



# Interativa

## Lógica de Programação e Algoritmos

**Autora:** Profa. Eliane Oliveira Santiago

**Colaboradores:** Profa. Vanessa Santos Lessa

Prof. Tiago Guglielmeti Correale

## Professora conteudista: Eliane Oliveira Santiago

É mestra em Engenharia Eletrônica e Computação pelo Instituto Tecnológico de Aeronáutica (2009), especialista em Engenharia de Projetos de Sistemas de Informação pela Faculdade Carlos Drummond de Andrade (2008) e bacharel em Ciência da Computação pela Universidade de Mogi das Cruzes (1996), com pesquisas na área de computação semântica e ontologias aplicadas em big data.

Atua na Educação Superior há 20 anos, com experiência no magistério superior nas áreas de ciência da computação e sistemas de informação com ênfase em inteligência artificial, engenharia de software, programação orientada a objetos e engenharia da informação (banco de dados e representação do conhecimento). É avaliadora do Inep/MEC desde 2011.

### Dados Internacionais de Catalogação na Publicação (CIP)

S235I Santiago, Eliane Oliveira.

Lógica de Programação e Algoritmos / Eliane Oliveira Santiago.  
– São Paulo: Editora Sol, 2021.

204 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Estrutura. 2. Operadores. 3. Algoritmos. I. Título.

CDU 517

U512.93 – 21

Prof. Dr. João Carlos Di Genio

**Reitor**

Profa. Dra. Marília Ancona Lopez

**Vice-Reitora de Graduação**

**Vice-Reitora de Pós-Graduação e Pesquisa**

Prof. Fábio Romeu de Carvalho

**Vice-Reitor de Planejamento, Administração e Finanças**

Profa. Melânia Dalla Torre

**Vice-Reitora de Unidades Universitárias**

### **Unip Interativa**

Profa. Dra. Cláudia Andreatini

Profa. Elisabete Brihy

Prof. Marcelo Vannini

Prof. Dr. Luiz Felipe Scabar

Prof. Ivan Daliberto Frugoli

### **Material Didático**

Comissão editorial:

Profa. Dra. Christiane Mazur Doi

Profa. Dra. Angélica L. Carlini

Profa. Dra. Ronilda Ribeiro

Apoio:

Profa. Cláudia Regina Baptista

Profa. Deise Alcantara Carreiro

Projeto gráfico:

Prof. Alexandre Ponzetto

Revisão:

Giovanna Oliveira

Vitor Andrade



# Sumário

## Lógica de Programação e Algoritmos

APRESENTAÇÃO .....	7
INTRODUÇÃO .....	7

### Unidade I

1 ALGORITMOS .....	11
1.1 Tipos de representação de algoritmos.....	12
1.1.1 Descrição narrativa: linguagem natural.....	12
1.1.2 Fluxograma.....	13
1.1.3 Pseudocódigo .....	15
1.2 A lógica de programação: entendendo como dados são processados e onde são armazenados.....	18
1.3 Tipos de dados.....	21
1.4 Variáveis e constantes.....	21
1.5 Identificadores.....	26
2 SOLUÇÕES DE PROBLEMAS.....	28
2.1 Comandos de entrada e saída.....	28
2.2 Fazendo operações com dados.....	31
2.2.1 O operador de atribuição (←) .....	31
2.2.2 Operadores aritméticos.....	32
2.2.3 Operadores relacionais.....	37
2.2.4 Operadores lógicos.....	38
2.2.5 Operador lógico NÃO .....	39
2.2.6 Operador lógico E .....	40
2.2.7 Operador lógico OU .....	41
2.3 Prioridades entre as expressões aritméticas, lógicas e relacionais.....	45
2.4 O ambiente de desenvolvimento integrado .....	49
2.5 Comentários no corpo do programa.....	50
2.6 Blocos de programação e indentação .....	51
2.7 Instâncias do problema.....	52
2.8 Algoritmos sequenciais.....	52

### Unidade II

3 ESTRUTURAS DE DECISÃO .....	61
3.1 Estrutura de decisão simples – SE... ENTÃO.....	61
3.2 Estrutura de decisão composta – SE... SENÃO .....	65

3.3 Estrutura de decisão encadeada SE... SENÃO .....	68
3.4 Estrutura de decisão ESCOLHA-CASO .....	78
3.5 Estrutura de decisão encadeada ESCOLHA-CASO .....	82
3.6 Algoritmos de decisão .....	84
4 ESTRUTURAS DE REPETIÇÃO .....	92
4.1 Estrutura de repetição simples com teste no início (Laço ENQUANTO) .....	93
4.1.1 Estrutura de repetição encadeada com teste no início .....	100
4.2 Estrutura de repetição com teste no fim (Laço FAÇA) .....	103
4.2.1 Estrutura de repetição encadeada com teste no fim .....	104
4.3 Estrutura de repetição com controle de iterações (Laço PARA) .....	109
4.4 Algoritmos com estruturas de repetição .....	114

### Unidade III

5 PROCEDIMENTOS .....	128
5.1 Procedimento sem parâmetros .....	131
5.2 Procedimento com parâmetro .....	137
5.3 Chamada de procedimento com ou sem parâmetro .....	139
6 FUNÇÕES .....	141
6.1 Função sem parâmetro .....	141
6.2 Função com parâmetro .....	143
6.3 Chamada de funções com e sem parâmetro .....	147
6.4 Escopo da variável .....	150
6.5 Exemplos de algoritmos com procedimentos e funções .....	150

### Unidade IV

7 ESTRUTURAS DE ARMAZENAMENTO DE DADOS .....	168
7.1 Vetores .....	169
7.1.1 Declaração de variáveis compostas homogêneas unidimensionais: vetores .....	169
7.1.2 Atribuindo valores a vetores .....	170
7.1.3 Pesquisando um determinado elemento no vetor .....	172
7.1.4 Escrevendo os elementos de um vetor .....	173
7.2 Matrizes .....	176
7.2.1 Declaração de variáveis compostas homogêneas bidimensionais: matrizes .....	177
7.2.2 Atribuindo valores a matrizes .....	179
7.2.3 Pesquisando um determinado elemento na matriz .....	181
7.2.4 Escrevendo os elementos de uma matriz .....	182
8 APLICAÇÕES COM VETORES E MATRIZES .....	183
8.1 Ordenando vetores .....	192

## APRESENTAÇÃO

A construção de sistemas de informações e sistemas computacionais requer reflexões sobre processos, alocação ou compartilhamento de recursos e usabilidade, a fim de oferecer soluções para problemas, as quais dependem de bons projetos de algoritmos.

É necessário entender domínios diversificados de aplicação, compreender necessidades de negócios e organizacionais para criar sistemas que atendam as necessidades de informação e automatizem os processos gerenciais e de negócios. A construção de algoritmos requer conhecimento sobre a lógica do negócio ou das operações, bem como técnicas para estruturar uma solução.

O objetivo deste livro-texto é auxiliar o estudante a desenvolver o raciocínio lógico aplicado à solução de problemas em nível computacional. Para isso, é necessário prepará-lo para analisar problemas, projetar e validar diferentes soluções por meio de técnicas e ferramentas de desenvolvimento de algoritmos envolvendo raciocínio estruturado e modularização.

Ao término do curso, espera-se que o estudante seja capaz de:

- compreender os fatos essenciais, os conceitos, os princípios e as teorias relacionadas à área de computação e informática, para o desenvolvimento de software e suas aplicações;
- reconhecer a importância do raciocínio lógico no cotidiano e sua aplicação em circunstâncias apropriadas e em domínios diversos;
- identificar e analisar requisitos e especificações para problemas específicos e planejar estratégias para suas soluções;
- especificar, projetar e implementar sistemas de computação, empregando teorias, práticas e ferramentas adequadas.

## INTRODUÇÃO

Os algoritmos e a lógica são fundamentais para solucionar problemas por meio da programação de computadores. Tais soluções podem abarcar desde a resolução de um problema matemático até a captura de dados, seu armazenamento e sua transformação em resultados.

A solução da maioria dos problemas requer um ou mais dados de entrada e a obtenção deles se dá por meio de interações entre o programa de computador e o usuário. Essas interações podem conduzir à necessidade de tomadas de decisões para a transformação de dados em resultados, assim como à necessidade de repetição de procedimentos ou de armazenamento dos dados obtidos (ou de seus resultados).

Para compreender os recursos de linguagem computacional para escrever programas de computador e estruturar soluções de problemas, este livro-texto faz um caminho de estudos que se inicia com a introdução na qual são definidos os principais conceitos aplicados, subdivididos em quatro seções.

A **primeira seção** apresenta os tipos de algoritmos: o descrito de forma narrativa e em linguagem natural, o descrito em forma gráfica, por meio de fluxograma, e o descrito em uma linguagem denominada pseudocódigo, a qual respeita a sintaxe da linguagem de programação, mas com termos traduzidos para o português.

A **segunda seção** apresenta a lógica de programação com uma explicação de como os dados são processados e armazenados na memória do computador, bem como os tipos primitivos de dados, os operadores e as operações permitidas.

Na **terceira seção** são apresentados os comandos de entrada e saída, ou seja, como programar as interações do computador com o usuário e vice-versa, de maneira compreensível e organizada, em blocos de programas.

