

ALGORITMO E LÓGICA DE PROGRAMAÇÃO

Carlos Veríssimo

```
real
'/\\

80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

return array(
    'code' => $captcha_config['code'],
    'image_src' => $image_src
);
}

if( !function_exists('hex2rgb') ) {
    function hex2rgb($hex_str, $return_string = false) {
        $hex_str = preg_replace("/[^0-9A-Fa-f]/", "", $hex_str);
        $rgb_array = array();
        if( strlen($hex_str) == 6 ) {
            $color_val = hexdec($hex_str);
            $rgb_array['r'] = 0xFF & ($color_val >> 16);
            $rgb_array['g'] = 0xFF & ($color_val >> 8);
            $rgb_array['b'] = 0xFF & $color_val;
        } elseif( strlen($hex_str) == 3 ) {
            $rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));
            $rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));
            $rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));
        } else {
            return false;
        }
        return $rgb_array;
    }
}

return $return_string ? implode($separator, $rgb_array) : $rgb_array;
}

// Draw the image
if( isset($image_src) ) {
    $image_src = $image_src;
}
```

Foto: Pixabay

E-book 1

FAM
ONLINE

Neste E-Book:

Introdução	4
Conceito de Lógica.....	5
Necessidade do uso da lógica	5
Aplicando a Lógica na Construção de Programas	6
Algoritmo.....	8
Descrição Narrativa	9
Fluxograma Convencional.....	9
Pseudocódigo	13
Introdução à Lógica	14
Princípio da Resolução de Problemas.....	14
Paradigmas de diagramação	18
Paradigma da Programação Estruturada.....	18
Estrutura SWITCH	26
Português Estruturado	29
Utilizando uma Pseudolinguagem	29
Elementos da Pseudolinguagem	33
Procedimento e funções	35
Constantes e Variáveis.....	37
Constantes	37
Variáveis	38
Tipos de Dados Primitivos	40

Dados Primitivos do Tipo Numérico	40
Dados Primitivos do Tipo Caractere	40
Dados Primitivos do Tipo Alfanumérico.....	41
Dados Primitivos do Tipo Lógico	41
Operadores.....	42
Operadores Aritméticos.....	42
Operadores Relacionais.....	43
Operadores Lógicos.....	43
Praticando com Pseudocódigo	45
Teste de Mesa.....	50
Considerações Finais.....	56
Síntese	57

INTRODUÇÃO

Nesta unidade, você terá contato com os conceitos que são fundamentais para desempenhar, de forma satisfatória, a lógica de programação. Aqui você entenderá o conceito de lógica e a importância de sua aplicação no desenvolvimento de programas de computadores. Você saberá o que é um algoritmo e o diagrama de blocos, que são recursos muito úteis na produção de um programa de computador. Você entenderá os princípios que compõem a resolução de problemas, bem como saberá identificar as técnicas de programação. Para finalizar, será introduzido o conceito de pseudolinguagem, adotada neste curso, para o desenvolvimento dos algoritmos. Por fim, apresentaremos a técnica de teste de mesa.

Espero que você tenha um bom proveito desta instigante disciplina, e que tenha um imenso prazer na arte de programar computadores.

CONCEITO DE LÓGICA

Para usar a lógica, é necessário ter domínio sobre o pensamento, bem como saber pensar, ou seja, dominar a “Arte de Pensar”. Alguns definem o raciocínio lógico como um conjunto de estudos que visa a determinar os processos intelectuais, que são as condições gerais do conhecimento verdadeiro. Isso é válido para a tradição filosófica clássica “aristotélico-tomista”.

Formalmente, podemos definir **lógica** como sendo a ciência que estuda as leis e critérios de validade que regem o pensamento e a demonstração, ou seja, **ciência dos princípios formais do raciocínio**. Definimos **lógica de programação** como sendo a técnica de encadear pensamentos para atingir determinado objetivo.

Necessidade do uso da lógica

Usar a lógica é um fator a ser considerado por todos, principalmente pelos profissionais da área da Tecnologia de Informação, pois seu dia a dia dentro das organizações é solucionar problemas e atingir os objetivos apresentados por seus usuários com eficiência e eficácia, utilizando recursos. Hoje é uma realidade: uma quantidade infinita de coisas com as

quais estamos direta e indiretamente em contato são controladas por softwares – de simples calculadoras a satélites.

Neste cenário, saber lidar com problemas de ordem administrativa, de controle, de planejamento e de estratégia requer atenção e boa performance de conhecimento de nosso raciocínio. É muito importante você saber que ninguém ensina ninguém a pensar. O nosso objetivo é proporcionar o seu desenvolvimento nesta técnica. Um ponto ao qual você deve estar muito atento: somente a prática constante traz desenvoltura no desenvolvimento de algoritmos.

FIQUE ATENTO

Você deve ser persistente e praticar constantemente, pois, somente assim você logrará sucesso!

Aplicando a Lógica na Construção de Programas

Programadores eficientes devem preparar um programa iniciando com um **diagrama de blocos** para demonstrar sua linha de raciocínio lógico. Esse diagrama, também denominado por alguns de **fluxograma**, estabelece a sequência de operações a se efetuar em um programa. Dando seguimento, o programador deve preparar o algoritmo e, por fim, o

programador deve testar seu algoritmo, aplicando técnicas de validação.

Este conjunto de técnicas permite uma posterior codificação **em qualquer linguagem** de programação de computadores, pois na elaboração do diagrama de blocos não se atinge um detalhamento de instruções ou comandos específicos, os quais caracterizam uma linguagem.

A técnica de estruturar um programa permite ao programador i) agilizar a codificação da escrita da programação; ii) facilitar a depuração da sua leitura; iii) identificação de possíveis falhas apresentadas pelos programas; iv) facilitar as alterações e atualizações dos programas.

ALGORITMO

Intuitivamente, podemos definir algoritmo com sendo uma sequência finita de instruções, as quais podem ser realizadas mecanicamente, em um tempo finito. Algoritmo é a especificação da sequência ordenada de passos que deve ser seguida para a realização de uma tarefa, garantindo a sua repetição.

Na realidade, a palavra algoritmo vem do nome do matemático iraniano Abu Abdullah Mohammed Ibn Musa al-Khawarizmi (século 17), considerado o fundador da álgebra. O termo algarismo também é utilizado em outras áreas do conhecimento, como engenharia, administração, entre outras. Então, por que a palavra algoritmo ficou tão associada à computação? Para compreender os motivos, é preciso entender, mesmo que superficialmente o funcionamento dos computadores.

Dentre as formas de representação de algoritmos mais conhecidas podemos citar:

- Descrição Narrativa;
- Fluxograma Convencional;
- Pseudocódigo, também conhecido como Linguagem Estruturada ou Portugol.

Descrição Narrativa

Nesta forma de representação, os algoritmos são expressos diretamente em linguagem natural, ou seja, usa-se a estrutura da nossa língua falada: Sujeito, verbo e predicado, como mostra a Tabela 1:

Linguagem natural	Ambiguidades e imprecisões
1. Obter as notas de provas	Quantas notas?
2. Calcular a média	Que tipo de Cálculo: Média aritmética/ponderada
3. Se a média for maior que 7	Qual o limite: Sete inclusive?
4. O aluno foi aprovado	
5. Senão ele foi reprovado	

Tabela 1: Exemplo de algoritmo representado em linguagem natural (Portugol). **Fonte:** Elaboração Própria.

Importante destacar que esta representação tende a ser pouco praticada, pois, a linguagem natural dá margens a más interpretações, ambiguidades e imprecisões.

Fluxograma Convencional

É uma técnica usada e desenvolvida pelo profissional que está envolvido diretamente com a programação. O objetivo dessa técnica é descrever o método e a sequência que o computador executará as instruções contidas no programa. A técnica utiliza símbolos geométricos, necessários para demonstrar as sequências de operações a serem efetuadas em um

processamento computacional. Esses símbolos são internacionalmente aceitos e estão configurados, segundo as normas ANSI X3.5:1970 e ISO 1028:1973 (a qual foi reeditada como **ISO 5807:1985**).

