vão variar de acordo com a necessidade da sua lógica. Veja também que, mesmo usando o nome igual das variáveis (valorBiscoito), um no programa principal e outro na função calcularPrecoBiscoito , eles são independentes, já que

o valor lido no teclado não é alterado quando a quantidade de biscoitos é maior que 10. Quando for maior que 10, o valor do biscoito é alterado com diminuição de 10% do seu valor.

Veja a seguir a sintaxe da função com parâmetro por valor.

SINTAXE DA DECLARAÇÃO DE UMA FUNÇÃO COM PARÂMETRO POR VALOR

```
programa{  
    //declaração da função  
    funcao nomeFuncao(tipoParametro nomeParametro){  
        // algoritmo da função  
    }  
  
    funcao inicio(){  
        // chamada para função  
        nomeFuncao(valorAserPassadoParaFunção)  
    }  
}
```

`tipoParametro` é o tipo de dado do parâmetro que a função deverá receber, que será utilizado como uma variável dentro da função. Os tipos disponíveis são os mesmos da declaração de uma variável. O `nomeParametro` é o nome da variável que será utilizada no parâmetro, as regras para criação de nomes são as mesmas para variáveis criadas no programa principal. Uma função pode definir quantos parâmetros forem necessários, separados por vírgula.

Na chamada da função, deve-se passar a quantidade e o tipo correto de cada parâmetro declarado na função. Pode-se utilizar valores fixos e/ou variáveis em cada parâmetro. Os parâmetros devem ser separados por vírgula.

Vale lembrar aqui que, assim como uma função sem parâmetros, as funções parametrizadas podem definir também um tipo de retorno. Nesse caso, teríamos a seguinte sintaxe de criação da função com parâmetros e com retorno.

SINTAXE DA DECLARAÇÃO DE UMA FUNÇÃO COM PARÂMETRO POR VALOR E COM RETORNO

```
programa{  
  
    //declaração da função  
    funcão tipoRetorno nomeFunção(tipoParametro1 nomeParametro1,  
        tipoParametro2 nomeParametro2, tipoParametro3 nomeParametro  
    3){  
        // algoritmo da função  
        retorno retornoFunção
```

```
}

funcao inicio(){
    // declaração da variável do tipo retorno da função
    tipoRetornoFuncao nomeVariavel

    // chamada para função, associando seu valor de retorno
    // a uma variável
    nomeVariavel = nomeFuncao(valorAserPassadoParaFunção1,
    valorAserPassadoParaFunção2, valorAserPassadoParaFunção3)
}
```

Na sintaxe, `tipoRetorno` é o tipo de dado que a função devolverá ao final de sua execução. Essa função utiliza três parâmetros diferentes e os tipos de parâmetros `tipoParametro1`, `tipoParametro2` e `tipoParametro3` definem os tipos dos valores que a função deverá receber em sua chamada, para ser executada corretamente. Os parâmetros podem ser de quaisquer tipos de dados e não precisam ser iguais (poderíamos utilizar o `tipoParametro1` como inteiro, o `tipoParametro2` como cadeia e `tipoParametro3` real, por exemplo).

O valor de `retornoFuncao` deverá ser o mesmo do `tipoRetorno` especificado na função.

A chamada da função passa a ser vinculada com uma variável, no escopo da função `inicio`. Nesse caso, após a execução do trecho de código definido dentro do escopo da função, será retornado um resultado que será armazenado em uma variável no escopo da função `inicio`.

No próximo capítulo, estudaremos as passagens de parâmetro por referência.

Resumo do capítulo

Vimos neste capítulo as funções com passagem de parâmetros por valor. Elas servem para enviar valores para dentro das funções. Na passagem por valor, é criada uma cópia do valor da variável de origem para a variável de destino (parâmetro). Caso ocorra alguma alteração no valor da variável dentro da função, essa alteração só ficará nessa variável. A variável que passou o valor para a função continuará com seu valor antigo. As funções podem ter N parâmetros de N tipos diferentes. Vimos também que uma variável pode ter seu escopo definido como local ou global. Variáveis locais só existem para o compilador enquanto não for encontrado o fechamento do par de chaves do local em que a variável foi criada. Por exemplo, uma variável criada dentro de uma estrutura de repetição `para` só existe enquanto o compilador estiver dentro do comando `para`. Ao encontrar o fechamento da chave do comando `para`, a variável não existirá mais. Já as variáveis globais existem em todo e qualquer escopo. Isso ocorre pois elas devem ser declaradas fora de toda e qualquer função. Elas são escritas logo na primeira linha após o comando `programa{`, assim em todo e qualquer lugar do programa a variável é visualizada. Porém, não se deve criar variáveis globais a todo momento, pois elas carregam a memória, o que pode deixar o seu programa mais lento e mais pesado.

17.2 EXERCÍCIOS

52. (M) Você foi contratado por uma loja de venda de livros usados na internet e tem que realizar a simulação do valor de entrega dos pedidos. O seu programa deve solicitar ao vendedor o total a ser pago pelo cliente e qual o prazo de entrega desejado (3, 5, 7 ou 10 dias úteis). Para cada tipo de entrega, deve ser criada uma nova função que receba o valor total pago pelo cliente. As entregas disponíveis são as seguintes: 1 - Entrega em 3 dias úteis (adicionar R\$ 25,00 reais ao valor pago pelo cliente), 2 - Entrega em 5 dias úteis (adicionar R\$ 20,00 reais ao valor pago pelo cliente), 3 - Entrega em 7 dias úteis (adicionar R\$ 15,00 reais ao valor pago pelo cliente) e 4 - Entrega em 10 dias úteis (adicionar R\$ 10,00 reais ao valor pago pelo cliente).

53. (T) Crie um programa para simular a gratificação de um vendedor de uma loja de carros usados. O sistema deve solicitar o salário básico do funcionário e o mês que deseja simular o salário

com a gratificação. A gratificação corresponde a 30% do salário básico do funcionário nos meses de janeiro até maio. De junho até novembro a gratificação corresponde a 40% do salário básico. Em dezembro, a gratificação equivale a 60% do salário. O sistema deve apresentar a gratificação dentro de uma função que receba o salário por parâmetro.

Legenda: F - fácil / M - médio / T - trabalhoso

Projeto de fixação: XPTO Bikes

Transforme o cálculo do desconto em uma função. A função deve receber por parâmetro o valor total do carrinho e retornar o valor do desconto.

Vídeo com o resumo do capítulo

<https://logica.comportugol.com.br/cap17>



Figura 17.2: QRCode

CAPÍTULO 18

FUNÇÃO - PASSAGEM DE PARÂMETRO POR REFERÊNCIA

No capítulo anterior, vimos as funções com passagem de parâmetro por valor. Agora veremos as funções com passagem de parâmetros por referência. Na passagem de parâmetro por referência, as variáveis criadas nos parâmetros da função não terão valores armazenados em suas áreas de memória. Em vez do valor, elas armazenam a referência para uma outra área da memória. Ou seja, é como se fossem duas variáveis apontando para a mesma região de memória. Complicado esse conceito, né? Veja a figura a seguir para facilitar o entendimento.

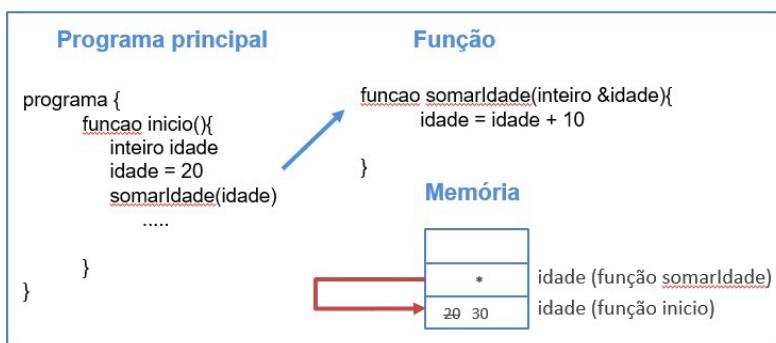


Figura 18.1: Memória na passagem de parâmetros por referência.

Para indicar que um parâmetro é passado por referência, basta incluir um `&` antes do nome da variável no parâmetro. Dessa maneira, o compilador indicará para a memória que aquela área não armazenará nenhum valor e sim apontará para a variável passada na chamada.

Qualquer alteração que ocorra em uma das duas variáveis é refletida na outra variável também. Essa é a diferença entre a passagem por valor e por referência.

E quando devemos usar cada uma delas? Isso depende da necessidade da função criada. Como estaremos criando as funções de acordo com uma necessidade, precisamos avaliar se as alterações realizadas na variável precisam ser replicadas para o programa que a chamou ou se elas não podem alterar o valor da variável que a chamou.

Vejamos o seguinte exemplo: imagine que você trabalha em uma empresa e quer calcular quanto de empréstimo na folha de pagamento você pode adquirir. No sistema de pagamento, existe uma funcionalidade para realizar essa simulação.

Ao realizar uma simulação, o programa apresenta como ficaria o seu novo salário, já com o desconto do empréstimo. Repare que você estará realizando uma simulação, logo as alterações não devem impactar no seu salário. Perceba que, nesse caso, o valor do seu salário precisa ser copiado para uma nova variável realizar o cálculo, sem impactar nos seus rendimentos mensais. Nesse caso, recomenda-se a utilização de uma função com passagem por valor, já que será realizada uma cópia do valor do seu salário para a função realizar a simulação, por exemplo.

Agora se você decide adquirir o empréstimo, veja que o cálculo realizado em uma função terá que refletir no seu salário final. Nesse caso, seria recomendado passar para a função uma referência para a variável que armazena o seu salário.

Veja a sintaxe da passagem por referência.

SINTAXE DA DECLARAÇÃO DE UMA FUNÇÃO COM PARÂMETRO POR REFERÊNCIA

```
programa{  
  
    //declaração da função  
    funcao nomeFuncao(tipoParametro &nomeParametro){  
        // algoritmo da função  
    }  
  
    funcao inicio(){  
        // chamada para função  
        nomeFuncao(valorAserPassadoParaFunção)  
    }  
}
```

Todos os campos são iguais à passagem por valor, exceto o `&`, que é incluído antes do nome do parâmetro.

Vejamos o exemplo a seguir.

```
//Exemplo 83: Exemplo de função com passagem de parâmetro por valor.  
programa {  
    funcao real calcularPrecoBiscoito(inteiro qtdBiscoito, real valorBiscoito ){  
        real resultado  
        se (qtdBiscoito > 10){  
            valorBiscoito = valorBiscoito * 0.9  
        }  
    }  
}
```

```

        escreva("Desconto de 10% devido à quantidade da compr
a. Valor R$", valorBiscoito)
    }
    resultado = valorBiscoito * qtdBiscoito
    retorno resultado
}
funcao inicio() {
    inteiro quantidadeBiscoito
    real valorBiscoito, valorPagar
    escreva ("Digite a quantidade de biscoitos desejada: ")
    leia(quantidadeBiscoito)
    escreva("Digite o valor do biscoito desejado:")
    leia(valorBiscoito)
    valorPagar = calcularPrecoBiscoito(quantidadeBiscoito, va
lorBiscoito)
    escreva("\n")
    escreva("O biscoito sem desconto R$", valorBiscoito)
    escreva("\n")
    escreva("Valor total a ser pago é de R$", valorPagar)
}
}

//Saída no console.
Digite a quantidade de biscoitos desejada: 12
Digite o valor do biscoito desejado:5
Desconto de 10% devido à quantidade da compra. Valor R$4.5
O biscoito custa sem desconto R$5.0
Valor total a ser pago é de R$54.0

```

Como dito anteriormente, na passagem por valor, a variável de origem não tem seu valor alterado. Veja que existem duas variáveis com o mesmo nome (valorBiscoito) e que a variável de origem é passada por valor para a função calcularPrecoBiscoito . O valor do preço do biscoito é alterado dentro da função, pois foi informada uma quantidade de pacotes maior que 10 (12 pacotes de biscoito no console). Porém, repare que essa alteração não é refletida no programa principal, já que a mensagem "O valor sem desconto" continua com o preço de R\$5.00, conforme informado pelo usuário.

