Unidade 3: Introdução à Lógica

Prof. Daniel Caetano

Objetivo: Apresentar o conceito de lógica, seu uso e sua representação.

Bibliografia: ASCENCIO, 2007; MEDINA, 2006; SILVA, 2010; SILVA, 2006.

INTRODUÇÃO

Um programa de computador é essencialmente composto por cálculos e decisões lógicas. Com os cálculos já estamos acostumados: são contas de soma, subtração, multiplicação... dentre tantas outras. Mas e a parte das decisões lógicas?

As decisões lógicas são aquelas que coordenam a execução dos cálculos, para que, em conjunto, eles produzam o resultado para um problema maior. São as decisões lógicas que definem se um cliente tem ou não desconto em sua compra, ou se um aluno foi ou não aprovado em seu curso.

Dada a importância da lógica neste processo, pode-se dizer que programar um computador é, em grande parte, um exercício de lógica.

Esta aula apresentará o conceito de lógica e como usá-la para construir algoritmos, isto é, coordenar a execução de cálculos para resolver um problema maior. Serão apresentadas também duas maneiras gráficas distintas de representar os algoritmos, para facilitar a compreensão dos mesmos.

1. O QUE É LÓGICA?

Ainda que não percebamos, a lógica está presente em nosso dia-a-dia. Quando o professor diz ao aluno que ele precisa ter frequência de pelo menos 75% **e** média de pelo menos 6,0 para ser aprovado, o aluno sabe que não adianta atender aos 75% de frequência se não tiver média 6,0; o contrário também vale: de nada adianta ter 10,0 de média se a frequência for inferior a 75%.

Caso o professor tivesse dito que basta o aluno ter frequência acima de 75% **ou** média de pelo menos 6,0 para ser aprovado, a história seria completamente diferente, não é mesmo?

A diferença de interpretação destas duas colocações está, exatamente, na lógica apresentada por elas. Repare nas pequenas partículas em negrito, **e** e **ou**. São elas que fazem toda a diferença.

Como a lógica tem muita relevância dentro do contexto da matemática, os filósofos e matemáticos trataram de "formalizar" a lógica, isto é, representá-la em termos matemáticos. Essa representação é sempre feita com base em sentenças declarativas, também conhecidas como **proposições**, que podem ser **falsas** ou **verdadeiras**.

As proposições consideradas na lógica matemática possuem duas características:

1. Uma proposição é verdadeira ou falsa.
2. Uma proposição não pode ser verdadeira e falsa, simultaneamente.

Podemos escrever proposições como:

p:	2+3 = 5	(proposição verdadeira)
q:	2+2 > 5	(proposição falsa)

As proposições da lógica podem ser agrupadas em uma proposição composta, com o uso de operadores lógicos E, OU e NÃO. O resultado da proposição composta - falsa ou verdadeira - depende das proposições originais simples e dos operadores.

Por exemplo: consideremos que, para que um aluno esteja aprovado, ele precisa de Média ≥ 6,0 e Frequência ≥ 75%. Usando o símbolo M para indicar a média e F para indicar a frequência, podemos escrever estas duas condições na forma de duas proposições lógicas simples:

p: M ≥ 6,0
q: F ≥ 75%

Ocorre que AMBAS as proposições precisam ser verdadeiras para que o aluno seja aprovado; assim, podemos escrever:

r: p **E** q

Em que a proposição **r** indica se o aluno está aprovado (verdadeira) ou não (falsa).

Como é possível verificar, o uso de letras para dscrever as operações E, OU e NÃO pode causar confusões na hora de representar as composições de proposições. Na matemática, para evitar a confusão, são usados os seguintes símbolos:

E: ^
OU: v
NÃO: ¬

Assim, a proposição r pode ser reescrita como:

r: p ^ q

Essa proposição indica que **r** será verdadeira se e somente se M e F forem, simultaneamente, verdadeiras. Se qualquer uma delas (ou ambas) forem falsas, **r** será falsa, indicando que o aluno não está aprovado.

Em um outro exemplo, podemos construir a proposição **u**, que indica se um funcionário é ruim. Consideremos F a frequência com que o empregado falta e A a frequência com que ele chega atrasado, podemos fazer as afirmações **s** e **t**:

s: F > 2%
t: A > 5%

Considerando que um mau empregado é aquele que falta muito ou chega muito atrasado (qualquer um dos dois), pode-se escrever a proposição **u** da seguinte forma:

u: s v t

Assim, **u** será verdadeira sempre que F ou A forem verdadeiras - isoladamente ou simultaneamente.