As formas de representações gráficas (segundo norma ISO 5807:1985) são nada mais do que uma maneira mais simples e concisa de representar os dados sobre uma superfície plana, por meio de diferentes formas geométricas preestabelecidas, de modo a facilitar a visualização completa e imediata da linha de raciocínio lógico de um programador, quando da representação dos dados e do fluxo de ação de um programa de computador.

Um fluxograma convencional é uma representação gráfica de algoritmos no qual formas geométricas diferentes implicam ações (instruções, comandos) distintas. Tal propriedade facilita o entendimento das ideias contidas nos algoritmos e justifica sua popularidade.

Esta forma é aproximadamente intermediária à descrição narrativa e ao pseudocódigo (subitem seguinte), pois é menos imprecisa que a primeira e, no entanto, não se preocupa com detalhes de implementação do programa, como o tipo das variáveis usadas.

Para o nosso curso, adotamos a notação simplificada, com os elementos gráficos que constam na Tabela 2.

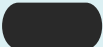








	Início ou final do fluxograma	Terminal - Símbolo utilizado como ponto para indicar o início ou fim do fluxo de um programa
	Fluxo de Dados	Permite indicar o sentido do fluxo de dados. Serve exclusivamente para conectar os símbolos ou blocos
	Processamento	Símbolo utilizado para indicar cálculo, atribuições de valores ou qualquer manipulação de dados
	Decisão	Símbolo que indica a decisão que deve ser tomada, indicando a possibilidade de desvios do fluxo. O resultado deverá ser somente " verdadeiro " (.v.) ou " falso " (.f.)
	Display	Símbolo para indicar uma interação com o usuário/operador do programa. Exibe em tela uma informação
	Dispositivo - Disco	Dispositivo de Armazenamento de dados - Memória de massa, onde podem ser gravados os dados manipulados pelo programa
	Cartão Perfurado	Dispositivo de Entrada ou saída de dados
	Entrada Manual	Dispositivo para entrada de dados - Simboliza a entrada de dados por teclado
	Conector	Conectar partes do digrama que foram particionadas. Por exemplo, o símbolo de decisão faz desvios (.v.e.f.), e depois precisa convergir para um único ponto. Isto é possível com a utilização do conector.

Tabela 2: Notação simplificada dos símbolos do fluxograma adotados neste curso. **Fonte:** Elaboração Própria.

Importante destacar que, partindo do símbolo inicial, há sempre um único caminho orientado a ser seguido, representando a existência de uma única sequência de execução das instruções, conforme podemos verificar na Figura 1, que retrata a representação do algoritmo de cálculo da média de um aluno sob a forma de um fluxograma, e sua respectiva representação em forma de algoritmo.

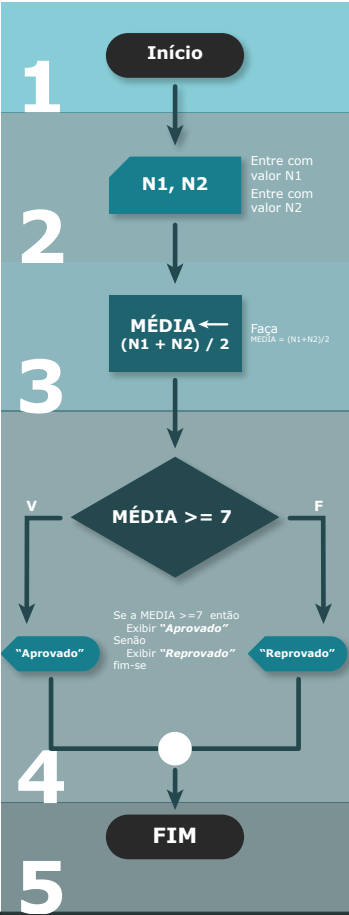


Figura 1: Algoritmo representado por fluxograma e português estruturado para calcular a média de duas notas. **Fonte:** Elaboração Própria.

SAIBA MAIS

Utilize gratuitamente o software de diagramação “visual Paradigm online: <https://online.visual-paradigm.com/diagrams.jsp#diagram:proj=0&type=Flowchart>

Pseudocódigo

Esta forma de representação de algoritmos é rica em detalhes, como a definição dos tipos das variáveis usadas no algoritmo. Por assemelhar-se bastante à forma em que os programas são escritos, encontra muita aceitação.

Esta representação é suficientemente geral para permitir a tradução de um algoritmo nela representado para que uma linguagem de programação específica seja praticamente direta. Abordaremos mais detalhadamente e praticaremos esta técnica na próxima seção.

INTRODUÇÃO À LÓGICA

A lógica de programação é necessária para os profissionais que praticam o desenvolvimento de programas. A lógica está presente na vida das pessoas constantemente. Isto é, está presente sempre que pensamos, falamos e escrevemos, pois, para realizar essas ações necessitamos que os pensamentos estejam ordenados de modo a alcançar o resultado esperado. Podemos entender que a lógica consiste simplesmente na organização e explicação de um pensamento.

Princípio da Resolução de Problemas

Começamos este tópico compreendendo a palavra “problema”. Pode-se dizer que problema é uma proposta duvidosa, que pode ter numerosas soluções, ou questão não solvida e que é objeto de discussão. Na nossa abordagem, problema é uma questão que foge a uma determinada regra. Importante destacar que os diagramas de blocos são realmente o melhor instrumento para avaliação do problema do fluxo de informações de um dado sistema.

Leia com atenção o problema proposto abaixo:

Uma determinada escola, cujo cálculo da média é realizado com as quatro notas bimestrais que determinam a aprovação ou reprovação dos seus alunos. Considere ainda que o valor da média deva ser maior ou igual a 7 para que haja aprovação.

Vamos agora desenvolver juntos uma solução, utilizando a técnica de desmembramento da solução, **com sucessivos refinamentos**. Para isso, considere que iremos partir do entendimento geral do problema para o mais específico. Acompanhe os passos:

■ Passo 1

Represente no diagrama a ideia geral do programa, conforme demonstrado na Figura 2:

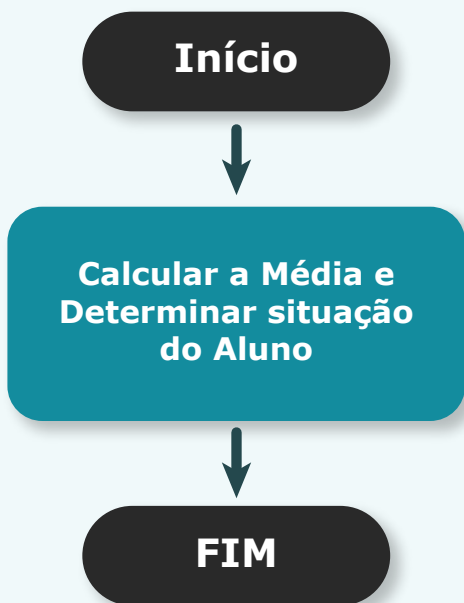


Figura 2: Passo 1 - Visão Geral do Problema. **Fonte:** Elaboração Própria.

Passo 2

Faça um detalhamento no que se refere à entrada e saída, ou seja, deve-se entrar com as quatro notas bimestrais para se obter, como resultado, o cálculo da média e assim definir a aprovação ou reprovação do aluno. A Figura 3 apresenta o diagrama de blocos com mais detalhes.

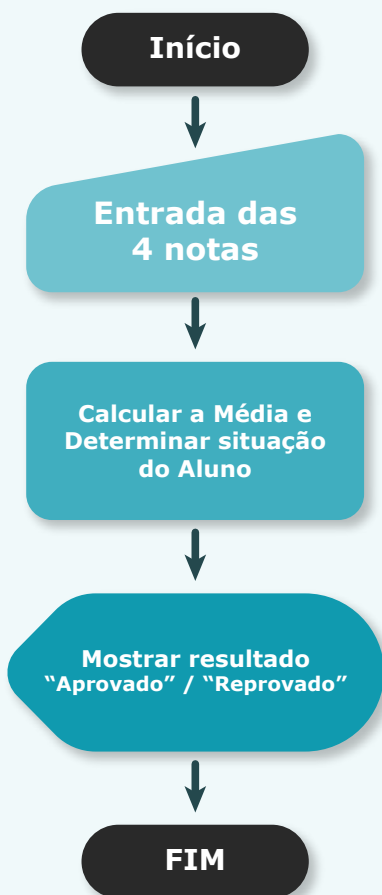


Figura 3: Passo 2 - Detalhamento de Entrada e Saída do Algoritmo.