Vejamos o mesmo exemplo a seguir só que agora utilizando passagem por referência.

```
//Exemplo 84: Exemplo de função com passagem de parâmetro por valor e por referência.
programa {
    funcao real calcularPrecoBiscoito(inteiro qtdBiscoito, real & valorBiscoito ){
        real resultado
        se (qtdBiscoito > 10){
            valorBiscoito = valorBiscoito * 0.9
            escreva("Desconto de 10% devido a quantidade da compra. Valor R$", valorBiscoito)
        }
        resultado = valorBiscoito * qtdBiscoito
        retorno resultado
    }
    funcao inicio() {
        inteiro quantidadeBiscoito
        real valorBiscoito, valorPagar
        escreva ("Digite a quantidade de biscoitos desejada: ")
        leia(quantidadeBiscoito)
        escreva("Digite o valor do biscoito desejado:")
        leia(valorBiscoito)
        valorPagar = calcularPrecoBiscoito(quantidadeBiscoito, valorBiscoito)
        escreva("\n")
        escreva("O biscoito sem desconto R$", valorBiscoito)
        escreva("\n")
        escreva("Valor total a ser pago é de R$", valorPagar)
    }
}

//Saída no console.
Digite a quantidade de biscoitos desejada: 12
Digite o valor do biscoito desejado:5
Desconto de 10% devido a quantidade da compra. Valor R$4.5
O biscoito custa sem desconto R$4.5
Valor total a ser pago é de R$54.0
```

Repare que a única diferença entre os exemplos 83 e 84 é o `&` na frente da declaração do parâmetro `valorBiscoito`. Como dito anteriormente, o `&` serve para indicar ao compilador que esse

parâmetro será por passagem por referência. Compare agora o console dos dois exemplos e veja que as entradas do usuário foram as mesmas. Já a saída teve o valor do biscoito alterado na mensagem "O biscoito custa sem desconto". No exemplo 83, o valor do biscoito continuava como R\$ 5.0 e agora, no 84, ele foi alterado para R\$4.5. O cálculo que gerou essa mudança foi realizado dentro da função `calcularPrecoBiscoito`.

A passagem por referência é muito utilizada quando temos uma função que precisa retornar mais de um valor para o programa principal. Como todas as funções podem retornar apenas um único valor, utiliza-se a passagem por referência para que as alterações dentro da função reflitam nas variáveis do programa principal.

Veja um exemplo com vários parâmetros por referência.

```
//Exemplo 85: Exemplo de função com várias passagens de parâmetro por referência.
funcao fase(inteiro faseAtual, inteiro &qtdMoedas, inteiro &pontuacao, inteiro &qtdChances){
    inteiro resposta
    se(faseAtual == 1){
        escreva("\nPergunta 1: Qual valor da soma de 325 + 122 = ")
        leia(resposta)
        se (resposta == 447){
            qtdMoedas = qtdMoedas + 5
            pontuacao = pontuacao + 500
        }senao{
            qtdChances--
            pontuacao = pontuacao - 5
            faca{
                escreva("\nNova pergunta 1: Qual valor da soma de
821 + 55 = ")
                leia(resposta)
                se (resposta != 876){
                    pontuacao = pontuacao - 10
                }
            }
        }
    }
}
```

```

        qtdChances--
    }senao{
        pontuacao = pontuacao + 100
        qtdMoedas = qtdMoedas + 2
    }
}enquanto (resposta != 876 e qtdChances > 0 )
}
}senao{
    escreva("\nPergunta 2: Qual valor da multiplicação de 20
* 6 = ")
    leia(resposta)
    se (resposta == 120){
        qtdMoedas = qtdMoedas * 5
        pontuacao = pontuacao * 50
    }senao{
        qtdChances--
        pontuacao = pontuacao - 50
        faca{
            escreva("\nNova pergunta 2: Qual valor da multipl
icação de 4 * 3 = ")
            leia(resposta)
            se (resposta != 12){
                pontuacao = pontuacao - 100
                qtdChances--
            }senao{
                pontuacao = pontuacao * 10
                qtdMoedas = qtdMoedas * 2
            }
}enquanto (resposta != 12 e qtdChances > 0 )
}
}
exibirPlacar(qtdMoedas, pontuacao, qtdChances)
}
funcao exibirPlacar(inteiro &qtdMoedas, inteiro &pontuacao, inteiro &qtdChances){
    escreva("\n##### PLACAR #####\n")
    escreva("Você ainda tem ", qtdChances, " chance(s).")
    escreva("\n")
    escreva("Moedas: ", qtdMoedas, " - Pontuação: ", pontuacao)
    escreva("\n##### FIM #####\n")
}
funcao inicio() {
    inteiro qtdMoedas = 0, pontuacao = 0, qtdChances = 3
    escreva("bem-vindo ao início do jogo de pergunta e resposta.\n")
}

```

```
n")
    exibirPlacar(qtdMoedas, pontuacao, qtdChances)
    fase(1, qtdMoedas, pontuacao, qtdChances)
    se(qtdChances > 0){
        fase(2, qtdMoedas, pontuacao, qtdChances)
    }
    escreva ("Fim do jogo")
}
```

```
//Saída no console.
bem-vindo ao início do jogo de pergunta e resposta.
```

```
##### PLACAR #####
Você ainda tem 3 chance(s).
Moedas: 0 - Pontuação: 0
##### FIM #####
```

```
Pergunta 1: Qual valor da soma de 325 + 122 = 447
```

```
##### PLACAR #####
Você ainda tem 3 chance(s).
Moedas: 5 - Pontuação: 500
##### FIM #####
```

```
Pergunta 2: Qual valor da multiplicação de 20 * 6 = 120
```

```
##### PLACAR #####
Você ainda tem 3 chance(s).
Moedas: 25 - Pontuação: 25000
##### FIM #####
Fim do jogo
Programa finalizado.
```

Repare que foi criado um exemplo de um jogo onde a cada jogada devem ser atualizadas as moedas, a pontuação e a quantidade de chances do usuário. Como são vários valores que precisam ser atualizados, não teria como usarmos somente o retorno da função, já que ele só pode retornar um único valor. Para resolvemos esse problema, passamos os parâmetros que devem ser atualizados via referência e dessa forma todos eles são atualizados no programa principal.

Resumo do capítulo

Vimos neste capítulo as funções com passagem de parâmetros por referência. A diferença entre a passagem por valor e por referência é que no valor é realizada uma cópia do valor da variável, ou seja, o valor é duplicado em uma nova variável do escopo da função. Já a passagem por referência significa que a variável criada na função não terá valor, ela terá um apontamento para a variável utilizada na chamada da função. Assim as duas variáveis apontam para o mesmo espaço de memória. Ao alterar o valor de uma dessas variáveis, essa alteração refletirá na outra variável também. Para indicar que um parâmetro utiliza a passagem por referência, basta incluir um & antes do nome da variável na declaração do parâmetro na função.

18.1 EXERCÍCIOS

54. (M) Altere o exercício 52 para que ele funcione com passagem de parâmetro por referência em vez da passagem de parâmetro por valor.

55. (M) Você foi contratado para desenvolver um sistema que atualiza o valor do preço dos combustíveis de um posto de gasolina. Seu sistema deve reajustar o valor da gasolina em uma função e do etanol em outra função. O reajuste da gasolina é somado ao valor atual da gasolina. O reajuste do etanol também deve ser adicionado ao valor atual do etanol, porém ao reajustar o etanol também deve-se atualizar a gasolina somando 27% do valor do reajuste do etanol no valor atual da gasolina. Seu programa deve solicitar o valor atual da gasolina, do etanol, valor do reajuste e qual o combustível do reajuste (G-gasolina E-etanol). Ao final, o seu programa deve mostrar o valor final da gasolina e do etanol.

Legenda: F - fácil / M - médio / T - trabalhoso

Vídeo com o resumo do capítulo

<https://logicacomportugol.com.br/cap18>



Figura 18.2: QRCode

CAPÍTULO 19

PARABÉNS

Parabéns pelo seu esforço e seu empenho, chegamos ao final do livro de Lógica de Programação com Portugol. Passou rápido, né? O primeiro contato com a lógica de programação é complicada, porém com o tempo ela vai se tornando natural.

Queríamos agradecer a confiança em acreditar no nosso livro para auxiliar no seu processo de aprendizagem. Colocamos aqui nossos vários anos de experiência vividos em sala de aula ensinando alunos de cursos técnicos e de graduação com as primeiras disciplinas de algoritmos e lógica de programação.

Acreditamos que, ao ler este livro e fazer todos os exercícios, você conseguiu subir vários degraus na área de algoritmos e lógica de programação.

Esse é o primeiro passo para se tornar um bom programador, uma boa desenvolvedora ou analista. Como próximo passo, sugerimos escolher uma linguagem de programação utilizada no mercado, como: C, Python, C#, PHP, JavaScript, Java, entre outras, e começar a estudar banco de dados também. Caso tenha interesse na área de programação para internet, sugerimos ainda estudar HTML5 e CSS3.

Qualquer dúvida, críticas ou sugestão entre em contato

conosco através do site do livro:

<http://www.logicacomportugol.com.br>

Nossos agradecimentos,

Professora Mestra Joice Barbosa Mendes

Professor Mestre Rafael da Silva Muniz

Resolução dos exercícios



Figura 1: Fonte: Imagem do Pixabay - peggy_marco-1553824/

Nestes capítulos, são apresentadas as resoluções dos exercícios do livro e do projeto da loja de bicicleta XPTO. Lembrando que um mesmo problema (exercício) pode ser resolvido de diversas maneiras e aqui apresentamos uma das possíveis soluções.

CAPÍTULO 20

RESOLUÇÃO DOS EXERCÍCIOS

CÓDIGO-FONTE DOS EXERCÍCIOS

As resoluções dos exercícios podem ser baixadas do link:
<https://logicacomportugol.com.br/codigo/exercicios.zip>

20.1 CAPÍTULO 1

//Exercício 1.

- a. Arrumar a cama.
- b. Preparar o café da manhã.
- c. Lavar a louça **do** café da manhã.
- d. Tomar banho.
- e. Colocar roupa **do** trabalho.
- f. Escovar o dente.
- g. Ir para escola.

//Exercício 2.

2.1 Resumido

- a. Abrir o navegador.
- b. Digitar o endereço **do** site.
- c. Apertar o enter ou o ir.

2.2 Detalhado

- a. Ligar o computador.
- b. Entrar com a senha **do** computador.
- c. Procurar o ícone **do** navegador.
- d. Clicar duas vezes no ícone.
- e. Clicar na barra de endereço **do** navegador.
- f. Digitar o endereço **do** site.
- g. Apertar a tecla enter ou o ícone ir **do** navegador.
- h. Aguardar o carregamento.

20.2 CAPÍTULO 2

//Exercício 3.

Não tem gabarito pois é para instalar o software no seu computador ou rodar o algoritmo **do** exemplo.

20.3 CAPÍTULO 3

```
// Exercício 4.  
programa {  
    funcao inicio() {  
        escreva("Sejam bem-vindos ao livro sobre Algoritmos da Ca  
sa do Código")  
    }  
}
```

20.4 CAPÍTULO 4

```
// Exercício 5.  
programa {  
    funcao inicio() {  
        real valorProduto, totalPagamentoCliente  
        inteiro codigoBarraProduto  
    }  
}
```

```
// Exercício 6.  
programa {  
    funcao inicio() {  
        cadeia nomeAluno, turmaAluno  
        inteiro idadeAluno
```

```

        real notaProva1, notaProva2
        nomeAluno = "Leonardo"
        turmaAluno = "1º série"
        idadeAluno = 7
        notaProva1 = 8.5
        notaProva2 = 9.0
        escreva("Nome: ", nomeAluno, " - Idade: ", idadeAluno)
        escreva("\n")
        escreva("Turma: ", turmaAluno)
        escreva("\n")
        escreva("Nota 1: ", notaProva1, " | Nota 2: ", notaProva2
    )
}
}

// Exercício 7.
programa {
    funcao inicio() {
        cadeia tipoRoupa = "Camiseta"
        inteiro tamanhoRoupa = 38
        real valorRoupa = 49.99
        escreva("A ", tipoRoupa, " no tamanho ", tamanhoRoupa, " "
está custando R$ ", valorRoupa)
    }
}

```

20.5 CAPÍTULO 5

```

// Exercício 8.
programa {
    funcao inicio() {
        inteiro num1 = 100, num2 = 5, num3 = 2
        escreva("Resultado: ", (num1 + num2) * num3)
    }
}

// Exercício 9.
programa {
    funcao inicio() {
        real nota1 = 7.5
        real nota2 = 6.0, resultado
        resultado = (nota1 + nota2) / 2
        escreva("Resultado: ", resultado)
    }
}

```

```
    }
}
```

20.6 CAPÍTULO 6

```
// Exercício 10.
programa {
    funcao inicio() {
        cadeia nome
        inteiro idade
        real altura
        escreva("Digite seu nome:")
        leia(nome)
        escreva("Digite sua idade:")
        leia(idade)
        escreva("Digite sua altura:")
        leia(altura)
        escreva("\n")
        escreva("Nome: ", nome, " - Idade: ", idade, " Altura: ",
altura)
    }
}

// Exercício 11.
programa {
    funcao inicio() {
        cadeia nome, frase
        escreva("Digite seu nome:")
        leia(nome)
        escreva("Digite sua frase preferida:")
        leia(frase)
        escreva("A frase preferida da(o) ", nome, " é ", frase, ".")
    }
}

// Exercício 12.
programa {
    funcao inicio() {
        cadeia vaga = "Técnico em informática"
        inteiro qtdVagas = 2
        real salario = 2.500
        caracter formacaoMinima = 'T'
```

```

logico vagaAberta = verdadeiro
escreva("Empresa contratando para a(s) seguinte(s) vaga(s)\n")
n")
    escreva("Vaga: ", vaga)
    escreva("\n")
    escreva("Quantidade: ", qtdVagas)
    escreva("\n")
    escreva("Salário: ", salario)
    escreva("\n")
    escreva("Formação: ", formacaoMinima, " - {onde: T - técnic
a e G - graduação}")
    escreva("\n")
    escreva("Vaga ainda aberta: ", vagaAberta)
}
}