A função do operador NÃO ¬ não é, diretamente, para compor proposições, mas para inverter o resultado de uma proposição. Por exemplo:

Se M ≥ 6,0, a primeira proposição será verdadeira; se **u** indica se o funcionário é ruim, ¬**u** indica se o funcionário é bom.

A tabela que relaciona todos os resultados possíveis para as composições de proposições com os operadores lógicos é chamada de "tabela verdade". A tabela verdade para duas proposições quaisquer, A e B, está representada abaixo:

A	Operação	B	Resultado
FALSO	*OU*	FALSO	FALSO
FALSO	*OU*	VERDADEIRO	VERDADEIRO
VERDADEIRO	*OU*	FALSO	VERDADEIRO
VERDADEIRO	*OU*	VERDADEIRO	VERDADEIRO
FALSO	*E*	FALSO	FALSO
FALSO	*E*	VERDADEIRO	FALSO
VERDADEIRO	*E*	FALSO	FALSO
VERDADEIRO	*E*	VERDADEIRO	VERDADEIRO
-	*NÃO*	FALSO	VERDADEIRO
-	*NÃO*	VERDADEIRO	FALSO

Na programação, a lógica é empregada para que se possa tomar decisões com base em resultados dos cálculos. Por exemplo: se estamos calculando a média de um aluno e, com

uma proposição lógica verifica-se que a média do aluno é menor que a média mínima de aprovação, é possível imprimir uma mensagem especificando que o aluno está reprovado.

Na prática, é uma combinação de cálculos com decisões baseadas em proposições lógicas que se constrói um programa; entretanto, para que isso fique mais claro, devemos discutir alguns outros conceitos.

2. RESOLVENDO PROBLEMAS: ALGORITMOS

O ato de programar um computador é, no fundo, o ato de **configurar o computador para que ele nos resolva um problema**.

Vejamos o que vários autores nos dizem sobre a palavra "algoritmo":

Segundo Forbellone, "Algoritmo é uma sequência de passos que visa atingir um objetivo bem definido"; segundo Ascencio, "Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma tarefa. Segunfo Manzano, "Algoritmo são regras formais para a obtenção de um resultado ou soluções de um problema, englobando fórmulas e expressões aritméticas".

Ora, segundo esses autores, algoritmo é exatamente a definição dos passos para resolver um problema. Em outras palavras, se tivermos o algoritmo para resolver nosso problema, o ato de programar pode ser descrito como **configurar o computador para que ele siga os passos de um algoritmo**.

Mas qual é a "cara" de um algoritmo? É muito complicado?

Na verdade, não. Como o ser humano sempre teve a necessidade de resolver problemas - e de transmitir o conhecimento de como fazê-lo para outros seres humanos - os algoritmos estão presentes no nosso dia-a-dia há muito mais tempo que os computadores ou máquinas. Um exemplo clássico são as receitas de culinária:

Algoritmo 1 - Fazer um Omelete:
- Passo 1: Em um prato fundo, bata 3 ovos.
- Passo 2: Acrescente sal.
- Passo 3: Acrescente cheiro-verde.
- Passo 4: Bata mais um pouco.
- Passo 5: Leve ao fogo médio em frigideira untada com manteiga.
- Passo 6: Depois de dourar um lado, vire e deixe dourar o outro lado.

Algoritmo 2 - Fazer um Misto Quente
- Passo 1: Pegar o presunto
- Passo 2: Grelhar o presunto
- Passo 3: Colocar o queijo sobre o presunto
- Passo 4: Pegar duas fatias de pão de forma
- Passo 5: Colocar uma fatia de pão sobre o queijo.
- Passo 6: Virar e colocar a outra fatia de pão.
- Passo 7: Deixe dourar ambos os lados.

Mas não é só na culinária que os algoritmos estão presentes:

Algoritmo 3 - Trocar uma Lâmpada
Passo 1: Pegar uma lâmpada nova.
Passo 2: Pegar uma escada.
Passo 3: Posicionar a escada sob a lâmpada queimada.
Passo 4: Subir na escada com a nova lâmpada.
Passo 5: Retirar a lâmpada queimada.
Passo 6: Colocar a nova lâmpada.
Passo 7: Descer a escada.
Passo 8: Testar interruptor.
Passo 9: Guardar a escada.
Passo 10: Jogar lâmpada queimada no lixo.

