LÓGICA E PROGRAMAÇÃO DE ALGORITMOS

AULA 2

Prof. Sandro de Araújo

CONVERSA INICIAL

Esta aula tem como base os livros *Lógica de programação e estruturas de dados* (Rissetti, 2016) e *Lógica de programação algorítmica* (Guedes, 2014). Em caso de dúvidas ou se desejar aprofundar as informações, consulte as publicações em nossa Biblioteca Virtual Pearson.

A aula apresenta a seguinte estrutura de conteúdo:

1. Definição de dados;
2. Tipos de dados;
3. Variáveis;
4. Constantes;
5. Operadores.

O objetivo desta aula é conhecer os conceitos de dados, os tipos de dados e seus desdobramentos; compreender o conceito, a aplicação e a identificação de variáveis e constantes; utilizar operadores de atribuição, aritméticos, relacionais e lógicos, tanto na notação algorítmica quanto na linguagem de programação C; dominar a construção de expressões de atribuição, aritméticas e lógicas; perceber a ordem de precedência matemática utilizada na resolução de problemas; reconhecer a tabela-verdade, recurso que facilita o entendimento do uso dos operadores lógicos.

TEMA 1 – DEFINIÇÃO DE DADOS

Puga e Rissetti (2016) definem o dado como uma sequência de símbolos quantificados ou quantificáveis. São valores fornecidos pelo usuário do programa, podendo ser obtidos a partir de processamentos, arquivos, banco de dados ou outros programas. Os estudos dos algoritmos baseiam-se nesse conceito de dados, pois para que ocorra a resolução de um problema faz-se necessária a escolha de representação da realidade, geralmente definida em termos de dados e suas representações, os quais são armazenados temporariamente em variáveis para que sejam processados de acordo com as especificações do algoritmo.

Para garantir a integridade do resultado obtido com o processamento, os dados devem ser classificados de acordo com o tipo do valor a ser armazenado

na variável, evitando problemas ocasionados pelo fornecimento de valores inadequados à operação realizada (Guedes, 2014; Puga; Rissetti, 2016).

Cada tipo de dado tem uma exigência pré-determinada quanto ao tamanho de memória com uma faixa associada de valores permitidos e armazenados em uma posição, onde poderemos guardar um determinado dado ou valor e modificá-lo ao longo da execução do programa de acordo com as especificações do algoritmo.

TEMA 2 – TIPOS DE DADOS

Os tipos de dados costumam ser definidos a partir dos tipos primitivos criados em função das características dos computadores. Após especificar o tipo do dado, todo o trabalho de guardá-lo na memória fica para o compilador. Esses tipos de dados seguem as características de formato e estrutura definidas para essa memória (Puga; Rissetti, 2016).

Definir o tipo de dado mais adequado para ser armazenado em uma variável é importante para garantir que o problema seja resolvido. Ao desenvolver um algoritmo, é necessário que se tenha conhecimento prévio do tipo de dado que será utilizado para resolver o problema proposto. A partir daí, escolhe-se o tipo de dado adequado para a variável que representará esse valor.

Apesar de internamente o computador manipular unicamente números, as linguagens de programação oferecem suporte para outros tipos de dados de forma transparente. Alguns tipos são formados por números inteiros e reais que suportam operações matemáticas como adição, subtração, multiplicação, entre outros. Esses tipos são particularmente importantes, visto que o computador trabalha naturalmente com números. Também podemos considerar as letras como um tipo de dado, sobre as quais poderíamos definir operações como escrever, ler, concatenar, entre outros (Medina; Ferting, 2006).

Medina e Ferting (2006) informam que a construção de algoritmos para o computador deve lidar exclusivamente com os tipos definidos nas linguagens de programação, por estas serem limitadas e não poderem manipular todos os tipos de dados existentes no mundo real. A maioria das linguagens de programação tipifica os dados em um grupo conhecido como *tipos primitivos*: **números inteiros**, **números reais**, **letras** e **objeto lógicos**, descritos a seguir.

2.1 Tipos primitivos

Tipos primitivos de dados são grupos de combinação de valores e de operações que uma variável pode executar, o que pode variar conforme o sistema operacional e a linguagem de programação (Guedes, 2014). Os mais utilizados são: numérico, lógico e literal ou caracteres, descritos a seguir.