```

20.7 CAPÍTULO 7

```

// Exercício 13.
programa {
    funcao inicio() {
        inteiro numero
        escreva("Digite um número: ")
        leia(numero)
        se (numero == 5){
            escreva("Você acertou!")
        }
    }
}

// Exercício 14.
programa {
    funcao inicio() {
        real velocidade
        escreva("Digite a velocidade: ")
        leia(velocidade)
        se (velocidade > 80.0){
            escreva("MULTADO")
        }
    }
}

```

```

// Exercício 15.
programa {
    funcao inicio() {
        real nota1, nota2, media
        escreva("Digite a primeira nota: ")
        leia(nota1)
        escreva("Digite a segunda nota: ")
        leia(nota2)
        media = (nota1+nota2)/2
        se (media >= 6.0){
            escreva("APROVADO(A)")
        }senao{
            escreva("EM RECUPERAÇÃO")
        }
    }
}

// Exercício 16.
programa {
    funcao inicio() {
        real nota1, nota2, media, recuperacao
        escreva("Digite a primeira nota: ")
        leia(nota1)
        escreva("Digite a segunda nota: ")
        leia(nota2)
        media = (nota1+nota2)/2
        se (media >= 6.0){
            escreva("APROVADO(A)")
        }senao{
            escreva("EM RECUPERAÇÃO\n")
            escreva("Digite a nota da recuperação: ")
            leia(recuperacao)
            se (recuperacao >= 5.0){
                escreva("APROVADO(A)")
            }senao{
                escreva("REPROVADO(A)")
            }
        }
    }
}

// Exercício 17.
programa {
    funcao inicio() {
        inteiro quantidadeSucos

```

```

        escreva("Loja de sucos.\n")
        escreva("Preços: 1 suco -> R$ 5.50 | Mais de 10 suco
s -> R$ 4.50 cada.\n")
        escreva("Digite a quantidade de sucos: ")
        leia(quantidadeSucos)
        se (quantidadeSucos >= 10){
            escreva("Você comprou ", quantidadeSucos, " suco
(s). \nPagar: ", quantidadeSucos * 4.50)
        }senao{
            escreva("Você comprou ", quantidadeSucos, " suco
(s). \nPagar: ", quantidadeSucos * 5.50)
        }
    }

// Exercício 18.
programa {
    funcao inicio() {
        cadeia prioridade
        escreva("Digite sim se precisa de prioridade:")
        leia(prioridade)
        se (prioridade == "sim"){
            escreva("Vá para os caixas 1, 2 e 3.")
        }senao{
            escreva("Vá para qualquer caixa, exceto os 1, 2
e 3 que são prioritários.")
        }
    }
}

// Exercício 19.
programa {
    funcao inicio() {
        real precoGasolina, precoEtanol, resultado
        escreva("Digite o preço da etanol:")
        leia(precoEtanol)
        escreva("Digite o preço da gasolina:")
        leia(precoGasolina)
        resultado = precoEtanol / precoGasolina
        se (resultado >= 0.7){
            escreva("Compensa abastecer com gasolina.")
        }senao{
            escreva("Compensa abastecer com etanol.")
        }
    }
}

```

```

        }
    }

// Exercício 20.
programa {
    funcao inicio() {
        inteiro numero
        escreva("Digite um número:")
        leia(numero)
        se (numero > 0){
            escreva("Número positivo")
        }senao{
            escreva("Número negativo")
        }
    }
}

```

20.8 CAPÍTULO 8

```

// Exercício 21.
programa {
    funcao inicio() {
        inteiro diaSemana
        escreva("Digite um número de 1 até 7:")
        leia(diaSemana)
        se (diaSemana == 1){
            escreva("Domingo")
        }senao se (diaSemana == 2){
            escreva("Segunda")
        }senao se (diaSemana == 3){
            escreva("Terça")
        }senao se (diaSemana == 4){
            escreva("Quarta")
        }senao se (diaSemana == 5){
            escreva("Quinta")
        }senao se (diaSemana == 6){
            escreva("Sexta")
        }senao se (diaSemana == 7){
            escreva("Sábado")
        }senao{
            escreva("Número inválido.")
        }
}

```

```

        }
    }

// Exercício 22.
programa
{
    funcao inicio ()
    {
        caracter letra
        escreva("Digite uma das seguintes letras: L, M, A e U.")
        leia(letra)
        se(letra == 'L'){
            escreva("Suco de laranja - vitamina C")
        }senao se(letra == 'M'){
            escreva("Suco de morango - vitamina A")
        }senao se(letra == 'A'){
            escreva("Suco de acerola - vitamina C")
        }senao se(letra == 'U'){
            escreva("Suco de uva - vitamina E")
        }senao{
            escreva("Letra inválida")
        }
    }
}

// Exercício 23.
programa
{
    funcao inicio ()
    {
        cadeia estacaoAno
        escreva("Digite uma estação do ano:")
        leia(estacaoAno)
        se(estacaoAno == "outono"){
            escreva("Outono começa no dia 20 de março")
        }senao se(estacaoAno == "inverno"){
            escreva("Inverno começa no dia 21 junho")
        }senao se(estacaoAno == "primavera"){
            escreva("Primavera começa no dia 22 setembro")
        }senao se(estacaoAno == "verão"){
            escreva("Verão começa no dia 21 de dezembro")
        }senao{
            escreva("Estação inválida")
        }
    }
}

```

```

        }
    }

// Exercício 24.
programa
{
    funcao inicio ()
    {
        caracter vogal
        escreva("Digite uma vogal:")
        leia(vogal)
        se(vogal == 'a'){
            escreva("Palavras: amora, acerola, amanhã, ajuda,...")
        }
        }senao se(vogal == 'e'){
            escreva("Palavras: escola, estudante, espada,... ")
        }senao se(vogal == 'i'){
            escreva("Palavras: igreja, irmão, imã,...")
        }senao se(vogal == 'o'){
            escreva("Palavras: obra, orquestra, ostra,...")
        }senao se(vogal == 'u'){
            escreva("Palavras: umbigo, universidade, uva,...")
        }senao{
            escreva("Vogal inválida")
        }
    }
}

```

20.9 CAPÍTULO 9

```

// Exercício 25.
programa
{
    funcao inicio ()
    {
        real peso, altura, imc=0.0
        escreva("Digite um peso:")
        leia(peso)
        escreva("Digite uma altura:")
        leia(altura)
        imc = peso / (altura * altura)
        escreva ("IMC :", imc)
        escreva ("\n")
    }
}

```

```

        se (imc < 18.5){
            escreva("Indicador: Magreza")
        }senao se(imc >= 18.5 e imc <= 24.9){
            escreva("Indicador: Normal")
        }senao se(imc > 24.9 e imc < 30.0){
            escreva("Indicador: Sobre peso")
        }senao se(imc >= 30.0){
            escreva("Indicador: Obesidade")
        }
    }
}

// Exercício 26.
programa
{
    funcao inicio ()
    {
        inteiro idade, habilitadoB, habilitadoC
        caracter infracaoUltimosDozeMeses
        escreva("Digite sua idade:")
        leia(idade)
        escreva("Digite em meses quanto tempo de habilitado na ca
rteira B: ")
        leia(habilitadoB)
        escreva("Digite em meses quanto tempo de habilitado na ca
rteira C: ")
        leia(habilitadoC)
        escreva("Digite s ou n se tiver alguma infração nos últim
os 12 meses: ")
        leia(infracaoUltimosDozeMeses)
        se (idade >= 21 e (habilitadoB >= 24 ou habilitadoC >= 12
) e infracaoUltimosDozeMeses == 'n'){
            escreva("APTO")
        }senao{
            escreva("NÃO APTO")
        }
    }
}

// Exercício 27.
programa
{
    funcao inicio ()
    {
        inteiro qtdPaes, qtdBisnaga, qtdForma, qtdQueijo

```

```

    inteiro qtdLeite, qtdPaoDoce, qtdSuspiro
    real precoPao = 0.30, precoBisnaga = 2.0, precoForma = 7.
20
    real precoQueijo = 12.00, precoLeite = 4.80, precoPaoDoce
= 5.30
    real precoSuspiro = 2.00, valorParcial = 0.0, valorTotal
= 0.0, desconto=0.0, porcentagemDesconto = 0.0
    escreva("Digite a qtd de pães:")
    leia(qtdPaes)
    escreva("Digite a qtd de bisnagas:")
    leia(qtdBisnaga)
    escreva("Digite a qtd de pão de forma:")
    leia(qtdForma)
    escreva("Digite a qtd de queijos:")
    leia(qtdQueijo)
    escreva("Digite a qtd de leites:")
    leia(qtdLeite)
    escreva("Digite a qtd de pão de doce:")
    leia(qtdPaoDoce)
    escreva("Digite a qtd de suspiro:")
    leia(qtdSuspiro)
    se ((qtdLeite >= 1 e qtdPaoDoce >=1) ou qtdSuspiro >=1 ){
        porcentagemDesconto = 5.0
    }
    se (qtdPaes >= 10 e qtdQueijo >= 1){
        porcentagemDesconto = 10.0
    }
    se (qtdBisnaga >= 1 ou qtdForma >= 1 ){
        porcentagemDesconto = 15.0
    }
    valorParcial = (qtdPaes * precoPao) + (qtdQueijo * precoQ
ueijo) + (qtdBisnaga * precoBisnaga) + (qtdForma * precoForma) +
(qtdSuspiro * precoSuspiro) + (qtdLeite * precoLeite)
    se(porcentagemDesconto > 0){
        desconto = valorParcial * (porcentagemDesconto/100.0)
    }
    valorTotal = valorParcial - desconto
    escreva("\nValor parcial: ", valorParcial)
    escreva("\nDesconto: ", porcentagemDesconto, "% | Valor d
esconto: R$", desconto)
    escreva("\nValor parcial: R$", valorTotal)
}
}

// Exercício 28.

```

```

programa
{
    funcao inicio ()
    {
        inteiro qtdFaltas
        real notaTeste, notaProva, media
        escreva("Digite a quantidade de faltas do bimestre: ")
        leia(qtdFaltas)
        escreva("Digite a nota do teste do bimestre: ")
        leia(notaTeste)
        escreva("Digite a nota da prova do bimestre:")
        leia(notaProva)
        media = (notaTeste + notaProva)/2
        se (qtdFaltas >= 10 e media >=7.0){
            escreva("APROVADA(0)")
        }senao se(qtdFaltas >= 10 e (media >= 5.9 e media < 7.0)){
            escreva("RECUPERAÇÃO")
        }senao{
            escreva ("REPROVADA(0)")
        }
    }
}

```

20.10 CAPÍTULO 10

```

// Exercício 29.
programa
{
    funcao inicio ()
    {
        inteiro numero
        escreva("Digite um número de 1 até 12: ")
        leia (numero)
        escolha(numero){
            caso 1:
                escreva("Janeiro")
                pare
            caso 2:
                escreva("Fevereiro")
                pare
            caso 3:
                escreva("Março")
                pare
        }
    }
}

```

```

caso 4:
    escreva("Abril")
pare
caso 5:
    escreva("Maio")
pare
caso 6:
    escreva("Junho")
pare
caso 7:
    escreva("Julho")
pare
caso 8:
    escreva("Agosto")
pare
caso 9:
    escreva("Setembro")
pare
caso 10:
    escreva("Outubro")
pare
caso 11:
    escreva("Novembro")
pare
caso 12:
    escreva("Dezembro")
pare
caso contrario:
    escreva("Número inválido")
}
}
}

// Exercício 30.
programa
{
    funcao inicio ()
    {
        caracter letra
        escreva("Digite uma letra: ")
        leia (letra)
        escolha(letra){
            caso 'a':
            caso 'e':
            caso 'i':

```

```

        caso 'o':
        caso 'u':
            escreva("Vogal")
        pare
        caso contrario:
            escreva("Não vogal")
    }
}
}

// Exercício 31.
programa
{
    funcao inicio ()
    {
        caracter tamanhoCamisa
        escreva("Digite o tamanho da camisa desejada P, M ou G: ")
        leia (tamanhoCamisa)
        escolha(tamanhoCamisa){
            caso 'p':
            caso 'P':
                escreva("P: 0.46 X 0.55")
            pare
            caso 'm':
            caso 'M':
                escreva("M: 0.51 X 0.56")
            pare
            caso 'g':
            caso 'G':
                escreva("G: 0.52 X 0.58")
            pare
            caso contrario:
                escreva("Tamanho inválido")
        }
    }
}