Fonte: Elaboração Própria.

Passo 3

Esta etapa consiste em trabalhar o termo “determinar a aprovação”. Para ser possível determinar algo, é necessário estabelecer uma condição. Assim sendo, uma condição envolve uma decisão a ser tomada segundo um determinado resultado. No caso, a média. Dessa forma, a condição de aprovação: média maior ou igual a 7 (sete) deve ser considerada no algoritmo. Com isso, inclui-se este bloco de decisão, como mostra a Figura 4:

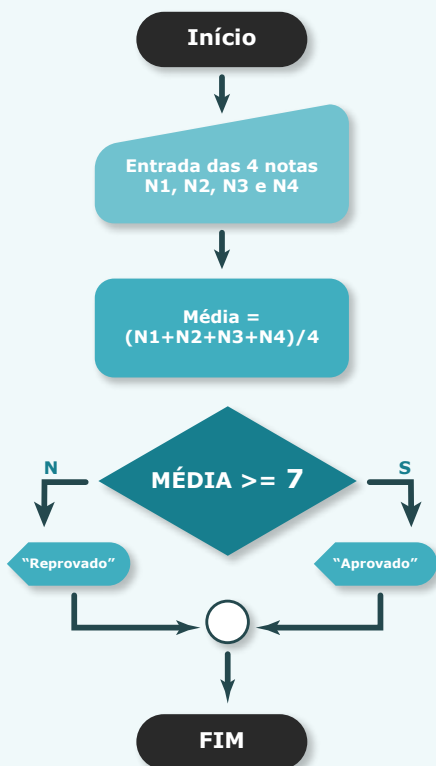


Figura 4: Passo 3 - Detalhamento do Processamento Central - neste caso: “Determinar a aprovação”. **Fonte:** Elaboração Própria.

PARADIGMAS DE DIAGRAMAÇÃO

As representações gráficas de um diagrama de blocos podem ser feitas de várias maneiras (Lógica linear, Lógica estruturada, Programação orientada por objeto, Modular, Diagrama de Chaplin e português estruturado) e possuem estruturas diferenciadas. Em nosso curso, utilizaremos as técnicas de programação estruturada e português estruturado.

Paradigma da Programação Estruturada

Soluciona problemas a partir de sua quebra em problemas menores, de mais fácil solução, denominados de sub-rotinas ou subprogramas. Normalmente, o trabalho de cada sub-rotina consiste em receber dados como entrada, processar esses dados e retornar o resultado do processamento para o módulo de software que o executou. Este paradigma ainda defende que todo processamento pode ser realizado pelo uso de três tipos de estruturas: sequencial, condicional e de repetição. É o paradigma adotado neste curso.

Com a utilização desse paradigma, podemos representar as seguintes estruturas: Sequência, Decisão (*if...then...else*); laços de repetição (*while / do...while*)

e case. Abaixo analisaremos detalhadamente cada uma dessas estruturas:

■ Estrutura de Sequência

Conforme podemos observar na Figura 5, temos uma estrutura de sequência, na qual podemos observar que, ao término de cada instrução, dá-se início a uma nova instrução. As instruções são executadas em uma sequência, parte-se da primeira instrução do algoritmo e, pelo fato de não haver desvios (Condições e laços de repetição), segue “gravitacionalmente” até alcançar a última instrução da sequência.

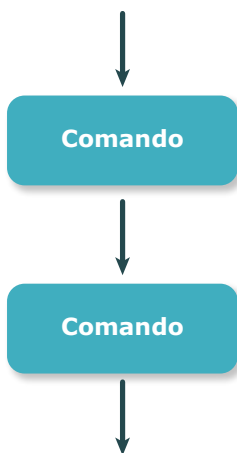


Figura 5: Estrutura de Sequência. **Fonte:** Elaboração Própria.

■ Estrutura if...then...else

A estrutura da Figura 6 representa um importante comando de decisão, no qual o resultado da condição pode ter dois caminhos: Verdadeiro ou falso.

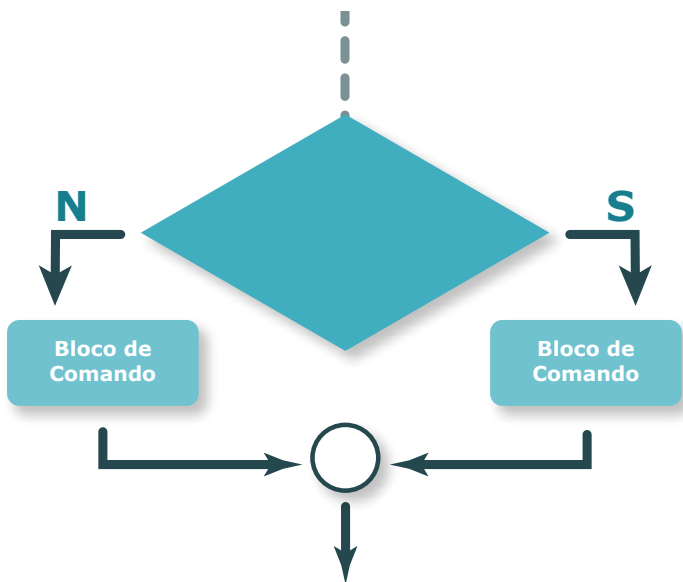


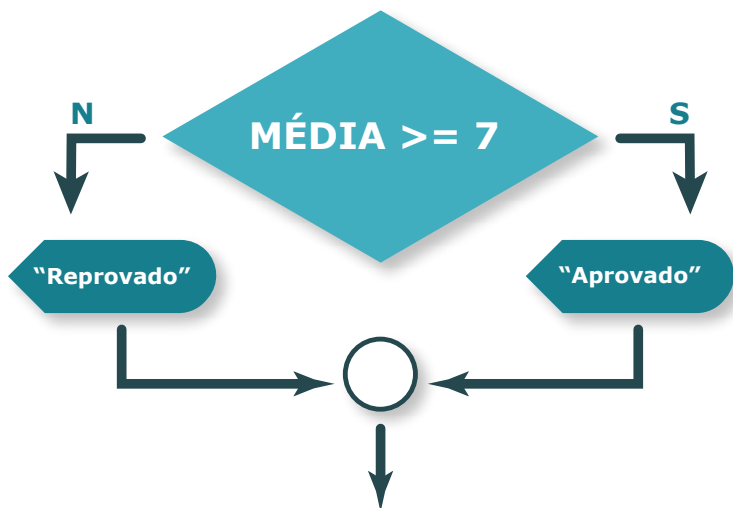
Figura 6: Estrutura *if...then...else*. **Fonte:** Elaboração Própria.

SAIBA MAIS

Explicando o termo “gravitacionalmente”

No contexto de programação é uma referência à ação exercida pelo fenômeno da gravidade, em que um objeto é sempre atraído para o centro da terra, ou seja, de cima para baixo. Se você observar o diagrama da estrutura sequencial, as instruções acontecem de cima para baixo.

Na linguagem hipotética *Portugol*, esta estrutura é codificada pelo comando **se...então...senão...fim_se**, como demonstrado na Figura 7:



1 PROGRAMA xxxxxxxxx

2 INICIO

■ ■ ...

■ ■ ...

SE a (media >=7) **ENTÃO**

Exibir "Aprovado"

SENAO

Exibir "Reprovado"

FIM_SE

Figura 7: Codificação (Portugol) da Estrutura se...então...senão... fim_se. **Fonte:** Elaboração Própria.

Estrutura while

A estrutura da Figura 8 representa a estrutura **while**. Nesse caso, o bloco de operações será executado enquanto a condição for verdadeira. O teste da condição será sempre realizado antes de qualquer operação do bloco de comandos.