- **Numéricos:** dividem-se em dois grupos: inteiros e reais. Os inteiros podem ser positivos e negativos, e não apresentam parte fracionária, como -357, -23, 0, 98, 237. Os reais também podem ser positivos e negativos, e apresentam parte fracionária, como - 23.45, -5.6, 0.0, 32.55, 222.02. Os números reais seguem a notação da língua inglesa, ou seja, a parte decimal é separada da parte inteira por um ponto, não por uma vírgula.
- **Lógicos:** são também chamados de *booleanos* e podem assumir os valores *verdadeiro* ou *falso*, utilizando-se o número 0 para falso e 1 para verdadeiro.
- **Literais ou caracteres:** são dados formados por um único caractere ou por uma cadeia de caracteres. Esses caracteres podem ser o alfabeto, com maiúsculas e minúsculos; os números, que não poderão ser usados para cálculos, pois não são valores; e os caracteres especiais, como @,#,$,?, +. Exemplos de literais: “aluno”, “12345”, “@ internet”, “0,56”, ‘S’, ‘6’. Nesses exemplos, o conjunto de caracteres é representado entre aspas duplas (“ ”) e um único caractere é representado entre aspas simples (‘ ’).

2.1.1 Tipos Primitivos na Linguagem C

Para a linguagem de programação C existem cinco tipos de dados primitivos pré-definidos. A Tabela 1 mostra quais são os dados primitivos na linguagem C e o seus tipos.

Tabela 1 – Dados primitivos na linguagem C

Dado Primitivo	Tipo
char	caracter
int	inteiro
float	real de precisão simples
double	real de precisão dupla
void	vazio (sem valor)

A faixa de valores e o tamanho dos dados sempre dependerá do compilador e do computador utilizado. A Tabela 2 mostra o tamanho e o intervalo que cada dado primitivo pode armazenar.

Tabela 2 – Tamanho e intervalo dos dados primitivos na linguagem C

Dado primitivo	Tipo	Tamanho	Intervalo
char	caracter	1	-128 a 127
int	inteiro	2	-32.768 a 32.767
float	real de precisão simples	4	3.4 E-38 a 3.4E+38
double	real de precisão dupla	8	1.7 E-308 a 1.7E+308
void	vazio (sem valor)	-	-

TEMA 3 – VARIÁVEIS

Uma variável pode ser entendida como uma posição identificada na memória que contém dados que podem ser modificados durante a execução do programa, podendo assumir qualquer valor de um conjunto de valores (Guedes, 2014; Puga; Rissetti, 2016).

A altura de uma pessoa, a cotação do *bitcoin* e a velocidade de um carro são exemplos de variáveis comuns no nosso dia a dia. Nesses exemplos, os valores dos dados sofrem alterações ou são dependentes da execução em certo instante ou circunstância. Nos algoritmos segue-se o mesmo conceito: as variáveis são utilizadas para representar valores desconhecidos, porém necessários, para a resolução de um problema, podendo ser alterados de acordo com a situação (Puga; Rissetti, 2016).

3.1 Identificação das variáveis para os algoritmos

Segundo Puga e Rissetti (2016), toda variável deve ser identificada, isto é, deve receber um nome ou identificador. O nome de uma variável deve ser único e estar de acordo com algumas regras:

- Não utilizar espaços entre as letras. Por exemplo, em vez de nome do cliente, o correto seria nome_do_cliente ou nomeDoCliente. O caractere *underline*, representado por _, pode ser utilizado para substituir o espaço entre as letras.
- Não iniciar o nome da variável com algarismos (números). Por exemplo: não usar 2valor, o correto seria valor2.

- Não utilizar palavras reservadas, isto é, palavras que são utilizadas nos algoritmos para representar ações específicas. Por exemplo:
  - ✓ se: palavra que representa uma condição ou teste lógico;
  - ✓ var: palavra que representa a área de declaração de variáveis.
- Não utilizar caracteres especiais, como acentos, símbolos (?/:@# etc.), ç, entre outros.
- Não utilizar nomes iguais para representar variáveis diferentes
- Ser conciso e utilizar nomes coerentes.

