Informação e Comunicação EaD

Lógica de Programação

Educação Profissional

fundação bradesco

Direitos desta edição

Fundação Bradesco

Homepage

https://fundacao.bradesco/

Autoria

Departamento de Educação Profissional e Educação de Jovens e Adultos

Revisão Técnica Pedagógica

Departamento de Educação Profissional e Educação de Jovens e Adultos
- Ana Cristina Venancio da Silva
- Luis Ricardo de Oliveira

Coordenação

Departamento de Educação Profissional e Educação de Jovens e Adultos
- Rosa Maria Pires Bueno
- Allyson Luiz de Cayres Lino
- Evelin Vanessa Correia dos Santos Marques

Design instrucional e revisão textual

FGV

Revisão técnica e pedagógica do *Design* Instrucional

Departamento de Educação Profissional e Educação de Jovens e Adultos
- Ana Cristina Venancio da Silva
- Luis Ricardo de Oliveira
- Adriana Brito

Publicação

2017

SUMÁRIO

CARTA AO ALUNO

Caro aluno,

Este conteúdo foi elaborado pensando em seu processo de aprendizagem. Nele você encontrará conceitos importantes de lógica de programação, úteis para que você aprenda a programar computadores.

Ao longo do estudo, você será convidado a fazer atividades e reflexões que contribuirão para seu desenvolvimento profissional. Essas atividades são importantes para que você tenha uma visão prática da utilização dos equipamentos e um conhecimento teórico sobre as tecnologias envolvidas.

Fique atento a todo o conteúdo apresentado, para que você tenha um aprendizado significativo e condições de aplicar os conhecimentos em suas atividades.

Para alcançar os objetivos propostos, lembramos que sua dedicação e seu comprometimento são fundamentais. Leia o material com atenção, responda aos exercícios propostos, e aproveite as dicas e os recursos educacionais disponibilizados sobre os assuntos relacionados à sua área de atuação.

Ao término de seus estudos, você realizará uma avaliação para verificar sua compreensão a respeito dos assuntos abordados.

Desejamos a você um bom aprendizado!

ESTRUTURA DE CONTEÚDO

Este conteúdo está estruturado em dois capítulos:

Capítulo 1: Lógica, Algoritmo e Pseudocódigo

- Tópico 1: Lógica
- Tópico 2: Algoritmo
- Tópico 3: Pseudocódigo

Capítulo 2: Controle de Fluxo e Estrutura de Repetição

- Tópico 1: Controle de Fluxo
- Tópico 2: Estrutura de Repetição

CAPÍTULO 1: LÓGICA, ALGORITMO E PSEUDOCÓDIGO

Tópico 1: Lógica

Neste tópico, vamos apresentar os temas Lógica e Raciocínio Lógico, relacionando-os aos conceitos de sequência e de instruções em lógica.

Conteúdos:

- Definição de Lógica
- Diferenciação entre Lógica e Lógica de Programação
- Instruções em Lógica
- O que são programas e para que eles servem?

Ao finalizar este tópico, você será capaz de:

- Compreender o conceito e a importância de Lógica de Programação.
- Aprender a construir e testar algoritmos.

Lógica Nossa de Cada Dia

É provável que você nem se dê conta, mas a lógica está presente na vida de todas as pessoas, todos os dias, praticamente todo o tempo.

Quando usamos o raciocínio para tomar uma decisão, buscamos sempre a saída mais lógica, ou seja, a opção que nos pareça ser a mais adequada para resolver um problema ou atingir um objetivo, não é mesmo?

Quando vamos a algum lugar pela primeira vez, pensamos sobre qual seria o melhor caminho a fazer e também sobre qual seria a melhor forma de transporte (avião, ônibus, carro, metrô, trem, navio, helicóptero, bicicleta, a pé etc.) para chegarmos ao destino desejado.

Frente a tantas possibilidades, procuramos a melhor opção a depender da nossa necessidade, afinal de contas podemos escolher o meio de transporte mais rápido (que poderá ser o mais caro), a opção de locomoção mais barata (que poderá ser a mais demorada) ou aquela que nos possibilite curtir mais o passeio, e assim por diante.

Ao usarmos o raciocínio para buscar a melhor solução para um problema ou uma situação, exercitamos a lógica.

Interessante, não?

A Lógica é, antes de tudo, uma área da Filosofia que se dedica a refletir sobre as diferentes formas de raciocinar.

Assim sendo, a Lógica se propõe a pensar sobre modos rigorosos de desenvolver o raciocínio em busca da melhor maneira de pensar ou fazer algo. Em outras palavras, a **Lógica define o encadeamento de ações mais coerentes para chegar a um objetivo**.

Vamos pensar mais um pouco sobre tudo isso?

Que tal pensarmos em um exemplo prático?

O computador é uma inovação tecnológica que surgiu no século XX para ajudar o homem a calcular mais rapidamente. Como ele foi feito para ajudar a raciocinar em menor tempo e com maior eficiência, podemos afirmar que sua invenção é fruto da Lógica (assim como a maioria das inovações tecnológicas) e que é a lógica que orienta o seu funcionamento (uma vez que a Matemática se baseia em princípios lógicos).

Em outras palavras, o computador é uma invenção lógica que funciona a partir de uma sequência de instruções ou comandos, e que, para ser eficiente, deve ser programado logicamente, isto é, codificado por meio de elementos e atributos de programação.

A Lógica defende o uso da razão para entender, calcular, processar, pesquisar, questionar e decidir sobre as coisas do mundo. Por isso, praticamente tudo que os homens fazem é orientado por ela, de pesquisas científicas a decisões do dia a dia. Curioso isso, não é?

Lógica no Contexto da Programação

Como você pôde notar, a Lógica é, sem dúvida, algo importante e presente no dia a dia de pessoas, comunidades, empresas, escolas, universidades etc.

No entanto, a Lógica passou a ter um papel ainda mais importante a partir do século XX. Você consegue imaginar por quê?

A Lógica é o princípio mais importante no desenvolvimento de programas de computador.

Todo aplicativo ou sistema de computador deve ser projetado e desenvolvido de forma racional para que possamos chegar a sequências lógicas de instruções que sejam coerentes e, principalmente, eficazes.

Você sabe o que é uma sequência lógica? De acordo com Moraes (2000):

Uma sequência lógica é um grupo de passos estabelecidos para chegar a um resultado, a um objetivo ou à solução de um problema.

Perceba que nesse tipo de encadeamento, tanto a ordem das ações como a natureza delas são importantes.

Na prática, as sequências lógicas trazem:

- As instruções que devem ser feitas (**o quê**).
- A ordem em que as instruções devem ser realizadas (**quando**).

> Em Informática, instrução é uma ordem ou um comando que indica ao computador uma ação elementar a ser executada (MORAES, 2000).

Então, vamos entender melhor o papel das instruções.

As instruções são regras criadas para serem realizadas em situações específicas.

Por exemplo, quando compramos aparelhos eletrônicos, recebemos manuais de instrução. Esses manuais descrevem o que deve ser feito para que os aparelhos funcionem corretamente.

Como você já pôde perceber, o computador é uma máquina que realiza, exclusivamente, o que ela for programada para fazer.

Dessa forma, um programa nada mais é do que uma sequência lógica de instruções organizadas para manipular informações inseridas pelos usuários. Tudo isso é feito para atingir determinado fim.

Anote essa ideia!

Importância do Aprendizado de Lógica de Programação

O objetivo principal deste conteúdo é propiciar a você o aprendizado a respeito do que é e de como se pode raciocinar de forma lógica.

Para isso, serão criadas situações que lhe possibilitarão desenvolver a capacidade de criar sequências lógicas de instruções capazes de resolver problemas e atingir objetivos práticos.

Além de aprender a melhorar seu raciocínio lógico, você aprenderá também a registrar suas sequências lógicas de uma forma padronizada, que permita a outras pessoas entenderem suas propostas de resolução de problemas. Dessa forma, você aprenderá a pensar de forma estruturada e a registrar/documentar suas ideias.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre este assunto. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Você explicou a um estagiário, na empresa de TI em que trabalha, que computadores são máquinas que realizam funções diversas conforme uma programação.

Explicou, também, que um programa é:

- o conjunto de dados criados por um programador e recebidos pelo usuário.
- uma sequência lógica de instruções organizadas para manipular informações inseridas pelos usuários.
- uma cadeia organizada de informações que permitem aos usuários realizarem comandos pré-estabelecidos.
- o plano de ações coordenadas estabelecidas por um programador para serem repetidas por usuários em geral.

Questão 2

Estudando com um colega para a prova do curso de TI, vocês reviram que a Lógica pode ser entendida como a área da Filosofia que estuda o processo racional. Nossas afirmativas, decisões e atitudes, muitas vezes, são decorrentes de relações que estabelecemos, mentalmente, entre fatos e ideias, e que julgamos ter ou não coerência com nosso entorno. Então, vocês resolveram verificar a coerência de algumas sentenças ligadas a questões cotidianas.

Marque **V** para afirmações verdadeiramente coerentes e **F** para aquelas incoerentes.

Afirmações	V	F
Escolhas, como as de roupas para sair, não demandam lógica.	○	○
Para buscar a solução de um crime, é necessário analisar logicamente os fatos.	○	○
Para ratear a conta de um restaurante entre cinco amigos, é preciso somar o consumo total e multiplicá-lo por cinco.	○	○
Se um primeiro objeto é igual ao segundo, e esse segundo é igual ao terceiro, então, o primeiro objeto é igual ao terceiro.	○	○

Encerramento do Tópico

Neste tópico, você aprendeu sobre Lógica e Raciocínio Lógico, relacionando-os aos conceitos de sequência lógica e de instruções em lógica.

Agora que você já compreende, na prática, a importância de Lógica no contexto da programação de computadores, é hora de seguir em seus estudos.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Vamos prosseguir em nossos estudos?

Siga para o próximo tópico.

Tópico 2: Algoritmo

Neste tópico, apresentaremos conceitos, regras e ferramentas que facilitam a estruturação do raciocínio lógico na criação de algoritmos.

Conteúdos:

- Funções e aplicações do algoritmo
- Regras para criação de algoritmo
- Fluxograma (diagrama de blocos)
- Pseudocódigo (português estruturado).

Ao finalizar este tópico, você será capaz de:

- Identificar a diferença entre os tipos de variáveis.
- Criar um algoritmo utilizando fluxograma.
- Desenvolver um algoritmo utilizando pseudocódigo.

Algoritmos e suas Aplicações em Informática

Você já se perguntou por que é necessário aprender uma forma padronizada de registro das sequências lógicas?

Para entender melhor a necessidade de usar padrões na construção das sequências lógicas, vamos ouvir um *podcast*.

Podcast

Acesse o ambiente *on-line* para ouvir o *podcast* sobre **Uso de padrões na construção de sequências lógicas**.

Agora que você já começou a entender a função dos algoritmos, é oportuno dizer que eles não são a solução de um problema. Vamos esclarecer!

O algoritmo não é a solução do problema porque ele é, na realidade, a descrição detalhada das etapas que devem ser percorridas para se chegar a uma solução. Em outras palavras, é preciso diferenciar a solução (programa) da proposta de solução (algoritmo).

Veja a definição de algoritmo, segundo o dicionário Houaiss.

Algoritmo
Conjunto de regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema por meio de um número finito de etapas.

Desse modo, o algoritmo tem começo, meio, fim e um objetivo a ser alcançado. Em outras palavras, o algoritmo deve ser visto como um projeto do programa, e seu desenvolvimento tem por foco completar uma missão de maneira inteligente, lógica e eficaz.

Exemplo

Vejamos um exemplo para esclarecer ainda mais o que é e qual é a principal função dos algoritmos.

> Você precisa fazer um bolo de chocolate pela primeira vez (problema) e alguém lhe dá uma receita (algoritmo).
>
> A receita ainda não é seu bolo, e sim uma descrição de tudo que precisa ser feito para que o bolo fique pronto.
>
> Ao final de todos os passos descritos na receita, se o resultado final for um bolo de chocolate, é sinal de que a receita (algoritmo) foi eficaz.
>
> Se algo der errado, e você não obtiver um bolo de chocolate no final do processo, temos duas opções:
>
> - A receita (algoritmo) não descreveu corretamente o que precisava ser feito.
> - O cozinheiro (computador) "não seguiu" as instruções corretamente (hipótese certamente incorreta!).
>
> Partindo da premissa de que o computador sempre executa as ações corretamente, quando um programa não funciona é porque houve algum equívoco na etapa de planejamento (estruturação do algoritmo).

Algoritmos e suas aplicações em informática

Agora, vamos observar as etapas necessárias para o desenvolvimento de um programa (*software*).

Por meio do esquema a seguir, podemos perceber a diferença entre as etapas de planejamento do programa (estruturação do algoritmo) e de programação (desenvolvimento do código).

1. Estudo do problema

Análise do problema e de seu contexto, para identificar a situação, refletir sobre suas características e apresentar as possíveis soluções.

2. Estruturação do algoritmo

Escolha da melhor solução possível para resolver o problema e estruturação da proposta em forma de "projeto de programa" (algoritmo).

3. Desenvolvimento do código (programação)

Escrita do algoritmo a partir das regras e dos padrões de uma linguagem de programação específica, tal como C#, Java, C, C++, VB, PHP.

É nesta etapa que ocorre a codificação do algoritmo.

4. Implantação da solução

Implantação do programa em situação de vida real, para verificar se o problema foi resolvido, e realização de eventuais ajustes (manutenção).

Agora que você já entendeu as etapas de desenvolvimento de um *software*, vamos pensar no que é preciso para começar.

Para resolver um problema fazendo uso de um computador, precisamos descrever o problema de forma clara e precisa. E como fazemos isso?

Primeiro, temos de montar uma sequência de passos que permita que o problema seja solucionado de maneira automática e repetitiva.

Já sabemos, mas não custa nada lembrar: essa sequência de passos é chamada de algoritmo!

No contexto de desenvolvimento de programas, existe mais de uma forma de representar algoritmos. Vamos aprender três formas diferentes de construir algoritmos, que normalmente são utilizadas.

- Descrição Narrativa
- Diagrama de Blocos ou Fluxograma
- Português Estruturado ou Pseudocódigo

Vamos conhecer melhor essas três formas de construir algoritmos no próximo capítulo.

Instruções em Lógica

Desenvolver técnicas de lógica de programação é uma habilidade necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas em geral.

Já vimos que a lógica de programação está baseada em um raciocínio estabelecido para resolver problemas usando computadores. Desse modo, o raciocínio precisa ser descrito por meio de instruções conhecidas como sequências lógicas.

Você lembra o que são sequências lógicas?

Sequências lógicas são as etapas que devem ser realizadas para que o objetivo seja atingido.

Em outras palavras, são os passos executados até se atingir a solução de um problema.

As sequências lógicas nos permitem utilizar a lógica para ordenar e corrigir pensamentos ou ações voltados para a solução de problemas. Lembra que essas ações são descritas como instruções? Vamos relembrar o que são as instruções?

Instruções são informações, ordens ou comandos que indicam a um computador o que ele deve fazer.

Uma ordem isolada não permite a realização de um processo completo. Para isso, é necessário um conjunto de instruções colocadas em uma sequência lógica.

Agora, precisamos ver como as instruções funcionam na prática! A seguir, vamos acompanhar dois exemplos do cotidiano.

Exemplo 1

Vamos ver o primeiro exemplo de como as instruções funcionam?

Se quisermos fazer uma omelete com batatas fritas, precisaremos colocar em prática uma série de ações.

1. Lavar e secar as batatas e os ovos.
2. Descascar as batatas.
3. Cortar as batatas em forma de palito.
4. Fritar as batatas.
5. Reservar as batatas fritas em papel toalha.
6. Quebrar os ovos.
7. Bater os ovos.
8. Fritar a omelete.
9. Unir as batatas fritas com a omelete em um prato.
10. Esperar esfriar.

Como vimos, para chegarmos ao objetivo desejado, as instruções precisam ser executadas em uma ordem adequada. Neste exemplo, se esquecermos de bater os ovos antes de fritá-los, teremos uma omelete?

Exemplo 2

Agora, veja como as instruções funcionam no exemplo 2!

Se quisermos ir ao banco sacar dinheiro em um caixa eletrônico, precisaremos seguir alguns passos, certo?

Observe, a seguir, a sequência com dez passos, na ordem que está apresentada, e reflita se seria possível realizar essa ação usando o seguinte encadeamento de ações para sacar o dinheiro. Caso não concorde, na próxima tela, você terá a oportunidade de numerar as etapas na sequência lógica correta.

1. Sair do banco.
2. Aguardar a vez do atendimento.
3. Entrar na fila do caixa.
4. Digitar a senha.
5. Entrar no banco.
6. Informar o valor a ser retirado.
7. Procurar a fila do caixa.
8. Pegar o dinheiro, o cartão eletrônico e o comprovante de saque.
9. Inserir o cartão eletrônico no caixa.
10. Conferir o valor em dinheiro.

Enumere a ordem da sequência.

( ) Sair do banco.
( ) Aguardar a vez do atendimento.
( ) Entrar na fila do caixa.
( ) Digitar a senha.
( ) Entrar no banco.
( ) Informar o valor a ser retirado.
( ) Procurar a fila do caixa.
( ) Pegar o dinheiro, o cartão eletrônico e o comprovante de saque.
( ) Inserir o cartão eletrônico no caixa.
( ) Conferir o valor em dinheiro.

Comentário

Seria possível sacar dinheiro seguindo a primeira sequência apresentada? Com certeza, não.

Embora as instruções estejam corretas, a sequência está errada. Veja que a primeira ação descrita é *sair do banco*. Como se pode sacar dinheiro em um banco estando fora dele?

Quando nós, seres humanos, recebemos uma instrução dada em uma ordem que não nos parece lógica, nós questionamos ou, simplesmente, desconsideramos a instrução.

O computador, no entanto, não tem essa habilidade e, por isso, as instruções que forem dadas a esse tipo de máquina precisam ser claras e estar na ordem correta. Só assim chegaremos ao resultado final previsto.

Agora, vejamos as mesmas ações na sequência correta:

1. Entrar no banco.
2. Procurar a fila do caixa eletrônico.
3. Entrar na fila do caixa eletrônico.
4. Aguardar a vez do atendimento.
5. Inserir o cartão eletrônico no caixa.
6. Informar o valor a ser retirado.
7. Digitar a senha.
8. Pegar o dinheiro, o cartão eletrônico e o comprovante de saque.
9. Conferir o valor em dinheiro.
10. Sair do banco.

Como já foi dito, toda sequência de instruções bem definidas para se chegar a determinado objetivo recebe o nome de algoritmo. Assim sendo, os dez passos que você acabou de ler compõem um **algoritmo** para sacar dinheiro em um caixa eletrônico.

Programas, Aplicativos ou *Softwares*

Vamos pensar um pouco sobre o que costumamos chamar de programas, aplicativos ou *softwares*. Você sabe o que são os programas?

Na realidade, programas são algoritmos que foram codificados em uma determinada linguagem de programação.

As linguagens de programação vêm mudando bastante com o tempo. No entanto, a forma de pensar em soluções computadorizadas e de fazer **projetos de programas** quase não sofreu alterações ao longo dos anos.

Você consegue perceber agora a importância de aprender a pensar de forma lógica e a elaborar, interpretar e corrigir algoritmos?

Saiba mais!

Existem várias linguagens sendo utilizadas no mercado atual, como C, C#, C++, Java, JavaScript, PHP, Phyton, Objective-C, Ruby, VB, entre outras.

Além disso, novas linguagens surgem de tempos em tempos, fruto do constante desenvolvimento tecnológico, sempre em busca de soluções mais inteligentes e eficientes em programação.

Dessa forma, toda pessoa que pretende atuar na área de programação precisa estar preparada para aprender as linguagens mais modernas que forem surgindo.

Nessa área, a atualização é uma constante.

Fontes:

Disponível em: http://www.dicasdeprogramacao.com.br. Acesso em: 4 ago. 2016.
Disponível em: http://www.impacta.com.br. Acesso em: 4 ago. 2016.
Disponível em: http://www.tecmundo.com.br. Acesso em: 4 ago. 2016.

Sabemos o quanto o computador é importante atualmente em nossas vidas, certo? No entanto, o computador só é capaz de facilitar nossas atividades e resolver nossos problemas se for programado.

Nesse sentido, existem programas específicos, denominados APLICATIVOS, que buscam resolver um problema específico a partir de dados informados pelos usuários.