```

20.11 CAPÍTULO 11

```

// Exercício 32.
programa {
    funcao inicio() {
        inteiro numero, soma = 0

```

```

        para(inteiro cont = 1; cont <= 5; cont++){
            escreva("Digite um número: ")
            leia(numero)
            soma = soma + numero
        }
        escreva("Total somando: ", soma)
    }
}

// Exercício 33.
programa {
    funcao inicio() {
        inteiro numero, qtdMaior50 = 0
        para(inteiro cont = 1; cont <= 10; cont++){
            escreva("Digite um número: ")
            leia(numero)
            se(numero > 50){
                qtdMaior50++
            }
        }
        escreva("Quantidade de números maiores que 50: ", qtdMaior50)
    }
}

// Exercício 34.
programa {
    funcao inicio() {
        inteiro qtdMaior190 = 0
        real altura
        para(inteiro cont = 1; cont <= 12; cont++){
            escreva("Digite uma altura: ")
            leia(altura)
            se(altura > 1.90){
                qtdMaior190++
            }
        }
        escreva("Quantidade de atletas com altura maior que 1.90: ", qtdMaior190)
    }
}

// Exercício 35.
programa {
    funcao inicio() {

```

```

inteiro qtdAlunos = 25
real nota, maiorNota=0.0, menorNota=0.0, totalNota = 0.0
para(inteiro cont = 1; cont <= qtdAlunos; cont++){
    escreva("Digite a nota: ")
    leia(nota)
    se(cont == 1){
        maiorNota = nota
        menorNota = nota
    }senao{
        se (maiorNota < nota){
            maiorNota = nota
        }
        se (menorNota > nota){
            menorNota = nota
        }
    }
    totalNota = totalNota + nota
}
escreva("\nRelatório\n")
escreva("Maior nota: ", maiorNota, "\n")
escreva("Menor nota: ", menorNota, "\n")
escreva("Média das notas: ", totalNota/qtdAlunos, "\n")
}

// Exercício 36.
programa {
    funcao inicio() {
        inteiro qtdBolinhas
        para(inteiro cont = 5; cont >= 1; cont--){
            escreva("Qual a quantidade de bolinhas do pote? ")
            leia(qtdBolinhas)
            se (qtdBolinhas == 82){
                escreva("Parabéns, você acertou.")
                cont = 0
            }senao se(qtdBolinhas < 82){
                escreva("Você errou! Existem mais bolinhas do que
você digitou")
                escreva("\nVocê ainda tem ", cont-1, " chances.\n"
)
            }senao{
                escreva("Você errou! Existem menos bolinhas do q
ue você digitou")
                escreva("\nVocê ainda tem ", cont-1, " chances.\n"
)
            }
        }
    }
}

```

```
        }
    }
}
}
```

20.12 CAPÍTULO 12

// Exercício 37.

```
programa {
    funcao inicio() {
        inteiro numero, soma = 0, cont = 1
        enquanto(cont <=5){
            escreva("Digite um número: ")
            leia(numero)
            soma = soma + numero
            cont++
        }
        escreva("Total somando: ", soma)
    }
}
```

// Exercício 38.

```
programa {
    funcao inicio() {
        inteiro numero, totalNumero = 0, qtdNumeros = 0
        enquanto(totalNumero < 100){
            escreva("Digite um número: ")
            leia(numero)
            totalNumero = totalNumero + numero
            qtdNumeros++
        }
        escreva("Quantidade de números digitados: ", qtdNumeros)
    }
}
```

// Exercício 39.

```
programa {
    funcao inicio() {
        inteiro qtdVoltas, cont = 1
        real volta, voltaMaisLenta=0.0, voltaMaisRapido=0.0
        real totalVoltas = 0.0
        escreva("Digite a quantidade de voltas desejadas:")
        leia(qtdVoltas)
```

```

        enquanto(cont <= qtdVoltas){
            escreva("Digite a velocidade da volta: ")
            leia(volta)
            se(cont == 1){
                voltaMaisLenta = volta
                voltaMaisRapido = volta
            }senao{
                se (voltaMaisRapido > volta){
                    voltaMaisRapido = volta
                }
                se (voltaMaisLenta < volta){
                    voltaMaisLenta = volta
                }
            }
            totalVoltas = totalVoltas + volta
            cont++
        }
        escreva("\nRelatório\n")
        escreva("Volta mais rápida: ", voltaMaisRapido, "\n")
        escreva("Volta mais lenta: ", voltaMaisLenta, "\n")
        escreva("Média das voltas: ", totalVoltas/qtdVoltas, "\n")
    )
}
}

// Exercício 40.
programa {
    funcao inicio() {
        inteiro qtdBolinhas, qtdRepeticoes = 0
        inteiro qtdErrosMenores82 = 0
        inteiro qtdErrosMaiores82 = 0
        caracter desejaContinuar = 's'
        enquanto(desejaContinuar == 's'){
            qtdRepeticoes++
            escreva("Qual a quantidade de bolinhas do pote? ")
            leia(qtdBolinhas)
            se (qtdBolinhas == 82){
                escreva("Parabéns, você acertou.")
            }senao se(qtdBolinhas < 82){
                escreva("Você errou! Existem mais bolinhas do que
você digitou")
                qtdErrosMenores82++
            }senao{
                escreva("Você errou! Existem menos bolinhas do q
ue você digitou")
}

```

```

        qtdErrosMaiores82++
    }
    escreva("\nDigite s se deseja continuar ou n para sair: ")
    leia(desejaContinuar)
}
escreva("-----")
escreva("\nRelatório\n")
escreva("Quantidade de tentativas realizadas: ", qtdRepeticoes)
escreva("\n")
escreva("Quantidade de erros menores que 82: ", qtdErrosMenores82)
escreva("\n")
escreva("Quantidade de erros maiores que 82: ", qtdErrosMaiores82)

}
}

```

20.13 CAPÍTULO 13

```

// Exercício 41.
programa {
    funcao inicio() {
        inteiro numero, totalNumero = 0, qtdNumeros = 0
        enquanto(totalNumero < 100){
            faca{
                escreva("Digite um número: ")
                leia(numero)
            }enquanto(numero <= 0)
            totalNumero = totalNumero + numero
            qtdNumeros++
        }
        escreva("Quantidade de números digitados: ", qtdNumeros)
    }
}

// Exercício 42.
programa {
    funcao inicio() {
        inteiro dataPagamento
        faca{

```

```

        escreva("Informe o dia 2, 5 ou 10 para pagamento do bo
leto: ")
        leia(dataPagamento)
    }enquanto(dataPagamento != 2 e dataPagamento != 5 e dataPa
gamento != 10)
        escreva("Boleto registrado com sucesso.")
    }
}

```

20.14 CAPÍTULO 14

```

// Exercício 43.
programa {
    funcao inicio() {
        real voltas[6]
        para(inteiro cont = 0; cont < 6; cont++){
            escreva("Digite a velocidade da volta: ")
            leia(voltas[cont])
        }
        escreva("-----\n Velocidades lidas na ordem decrescent
e:")
        para(inteiro cont = 5; cont >=0; cont--){
            escreva("\n")
            escreva(voltas[cont])
        }
    }
}

// Exercício 44.
programa {
    funcao inicio() {
        real pagamento, cartaoFidelizacao[10]
        para(inteiro cont = 0; cont < 10; cont++){
            escreva("Informe o valor do pagamento: ")
            leia(cartaoFidelizacao[cont])
        }
        escreva("Hoje o seu almoço é uma cortesia da casa, Parabéns
!")
    }
}

// Exercício 45.
programa {

```

```

funcao inicio() {
    caracter senha[6], senhaCriptografada[6]
    escreva("Cadastro de senha\n")
    escreva("Sua senha só pode ter vogais.\n")
    para(inteiro cont = 0; cont < 6; cont++){
        faca{
            escreva("\nDigite o ", cont+1, "º caracter da sua sen-
ha: ")
            leia(senha[cont])
            }enquanto (senha[cont] != 'a' e senha[cont] != 'e' e sen-
ha[cont] != 'i' e senha[cont] != 'o' e senha[cont] != 'u' )
        }
        para(inteiro cont = 0; cont < 6; cont++){
            escolha(senha[cont]){
                caso 'a':
                    senhaCriptografada[cont] = 'z'
                pare
                caso 'e':
                    senhaCriptografada[cont] = '3'
                pare
                caso 'i':
                    senhaCriptografada[cont] = '1'
                pare
                caso 'o':
                    senhaCriptografada[cont] = '0'
                pare
                caso 'u':
                    senhaCriptografada[cont] = '$'
                pare
            }
        }
        escreva("\n-----\n")
        escreva("Senha digitada:")
        para(inteiro cont = 0; cont < 6; cont++){
            escreva(senha[cont])
        }
        escreva("\nSenha criptografada: ")
        para(inteiro cont = 0; cont < 6; cont++){
            escreva(senhaCriptografada[cont])
        }
    }
}

// Exercício 46.
programa {

```

```

funcao inicio() {
    cadeia nome[11]
    inteiro quantidadeGols[11], posicaoArtilheiro=0, qtdGolsA
    rtilheiro = 0
    para (inteiro cont = 0; cont < 11; cont++){
        escreva("Digite o nome do ", cont + 1, "º jogador: ")
        leia(nome[cont])
        escreva("Digite a quantidade de gols do ", cont + 1,
        "º jogador: ")
        leia(quantidadeGols[cont])
        se (cont == 0){
            qtdGolsArtilheiro = quantidadeGols[cont]
            posicaoArtilheiro = cont
        }senao{
            se(qtdGolsArtilheiro < quantidadeGols[cont]){
                qtdGolsArtilheiro = quantidadeGols[cont]
                posicaoArtilheiro = cont
            }
        }
        escreva("\n")
    }
    escreva ("\n-----\n")
    escreva("Artilheiro: ", nome[posicaoArtilheiro], " - Gols
    : ", quantidadeGols[posicaoArtilheiro])
}
}

```

20.15 CAPÍTULO 15

```

// Exercício 47.
programa {
    funcao inicio() {
        inteiro assentos[21][2], fileira, poltrona
        caracter continuar = 's'

        // Zerando as poltronas
        para(inteiro linha = 0; linha < 21 ; linha++ ){
            para(inteiro coluna = 0; coluna < 2; coluna++ ){
                assentos[linha][coluna] = 0
            }
        }

        enquanto (continuar == 's'){

```

```

        escreva("Poltronas livres para venda:")
//Exibindo as poltronas
para(inteiro linha = 0; linha < 21 ; linha++){
    escreva ("\\n")
    se(linha < 9){
        escreva ("Fileira: 0", linha + 1, " - > Po
ltronas: ")
    }senao{
        escreva ("Fileira: ", linha + 1, " - > Po
ltronas: ")
    }
}

para(inteiro coluna = 0; coluna < 2; coluna++)
{
    se(asientos[linha][coluna] == 0){
        escreva ( coluna + 1 , "      ")
    }senao{
        escreva ( "-" , "      ")
    }
}
faca{
    escreva("\nDigite a fileira desejada:")
leia(fileira)
    }enquanto(fileira < 1 ou fileira > 21)
faca{
    escreva("\nDigite a poltrona desejada da filei
ra ", fileira, " :")
        leia(poltrona)
    }enquanto(poltrona !=1 e poltrona != 2)
    se(asientos[fileira][poltrona-1] == 0){
        escreva("Poltrona vendida com sucesso")
        asientos[fileira][poltrona-1] = 1
    }senao{
        escreva("Essa poltrona já está ocupada, selecione
outra fileira e/ou poltrona.")
    }
    escreva("\nDigite s para continuar a comprar ou
n para sair: ")
        leia(continuar)
}

//Exibindo as poltronas vendidas e não vendidas
escreva("\n Final das vendas")

```

```

        para(inteiro linha = 0; linha < 21 ; linha++){
            escreva ("\\n")
            se(linha < 9){
                escreva ("Fileira: 0", linha + 1, " - > Poltronas: ")
            }
            }senao{
                escreva ("Fileira: ", linha + 1, " - > Poltronas: ")
            }

        para(inteiro coluna = 0; coluna < 2; coluna++){
            se(assentos[linha][coluna] == 0){
                escreva ( coluna + 1 , "      ")
            }senao{
                escreva ( "-" , "      ")
            }
        }
    }

// Exercício 48.
programa {
    funcao inicio() {
        cadeia estacionamento[9][2], placa
        inteiro opcao, vaga, vagaLinha, vagaColuna
        // Zerando as vagas
        para (inteiro l = 0; l < 9; l++){
            para (inteiro c = 0; c < 2; c++){
                estacionamento[l][c] = "      "
            }
        }

        faca{
            escreva("\n-----MENU DO SISTEMA-----")
            escreva("\nDigite 1 para exibir as vagas do estacionamento")
            escreva("\nDigite 2 para estacionar um carro em uma vaga")
            escreva("\nDigite 3 para liberar uma vaga ocupada")
            escreva("\nDigite 4 para sair")
            escreva("\nDigite a opção desejada: ")
            leia(opcao)
            escolha(opcao){