Vale lembrar que cada linguagem de programação tem sua particularidade para a declaração de variáveis. Essas particularidades devem ser conhecidas e observadas, quando da atribuição dos nomes.

TEMA 4 – CONSTANTES

Uma constante segue as mesmas regras de variável, mas com a certeza de que o dado ou valor não será alterado durante a execução do programa, ou seja, será sempre o mesmo, sendo obrigatória a atribuição de um valor no momento da declaração.

Um exemplo de uma constante matemática é o número Pi, que é um valor fixo de aproximadamente 3,1415 e que continuará assim até o final da execução (Puga; Rissetti, 2016).

TEMA 5 – OPERADORES

Os operadores são utilizados para representar expressões de cálculo, comparação, condição e atribuição. Para a construção de algoritmos temos os seguintes tipos de operadores: 1. atribuição; 2. aritméticos; 3. relacionais; 4. lógicos (Puga; Rissetti, 2016).

5.1 Operadores de atribuição

Um dos operadores mais utilizadas na programação é o operador de atribuição, representado no pseudocódigo pela seta ← . Para o exemplo a seguir adotaremos a representação pseudocódigo.

- **nomeDaVariavel** ← expressão;

  Exemplos:

✓ nomeDoCliente ← “Joãozinho da Silva”;

✓ resultado ← a + 5;

✓ valor ← 3.5;

Na linguagem C, o sinal de atribuição é representado pelo **=**, conforme exemplo a seguir.

- **nomeDaVariavel** = expressão;

  Exemplos:

  ✓ nomeDoCliente **=** “Joãozinho da Silva”;

  ✓ resultado = a + 5;

  ✓ valor = 3.5;

5.2 Operadores aritméticos

Chamamos de *operadores aritméticos* o conjunto de símbolos que representa as operações básicas da matemática (Forbellone; Eberspacher, 2005). Veja na Tabela 3:

Tabela 3 – Operadores aritméticos

Operador	Notação Algorítmica
Incremento	Utiliza-se uma expressão. Exemplo: a ← a + 1
Decremento	Utiliza-se uma expressão. Exemplo: a ← a - 1
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	^ ou ** Exemplo: pot ← 2**3 ou 2^3 raiz ← 4**(½) ou raiz ← 4^(½)
Módulo	Mod Exemplo: resto ← a mod b

Fonte: Adaptado de Guedes, 2014.

Nem todos os operadores aritméticos utilizados na realização de cálculos podem ser representados diretamente por símbolos computacionais. Alguns deles são representados por funções matemáticas, como no caso da exponenciação e da radiação (Guedes, 2014; Puga; Rissetti, 2016).

5.3 Operadores relacionais

Os operadores relacionais são aqueles que comparam dois valores (valores, variáveis, constantes ou chamadas de funções) e/ou expressões e verificam quem é maior ou menor e/ou se há igualdade entre eles. O resultado dessa comparação é sempre um valor lógico (booleano) **verdadeiro** ou **falso** (Guedes, 2014; Puga; Rissetti, 2016), conforme a Tabela 4:

Tabela 4 – Operadores relacionais

Operador	Representação algorítmica	Notação para C	Descrição para C
Maior que	>	>	x > y: Se o valor de x for maior do que o de y, retorna verdadeiro; se não, falso.
Maior ou igual	>=	>=	x >= y: Se o valor de x for maior ou igual ao de y, retorna verdadeiro; se não, falso.
Menor que	<	<	x < y: Se o valor de x for menor do que o de y, retorna verdadeiro; se não, falso.
Menor ou igual	<=	<=	x <= y: Se o valor de x for menor ou igual ao de y, retorna verdadeiro; se não, falso.
Igual a	=	==	x == y: Se o valor de x for igual ao de y, retorna verdadeiro; se não, falso.
Diferente de	<>	!=	x != y: Se o valor de x for diferente ao de y, retorna verdadeiro; se não, falso.

Fonte: Adaptado de Puga; Risseti, 2016.

Como resultado dessas operações, teremos como retorno:

- O valor UM (1), se a expressão relacional for verdadeira;
- O valor ZERO (0), se a expressão relacional for falsa.