Cada aplicativo (programa específico) tem sua função! Por exemplo, existem programas para:

Editar textos

Fazer tabelas e gráficos

Encontrar restaurantes em determinado local

Trocar mensagens entre celulares

Quem utiliza computadores sabe que os programas têm limites e podem apresentar erros de vez em quando. Isso ocorre porque, diferentemente dos seres humanos, os programas só resolvem situações previamente pensadas.

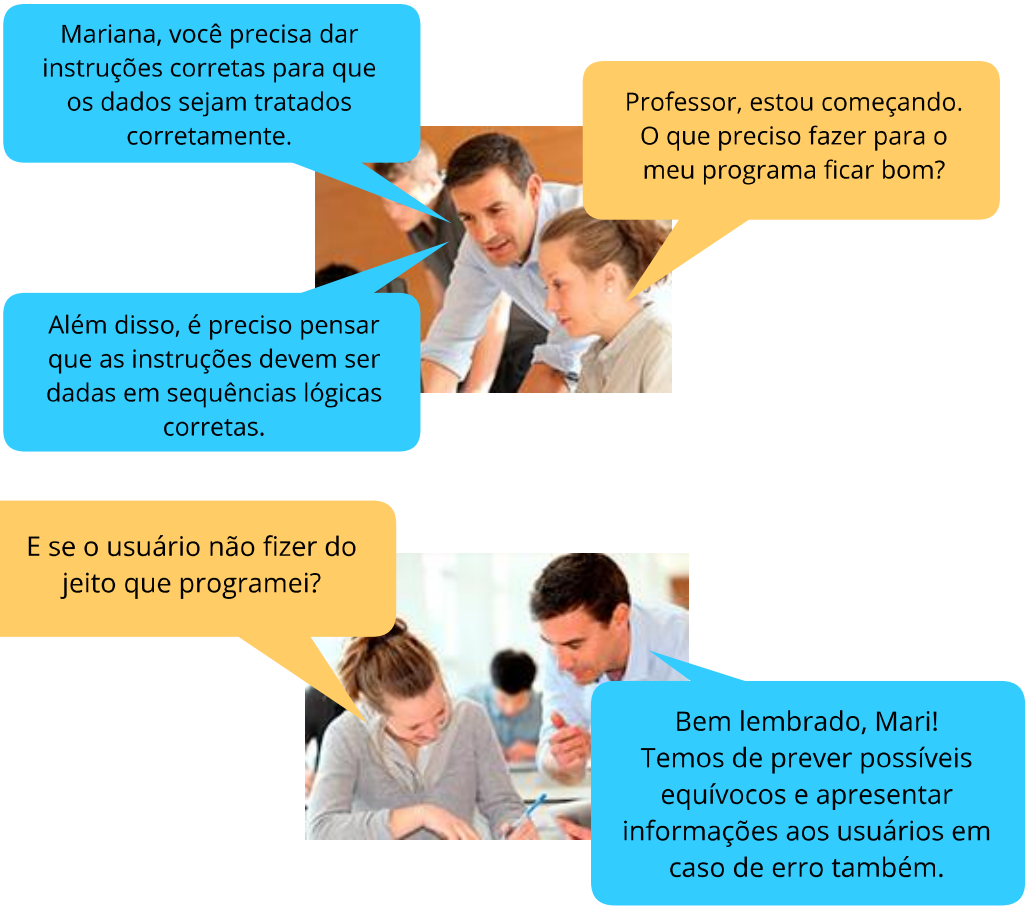
> Nesse sentido, programar é prever situações e tomar decisões sobre como alcançar os resultados esperados.

Não podemos esquecer que sempre existe mais de uma forma de resolver um problema.

E como podemos prever as situações que um programa precisará resolver?

Vamos adiante para descobrir!

A seguir, vejamos o caso de uma aluna que está começando a programar e pede orientação ao professor.



Hoje em dia, o bom profissional de informática é valorizado por várias competências.

No entanto, a capacidade de raciocinar de forma lógica é fundamental para ter reconhecimento no mercado de trabalho.

> Na prática, isso significa que a Lógica é o pilar de sustentação do profissional da área da Informática.

Os *softwares* considerados funcionais e estáveis são aqueles cujos programadores se dedicaram muito em busca de uma sequência lógica perfeita.

Cada algoritmo produzido deve ser analisado e repensado, a fim de identificar o que pode ser feito para obter a melhor solução possível. Nesse sentido, a lógica aplicada deve ser clara e concisa, o que poderá gerar um menor tempo de processamento, mas não significa, necessariamente, uma melhor *performance* da máquina ou vice-versa.

De qualquer forma, tudo visa a resultados corretos, entregues com rapidez e eficiência. Esse é o objetivo de qualquer programa!

> Na área de Informática, o sucesso de um programador requer aperfeiçoamento da lógica, raciocínio cada vez melhor, crítica sobre o que se faz e análise da situação como um todo. Com isso, é possível conseguir ótimos resultados.

Descrição Narrativa

Agora que você já sabe o que são algoritmos, está na hora de aprender a construí-los. Está preparado?

Em primeiro lugar, existe mais de uma forma de registrar algoritmos. Para entender isso, vamos lembrar as instruções utilizadas para sacar dinheiro em um caixa eletrônico.

1. Entrar no banco.
2. Procurar a fila do caixa eletrônico.
3. Entrar na fila do caixa eletrônico.
4. Aguardar a vez do atendimento.
5. Inserir o cartão eletrônico no caixa eletrônico.
6. Informar o valor a ser retirado.
7. Digitar a senha.
8. Pegar o dinheiro, o cartão eletrônico e o comprovante de saque.
9. Conferir o valor em dinheiro.
10. Sair do banco.

Essa sequência de frases curtas, com comandos claros e encadeados em ordem lógica, é a forma mais simples de criar algoritmos.

Esse jeito de elaborar algoritmos se chama **descrição narrativa** e é muito usado em documentos como manuais de equipamentos, receitas culinárias, bulas de remédio, descrições do tipo *faça você mesmo* etc. Já falamos sobre isso, lembra?

A técnica da descrição narrativa pode ser considerada a mais fácil e simples de ser utilizada, pois é a mais próxima da linguagem que usamos no dia a dia para nos comunicarmos.

Regras para o Desenvolvimento de Algoritmos

Para descrever sequências de instruções de maneira simples e objetiva é necessário observar as seguintes orientações.

1. **Seja direto**

   Escreva frases curtas e simples, com apenas um verbo.

2. **Seja detalhista**

   Descreva, com detalhes, todos os passos necessários.

3. **Seja lógico**

   Encadeie as ações na ordem em que elas devem acontecer.

4. **Seja objetivo**

   Procure atingir o objetivo proposto com o menor número de passos, tomando o cuidado de não pular passos importantes.

5. **Seja claro**

   Evite o uso de palavras e expressões com duplo sentido (termo conhecido tecnicamente como ambiguidade).

6. **Seja compreensível**

   Redija instruções de forma que qualquer pessoa possa entendê-las, não somente profissionais da área de Informática.

7. **Seja crítico**

   Faça a leitura crítica de seu algoritmo para melhorar a escrita e o encadeamento lógico dele.

Depois de lermos essas orientações, que tal voltarmos ao caso do algoritmo para sacar dinheiro em um caixa eletrônico e vermos esses princípios na prática?

Vamos relembrar a descrição das ações do exemplo 2?

Relembre!

Exemplo 2

Para sacarmos dinheiro em um caixa eletrônico, precisamos colocar em prática a seguinte série de ações:

1. **Entrar** no banco.
2. **Procurar** a fila do caixa eletrônico.
3. **Entrar** na fila do caixa eletrônico.
4. **Aguardar** a vez do atendimento.
5. **Inserir** o cartão eletrônico no caixa.
6. **Informar** o valor a ser retirado.
7. **Digitar** a senha.
8. **Pegar** o dinheiro, o cartão eletrônico e o comprovante de saque.
9. **Conferir** o valor em dinheiro.
10. **Sair** do banco.

Você viu como a descrição do exemplo segue à risca as orientações apresentadas para o desenvolvimento de algoritmos? E por que isso acontece?

- ○ Porque as frases são longas, complexas e com vários verbos.
- ○ Porque as frases são curtas, simples e com apenas um verbo.

Comentário

As instruções também estão encadeadas de forma lógica e sem etapas a mais ou a menos.

Com isso, podemos afirmar que se trata de um algoritmo correto, escrito a partir da técnica da descrição narrativa.

Além da descrição narrativa, vamos aprender outras duas formas muito utilizadas de elaborar algoritmos: o diagrama de blocos (fluxograma) e o português estruturado (pseudocódigo). Vamos lá?!

Diagrama de Blocos ou Fluxograma

Os algoritmos também podem ser representados por um método gráfico chamado **diagrama de blocos** ou **fluxograma**.

O diagrama de blocos é uma forma padronizada e eficaz de representarmos os passos lógicos de um processamento.

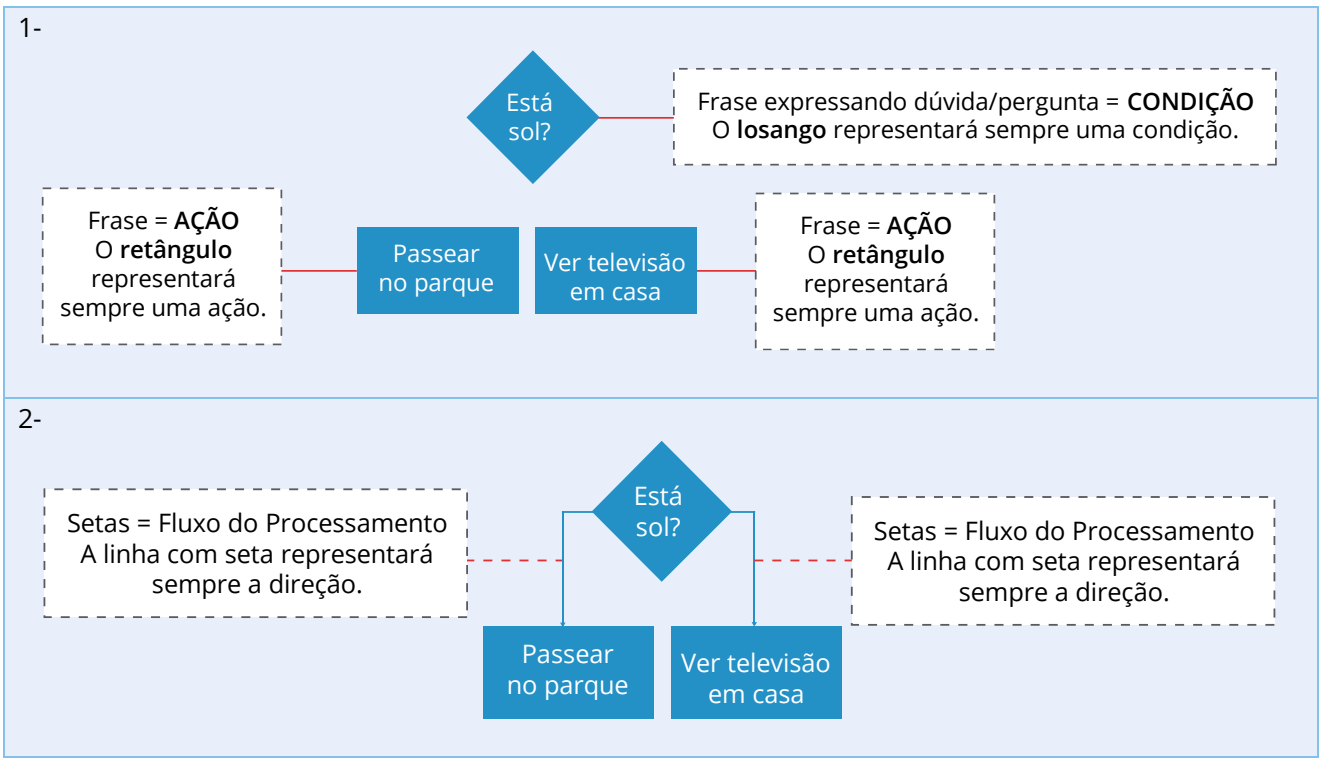
Com o diagrama, podemos definir uma sequência de símbolos com significado bem definido.

Desse modo, a principal função do diagrama é a de facilitar a visualização dos passos de um processamento.

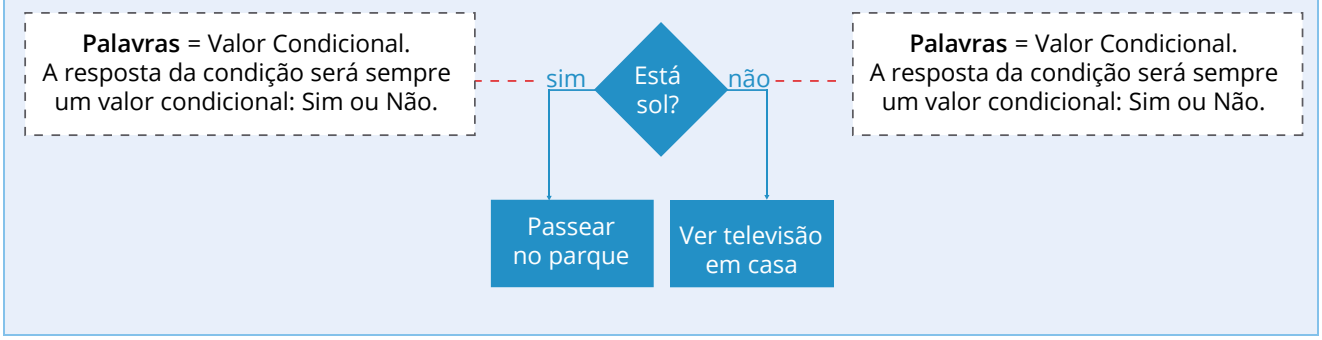
A aparência do método de diagrama de blocos é bem diferente da aparência da descrição narrativa! Está curioso para conhecê-la?

Para construir o diagrama de blocos é preciso fazer o uso padronizado dos símbolos e das palavras. Esses elementos são capazes de mostrar soluções para processos de qualquer grau de complexidade.

Vamos descobrir como isso funciona?



3-



Os fluxogramas são bastante populares uma vez que seus esquemas são relativamente fáceis de entender. Além disso, por meio deles é possível desenhar soluções de processos complexos usando pouca escrita.

Nos diagramas de bloco, cada ação, instrução ou processamento deve ser descrito dentro de um símbolo geométrico de modo resumido.

Nesse sentido, setas e palavras são usadas para indicar o caminho a ser percorrido, ou seja, o fluxo do processamento.

Diagrama de Blocos – Padrão de Utilização das Principais Formas Geométricas

Vamos conhecer os principais símbolos usados na construção de fluxogramas?

Preste bastante atenção! Cada elemento tem função e significado diferentes.

Terminal

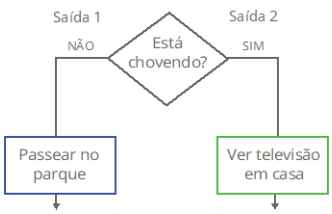
Indica o ponto de início ou término de uma sequência lógica.

Deve ser utilizado com a palavra *início* ou *fim* escrita dentro da figura.

Verificação Condição ou decisão

Indica o ponto de teste (verdadeiro ou falso) ou de tomada de decisão (sim ou não). Conta com uma entrada e permite indicar duas saídas diferentes, sendo uma para cada situação/resposta.

Exemplo:



Seta de fluxo

Indica o sentido do fluxo de informações e instruções. Deve ser utilizada para conectar todos os símbolos do diagrama.

Ação ou processamento

Indica uma ação ou um processamento de qualquer tipo.

Exemplos:

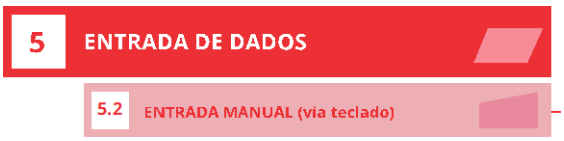
- Calcular o preço final do produto.
- Jogar a embalagem no lixo.

Entrada/saída de dados

Símbolo genérico que indica a entrada ou a saída de dados, respectivamente uma leitura/captura de dados ou uma gravação/apresentação da informação após o processamento, independentemente da forma como essas informações serão inseridas ou disponibilizadas aos usuários.

Entrada de dados (via cartão)

Indica a entrada de dados, através da leitura em cartão perfurado. Este símbolo está em desuso atualmente.



Entrada manual (via teclado)

Indica a entrada de dados feita manualmente pelo usuário, isto é, por meio de digitação.

Exemplos:

- Digitar o preço do produto.
- Entrar/Digitar a nota da avaliação.

Saída de dados (via monitor)

Indica que algum resultado, dado ou informação será exibido na tela do dispositivo ao usuário.

Exemplos:

- Mostrar/Exibir/Apresentar a média do aluno.
- Exibir o valor total do produto.

Agora que você conhece os símbolos mais utilizados, que tal analisar alguns algoritmos descritos em forma de diagrama de blocos para perceber o uso deles na prática?

Veja alguns exemplos de diagrama de blocos.

Exemplo 1

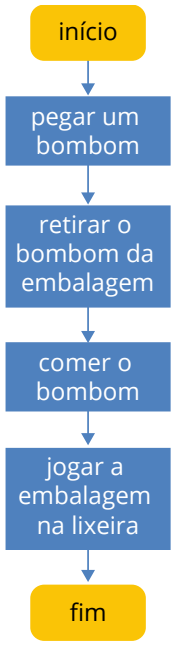
Suponha que você precise criar um algoritmo para comer um bombom.

Como esse algoritmo seria apresentado pela descrição narrativa e pelo fluxograma?

Descrição Narrativa – Algoritmo para comer um bombom

1. **Pegar** um bombom.
2. **Retirar** o bombom da embalagem.
3. **Comer** o bombom.
4. **Jogar** a embalagem na lixeira.

Fluxograma – Algoritmo para comer um bombom



Analise as diferenças entre a Descrição Narrativa e o Fluxograma.

Exemplo 2

Agora é hora de pensarmos em uma situação um pouco mais complexa e contextualizada. Vamos lá?

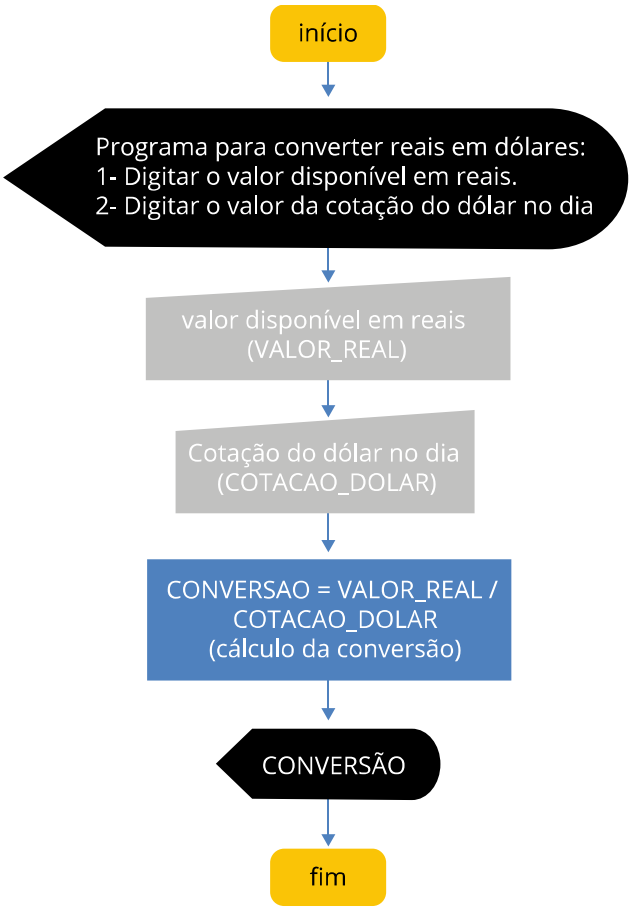
Imagine que você está com uma viagem marcada para os EUA e dispõe de certa quantia em reais para comprar a moeda americana (dólar). Como ficaria o algoritmo para saber quantos dólares será possível comprar com a quantia em reais de que você dispõe?

Veja as duas opções de representação de algoritmo.

Descrição Narrativa – Algoritmo para converter real em dólar

1. **Informar** o valor disponível em reais para comprar dólares.
2. **Informar** a cotação do dólar do dia.
3. **Dividir** o valor disponível pela cotação do dia.
4. **Exibir** o resultado do cálculo para o usuário.