Algoritmo 4 - Usar um Novo DVD
Passo 1: Ligue os cabos
Passo 2: Ligue o aparelho de TV
Passo 3: Ligue o DVD
Passo 4: Insira o DVD

Você deve estar pensando: "Ah, então construir um algoritmo é fácil!" Realmente não é difícil, mas antes prestemos atenção em alguns detalhes dos algoritmos apresentados. Exemplos:

No Algoritmo 1, onde devo encontrar os ovos? E o que significa "bater os ovos"? Jogá-los contra o prato? No Algoritmo 2, o que significa "grelhar o presunto"? No algoritmo 3, onde a lâmpada queimada ficou quando se desceu a escada... e como ela surgiu de volta para ser jogada fora? Jogar fora em qual lixo? No algoritmo 4, quais são os cabos? Ligá-los onde?

Peguemos o Algoritmo 4 e vamos tentar refiná-lo:

Algoritmo 4 - Usar um Novo DVD
Passo 1: Pegue os cabos de áudio e vídeo.
Passo 2: Conecte a TV ao DVD com o uso destes cabos, com base em sua cor.
Passo 3: Ligue o conector de energia da TV na tomada.
Passo 4: Ligue o conector de energia do DVD na tomada.
Passo 5: Aperte o botão "Ligar" da TV.
Passo 6: Aperte o botão "Ligar" do DVD.
Passo 7: Aperte o botão "Eject" do DVD.
Passo 8: Insira o disco DVD na bandeja.
Passo 9: Aperte o botão "Eject" do DVD.
Passo 10: Aperte o botão "Play".

Esse "refinamento" feito teve o objetivo de "minimizar o conhecimento prévio", isto é, reduzir a necessidade do executor das instruções conhecer previamente os passos sobre como realizar as tarefas que o algoritmo propõe.

Note, porém, que mesmo a versão refinada do Algoritmo 4 ainda traz algumas dúvidas: onde estão os cabos? Onde está o botão "play"? Um refinamento ainda maior

exigiria que fossem descritas as tarefas de "pegar", "conectar", "ligar", "apertar", "inserir"... que são todas tarefas que, se pressupõe, são conhecidas previamente.

Assim, escrever um algoritmo completo depende de saber quais são os conhecimentos prévios disponíveis pela pessoa que vai executar o algoritmo. Por exemplo: para um programador experiente, que trabalhe em uma empresa que só faz aplicações Web, este algoritmo diz tudo:

Algoritmo para Cálculo de Índice de Massa Corporal
Passo 1: faça um programa que calcule o IMC = peso / (altura*altura).

Obviamente se ele não estiver trabalhando em uma empresa Web, fica a dúvida: é para usar web, é para fazer com uma janela, é para fazer como um aplicativo de "modo texto", é para fazer para celular...?

Quanto menos conhecimento prévio o executor do algoritmo tem, mais detalhadas precisam ser as ordens. É neste sentido que desenvolver algoritmos para computadores se torna, por vezes, um pouco massante: os computadores têm um conhecimento prévio muito pequeno, se comparado a um ser humano. Basicamente, o conhecimento prévio dos computadores se resume a:

a) Fazer contas.
b) Tomar decisões com base em proposições lógicas.
c) Obter informações do usuário.
d) Transmitir informações para o usuário.

3. REPRESENTANDO ALGORITMOS

O desenvolvimento de algoritmos para computador é, como visto, um processo um tanto burocrático. Temos de descrever como um problema é resolvido em todos os detalhes, representando todas as tarefas na forma de cálculos e decisões lógicas. Um algoritmo pode ser descrito na forma narrativa, na forma gráfica ou na forma de código ou pseudo-código.

A forma narrativa, usando linguagem natural, é usualmente muito ambigua para ser usada na programação. Por exemplo, considere a seguinte frase:

"O sapo ouviu um ruído da porta".

Não é possível identificar se:

a) O sapo estava junto à porta quando ouviu um ruído.
b) O sapo ouviu um ruído emitido pela porta.
c) O sapo ouviu um ruído de algum lugar que entrou pela porta.