Na **quarta seção** são apresentados os comandos de decisão e o modo sistemático de estruturar o raciocínio lógico das interações entre usuário e computador a fim de capacitar o programa para tomar decisões.

Feita essa introdução, falaremos das três estruturas de repetição utilizadas na programação de computadores, bem como de sua forma encadeada e do raciocínio lógico para escolher a mais adequada para cada solução de problema que estiver sendo estruturada.

E depois de ter abordado esses assuntos importantes, faremos uso de tudo que foi discutido para falar de um modo avançado de programar, o qual, inclusive, exige maturidade do estudante com relação aos conceitos aprendidos até este ponto do livro-texto.

Programas de computadores são soluções para questões que podem variar desde um problema matemático até a estruturação de um fluxo de dados e suas transformações em um processo gerencial. Alguns problemas podem ser grandes e difíceis de serem resolvidos e por isso precisam ser particionados em problemas menores para que sejam resolvidos por partes, de forma que a solução final seja alcançada a partir do conjunto de soluções.

Estudaremos as duas formas de modularizar o programa: procedimentos e funções, bem como o raciocínio lógico para escolher a mais adequada para cada situação. Serão apresentadas também técnicas de programação em blocos e regras de indentação a fim de tornar a leitura do código compreensível.

Por fim, falaremos das duas estruturas para armazenar e organizar dados com o objetivo de facilitar o acesso e recuperar a informação. São elas o vetor e a matriz, ambas estruturas de dados compostas homogêneas. Aproveitaremos o gancho desse assunto para abordar também algoritmos para a manipulação de dados.



Em todo o livro-texto, os algoritmos são apresentados em pseudocódigo a fim de tornar a solução da linguagem de programação compreensível pela máquina. Para tornar o curso prático, foi utilizada a sintaxe do VisuAlg para o pseudocódigo e os objetos do Visual Paradigm para o fluxograma.



### Observação

**VisuAlg** é um programa gratuito de edição, interpretação e execução de algoritmos de uso e distribuição livres, frequentemente empregado em diversas instituições de ensino no Brasil para o ensino de lógica de programação.

**Visual Paradigm** é uma ferramenta que oferece suporte para modelagem de sistemas por meio de notações gráficas, usadas neste livro-texto para modelar a lógica dos algoritmos.

Todos os algoritmos dos exemplos são representados por pseudocódigo e fluxogramas. A linguagem natural narrativa é pouco explorada, pois serve para o entendimento da solução. O estudante perceberá, com a evolução da aprendizagem e a experiência em programação, que a leitura do pseudocódigo, quando este está bem-estruturado, pode ser tão compreensível quanto a de um texto em linguagem natural.

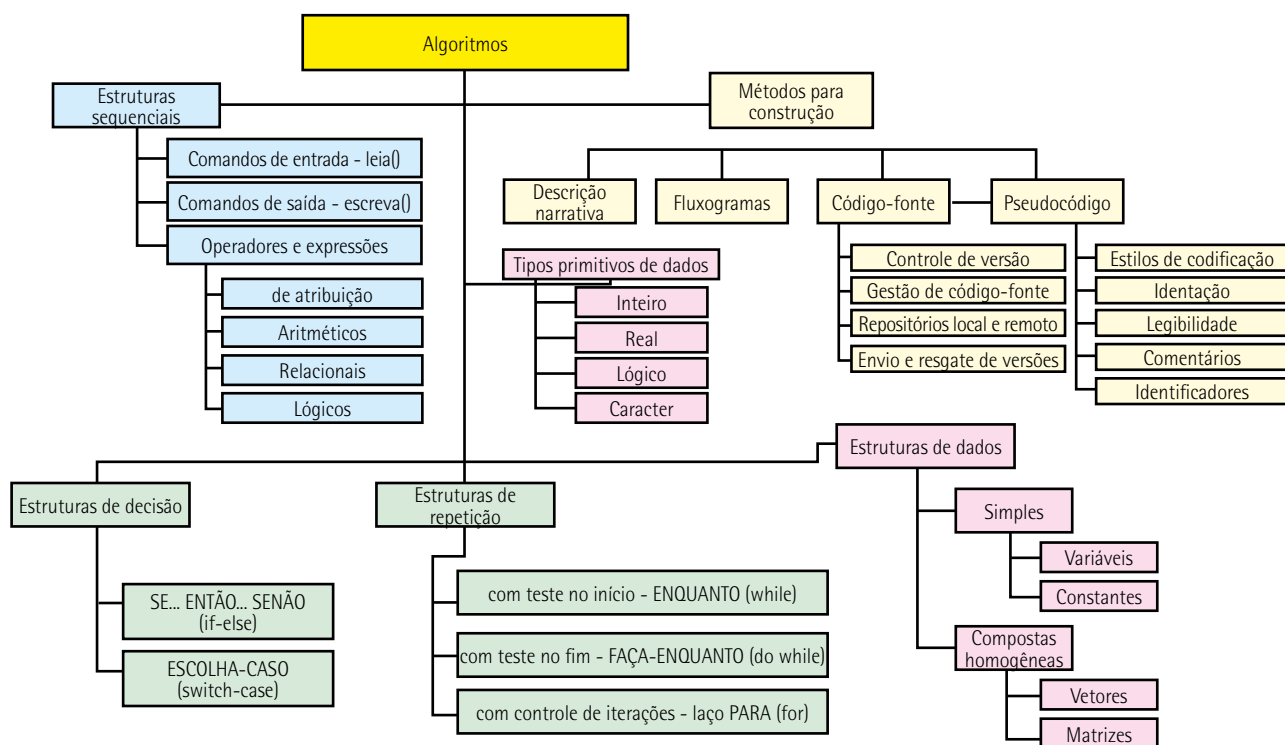


Figura 1 – Organização dos conteúdos



# Unidade I

## 1 ALGORITMOS

Computadores são capazes de processar milhares de instruções por segundo, armazenar um grande número de dados e recuperar informações com uma enorme rapidez. Programas são escritos de forma a usar todas essas capacidades dos computadores para resolver problemas envolvendo desde operações matemáticas (simples ou complexas) até a sistematização ou automação de processos. Em todos os casos, o entendimento do problema é o primeiro passo para resolvê-lo.

Na análise do problema devem ser identificados quais são os **dados de entrada** necessários para resolvê-lo, quais as transformações ou **processamento** que esses dados sofrerão e quais os resultados ou **dados de saída** esperados.

Os procedimentos para a captura dos dados de entrada, de processamento e de escrita dos dados de saída na ordem em que são executados formam a solução do problema, e é isso que denominamos **algoritmo**.

"Informalmente, algoritmo é qualquer procedimento computacional bem definido, que toma algum valor ou conjunto de valores como entrada e produz um valor ou conjunto de valores como saída." (CORMEN *et al.*, 2002, p. 3).

Algoritmos devem ser corretos e compreensíveis. De acordo com Cormen *et al.* (2002), o algoritmo é correto quando é aplicável a diferentes instâncias do problema e para ser compreensível precisa ser bem-escrito, com variáveis identificadas com nomes significativos e a lógica do processo documentada, quando for complexa e precisar ser explicada.

Muitos fatores afetam a escrita de um algoritmo compreensível, e, via de regra, assim como num texto em linguagem natural, repetições sucessivas da mesma ideia não são aceitas e repetições de procedimentos devem ser evitadas.

Adicionalmente, ao projetar um algoritmo, é necessário identificar quais são as variáveis do problema para as quais os dados de entrada serão atribuídos para cada instância do problema, a sequência correta para a resolução, além dos resultados esperados expressos nos dados de saída.

## 1.1 Tipos de representação de algoritmos

Há três formas de descrever os projetos de algoritmos:

- **Descrição narrativa:** o algoritmo é escrito em linguagem natural, de maneira sequencial, na ordem em que os procedimentos devem ser realizados para resolver o problema.
- **Fluxograma:** utiliza uma linguagem gráfica, na qual os objetos da linguagem possuem significados que expressam a intenção.
- **Pseudocódigo:** uma codificação escrita na estrutura e sintaxe da linguagem de programação, mas com os termos em português.

### 1.1.1 Descrição narrativa: linguagem natural

Suponha que você deseje comer um delicioso bolo e para fazê-lo precise de uma receita. O problema aqui proposto consiste em fazer o bolo, a solução é o algoritmo que descreve as entradas do processo de fabricação do bolo (ingredientes), e o processamento consiste no modo de fazer para obter como saída do processo o bolo.

Assim como existem diferentes receitas para um mesmo tipo de bolo, podem ser projetados diferentes algoritmos para resolver um único problema. A representação do algoritmo como uma descrição narrativa é útil para explicar a ideia de solução e a sequência dos procedimentos a serem realizados sem se preocupar com formalismos e detalhes da linguagem de programação.

#### Descrição narrativa: algoritmo para fazer um bolo de liquidificador

Separar os ingredientes.

Untar a forma.

Ligar o forno para pré-aquecer.

Bater todos os ingredientes no liquidificador por 5 minutos (exceto o fermento).

Adicionar o fermento e mexer delicadamente.