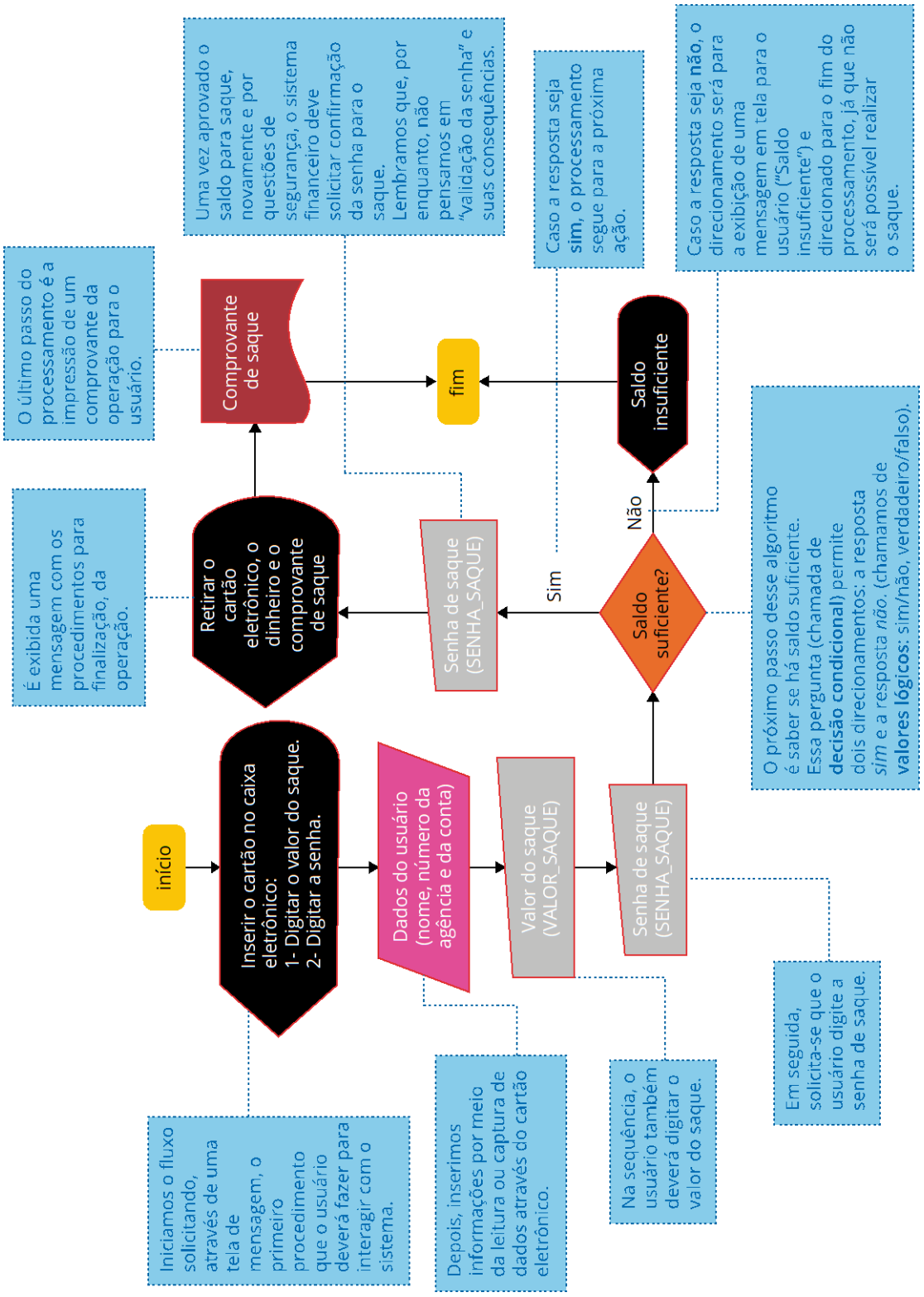
Fluxograma – Algoritmo para converter real em dólar



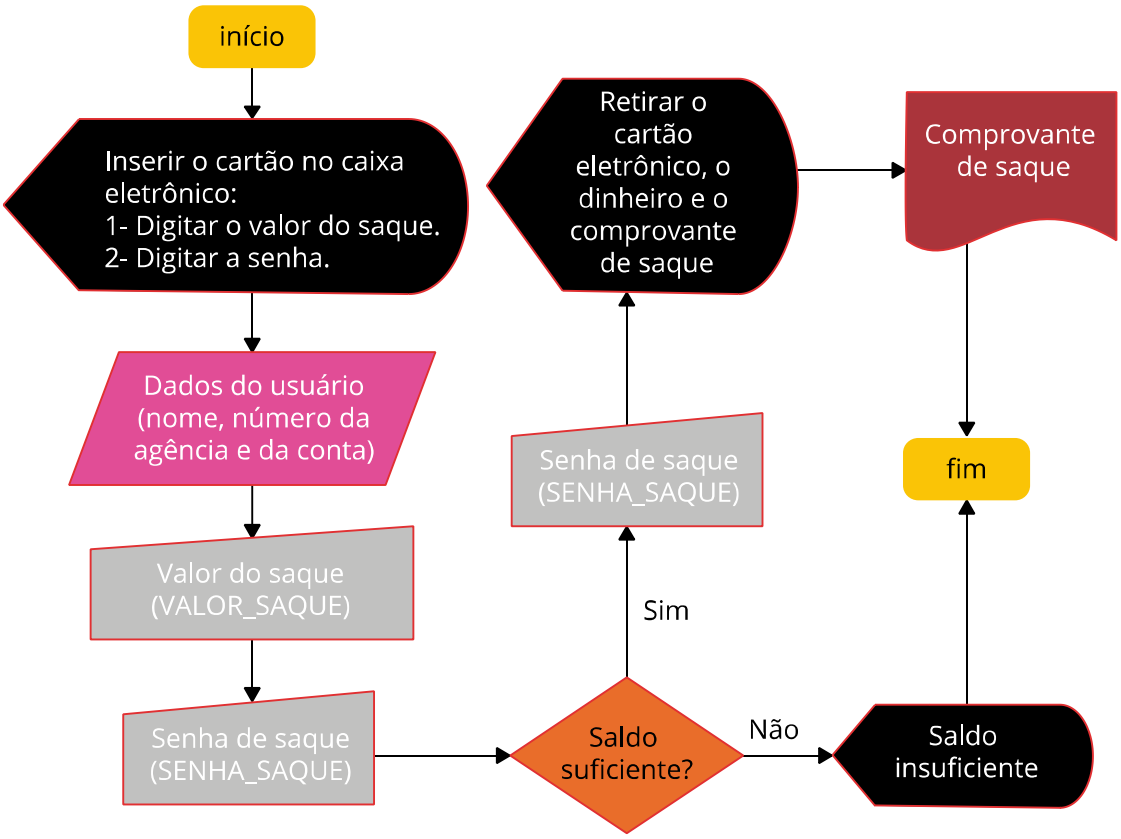
Analise as diferenças entre a Descrição Narrativa e o Fluxograma.

Vamos a mais um exemplo?

Suponha que você precise criar um algoritmo para sacar uma quantia em dinheiro em um caixa eletrônico de um banco. Como ficaria o fluxograma desse algoritmo?



O fluxograma que acabamos de ver é um pouco mais complexo do que os anteriores. Além disso, há outra diferença entre ele e o dos exemplos anteriores. Você consegue perceber? Vamos observá-lo de novo.



O diagrama de blocos que você acabou de ver representa os passos a serem realizados pelo *software* que deverá funcionar no computador do caixa eletrônico.

Veja o fluxograma do caixa eletrônico mais uma vez e imagine o funcionamento do *software*!

Exercícios de Fixação

Agora, veja o quanto você sabe sobre este assunto. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Você recebeu um novo estagiário na empresa em que trabalha e, para auxiliá-lo, precisa passar-lhe algumas informações sobre os algoritmos e as aplicações na informática.

Relacione cada uma das definições a seguir a seu respectivo termo, numerando a segunda coluna de acordo com a primeira.

TERMOS		DEFINIÇÕES
(1) CODIFICAÇÃO	☐	é a informação que indica a um computador o que ele tem de fazer.
(2) INSTRUÇÃO	☐	é a descrição detalhada das etapas que devem ser percorridas para se chegar a uma solução.
(3) ALGORITMO	☐	permite que diferentes profissionais de informática possam compreender o algoritmo.
(4) PADRONIZAÇÃO	☐	é a escrita do algoritmo a partir das regras e dos padrões de uma linguagem de programação.

Questão 2

Você foi designado, na empresa de TI em que trabalha, para fornecer informações básicas a estagiários iniciantes. Um deles perguntou sobre algoritmos e suas funções.

Uma das explicações corretas que você forneceu foi que:

- ○ a criptografia de um programa é feita por meio de algoritmos.
- ○ para criar um programa, não é necessário desenvolver algoritmos.
- ○ o algoritmo é desenvolvido a partir de uma solução para um problema.
- ○ algoritmos podem ser desenvolvidos após a codificação, utilizando uma linguagem de Programação.

Encerramento do Tópico

Neste tópico, você aprendeu a desenvolver algoritmos por meio do uso de descrições narrativas e fluxogramas.

Agora você já compreende esses dois métodos de desenvolvimento de algoritmos e a importância de cada passo desses processos.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Vamos prosseguir em nossos estudos?

Siga para o próximo tópico!

Tópico 3: Pseudocódigo

Neste tópico, apresentaremos recursos de comparação e cálculo que enriquecem o raciocínio lógico. Com isso, teremos maior flexibilidade de pensamentos e encontraremos várias soluções em diferentes cenários.

Conteúdos:

- Pseudocódigo
- Recursos de comparação e cálculo.

Ao finalizar este tópico, você será capaz de:

- Atribuir valores às variáveis.
- Identificar quando e como usar os operadores aritméticos, relacionais e lógicos.

Pseudocódigo ou Português Estruturado

Agora, vamos conhecer o método pseudocódigo[1], também chamado de **Português estruturado**.

O pseudocódigo usa termos da língua portuguesa encadeados e organizados de forma parecida com as diferentes linguagens de programação. Seu objetivo é simplificar o processo de programação.

Para entender um algoritmo escrito em pseudocódigo, você não precisa ter o conhecimento prévio de nenhuma linguagem de programação específica.

1. Pseudocódigo – é uma técnica natural, comum e originário a quem o codifica por meio de uma linguagem natural, no nosso caso, a língua portuguesa. Permite a compreensão por parte de qualquer pessoa e NÃO exige que nenhum padrão de sintaxe seja seguido na linguagem da programação.

Nesse tipo de algoritmo, as estruturas lógicas são escritas em uma linguagem próxima ao português e na sequência em que devem ser executadas.

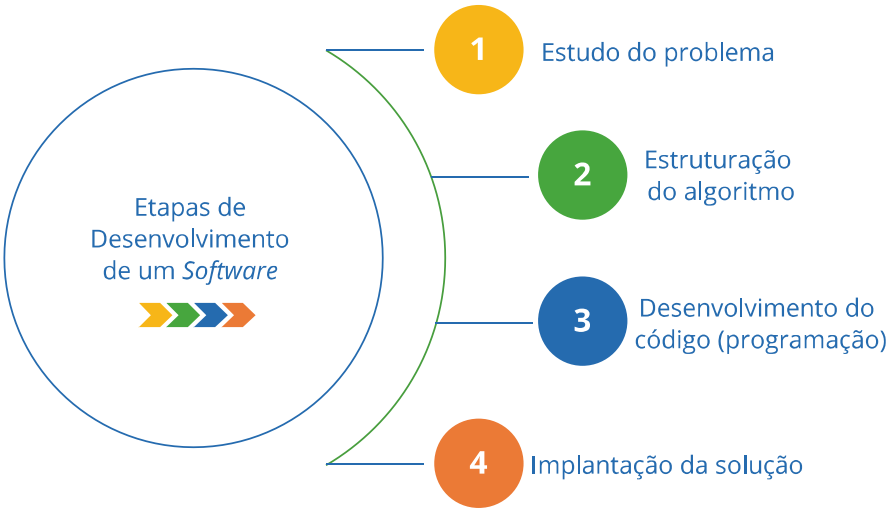
Ficou curioso? Vamos entender melhor com uma demonstração a seguir.

Saiba mais!

Nos países em que o idioma principal é o português, muitos se referem ao pseudocódigo como "Portugol".

Ao construir um projeto de programa (algoritmo) em pseudocódigo, você estará representando o encadeamento lógico necessário para resolver um problema da forma mais próxima que existe de uma linguagem de programação.

A opção pelo pseudocódigo na etapa de estruturação do algoritmo proporciona um ganho de tempo na etapa de desenvolvimento do código. Parece vantajoso, não? E você se lembra das quatro etapas necessárias para a elaboração de um programa? Reveja as etapas de desenvolvimento de um *software*.



1. Estudo do problema

Análise do problema e de seu contexto, para identificar a situação, refletir sobre suas características e apresentar as possíveis soluções.

2. Estruturação do algoritmo

Escolha da melhor solução possível para resolver o problema e estruturação da proposta em forma de "projeto de programa" (algoritmo).

3. Desenvolvimento do código (programação)

Escrita do algoritmo a partir das regras e dos padrões de uma linguagem de programação específica, tal como C#, Java, C, C++, VB, PHP.

É nesta etapa que ocorre a codificação do algoritmo.

4. Implantação da solução

Implantação do programa em situação de vida real, para verificar se o problema foi resolvido, e realização de eventuais ajustes (manutenção).

Agora, vejamos como utilizar o português estruturado na prática! Para isso, vamos voltar ao algoritmo de conversão de real em dólar.

Nesse caso, primeiro relembraremos o algoritmo usando a descrição narrativa e, em seguida, veremos como fica o algoritmo usando o pseudocódigo.

Descrição Narrativa

Algoritmo para converter real em dólar:

1. **Informar** o valor disponível em reais para comprar dólares.
2. **Informar** a cotação do dólar do dia.
3. **Dividir** o valor disponível pela cotação do dia.
4. **Exibir** o resultado do cálculo para o usuário.

Pseudocódigo

```
Algoritmo Conversao_real_dolar
     // rotina que converte um determinado valor em reais em dólares
          Var
          VALOR_REAL, COTACAO_DOLAR, CONVERSAO: real
               VALOR_REAL = 0
               COTACAO_DOLAR = 0
               CONVERSAO = 0
     Início
          Escreva "Programa para converter reais em dólares"
          Escreva "Informe o valor disponível em reais (para comprar dólares):
          R$"
          Leia VALOR_REAL
          Escreva "Informe o valor do dólar em reais (cotação do dia): R$"
          Leia COTACAO_DOLAR
          CONVERSAO = VALOR_REAL / COTACAO_DOLAR
          Escreva "Com essa quantia será possível comprar: US$"
          Escreva CONVERSAO
          Escreva "Boa viagem!"
     Fim
```

Analise as diferenças entre a Descrição Narrativa e o Pseudocódigo.

Tal como você pôde perceber, o pseudocódigo é uma técnica que organiza os algoritmos em blocos formados por comandos escritos.

Para isso, são usados, por exemplo, os **comandos** Leia e Escreva para inserção (captura) e exibição (apresentação) de dados, respectivamente.

Além dos comandos, o pseudocódigo permite a declaração de variáveis e expressões aritméticas para realizar contas.

```
Algoritmo Conversao_real_dolar
// rotina que converte um determinado valor em reais em dólares
     Var
     VALOR_REAL, COTACAO_DOLAR, CONVERSAO: real
           VALOR_REAL = 0
           COTACAO_DOLAR = 0
           CONVERSAO = 0
Início
     Escreva “Programa para converter reais em dólares”
     Escreva “Informe o valor disponível em reais (para comprar dólares): R$”
     Leia VALOR_REAL
     Escreva “Informe o valor do dólar em reais (cotação do dia): R$”
     Leia COTACAO_DOLAR
     CONVERSAO = VALOR_REAL / COTACAO_DOLAR
     Escreva “Com essa quantia será possível comprar: US$”
     Escreva CONVERSAO
     Escreva "Boa viagem!"
Fim
```

Você notou como o pseudocódigo é uma forma bastante detalhada de **escrever sequências lógicas de instruções e processamentos**?

Nesse sentido, a principal característica desse método é ser muito semelhante ao jeito como as linguagens de programação são codificadas. E foi justamente o fato de ser muito parecido com os códigos de programação que fez com que surgisse o nome pseudocódigo, ou seja, “falso” (fictício) código.

É importante que você saiba que o pseudocódigo tem grande aceitação entre os programadores. Mas por que será que isso ocorre?

O principal motivo está no fato de o pseudocódigo possibilitar uma tradução praticamente direta para qualquer linguagem de programação específica. Justamente por isso, iremos apresentar a você vários exemplos de algoritmos em pseudocódigo daqui em diante.

Como programador da área de Informática, você deverá ser capaz de escrever, interpretar, testar, localizar e resolver erros de lógica tanto em fluxogramas como em pseudocódigos.

Por isso, mãos ao *mouse* (ou ao teclado)!

Apresentaremos, a seguir, a forma básica de um algoritmo em português estruturado:

```
Algoritmo<nome_do_algoritmo>
// <explicação resumida do que a rotina irá fazer>
     Var
     <declaração das variáveis e do tipo de informação que elas
     receberão>
Início
     <instruções e processamentos encadeados de forma lógica>
Fim
```

Todo algoritmo começa com a definição de um nome de identificação para ele, na primeira linha: <nome_do_algoritmo>

Na linha de baixo, deve ser feita a declaração de variáveis, cuja palavra reservada é **Var**. O termo **Var** é uma redução da palavra "variável". Nesse sentido, o termo **Var** é um marcador, assim como as palavras INÍCIO, FIM, SE, SENÃO, ENTÃO, FAÇA CASO etc. Um marcador ou uma palavra reservada são termos interpretados de forma diferenciada pelos programas. Por isso, não devem ter outros usos, como nomear variáveis.

Agora ficou mais fácil identificar um algoritmo estruturado em pseudocódigo, não é mesmo?

Alguns recursos disponíveis na internet podem ser úteis para que você desenvolva a habilidade de construir, interpretar e corrigir pseudocódigos.

A seguir, indicamos dois aplicativos que permitem a construção e o teste de algoritmos escritos em pseudocódigo. Eles são gratuitos e fáceis de usar.

VisuAlg (www.apoioinformatica.inf.br/produtos/visualg/**)**

O **VisuAlg** (visualizador de algoritmo) é um programa utilizado para edição, interpretação e execução de algoritmos com linguagem próxima a do português estruturado, como um programa normal de computador.

O *software* é de livre uso e distribuição, além de ser empregado no ensino de programação em várias escolas e universidades, no Brasil e no exterior.

Portugol Studio (http://lite.acad.univali.br/portugol/)

O **Portugol Studio** é um ambiente para aprender a programar, voltado para os iniciantes em programação que falam o idioma português.

O *software* possui uma sintaxe fácil, diversos exemplos e materiais de apoio à aprendizagem.

Também possibilita a criação de jogos e outras aplicações.

Anote essas dicas e pratique bastante!

Tipos de Dado

Programar computadores requer a definição e o uso de espaços em memória para receber informações inseridas pelos usuários. Os espaços em memória recebem informações que podem variar no decorrer de um processamento. Por isso, é muito importante aprender a criar e definir o tipo, bem como manipular variáveis e constantes.

Em linhas gerais, variáveis são espaços em memória que servem para guardar um tipo de dado ou informação. Esses espaços em memória devem receber nomes específicos e únicos, para que possam ser referenciados exclusivamente e usados sempre que necessário.