```

```

caso 1://exibindo as vagas livres e ocupadas
    //exibindo as vagas livres e ocupadas
    escreva("---- VAGAS ----")
    para (inteiro l = 0; l < 9; l++){
        escreva("\n")
        para (inteiro c = 0; c < 2; c++){
            se(c == 1){
                escreva("\t\t")
            }
            escreva("Vaga ",l+1,c+1,":" ,esta
cionamento[l][c])
        }
    }
pare
caso 2://estacionando um carro
    escreva("Digite a posição da vaga:")
    leia(vaga)
    escreva("Digite a placa do carro:")
    leia(placa)
    //descobrindo a linha da matriz
    vagaLinha = vaga / 10
    //descobrindo a coluna da matriz
    vagaColuna = vaga % 10

    estacionamento[vagaLinha-1][vagaColuna-1]
=  placa
pare
caso 3://Liberando uma vaga
    escreva("Digite a placa do carro:")
    leia(placa)
    para (inteiro l = 0; l < 9; l++){
        para (inteiro c = 0; c < 2; c++){
            se(placa == estacionamento[l][c])
{
            escreva("Carro encontrado na
vaga ", l+1, c+1)
            estacionamento[l][c] =
"
        }
    }
}
pare
caso 4:
    escreva("Obrigado por usar o sistema.")
pare

```

```

        caso contrario:
            escreva("\n\nOpção inválida\n")
            pare
        }
    }enquanto(opcao !=4)
}
}

```

20.16 CAPÍTULO 16

```

// Exercício 49
programa {
    funcao inteiro verificaVolga(caracter letra){
        inteiro retorno = 0 // 0 significa q não é vogal e 1 é vogal
        escolha(letra){
            caso 'a':
            caso 'e':
            caso 'i':
            caso 'o':
            caso 'u':
                retorno = 1
            pare
        }
        retorno retorno
    }
    funcao inicio() {
        caracter letra
        inteiro retorno
        escreva("Digite uma letra: ")
        leia(letra)
        retorno = verificaVolga(letra)
        se(retorno == 1){
            escreva("É vogal")
        }senao{
            escreva("Não é vogal")
        }
    }
}

// Exercício 50.
programa {
    funcao inteiro soma(inteiro num1, inteiro num2){

```

```

        retorno = num1 + num2
    }
    funcao inteiro subtracao(inteiro num1, inteiro num2){
        retorno = num1 - num2
    }
    funcao inteiro multiplicacao(inteiro num1, inteiro num2){
        retorno = num1 * num2
    }
    funcao inteiro divisao(inteiro num1, inteiro num2){
        // Como é inteiro só retornará o inteiro da divisão
        retorno = num1 / num2
    }
    funcao inicio() {
        inteiro n1, n2, retorno
        caracter operacao
        escreva("Digite o numero 1: ")
        leia(n1)
        escreva("Digite o numero 2: ")
        leia(n2)
        escreva("Digite a operação desejada (+ - * ou /): ")
        leia(operacao)
        se(operacao == '+'){
            retorno = soma(n1,n2)
            escreva("Resultado da soma: ", retorno)
        }senao se(operacao == '-'){
            retorno = subtracao(n1,n2)
            escreva("Resultado da subtração: ", retorno)
        }senao se(operacao == '*'){
            retorno = multiplicacao(n1,n2)
            escreva("Resultado da multiplicação: ", retorno)
        }senao se(operacao == '/') {
            retorno = divisao(n1,n2)
            escreva("Resultado inteiro da divisão: ", retorno)
        }
    }
}

// Exercício 51.
programa {
    funcao inteiro soma(inteiro num1, inteiro num2){
        retorno = num1 + num2
    }
    funcao inteiro subtracao(inteiro num1, inteiro num2){
        retorno = num1 - num2
    }
}

```

```

funcao inteiro multiplicacao(inteiro num1, inteiro num2){
    retorno num1 * num2
}
funcao inteiro divisao(inteiro num1, inteiro num2){
    // Como é inteiro só retornará o inteiro da divisão
    retorno num1 / num2
}
funcao inicio() {
    inteiro n1, n2, retorno, somaOperacoes = 0
    caracter operacao
    escreva("Digite o numero 1: ")
    leia(n1)
    escreva("Digite o numero 2: ")
    leia(n2)
    retorno = soma(n1,n2)
    somaOperacoes = retorno
    retorno = subtracao(n1,n2)
    somaOperacoes = somaOperacoes + retorno
    retorno = multiplicacao(n1,n2)
    somaOperacoes = somaOperacoes + retorno
    retorno = divisao(n1,n2)
    somaOperacoes = somaOperacoes + retorno
    escreva("A soma do retorno das operações é igual a : ", somaOperacoes)
}
}

```

20.17 CAPÍTULO 17

```

// Exercício 52.
programa {
    funcao real prazo3dias(real valor){
        retorno valor + 25.00
    }
    funcao real prazo5dias(real valor){
        retorno valor + 20.00
    }
    funcao real prazo7dias(real valor){
        retorno valor + 15.00
    }
    funcao real prazo10dias(real valor){
        retorno valor + 10.00
    }
}

```

```

funcao inicio() {
    inteiro diasEntrega
    real valorTotal, retorno
    escreva("Informe o valor total do pedido: ")
    leia(valorTotal)
    faca{
        escreva("Informe o prazo desejado pelo cliente (3, 5,
7 ou 10 dias úteis): ")
        leia(diasEntrega)
        }enquanto(diasEntrega != 3 e diasEntrega != 5 e diasEntre
ga != 7 e diasEntrega != 10)
        se(diasEntrega == 3){
            retorno = prazo3dias(valorTotal)
            escreva("Valor total com o prazo de entrega de ",
diasEntrega, " dias úteis: ",retorno )
        }senao se(diasEntrega == 5){
            retorno = prazo5dias(valorTotal)
            escreva("Valor total com o prazo de entrega de ",
diasEntrega, " dias úteis: ",retorno )
        }senao se(diasEntrega == 7){
            retorno = prazo7dias(valorTotal)
            escreva("Valor total com o prazo de entrega de ",
diasEntrega, " dias úteis: ",retorno )
        }senao se(diasEntrega == 10){
            retorno = prazo10dias(valorTotal)
            escreva("Valor total com o prazo de entrega de ",
diasEntrega, " dias úteis: ",retorno )
        }senao{
            escreva("Data inválida")
        }
    }
}

// Exercício 53.
programa {
    funcao simulaGratificacao(real salario, inteiro mes){
        se (mes > 0 e mes < 6){
            salario = salario + (salario * 0.3)
        }senao se (mes >=6 e mes <= 11){
            salario = salario + (salario * 0.4)
        }senao se (mes == 12){
            salario = salario + (salario * 0.6)
        }
        escreva("Simulação do salário + gratificação: ", salario)
    }
}

```

```

funcao inicio() {
    real salario
    inteiro mes
    escreva("Digite o salário básico do funcionário: ")
    leia(salario)
    faca{
        escreva("Digite o mês que deseja simular o salário (1
até 12): ")
        leia(mes)
    }enquanto(mes < 1 ou mes > 12)
    simulaGratificacao(salario, mes)
}
}

```

20.18 CAPÍTULO 18

```

// Exercício 54.
programa {
    funcao prazo3dias(real &valor){
        valor = valor + 25.00
    }
    funcao prazo5dias(real &valor){
        valor = valor + 20.00
    }
    funcao prazo7dias(real &valor){
        valor = valor + 15.00
    }
    funcao prazo10dias(real &valor){
        valor = valor + 10.00
    }
    funcao inicio() {
        inteiro diasEntrega
        real valorTotal
        escreva("Informe o valor total do pedido: ")
        leia(valorTotal)
        faca{
            escreva("Informe o prazo desejado pelo cliente (3, 5,
7 ou 10 dias úteis): ")
            leia(diasEntrega)
        }enquanto(diasEntrega != 3 e diasEntrega != 5 e diasEntre
ga != 7 e diasEntrega != 10)
            se(diasEntrega == 3){
                prazo3dias(valorTotal)

```

```

        escreva("Valor total com o prazo de entrega de ",
diasEntrega," dias úteis: ",valorTotal )
    }senao se(diasEntrega == 5){
        prazo5dias(valorTotal)
        escreva("Valor total com o prazo de entrega de ",
diasEntrega," dias úteis: ",valorTotal )
    }senao se(diasEntrega == 7){
        prazo7dias(valorTotal)
        escreva("Valor total com o prazo de entrega de ",
diasEntrega," dias úteis: ",valorTotal )
    }senao se(diasEntrega == 10){
        prazo10dias(valorTotal)
        escreva("Valor total com o prazo de entrega de ",
diasEntrega," dias úteis: ",valorTotal )
    }senao{
        escreva("Data inválida")
    }
}
}

// Exercício 55.
programa {
    funcao reajustarGasolina(real &gasolina, real reajuste){
        gasolina = gasolina + reajuste
    }
    funcao reajustarEtanol(real &etanol, real reajuste){
        etanol = etanol + reajuste
    }
    funcao inicio() {
        real gasolinaAtual, etanolAtual, valorReajuste, valorReajuste27Gasolina
        caracter combustivelReajuste
        escreva("Digite o valor atual da gasolina:")
        leia(gasolinaAtual)
        escreva("Digite o valor atual do etanol: ")
        leia(etanolAtual)
        escreva("Informe o valor do reajuste: ")
        leia(valorReajuste)
        faca{
            escreva("Combustível do reajuste (G-gasolina ou E-eta
nol: ")
            leia(combustivelReajuste)
            }enquanto(combustivelReajuste != 'E' e combustivelReajust
e != 'G') {
                se (combustivelReajuste == 'G'){

```

```
    reajustarGasolina(gasolinaAtual, valorReajuste)
}senao se (combustivelReajuste == 'E'){
    reajustarEtanol(etanolAtual, valorReajuste)
    valorReajuste27Gasolina = valorReajuste * 0.27
    reajustarGasolina(gasolinaAtual, valorReajuste2
7Gasolina)
}
    escreva("----- Valor atualizado ----\n")
escreva("\nGasolina: ", gasolinaAtual)
escreva("\nEtanol: ", etanolAtual)
}
}
```

CAPÍTULO 21

RESOLUÇÃO DO PROJETO DE FIXAÇÃO

CÓDIGO-FONTE DO PROJETO

Os códigos do nosso projeto da loja XPTO Bikes podem ser consultados no link:

<https://logicacomportugol.com.br/codigo/projeto.zip>

21.1 CAPÍTULO 3

```
programa {
    funcao inicio() {
        escreva("Sejam bem-vindos à loja XPTO Bikes! Em breve teremos um sistema de autoatendimento")
    }
}
```

21.2 CAPÍTULO 4

```
programa {
    funcao inicio() {
        cadeia enderecoLoja
        inteiro numeroLoja
```

```

        enderecoLoja = "Avenida XPTO"
        numeroLoja = 123
        escreva("Sejam bem-vindos à loja XPTO Bikes! Em breve ter
emos um sistema de autoatendimento\n")
        escreva("Endereço: ",enderecoLoja, " - número: ", numero
Loja)
    }
}

```

21.3 CAPÍTULO 6

```

programa {
    funcao inicio() {
        cadeia nome
        escreva("Informe seu nome:")
        leia(nome)
        escreva("Prezado(a), ", nome, ". Seja muito bem-vindo(a) à
nossa loja")
        escreva("\n")
        escreva("Oferecemos em nossa loja venda e manutenção de b
icicletas.\nPara venda de bicicletas procure o colaborador Junior
, e para manutenção procure o colaborador Neto.\n\nObrigado e esp
eramos que tenha uma ótima experiência em nossa loja.")
    }
}

```

21.4 CAPÍTULO 8

```

programa {
    funcao inicio() {
        cadeia nome
        inteiro opcao
        escreva("Informe seu nome:")
        leia(nome)
        escreva("Prezado(a), ", nome, ". Seja muito bem-vindo(a) à
nossa loja")
        escreva("\n")
        escreva("Oferecemos em nossa loja venda e manutenção de b
icicletas.\nPara venda de bicicletas procure o colaborador Junior
, e para manutenção procure o colaborador Neto.\n\nObrigado e esp
eramos que tenha uma ótima experiência em nossa loja.")
        escreva("\n\n--- MENU ---\n")
    }
}

```

```

escreva("Código 1 - Ver ofertas de bicicletas usadas \n")
escreva("Código 2 - Ver ofertas de bicicletas novas \n")
escreva("Código 3 - Ver ofertas de acessórios \n")
escreva("Código 4 - Ver novos serviços \n")
escreva("Digite o código desejado: ")
leia(opcao)
se(opcao == 1){
    escreva("\nBicicleta usada na cor azul, aro 26, com 1
8 marchas e com o valor promocional de R$ 400,00\n\n")
}senao se(opcao == 2){
    escreva("\nBicicleta nova na cor amarela, aro 26, com
18 marchas e na promoção pelo preço de R$ 999,99\n\n")
}senao se(opcao == 3){
    escreva("\nAcessório em oferta - Capacete de proteção
por R$59,99\n\n")
}senao se(opcao == 4){
    escreva("\nNovos serviços oferecidos:\n| Lavagem comp
leta da sua bicicleta por R$ 12,99 \n| Manutenção dos freios por
R$ 10,99 \n| Troca de pneus por R$ 55,99")
}
}
}
}