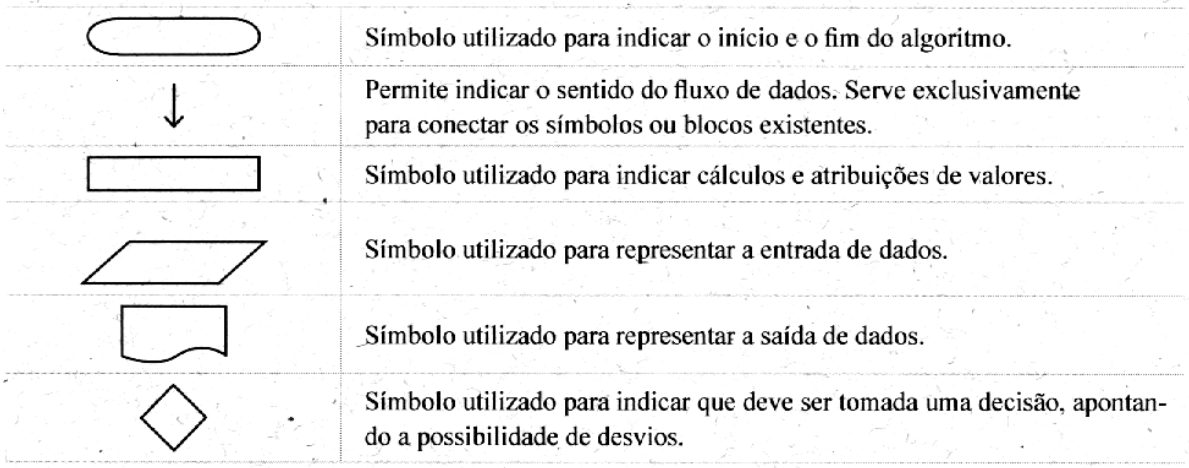
Assim, a narrativa em linguagem natural usualmente não é empregada para representar algoritmos computacionais. Em seu lugar, sobram as alternativas de representação gráfica ou em código. Como as representações com código são muito burocráticas, vamos deixá-las para um momento posterior.

As representações gráficas de algoritmos foram criadas para facilitar o raciocínio e a compreensão inicial, sendo as mais populares o fluxograma e, em alguns meios, o diagrama de Chapin.

Nessa sessão veremos como representar cada um deles.

3.1. FLUXOGRAMAS

Os fluxogramas representam o fluxo de execução de um algoritmo, usando para isso uma simbologia própria, como a representada abaixo:

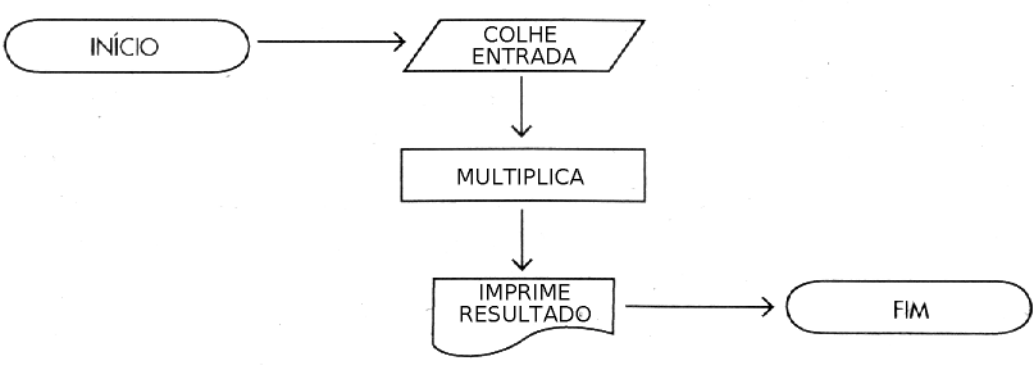


	Símbolo utilizado para indicar o início e o fim do algoritmo.
	Permite indicar o sentido do fluxo de dados. Serve exclusivamente para conectar os símbolos ou blocos existentes.
	Símbolo utilizado para indicar cálculos e atribuições de valores.
	Símbolo utilizado para representar a entrada de dados.
	Símbolo utilizado para representar a saída de dados.
	Símbolo utilizado para indicar que deve ser tomada uma decisão, apontando a possibilidade de desvios.

Consideremos o seguinte algoritmo, escrito na forma de linguagem natural:

Passo 1: Receber dois números que serão multiplicados.
Passo 2: Multiplicar os números.
Passo 3: Mostrar o resultado obtido na multiplicação.