Ao criar um espaço em memória, com conteúdo variável ou constante, precisamos definir e registrar o tipo de dado que será inserido e manipulado. Vamos conhecer os três tipos de dados:

literais | numéricos | lógicos

literais

Dados literais – ou caracteres – são sequências contendo letras, números e outros símbolos especiais. Uma sequência de caracteres deve ser indicada entre aspas (“”). Esse tipo de dado é conhecido também como Alfanumérico, *String*, Literal ou Cadeia. Como exemplos, temos: “Fundação Bradesco”, “Técnico em Desenvolvimento de Sistemas”, “84”, “843.48”, entre outros.

numéricos

numéricos reais

Os dados numéricos reais são números positivos, negativos e fracionários. Como exemplo, temos: 584.87, -848.43, 84, 82, -19.20. Lembrando que, na formatação universal, a parte fracionária é separada por ponto “.” e não por vírgula “,”, como fazemos na formatação brasileira.

numéricos inteiros

Os dados numéricos inteiros são definidos como tipos inteiros. Podem ser dados numéricos positivos ou negativos. Nesse tipo, não se encaixam números fracionários. Como exemplo, temos: 10, -10, 5, 85, -33, 88, -67, entre outros.

lógicos

Os dados lógicos são apenas dois valores utilizados por elementos condicionais para tomadas de decisão. Algumas bibliografias escrevem os valores lógicos entre pontos, para dar mais ênfase e diferenciá-los de outros conceitos ou definições (como variáveis ou valores literais). Por exemplo: .***V***.ou .***F***.Esses dois valores podem ser:

- Verdadeiro ou Falso, ou .***V***. ou .***F***.
- Sim ou Não
- 0 ou 1

São também conhecidos como dados booleanos, por referência a George Boole, matemático que deu nome à álgebra booleana, expressão que trata desses tipos de dado.

Uso e Funções de Constantes e Variáveis

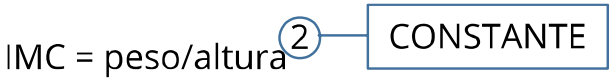
Já mencionamos algumas vezes as palavras "constante" e "variável". Você deve estar curioso para saber melhor o que elas significam, não é mesmo? Agora, vamos definir cada uma delas e dar exemplos de suas funções.

Chamamos de constante todo valor fixo que não será alterado durante a execução de um programa.

Conforme o contexto de utilização, as constantes podem ser classificadas como:

- Numéricas
- Literais
- Lógicas

Vejamos um exemplo de uso de **CONSTANTE**:



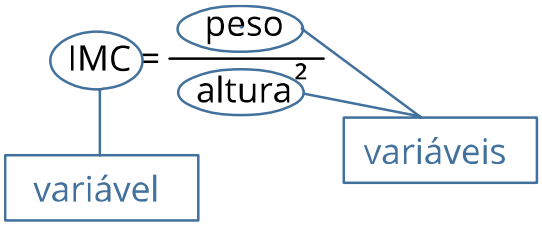
Observe que, na fórmula para calcular o IMC (Índice de Massa Corporal), o número dois é um valor constante, pois não muda. Desse modo, o número dois é diferente dos demais elementos (IMC, peso e altura), que vão variar em cada caso. Em relação ao tipo do dado, o número dois deve ser classificado como uma constante do tipo numérica inteira.

Ficou claro o conceito de constante agora?

Agora, vamos entender melhoras as **VARIÁVEIS**.

Uma variável representa um endereço da memória RAM que armazena, temporariamente, valores e informações. Esses endereços recebem um nome e um conteúdo. Cada vez que mencionamos o nome da variável, seu conteúdo é manipulado.

Vamos retomar o exemplo da fórmula para calcular o IMC (Índice de Massa Corporal)? Observe a seguir:



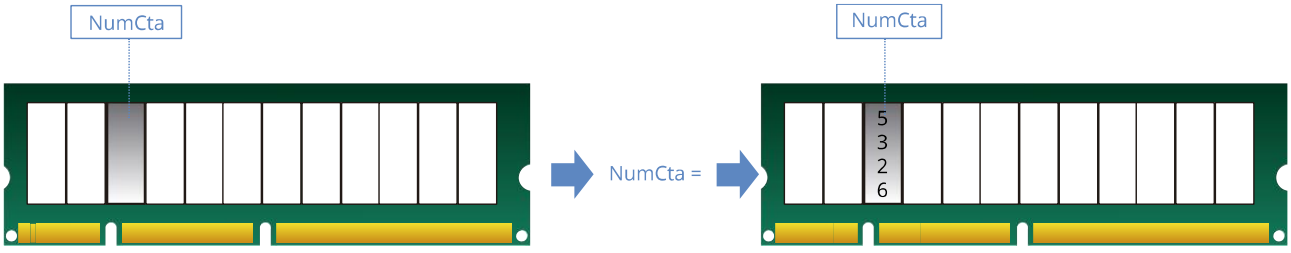
Simples, não?

É importante sabermos que as variáveis podem armazenar todos os tipos de dados. Além disso, para criar variáveis ou constantes, precisamos definir um nome simbólico, uma identificação a partir da qual elas serão chamadas, o tipo de dado que elas receberão e um valor inicial.

A criação de variáveis ocorre no momento de codificação do programa. Desse modo, precisamos atribuir um nome simbólico a determinado endereço da memória RAM.

No decorrer do programa, esse nome vai ser utilizado para manipular a informação contida no endereço da memória relacionado à variável.

Vejamos a imagem a seguir, apenas para ilustrar uma simulação de armazenamento de dado em uma célula da memória RAM:



Para nomear uma variável, precisamos seguir algumas regras. Você sabe quais são essas regras? Vamos descobrir!

Os nomes das variáveis:

- Devem sempre começar por uma letra.
- Não devem ter espaços em branco.
- Não devem ter caracteres especiais – no máximo, o símbolo sublinhado.

Por exemplo:

```
Nome_cliente
Num_funcionario
Datanascimento
```

Além disso, precisamos saber que as variáveis devem ser declaradas sempre no início dos algoritmos e dos programas. Veremos um exemplo mais adiante.

Você sabe até quando os valores permanecem armazenados nas variáveis?

Os valores armazenados nas variáveis permanecem:

- Até que o computador seja desligado pelo usuário ou por falta de energia, já que são armazenados na memória RAM.
- Por término do programa ou da rotina onde foram criados.
- Até que seja atribuído um novo valor para a mesma variável.

Vejamos um exemplo:

Se escrevermos **A = 5**, o valor *cinco* será o conteúdo da variável **A**.

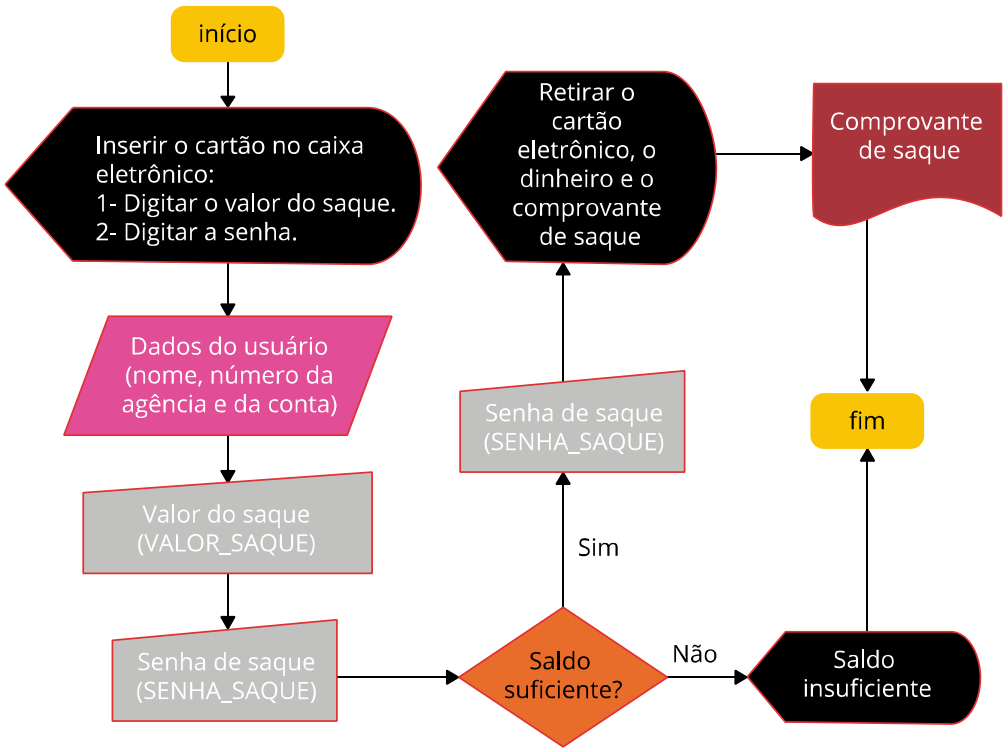
Se, mais adiante, escrevermos **A = 20**, o valor *vinte* passará a ser o novo conteúdo da variável **A**.

Nesse caso, o valor anterior **5** (cinco) será eliminado (substituído).

Depois de tudo o que vimos até aqui, está na hora de interpretarmos um algoritmo mais complexo, escrito em portugol (pseudocódigo). Aceita o desafio?!

Para isso, vamos retomar o fluxograma para sacar dinheiro em um caixa eletrônico.

Fluxograma



Agora, veja como essa sequência poderia ficar, com alguns ajustes, em portugol.

Portugol

```
Algoritmo Saque_dinheiro_caixa_eletronico
// rotina para sacar dinheiro em um caixa eletrônico
       Var
       VALOR_SAQUE, SALDO: real
       SENHA_SAQUE, NUM_AGENCIA, NUN_CONTA: inteiro
            VALOR_SAQUE = 0
            SENHA_SAQUE = 0
            NUN_CONTA = 0
Início
       Escreva "Seja bem-vindo!"
       Escreva "Informe o número da sua agência:"
       Leia NUM_AGENCIA
       Escreva "Informe o número da sua conta:"
       Leia NUN_CONTA
// após ler os dados, o sistema deverá trazer do banco de dados o valor do
saldo do usuário na variável SALDO
       Escreva "Informe o valor do saque:"
       Leia VALOR_SAQUE
       Se VALOR_SAQUE> 0 e VALOR_SAQUE<= SALDO Então
            SALDO = SALDO - VALOR_SAQUE
            Escreva "Saque efetuado com sucesso. Seu saldo atual é R$:"
            Escreva SALDO
// após apresentar o novo saldo o sistema deve fazer a contagem das notas e
disponibilizar a quantia sacada ao usuário
            Escreva "Retire o cartão, o dinheiro e o comprovante de saque."
      Senão
            Escreva "Saldo insuficiente. Não será possível realizar a
            operação."
      Fim Se
Fim
```

Vejamos a Descrição narrativa.

Descrição Narrativa

1. **Entrar** no banco.
2. **Procurar** a fila do caixa eletrônico.
3. **Entrar** na fila do caixa eletrônico.
4. **Aguardar** a vez do atendimento.
5. **Inserir** o cartão eletrônico no caixa.
6. **Informar** o valor a ser retirado.
7. **Digitar** a senha.
8. **Pegar** o comprovante de saque, o cartão eletrônico e o dinheiro.
9. **Conferir** o valor em dinheiro.
10. **Sair** do banco.

Analise as diferenças entre a Descrição Narrativa, o Fluxograma e o pseudocódigo.

Como já observamos, a sequência lógica para sacar dinheiro em um caixa eletrônico foi se ajustando para que o mesmo problema fosse resolvido de três formas diferentes: descrição narrativa, fluxograma e pseudocódigo. Curioso isso, não é mesmo?

Esse exemplo mostra o quanto essas três formas de algoritmos são diferentes e como os processos podem ser mais ou menos detalhados, a depender da técnica utilizada. A partir dessa experiência, podemos chegar a algumas conclusões, concorda? Vamos conferir as técnicas:

Descrição Narrativa

A **descrição narrativa** é a forma mais próxima da linguagem comum.

Diagrama de Blocos

O **diagrama de blocos** ou fluxograma é uma técnica intermediária que usa formas geométricas e frases curtas para descrever um processamento.

Pseudocódigo

O português estruturado ou **pseudocódigo** é a forma mais detalhada e mais próxima dos códigos de programação.

Saiba mais!

De acordo com Oliveira (2004), “há diversas formas de representação de algoritmos que diferem entre si pela quantidade de detalhes de implementação que fornecem ou, inversamente, pelo grau de abstração que possibilitam com relação à implementação do algoritmo em termos de uma linguagem de programação específica.”

As três técnicas são úteis e importantes. Dessa forma, caberá a você definir qual é a mais apropriada, tendo em vista o contexto em que estiver planejando um programa. Boa escolha!

Teste de Mesa

Uma vez que escolhemos a forma de representar o algoritmo e concluímos sua estrutura, como saber se o encadeamento lógico está correto?

Ao desenvolver um algoritmo, devemos submeter a rotina elaborada a um processo de verificação.

O objetivo da verificação é simular o funcionamento e a eficiência do código ao resolver o problema proposto.

Em outras palavras, a ideia é testar a sequência lógica para ver se o encadeamento das instruções e dos processamentos está correto ou para ver se há necessidade de ajustes.

No jargão da Informática, esse tipo de teste é chamado de **teste de mesa**.

Na prática, esse termo significa seguir as instruções dos algoritmos, de maneira precisa, para verificar se o procedimento utilizado está correto ou não.

Para aplicar a técnica denominada teste de mesa, precisaremos de papel, caneta e paciência, para ler e realizar cada instrução encadeada como se fôssemos um computador.

Podemos aplicar o teste de mesa a qualquer tipo de algoritmo, seja uma descrição narrativa, um fluxograma ou um pseudocódigo.

Vamos descobrir como aplicar esse teste na prática? Mas antes, leia a tirinha para descontrair um pouco.

Fonte: NOEL, 2013. Disponível em: http://vidadeprogramador.com.br/. Acesso em: 28 jan. 2016.

Para demonstrar o teste de mesa, partiremos do algoritmo **Quatro_operacoes_basicas**, representado no pseudocódigo a seguir. Esse algoritmo foi construído com o objetivo de solicitar ao usuário a inserção de dois números quaisquer para a realização das quatro operações matemáticas básicas. Preparado? Então vamos lá!

```
Algoritmo Quatro_operacoes_basicas
// rotina que recebe dois números e que apresenta o resultado
das quatro operações feitas com eles
  Var
  NUM_A, NUM_B, SOMA, SUBTRACAO, MULTIPLICACAO, DIVISAO: real
    NUM_A = 0
    NUM_B = 0
    SOMA = 0
    SUBTRACAO = 0
    MULTIPLICACAO = 0
    DIVISAO = 0
Início
    Escreva "Programa que executa as quatro operações básicas"
    Escreva "Informe o primeiro número:"
    Leia NUM_A
    Escreva "Informe o segundo número:"
    Leia NUM_B
    SOMA = NUM_A + NUM_B
    SUBTRACAO = NUM_A - NUM_B
    MULTIPLICACAO = NUM_A * NUM_B
    DIVISAO = NUM_A / NUM_B
    Escreva "A soma dos dois números é"
    Escreva SOMA
    Escreva "A subtração dos dois números é"
    Escreva SUBTRACAO
    Escreva "A multiplicação entre os dois números é"
    Escreva MULTIPLICACAO
    Escreva "A divisão entre os dois números é"
    Escreva DIVISAO
Fim
```

Para testar algoritmos como esse, que conta com a inserção de dados, processamentos e exibição de resultados, o primeiro passo é criar uma tabela, colocando, em cada coluna, o nome das variáveis declaradas.

Veja, a seguir, a tabela criada para nosso exemplo:

TESTE	NUM_A	NUM_B	SOMA	SUBTRACAO	MULTIPLICACAO	DIVISAO
1º						
2º						
3º						

Observe que tabela prevê a realização de três testes. No entanto, esse número pode variar de acordo com o contexto e a complexidade do algoritmo. Depois de montada a tabela, é preciso ler atentamente cada instrução a fim de verificar o encadeamento das ações, a clareza das instruções e a escrita dos processamentos.

Lembre-se de que a proposta do teste de mesa é simular a realização da rotina do algoritmo por meio do preenchimento da tabela.

Tabela pronta? Vamos ver como ficaria o teste do algoritmo das quatro operações?

Demonstração

1º TESTE

Considere os valores 30 e 60 como os dois números (**NUM_A** e **NUM_B**, respectivamente) solicitados pelo algoritmo. A partir desses dois valores, siga o encadeamento de ações proposto nas linhas do pseudocódigo.

Quando o algoritmo solicitar a ação **Escreva**, preencha o campo correspondente na tabela de caneta azul. Para marcar as ações **Leia**, coloque o valor lido entre parênteses de caneta vermelha. Essas regras simples são importantes para registrar a verificação de todas as ações previstas no algoritmo.

TESTE	NUM_A	NUM_B	SOMA	SUBTRACAO	MULTIPLICACAO	DIVISAO
1º	(30)	(60)	90	-30	1800	0,5
2º						
3º						

2º TESTE

Repita o procedimento realizado no primeiro teste, considerando os valores -20 e 40 como NUM_A e NUM_B, respectivamente.

TESTE	NUM_A	NUM_B	SOMA	SUBTRACAO	MULTIPLICACAO	DIVISAO
1º	(30)	(60)	90	-30	1800	0,5
2º	(-20)	(40)	20	-60	-800	0,5
3º						

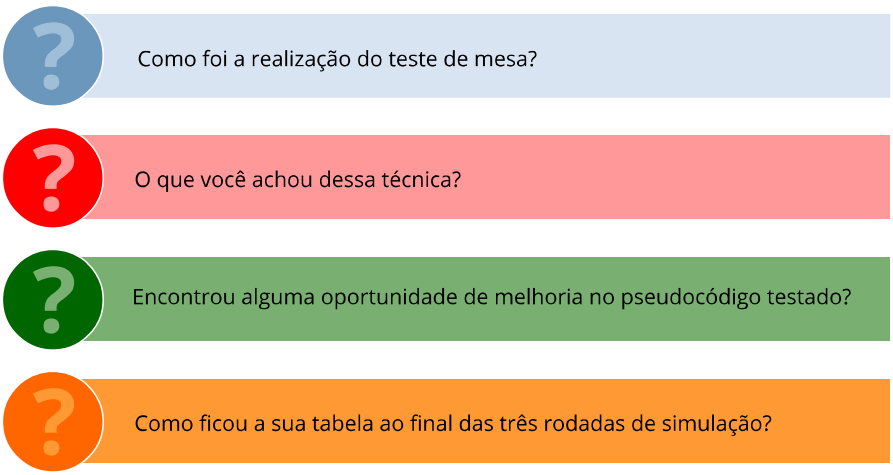
3º TESTE

Repita o procedimento, considerando, agora, os valores 20 e -40.

TESTE	NUM_A	NUM_B	SOMA	SUBTRACAO	MULTIPLICACAO	DIVISAO
1º	(30)	(60)	90	-30	1800	0,5
2º	(-20)	(40)	20	-60	-800	0,5
3º	(20)	(-40)	-20	60	-800	0,5

Como foi a realização do teste de mesa? O que você achou da técnica? Encontrou alguma oportunidade de melhoria no pseudocódigo testado?

Após a realização do passo a passo, queremos saber:



Para facilitar seus estudos, compare sua tabela com a tabela que elaboramos. Verifique, com atenção, se você aplicou a técnica do teste de mesa corretamente.

Então, o que achou?

Tabela para a realização do teste de mesa

	NUM_A	NUM_B	SOMA	SUBTRACAO	MULTIPLICACAO	DIVISAO
1º teste	(30)	(60)	90	-30	1800	0,5
2º teste	(-20)	(40)	20	-60	-800	-0,5
3º teste	(20)	(-40)	-20	60	-800	-0,5

Utilização de Pseudocódigos

Como você deve ter percebido, a criação de algoritmos em pseudocódigo é relativamente simples, não é mesmo?

O pseudocódigo é um método que não usa estruturas complexas e rígidas – como as sintaxes de linguagens de programação –, nem exige ambientes especiais para edição.

Em outras palavras, o pseudocódigo pode ser encarado com uma versão similar e simplificada dos códigos de programação, com o benefício de poder ser escrito em qualquer editor de texto.

Isso significa que podemos escrever um algoritmo em pseudocódigo usando o Bloco de Notas, por exemplo.

Essa característica faz do pseudocódigo uma ferramenta útil e simplespara elaboração de projetos de programas.

Agora que você já entendeu a utilidade do pseudocódigo, vamos analisar a estrutura de um algoritmo em pseudocódigo.

Neste exemplo, a rotina deverá definir os tipos de variáveis relacionadas ao cadastro de um livro, receber os dados do usuário e exibi-los na tela.