Despejar a massa na forma untada.

Assar em forno médio (180°) por 40 minutos.

Nesse exemplo foram omitidas as especificações dos ingredientes e o tipo de bolo que está sendo feito, mas é possível entender a sequência do processo de fabricação do bolo, a qual entende ser aplicada a qualquer tipo de bolo simples que possa ser feito no liquidificador. As entradas do processo são os ingredientes. A saída é o bolo. O processamento são os procedimentos a serem realizados para transformar os ingredientes (dados de entrada) em um bolo (dado de saída).

## 1.1.2 Fluxograma

O fluxograma é uma forma gráfica de descrever algoritmos capaz de representar o processo de solução do problema e comunicar a ideia do processo sem textos longos. No fluxograma não é necessário expressar a interação com o usuário, mas o fluxo processual de entrada, o processamento e a saída. Sua representação é baseada nos objetos apresentados na figura a seguir:

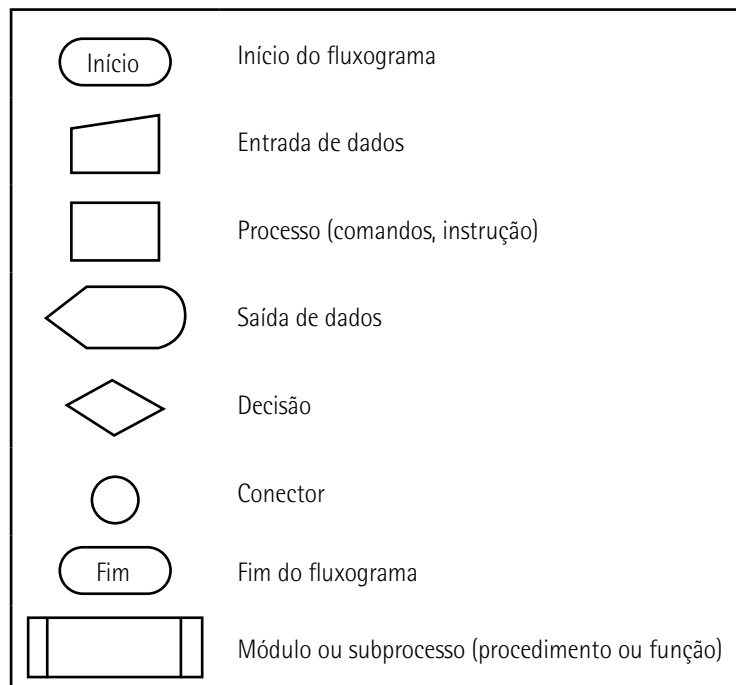


Figura 2 – Objetos para construção de fluxogramas

A figura do exemplo a seguir corresponde a algoritmos para calcular a soma de dois números, escritos em linguagem gráfica e descrição narrativa, respectivamente.

### Exemplo

Problema: escreva um algoritmo para receber dois números e calcular e mostrar a soma.

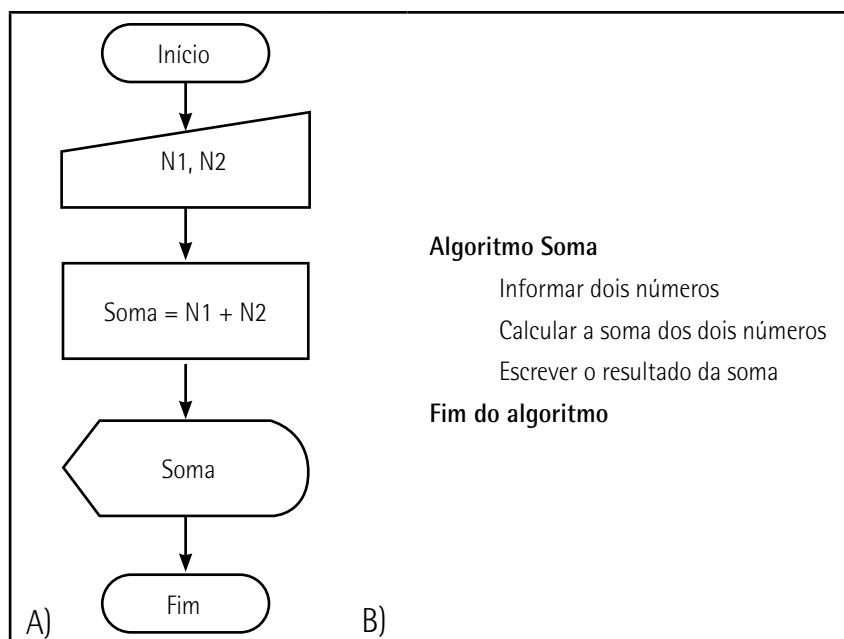
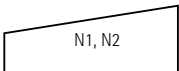
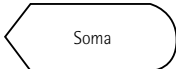
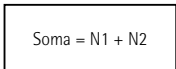


Figura 3 – Representações do algoritmo para calcular a soma de dois números:  
A) fluxograma; B) descrição narrativa

O algoritmo estará correto se para quaisquer dois números informados o cálculo da soma for sempre correto. Nesse exemplo, os dados de entrada correspondem aos números a serem somados, e o processamento é a operação matemática de adição, que transformará os dois números de entrada em um resultado – que é o dado de saída.

Na figura anterior, os objetos rotulados como Início e Fim representam, respectivamente, o início e o fim do algoritmo. Já no objeto , N1 e N2 são variáveis que guardam os dados de entrada, significam que os números devem ser informados para que possam ser calculados. É dispensável escrever no fluxograma qualquer palavra que indique o comando de entrada dos dados, visto que a leitura do objeto já é o suficiente.

Da mesma forma, o objeto  indica que o resultado da soma deve ser apresentado para o usuário. O objeto de saída de dados escreverá a soma num dispositivo de saída, ou seja, a tela do computador ou a impressora.

O objeto  indica o processamento a ser realizado, ou seja, o processo de transformação dos dados de entrada (N1 e N2) e atribuição de um valor a uma variável identificada por soma.

O algoritmo escrito em descrição narrativa na figura anterior expressa a ideia do processo, ou seja, a lógica do processo de solução. A cada execução do algoritmo, dois números serão informados pelo usuário, o algoritmo efetuará a soma e apresentará o resultado.

## 1.1.3 Pseudocódigo

O pseudocódigo é a forma de representação do algoritmo que mais se aproxima da linguagem de programação, mas também é uma linguagem que se aproxima da linguagem natural. Pode ser escrito em qualquer idioma, mas corresponde à tradução simples dos comandos originalmente definidos em inglês.

Nesse tipo de representação, a ideia de solução é estruturada usando o mesmo vocabulário aplicado na linguagem de programação, e utiliza uma estrutura sintática para organizar os comandos de acordo com a técnica de programação estruturada.

A técnica de programação estruturada é um modelo de programação no qual os comandos são executados em ordem sequencial, permitindo desvios por meio de estruturas de decisão ou laços de repetições, bem como pela modularização do código.

A estrutura sintática do pseudocódigo é apresentada na figura seguinte:

```
Algoritmo <nome_do_algoritmo>  
Var  
    //declaracao das variaveis  
Inicio  
    //sequencia de comandos  
Fimalgoritmo
```

Figura 4 – Estrutura do algoritmo escrito em pseudocódigo

As linguagens de programação possuem sintaxes próprias e rígidas, que impedem o funcionamento do programa quando não são especificadas corretamente. O propósito de usar o pseudocódigo é projetar algoritmos para resolver problemas com independência da linguagem de programação, com uma estrutura que facilita o entendimento da lógica de programação e a posterior tradução para a linguagem de programação a ser escolhida. A figura a seguir traz o pseudocódigo para o algoritmo apresentado na figura 3 para calcular a soma de dois números.

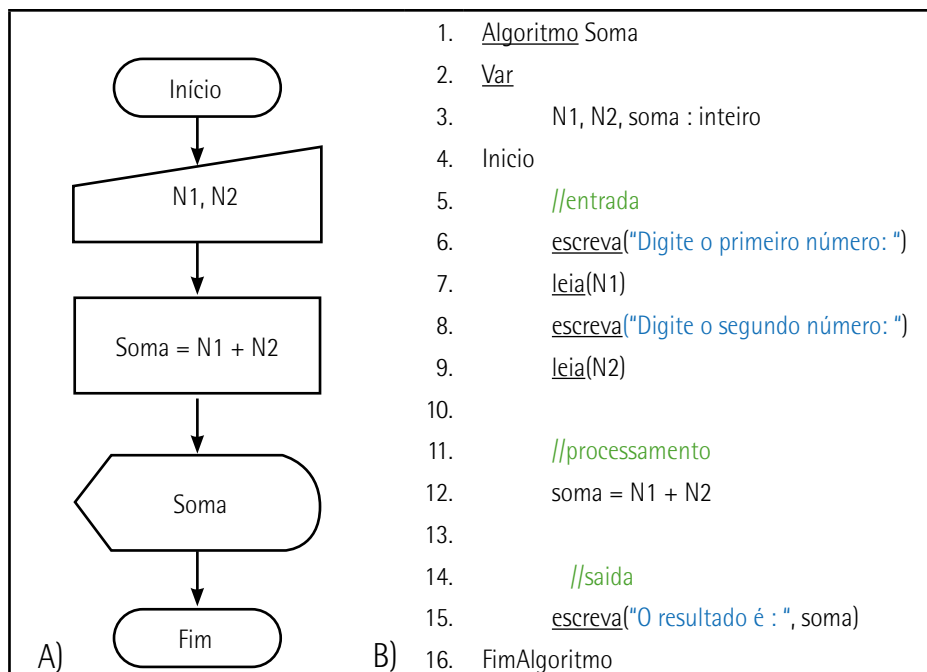


Figura 5 – Representações do algoritmo para calcular a soma de dois números:  
A) fluxograma; B) pseudocódigo

No pseudocódigo a interação com o usuário é expressa a fim de comunicar ao usuário a intenção do programa, além de informar o que é esperado que ele faça em prol de resolver o problema. As linhas 6, 8 e 15 são comandos que não aparecem no fluxograma da figura 3, repetido na figura 5.