Enquanto a condição for verdadeira, o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

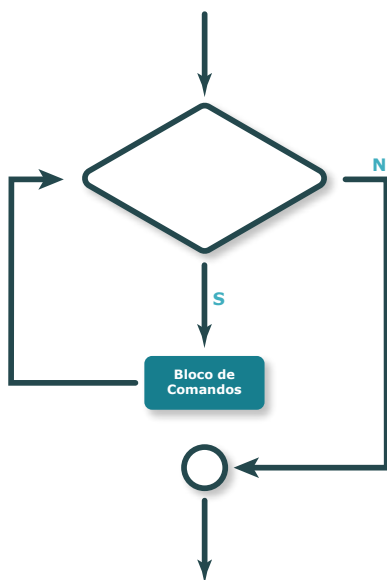


Figura 8: Estrutura de Repetição **while**. Fonte: Elaboração Própria.

Na linguagem hipotética *Portugol*, essa estrutura pode ser codificada pelos comandos **enquanto... faça...fim_enquanto** ou comando **para...de..até... faça...fim_para**, como demonstrado à seguir, cujo objetivo é demonstrar um laço de repetição (*looping*)

para uma contagem automática de 1 a 100. Em linguagens de programação, como C e Java, os respectivos comandos são representados por **while** e **for**.

1	PROGRAMA ExemploEnquanto
2	INICIO
3	Criar VARIÁVEIS
4	Contador tipo inteiro
5	Contador = 1
6	ENQUANTO (Contador <= 100) FAÇA
7	//Mostra o valor do contador em vídeo - De 1 a 100 (Um de cada vez)
8	EXIBIR ("O valor do contador é: ", Contador)
9	Contador = Contador + 1
10	FIM_ENQUANTO
11	FIM

1	PROGRAMA ExemploPara
2	INICIO
3	Criar VARIÁVEIS
4	Contador tipo inteiro
5	PARA Contador DE 1 ATÉ 100 FAÇA
6	//Mostra o valor do contador em vídeo - De 1 a 100 (Um de cada vez)
7	EXIBIR ("O valor do contador é: ", Contador)
8	Contador = Contador + 1
9	FIM_PARA
10	FIM

A instrução **enquanto...faça...fim_enquanto** indica o início de um laço e recebe como parâmetro uma condição. Essa condição é chamada de **condição de parada**, pois, quando ela for falsa, o laço é encerrado.

A instrução ***para...de..até...faça...fim_para*** é outra instrução de repetição e tem a mesma finalidade da instrução ***enquanto...faça...fim_enquanto***.

Na maioria dos casos, podemos resolver questões que envolvem repetições com ***enquanto...faça...fim_enquanto*** ou ***para...de..até...faça...fim_para***. A diferença é que, geralmente, utilizamos a instrução ***para...de..até...faça...fim_para*** nos casos em que precisamos de um contador em nossa condição de parada. Ou seja, já sabemos a quantidade de vezes que a repetição ocorrerá. Por exemplo: Para exibir a tabuada de um determinado número (n), faremos dez vezes, pois, a tabuada será $1 \times n$, $2 \times n$... até $10 \times n$.

Estrutura do...while

A Figura 9 representa a estrutura ***do...while***. Essa estrutura de repetição tem a particularidade de executar **ao menos uma vez** o bloco de comando, pois, o teste da condição será sempre realizado após a execução do bloco de comandos (**Pós-testado**). Ou seja, primeiro faz-se o bloco de comandos; e, depois, testa-se a **condição de parada**. Nesse caso, o bloco de comandos será executado enquanto a condição **for falsa**.

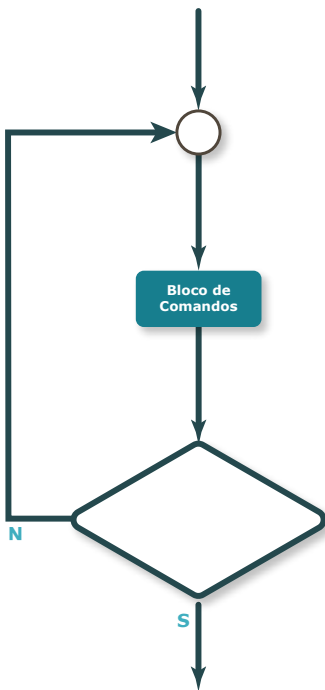


Figura 9: Estrutura de Repetição do...while. **Fonte:** Elaboração Própria.

Na linguagem hipotética *Portugol*, essa estrutura pode ser codificada pelo comando **repita...até_que... faça**, como demonstrado no código a seguir, cujo objetivo é demonstrar um laço de repetição (*looping*) para uma contagem automática de 1 a 100. Em linguagens de programação, como C e Java, é representada pelo comando **do...while**.

SAIBA MAIS

Bloco de comandos: É um conjunto composto por 1 ou “n” linhas de comandos que formam

uma sequência de instruções para o computador.
Por exemplo:

Comando 1- EXBIR ("Entre com a Nota 1");

Comando 2- LEIA Nota1;

Comando 3- EXBIR ("Entre com a Nota 2");

Comando 4- LEIA Nota 2;

Comando 5- Média = (Nota1 + Nota2)/2.

Comando n-

1	PROGRAMA ExemploRepita
2	INICIO
3	Criar VARIÁVEIS
4	Contador tipo inteiro
5	Contador = 1
6	REPITA
7	//Mostra o valor do contador em vídeo - De 1 a 100 (Um de cada vez)
8	EXIBIR ("O valor do contador é: ", Contador)
9	Contador = Contador + 1
10	ENQUANTO (Contador >= 100) FAÇA
11	FIM

Estrutura SWITCH

A estrutura **SWITCH**, ilustrada na Figura 10 estrutura condicional SWITCH. Fonte: Elaboração Própria." na página 25, proporciona fazer testes condicionais, em que podemos usar várias opções de comparações, à semelhança da estrutura *if...then...else*.

Assim como o nosso conhecido par IF ELSE, o comando **switch** também serve para fazer testes condicionais.

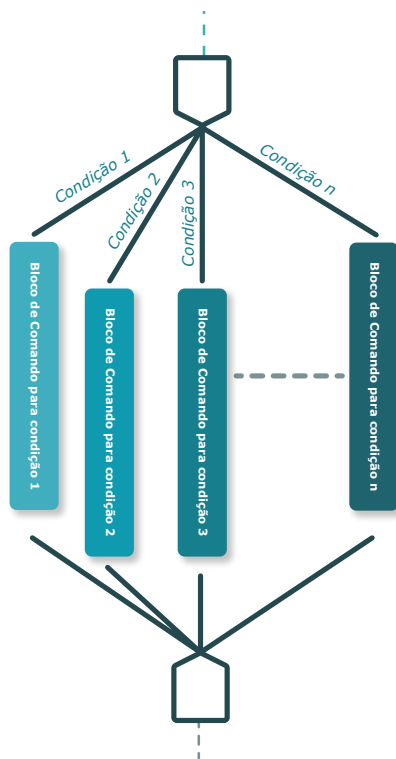


Figura 10: estrutura condicional SWITCH. **Fonte:** Elaboração Própria.

Na linguagem hipotética *Portugol*, essa estrutura pode ser codificada por meio do comando **escolha... caso...fim_escolha**, como demonstrado nos exemplos a seguir. Observe e compare a codificação desta com a estrutura *if...then...else*. Em linguagens de programação, como C e Java, a estrutura **escolha...caso...fim_escolha** é representada pelo comando **switch...case**.

1	PROGRAMA ExemploEscolha
2	INICIO
3	Numero1 = 2
4	Numero2 = 3
5	Opcao = 2
6	ESCOLHA (Opcao)
7	CASO 1:
8	Exibir("Soma é: ", Numero1+Numero2)
9	PARE
10	CASO 2:
11	Exibir("Multiplicação é:", Numero1*Numero2)
12	PARE
13	OUTRO CASO :
14	Exibir("Opção Inválida")
15	FIM-ESCOLHA
16	FIM

1	PROGRAMA ExemploSe
2	INICIO
3	Numero1 = 2
4	Numero2 = 3
5	Opcao = 2
6	SE (Opção = 1) ENTÃO
7	Exibir("Soma é:", Numero1+Numero2)
8	SENÃO
9	SE (Opção = 2) ENTÃO
10	Exibir("Multiplicação é:", Numero1*Numero2)
11	SENÃO
12	Exibir("Opção Inválida")
13	FIM_SE
14	FIM_SE
15	FIM
16	FIM

PORTUGUÊS ESTRUTURADO

O diagrama de blocos é uma forma de notação gráfica, mas existe outra forma, não gráfica, que é uma técnica narrativa denominada pseudocódigo, também conhecida como português estruturado ou ainda chamada por alguns de “**portugol**”.