Um erro bastante comum na programação em linguagem C é confundir o operador de atribuição = com o operador de comparação ==. O operador de atribuição é definido por **UM** símbolo de igual (=), enquanto o operador de comparação é definido por **DOIS** símbolos de igual (==). Quando colocamos o operador de **comparação** em uma operação de **atribuição**, o compilador apresentará um erro. Esse erro não aparece quando colocado o operador de **atribuição** = no lugar do operador de **comparação** ==.

5.4 Operadores lógicos

Operadores lógicos são usados para concatenar ou associar expressões que estabelecem uma relação de comparação entre valores. O resultado dessas expressões é sempre um valor lógico, verdadeiro ou falso, uma vez que operam sobre valores booleanos. Para saber se uma determinada variável está dentro

de uma faixa de valores deve-se criar expressões com outros operadores, além dos operadores aritméticos e/ou relacionais. Por exemplo, a expressão matemática 0 < x < 20 indica que o valor de x deve ser maior do que 0 (zero) e menor do que 20 (Guedes, 2014; Puga; Rissetti, 2016).

A Tabela 5 mostra um conjunto de três operadores lógicos para modelar situações como essas.

Tabela 5 – Operadores lógicos

Operador	Representação algorítmica	Notação para C	Descrição para linguagem C
E	**.e.**	&&	realiza a operação E, exemplo: (x >= 0 && x <= 8)
OU	**.ou.**	||	realiza a operação OU, exemplo: (a == 'G' || b != 33)
NÃO	**.não.**	!	realiza a operação NÃO, exemplo: !(x == 11)

Fonte: Adaptado de Puga; Risseti, 2016.

Esses operadores permitem representar situações lógicas unindo duas ou mais expressões relacionais simples numa composta:

- **Operador E (&&)**: a expressão resultante só é verdadeira se ambas as expressões também forem verdadeiras. Por exemplo, a expressão (x >= 0 && x <= 8) será verdadeira somente se as expressões (x >= 0) e (x <= 8) forem verdadeiras.

  Diante da sentença "José é professor e Paula é estudante", só poderemos concluir que a proposição composta é verdadeira se as duas afirmações forem verdadeiras. Do mesmo modo, a expressão x >= 0 && x <= 8 só será verdadeira se o valor de x for maior ou igual a zero e ao mesmo tempo o valor de x também for menor ou igual a oito. Basta que uma ou as duas proposições componentes sejam falsas, e toda conjunção será falsa.

  Na Tabela 6 veremos o exemplo do José e Paula usando o operador **.e.**.

Tabela 6 – Operador lógico .e.

José é professor	Paula é estudante	José é professor **.e.** Paula é estudante
V	V	Verdadeiro
F	V	Falso
V	F	Falso
F	F	Falso

- **Operador OU (||)**: a expressão só será verdadeira se alguma das expressões unidas por esse operador também for verdadeira. Por exemplo, a expressão (a == 'G' || b != 33) será verdadeira se uma de suas duas expressões, (a == 'G') ou (b != 33), for verdadeira.

  Diante da sentença "José é professor ou Paula é estudante", só poderemos concluir que a proposição composta é verdadeira se, pelo menos, uma das duas afirmações for verdadeira. Do mesmo modo, a expressão x >= 0 || x <= 8 só será verdadeira se o valor de x for maior ou igual a zero ou se o valor de x for menor ou igual a oito. Basta que uma das duas proposições componentes seja verdadeira para que toda conjunção seja verdadeira.

  Na Tabela 7 veremos o exemplo de José e Paula usando o operador **.ou.**.

Tabela 7 – Operador lógico .ou.

José é professor	Paula é estudante	José é professor **.ou.** Paula é estudante
V	V	Verdadeiro
F	V	Verdadeiro
V	F	Verdadeiro
F	F	Falso

- **Operador NEGAÇÃO .não. (!):** inverte o valor lógico da expressão. Por exemplo, a expressão !(x == 11) se transforma em (x > 11 | | x < 11).

  Veremos algo de suma importância: como negar uma proposição.