No pseudocódigo, assim como na linguagem de programação, o projetista de software deve projetar a interface do programa e as interações entre o sistema e o usuário a fim de obter eficiência no uso. As interações são as mensagens que o sistema apresenta para o usuário para induzir o comportamento do usuário de digitar um número ou uma letra ou palavra, ou clicar em algum botão, ou outra ação que responderá às perguntas que o sistema fará.

Uma **dica** interessante é que a sintaxe de uma linguagem define a maneira como o programador deve escrever suas instruções. A sintaxe utiliza os símbolos < > e [], os quais indicam, respectivamente, obrigatoriedade e opcionalidade. Os termos encontrados entre <> devem constar na instrução codificada. Um exemplo de uso dos marcadores < > e [] para escrever um algoritmo na Linguagem VisuAlg é:

```

<Algoritmo> <"nome_do_algoritmo">
<var>
    <var1> [, var2, var3, varN] <:=> <tipo>
<Inicio>
    // instrucoes
<Fimalgoritmo>
  
```

Figura 6



Os textos escritos entre os símbolos < e > são obrigatórios e os escritos entre os símbolos [ e ] são opcionais.



## Saiba mais

Duas fontes muito interessantes de informação sobre esse assunto são o site do VisuAlg e do Visual Paradigm:

Disponível em: <https://visualg3.com.br/>. Acesso em: 30 set. 2021.

Disponível em: <https://www.visual-paradigm.com/>. Acesso em: 30 set. 2021.

O fluxograma e o pseudocódigo da figura 5 **são soluções para o mesmo problema, mas o pseudocódigo possui detalhes de interações desnecessárias no fluxograma**, motivo pelo qual elas não foram representadas. Enquanto o pseudocódigo, pela aproximação com a linguagem de programação, codifica a lógica e estabelece a comunicação com o usuário para juntos (sistema e usuário) resolverem um problema, o fluxograma tem a intenção de representar de maneira simplificada a lógica que transformará os dados de entrada em dados de saída, ou seja, o processamento.

Adicionalmente, por se tratar de uma codificação estruturada, os comandos devem expressar a sequência de interações entre o usuário e o computador sempre que necessário.

O programa interage com o usuário a partir das mensagens apresentadas na tela do computador (comandos de saída) e o usuário interage com o programa respondendo as mensagens utilizando os dispositivos de entrada tais como teclado e mouse.

Essas interações são realizadas na ordem em que são programadas. Dessa forma, o pseudocódigo deve expressar as interações do usuário com o programa e a futura codificação em uma linguagem de programação específica será apenas uma tradução do algoritmo.



## Observação

Os algoritmos desenvolvidos neste livro-texto estão representados em pseudocódigo e fluxograma.

Uma dica interessante é que **comentários** são anotações no pseudocódigo para documentar algo importante para a compreensão do código e podem ser escritos em linguagem natural. Os comentários podem ser anotados em uma única linha e, nesse caso, inicia-se com dupla barra //, mas se duas ou mais linhas de comentários forem necessárias, recomenda-se o uso da anotação de bloco de comentários delimitado pelos símbolos /\* e \*/ , como apresentado no exemplo a seguir:

```
// comentario de uma unica linha.  
/* inicio do bloco comentado.  
...  
...  
*/
```

Figura 7

Os comentários ficam destacados em verde ou cinza na maioria das linguagens de programação e são ignorados pelo compilador.

No pseudocódigo da figura 5 a estrutura sequencial entre os comandos de entrada, processamento e saída está comentada nas linhas 5, 11 e 14. Tais linhas são comentários importantes para a compreensão do código, mas são ignoradas pelo compilador. Os comentários são úteis para explicar em descrição narrativa uma lógica complexa para cujo entendimento o pseudocódigo não é suficiente.

**Algoritmo** <nome\_do\_algoritmo>

**Var**

//comentario

Início

comando1

comando2

/\* inicio do bloco de comentario

\* comentario

\* comentario

\* comentario

\*/ fim do bloco de comentario

comando3

:

comandoN

Fimalgoritmo

Figura 8 – Exemplo de comentários no corpo do pseudocódigo

### 1.2 A lógica de programação: entendendo como dados são processados e onde são armazenados

Para entender a lógica de programação, é importante compreender como os dados são processados e onde são armazenados. Um programa de computador é uma aplicação prática que manipula dados e os transforma em informação.

Dados são valores brutos que, por si só, não possuem significado. Exemplos de dados são os textos Eliane, São Paulo, UNIP, Algoritmos ou números tais como 18, 16, 466, R\$ 1.030,75, ou 4,393. Pelo senso comum, é possível deduzir que "Eliane" é o nome de uma pessoa e "UNIP" de uma universidade, mas não identifica Eliane como professora, aluna ou qual o papel que Eliane assume em algum contexto. Da mesma forma, embora se possa deduzir que R\$ 1.030,75 é um valor monetário, não é possível afirmar que seja o salário de alguém ou o preço de um objeto ou serviço.

Informação são dados agregados, em outras palavras: dados combinados com outros dados constituem uma informação. Por exemplo, professora Eliane – disciplina Algoritmos; professor João – disciplina Matemática. A informação aqui representada é a de que Eliane é a professora da disciplina Algoritmos e João é o professor de Matemática. Novos dados poderiam ser agregados e tornar a informação completa, por exemplo, Instituição de Ensino Superior: Unip – disciplina: Algoritmos – professora Eliane.

Todos os dados são armazenados na memória do computador. O usuário interage com o programa usando dispositivos de entrada (teclado, mouse, microfone, webcam) e o programa interage com o usuário por meio dos dispositivos de saída (monitor, impressora, caixas acústicas). Os dispositivos de entrada, saída e armazenamento são gerenciados pela unidade central de processamento (CPU, do inglês central processing unit).

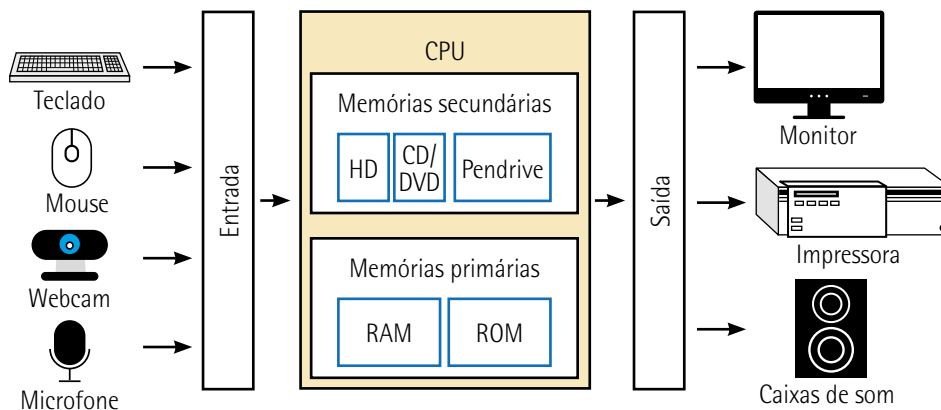


Figura 9 – Organização básica e simplificada dos computadores

A sequência lógica de processamento dos dados é a ilustrada na figura a seguir:



Figura 10 – Sequência lógica de processamento de dados

Nessa sequência, os comandos de entrada acionam os dispositivos de entrada para capturar os dados de entrada, e os comandos de saída acionam os dispositivos de saída para exibir os resultados (dados de saída).

No meio desse caminho, os dados são armazenados na memória RAM para serem processados pela CPU.

A memória do computador é finita e quantificada em bytes. Um byte corresponde a um caractere. Exemplos de caracteres são:

- as letras do alfabeto maiúsculas e minúsculas: A, a, B, b, Z;
- os números 0, 1, 2, 3, ... 9;
- os símbolos, como:  $\neq$ ,  $^{\circ}$ ,  $\pi$ ,  $\Sigma$ ;
- o espaço em branco.

A menor unidade de dados é o dígito binário (BIT, do inglês binary digit) representado pelo 0 e pelo 1, e cada byte é formado por oito ( $2^3$ ) bits.