Vejamos o pseudocódigo:

```
ALGORITMO Livro {definição do nome do programa}
    VAR
    CodigoDoLivro: inteiro {Nome da Variável: Tipo da variável}
    Titulo, Autor, Editora: literal {declaração de variáveis, todas como
    literal}
    CodigoDoLivro = 0
INICIO
    Escreva "Este é um programa em pseudocódigo que exibe na tela os
    dados de um livro"
    Escreva "Digite o código do livro"
    Leia CodigoDoLivro
    Escreva "Digite o título do livro"
    Leia Titulo
    Escreva "Digite o autor do livro"
    Leia Autor
    Escreva "Digite a editora do livro"
    Leia Editora
    Escreva "O código do livro é", CodigoDoLivro
    Escreva "O título do livro é", Titulo
    Escreva "O autor do livro é", Autor
    Escreva "A Editora do livro é", Editora
FIM
```

Você percebeu como a descrição é clara e de fácil compreensão?

Uma das principais vantagens desse método é que ele pode ser entendido por “não programadores”.

Com isso, os desenvolvedores podem apresentar suas ideias e validar sequências instrucionais com profissionais que não são da área de programação.

Trabalhar com pseudocódigo parece ser muito eficiente, não acha?

A partir de agora, vamos estipular algumas regras de padronização para a construção de um pseudocódigo. Fique atento a elas!

Regra 1

Todo algoritmo em pseudocódigo deve ser iniciado com **Algoritmo**: **NomeDoAlgoritmo**.

Regra 2

O início e fim do programa são limitados pelos marcadores **Início** e **Fim**.

Regra 3

As variáveis são declaradas no início do algoritmo, abaixo do marcador **Var**, da seguinte forma: **NomeDaVariável**: Tipo da variável.

Regra 4

Os nomes das variáveis **NÃO** podem:

- Iniciar por número (erro: 1nome)
- Ter espaço (erro: nome completo)
- Ter caracteres especiais (´, `, ~,ç ,- e outros).

Regra 5

As palavras-reservadas devem ser evitadas: **Início**, **Fim**, **Var**, **Se** e **Senão**.

Regra 6

Os nomes das variáveis são *case sensitive*[2]. Dessa forma, ao manipularmos variáveis, devemos usar o mesmo nome declarado no início, considerando o uso de letras maiúsculas e minúsculas.

2. *Case sensitive*–Considera a diferença no uso de letras maiúsculas e minúsculas.

Regra 7

O comando **Leia** deve ser usado para receber (capturar) dados do usuário, fase do processamento conhecida como "Entrada de Dados".

Regra 8

O comando **Escreva** deve ser usado para exibir (apresentar, mostrar) dados ao usuário, fase do processamento conhecida como "Saída de Dados".

Regra 9

Os textos a serem exibidos na tela ou que devam ser inseridos como caractere são colocados entre "aspas" (representação universal de um valor literal ou *string* – "Sistemas").

Regra 10

Os comentários sobre o código podem ser inseridos {entre chaves} (incomum por confundir com agrupamentos) ou inseridos utilizando // (mais comum) no início da linha de instruções. O comentário não altera a execução do código. Contudo, ele é de fundamental importância para documentar e tornar inteligíveis as escolhas de programação realizadas no código para outros programadores.

As regras são muito importantes, já que todos os elementos do pseudocódigo – e também dos códigos – são fundamentais para o funcionamento correto do programa. Desse modo, entender o rigor da padronização e aplicá-la é essencial a quem deseja atuar com programação de computadores.

Operadores: Tipos e Usos

Com tudo o que vimos até aqui, podemos afirmar que programar computadores é lidar com informações e instruções para manipular dados e chegar a resultados.

Nesse sentido, os programas podem realizar cálculos simples e complexos, comparar dados inseridos pelos usuários e dar diferentes encaminhamentos a esses dados, dependendo do processamento.

O que o computador fará depende do uso dos chamados **operadores**. E o que são operadores?

> Operadores são elementos que atuam sobre os operandos (variáveis ou constantes), ou seja, são símbolos ou palavras reservadas (*true*/*false*) por meio dos quais é possível fazer cálculos, comparar resultados e atribuir valores às variáveis.
>
> Com isso, os operadores são a chave para que possamos incrementar, decrementar, comparar e avaliar dados dentro do computador.

Percebeu como é importante conhecer e saber utilizar corretamente os operadores?

Agora que já sabemos o que é um operador, vamos conhecer os tipos de operadores.

Existem três tipos diferentes de operadores: aritméticos, relacionais e lógicos. Vejamos mais detalhes:

Operadores aritméticos

Possibilitam realizar cálculos em expressões matemáticas ou aritméticas. Suas palavras ou símbolos reservados são:

Operação	Símbolo
Soma	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	** ou ^

Operadores relacionais

Possibilitam fazer comparações ou relacionamentos entre dados numéricos e literais (caracteres). Suas palavras ou símbolos reservados são:

Operação	Símbolo
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=
Igual a	=
Diferente de	<> ou # ou !=

Saiba mais!

Os operadores relacionais ou de comparação são muito utilizados nas codificações pelas Linguagens de Programação. Entretanto, alguns operadores relacionais possuem outros símbolos para representá-los, mas com as mesmas finalidades. Por exemplo:

= (igual a), que representa uma **igualdade** na lógica, normalmente quando se deseja fazer uma comparação em uma decisão condicional, é simbolizado nas linguagens de programação com duplo sinal de igual ==, x==2 (SE x é igual a 2). Apenas = é utilizado como atribuição, ou seja, x = 2 (atribuição do valor 2 para x), o que não é uma comparação.

<> (diferente de), que representa uma **desigualdade** na lógica, é simbolizado de diferentes maneiras nas linguagens de programação. Para utilizar o <> (diferente de) nas linguagens, você encontrará outros símbolos, como #, != e <> entre outros. Esse detalhamento você aprenderá quando for estudar as linguagens de programação.

Operadores lógicos

Também chamados de booleanos. Permitem comparar ou relacionar resultados de expressões. Suas palavras ou símbolos reservados são:

Operação	Símbolo
Conjunção	E
Disjunção (não exclusiva)	OU
Negação	NÃO
Disjunção (exclusiva)	XOU (lê-se “ou exclusivo”)

Saiba mais!

Os operadores lógicos ou booleanos também são muito utilizados nas codificações pelas Linguagens de Programação. Quando utilizamos operadores lógicos, obrigatoriamente, estamos nos colocando em uma situação de decisão condicional **SE**, isto é, um questionamento para resultar em uma situação verdadeira ou falsa. Por exemplo:

SE curso == “sistemas” **E** período == “matutino” então faça... SENÃO faça...

Observe que, nessa decisão condicional **SE**, além de ser utilizado o operador lógico **E**, também foi empregado o operador relacional/comparação de igualdade ==. Assim será na codificação!

Os operadores lógicos são utilizados na programação, conforme a linguagem. Para codificar **E**, você encontrará os símbolos **&&**. Para codificar **OU**, você encontrará os símbolos **!= ||**. Para codificar **NÃO**, você encontrará o símbolo **! !=**.

XOU não é tão empregado, mas você aprenderá com detalhes quando for estudar as linguagens de programação que emprega este elemento.

Atribuição de Valores às Variáveis

Ao longo da rotina estabelecida em um algoritmo ou programa, as variáveis devem receber valores de acordo com o tipo de dado que foi declarado inicialmente. Nesse sentido, se declararmos uma variável como dado literal (caractere), não poderemos fazer cálculos com ela.

No exemplo do pseudocódigo para cadastro de livros, associamos o valor digitado pelo usuário às variáveis.

Pseudocódigo para cadastro de livros

```
ALGORITMO Livro {definição do nome do programa}
        VAR
        CodigoDoLivro: inteiro
        Titulo, autor, editora: caractere {declaração de variáveis}

INICIO
        Escreva "este é um programa em pseudocódigo que exibe na tela os dados de um livro"
        Codigodolivro = 1
        Titulo = "Inferno"
        Autor = "Dan Brown"
        Editora = "Editora Arqueiro"
        Escreva "o código do livro é", CodigoDoLivro {irá exibir 1}
        Escreva "o título do livro é", Titulo {Inferno}
        Escreva "o autor do livro é", Autor {irá exibir Dan Brown }
        Escreva "a editora do livro é", Editora {irá exibir Editora Arqueiro}
FIM
```

Perceba que podemos utilizar o sinal de igual "=", que **não é considerado uma igualdade na programação**, mas sim um sinal de atribuição, para associar valores, já que esse operador tem a função de associar (atribuir) um valor a um identificador, um operando, isto é, uma variável.

A seguir, vamos nos aprofundar em alguns aspectos da declaração de variáveis. Preparado?

Agora, vamos tratar de três aspectos fundamentais ao desenvolvimento de um algoritmo. No entanto, antes de apresentá-los, vamos a uma questão.

Imagine que você esteja fazendo a codificação de um programa para uma escola. No início do programa, você declarou a variável MEDIA_FINAL como sendo do tipo literal.

Nesse contexto, será que essa variável poderá ser usada para fazer cálculos?

MEDIA_FINAL: LITERAL
OU
MEDIA_FINAL: REAL

- Sim.
- Não.

Comentário

A resposta é *não*!

Mesmo que a variável receba números no decorrer do processamento, não será possível fazer cálculos com ela. Isso ocorre porque, no início do código, foi dito que ela receberia letras e palavras, e não números. Números serão armazenados, mas serão considerados alfanuméricos, indisponíveis para cálculos matemáticos. Por exemplo: “23”, e não 23.

Dessa forma, desse exemplo, torna-se claro como é importante definir e declarar corretamente o tipo da variável em algoritmos e códigos de programação. Ao declarar uma variável de forma errada, geraremos um erro de programação que irá comprometer o programa como um todo. Dessa forma, toda a atenção é necessária! Anote essa dica!

Para cada variável criada no desenvolvimento de um algoritmo, precisamos pensar em três aspectos: nome, tipo e conteúdo. Você já ouviu falar deles?

Nome

O nome da variável deve ser escolhido com cuidado, de forma a dar pistas do tipo de informação que ela vai receber.

Nesse sentido, uma variável chamada MEDIA_FINAL indica que ela será usada para receber e armazenar o cálculo da média final de alunos.

Ao receber um conteúdo numérico com casas decimais, fica claro que essa variável deve ter um nome de fácil associação com sua utilização efetiva. Dessa forma, algumas possibilidades de nome são MEDIA_FINAL ou MEDIA_ALUNO.

Tipo

Como a nota pode não ser um valor inteiro, a variável MEDIA_FINAL deve ser declarada com o tipo *numérica real*.

Se ela for declara com o tipo *numérica inteira* e receber um valor **9,75**, o programa poderá dar erro.

Outro problema pode ser o processamento realizado de forma equivocada ao considerar apenas a parte inteira do número (9) na hora do cálculo.

Conteúdo

É necessário que esse conteúdo seja coerente com a natureza da informação que foi definida no Tipo.

Isso é fundamental para que o processamento aconteça corretamente.

As três definições devem ter coerência entre si para que o programa funcione corretamente e para que seu trabalho como desenvolvedor seja mais fácil.

Está clara a correlação entre **nome**, **tipo** e **conteúdo** ao definirmos as variáveis de um algoritmo? Vamos adiante!

Para finalizar este capítulo, cabe lembrar que as etapas de **estruturação do algoritmo** e de **teste de mesa** existem para que ajustes de definição de variáveis (nome e tipo), de processamento e de encadeamentos lógicos possam ser realizados.

O planejamento de uma solução computadorizada e o teste em situação real são fundamentais para que o algoritmo funcione como um verdadeiro projeto do programa.

Tais etapas servem para apurar o processo, sendo fundamentais para que a codificação de programas aconteça sem perda de tempo ou retrabalhos.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre este assunto. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Na empresa de TI em que você trabalha, o gerente solicitou que você analisasse um algoritmo criado por um novato para atender o pedido de um cliente (empresa de câmbio). A função desse algoritmo é a de converter Real em Dólar.

O algoritmo criado é:

```
Algoritmo Conversao_real_dolar
// rotina que converte um determinado valor em reais em dólares
VALOR_REAL, COTACAO_DOLAR, VALOR_DOLAR: real
VALOR_REAL = 0
COTACAO_DOLAR = 0
VALOR_DOLAR = 0
Início
Escreva "Programa para converter reais em dólares"
Escreva "Informe o valor disponível em reais (para comprar dólares): R$"
Leia VALOR_REAL
Escreva "Informe o valor de cotação do dólar do dia: R$"
VALOR_DOLAR = VALOR_REAL / COTACAO_DOLAR
Escreva "Com essa quantia será possível comprar: US$"
Escreva VALOR_DOLAR
Escreva "Boa viagem!"
Fim
```

Após a análise, você concluiu que o algoritmo apresenta erro na:

- ordenação dos comandos.
- escolha do Tipo das Variáveis.
- variável VALOR_REAL, que não deveria iniciar com o valor zero.
- variável COTACAO_DOLAR, que não recebe valor digitado pelo usuário.

Questão 2

Você precisa elaborar um pequeno manual para descrever a estagiários as nomenclaturas sobre algoritmos. Para tanto, você escolheu algumas definições.

Relacione cada uma das definições a seguir a seu respectivo termo, numerando a segunda coluna de acordo com a primeira.

TERMOS		DEFINIÇÕES
(1) CONSTANTE	☐	é todo valor fixo que não será alterado durante a execução de determinado programa.
(2) OPERADORES	☐	são referências que ocupam um espaço em memória e podem guardar um tipo determinado de dado/informação.
(3) PSEUDOCÓDIGO	☐	é a técnica de escrita que organiza os algoritmos em blocos e que faz uso de comandos para inserir os pontos iniciais.
(4) VARIÁVEIS	☐	são símbolos ou palavras reservadas por meio dos quais é possível fazer contas, comparar resultados e atribuir valores às variáveis e constantes.

Encerramento do Tópico e do Capítulo

Neste tópico, você aprendeu as funções, características e também as regras e os padrões necessários para escrever e compreender pseudocódigos. Também foram apresentadas informações importantes sobre os diferentes tipos de dados, o uso de variáveis, constantes e operadores, e a técnica do Teste de Mesa, tão importante para checar os encadeamentos lógicos de algoritmos e programas.

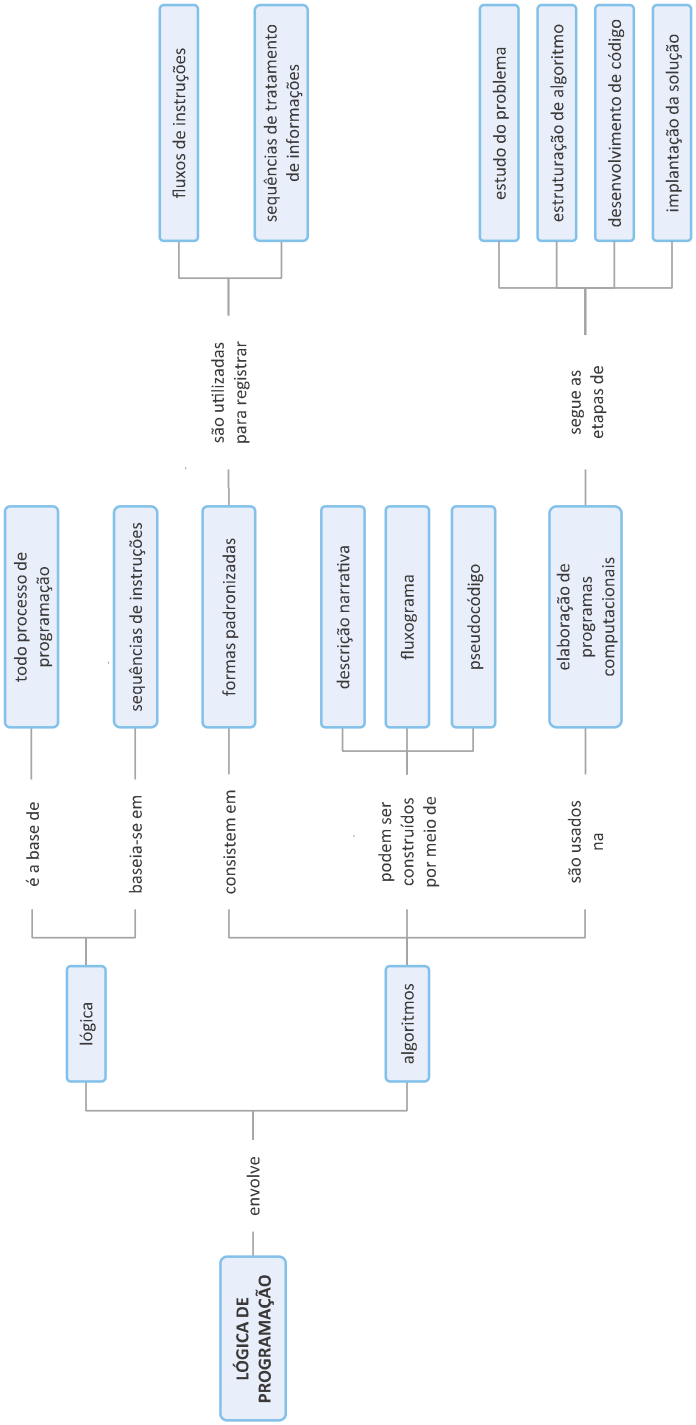
Agora que você compreende conceitos importantes de Lógica de Programação, já é capaz de construir e testar algoritmos. Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Você está prestes a concluir o capítulo.

No ambiente *on-line*, para facilitar seus estudos, disponibilizamos um pdf com um resumo dos conteúdos abordados neste capítulo, que também se encontra na próxima página.

Síntese

Observe, a seguir, o mapa conceitual que sintetiza o conteúdo que acabamos de trabalhar:



CAPÍTULO 2: CONTROLE DE FLUXO E ESTRUTURAS DE REPETIÇÃO

Tópico 1: Controle de Fluxo

Neste tópico, vamos apresentar o conteúdo que permite controlar o fluxo de um programa, dando mais complexidade ao código desenvolvido.

Conteúdos:

- Estruturas de controle: tipos e usos
  - Estrutura sequencial
  - Estrutura de seleção
    - Simples
    - Composta
    - Múltipla

Ao finalizar este tópico, você será capaz de:

- Utilizar controles de fluxo.
- Entender e construir pseudocódigos mais complexos.

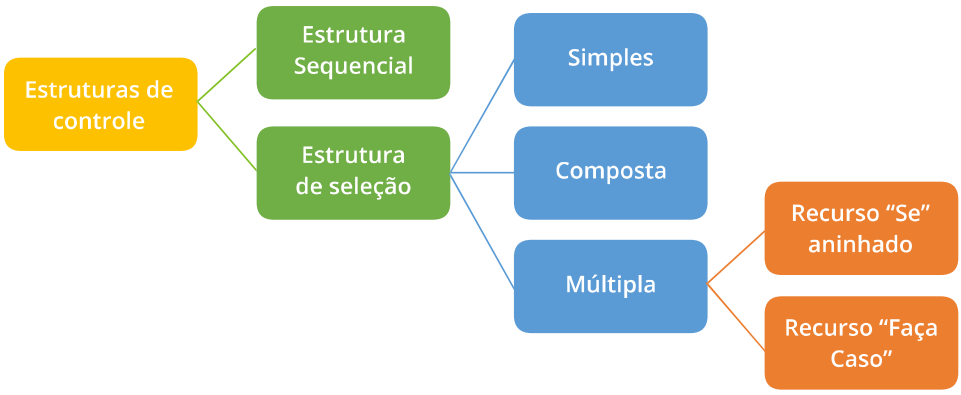
Estruturas de Controle

Você aprendeu muita coisa no capítulo 1, não é mesmo?

Compreendeu o que é um programa e como ele funciona, como armazenar dados na memória do computador usando variáveis e como realizar operações de diversos tipos utilizando operadores.

Agora, você estudará instruções que permitem controlar o fluxo de um programa a fim de dar um pouco mais de complexidade ao código. Essas estruturas de controle servem para **direcionar o fluxo do processamento dos dados**.

Para que você não fique com nenhuma dúvida, preparamos um esquema com os tipos de estrutura de controle.