```

21.5 CAPÍTULO 9

```

programa {
    funcao inicio() {
        cadeia nome
        inteiro opcao
        escreva("Informe seu nome:")
        leia(nome)
        escreva("Prezado(a),", nome, ". Seja muito bem-vindo(a) à
nossa loja")
        escreva("\n")
        escreva("Oferecemos em nossa loja venda e manutenção de b
icicletas.\nPara venda de bicicletas procure o colaborador Junior
, e para manutenção procure o colaborador Neto.\n\nObrigado e esp
eramos que tenha uma ótima experiência em nossa loja.")
        escreva("\n\n--- MENU ---\n")
        escreva("Código 1 - Ver ofertas de bicicletas usadas \n")
        escreva("Código 2 - Ver ofertas de bicicletas novas \n")
        escreva("Código 3 - Ver ofertas de acessórios \n")
        escreva("Código 4 - Ver novos serviços \n")
    }
}
}
}

```

```

escreva("Código 5 -Promoção I 10% de desconto \n")
escreva("6 - Promoção II 20% de desconto \n")
escreva("Digite o código desejado: ")
leia(opcao)
se(opcao == 1{
    escreva("\nBicicleta usada na cor azul, aro 26, com 18 marchas e com o valor promocional de R$ 400,00\n\n")
}senao se(opcao == 2){
    escreva("\nBicicleta nova na cor amarela, aro 26, com 18 marchas e na promoção pelo preço de R$ 999,99\n\n")
}senao se(opcao == 3{
    escreva("\nAcessório em oferta - Capacete de proteção por R$59,99\n\n")
}senao se(opcao == 4{
    escreva("\nNovos serviços oferecidos:\n| Lavagem completa da sua bicicleta por R$ 12,99\n| Manutenção dos freios por R$ 10,99\n| Troca de pneus por R$ 55,99\n\n")
}senao se(opcao == 5{
    escreva("\nLave sua bicicleta (R$ 12,99) e realiz e manutenção no freio(R$ 10,99) com desconto de 10% no total do pagamento\n\n")
}senao se(opcao == 6{
    escreva("\nTroque um pneu da bicicleta (R$ 55,99) e realize a manutenção nos freios (R$ 10,99) com 20% de desconto no total de pagamento\n\n")
}
}
}
}

```

21.6 CAPÍTULO 10

```

programa {
    funcao inicio() {
        cadeia nome
        inteiro opcao
        real valorTotalSemDesconto, percentualDesconto = 0.0, valorDoDesconto, valorTotalComDesconto
        caracter lavouBicicleta, trocouPneu, manutencaoFreio
        escreva("Informe seu nome:")
        leia(nome)
        se(nome != "xpto restrito"){ // VERIFICA TIPO DE ACESSO
            escreva("Prezado(a)," , nome, ". Seja muito bem-vindo(a) à nossa loja")
        }
    }
}

```

```

        escreva("\n")
        escreva("Oferecemos em nossa loja venda e manutenção de bicicletas.\nPara venda de bicicletas procure o colaborador Junior, e para manutenção procure o colaborador Neto.\n\nObrigado e esperamos que tenha uma ótima experiência em nossa loja.")
        escreva("\n\n--- MENU ---\n")
        escreva("Código 1 - Ver ofertas de bicicletas usadas \n")
        escreva("Código 2 - Ver ofertas de bicicletas novas \n")
        escreva("Código 3 - Ver ofertas de acessórios \n")
        escreva("Código 4 - Ver novos serviços \n")
        escreva("Código 5 - Promoção I 10% de desconto \n")
        escreva("Código 6 - Promoção II 20% de desconto \n")
        escreva("Digite o código desejado: ")
        leia(opcao)
        se(opcao == 1){
            escreva("\nBicicleta usada na cor azul, aro 26, com 18 marchas e com o valor promocional de R$ 400,00\n\n")
        }
        }senao se(opcao == 2){
            escreva("\nBicicleta nova na cor amarela, aro 26, com 18 marchas e na promoção pelo preço de R$ 999,99\n\n")
        }
        }senao se(opcao == 3){
            escreva("\nAcessório em oferta - Capacete de proteção por R$59,99\n\n")
        }
        }senao se(opcao == 4){
            escreva("\nNovos serviços oferecidos:\n| Lavagem completa da sua bicicleta por R$ 12,99 | Manutenção dos freios por R$ 10,99 |\n| Troca de pneus por R$ 55,99|\n")
        }
        }senao se(opcao == 5){
            escreva("\nLave sua bicicleta (R$ 12,99) e realize manutenção no freio(R$ 10,99) com desconto de 10% no total do pagamento\n")
        }
        }senao se(opcao == 6){
            escreva("\nTroque um pneu da bicicleta (R$ 55,99) e realize a manutenção nos freios (R$ 10,99) com 20% de desconto no total de pagamento\n")
        }
    }senao{// ACESSO RESTRITO PARA ABERTURA DE OS
        escreva("\n\n ----- ACESSO RESTRITO -----")
    }
}

```

```

        escreva("\nEntre com valor a ser pago antes do desco
nto:")
        leia(valorTotalSemDesconto)
        escreva("\nA) O cliente lavou a bicicleta (digite S
ou N)?")
        leia(lavouBicicleta)
        escreva("\nB) O cliente trocou pneu da bicicleta (d
igite S ou N)?")
        leia(trocouPneu)
        escreva("\nC) O cliente realizou manutenção freio (d
igite S ou N)?")
        leia(manutencaoFreio)
        se ((lavouBicicleta == 'S') e (manutencaoFreio == 'S
'))
    ){
        percentualDesconto = 10.0
    }
    se ((trocouPneu == 'S') e (manutencaoFreio == 'S')){
        percentualDesconto = 20.0
    }
    escreva("\n\n-----\n\n")
    escreva("Valor total sem desconto: ", valorTotalSe
mDesconto, "\n")
    escreva("Desconto: ", percentualDesconto, "% \n")
    valorDoDesconto = (percentualDesconto / 100.00 ) *
valorTotalSemDesconto
    escreva("Valor do desconto: ", valorDoDesconto , "\n")
    escreva("\nVALOR FINAL: ", valorTotalSemDesconto -
valorDoDesconto , "\n")
}
}
}

```

21.7 CAPÍTULO 11

```

programa {
    funcao inicio() {
        cadeia nome
        inteiro opcao
        real valorTotalSemDesconto, percentualDesconto = 0.0, val
orDoDesconto, valorTotalComDesconto
        caracter lavouBicicleta, trocouPneu, manutencaoFreio
    }
}

```

```

para (inteiro cont = 0; cont < 18; cont++){
    escreva("Informe seu nome:")
    leia(nome)
    se(nome != "xptorestrito"){ // VERIFICA TIPO DE ACESSO
        escreva("Prezado(a), ", nome, ". Seja muito bem-vindo(a) à nossa loja.")
        escreva("\n")
        escreva("Oferecemos em nossa loja venda e manutenção de bicicletas.\nPara venda de bicicletas procure o colaborador Junior, e para manutenção procure o colaborador Neto.\n\nObrigado e esperamos que tenha uma ótima experiência em nossa loja.")
    }
    escreva("\n\n--- MENU ---\n")
    escreva("Código 1 - Ver ofertas de bicicletas usadas \n")
    escreva("Código 2 - Ver ofertas de bicicletas novas \n")
    escreva("Código 3 - Ver ofertas de acessórios \n")
    escreva("Código 4 - Ver novos serviços \n")
    escreva("Código 5 - Promoção I 10% de desconto \n")
    escreva("Código 6 - Promoção II 20% de desconto \n")
    escreva("Digite o código desejado: ")
    leia(opcao)
    se(opcao == 1){
        escreva("\nBicicleta usada na cor azul, aro 26, com 18 marchas e com o valor promocional de R$ 400,00\n")
    }senao se(opcao == 2){
        escreva("\nBicicleta nova na cor amarela, aro 26, com 18 marchas e na promoção pelo preço de R$ 999,99\n\n")
    }senao se(opcao == 3){
        escreva("\nAcessório em oferta - Capete de proteção por R$59,99\n\n")
    }senao se(opcao == 4){
        escreva("\nNovos serviços oferecidos:\n| Lavagem completa da sua bicicleta por R$ 12,99\n| Manutenção dos freios por R$ 10,99\n| Troca de pneus por R$ 55,99\n\n")
    }senao se(opcao == 5){
        escreva("\nLave sua bicicleta (R$ 12,99) e realize manutenção no freio(R$ 10,99) com desconto de 10% n

```


21.8 CAPÍTULO 12

```
programa {
    funcao inicio() {
        cadeia nome
        inteiro opcao = 0 , qtdClientes = 0
        real valorTotalSemDesconto, percentualDesconto = 0.0, valorDoDesconto, valorTotalComDesconto
        caracter lavouBicicleta, trocouPneu, manutencaoFreio
        enquanto(opcao != 10){
            escreva("Informe seu nome:")
            leia(nome)
            se(nome != "xptorestrito"){ // VERIFICA TIPO DE ACESSO
                escreva("Prezado(a), ", nome, ". Seja muito bem-vindo(a) à nossa loja")
                escreva("\n")
                escreva("Oferecemos em nossa loja venda e manutenção de bicicletas.\nPara venda de bicicletas procure o colaborador Junior, e para manutenção procure o colaborador Neto.\n\nObrigado e esperamos que tenha uma ótima experiência em nossa loja.")
            }
            escreva("\n\n--- MENU ---\n")
            escreva("Código 1 - Ver ofertas de bicicletas usadas \n")
            escreva("Código 2 - Ver ofertas de bicicletas novas \n")
            escreva("Código 3 - Ver ofertas de acessórios \n")
            escreva("Código 4 - Ver novos serviços \n")
            escreva("Código 5 - Promoção I 10% de desconto \n")
            escreva("Código 6 - Promoção II 20% de desconto \n")
            escreva("Código 10 - SAIR \n")
            escreva("Digite o código desejado: ")
            leia(opcao)
            qtdClientes++
            se(opcao == 1){
                escreva("\nBicicleta usada na cor azul, aro 26, com 18 marchas e com o valor promocional de R$ 400,00\n")
            }senao se(opcao == 2){
                escreva("\nBicicleta nova na cor amar
```

```

ela, aro 26, com 18 marchas e na promoção pelo preço de R$ 999,99
\n\n")
}senao se(opcao == 3){
    escreva("\nAcessório em oferta - Capacete de proteção por R$59,99\n\n")
}senao se(opcao == 4){
    escreva("\nNovos serviços oferecidos:
\n| Lavagem completa da sua bicicleta por R$ 12,99 \n| Manutenção dos freios por R$ 10,99 \n| Troca de pneus por R$ 55,99\n\n")
}senao se(opcao == 5){
    escreva("\nLave sua bicicleta (R$ 12,99) e realize manutenção no freio(R$ 10,99) com desconto de 10% no total do pagamento\n\n")
}senao se(opcao == 6){
    escreva("\nTroque um pneu da bicicleta (R$ 55,99) e realize a manutenção nos freios (R$ 10,99) com 20% de desconto no total de pagamento\n\n")
}
}senao{// ACESSO RESTRITO PARA ABERTURA DE OS
    escreva("\n\n ----- ACESSO RESTRITO -----")
    escreva("\nEnter com valor a ser pago antes do desconto:")
    leia(valorTotalSemDesconto)
    escreva("\nA) O cliente lavou a bicicleta (digite S ou N)?")
    leia(lavouBicicleta)
    escreva("\nB) O cliente trocou pneu da bicicleta (digite S ou N)?")
    leia(trocouPneu)
    escreva("\nC) O cliente realizou manutenção freio (digite S ou N)?")
    leia(manutencaoFreio)
    se ((lavouBicicleta == 'S') e (manutencaoFreio == 'S')){
        percentualDesconto = 10.0
    }
    se ((trocouPneu == 'S') e (manutencaoFreio == 'S')){
        percentualDesconto = 20.0
    }
    escreva("\n\n-----\n\n")
    escreva("Valor total sem desconto: ", valorTotalSemDesconto, "\n")
    escreva("Desconto: ", percentualDesconto, "%\n")
}

```

```

        valorDoDesconto = (percentualDesconto / 100.0)
    ) * valorTotalSemDesconto
        escreva("Valor do desconto: ", valorDoDesconto, "\n")
        escreva("\nVALOR FINAL: ", valorTotalSemDesconto - valorDoDesconto, "\n")
    }
    escreva("\nTotal de clientes que utilizaram o sistema hoje: ", qtdClientes)
}
}

```

21.9 CAPÍTULO 14

```

programa {
    funcao inicio() {
        const inteiro tamanhoVetorCarrinho = 3
        cadeia nome="", carrinhoNomeProduto[tamanhoVetorCarrinho]
        inteiro opcao=0, carrinhoCodigoProduto[tamanhoVetorCarrinho], subOpcao=0, qtdProdutoNoCarrinho=0
        inteiro addCarrinho, posicaoAtual=-1, cont, qtdVendaCliente = 0
        real carrinhoValorProduto[tamanhoVetorCarrinho], valorTotalCarrinho = 0.0, desconto = 0.0
        real totalVendaDiaria = 0.0
        caracter tipoPagamento
        escreva("Bem-vindo ao autoatendimento da bicileataria XPTO Bikes!")
        faca{
            escreva("\nDigite seu nome:")
            leia(nome)
            posicaoAtual=-1
            valorTotalCarrinho=0.0
        faca{
            se (nome == "sair"){
                escreva("Obrigado por usar o sistema")
            }senao se (nome != "xptorestrito"){
                //Acesso cliente
                escreva("\n***** MENU *****\n")
                escreva("Opção 1 - Ver promoções.\n")
                escreva("Opção 2 - Solicitar serviço de manutenção.\n")
            }
        }
    }
}