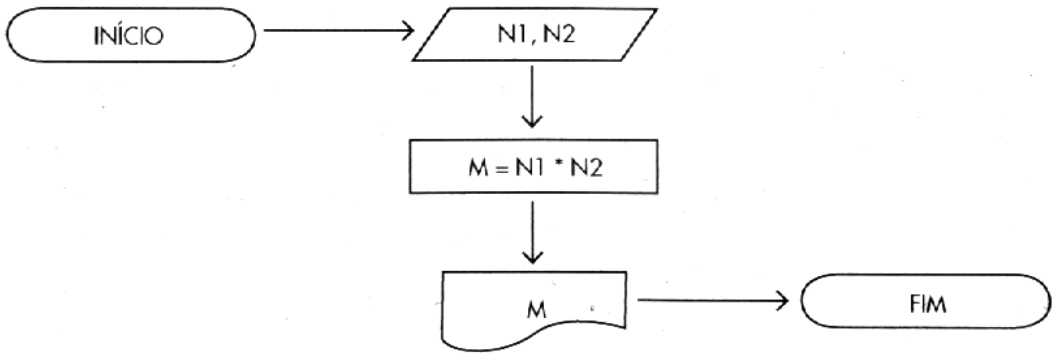
De maneira bem rudimentar, isso pode ser expresso com o seguinte fluxograma:



Por que "rudimentar"? Bem, este fluxograma não explicita algumas coisas. Por exemplo: quantas são as entradas? Exatamente quais são operações feitas? Qual o resultado a ser impresso? Para melhorar o fluxograma, é preciso especificar esses detalhes de um maneira mais clara.

Como não sabemos qual é o número que o usuário vai digitar, usaremos **apelidos**, ou seja, **nomes alfanuméricos** para representar estes valores que não sabemos precisamente. Por exemplo, podemos chamar o primeiro número digitado pelo usuário de **N1** e o segundo número digitado pelo usuário de **N2**. Se chamarmos o resultado da multiplicação de **M**, a operação da multiplicação pode ser escrita como **M = N1 * N2**!

Vamos usar esses "apelidos" que demos aos valores para representar o mesmo fluxograma anterior:



Apesar da leitura mais "codificada", este fluxograma é muito mais claro com relação às operações a serem realizadas. Os símbolos oferecem todas as informações necessárias: após o início, são colhidos dois números: N1 e N2. Estes números são multiplicados, com resultado M. O valor de M é impresso e o algoritmo chega ao fim.

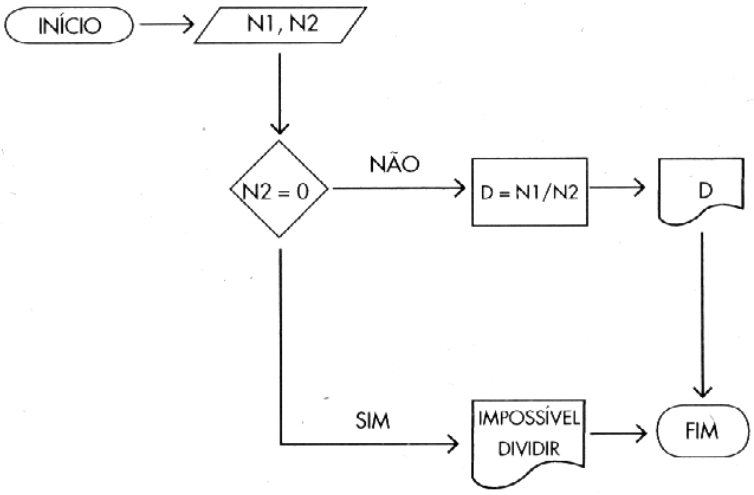
Esses valores que não sabemos quais são e aos quais demos um "apelido" é dado o nome de **variável**. Uma variável é, pois, um valor numérico que, pela impossibilidade de determinação no momento de elaboração do algoritmo, é subsituído por um nome alfanumérico.

Observe que **todo algoritmo tem um único início** e sempre chega a um final.