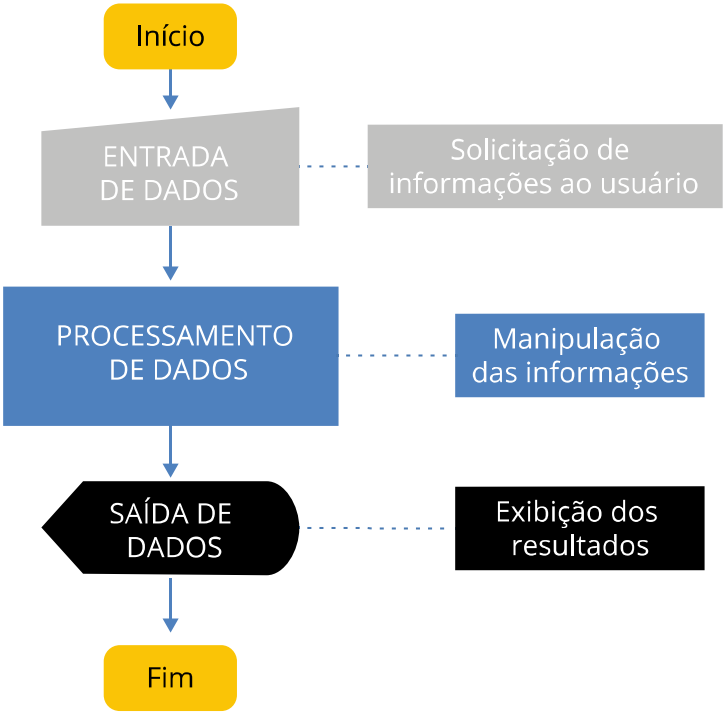
Estruturas Sequenciais

A estrutura sequencial é uma sequência de instruções que acontecem uma após a outra, sem desvios ou interrupções.

Esse tipo de estrutura é composto somente de linhas de comando sucessivas (que compõe um único fluxo possível de processamento) e está sempre limitado pelos marcadores INÍCIO e FIM do algoritmo ou de uma outra estrutura.

As estruturas sequenciais podem ser representadas por fluxogramas e pseudocódigos. Vamos ver exemplos?

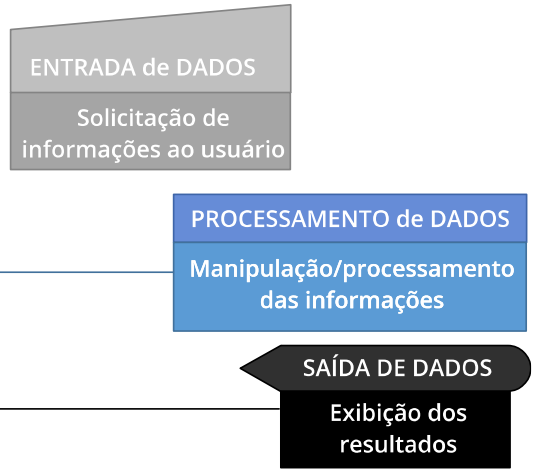
Fluxograma



Pseudocódigo

O algoritmo a seguir foi estruturado para ler o nome do aluno, a nota de duas avaliações e calcular a média final. Desse modo, ao final do processamento, a rotina deve mostrar o nome do aluno, as duas notas das avaliações e a média final calculada.

```
ALGORITMO CalculaMedia
  VAR
    N1, N2, Media: real
    Nome: literal
      N1 = 0
      N2 = 0
      Media = 0
  INICIO
    Escreva "Informe a primeira nota:"
    Leia N1
    Escreva "Informe a segunda nota:"
    Leia N2
    Escreva "Informe o nome do(a) aluno(a):"
    Leia Nome
    Media = (N1+N2)/2
    Escreva Nome, N1, N2, Media
  FIM
```



N1 = 0

Sempre que temos uma variável de tipo "numérico", colocamos, ao lado das variáveis, que elas são = 0. No exemplo, temos: N1 = 0, N2 = 0 e Média = 0, pois as três são variáveis numéricas e, no início da programação, ainda desconhecemos o seu valor.

Se já soubéssemos que N1 = 7, por exemplo, já poderia começar definido esse valor.

Comandos Escreva e Leia

São comandos dados ao computador. Não são para o usuário.

Tudo o que vem após o comando Escreva é o que vai aparecer na tela. O programador está informando ao computador que ele deve "escrever" tal coisa. Então, de acordo com o exemplo, ao receber o comando Escreva, o computador apresenta na tela a informação: "Informe a primeira nota".

O comando Leia é utilizado quando esperamos que haja um *input* (uma entrada) de alguma informação. O programador está informando ao computador que ele deverá *Ler* (capturar).

Em nosso exemplo, vamos colocar a N1 (Nota 1), a N2 (Nota 2) e o Nome para serem lidos e posteriormente processados pelo programa.

É por isso que o pseudocódigo termina com Escreva, pois é a fase de saída (*output*), isto é, o computador "escrevendo" (apresentando/exibindo) o resultado esperado.

Estruturas de Seleção

As estruturas de seleção são usadas para que sejam estabelecidos caminhos diferentes de instruções, a serem percorridos a partir de tomadas de decisão. Justamente por isso, esses recursos podem ser chamados de estruturas de seleção ou estruturas de decisão.

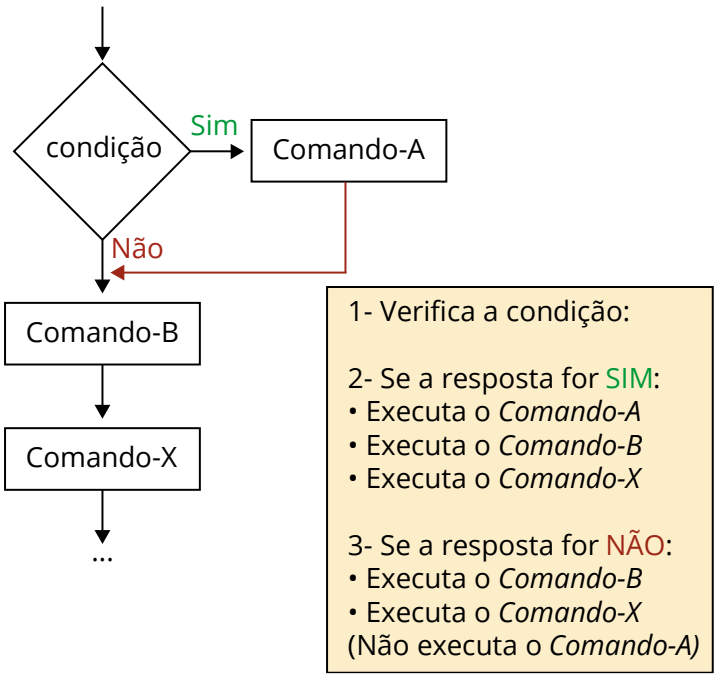
Como programador, você deverá utilizar os recursos de estruturas de seleção sempre que tiver de estruturar sequências de ações que poderão ser executadas ou não, a depender de um resultado frente a uma ou mais condições.

Existem três tipos de estruturas de seleção, cuja aplicação irá depender do contexto de utilização.

Simples

- **Marcadores**: SE, ENTÃO e FIM SE
- **Uso**: recurso a ser empregado em situações em que se faz necessário testar uma única condição/variável que, se verdadeira, irá desencadear a realização de um ou mais comandos. Temos, então, um teste e um grupo de ações que só acontecerão se a resposta for verdadeira.

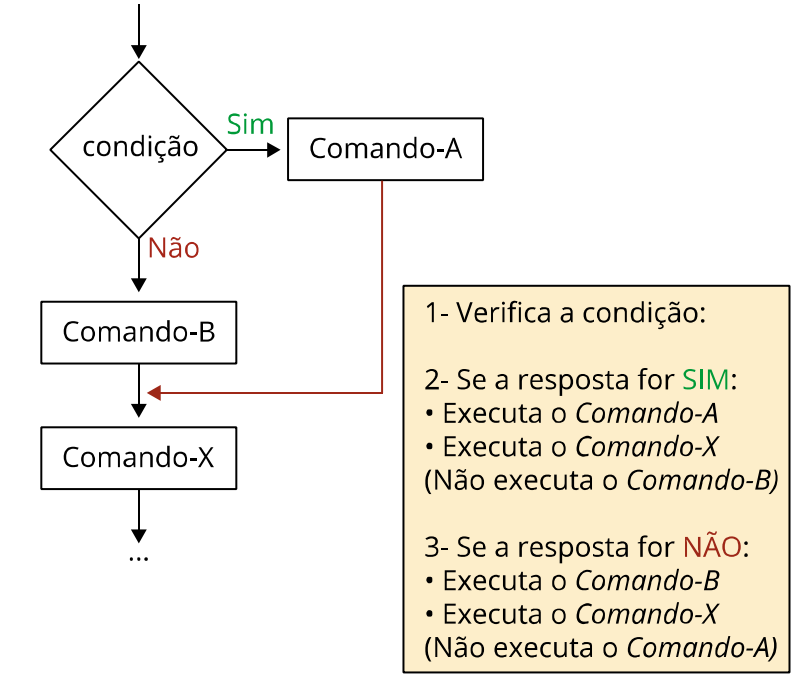
if (Decisão condicional simples)



Composta

- **Marcadores**: SE, ENTÃO, SENÃO e FIM SE
- **Uso**: recurso a ser empregado em situações em que se faz necessário testar uma única condição/variável que, se verdadeira, irá desencadear a realização de um ou mais comandos e que, se for falsa, irá desencadear outro grupo de ações. Temos então um teste e dois grupos de ações possíveis; um que acontecerá se a condição for verdadeira, e outro que acontecerá se a condição for falsa.

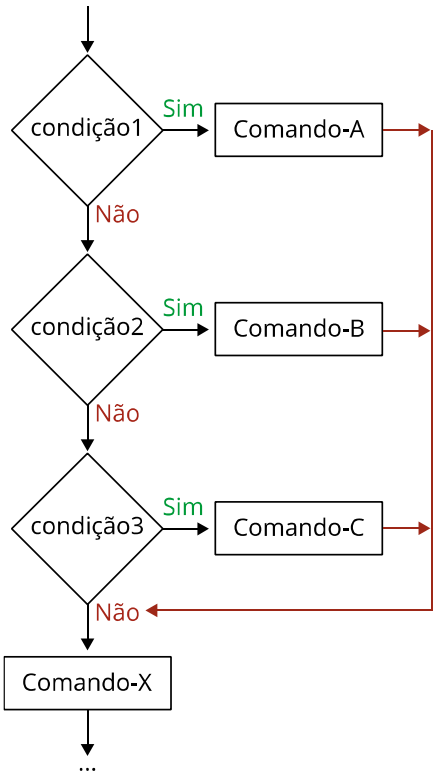
if-else (Decisão condicional composta)



Múltipla

Recursos a serem empregados em situações em que se faz necessário testar várias vezes a condição/variável. O resultado de cada teste irá desencadear um determinado grupo de ações.

- Utilizando recurso "Se" de forma encadeada (um dentro do outro)
  - Marcadores: SE, ENTÃO, SENÃO e FIM SE
- Utilizando recurso "Faça Caso"
  - Marcadores: FAÇA CASO, CASO, OUTRO CASO, FIM CASO.



1- Verifica a **CONDIÇÃO1**:

2- Se a resposta for SIM:
• Executa o *Comando-A*
• Executa o *Comando-X*
(Não executa o *Comando-B* e *Comando-C)*

3- Se a resposta for NÃO:

4- Verifica a **CONDIÇÃO2**:

5- Se a resposta for SIM:
• Executa o *Comando-B*
• Executa o *Comando-X*
(Não executa o *Comando-A* e *Comando-C)*

6- Se a resposta for NÃO:

7- Verifica a **CONDIÇÃO3**:

8- Se a resposta for SIM:
• Executa o *Comando-C*
• Executa o *Comando-X*
(Não executa o *Comando-B* e *Comando-A)*

9- Se a resposta for NÃO:
• Executa o *Comando-X*
(Não executa o *Comando-A, Comando-B* e *Comando-C)*
Finalização do Aninhamento

Estruturas de Seleção Simples

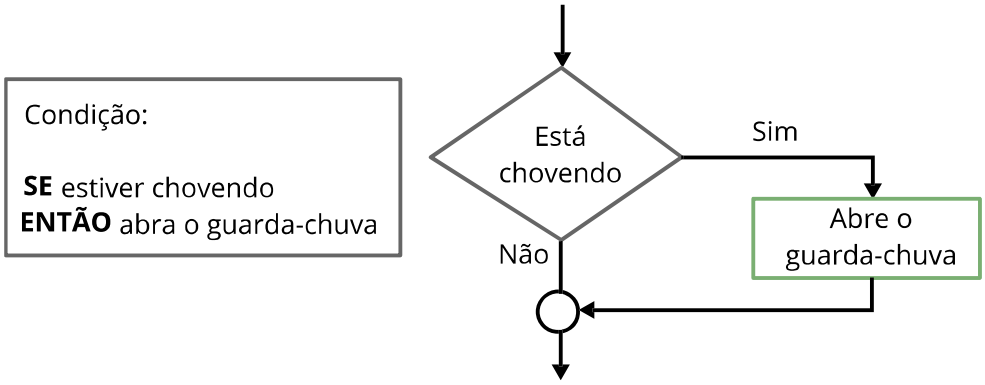
A estrutura de seleção simples faz uso da instrução **SE (IF)**. Ela é utilizada quando queremos testar uma condição antes de executarmos uma ou mais instruções.

Nessa estrutura, somente teremos uma ação **SE** o resultado da condição for **verdadeiro**. Desse modo, não há ação a ser executada caso o resultado seja falso.

SE CONDIÇÃO = VERDADEIRO → AÇÃO

SE CONDIÇÃO = FALSO → **NÃO HÁ AÇÃO A SER EXECUTADA**

Para compreender melhor a utilização desse tipo de estrutura de seleção, veja o fluxograma a seguir.



As palavras-chave (marcadores) de uma **estrutura de seleção simples** são:

```
SE <CONDIÇÃO> ENTÃO
    Comandos a serem executados somente se a condição for verdadeira
FIM SE
```

Veja um exemplo para entender melhor como funciona a estrutura de seleção simples.

Em nosso exemplo, vamos desenvolver um algoritmo em pseudocódigo para ler o nome de um aluno, a nota de duas avaliações e calcular a média final dele.

Ao final do processamento, a rotina deverá mostrar o nome do aluno, as duas notas das avaliações e a média final calculada. Se a média calculada do aluno for menor que seis (media < 6,0), o algoritmo deverá exibir a mensagem “Aluno(a) reprovado(a)”.

Vejamos um exemplo de estrutura de seleção simples:

Algoritmo

```
ALGORITMO SituacaoAluno
  VAR
  N1, N2, Media:real
  Nome: literal
    N1 = 0
    N2 = 0
    Media = 0
INICIO
  Escreva "Informe a primeira nota:"
  Leia N1
  Escreva "Informe a segunda nota:"
  Leia N2
  Escreva "Informe o nome do(a) aluno(a):"
  Leia Nome
  Media = (N1+N2) /2
  Escreva Nome, N1, N2, Media
  SE Media <6 ENTÃO
    Escreva "Aluno(a) reprovado(a)"
  FIM SE
FIM
```

Estrutura de seleção simples

Ação a executar caso a condição seja verdadeira

Estruturas de Seleção Composta

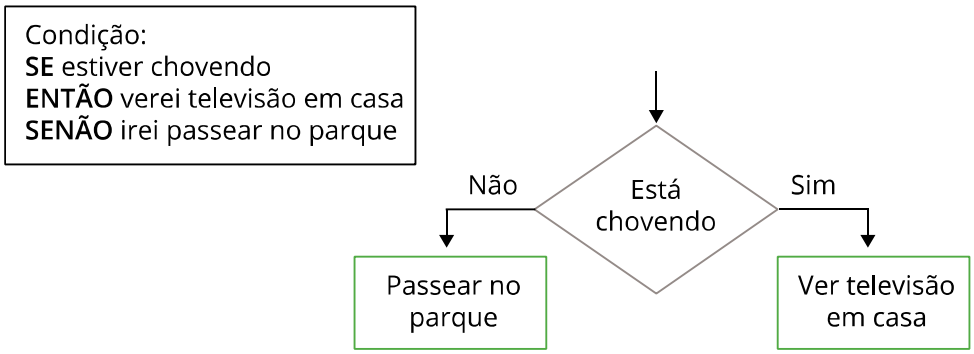
Na estrutura de seleção composta **existem dois caminhos diferentes predeterminados**.

Dessa maneira, haverá a execução de um comando ou grupo de comandos caso o resultado da condição seja **verdadeiro**, OU a execução de outro comando ou grupo de comandos diferente caso o resultado da condição seja **falso**.

SE CONDIÇÃO = VERDADEIRO → AÇÃO 1

SE CONDIÇÃO = FALSO → AÇÃO 2

Para compreender melhor a utilização desse tipo de estrutura de seleção composta, veja o fluxograma a seguir.



Agora, vamos conhecer as palavras-chave (marcadores) de uma **estrutura de seleção composta**:

```
SE <CONDIÇÃO> ENTÃO
    Comandos a serem executados somente se a condição for verdadeira
SENAO
    Comandos a serem executados somente se a condição for falsa
FIM SE
```

Vejamos um exemplo para entender como fica a utilização prática da estrutura de seleção composta.

Em nosso exemplo, vamos desenvolver um algoritmo em pseudocódigo para ler o nome de um aluno, a nota de duas avaliações e calcular a média final. Ao final do processamento, a rotina deverá mostrar o nome do aluno, as duas notas das avaliações e a média final calculada.

Se a média calculada do aluno for menor do que seis (media < 6,0), o algoritmo deverá exibir a mensagem “Aluno(a)reprovado(a)”. Caso contrário (media >= 6,0), o sistema deverá exibir a mensagem “Aluno(a)aprovado(a)”.

Vejamos um exemplo de estrutura de seleção composta:

Algoritmo

```
ALGORITMO SituacaoAluno
  VAR
  N1, N2, Media:real
  Nome: literal
    N1 = 0
    N2 = 0
    Media = 0
INICIO
    Escreva "Informe a primeira nota:"
    Leia N1
    Escreva "Informe a segunda nota:"
    Leia N2
    Escreva "Informe o nome do(a) aluno(a):"
    Leia Nome
    Media = (N1+N2) /2
    Escreva Nome, N1, N2, Media
    SE Media <6 ENTÃO
      Escreva"Aluno(a)reprovado(a)"
    SENÃO
      Escreva"Aluno(a)aprovado(a)"
    FIM SE
FIM
```

Estrutura de seleção composta

Ação a executar caso a condição seja verdadeira

Ação a executar caso a condição seja falsa

Estruturas de Seleção Múltiplas

As estruturas de seleção múltiplas são utilizadas em situações mais complexas. Vamos conhecer essas situações?

Quando precisamos fazer vários testes em uma mesma variável, temos de usar estruturas de seleção múltipla. Também usamos essa estrutura para testar o valor de uma variável que depende do teste do valor de outra variável.

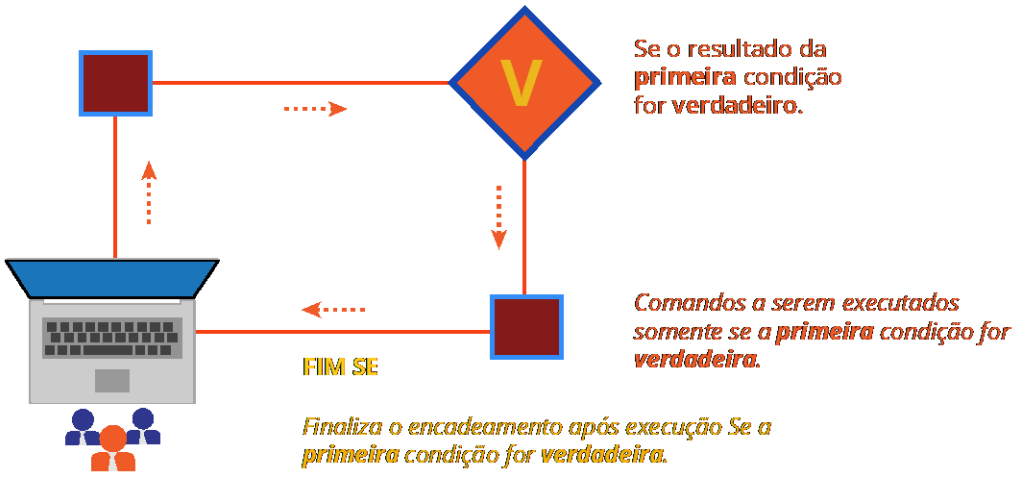
A estrutura de seleção múltipla pode ser construída de duas maneiras diferentes:

- Estruturas de seleção compostas encadeadas (aninhadas, agrupadas), ou seja, utilização de estruturas SE "uma dentro da outra".
- Estrutura FAÇA CASO.