```

```

        escreva("Opção 3 - Listar carrinho
de compra.\n")
        escreva("Opção 4 - Finalizar carrin
ho de compra.\n")
        escreva("Opção 0 - Sair")
        escreva("\n*****\n")
    )
    escreva("Digite sua opção desejada:
")
    leia(opcao)
    escolha(opcao){
        caso 0:
            escreva("Até logo")
            pare
        caso 1:
            faca{
                escreva("\n***** PROMOÇ
ÃO *****")
                escreva("\n[Cód 101] Bici
cleta nova na cor amarela, aro 26, com 18 marchas e na promoção p
elo preço de R$ 999,99 ")
                escreva("\n[Cód 102] Bici
cleta usada na cor azul, aro 26, com 18 marchas e com o valor pro
mocional de R$ 400,00 ")
                escreva("\n[Cód 103] Capa
cete de proteção por R$59,99 ")
                escreva("\n[Cód 104] Frei
o a disco por R$ 89,99 ")
                escreva("\nOpção 8 - Adic
ionar ao carrinho de compras")
                escreva("\nOpção 0 - Volt
ar")
                escreva("\n*****\n")
                escreva("Digite a opção 8
ou 0: ")
                leia(sub0pcao)
            }enquanto(sub0pcao != 8 e sub0pca
o != 0)
            se(sub0pcao == 8){
                se (posicaoAtual == tamanhoVe
torCarrinho-1){
                    escreva("Carrinho cheio !"
)
                }senao{

```

```

        escreva("Digite o código
do produto que deseja adicionar ao carrinho: ")
        leia(addCarrinho)
        posicaoAtual++
        carrinhoCodigoProduto[posicaoAtual] = addCarrinho
        se(addCarrinho == 101){
            carrinhoValorProduto[posicaoAtual] = 999.99
            carrinhoNomeProduto[posicaoAtual] = "Bicicleta nova na cor amarela"
        }senao  se(addCarrinho ==
102){
            carrinhoValorProduto[posicaoAtual] = 400.00
            carrinhoNomeProduto[posicaoAtual] = "Bicicleta usada na cor azul"
        }senao  se(addCarrinho ==
103){
            carrinhoValorProduto[posicaoAtual] = 59.99
            carrinhoNomeProduto[posicaoAtual] = "Capacete de proteção"
        }senao  se(addCarrinho ==
104){
            carrinhoValorProduto[posicaoAtual] = 89.99
            carrinhoNomeProduto[posicaoAtual] = "Freio a disco"
        }senao{
            escreva("Código invál
ido")
            posicaoAtual--
        }
    }
    pare
    caso 2:
        faca{
            escreva("\n***** SERVIÇOS *
*****")
            escreva("\n[Cód 201] Troca de
pneu - R$ 55,99 ")
            escreva("\n[Cód 202] Lavagem
")

```

```

completa -R$ 12,99 ")
escreva("\n[Cód 203] Freio -
R$ 10,99 ")
escreva(" ao carrinho de compras")
escreva("\nOpção 8 - Adiciona
r ao carrinho de compras")
escreva("\nOpção 0 - Voltar")
escreva("\n*****")
escreva("Digite a opção 8 ou
0: ")
leia(subOpcão)
}enquanto(subOpcão != 8 e subOpcão
o != 0)
se(subOpcão == 8){
se (posicaoAtual == tamanhoVe
torCarrinho-1){
escreva("Carrinho cheio !"
)
}senao{
escreva("Digite o código
do produto que deseja adicionar ao carrinho: ")
leia(addCarrinho)
posicaoAtual++
carrinhoCodigoProduto[pos
icaoAtual] = addCarrinho
posicaoAtual] = 55.99
carrinhoValorProduto[
osicaoAtual] = "Troca de pneu"
carrinhoNomeProduto[p
202){
posicaoAtual] = 12.99
carrinhoValorProduto[
osicaoAtual] = "Lavagem completa"
carrinhoNomeProduto[p
203){
posicaoAtual] = 10.99
carrinhoValorProduto[
osicaoAtual] = "Freio"
carrinhoNomeProduto[p
ido")
}senao{
escreva("Código invál

```

```

                posicaoAtual--
            }
        }
    }
pare
// LISTAR CARRINHO DE COMPRAS
caso 3:
    escreva("\n-----\n")
    escreva("Carrinho de compras\n")
    se(posicaoAtual == -1){
        escreva("vazio")
    }senao {
        para(cont = 0; cont <= posica
oAtual; cont++){
            escreva("\n", cont+1, "º
item do carrinho: ", carrinhoNomeProduto[cont], "\n")
            escreva("Código do produ
to: ",carrinhoCodigoProduto[cont], " | Valor: ", carrinhoValorProd
uto[cont])
            valorTotalCarrinho = valo
rTotalCarrinho + carrinhoValorProduto[cont]
        }
        escreva("\n\nTotal do carrinh
o: ", valorTotalCarrinho)
    }
    escreva("\n-----\n")
pare
caso 4:
    escreva("\n-----\n")
    escreva("Finalizar Carrinho de co
mpras\n")
    se(posicaoAtual == -1){
        escreva("vazio")
    }senao {
        valorTotalCarrinho=0.0
        para(cont = 0; cont <= posica
oAtual; cont++){
            valorTotalCarrinho = valo
rTotalCarrinho + carrinhoValorProduto[cont]
        }
        escreva("\n\nTotal do carrinh
o: ", valorTotalCarrinho)
    }
}

```

```

o: ", valorTotalCarrinho)
escreva("\nDigite o tipo de p
agamento D(dinheiro) ou C(cartão)")
leia(tipoPagamento)
se(tipoPagamento == 'D'){
    desconto = valorTotalCarr
    inho * 0.10
    valorTotalCarrinho = valo
rTotalCarrinho - desconto
}
qtdVendaCliente++
totalVendaDiaria = totalVendaDiaria
+ valorTotalCarrinho
escreva("\n\nDesconto: ", descont
o)
escreva("\nValor final: ", valor
TotalCarrinho)
posicaoAtual=-1
valorTotalCarrinho=0.0
}
escreva("\n-----\n-----\n")
pare
caso contrario:
    escreva("Opção inválida")
pare
}

}senao{
//Acesso restrito
escreva("\n***** ACESSO RESTRITO *****\n")
escreva("Digite o nome do cliente:")
leia(nome)
escreva("\nDigite o valor total: ")
leia(valorTotalCarrinho)
escreva("\nDigite o tipo de pagamento D(d
inheiro) ou C(cartão): ")
leia(tipoPagamento)
se(tipoPagamento == 'D'){
    desconto = valorTotalCarrinho * 0.10
    valorTotalCarrinho = valorTotalCarr
    inho - desconto
}
qtdVendaCliente++
totalVendaDiaria = totalVendaDiaria + val

```

```

    orTotalCarrinho
        escreva("\n\nDesconto: ", desconto)
        escreva("\nValor final: ", valorTotalCarr
    inho)

    }
    }enquanto(opcao != 0)
}enquanto(nome != "sair")
escreva("\n\n #####\n")
escreva("Relatório do dia")
escreva("\nQuantidade de clientes:", qtdVendaCliente)
escreva("\nTotal vendido: ", totalVendaDiaria)
}
}

```

21.10 CAPÍTULO 16

```

programa {
    funcao inicio() {
        const inteiro tamanhoVetorCarrinho = 3
        cadeia nome="", carrinhoNomeProduto[tamanhoVetorCarrinho]
        inteiro opcao=0, carrinhoCodigoProduto[tamanhoVetorCarrin
        ho], subOpcao=0, qtdProdutoNoCarrinho=0
        inteiro addCarrinho, posicaoAtual=-1, cont, qtdVendaClien
        te = 0
        real carrinhoValorProduto[tamanhoVetorCarrinho], valorTot
        alCarrinho = 0.0, desconto = 0.0
        real totalVendaDiaria = 0.0
        caracter tipoPagamento
        escreva("Bem-vindo ao autoatendimento da bicicletaria XPT
        O Bikes.")
        faca{
            escreva("\nDigite seu nome:")
            leia(nome)
            posicaoAtual=-1
            valorTotalCarrinho=0.0
            faca{
                se (nome == "sair"){
                    escreva("Obrigado por usar o sistema")
                }senao se (nome != "xptorestrito"){
                    //Acesso cliente
                    exibirMenu()
                    escreva("Digite sua opção desejada: ")
                }
            }
        }
    }
}

```

```

        leia(opcao)
        escolha(opcao){
            caso 0:
                escreva("Até logo")
            pare
            caso 1:
                faca{
                    exibirMenuPromocao()
                    escreva("Digite a opção 8 ou
                            0: ")
                }
            o != 0)
                se(subOpcao == 8){
                    se (posicaoAtual == tamanhoVe-
                        torCarrinho-1){
                        escreva("Carrinho cheio !"
)
                    }
                }senao{
                    escreva("Digite o código
                            do produto que deseja adicionar ao carrinho: ")
                    leia(addCarrinho)
                    posicaoAtual++
                    carrinhoCodigoProduto[pos-
                        icaoAtual] = addCarrinho
                    se(addCarrinho == 101){
                        carrinhoValorProduto[
                            posicaoAtual] = 999.99
                        carrinhoNomeProduto[p-
                            osicaoAtual] = "Bicicleta nova na cor amarela"
                    }senao  se(addCarrinho ==
                            102){
                        carrinhoValorProduto[
                            posicaoAtual] = 400.00
                        carrinhoNomeProduto[p-
                            osicaoAtual] = "Bicicleta usada na cor azul"
                    }senao  se(addCarrinho ==
                            103){
                        carrinhoValorProduto[
                            posicaoAtual] = 59.99
                        carrinhoNomeProduto[p-
                            osicaoAtual] = "Capacete de proteção"
                    }senao  se(addCarrinho ==
                            104){
                        carrinhoValorProduto[

```

```

posicaoAtual] = 89.99
carrinhoNomeProduto[p
osicaoAtual] = "Freio a disco"
}senao{
    escreva("Código invál
ido")
    posicaoAtual--
}
}
pare
caso 2:
faca{
    exibirMenuServicos()
    escreva("Digite a opção 8 ou
0: ")
    leia(subOpcão)
}enquanto(subOpcão != 8 e subOpc
o != 0)
se(subOpcão == 8){
    se (posicaoAtual == tamanhoVe
torCarrinho-1){
        escreva("Carrinho cheio !"
)
    }senao{
        escreva("Digite o código
do produto que deseja adicionar ao carrinho: ")
        leia(addCarrinho)
        posicaoAtual++
        carrinhoCodigoProduto[pos
icaoAtual] = addCarrinho
    se(addCarrinho == 201){
        carrinhoValorProduto[
posicaoAtual] = 55.99
        carrinhoNomeProduto[p
osicaoAtual] = "Troca de pneu"
    }senao  se(addCarrinho ==
202){
        carrinhoValorProduto[
posicaoAtual] = 12.99
        carrinhoNomeProduto[p
osicaoAtual] = "Lavagem completa"
    }senao  se(addCarrinho ==
203){
        carrinhoValorProduto[
```

```

posicaoAtual] = 10.99
osicaoAtual] = "Freio"
}senao{
    escreva("Código invál
ido")
    posicaoAtual--
}
}
pare
// LISTAR CARRINHO DE COMPRAS
caso 3:
    escreva("\n-----\n")
    escreva("Carrinho de compras\n")
    se(posicaoAtual == -1){
        escreva("vazio")
    }senao {
        para(cont = 0; cont <= posica
oAtual; cont++){
            escreva("\n", cont+1, "º
item do carrinho: ", carrinhoNomeProduto[cont], "\n")
            escreva("Código do produ
to: ",carrinhoCodigoProduto[cont], " | Valor: ", carrinhoValorProd
uto[cont])
            valorTotalCarrinho = valo
rTotalCarrinho + carrinhoValorProduto[cont]
        }
        escreva("\n\nTotal do carrinh
o: ", valorTotalCarrinho)
    }
    escreva("\n-----\n")
}
pare
caso 4:
    escreva("\n-----\n")
    escreva("Finalizar Carrinho de co
mpras\n")
    se(posicaoAtual == -1){
        escreva("vazio")
    }senao {
        valorTotalCarrinho=0.0
        para(cont = 0; cont <= posica