  No caso de uma proposição simples se tornar uma negação, basta pôr a palavra *não* antes da sentença e já a tornamos uma negativa.

  Exemplos:

  José é professor. Negativa: José **não** é professor.

  Paula é estudante. Negativa: Paula **não** é estudante.

  Reparemos que, caso a sentença original já seja uma negativa (já traga a palavra *não*), então para negar a negativa teremos que excluir a palavra *não*. Assim:

  José não é professor. Negativa: José é professor.

  Paula não é estudante. Negativa: Paula é estudante.

  Na Tabela 8 temos a tabela-verdade com o operador lógico de negação.

Tabela 8 – Operador lógico .não.

X é igual a 11	.não. (X é igual a 11)
x == 11	**!(x == 11)**
Verdadeiro	Falso
Falso	Verdadeiro

Os operadores lógicos atuam sobre valores lógicos e retornam um valor lógico:

- 1: se a expressão for verdadeira;
- 0: se a expressão for falsa.

A Tabela 9 apresenta a tabela-verdade, onde os termos a e b representam duas expressões relacionais.

Tabela 9 – Tabela-verdade

a	b	a . e. b (a && b)	a .ou. B (a | | b)	.não. a
falso	falso	falso	falso	verdadeiro
falso	verdadeiro	falso	verdadeiro	verdadeiro
verdadeiro	falso	falso	verdadeiro	falso
verdadeiro	verdadeiro	verdadeiro	verdadeiro	falso

Fonte: Puga; Risseti, 2016.

Observando a Tabela 9, nota-se que, para o operador .e., o resultado será verdadeiro, que corresponde a UM (1) somente se ambas as variáveis associadas assumirem o resultado verdadeiro. Já para o operador .ou. a condição de verdadeiro se dá, pelo menos, se uma das expressões assumir o resultado verdadeiro.

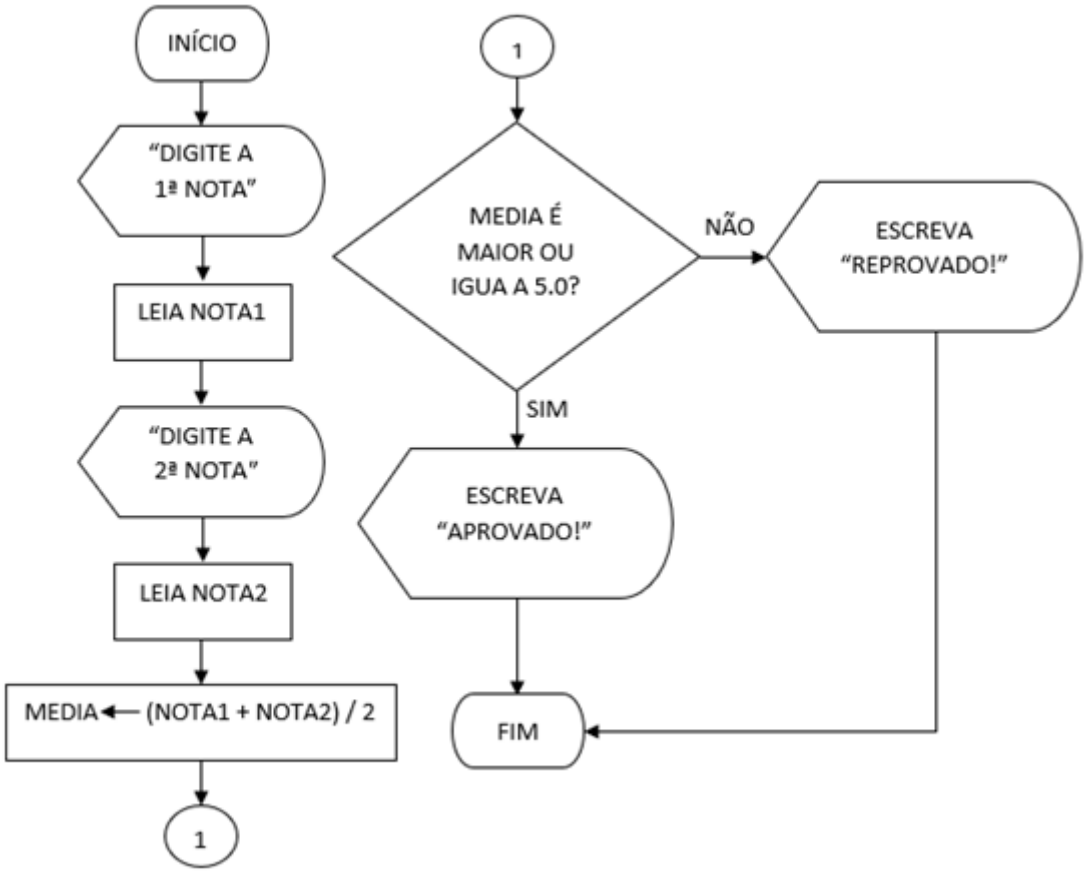
Na tabela-verdade é expresso o conjunto de possibilidades existentes para a combinação de variáveis ou expressões e operadores lógicos.