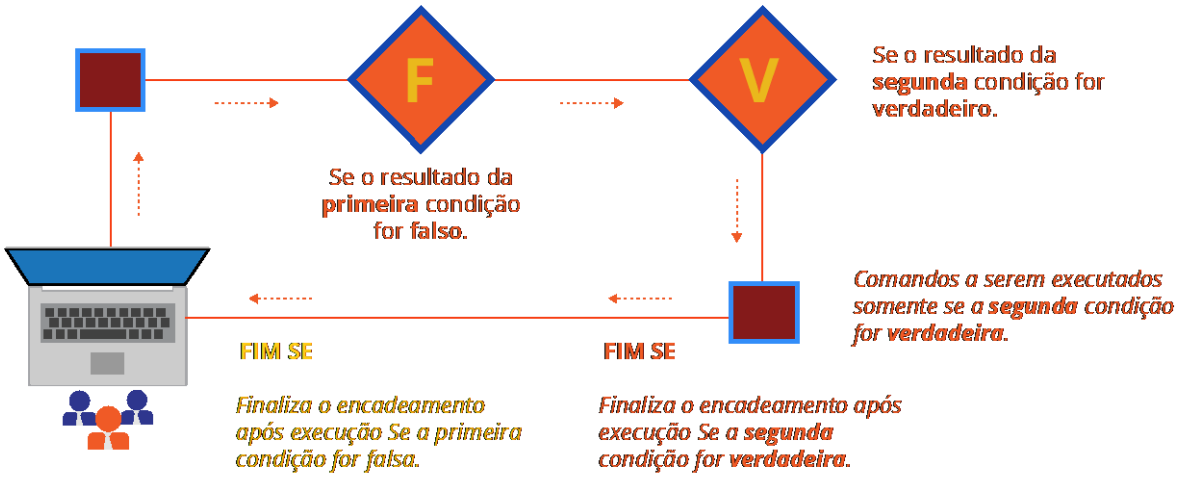
Vamos ver como funciona cada uma delas?

Estrutura de Seleção Múltipla Composta Encadeada

Vamos começar com as **estruturas de seleção múltipla compostas encadeadas**. Se utilizarmos essas estruturas, temos três casos de encadeamento diferentes todos representados nos infográficos a seguir.




```
SE <CONDIÇÃO> ENTÃO
    Comandos a serem executados somente se a primeira condição for verdadeira
  SENÃO
    SE <CONDIÇÃO>ENTÃO
        Comandos a serem executados somente se a segunda condição for verdadeira
    SENÃO
        Comandos a serem executados somente se a segunda condição for falsa
    FIM SE
FIM SE
```




```
SE <CONDIÇÃO> ENTÃO
    Comandos a serem executados somente se a primeira condição for verdadeira
SENÃO
    SE <CONDIÇÃO>ENTÃO
        Comandos a serem executados somente se a segunda condição for verdadeira
    SENÃO
        Comandos a serem executados somente se a segunda condição for falsa
    FIM SE
FIM SE
```


```
SE <CONDIÇÃO> ENTÃO
    Comandos a serem executados somente se a primeira condição for verdadeira
SENÃO
    SE <CONDIÇÃO>ENTÃO
        Comandos a serem executados somente se a segunda condição for verdadeira
    SENÃO
        Comandos a serem executados somente se a segunda condição for falsa
    FIM SE
FIM SE
```

Estrutura de Seleção Múltipla FAÇA CASO

Agora, vamos entender como funciona a estrutura de seleção múltipla com FAÇA CASO.

As palavras-chave (marcadores) dessa estrutura são:

```
FAÇA CASO
     CASO <CONDIÇÃO>:
          Comandos a serem executados somente se a condição for verdadeira
     CASO <CONDIÇÃO>:
          Comandos a serem executados somente se a condição for verdadeira
     CASO <CONDIÇÃO>:
          Comandos a serem executados somente se a condição for verdadeira
     OUTRO CASO:
          Comandos a serem executados se todos os casos mapeados retornarem
          "falso"

FIM CASO
```

A seguir, veremos um exemplo prático com base no que você aprendeu sobre as estruturas compostas encadeadas e a estrutura FAÇA CASO.

Exemplo

Neste exemplo, vamos usar uma solução com estrutura de **seleção composta encadeada** e outra com **estrutura FAÇA CASO**.

O algoritmo que vamos desenvolver deve ler o estado civil (EC) de uma pessoa como valor numérico. Os parâmetros são:

- "1" corresponde a Solteiro.
- "2" corresponde a Casado.
- "3" corresponde a Outros.

A seguir, temos as duas opções de desenvolvimento.

Estrutura de Seleção Composta Encadeada

```
ALGORITMO ESTADOCIVIL
  VAR
  EC: inteiro
  EC = 0
INICIO
  Escreva "Digite o estado civil (1 - solteiro, 2 - casado, 3 - outros)"
  Leia EC
  SE EC = 1 ENTAO
       Escreva "Solteiro"
       SENAO
          SE EC = 2 ENTAO
             Escreva "Casado"
             SENAO
                SE EC = 3 ENTAO
                Escreva "Outros"
                SENAO
                Escreva "Valor inválido"
                FIM SE
          FIM SE
  FIM SE
FIM
```

Estrutura FAÇA CASO

```
ALGORITMO ESTADOCIVIL
  VAR
  EC: inteiro
  EC = 0
INICIO
  Escreva "Digite o estado civil (1 - solteiro, 2 - casado, 3 -
  outros)"
  Leia EC
    FAÇA CASO
      CASO EC = 1
        Escreva "Solteiro"
      CASO EC = 2
        Escreva "Casado"
      CASO EC = 3
        Escreva "Outros"
      OUTRO_CASO
        Escreva "Valor inválido"
    FIM CASO
FIM
```

Analise as diferenças entre as estruturas de SELEÇÃO COMPOSTA e FAÇA CASO.

Estruturas de Seleção Múltipla na Prática: Qual Utilizar?

Vídeo

Vamos aprender um pouco mais? Acesse o ambiente *on-line* para assistir a um vídeo sobre **como a seleção múltipla pode ser aplicada na prática** e **como estruturar um algoritmo no desenvolvimento de um aplicativo**.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre este assunto. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Você está desenvolvendo um novo sistema para a escola em que trabalha como técnico de TI e necessita realizar o Teste de Mesa no algoritmo abaixo, levando em conta, para simulação, os valores iniciais que você deverá informar para N1=8,0 e N2=5,0:

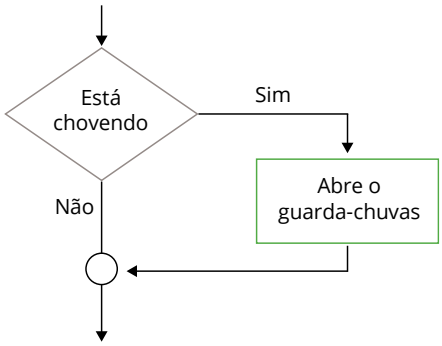
```
Algoritmo SituacaoAluno
VAR
     N1, N2, Media:real
N1 = 0
N2 = 0
Media = 0
INICIO
     Escreva "Informe a primeira nota:"
     Leia N1
     Escreva "Informe a segunda nota:"
     Leia N2
     Media = (N1+N2) /2
     Escreva Media
SE Media >=6 ENTAO
     Escreva "Aluno(a) aprovado(a)"
SENAO
     Escreva "Aluno(a) reprovado(a)"
FIM SE
FIM
```

Sabendo que N1 = 8,0 e N2 = 5,0, complete a setença com os **termos** que preenchem corretamente as lacunas a seguir e formam a mensagem que aparecerá impressa em Média:

_____ (RESULTADO ou MÉDIA) _____ (6.0 ou 6.5) e o aluno estará _____ (APROVADO ou REPROVADO).

Questão 2

No curso de TI, você está estudando as Estruturas de Seleção, e o professor pediu que o seguinte diagrama fosse analisado:



Marque **V** para verdadeiro e **F** para falso em relação às suas conclusões sobre o diagrama.

Conclusões	V	F
Trata-se de uma Estrutura de Seleção Simples.	○	○
Há uma alternativa a ser escolhida pelo algoritmo.	○	○
A ação proposta somente ocorrerá em uma hipótese.	○	○
O algoritmo será encerrado caso não esteja chovendo.	○	○

Encerramento do Tópico

Neste tópico, você aprendeu a controlar o fluxo de um algoritmo, além de ter entendido a importância de compreender e aplicar estruturas de controle de Fluxo em contextos de vida real.

Agora que você já sabe o conceito e a utilização prática das chamadas “Estruturas de seleção”, que podem ser simples, compostas e múltiplas, tornou-se capaz de elaborar pseudocódigos mais longos e complexos.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Vamos prosseguir em nossos estudos?

Siga para o próximo tópico!

Tópico 2: Estrutura de Repetição

As estruturas de repetição são necessárias, úteis e importantes no desenvolvimento de rotinas de programação. Neste tópico, vamos apresentar as condições que permitem a realização de rotinas repetitivas e fornecem mais sofisticação e inteligência ao seu código.

Conteúdos:

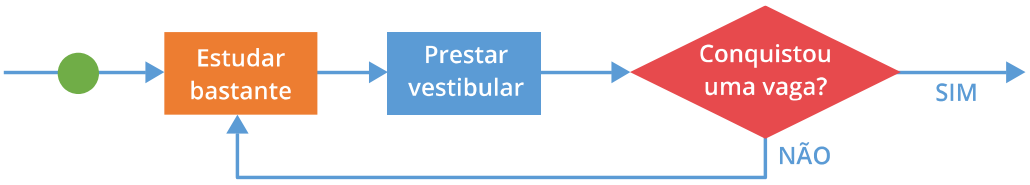
- Estruturas de repetição
- *Flag* de resposta (sinalização do usuário)
- *Flag* predeterminado (sinalização predeterminada).

Ao finalizar este tópico, você será capaz de:

- Otimizar códigos usando estruturas de repetição.
- Criar blocos de códigos usando a estrutura WHILE (FAÇA ENQUANTO).
- Usar a estrutura FOR NEXT (PARA O PRÓXIMO) para realizar tarefas repetitivas.

Em certos momentos, surgem circunstâncias específicas em programação que exigem que uma ação ou um grupo de ações seja realizado várias vezes.

Para resolver esses casos de forma lógica foram criadas as chamadas “Estruturas de Repetição”, que permitem criar blocos de repetição até que determinada condição se torne verdadeira ou por um número predeterminado de vezes. Vamos pensar em um exemplo?



Quem está se preparando para entrar na universidade, deve estudar até passar no vestibular, não é mesmo? Com isso, podemos dizer que os estudantes repetem **ações** (estudar bastante; prestar vestibular) até que uma **condição** (conquistou uma vaga?) seja conquistada.

Note que, no fluxograma, o bloco de ações “Estudar bastante” e “Prestar vestibulares” deve repetir-se por tempo indeterminado, até que a **condição** “Conquistar uma vaga” se **torne verdadeira**.

Este exemplo simples permite perceber que a **repetição** é algo que faz parte da vida cotidiana. Tanto é assim que ela é parte crucial em praticamente todos os processos produtivos.

Que tal pensarmos nos processos de repetição no contexto da informática?

Com o advento do computador, passou a ser possível automatizar a produção de artefatos que até então eram feitos manualmente. Com os avanços tecnológicos do século XXI, isso tornou-se cada vez mais frequente.

De modo geral, transformar algo em automático é prever a repetição de eventos durante determinado número de vezes, até que uma condição seja alcançada ou até que o usuário decida encerrar o processo. Por isso, a repetição é algo muito presente na programação de computadores.

Como as repetições são programadas no computador?

Para resolver situações envolvendo repetição de forma lógica e inteligente, foram criadas as chamadas **estruturas de repetição** ou **laços de repetição**.

As estruturas de repetição permitem programar o *loop*[3] (laço, ciclo) de blocos de instrução a partir de parâmetros previamente estabelecidos em algoritmos ou códigos de programação.

Veja a seguir como funcionam as estruturas de repetição.

Para o uso correto de estruturas de repetição, um item fundamental que deve ser compreendido é o controle do número de vezes que a estrutura será repetida. Desse modo, as repetições podem ser controladas a partir de alguns recursos.

Contador fixo

Recurso que utiliza uma variável *Contador*[4] para contar o número de vezes que uma determinada sequência será repetida. Quando o valor do contador atingir o número predeterminado, a repetição será encerrada.

```
Contador = Contador + 1[5]
```

***Flag* de resposta (sinalização do usuário)**

Recurso que, ao final do processamento, pergunta ao usuário se ele deseja ou não executar a rotina de novo, armazenando o resultado da resposta em um *flag* de resposta, isto é, uma variável, por exemplo *Resp*[6].

3. *Loop* – Conjunto de instruções que um programa de computador percorre e repete um significativo número de vezes, até que sejam alcançadas as condições desejadas.

4. Contador – Definição de uma variável qualquer para controlar o número de repetições de um *loop* (laço).

5. Contador = Contador + 1 – Significa somar e acumular mais um (1) ao valor que está armazenado na variável ***Contador***.

6. Resp = "Deseja continuar?" – Neste caso, a opção *"s"* ou *"n"* será armazenada na variável ***Resp***, que estará condicionada ao controle de repetição, para continuar o processamento (***Resp***="s") ou não (***Resp***="n").

Neste caso, o *loop* (ciclo ou laço) é interrompido pelo usuário.

Resp = “Deseja continuar <s/n>?”

***Flag* predeterminado (sinalização predeterminada)**

Recurso que possibilita ao usuário encerrar o *loop* quando quiser, digitando uma palavra predeterminada, ou quando se chegar a uma condição específica, determinada pelo valor de um campo específico.

"*Digite o nome do aluno ou FIM para sair:*"[7]

Para construir estruturas de repetição, podemos utilizar dois tipos de recursos diferentes.

7. "Digite o nome do aluno ou FIM para sair:" – Neste caso, o **campo específico** de controle do *loop* (laço de repetição) será **nome**.

Se o usuário digitar um valor diferente de "FIM" no campo **nome**, o processamento continuará, ou seja, executará a estrutura de repetição enquanto o valor "FIM" não for digitado.

Caso o usuário preencha o campo **nome** com "FIM", o processamento do laço de repetição será finalizado.

ENQUANTO...FAÇA (WHILE)

Serve tanto para repetições com contador fixo quanto para repetições por sinalizações (*flag*).

PARA PRÓXIMO (FOR NEXT)

Serve apenas para repetições por contador fixo.

Veremos, a seguir, como cada um desses recursos funciona!

Estrutura ENQUANTO...FAÇA (WHILE)

A instrução ENQUANTO...FAÇA (WHILE) indica o início de um *loop* e recebe uma condição como parâmetro. Tal condição é chamada de **condição de parada**, já que o *loop* é encerrado quando a condição retornar um valor lógico falso ou o *loop* nem é iniciado se a verificação condicional resultar em um valor lógico falso assim que a estrutura de repetição for iniciada.

O programa executa tudo que estiver contido entre os marcadores **ENQUANTO... FAÇA e FIM ENQUANTO.**

```
ENQUANTO<CONDIÇÃO>FAÇA
    Comandos a serem repetidos.
FIM ENQUANTO
```

Quando é encontrada a palavra FIM ENQUANTO, a rotina retorna ao início da estrutura e testa novamente a condição. Enquanto a condição for verdadeira, continua a repetição (*loop*).

Nessa estrutura, a rotina de repetição só é executada caso a condição seja verdadeira. Se a condição for falsa, a estrutura é ignorada e finalizada.

A seguir, veremos alguns exemplos da instrução ENQUANTO...FAÇA!

Saiba mais!

Para saber mais sobre as estruturas ENQUANTO...FAÇA (WHILE); FAÇA...ENQUANTO (DO...WHILE) ou REPITA...ATÉ QUE, indicamos a você uma videoaula gratuita na internet, em que o programador Gustavo Guanabara trata dessas estruturas de forma detalhada e fazendo uso de exemplos práticos.

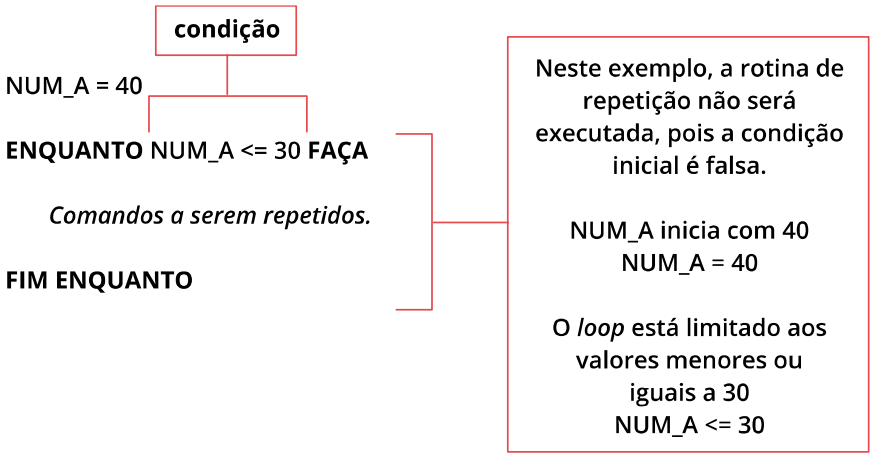
Aproveite essa dica!

Acesse **Estruturas de repetição - Parte 1. Disponível em:** https://youtu.be/U5PnCt58Q68?list=PLHz_AreHm4dmSj0MHol_aoNYCSGFqvfXV.

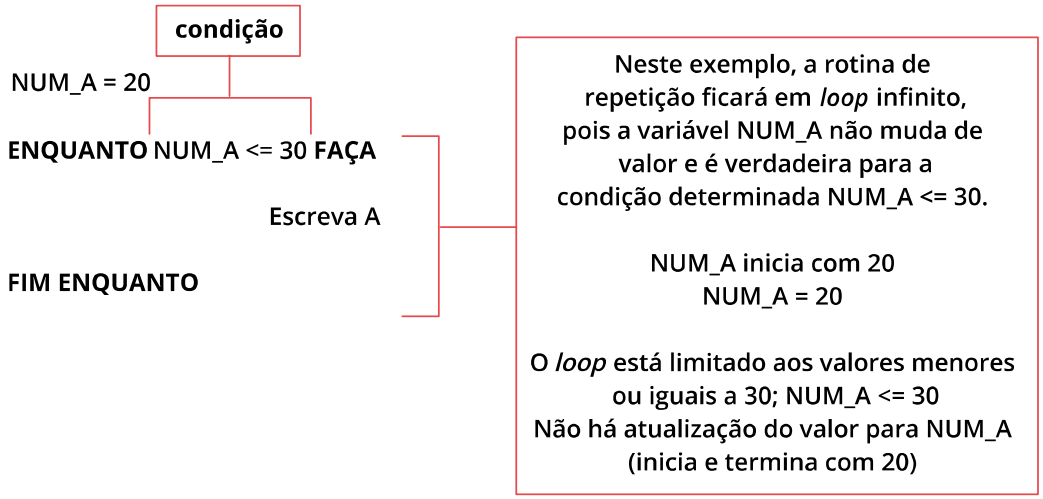
Exemplo

Acompanhe três condições diferentes em que a instrução ENQUANTO...FAÇA está representada.