Mas... e as decisões com base em proposições lógicas? Onde ficam? Vamos analisar um outro exemplo, um algoritmo de divisão de dois números, onde ocorre uma decisão:

Passo 1: Receber o dividendo e o divisor.
Passo 2: Se o divisor for igual a zero, imprime mensagem de erro e finaliza.
Se o divisor for diferente de zero, calcula divisão.
Passo 3: Mostra resultado da divisão

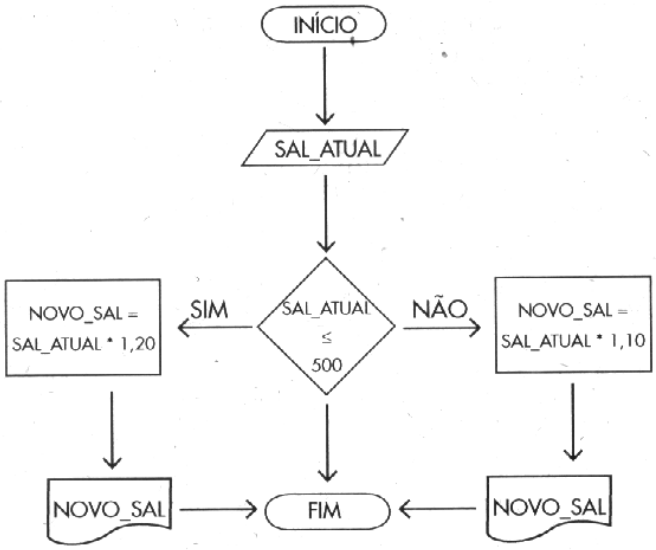
Repare que agora surge uma palavra nova nesse algoritmo: **se**. É essa palavra que reprenta a decisão lógica. Observe o fluxograma gerado a partir do algoritmo acima:



Após o início, os números N1 e N2 são solicitados ao usuário. Em seguida vem uma decisão com base em uma proposição: "N2 = 0" for **verdadeira** (SIM), o fluxo segue paraa a impressão da mensagem de erro: "IMPOSSÍVEL DIVIDIR" e o algoritmo chega ao final. Caso a proposição "N2 = 0" seja **falsa** (NÃO), a divisão é feita, com o resultado armazenado na variável D, que é posteriormente impressa e o algoritmo chega ao fim.

Consideremos agora o caso de uma empresa que pretende aumentar os salários de seus funcionários. Se o salário de um funcionário for de até R$500,00, ele receberá um aumento de 20%. Se, por outro lado, o salário for superior a R$500,00, o aumento será de apenas 10%. Vejamos o algoritmo para calcular o novo salário dos funcionários:

Passo 1: Receber o salário atual de um funcionário.
Passo 2: Se salário atual é de até R$500,00, calcular salário ampliado de 20%.
Caso contrário, calcular salário ampliado de 10%.
Passo 3: Mostrar o novo salário.



3.2. DIAGRAMA DE CHAPIN

O diagrama de Chapin é uma forma alternativa de representar algoritmos, que usa uma tabela retangular para indicar todo o processo. Um exemplo está representado abaixo:

O diagrama de Chapin é prático para algoritmos pequenos mas, para casos mais complexos, com várias decisões lógicas em sequência, ele se torna um tanto desajeitado, visto que sua largura tende a crescer com o número de decisões aninhadas.

4. O PORTUGUÊS ESTRUTURADO OU PORTUGOL

O português estruturado é uma forma de representar um algoritmo de um maneira simplificada e, em certo nível, universal. Escrever um código em *portugol* é como programar em uma linguagem específica e, como veremos, ela pode até mesmo ser executada.

Outra característica da linguagem portugol é que um algoritmo escrito neste formato pode ser facilmente ser convertido para quase qualquer outra linguagem de programação, o que a torna muito útil no aprendizado de algoritmos. Abaixo segue o algoritmo que calcula a média de dois números, escrito em *portugol*:

```
1     Algoritmo "Cálculo da Média Aritmética"
2     VAR
3     N1,N2,M : REAL
4     Inicio
5     Escreva ("Programa que calcula a média de dois valores.")
6     Escreva ("Digite o primeiro valor : ")
7     Leia (N1)
8     Escreva ("Digite o segundo valor : ")
9     Leia (N2)
10    M <- (N1+N2)/2
11    Escreva ("A média dos dois valores é : ", M)
12    FimAlgoritmo
```

Os números de linha foram crescentados apenas para facilitar a explicação abaixo!

Todo algoritmo em português recebe um nome, como pode ser visto na linha 1. Para isso, usamos a palavra **Algoritmo** para especificar onde ele começa, seguido do texto com o nome entre aspas. Se usamos uma palavra para indicar o início do algoritmo, também usamos uma para indicar o fim: **FimAlgoritmo**, como na linha 12.