A seguir apresentam-se exemplos em fluxograma, pseudocódigo e linguagem C para contextualizar o que aprendemos até o momento.

Considere um algoritmo que irá receber dois dados numéricos reais – as duas notas de que o aluno precisará para passar de ano. Ao fazer a média dos dois números e o resultado for maior ou igual a 5,0, o algoritmo vai imprimir uma mensagem de *aprovado*, caso contrário, a mensagem que vai aparecer será *reprovado*.

Já sabemos que, para resolver o problema, teremos dois dados de entrada e um de saída, o qual dependerá do que for processado. Para os dois dados de entrada vamos criar duas variáveis (NOTA1 e NOTA2) e para o dado de saída vamos criar uma variável (MEDIA). Agora analise o fluxograma a seguir.

Figura 1 – Fluxograma



A Tabela 10 vai detalhar cada passo dado no fluxograma para resolver o problema.

Tabela 10 – Passos do fluxograma

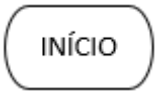
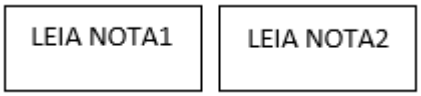
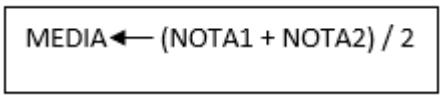
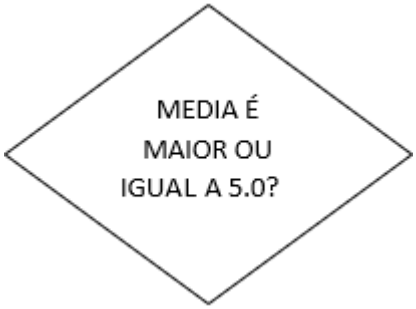
	Marca o início do algoritmo.
	Comando para imprimir mensagem na tela do usuário.
	Comando para capturar (ler) os dados que o usuário digitou e guarda-los nas variáveis NOTA1 e NOTA2.
	Processamos dos dois dados e atribuição do resultado em uma variável com o nome de MEDIA.
	Conector (Unir o fluxograma).
	Essa figura geométrica representa decisão. A resposta dependerá do resultado gerado após a comparação. Se a MEDIA for maior ou igual executará uma ação, se não for maior ou igual executará outra ação.
	Comando para imprimir uma mensagem na tela do usuário. Nesse caso temos duas mensagens (APROVADO! e REPROVADO!) e apenas uma será exibida após o resultado da comparação.
	Marca o final do algoritmo.

Figura 2 – Pseudocódigo

```
algoritmo "CalcularMedia"

var nota1, nota2, media: real

inicio

    escreval("Digite a 1ª nota: ")
    leia(nota1)
    escreval("Digite a 2ª nota: ")
    leia(nota2)

    media ← (nota1 + nota2) / 2

    Se (media >= 5) então
        escreval("APROVADO!")
    Senao
        escreval("REPROVADO!")
    Fimse

fimalgoritmo
```

A Tabela 11 vai detalhar cada passo dado no pseudocódigo para resolver o problema.