A potência de 2 é muito presente na computação, e base para as unidades de medida dos dados. A cada  $2^{10}$  unidades obtém-se uma unidade da próxima escala. Por exemplo, a cada 1024 bytes ou  $2^{10}$  bytes, obtém-se 1 Kibibyte (KiB). Popularmente é usado o prefixo da expressão em decimal (KB). A cada 1.024 KiB obtém-se 1 mebibyte e assim por diante.

O armazenamento de dados na memória secundária (disco rígido, CD, DVD ou pendrive) é quantificado em bytes. Quando o prefixo em decimal é utilizado, a escala de conversão é 1:1000, ou seja, uma unidade para cada mil da escala anterior, enquanto o prefixo em binário considera a proporção de 1:1024. O quadro a seguir apresenta todas as unidades de medidas para dados.

**Quadro 1 – Unidades de medidas dos dados**

Unidade	Prefixo em decimal	Nome e prefixo em binário	Valor equivalente em binário
Bit	0 e 1		
Byte	B		8 bits
Kilobyte	KB	Kibibyte (Kib)	1024 B
Megabyte	MB	Mebibyte (MiB)	1024 KiB
Gigabyte	GB	Gibibyte (GiB)	1024 MiB
Terabyte	TB	Tebibyte (TiB)	1024 GiB
Petabyte	PB	Pebibyte (PiB)	1024 TiB
Exabyte	EB	Exbibyte (EiB)	1024 PiB
Zettabyte	ZB	Zebibyte (ZiB)	1024 EiB
Yottabyte	YB	Yobibyte (YiB)	1024 ZiB

Adaptado de: IEC Technical Committee (2008).

## 1.3 Tipos de dados

Dados são representados por letras, números ou valores lógicos que especificam os conjuntos de valores que poderão assumir. O computador reconhece quatro tipos primitivos de dados:

- tipo numérico inteiro;
- tipo numérico real;
- tipo caractere, que aceita valores alfanuméricos;
- tipo lógico, que representa os valores lógicos **verdadeiro** e **falso**.

O quadro a seguir apresenta o conjunto de valores que cada tipo de variável pode aceitar.

**Quadro 2 – Tipos primitivos de dados e respectivos valores que aceitam**

Tipo de dado	Conjunto de valores que aceita
Inteiro	Conjunto dos números inteiros $Z = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$
Real	Conjunto dos números reais, ou seja, todos os inteiros e todos os números do intervalo entre dois inteiros $R = \{-2, ..., -2,999, ..., -2,99, ..., -2,9, ..., -2,8, ..., -2, ... -1, ..., 0, ..., 0,001, ..., 1, ... 1,5, ..., 2, ...\}$
Caractere	Um único símbolo disponível no teclado, seja letra, número, símbolo ou espaço em branco. Um dado definido como caractere pode receber um número, mas operações matemáticas não poderão ser realizadas sobre ele
Lógico	Verdadeiro; falso



### Observação

Os valores lógicos verdadeiro e falso podem ser representados pelos numerais 1 e 0, respectivamente.

## 1.4 Variáveis e constantes

Os dados são armazenados na memória primária e de acesso aleatório (RAM, do inglês random access memory) em espaços alocados em tamanhos diferentes, de acordo com o tipo do dado que será armazenado. Dados podem ser armazenados como **variáveis** ou **constantes**.

**Variáveis** são espaços alocados na memória RAM durante a execução do programa. A memória alocada é liberada quando o programa é encerrado. Durante a execução do programa, uma variável poderá assumir diferentes dados do tipo em que foi declarada. A identificação das variáveis deve respeitar as regras dos identificadores, das quais falaremos melhor mais adiante neste livro-texto.

```
var
    <nome_variável> : <tipo_de_dado>
    <var1>, <va2>, ..., <var_n> : <tipo_de_dado>
```

Figura 11 – Sintaxes para declaração de uma ou mais variáveis do mesmo tipo

### Exemplo

A figurar a seguir apresenta um exemplo de declarações de variáveis. Ao declarar duas ou mais variáveis do mesmo tipo, separe-as por vírgula. No exemplo da figura seguinte, as variáveis **idade** e **qtde** são ambas do tipo inteiro, a variável **salário** é do tipo real, a variável **sexo** do tipo caractere e a variável **ligado** do tipo lógico e armazena os valores lógicos **verdadeiro** ou **falso**.

```
Var
    idade, qtde : inteiro
    salário      : real
    sexo         : caractere
    ligado       : lógico
```

Figura 12 – Exemplo de declarações de variáveis de tipos primitivos

As variáveis **idade** e **quantidade (qtde)** ambas armazenam um valor inteiro. A variável **salário** armazena um valor real. A variável **sexo** armazena apenas um caractere. A variável **"ligado"** armazena um dos valores lógicos **verdadeiro** ou **falso**.

O exemplo da figura a seguir é um algoritmo que recebe dois números inteiros como dados de entrada e os armazena nas variáveis **x** e **y**. Após guardar os dados em variáveis, realiza a troca dos valores das variáveis, ou seja, o conteúdo da variável **x** passa a ter o valor da variável **y** e **y** passa a ter o valor de **x**. Ao término do algoritmo, os valores trocados são exibidos.

```
1. Algoritmo TrocaValoresDasVariaveis
2. Var
3.   x, y, aux : inteiro
4. Inicio
5.   // entrada
6.   escreva("Digite um valor para x: ")
7.   leia(x)
8.   escreva("Digite um valor para y: ")
9.   leia(y)
10.  // processamento
11.  aux ← x
12.  x ← y
13.  y ← aux
14.  // saida
15.  escreva("x = ", x, " e y = ", y)
16.  FimAlgoritmo
```

Figura 13 – Pseudocódigo para trocar os conteúdos das variáveis x e y

Ao declarar as variáveis **x**, **y** e **aux** na linha 3 do pseudocódigo da figura anterior, três endereços de memória são alocados e identificados por x, y e aux, respectivamente. Os comandos serão executados na sequência em que são instruídos. Quando o algoritmo se inicia na linha 4, o primeiro comando executado é o comando da linha 6.

```
escreva("Digite um valor para x: ")
```

Figura 14

O comando a seguir é um comando de entrada **leia(x)** e o cursor ficará piscando na tela aguardando a entrada de um valor para ser armazenado na variável **x**.

```
Digite um valor para x: ____
```

Tela

Figura 15 – Exemplificação da tela após a execução do comando de entrada leia(x)

Suponha que sejam digitados os valores 10 e 35 para x e y, respectivamente. Ao término da execução das linhas 7 e 9, os valores 10 e 35 estarão armazenados nas variáveis x e y e a variável aux estará vazia, conforme ilustrado na figura seguinte.

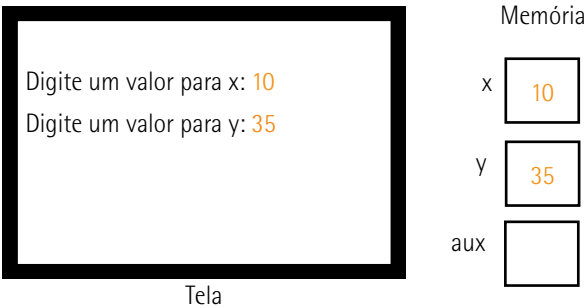


Figura 16 – Visualização dos comandos de saída do algoritmo da figura 13

O processamento desse algoritmo consiste em mudar os conteúdos das caixinhas, ou seja, das variáveis. Após o processamento, a variável **x** deverá guardar 35 e a variável **y**, o valor 10. Para isso, será necessário usar a variável **aux** como auxiliar. As linhas 10 a 13 fazem o processamento da troca das variáveis, conforme ilustrado na figura a seguir:

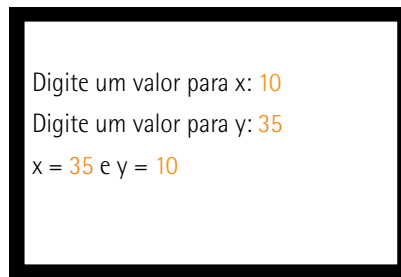
Código	Estado das variáveis x, y e aux na linha 10	Estado das variáveis x, y e aux quando a linha 11 é executada
10. //processamento 11. aux ← x 12. x ← y 13. y ← aux	<div><div>x10</div><div>y35</div><div>aux</div></div>	<div><div>x10</div><div>y35</div><div>aux10</div></div> <div>aux ← x</div>
Estado das variáveis x, y e aux quando a linha 12 é executada		Estado das variáveis x, y e aux quando a linha 13 é executada
<div><div>x35</div><div>y35</div><div>aux10</div></div> <div>x ← y</div>		<div><div>x10</div><div>y35</div><div>aux10</div></div> <div>aux ← x</div>

Figura 17 – Estado das variáveis durante o processamento do algoritmo

Ao término do processamento, o comando de saída é executado e as variáveis são trocadas. O comando de saída escrito na linha 15 mostrará as variáveis **x** e **y**, conforme ilustrado na figura subsequente:



```
escreva("x = ", x, " e y = ", y)
```



Tela

Figura 18 – Ilustração da interface do usuário após a execução dos comandos de saída do algoritmo da figura 13

**Constantes** são variáveis cujo valor é fixo, ou seja, não pode ser alterado durante a execução do programa. Constantes podem ser definidas com os tipos caractere, lógica ou numérica e são úteis para identificar números por seus conceitos ou para atribuir um valor único no início do programa e definir um único ponto de acesso para sua alteração. A identificação das constantes deve respeitar a regra dos identificadores.

```
var  
    <nome_constante> ← <valor> <tipo_de_dado> <const>
```

Figura 19 – Sintaxe para declaração de constante



## Observação

Identificadores de variáveis devem ser significativos ao propósito do valor que irão armazenar, se iniciar com letras (de a até z, minúsculas ou maiúsculas) ou o caractere sublinhado ( \_ ). Como boa prática de programação, utilizam-se letras minúsculas para variáveis e, quando os nomes são compostos, devem ser separados pelo caractere sublinhado, por exemplo: **salario\_liquido**, **nome\_completo**.