Essa técnica é baseada em uma PDL - *Program Design Language* (linguagem de Projeto de Programação). Em nosso curso, adotaremos a língua portuguesa do Brasil, que será detalhada na próxima seção (“*Utilizando Pseudolingagem*”).

Para saber mais sobre os princípios da lógica, acesse o **Podcast – Entendendo os princípios da Lógica de Programação**

Podcast 1



Utilizando uma Pseudolingagem

Este curso adota a utilização de uma pseudolingagem denominada por uns “**Portugol**” e, por outros, **Português Estruturado**. É muito importante considerar que esta linguagem, na verdade, não existe; pois se trata de uma linguagem hipotética; e não

foi criado para ela o compilador que executasse os seus comandos dentro de um computador, servindo apenas como um instrumento didático e permitindo dar ao novato a destreza necessária na montagem das estruturas de programa.

Deste ponto em diante, você terá contato com instruções do pseudocódigo português estruturado, adotado para o nosso curso. A lista de comandos e seus significados constam à Tabela 3.

Classificação	Comando	Descrição	Exemplo
Controle de looping	enquanto... fim_enquanto	Controla looping: Faz bloco de comando condicionado a uma situação	ENQUANTO (contador < 10) FAÇA EXIBIR ("valor do contador e: ", contador) contador = contador + 1 FIM_ENQUANTO
	para...De...Até...Faça ... fim_para		PARA contador DE 0 ATÉ 10 FAÇA EXIBIR ("valor do contador e: ", contador) contador = contador + 1 FIM_PARA
	repita...até_que...Faça		REPITA EXIBIR ("valor do contador e: ", contador) contador = contador + 1 ATE_QUE (i>10) FAÇA
Controle de programa	fim	Marcação de fim do algoritmo. O programa para a sua execução	INICIO FIM
	função...fim	Marcação do início de uma função (sub-rotina)	FUNCAO int calculadora (parametros) início ... FIM
	início	Marcação de início do algoritmo. O programa inicia a sua execução	INICIO FIM
	programa	Identificação do nome do algoritmo que será executado	PROGRAMA calculoImpostoRenda INICIO FIM

Classificação	Comando	Descrição	Exemplo
Decisão	ESCOLHA...CASO... FIM_ESCOLHA	Estrutura de decisão, coordenada a uma situação	<p>Numero1 = 2 Numero2 = 4 Opcao = 2 ESCOLHA (Opcao) CASO 1: EXIBIR (A soma é", Numero1 + Numero2) CASO 2: EXIBIR (A Multiplicação é", Numero1 * Numero2) OUTRO CASO: EXIBIR ("Opção Invalida") FIM_ESCOLHA</p>
	se...Então... senão...fim_se		<p>SE (Numero1 > 0) ENTÃO Resultado = Numero2 / Numero1 SENÃO Resultado = Numero2 FIM_SE</p>
Definição de dados	caractere	Indicar que uma variável é do tipo caractere	<p>VAR Nome: caractere Codigo: inteiro</p>
	conjunto	Designar uma matriz de dados	<p>notas: conjunto [1 .. 4] de real Designa uma matriz de 4 notas do tipo real</p>
	inteiro	Indicar que uma variável é do tipo numérico inteiro	<p>VAR Codigo: inteiro Nome: caractere</p>
	lógico	Indicar que uma variável é do tipo lógico (Booleano: Verdadeiro ou falso)	<p>VAR Codigo: logico Nome: caractere</p>
	real	Indica que uma variável é do tipo numérico real	<p>VAR Codigo: real Nome: caractere</p>
	registro... fim_registro	Indica uma estrutura de registro	<p>xx</p>
	var		<p>VAR Codigo: inteiro Nome: caractere</p>
	ESTRUTURA	Define uma estrutura de dados	<p>VAR Codigo: inteiro Nome: caractere</p>

Classificação	Comando	Descrição	Exemplo
Manipulação entrada/saída de dados	escreva	Comando imperativo para escrever dados em um dispositivo (Arquivo)	<div>xx</div> <div>...</div> <div>...</div>
	exibir	Comando imperativo para exibir informação na tela do operador	<div>...</div> <div>EXIBIR ("Entre com o código do cliente")</div> <div>...</div>
	leia	Comando imperativo para obter dados de um dispositivo de entrada de dados (Teclado ou arquivo)	<div>VAR</div> <div>Código: inteiro</div> <div>EXIBIR ("Entre com o código do cliente")</div> <div>LEIA Código</div> <div>...</div>
Comando especial	//	São duas barras "/" que Designam um comentário no pseudocódigo. Este comando não produz qualquer ação para o computador. Tem apenas a função de documentar (registrar) comentário	<div>...</div> <div>var CODCLI int //Variável Código do Cliente</div> <div>...</div>

Tabela 3: Lista de Comando Portugal e seus significados

Estas instruções, colocadas de forma estratégica, formarão os blocos de programa, conforme demonstrado à seguir, em que podemos observar a solução do problema que foi proposto na subseção “*Princípio da Resolução de Problemas*”. Observe nesta código que as instruções referentes à pseudolinguagem estão grafadas em negrito, cor vermelha.

1	programa MÉDIA
2	var
3	RESULTADO: caractere
4	N1, N2, N3, N4: real
5	SOMA, MÉDIA: real
6	Início
7	leia N1, N2, N3, N4
8	SOMA 1 \leftarrow N1 + N2 + N3 + N4
9	MÉDIA \leftarrow SOMA / 4
10	se (MÉDIA >= 7) então
11	RESULTADO \leftarrow "Aprovado"
12	senão
13	RESULTADO \leftarrow "Reprovado"
14	Fim-se
15	escreva "Nota 1: " N1
16	escreva "Nota 2: " N2
17	escreva "Nota 3: " N3
18	escreva "Nota 4: " N4
19	escreva "Média: ", MÉDIA
20	escreva "Resultado: ", RESULTADO
21	fim

Elementos da Pseudolinguagem

Uma pseudolinguagem possui uma estrutura e diversas palavras reservadas, similares a comandos e instruções das linguagens tradicionais (Java, C, Python, etc.). Para diferenciá-las de outros identificadores, como nome de variáveis ou de procedimentos e funções, as palavras reservadas serão escritas em

letras maiúsculas em nosso curso. O que estamos a definir neste curso deverá ser tratado como um padrão de fato. Adotaremos aqui a seguinte estrutura para a nossa pseudolinguagem, conforme demonstrado na Figura 11.

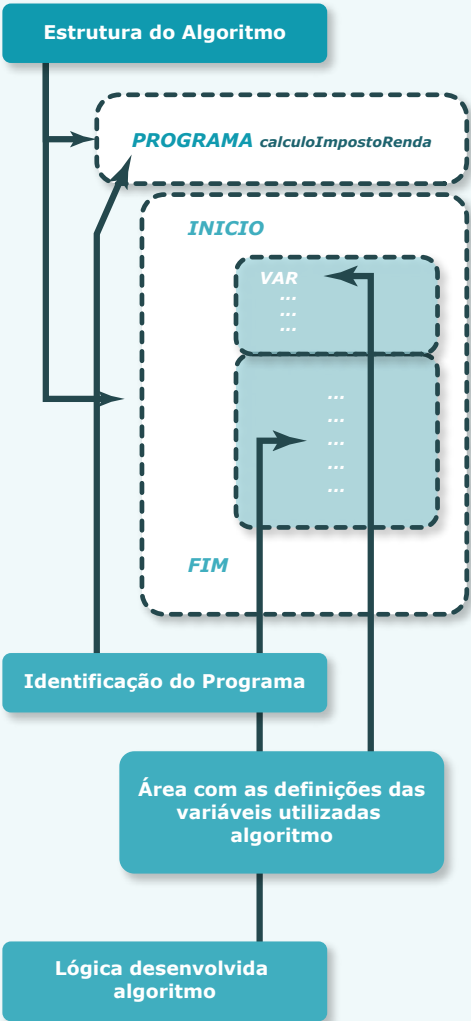


Figura 11: Estrutura de um Algoritmo. **Fonte:** Elaboração Própria.