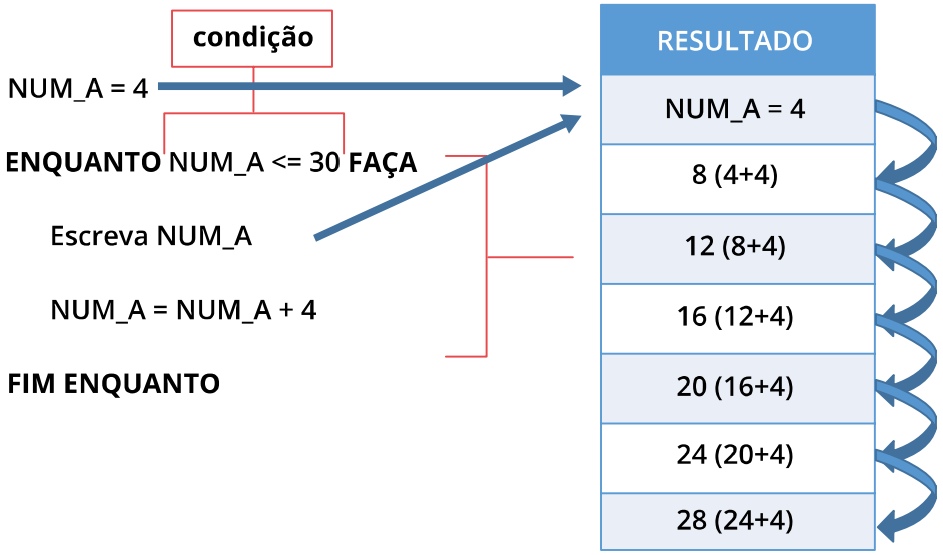
a) Se a condição da estrutura for falsa, temos o seguinte:



b) Se a variável não mudar de valor, temos o seguinte:



c) Se incrementarmos o valor de A, temos o seguinte:

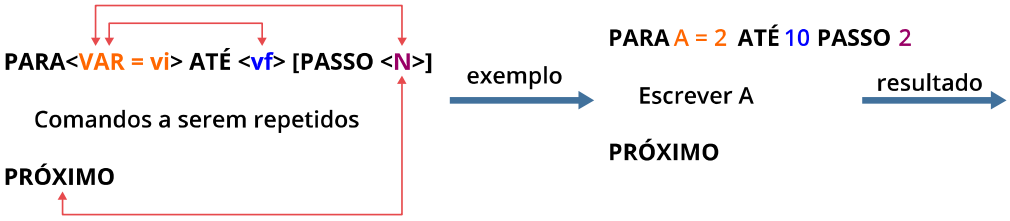


Áudio

Acesse o ambiente *on-line* para ouvir uma explicação sobre essa condição.

Estrutura PARA PRÓXIMO (FOR NEXT)

A estrutura PARA PRÓXIMO tem uma estrutura que lhe é particular. Vamos conhecê-la?



RESULTADO
2
4
6
8
10

Legenda: vi = valor inicial; vf = valor final (Neste exemplo, vi e vf minúsculas não são variáveis, são apenas valores). Para a variável A, note que foram atribuídos os valores vi = 2 (A=2); vf = 10 (Até 10).

Nesse tipo de construção, a rotina sempre entra na estrutura na primeira vez. Como temos a atribuição **VAR = vi** (valor inicial) no início da estrutura, tudo o que estiver dentro dela se repetirá até encontrar a palavra **PRÓXIMO**.

Quando encontra a palavra **PRÓXIMO, a rotina soma o valor do passo (somar 2) à variável de controle (A)** e realiza um teste com o valor final (vf). Enquanto o valor da variável de controle não atender a condição determinada para o valor final (vf), a rotina continuará se repetindo.

Estruturas de Repetição

Para entendermos melhor o uso das estruturas de repetição, vamos acompanhar três exemplos em que desenvolvemos um algoritmo para cada uma das estruturas que acabamos de estudar.

Exemplo 1 – contador fixo

Vamos construir um algoritmo em pseudocódigo para ler o nome e duas notas de cada aluno de uma turma de 15 alunos. Além disso, precisaremos calcular a média, bem como mostrar o valor das duas notas e a média de cada aluno. Vejamos duas formas estruturais de obtermos a solução:

Estrutura ENQUANTO FAÇA

```
ALGORITMO CalculaMedia
VAR
    Nome: literal
    N1, N2, Media, Aluno: inteiro
    N1 = 0
    N2 = 0
    Media = 0
    Aluno = 0
INICIO
        ENQUANTO Aluno < = 15 FACA
        Escreva"Digite o nome N1, N2"
            Leia Nome
            Leia N1
            Leia N2
                Media = (N1+N2) /2
                Escreva Nome, Media
        Aluno = Aluno + 1
        FIM ENQUANTO
FIM SE
```

Início da estrutura de repetição.
Neste exemplo, o número de repetições foi predeterminado (15).

Instrução que incrementa o número um (1) à variável Aluno, para controlar a quantidade de vezes que o bloco será repetido.

Estrutura PARA PRÓXIMO

```
ALGORITMO CalculaMedia
VAR
    Nome: literal
    N1, N2, Media, Aluno: inteiro
    N1 = 0
    N2 = 0
    Media = 0
    Aluno = 0
INICIO
PARA Aluno = 1 ATÉ 15 PASSO 1
    Escreva "Digite o nome N1, N2"
    Leia Nome
    Leia N1
    Leia N2
    Media = (N1+N2) /2
    Escreva Nome, N1, N2, Media
PRÓXIMO
FIM
```

valor inicial (vi)

valor final (vf)

valor a ser incrementado (n)

Bloco de instruções que será repetido do valor inicial (vi) até o valor final (vf). Neste exemplo, o bloco será repetido 15 vezes.

Definir a instrução como PASSO 1 significa que o valor "um" deve ser adicionado à variável Aluno, que, neste exemplo, é a variável que controla o número de repetições.

Analise as diferenças entre a estrutura ENQUANTO FAÇA e a PARA PRÓXIMO.

Exemplo 2 – *flag* de resposta

Vamos construir um algoritmo em pseudocódigo para ler o nome e duas notas de diversos alunos. Além disso, é preciso calcular a média, bem como mostrar o nome, as duas notas e a média de cada aluno. Após processar cada aluno, deve-se perguntar se o usuário deseja continuar.

Quando o usuário responder "sim" (**S**), o programa continuará. Quando a resposta for "não" (**N**), o programa terminará.

Algoritmo – Exemplo 2 – *Flag* de resposta

```
ALGORITMO CalculaMedia
VAR
    Nome, capturaFlag: literal
    N1, N2, Media: inteiro
    N1 = 0
    N2 = 0
    Media = 0
    CapturaFlag = "S"
INICIO
        ENQUANTO CapturaFlag = "S" FAÇA
            Escreva "Digite o Nome, N1, N2"
            Leia Nome
            Leia N1
            Leia N2
            Media = (N1 + N2) / 2
            Escreva Nome, N1, N2, Media
            Escreva "Deseja continuar (S/N)?"
            Leia CapturaFlag
        FIM ENQUANTO
FIM
```

Teste da variável CapturaFlag para checar o valor inserido pelo usuário ("S" ou "N").

Exibição da mensagem e leitura da sinalização do usuário (*flag* de resposta).

Exemplo 3 – *flag* de resposta

Vamos construir um algoritmo em pseudocódigo para ler o nome e duas notas de uma turma de alunos. Além disso, precisaremos calcular a média, bem como mostrar o nome, as duas notas e a média de cada aluno.

O programa vai finalizar quando o usuário digitar a palavra **FIM** como conteúdo da variável Nome, em vez de um **nome** válido.

Desse modo, predeterminamos que o programa irá terminar quando o usuário digitar a palavra **FIM** no lugar do nome de um aluno.

Algoritmo – Exemplo 3 – *Flag* predeterminado

```
PROGRAMA CalculaMedia
VAR
    Nome: literal
    N1, N2, Media: inteiro
    N1 = 0
    N2 = 0
    Media = 0
INICIO
        Escreva "Digite o nome ou FIM para sair"
        Ler Nome
        ENQUANTO Nome <>"FIM"FAÇA
        SE Nome <>"FIM"
              Escreva "Digite N1, N2"
              Leia N1
              Leia N2
              Media = (N1 + N2) / 2
              Escreva Nome, N1, N2, Media
              Escreva"Digite o nome ou FIM para sair"
              Ler Nome
        FIM SE
        FIM ENQUANTO
FIM
```

Teste da variável Nome para checar se foi inserido um valor válido ou o termo FIM (*flag* predeterminado).

Exibição da mensagem e leitura da variável Nome.

Saiba mais!

Às vezes, fica difícil decidir entre duas estruturas com a mesma finalidade, como é o caso do ENQUANTO FAÇA e do PARA PRÓXIMO, não é mesmo?

Pensando nisso, indicamos a você uma videoaula gratuita na internet, em que o programador Gustavo Guanabara trata dessas duas estruturas de forma detalhada e fazendo uso de exemplos práticos.

Aproveite essa dica!

Acesse o vídeo **Estruturas de repetição - Parte 3**.

Disponível em: https://youtu.be/WJQz20i7CyI. Acesso em: 29 set. 2016.

Dicas Digitais

Esse conteúdo está quase acabando, mas antes que isso aconteça, é oportuno indicar recursos que possibilitem a você ampliar seus conhecimentos sobre lógica de programação.

Foi pensando nisso que indicamos, a seguir, dois cursos gratuitos organizados em videoaulas disponíveis na internet. Aproveite essas dicas!

Curso de Lógica de Programação da Rbtech

Série sobre lógica de programação composta de sete videoaulas gratuitas. Nelas o programador Ricardo Bernardi explica, didaticamente, os fundamentos da lógica no contexto da programação e também como ler, analisar, interpretar e desenvolver algoritmos em diferentes formatos.

Acesse: http://dev.rbtech.info/curso-logica-programacao-aula-01/.

Curso de Lógica de Programação de Cursos em Vídeo

Série de 15 aulas que ensina tudo sobre Algoritmos e Lógica de Programação. Nelas o programador Gustavo Guanabara, com mais de 20 anos de experiência, mostra conceitos, exemplos e aplicações práticas dos principais assuntos ligados ao tema.

Acesse:https://www.youtube.com/playlist?list=PLHz_AreHm4dmSj0MHol_aoNYCSGFqvfXV.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre este assunto. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Na empresa de TI em que trabalha, seu supervisor pediu que você examinasse o trecho de código apresentado a seguir, utilizando a instrução ENQUANTO...FAÇA (*while*):

```
NUM_A = 0
ENQUANTO NUM_A<20 FAÇA
    NUM_A = NUM_A + 4
    Escreva NUM_A
FIM ENQUANTO
```

Analisando o trecho, você verificou que:

- o trecho ficará em *loop* infinito.
- a repetição não será executada.
- o resultado será: 4, 8, 12, 16, 20.
- a variável NUM_A sempre terá o resultado 0.

Questão 2

Sabendo que você estuda TI, um professor amigo seu pediu que analisasse o seguinte algoritmo, desenvolvido para calcular a média dos 15 alunos de uma turma:

```
ALGORITMO CalculaMedia
VAR
        Nome: literal
        N1, N2, Media, Contador: inteiro
        N1 = 0
        N2 = 0
        Media = 0
        Contador = 0
INICIO
        ENQUANTO Contador < 15 FACA
            Escrever "Digite o nome N1, N2"
            Ler Nome
            Ler N1
            Ler N2
            Media = (N1+N2) /2
            Escrever Nome, Media
            Contador = Contador + 1
        FIM ENQUANTO
FIM
```

Após essa análise, você conclui que:

- serão impressas as médias de 14 alunos.
- há um erro na escrita do algoritmo e não irá funcionar.
- a variável *Contador* permite a impressão das médias de 15 alunos.
- os 16 alunos terão suas médias impressas com seus respectivos nomes.

Encerramento do Tópico e do Capítulo

Neste tópico, você aprendeu sobre o conceito, as formas de uso e as aplicações práticas das estruturas de repetição.

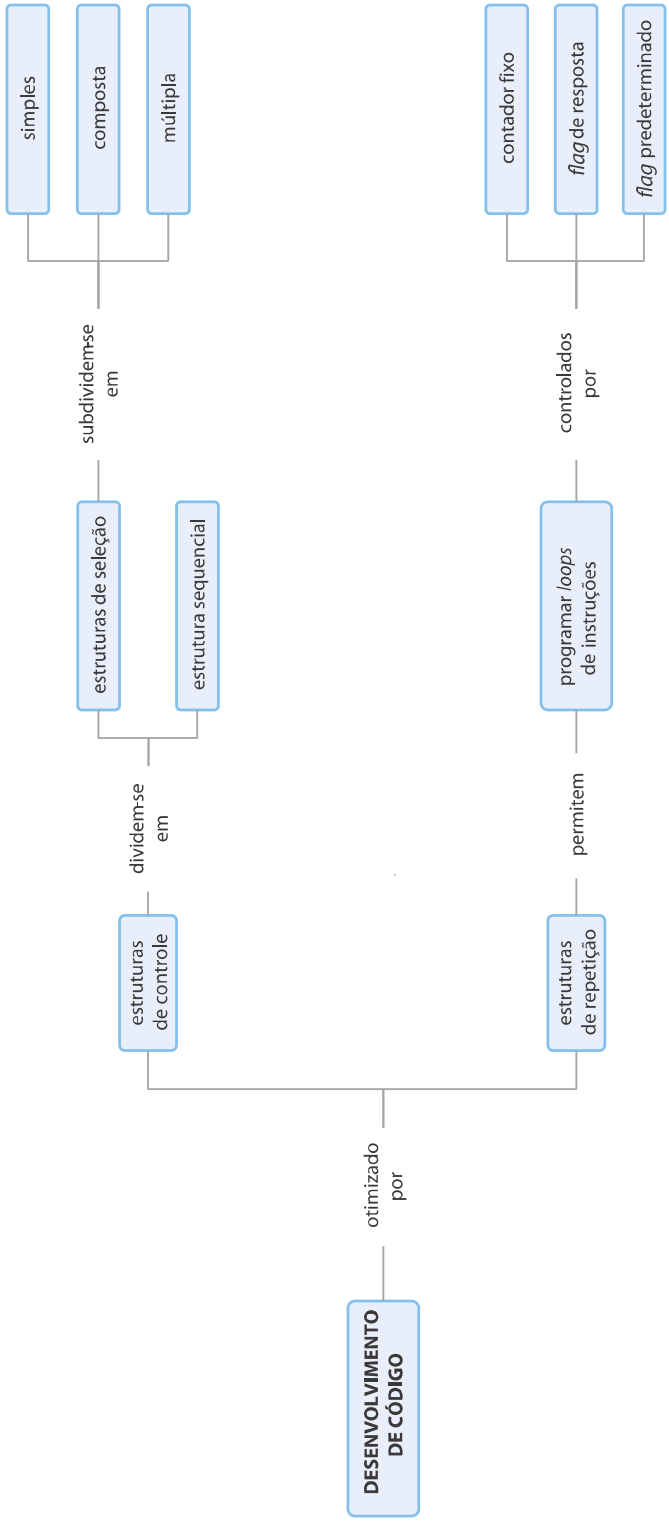
Agora que você já sabe utilizar as estruturas FAÇA ENQUANTO e PARA PRÓXIMO, poderá otimizar seus algoritmos e futuros códigos criando laços de repetição predeterminados ou encerrados por meio de sinalização do usuário. Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Você está prestes a concluir o capítulo.

No ambiente *on-line*, para facilitar seus estudos, disponibilizamos um pdf com um resumo dos conteúdos abordados neste capítulo, que também se encontra na próxima página.

Síntese

Observe, a seguir, o mapa conceitual que sintetiza o conteúdo que acabamos de trabalhar:



Jogo didático

Acesse, no ambiente *on-line*, um jogo didático sobre o assunto deste tópico.

LEITURA RECOMENDADA

Amplie seu conhecimento com algumas leituras.

ALMEIDA, Marilane. *Curso essencial de lógica de programação*. [S.l.]: Brasil Universo dos Livros. 112 p. (edição digital – B009SK3R4O).

LÓGICA de programação - Aula 02 – Tipos de algoritmos. Realização de Ricardo Bernardi. 2015. (8 min.), son., color. Série Lógica de Programação. Disponível em: https://www.youtube.com/watch?v=JLlTo3SwxJE&index=2&list=PLInBAd9OZCzxxk0VvMGrq7l-ZMu5lOSwC. Acesso em: 9 nov. 2015.

MORAES, Paulo Sérgio de. *Lógica de programação*. 2000. São Paulo: Unicamp – Centro de Computação – DSC, 2000. Disponível em: http://www.inf.ufsc.br/~vania/teaching/ine5231/Logica.pdf. Acesso em: 6 nov. 2015.

OLIVEIRA, Luiz Affonso Henderson Guedes de; ROCHA, Kliger Kissinger F.; BITTENCOURT, Valnaide Gomes. *Algoritmos*. Natal: Universidade Federal do Rio Grande do Norte – Centro de Tecnologia – Departamento de Computação e Automação, 2004. Disponível em: http://www.dca.ufrn.br/~lmarcos/courses/DCA800/pdf/algoritmos_apres_parte1.pdf. Acesso em: 19 nov. 2015.

SAID, Ricardo. *Curso de lógica de programação*. [S.l.]: Brasil: Universo dos Livros, 2014. 163 p. (edição digital –9788579304286).

SENAC-RS (Rio Grande do Sul). *Lógica de programação*. Porto Alegre: Senac, 2003. 68 p.

REFERÊNCIAS

CARVALHO, Flávia Pereira de. *Apostila de lógica de programação*: algoritmos. Disponível em: <https://fit.faccat.br/~fpereira/apostilas/apostila_algoritmos_mar2007.pdf>. Acesso em: 6 nov. 2015.

ESTRUTURA de Repetição. Disponível em: <http://www.ufpa.br/sampaio/curso_de_icc/icc/aula%2011/preliminares.htm>. Acesso em: 5 out. 2014.

HOUAISS, Antônio; VILLAR, Mauro de Salles; MELLO, Francisco Manoel de. *Dicionário Houaiss da LínguaPortuguesa*. Rio de Janeiro: Objetiva, 2009.

LINDER, Marcelo. *Conceito básico de algoritmo*. Disponível em: <http://www.univasf.edu.br/~marcelo.linder/arquivos_iaa/aulas/aula3.pdf>. Acesso em: 5 out. 2014.>

LÓGICA de programação – Aula 2 – Tipos de algoritmos. Realização de Ricardo Bernardi. 2015. (8 min.), son., color. Série Lógica de Programação. Disponível em: <https://www.youtube.com/watch?v=JLlTo3SwxJE&index=2&list=PLInBAd9OZCzxxk0VvMGrq7lZMu5lOSwC>. Acesso em: 9 nov. 2015.

LÓGICA de programação – Aula 1 – Introdução. Realização de Ricardo Bernardi. 2015.
(8 min.), son., color. Série Lógica de Programação. Disponível em: <https://www.youtube.com/watch?v=Ds1n6aHchRU>. Acesso em: 9 nov. 2015.

LÓGICA de Programação – Aula 1 – Realização de Pedro Messias L. Nascimento. Avaré - São Paulo: ProDigio, 2013. (29 min.), son., color. Série Lógica de Programação. Disponível em: <https://www.youtube.com/watch?v=ODEWdvms8NE&list=PL7Gd-vyL4-JhoaJobhk211cT4TncP8Cei&index=3>. Acesso em: 6 nov. 2015.

MORAES, Paulo Sérgio de. *Lógica de programação*. Disponível em: <http://www.inf.ufsc.br/~vania/teaching/ine5231/Logica.pdf>. Acesso em: 6 nov. 2015.

NOEL, André. *O código faz o que você manda*. Disponível em: <http://vidadeprogramador.com.br/2013/11/29/o-codigo-faz-o-que-voce-manda/>. Acesso em: 19 nov. 2015.

OLIVEIRA, L. A. H. G. de; ROCHA, K. K. F.; BITTENCOURT, V. G. *Algoritmos*. Disponível em: <http://www.dca.ufrn.br/~lmarcos/courses/DCA800/pdf/algoritmos_apres_parte1.pdf>. Acesso em: 19 nov. 2015.

SAID, Ricardo. *Curso de lógica de programação*. [S.l.]: Brasil: Universo dos Livros, 2014. 163 p. (edição digital - 9788579304286).

SENAC-RS. *Lógica de programação*. Disponível em: <https://cld.pt/dl/download/e1946c4f-499c-4961-b63a-179c2fb3526e/Livros Linux/Programação/SENAC - Lógica de Programação.pdf>. Acesso em: 23 nov. 2015.

SHUTTERSTOCK. Disponível em: <http://www.shutterstock.com/>. Acesso em: 29 jan. 2016.

TI VERDE: Sustentabilidade e eficiência. Disponível em: <http://homologlx.addtech.com.br/cpqd/midia eventos/fatos/fatos-157/ti-verde-sustentabilidade-e-eficiencia>. Acesso em: 2 set. 2014.

WIKIBOOKS. *Introdução à programação*. Disponível em: <pt.wikibooks.org/wiki/Introdução_à_ programação/Definições_sobre_Lógica_de_Programação>. Acesso em: 5 out. 2014.

ENCERRAMENTO

Parabéns!

Você chegou ao final do conteúdo de **Lógica de Programação**.

Lembre-se de que, para ser um profissional de destaque, você precisa estar sempre se aperfeiçoando. Então, não pare por aqui, mantenha-se atualizado e faça de todas as suas experiências profissionais um caso de sucesso.

https://fundacao.bradesco/