Identificadores não podem se iniciar com números ou caracteres especiais, tais como @, #, \$, % ou outros. Exemplos de identificadores inválidos são: **#nomevar**, **123**, **1var**.

Identificadores de constantes seguem as mesmas regras das variáveis, contudo, como boa prática, utilizam-se letras maiúsculas para as constantes, como pode ser observado no exemplo da figura seguinte.

### Exemplo

A figura a seguir é um algoritmo para calcular a área do círculo, no qual há três variáveis do tipo real, identificadas por **diâmetro**, **área** e **raio**, e uma constante identificada por **NUMERO\_PI**.

```
1.  Algoritmo "Área do círculo»
2.  Var
3.    NUMERO_PI: real const
4.    diametro, area, raio : real
5.  Inicio
6.    PI <- 3,14
7.    escreva("Informe o diâmetro do círculo: ")
8.    leia(diametro)
9.    raio ← diametro/2
10.   area <- NUMERO_PI*(raio*raio)
11.   escreva("Área da circunferência é ", area)
12.  Fimalgoritmo
```

Figura 20 – Exemplo de aplicação da constante  $\pi$ .r2

As constantes são valores inteiros ou reais alocados numa posição de memória seguindo as regras de alocação de uma variável, ou seja, identificação por um nome, mas valor armazenado estático (que não muda durante todo o processo de execução do programa).

Utilizam-se constantes para substituir um "número mágico" por um nome significativo, tal como exemplificado na figura anterior. Isso é feito porque, para dar um exemplo famoso, é mais fácil usar o número do  $\pi$  fazendo referência ao conceito Pi do que lembrar-se do número completo (3,14159265358979323846).

### 1.5 Identificadores

Os identificadores são os nomes dados às variáveis, constantes, funções e procedimentos dentro de um programa, e devem respeitar as seguintes regras:

- o primeiro caractere deve ser uma letra;
- os demais podem ser letras, números ou sublinhado;
- não são permitidos símbolos especiais (&, ?, !, +, -, /, \*, :, ., dentre outros);
- não se podem usar palavras reservadas do pseudocódigo ou da linguagem de programação.

O quadro a seguir relaciona algumas palavras reservadas do pseudocódigo. Nos algoritmos deste livro-texto, as palavras reservadas aparecem sublinhadas nos pseudocódigos.

**Quadro 3 – Principais palavras reservadas do pseudocódigo**

algoritmo	inteiro	se	escolha	enquanto	repita	para
var	real	então	caso	faça	Até	passo
inicio	lógico	senão	outrocaso	fimenquanto	fimrepita	fimpara
fimalgoritmo	caractere	fimse	fimescolha			

escreva	função	procedimento	mod	vetor
leia	retorne	fimprocedimento	raizq	matriz
	fimfunção			

Como boa prática de programação, com a finalidade de tornar o código legível e compreensível, está convencionado neste livro-texto que os identificadores respeitarão as regras anteriormente descritas e as seguintes:

- Identificadores usarão nomes significativos para todas as variáveis e constantes.
- Variáveis serão sempre identificadas com letras minúsculas e, quando necessário, identificarão variáveis com nomes compostos, separando os termos com sublinhado. Exemplo:

```
var
    idade, qtde : inteiro
    salario_bruto : real
```

Figura 21

- Constantes serão sempre identificadas com letras maiúsculas e, quando necessário, serão usados nomes compostos, separando os termos com sublinhado. Exemplo:

```
var
    PI ← 3,14 : real const
    MAX ← 100 : inteiro const
```

Figura 22

- Funções e procedimentos serão sempre identificados com letras minúsculas e, quando necessário, serão usados nomes compostos, sendo que a inicial da segunda palavra fica em letra maiúscula.

função nomeFuncao() : tipo

Figura 23

## 2 SOLUÇÕES DE PROBLEMAS

### 2.1 Comandos de entrada e saída

A programação é um processo de solução para um dado problema elaborado com base num conjunto de comandos que, na sequência em que serão executados, transformam dados de entrada em dados de saída. Os dados de saída correspondem ao resultado do problema ou a um resultado parcial equivalente a uma etapa do processo de solução.

Os dados de entrada são digitados pelo usuário usando os dispositivos de entrada, tais como o teclado, o mouse ou outro recurso. Já os dados de saída são apresentados ao usuário por meio de um dispositivo de saída.

Na programação, há comandos específicos para acionar os dispositivos de entrada e saída. A entrada de dados é acionada por meio do comando **leia()** e a saída de dados, pelo comando **escreva()**.

O comando **leia** captura o dado do teclado para que seja processado e o resultado seja exibido na tela. A leitura dos comandos de entrada e saída é feita de modo imperativo, como se estivesse dando uma ordem. Por exemplo:

**leia(x)** ↔ Leia a variável x  
**escreva(y)** ↔ Escreva a variável y

Figura 24 – Interpretação dos comandos de entrada e saída

As formas sintáticas do pseudocódigo para os comandos de entrada e saída são apresentadas no quadro a seguir:

**Quadro 4 – Comandos de entrada e saída**

Comandos	Sintaxes	Exemplos
Entrada	<b>leia</b> (<variavel>)	<b>leia</b> (x)
	<b>leia</b> (<variavel>,<variavel>)	<b>leia</b> (x,y)
Saída	<b>escreva</b> (<variavel>)	<b>escreva</b> (y)
	<b>escreval</b> (<variavel>)	<b>escreval</b> (y)
	<b>escreva</b> (<"texto">)	<b>escreva</b> ("mensagem")
	<b>escreva</b> (<"texto">,<variável>,<"texto">)	<b>escreva</b> ("O valor de x é ", x)

A figura 20 traz um pseudocódigo com aplicação dos comandos de entrada e saída e o fluxograma é apresentado apenas para expressar a lógica do algoritmo.

O exemplo a seguir corresponde ao programa para calcular a área de um retângulo. Escrever um programa de computador implica entender o problema com clareza e para fazê-lo é importante identificar as variáveis do problema. Para calcular a área do retângulo, é importante saber quais são as medidas de base e altura, conforme ilustrado a seguir:

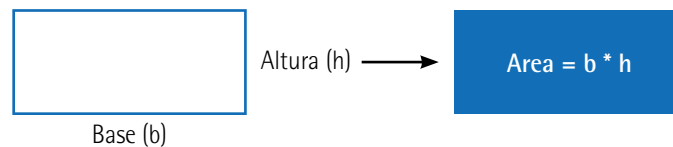


Figura 25 – Representação do entendimento do algoritmo para calcular a área do retângulo

A base e a altura informados são os dados de entrada para a resolução do problema. A saída é a área calculada do retângulo. O processamento é a operação de multiplicação realizada a partir dos dados de entrada. A resolução desse problema em um programa de computador é dada na figura seguinte:

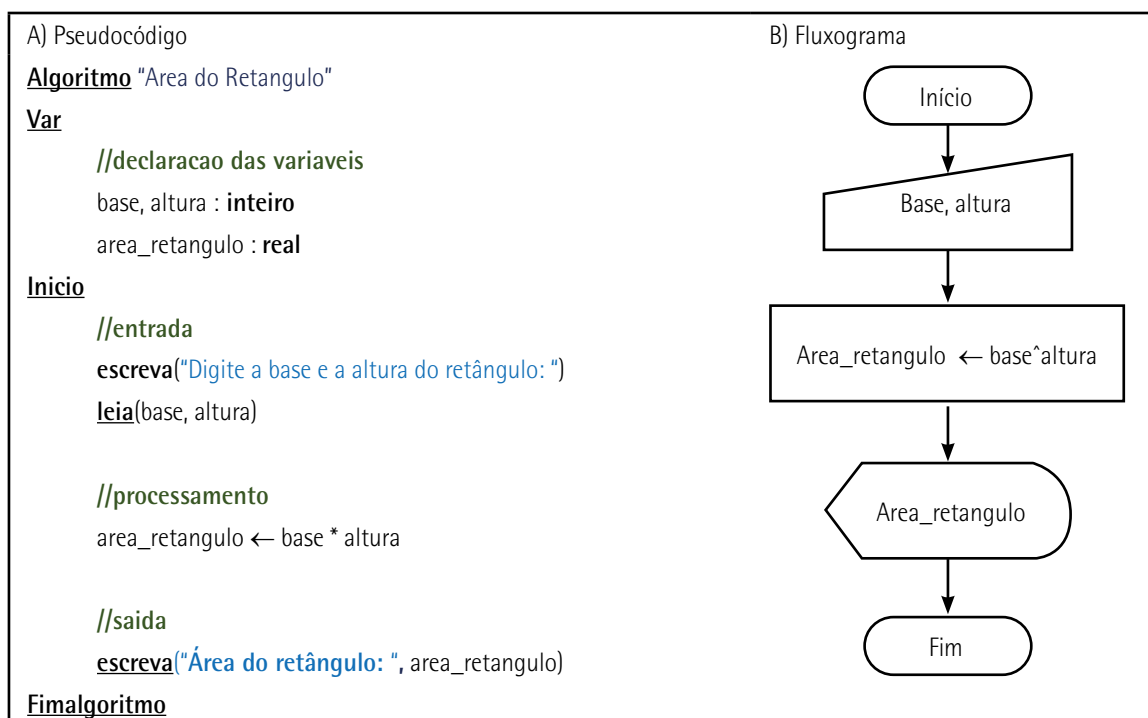


Figura 26 – Algoritmo para calcular a área do retângulo

Nesse exemplo, as variáveis base e altura foram definidas como **inteiras**, mas poderiam ter sido declaradas como do tipo **real**. Da forma como estão, apenas números inteiros poderão ser informados como medidas para as variáveis base e altura.