Tabela 11 – Passos do pseudocódigo

algoritmo "CalcularMedia"	Nome do algoritmo e início do algoritmo.
var **nota1**, **nota2, media**: real	Declaração das variáveis que irão guardar os dados.
inicio	Início da Seção de Comandos
escreval("**Digite a 1ª nota:** ") escreval("**Digite a 2ª nota:** ")	Comando (escreval) para imprimir mensagem na tela do usuário.
leia(**nota1**) leia(**nota2**)	Comando (leia) para capturar (ler) os dados que o usuário digitou e guarda-los nas variáveis nota1 e nota2.
media ← (nota1 + nota2) / 2	Processamos dos dois dados e atribuição do resultado em uma variável com o nome de media. **Observação**: Variável sempre em minúsculo; a seta é um comando de atribuição ←.
Se (**media** >= **5.0**) então	Tomada de decisão após comparação dos dados. Compara-se o conteúdo da variável media com o número 5.0 usando o operador relacional >=.
escreval("**APROVADO!**") escreval("**REPROVADO!**")	Comando para imprimir uma mensagem na tela do usuário. Nesse caso temos duas mensagens (APROVADO! e REPROVADO!) e apenas uma será exibida após o resultado da comparação.
fimalgoritmo	Marca o final do algoritmo.

Figura 3 – Linguagem C

```
#include <stdio.h>
int main()
{
 float nota1, nota2, media;
printf("Digite a 1ª nota: \n");
scanf("%f", &nota1);
printf("Digite a 1ª nota: \n");
            scanf("%f", &nota2);

media = (nota1 + nota2) /2;

if (media >= 5)
{
            printf("APROVADO!\n");
}
else
{
             printf("REPROVADO! \n”);
 }

system ("pause");
return(0);
}
```

A Tabela 12 vai detalhar cada passo dado no algoritmo representado na linguagem C para resolver o problema. Não vamos entrar nos detalhes da sintaxe da linguagem C por não ser o objetivo da disciplina. Na Tabela 12 dividiremos o código de uma maneira que possa facilitar a sua compreensão.

Tabela 12 – Passos do algoritmo na linguagem C

#include <stdio.h> int main() {	Carregar a biblioteca stdio e marca o início do algoritmo.
float nota1, nota2, media;	Declaração das variáveis que irão guardar os dados do tipo float.
printf("**Digite a 1ª nota:** \n"); printf("**Digite a 2ª nota:** \n");	Comando (printf())para imprimir mensagem na tela do usuário.
scanf("%f", &nota1); scanf("%f", &nota2);	Comando (scanf()) para capturar (ler) os dados que o usuário digitou e guarda-los nas variáveis nota1 e nota2.
media = (nota1 + nota2) /2;	Processamos dos dois dados e atribuição do resultado em uma variável com o nome de media. **Observação:** Variável sempre em minúsculo e a seta é um comando de atribuição =.

if (media >= 5)
{
}
else
{
}
	Tomada de decisão após comparação dos dados. Compara-se o conteúdo da variável media com o número 5.0 usando o operador relacional >=.

printf("**APROVADO!**\n"); printf("**REPROVADO!** \n”);	Comando (printf()) para imprimir uma mensagem na tela do usuário. Nesse caso temos duas mensagens (APROVADO! e REPROVADO!) e apenas uma será exibida após o resultado da comparação.

system ("pause");
return(0);
}
	Pausa a tela mostrando o resultado, retorna 0 para o compilador caso não tiver erro e finaliza o algoritmo.

FINALIZANDO

Nesta aula aprendemos o conceito de dados e os tipos básicos usados na construção de algoritmos. Compreendemos o conceito, a aplicação e a identificação de variáveis e constantes, e vimos como utilizar operadores de atribuição, aritméticos, relacionais e lógicos, tanto na notação algorítmica quanto na linguagem de programação C.

REFERÊNCIAS

FOBERLLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**: a construção de algoritmos e estrutura de dados. São Paulo: Pearson, 2005.

GUEDES, S. **Lógica de programação algorítmica**. São Paulo: Pearson, 2014.

MEDINA, M.; FERTING, C. **Algoritmos e programação**: teoria e prática. São Paulo: Novatec, 2006.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados**. São Paulo: Pearson, 2016.