SAIBA MAIS

Palavra reservada: É um conjunto de termos que são utilizados exclusivamente pela linguagem. Cada palavra quer dizer alguma coisa específica para a linguagem. As instruções são executadas por meio do uso de palavras-chave previamente determinadas, que chamaremos de palavras reservadas. A priori, elas não poderão ser usadas para outras coisas, além do determinado para a linguagem. Exemplo: O programador não pode utilizar uma variável com nome **LER**, pois este é um comando para obter informações de um dispositivo (Teclado/arquivo)

Procedimento e funções

Uma técnica comumente utilizada na programação é a divisão de um grande programa em porções menores (*dividir para conquistar*) denominadas módulos. Esses módulos podem consistir em arquivos de código-fonte ou apenas rotinas. Essas rotinas podem ser classificadas em dois tipos: funções ou procedimentos.

O que diferencia funções de procedimentos é que, na primeira, operam valores passados em parâmetros (argumentos) e devolvem um valor resultante. Já na segunda, utilizam ou não argumentos, e não devolvem valor resultante.

No exemplo a seguir, podemos observar uma função que recebe dois valores (*Operando1* e *Operando2*) e calcula os produtos destes, devolvendo o valor calculado na variável *Retorno*.

1	<i>FUNÇÃO</i> ProdutoValores (Operando1:INTEIRO, Operando2:INTEIRO):INTEIRO
2	VAR
3	Resultado: INTEIRO
4	//faz a multiplicação entre Operando 1 e o Operando 2
5	Resultado = Operando1 * Operando2
6	RETORNAR (Resultado)
7	<i>FIM_FUNÇÃO</i>

CONSTANTES E VARIÁVEIS

Um programa de computadores tem a função elementar de processar dados em memória do computador. Para isso, reservamos espaços em memória, por intermédio de endereçamento. O programa poderá ter as duas formas de manipular estes endereçamentos, que são as constantes e as variáveis.

Constantes

Constante é um espaço em memória com determinado valor fixo que não se modifica durante a execução de um algoritmo. Uma constante pode assumir valores do tipo numérico, lógica ou literal.

No exemplo abaixo, podemos ver que a expressão possui em seu denominador o valor **constante numérico 2**:

$$\text{Resultado} = (\text{Nota1} + \text{Nota2}) / 2$$

Outro exemplo de constantes são as mensagens que o programa pode exibir em tela: “Entre com o código do cliente”, “Valor digitado é Inválido”, “Inclusão Efetuada com Sucesso”, etc.

Variáveis

Variável é a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a uma posição de memória RAM, cujo conteúdo pode ser alterado ao longo do tempo, durante a execução de um programa. É muito importante observar que uma variável só pode armazenar um valor a cada instante, conforme demonstrado na Figura 12, em que podemos observar que a variável “Código” possui o valor **123**.

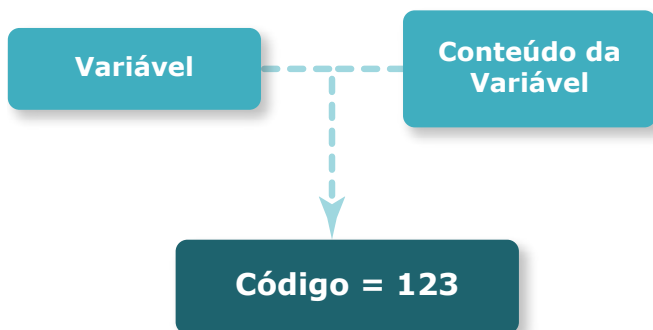
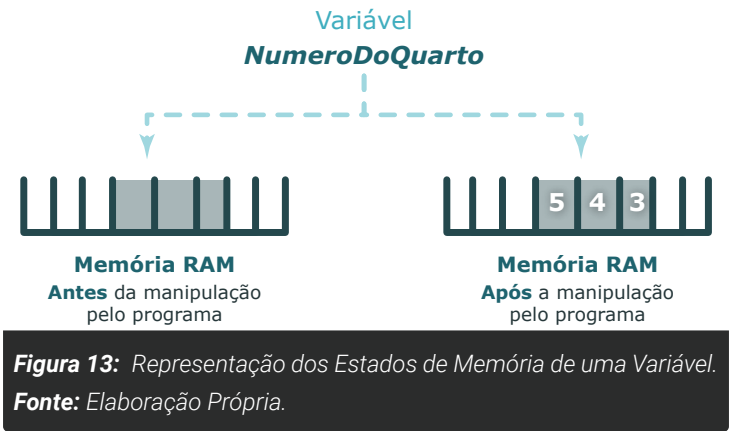


Figura 12: Representação de Variável de Memória. **Fonte:** Elaboração Própria.

Em uma variável é possível armazenar dados de vários tipos: numéricos, strings (texto), booleanos (verdadeiro ou falso), referências, entre outros.

Quando declaramos uma variável, estamos atribuindo um nome simbólico a um endereço da memória RAM. Dentro de nosso programa, utilizaremos esse nome para manipular a informação contida no endereço da memória relacionado à variável.

A Figura 13 representa uma atribuição do valor (constante) numérico **543** à variável **NumeroDoQuarto** .



TIPOS DE DADOS PRIMITIVOS

Um programa de computador pode gastar boa parte de seu tempo a processar a movimentação de valores entre as diversas posições de memória. É importante destacar que essas posições de memória podem ser classificadas de acordo com o seu tipo de informação que esta representa. Abaixo discorreremos sobre cada um destes tipos.

Dados Primitivos do Tipo Numérico

São específicos para armazenamento de números, com os quais é possível realizar operações matemáticas. Podem ser ainda classificadas como Inteiras ou *Reais*. As variáveis do tipo inteiro são para armazenamento de números inteiros e as *Reais* são para o armazenamento de números que possuam casas decimais.

Dados Primitivos do Tipo Caractere

Específicos para armazenamento de conjunto de caracteres que não contenham números.

Dados Primitivos do Tipo Alfanumérico

Específicos para dados que contenham letras e/ou números. Pode, em determinados momentos, conter somente dados numéricos ou somente literais. Se usado somente para armazenamento de números, não poderá ser utilizada para operações matemáticas.

Dados Primitivos do Tipo Lógico

Específicos para dados que armazenam somente dados lógicos que podem ser **Verdadeiro** ou **Falso**.

OPERADORES

Em linguagens de programação, podemos encontrar operadores para que diversas operações possam ser executadas com os dados. Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos três tipos de operadores:

- Operadores Aritméticos
- Operadores Relacionais
- Operadores Lógicos.

Operadores Aritméticos

Os operadores aritméticos permitem que sejam efetuados cálculos com dados armazenados em memória, e estes devem ser feitos somente com dados do tipo primitivo numérico. A Tabela 4 apresenta os operadores aritméticos permitidos nas principais linguagens de programação

Operador	Descrição
+	Adição: Efetua a soma entre os operandos
-	Subtração: Efetua a subtração entre os operandos
*	Multiplicação: Efetua a multiplicação entre os operandos
/	Divisão: Efetua a divisão entre dois operandos
%	Resto da Divisão: Obtem o resultado do RESTO da divisão entre dois operandos

Tabela 4: Operadores Aritméticos Possíveis nas Linguagens de Programação. **Fonte:** Elaboração Própria

Operadores Relacionais

Também chamados de operadores de comparação, permitem que sejam relacionados o conteúdo de duas ou mais variáveis. A Tabela 5 apresenta os operadores aritméticos permitidos nas principais linguagens de programação.

Operador	Descrição
<	Menor que
>	Maior que
<=	Menor que ou igual a
>=	Maior que ou igual a
=	Igual a
<>	Diferente de

Tabela 5: Operadores Relacionais Possíveis nas Linguagens de Programação. **Fonte:** Elaboração Própria.