O comando **leia(base, altura)** poderá receber uma ou mais variáveis. Quando duas variáveis são lidas numa única instrução, não haverá interação do programa com o usuário, que deverá digitar um número, pressionar a tecla **enter**, digitar outro número e pressionar a tecla enter novamente para que o algoritmo continue a execução da sequência de comandos.

O comando **escreva()** está exibindo o texto "Área do retângulo: " e o conteúdo da variável **área\_retangulo**.

A comunicação entre o usuário e o programa deve ser dialógica, ou seja, o programa deve informar ao usuário o que espera que ele faça. A interação usuário-computador estabelece a facilidade com que o usuário irá se comunicar com o sistema.

A figura a seguir é um exemplo de algoritmo escrito com interações em forma de diálogo entre o usuário e o programa para calcular a média aritmética das notas de um aluno.

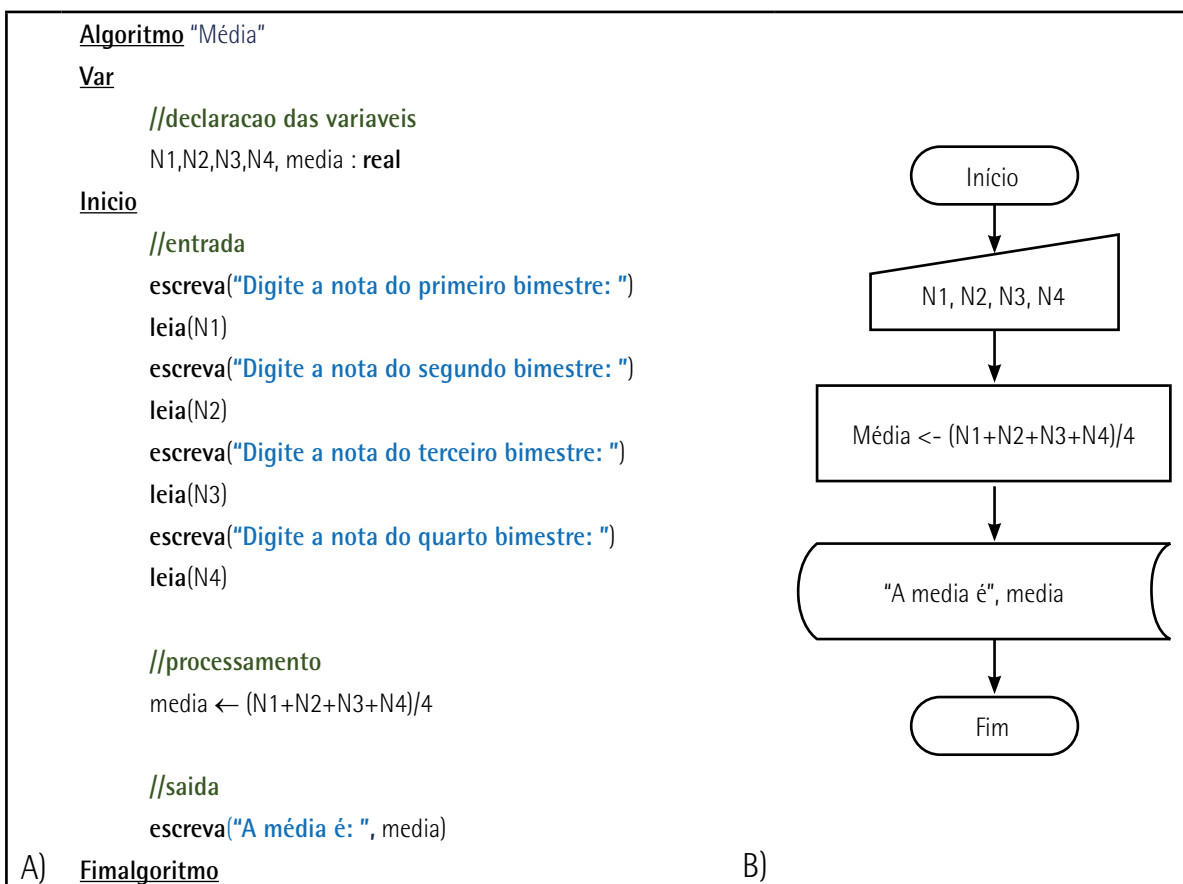


Figura 27 – Algoritmo para calcular a média aritmética de um aluno:  
A) pseudocódigo com interação entre o usuário e o programa; B) fluxograma

Duas dicas úteis são:

- no fluxograma as interações não precisam ser representadas. O pseudocódigo faz a leitura das variáveis N1, N2, N3 e N4 em linhas de comandos diferentes.
- uma variação do comando **escreva()** é o **escreval()**. O primeiro escreve na posição do cursor. O segundo escreve a mensagem e pula uma linha. Para que a interface do usuário fique mais amigável, a utilização de tais recursos pode ser necessária.

## 2.2 Fazendo operações com dados

Resolver problemas implica fazer cálculos, comparações ou tomar decisões ao longo do processo de resolução. A partir de agora estudaremos o operador de atribuição, os operadores aritméticos, lógicos e relacionais necessários para o desenvolvimento da lógica de programação num processo de solução de problemas.

### 2.2.1 O operador de atribuição ( $\leftarrow$ )

Dados são atribuídos às variáveis considerando o tipo do dado e da variável que o receberá. Atribuir um dado a uma variável é o comando responsável por armazenar no endereço de memória alocado pela variável o valor a ela atribuído. A uma variável pode ser atribuída uma constante, uma expressão ou outra variável, conforme ilustrado no algoritmo da figura a seguir.

```
1.  Algoritmo "Cálculo salário líquido"
2.  Var
3.      salario_bruto      : real
4.      salario_liquido    : real
5.      x, y, z : inteiro
6.  Inicio
7.      salario_bruto  $\leftarrow$  1798,50
8.      salario_liquido  $\leftarrow$  salario_bruto * 0,8
9.      x  $\leftarrow$  10
10.     y  $\leftarrow$  x + 10
11.     z  $\leftarrow$  x + y * 2
12.  Fimalgoritmo
```

Figura 28 – Algoritmo para ilustrar diferentes atribuições para variáveis

A variável **salario\_bruto**, na linha 7, está recebendo uma constante real. A variável **salario\_liquido** está recebendo uma expressão aritmética, ou seja, o produto de uma variável por uma constante. A constante inteira 10 está sendo atribuída à variável **x** na linha 9. Na linha 10, a exemplo da atribuição realizada na linha 8, a variável **y** está recebendo uma expressão aritmética e será atribuído à variável **y** o resultado da soma do valor de **x** mais a constante 10, ou seja, **y** recebe 20. Na linha 11, a variável **z** recebe  $x + y * 2$ , ou seja, **z** recebe 60, que é o resultado de  $10 + 20 * 2$ .

Diferentes expressões podem ser elaboradas para atribuir valores às variáveis, realizar cálculos ou tomar decisões no processo de resolução de problemas. A seguir são apresentados os operadores aritméticos, lógicos e relacionais empregados no processo de desenvolvimento de lógica de programação.

## 2.2.2 Operadores aritméticos

O computador é um ótimo processador de cálculos matemáticos, capaz de executar as operações aritméticas básicas e avançadas numa velocidade muito maior do que qualquer ser humano, por mais inteligente que seja. As operações aritméticas básicas são as de adição, subtração, divisão e multiplicação, cujos operadores são os mesmos presentes nas calculadoras.

As expressões aritméticas podem incluir uma ou mais operações matemáticas básicas e retornam o resultado como um dado do tipo inteiro ou real. Portanto, uma expressão aritmética deve ser atribuída a uma variável do tipo de retorno da expressão, ou simplesmente escrita num comando de saída, tal como exemplificado na tabela a seguir.