Uma das partes burocráticas do português estruturado - assim como a grande maioria das linguagens, é que temos de definir quais variáveis usaremos no algoritmo já no início do programa. Para isso, usamos a palavra **VAR**, como na linha 2. As próximas linhas devem ser usadas para declarar os nomes das variáveis que iremos usar, separadas por vírgula.

Na linha 3, portanto, encontramos a declaração das variáveis **N1**, **N2** e **M** - observe que não se usa "espaço", acentos ou quaisquer caracteres que não sejam **letras e números** nos nomes das variáveis. Logo em seguida, depois dos nomes das variávis, aparece um sinal de "dois pontos" e a palavra **REAL**. Isso significa o **tipo de dado** que iremos guardar nessas variáveis. Real é qualquer número, fracionário ou não, negativo ou positivo. Os tipos de dados possíveis no *portugol* são: REAL, INTEIRO, CARACTERE e LOGICO. O tipo INTEIRO não armazena números fracionários; o tipo CARACTERE serve para guardar textos e o tipo LOGICO serve para guardar apenas informações do tipo FALSO ou VERDADEIRO.

NOTA: Para entender a importância de declarar o tipo de variável é preciso relembrar a aula de Organização de Computadores: lembre-se que cada posição de memória só é capaz de armazenar 8 bits que podemos interpretar das mais variadas formas. A mais comum é interpretá-los como um número de 1 byte, isto é, de 0 a 255; Porém, se considerarmos um bit de sinal, podemos interpretá-lo como um número de -128 a 127. Também podemos intepretá-lo como uma letra, seguindo o código ASCII...

Uma vez que, na prática, uma variável é um "apelido" que damos para uma ou mais posições de memória, onde podemos armazenar valores, definir o TIPO da variável significa instruir o computador sobre **como ele deve interpretar os bits** que lá estão! Assim, é **fundamental** definir o tipo de cada variável!

Como podemos ter várias linhas dedicadas à declaração de variáveis, é usada a palavra **Inicio**, como na linha 4, para indicar o fim das declarações de variáveis e o início do algoritmo propriamente dito.

As linhas 5 e 6, com o comando **Escreva**, não têm função para a execução do cálculo em si, mas elas tem a função primordial de informar ao usuário **o que ele deve digitar** no computador, e que o computador está esperando que ele digite algo!

A linha 7 contém o comando **Leia**, é usada para ler um valor digitado pelo usuário e armazená-lo na variável dentro do parênteses; no caso, a variável N1.

A linha 8 informa ao usuário que ele digite o segundo valor e a linha 9 colhe o valor da variável N2.

Na linha 10, aparece um símbolo diferente: **<-** . Este símbolo pode ser lido como "recebe". A leitura da linha 10 é: "M recebe (N1+N2)/2", em uma operação chamada "atribuição": É atribuído à varivel M o valor de (N1+N2)/2. Toda linguagem de programação, ao processar uma atribuição, resolve primeiro integralmente a operação à DIREITA do sinal de atribuição - no caso (N1 + N2)/2 - e, só então, armazena o resultado na variável, que deve estar representada à ESQUERDA da atribuição.

> Nota: Em uma atribuição, o valor do lado ESQUERDO **sempre** será uma variável!

Finalmente, o comando **Escreva** é usado de uma maneira um pouco diferente para imprimir, além de uma mensagem texto, também o valor de uma variável.

5. BIBLIOGRAFIA

ASCENCIO, A.F.G; CAMPOS, E.A.V. **Fundamentos da Programação de Computadores**. 2ed. Rio de Janeiro, 2007.

MEDINA, M; FERTIG, C. **Algoritmos e Programação: Teoria e Prática**. 2ed. São Paulo: Ed. Novatec, 2006.

SILVA, I.C.S; FALKEMBACH, G.M; SILVEIRA, S.R. **Algoritmos e Programação em Linguagem C**. 1ed. Porto Alegre: Ed. UniRitter, 2010.

SILVA, L.A.M. **Material de Apoio: Algoritmos**. UniNorte, 2006. Disponível em: < http://www.luizmatos.eti.br/disciplinas/docs/algoritmos/Capitulos_1-2.pdf >.

TONET, B; KOLIVER, C. **Introdução aos Algoritmos**. Apostila da Universidade de Caixias do Sul usada como manual do VisuAlg.