Operadores Lógicos

Estes operadores lógicos são empregados na construção de expressões lógicas e resultam em um valor binário: **Verdadeiro** ou **Falso**. A Tabela 6 apresenta os operadores lógicos permitidos nas principais linguagens de programação.

Operador	Descrição
E	Efetua a operação lógica E . Somente se <u>ambas expressões</u> forem verdadeiras, o resultado será verdadeiro , caso contrário, retornará falso
OU	Efetua a operação lógica OU . Se <u>uma das expressões</u> forem verdadeiras, o resultado será verdadeiro , caso contrário, retornará falso
NÃO	Uma expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa

Tabela 6: Operadores Lógicos Possíveis nas Linguagens de Programação. **Fonte:** Elaboração Própria

PRATICANDO COM PSEUDOCÓDIGO

Neste capítulo, faremos juntos o desenvolvimento de uma solução para o seguinte problema:

“Elaborar um algoritmo para exibir a tabuada de um número. Este número deverá ser solicitado para que seja entrado pelo teclado (Figura 14).”



Figura 14: Requisito Entrada de Dados. **Fonte:** Elaboração Própria.

“O Número deverá estar no intervalo de 1 a 10. Se estiver fora desse intervalo, informar no vídeo: “Valor informado é inválido”, e encerrar o programa. Deverá ser mostrado no vídeo a sequência da tabuada de 1 a 10. Por exemplo: Se o numero informado for 2, então deverá ser apresentado em tela (um por vez) , conforme demonstrado na Figura 15.

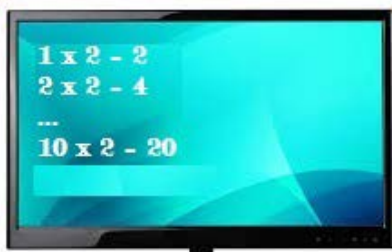


Figura 15: Resultado que o Algoritmo Deverá Produzir. **Fonte:** Elaboração Própria.

Como devemos organizar os elementos desta solução? Faremos uma análise, de forma progressiva, partindo do geral para o detalhe:

1. Começamos a solução pelo entendimento dos requisitos do problema proposto (Geral): Produzir uma tabuada, a partir de um número dado.
2. Identificamos as diversas partes que compõem o problema: Entrada / processamento e Saída.
3. Particularidades identificadas no problema: Deverá ser informado um número válido: Maior que zero e menor que 10.
4. Tratamentos específicos: Se o valor informado for menor que 1, ou se o valor informado for maior que 10, deverá ser informado ao operador, via vídeo, com mensagem explicativa e terminar o programa.
5. Tratamento **principal**: calcular o produto do número informado pela sequência de 1 a 10 e, logo em seguida, deverá ser mostrado em vídeo.
6. Analisar que **variáveis** o algoritmo irá necessitar. Comece com os dados de entrada. Neste caso, te-

remos somente uma entrada pelo teclado (número desejado). Escolha um nome significativo para esta variável. No caso, utilizaremos a variável **Número** para receber o número digitado. Em seguida, identifique variáveis intermediárias (são variáveis que servirão, por exemplo, para guardar o resultado de uma operação matemática). Para o nosso programa, identificamos que serão necessárias duas: **Contador** e **Resultado**. A primeira guardará a contagem do no laço de repetição **for** (explicado mais a diante). A segunda guardará o resultado do produto entre **Número** e **Contador**.

7. Processamento central: Identifique quais são os procedimentos que farão parte deste ponto do programa. Como estudado anteriormente, o programa deverá produzir uma tabuada. Sabemos que, para isso, deverá ser feito o produto de **Número** x 1, **Número** x 2...até **Número** x 10. Você notou que este procedimento é repetitivo? Então, isso nos sugere que podemos fazer uso de algum tipo de laço de repetição. Identificamos que o modelo mais adequado para este problema é o laço **for** (se precisar, retome o capítulo que trata desse assunto), pois faremos o procedimento que se repete por 10 vezes, nesta estrutura.

8. O programa terminará quando ele mostrar o último produto: **Número** x 10.

9. Faça o diagrama de blocos (fluxograma), conforme demonstrado na Figura 16.

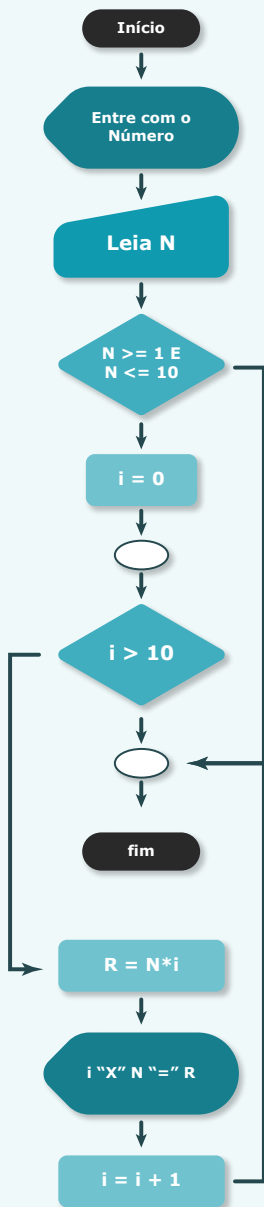


Figura 16: Fluxograma da Solução. **Fonte:** Elaboração Própria.

10. Codifique em pseudolinguagem (Portugol)

Acompanhe, neste próximo exemplo, a codificação do Pseudocódigo (Portugol) do problema da tabuada. Importante atentar que a codificação tem como base o diagrama de blocos desenvolvido anteriormente. Observe também que a marcação “//” (Duas barras) tem como objetivo a colocação de um comentário no código. Este comando não tem ação na lógica do algoritmo.

1	PROGRAMA Tabuada
2	INICIO
3	VAR
4	Numero:INT
5	Resultado:INT
6	Contador:INT
7	EXIBIR("Entre com Numero")
8	LEIA Numero
9	SE (Numero >= 1 E Numero <= 10) ENTAO
10	PARA Contador DE 1 ATÉ 10 FAÇA //A variável Contador será incrementada automaticamente de +1
11	Resultado = Contador x Numero
12	EXIBIR(Contador "x"Numero " = " Resultado) //exibe em vídeo o resultado da tabuada
13	FIM_PARA
14	SENÃO
15	EXIBIR("Número inválido")
16	FIM_SE
17	FIM

Observe também que foi utilizada a estrutura de repetição “PARA” [FOR] com o objetivo de resolver o problema proposto.

TESTE DE MESA

Após escrever o seu algoritmo, é fundamental testá-lo para averiguar se ele funciona corretamente. O teste de mesa é a execução passo a passo do algoritmo, sem utilizar o computador, empregando apenas papel e caneta/lápis, como se ele fosse executado no computador. O teste de mesa é um recurso importante na construção e aprendizado de algoritmos e possui as seguintes características:

- Facilita o entendimento do fluxo de execução do algoritmo;
- Permite identificar erros lógicos na construção do algoritmo;
- Permite verificar se o algoritmo leva ao resultado esperado por meio de uma simulação de valores;
- Possibilita a verificação da evolução do conteúdo das variáveis.

Passos para realização do teste de mesa:

- Identifique todas as variáveis manipuladas em seu algoritmo
- Crie uma tabela com linhas e colunas em que:
 - Cada coluna represente uma variável – reserve a primeira coluna para identificar a sequência de comandos do seu algoritmo
 - Crie coluna para representar o(s) dispositivo(s) de saída (Vídeo/Arquivo)

- As linhas corresponderão às instruções observadas pelo teste
- Em seguida, siga cada instrução do seu algoritmo (linha por linha) e reflita as alterações de memória nas respectivas colunas (variáveis). Note que cada linha de sua tabela é uma “*fotografia instantânea*” da memória com a posição de cada variável
 - Atribua valores hipotéticos para as entradas de dados (Teclado/arquivo), em suas respectivas linhas, e anote na linha correspondente (Comandos **LEIA**)
- Siga cada instrução até a última linha de seu algoritmo

Vamos fazer juntos um teste de mesa para um algoritmo que implementa uma solução para o seguinte problema:

Escreva um algoritmo para ler dois números (N1 e N2), e trocar seus valores. Ao final, mostre o valor das variáveis N1 e N2.