```

```

oAtual; cont++){
                valorTotalCarrinho = valo
rTotalCarrinho + carrinhoValorProduto[cont]
}
escreva("\n\nTotal do carrinh
o: ", valorTotalCarrinho)
escreva("\nDigite o tipo de p
agamento D(dinheiro) ou C(cartão") )
leia(tipoPagamento)
se(tipoPagamento == 'D'){
    desconto = valorTotalCarr
inho * 0.10
valorTotalCarrinho = valo
rTotalCarrinho - desconto
}
qtdVendaCliente++
totalVendaDiaria = totalVendaDiaria
+ valorTotalCarrinho
escreva("\n\nDesconto: ", descont
o)
escreva("\nValor final: ", valorT
otalCarrinho)
posicaoAtual=-1
valorTotalCarrinho=0.0
}
escreva("\n-----\n-----\n")
pare
caso contrario:
    escreva("Opção inválida")
pare
}
}senao{
//Acesso restrito
escreva("\n***** ACESSO RESTRITO *****\n"
)
escreva("Digite o nome do cliente:")
leia(nome)
escreva("\nDigite o valor total: ")
leia(valorTotalCarrinho)
escreva("\nDigite o tipo de pagamento D(d
inheiro) ou C(cartão): ")
leia(tipoPagamento)
se(tipoPagamento == 'D'){
    desconto = valorTotalCarrinho * 0.10
}

```

```

        valorTotalCarrinho = valorTotalCarrin
ho - desconto
    }
    qtdVendaCliente++
    totalVendaDiaria = totalVendaDiaria + val
orTotalCarrinho
    escreva("\n\nDesconto: ", desconto)
    escreva("\nValor final: ", valorTotalCarr
inho)
}
}enquanto(opcao != 0)
}enquanto(nome != "sair")
    escreva("\n\n #####\n")
    escreva("Relatório do dia")
    escreva("\nQuantidade de clientes: ", qtdVendaCliente
)
    escreva("\nTotal vendido: ", totalVendaDiaria)
}
funcao exibirMenu(){
    escreva("\n***** MENU *****\n")
    escreva("Opção 1 - Ver promoções.\n")
    escreva("Opção 2 - Solicitar serviço de manutenção.\n"
)
    escreva("Opção 3 - Listar carrinho de compra.\n")
    escreva("Opção 4 - Finalizar carrinho de compra.\n")
    escreva("Opção 0 - Sair")
    escreva("\n*****\n")
}
funcao exibirMenuPromocao(){
    escreva("\n***** PROMOÇÃO *****")
    escreva("\n[Cód 101] Bicicleta nova na cor amarela, a
ro 26, com 18 marchas e na promoção pelo preço de R$ 999,99 ")
    escreva("\n[Cód 102] Bicicleta usada na cor azul, aro
26, com 18 marchas e com o valor promocional de R$ 400,00 ")
    escreva("\n[Cód 103] Capacete de proteção por R$59,99
")
    escreva("\n[Cód 104] Freio a disco por R$ 89,99 ")
    escreva("\nOpção 8 - Adicionar ao carrinho de compras'
)
    escreva("\nOpção 0 - Voltar")
    escreva("\n*****\n")
}
funcao exibirMenuServicos(){
    escreva("\n***** SERVIÇOS *****")
    escreva("\n[Cód 201] Troca de pneu - R$ 55,99 ")

```

```

        escreva("\n[Cód 202] Lavagem completa -R$ 12,99 ")
        escreva("\n[Cód 203] Freio - R$ 10,99 ")
        escreva("\nOpção 8 - Adicionar ao carrinho de compras")
    )
    escreva("\nOpção 0 - Voltar")
    escreva("\n*****\n")
}
}

```

21.11 CAPÍTULO 17

```

programa {
    funcao inicio() {
        const inteiro tamanhoVetorCarrinho = 3
        cadeia nome="", carrinhoNomeProduto[tamanhoVetorCarrinho]
        inteiro opcao=0, carrinhoCodigoProduto[tamanhoVetorCarrinho], subOpcao=0, qtdProdutoNoCarrinho=0
        inteiro addCarrinho, posicaoAtual=-1, cont, qtdVendaCliente = 0
        real carrinhoValorProduto[tamanhoVetorCarrinho], valorTotalCarrinho = 0.0, desconto = 0.0
        real totalVendaDiaria = 0.0
        caracter tipoPagamento
        escreva("Bem-vindo ao autoatendimento da bicicletaria XPTO Bikes.")
        faca{
            escreva("\nDigite seu nome:")
            leia(nome)
            posicaoAtual=-1
            valorTotalCarrinho=0.0
            desconto=0.0
            faca{
                se (nome == "sair"){
                    escreva("Obrigado por usar o sistema")
                }senao se (nome != "xptorestrito"){
                    //Acesso cliente
                    exibirMenu()
                    escreva("Digite sua opção desejada: ")
                    leia(opcao)
                    escolha(opcao){
                        caso 0:
                            escreva("Até logo")
                        pare
                }
            }
        }
    }
}

```

```

caso 1:
faca{
    exibirMenuPromocao()
    escreva("Digite a opção 8 ou
0: ")
    leia(subOpcão)
}enquanto(subOpcão != 8 e subOpcão
o != 0)
se(subOpcão == 8){
    se (posicaoAtual == tamanhoVe-
torCarrinho-1){
        escreva("Carrinho cheio !"
)
    }senao{
        escreva("Digite o código
do produto que deseja adicionar ao carrinho: ")
        leia(addCarrinho)
        posicaoAtual++
        carrinhoCodigoProduto[posi-
caoAtual] = addCarrinho
        se(addCarrinho == 101){
            carrinhoValorProduto[po-
sicaoAtual] = 999.99
            carrinhoNomeProduto[po-
sicaoAtual] = "Bicicleta nova na cor amarela"
            }senao  se(addCarrinho ==
102){
            carrinhoValorProduto[po-
sicaoAtual] = 400.00
            carrinhoNomeProduto[po-
sicaoAtual] = "Bicicleta usada na cor azul"
            }senao  se(addCarrinho ==
103){
            carrinhoValorProduto[po-
sicaoAtual] = 59.99
            carrinhoNomeProduto[po-
sicaoAtual] = "Capacete de proteção"
            }senao  se(addCarrinho ==
104){
            carrinhoValorProduto[po-
sicaoAtual] = 89.99
            carrinhoNomeProduto[po-
sicaoAtual] = "Freio a disco"
            }senao{
                escreva("Código invál

```

```

ido")
    posicaoAtual--
}
}
}
pare
caso 2:
faca{
    exibirMenuServicos()
    escreva("Digite a opção 8 ou
0: ")
    leia(subOpcao)
}enquanto(subOpcao != 8 e subOpca
o != 0)
se(subOpcao == 8){
    se (posicaoAtual == tamanhoVe
torCarrinho-1){
        escreva("Carrinho cheio !"
)
    }senao{
        escreva("Digite o código
do produto que deseja adicionar ao carrinho: ")
        leia(addCarrinho)
        posicaoAtual++
        carrinhoCodigoProduto[pos
icaoAtual] = addCarrinho
        se(addCarrinho == 201){
            carrinhoValorProduto[
posicaoAtual] = 55.99
            posicaoAtual] = "Troca de pneu"
        }senao  se(addCarrinho ==
202){
            carrinhoValorProduto[
posicaoAtual] = 12.99
            posicaoAtual] = "Lavagem completa"
        }senao  se(addCarrinho ==
203){
            carrinhoValorProduto[
posicaoAtual] = 10.99
            posicaoAtual] = "Freio"
        }senao{

```

```

        escreva("Código invál
ido")
        posicaoAtual--
    }
}
pare
// LISTAR CARRINHO DE COMPRAS
caso 3:
    escreva("\n-----
-----\n")
    escreva("Carrinho de compras\n")
    se(posicaoAtual == -1){
        escreva("vazio")
    }senao {
        para(cont = 0; cont <= posica
oAtual; cont++){
            escreva("\n", cont+1, "o
item do carrinho: ", carrinhoNomeProduto[cont], "\n")
            escreva("Código do produ
to: ",carrinhoCodigoProduto[cont], " | Valor: ", carrinhoValorProd
uto[cont])
            valorTotalCarrinho = valo
rTotalCarrinho + carrinhoValorProduto[cont]
        }
        escreva("\n\nTotal do carrinh
o: ", valorTotalCarrinho)
    }
    escreva("\n-----
-----\n")
    pare
    caso 4:
        escreva("\n-----
-----\n")
        escreva("Finalizar Carrinho de co
mpras\n")
        se(posicaoAtual == -1){
            escreva("vazio")
        }senao {
            valorTotalCarrinho=0.0
            para(cont = 0; cont <= posica
oAtual; cont++){
                valorTotalCarrinho = valo
rTotalCarrinho + carrinhoValorProduto[cont]
            }
}

```

```

        escreva("\n\nTotal do carrinho
o: ", valorTotalCarrinho)
        escreva("\nDigite o tipo de p
agamento D(dinheiro) ou C(cartão)")
        leia(tipoPagamento)
        se(tipoPagamento == 'D'){
            desconto = calculaDescont
o(valorTotalCarrinho)
            valorTotalCarrinho = valo
rTotalCarrinho - desconto
        }
        qtdVendaCliente++
        totalVendaDiaria = totalVendaDiaria
        + valorTotalCarrinho
        escreva("\n\nDesconto: ", descont
o)
        escreva("\nValor final: ", valorT
otalCarrinho)
        posicaoAtual=-1
        valorTotalCarrinho=0.0
    }
    escreva("\n-----\n-----\n")
    pare
    caso contrario:
        escreva("Opção inválida")
    pare
}
}senao{
    //Acesso restrito
    escreva("\n***** ACESSO RESTRITO *****\n")
    escreva("Digite o nome do cliente:")
    leia(nome)
    escreva("\nDigite o valor total: ")
    leia(valorTotalCarrinho)
    escreva("\nDigite o tipo de pagamento D(dinheiro)
ou C(cartão): ")
    leia(tipoPagamento)
    se(tipoPagamento == 'D'){
        desconto = calculaDesconto(valorTotal
Carrinho)
        valorTotalCarrinho = valorTotalCarrin
ho - desconto
    }
}

```

```

        qtdVendaCliente++
        totalVendaDiaria = totalVendaDiaria + valorTotal
Carrinho
        escreva("\n\nDesconto: ", desconto)
        escreva("\nValor final: ", valorTotalCarrinho)
    }
}enquanto(opcao != 0)
}enquanto(nome != "sair")
    escreva("\n\n #####\n")
    escreva("Relatório do dia")
    escreva("\nQuantidade de clientes: ", qtdVendaCliente)
    escreva("\nTotal vendido: ", totalVendaDiaria)
}
funcao exibirMenu(){
    escreva("\n***** MENU *****\n")
    escreva("Opção 1 - Ver promoções.\n")
    escreva("Opção 2 - Solicitar serviço de manutenção.\n")
}
    escreva("Opção 3 - Listar carrinho de compra.\n")
    escreva("Opção 4 - Finalizar carrinho de compra.\n")
    escreva("Opção 0 - Sair")
    escreva("\n*****\n")
}
funcao exibirMenuPromocao(){
    escreva("\n***** PROMOÇÃO *****")
    escreva("\n[Cód 101] Bicicleta nova na cor amarela, aro 2
6, com 18 marchas e na promoção pelo preço de R$ 999,99 ")
    escreva("\n[Cód 102] Bicicleta usada na cor azul, aro 26,
com 18 marchas e com o valor promocional de R$ 400,00 ")
    escreva("\n[Cód 103] Capacete de proteção por R$59,99 ")
    escreva("\n[Cód 104] Freio a disco por R$ 89,99 ")
    escreva("\nOpção 8 - Adicionar ao carrinho de compras")
    escreva("\nOpção 0 - Voltar")
    escreva("\n*****\n")
}
funcao exibirMenuServicos(){
    escreva("\n***** SERVIÇOS *****")
    escreva("\n[Cód 201] Troca de pneu - R$ 55,99 ")
    escreva("\n[Cód 202] Lavagem completa -R$ 12,99 ")
    escreva("\n[Cód 203] Freio - R$ 10,99 ")
    escreva("\nOpção 8 - Adicionar ao carrinho de compras")
    escreva("\nOpção 0 - Voltar")
    escreva("\n*****\n")
}
funcao real calculaDesconto(real valorTotal){

```

```
    real desconto
    desconto = valorTotal * 0.10
    retorno desconto
}
}
```

CAPÍTULO 22

REFERÊNCIAS

BACKES, André. *Linguagem C completa e descomplicada.* 1^a ed. São Paulo: Editora Elsevier, 2013.

DEITEL, Harvey M.; DEITEL, Paul J. *Como programar em C.* Rio de Janeiro: LTC, 1999.

FORBELLONE, L.V, André; EBERSPACHER, F, Henri. *Lógica de programação. A estrutura de algoritmos e estruturas de dados.* 3^a edição. São Paulo: Editora Pearson, 2005.

LITE, LABORATÓRIO DE INOVAÇÃO TECNOLÓGICA NA EDUCAÇÃO UNIVALI. 2021

LOPES, Anita; GARCIA, Guto. *Introdução à programação: 500 algoritmos resolvidos.* São Paulo: Elsevier, 2002.

PORUTUGOL STUDIO. Ajuda da ferramenta IDE Portugol Studio. 2021.

SOFFNER, Renato Kraide . *Algoritmos e programação em linguagem C.* São Paulo: Saraiva Educação SA., 2017.

VILARIM, Gilvan. *Algoritmos: programação para iniciantes.* Rio de Janeiro: Ciência Moderna, 2017.