Vamos analisar o resultado do algoritmo, bem como acompanhar e detalhar a elaboração do teste de mesa, a partir do exemplo a seguir, na qual podemos ver a solução dada, bem como a Tabela 7 com o teste de mesa.

1	PROGRAMA TrocaValores
2	INICIO
3	VAR
4	Auxiliar; N1; N2:inteiro
5	EXIBIR ("Digite o valor de N1: ")
6	LEIA N1
7	EXIBIR ("Digite o valor de N2: ")
8	LEIA N2
9	Auxiliar = N1
10	N1 = N2
11	N2 = Auxiliar
12	EXIBIR ("O valor de N1 é:", N1)
13	EXIBIR ("O valor de N2 é:", N2)
14	FIM

Estados						
Memória				Entrada		Saída
Linha	N1	N2	Auxiliar	LEIA	Teclado	Vídeo
5	--	--	--	--	--	Digite o valor de N1
6	23	--	--	23	23	Digite o valor de N1
7	23	--	--	--	--	Digite o valor de N2
8	23	44	--	44	44	Digite o valor de N2
9	23	44	23	--		Digite o valor de N2
10	44	44	23	--		Digite o valor de N2
11	44	23	23	--		Digite o valor de N2
12	44	23	23	--		O valor de N1 é: 44
13	44	23	23	--		O valor de N2 é: 23

Tabela 7: Teste de Mesa. **Fonte:** Elaboração Própria.

Começamos a nossa tabela de teste de mesa criando as colunas “Linha”, “N1”, “N2”, “Auxiliar”, “Vídeo” e “Teclado”. Essas colunas correspondem, respectivamente, à identificação da linha do algoritmo, variáveis N1, N2 e Auxiliar, e, por último, os estados dos dispositivos de entrada e saída: teclado e vídeo respectivamente.

Note que começamos nossa tabela com a **linha 5**, pois, se olharmos para o algoritmo, esta é a primeira linha que mudará um estado no programa, com o comando **EXIBIR** (Nesse caso, aparecerá no vídeo a mensagem “Digite o valor de N1”).

Em seguida, fomos para **linha 6** do algoritmo e observamos que esta alterou o estado de memória, resultado do comando **LEIA**. Nesse caso, a variável **N1** foi alterada para o valor 23, que foi digitado (HIPOTETICAMENTE), via **teclado**, e apareceu na tela do usuário. Note que as outras variáveis não foram alteradas.

A **linha 7** do programa solicita que o operador entre com um novo valor (Comando EXIBIR). O estado da tela é alterado para “Digite o valor de N2”.

Em seguida, fomos para **linha 8** do algoritmo e observamos que esta alterou o estado de memória, resultado do comando **LEIA**. Nesse caso, a variável N2 foi alterada para o valor 44, que foi digitado (HIPOTETICAMENTE), via teclado, e apareceu na tela do usuário. Note que as outras variáveis não foram alteradas.

Próximo passo, vamos refletir o evento da **linha 9**, em que é feita o comando de atribuição de valor para a variável Auxiliar. Esta variável recebe o conteúdo da variável N1. Note que alteramos na tabela somente a coluna que corresponde à variável Auxiliar. O restante continua inalterado, inclusive o vídeo.

Vamos para a **linha 10**, em que é feito o comando de atribuição de valor para a variável N1. Esta variável recebe o conteúdo da variável N2. Note que alteramos na tabela somente a coluna que corresponde à variável N1. O restante continua inalterado, inclusive o vídeo.

Seguimos para acompanhar para a **linha 11**, em que é feito o comando de atribuição de valor para a variável N2. Esta variável recebe o conteúdo da variável Auxiliar. Note que alteramos na tabela somente a coluna que corresponde à variável N2. O restante continua inalterado, inclusive o vídeo.

O comando da **linha 12** mostra o resultado em tela, com o comando EXIBIR. Neste comando, não houve nenhuma alteração de estado memória, temos somente alteração em vídeo, em que aparece o resultado do comando: **O valor de N1 é: 44.**

Finalmente, chegamos ao comando da **linha 13**, que mostra o resultado em tela, com o comando EXIBIR. Neste comando, não houve nenhuma alteração de estado memória. Mas é alterado o resultado do vídeo, onde aparece o resultado do comando: **O valor de N2 é: 23.** A linha 13 é o último comando, portanto, nesta,

temos uma **fotografia do estado final da memória**, ao término do programa.

Para saber mais sobre os ciclos do processo da produção de um programa de computador, acesse o **Podcast – A arte de Programar: Do início ao fim.**

Podcast 2



CONSIDERAÇÕES FINAIS

Caro estudante, estudamos juntos a conceituação de lógica e sua aplicação na solução de problemas computacionais. Observamos como lidar com problemas que requerem atenção e boa performance de nosso raciocínio, porém, enfatizamos que não se ensina a pensar logicamente. O exercício da lógica formal constante leva ao desenvolvimento gradual na formação de um programador. Lembrando que lógica não se ensina, lógica se pratica.

Aprendemos, também, o processo de produção de um programa, em que começamos com o **entendimento do problema** (visão geral), depois, por sucessivos refinamentos, chegamos ao detalhamento deste entendimento. Logo após, aplicamos as técnicas de solução do problema, partindo do **diagrama de blocos** (fluxograma), passando pela produção do **algoritmo** e, finalmente, chegamos ao **teste de mesa**, para validar a proposta de solução do problema, em que mostramos a técnica de verificação do estado de memória para cada alteração de uma variável manipulada no programa.

Síntese



LÓGICA DE PROGRAMAÇÃO

CONCEITOS FUNDAMENTAIS

Nesta unidade temos desenvolvidos os conceitos que são fundamentais para você desempenhar de forma satisfatória na lógica de programação. Abordamos o conceito de lógica e a importância de sua aplicação no desenvolvimento de programas de computadores.

Conceitos de algoritmo e o diagrama de blocos.

Abordamos os princípios que compõem a resolução de problemas, bem como saberá identificar as técnicas de programação. Para finalizar introduzimos o

conceito de uma pseudolinguagem para o desenvolvimento dos algoritmos e apresentaremos a técnica de teste de mesa.

Para este fim, estruturamos este material da seguinte forma:

Fundamentos de Lógica de programação

Conceito de lógica

Necessidades do uso de lógica

Aplicação da lógica na programação

Conceito de algoritmo

Pseudocódigo

Introdução à lógica

Princípio da resolução de problemas

Paradigmas de diagramação

Estrutura de Sequencia

Estrutura if..then...else

Estrutura while

Estrutura do...while

Estrutura SWITCH

Português Estruturado

Utilizando uma Pseudolinguagem

Elementos da Pseudolinguagem

Procedimento e funções

Constantes e Variáveis

Tipos de Dados primitivos

Operadores Aritméticos

Praticando com Pseudocódigo

Teste de Mesa

Conceito de teste de mesa

Passos para realização do teste de mesa

Referências

ALVES, W. P. **Linguagem e Lógica de Programação**. São Paulo: Érica, 2015.

EDELWEIS, N.; LIVI, M. A. C.; LOURENÇO, A. E. **Algoritmos e programação com exemplos em Pascal e C**. São Paulo: Bookman, 2014.

FORBELLONE, A. L. V; EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. São Paulo: Prentice Hall, 2015.

MANZANO, J. A. N. G; MATOS, E; LOURENÇO, A. E. **Algoritmos**: Técnicas de Programação. São Paulo: Érica, 2015.

MANZANO, J. A. N. G; OLIVEIRA, J. F. **Algoritmos**: Lógica para desenvolvimento de programação de computadores. São Paulo: Saraiva, 2016

PERKOVIC, L. **Introdução à computação usando Python**. Rio de Janeiro: LTC-Livros Técnicos e Científicos, 2016.

PUGA, S.; RISSETTI, G. **Lógica de programação e estrutura de dados**: com aplicação em Java. São Paulo: Pearson Education, 2004.

ZIVIANI, N. **Projeto de Algoritmos**: Com implementações em Pascal e C. São Paulo: Cengage Learning, 2011.

FaM
ONLINE