**Tabela 1 – Operadores aritméticos e exemplos de aplicação**

Operador	Significado	Expressões aritméticas
+	Adição	$x \leftarrow 5 + 15$ $y \leftarrow x + 5$ $z \leftarrow 10 + y$ $r \leftarrow x + y + z$
-	Subtração	$x \leftarrow 115 - 20$ $y \leftarrow x - 15$ $z \leftarrow (y - 10) - 2$ $r \leftarrow x - y$ $\text{escreva}(x + y * z - r)$
/	Divisão	$x \leftarrow 150 / 3$ $y \leftarrow x / 10$ $z \leftarrow (x + y) / 2$ $r \leftarrow y / x$ $\text{escreva}(10 * (x + y) / 2 * (z + r))$
*	Multiplicação	$x \leftarrow 2 * 3$ $y \leftarrow x * x * x$ $z \leftarrow x * y$ $\text{escreva}(10 * (x + y) / 2 * (z + r))$

Em pseudocódigo, existem alguns operadores aritméticos auxiliares que facilitam a programação sem a necessidade de desenvolver o cálculo com todos os passos da matemática. A tabela seguinte apresenta os operadores aritméticos auxiliares que poderão ser usados no pseudocódigo.



**Tabela 2 – Operadores aritméticos auxiliares e exemplos de aplicação**

Operador	Significado	Expressões aritméticas com operadores auxiliares	Resultado
pot ou exp	Potenciação exponenciação	exp(base,expoente) $\leftrightarrow$ $b^e$ exp(2,5) exp(5,2) escreva(exp(3,2)) $x \leftarrow \text{exp}(5,8)$	$\text{exp}(b,e) = b^e$ $2^5 = 32$ $5^2 = 25$
rad ou raiz ou raizq	Radiciação ou raiz quadrada	rad(9) $\leftrightarrow$ raizq(9) rad(49) $\leftrightarrow$ raizq(49) escreva(raizq(81)) $x \leftarrow \text{raizq}(49)$	$\sqrt{9} = 3$ $\sqrt{49} = 7$
div	Quociente da divisão	15 div 2 9 div 4 escreva(25,"div", (div(25,3))) $x \leftarrow \text{raiz}(49)$	15 div 2 = 7 9 div 4 = 2
mod	Resto da divisão	5 mod 2 5 mod 3	5 mod 2 = 1 5 mod 3 = 2

Expressões aritméticas podem ser construídas apenas com valores numéricos inteiros ou reais, mas também podem ser combinadas com valores, variáveis e constantes. Expressões aritméticas resultam num valor que poderá ser armazenado em uma variável, conforme mostram os exemplos das duas tabelas anteriores.



## Observação

No VisuAlg, o operador de exponenciação funciona apenas para variáveis reais. Contudo, na linguagem de programação, a expressão poderá ser realizada tendo como base números inteiros.

**Tabela 3 – Operadores aritméticos unários e exemplos de aplicação**

Operador	Significado	Expressões aritméticas com operadores unários	Resultado
-	Negativo	$a \leftarrow -25$	$a = -25$
++	Pós-incremento	$a \leftarrow 4$ <b>a++</b>	$a = 5$
++	Pré-incremento	$a \leftarrow 4$ <b>++a</b>	$a = 5$
--	Pós-decremento	$c \leftarrow 3$ <b>c--</b>	$c = 2$
--	Pré-decremento	$c \leftarrow 3$ <b>--c</b>	$c = 2$

Os operadores aritméticos unários realizam duas ações em uma única instrução, mas o VisuAlg não dá suporte para todos, embora sejam muito utilizados nas linguagens de programação.

O operador unário negativo é usado para inverter o sinal do número e deve ser aplicado apenas com valores numéricos. Como no exemplo da tabela anterior, é possível atribuir um número negativo a uma variável numérica inteira ou real. Adicionalmente, é permitido utilizar as operações matemáticas sobre números, constantes ou variáveis por -1 ou outro valor negativo.

Os operadores de incremento e decremento, aplicados antes ou após a variável, respectivamente somarão ou subtrairão um ao valor que ela já guarda. O VisuAlg não reconhece os operadores de incremento e decremento, mas eles são muito importantes nas linguagens de programação.

Uma aplicação muito útil em operadores de incremento e decremento é a combinação deles numa instrução de atribuição. Como o VisuAlg não os reconhece, o exemplo a seguir será codificado na linguagem programação C.

### Exemplo

Este programa declara duas variáveis inteiras, aplica os operadores de incremento e decremento e escreve o resultado:

```
1.  int main()
2.  {      // Início do programa
3.      int x, y;    //declaração das variáveis
4.
5.      x = 10;
6.      y = 10;
7.
8.      x++;          //equivalente a x <- x+1
9.      ++y;          //equivalente a x <- y+1
10.
11.     //equivalente a escreva("x = ", x, "e y = ", y)
12.     printf("\n x = %d e y = %d", x, y);
13.
14.     x--;
15.     --y;
16.     printf("\n x = %d e y = %d", x, y);
17. }
```

Figura 29 – Exemplo de uso dos operadores de incremento e decremento

Na linha 3 desse programa, as variáveis **x** e **y** são declaradas como inteiras e nas linhas 5 e 6, as mesmas são inicializadas, ambas com o valor 10.

Na linguagem C, o operador de atribuição é o **=**, nas linhas 8 e 9, **x** e **y** foram incrementadas, sendo **x** (linha 8) com o comando de pós-incremento, ou seja, quando o operador unário está posicionado após a variável, e **y** (linha 9) com o comando de pré-incremento, quando o operador está posicionado antes da variável. Em ambos os casos, foi somado 1 às variáveis, e a saída do comando da linha 12, equivalente ao comando escreva, é **x** e **y** iguais, ou seja, **x = 11** e **y = 11**.

Da mesma forma, nas linhas 14 e 15, **x** e **y** são decrementadas, voltando ao valor 10 inicialmente atribuído. A saída do comando da linha 16 é **x = 10** e **y = 10**.

Não há diferença no resultado das operações pré e pós-incremento ou pré e pós-decremento quando os comandos são aplicados numa linha de comando isoladamente. A diferença se dá quando são combinadas com uma atribuição, conforme apresentado no programa do exemplo a seguir.

## Exemplo

Este programa declara duas variáveis inteiras, aplica os operadores de incremento e decremento junto com a atribuição e escreve o resultado.

```
1.  int main()
2.  { // Início do programa
3.      int x, y, z, w;
4.      x = 10;
5.      y = 10;
6.
7.      z = ++x;
8.      w = y++;
9.
10.     printf("\n x = %d, y = %d, z = %d e w = %d", x, y, z, w);
11. } //fim do programa
```

Figura 30 – Exemplo de aplicação dos operadores de incremento e decremento

Nesse exemplo de programa, quatro variáveis inteiras são declaradas e identificadas, respectivamente, por **x**, **y**, **z** e **w**. Nas linhas 4 e 5, as variáveis **x** e **y** são inicializadas com o mesmo valor, 10.

Na linha 7, à variável **z** é atribuída à expressão aritmética **++x**. Nessa linha, duas ações são executadas: o pré-incremento e a atribuição. Da mesma forma, duas ações são executadas na linha 8, a atribuição e o pré-incremento. A pergunta que precisamos responder para a saída das linhas 7 e 8 é:

Dado que  $x = 10$  e  $y = 10$ , qual é o valor das variáveis  $z$  e  $w$  após a execução das linhas 7 e 8 a seguir?

7.	$z = ++x;$
8.	$w = y++;$

Figura 31

Na expressão  $z = ++x$ , primeiro será incrementado o  $x$  depois será realizada a atribuição do resultado para  $z$ . Portanto,  $x$  passa a valer 11 e a  $z$  é atribuído o valor 11. Ao término da execução da linha 7,  $x$  e  $z$  são iguais a 11.

Na expressão  $w = y++$ , primeiro será atribuído o valor de  $y$  à variável  $w$ , ou seja,  $w$  recebe 10, depois  $y$  é incrementado. Ao término da execução,  $w = 10$  e  $y = 11$ .

A saída do comando `printf` equivalente ao comando escreva do pseudocódigo e é  $x = 11$ ,  $y = 11$ ,  $z = 11$  e  $w = 10$ .

Os programas dos dois exemplos anteriores foram escritos em Linguagem C. Para funcionarem, deve ser incluída a instrução **#include <stdio.h>** no cabeçalho do programa, linha 0.

A sintaxe do comando de saída **printf** da Linguagem C exige especificar o formato do dado da variável. No exemplo, **%d** indica a posição em que um valor inteiro decimal será apresentado. Para cada **%d**, uma variável inteira é associada, respeitando a ordem em que é listada.

Pseudocódigo	escreva("x = ", x, "y = ", y, "z = ", z, "e w = ", w)
Linguagem C	printf("\n x = %d, y = %d, z = %d e w = %d", x, y, z, w);

Figura 32 – Exemplo de aplicação do comando de saída em pseudocódigo e na linguagem de programação

Expressões aritméticas com dois ou mais operadores respeitarão a ordem de precedência estabelecida na matemática, sendo executadas primeiro as operações dentro dos parênteses internos e as demais na seguinte ordem: funções potenciação, radiciação, div, mod e operadores unários, multiplicação, divisão, adição e subtração.

## 2.2.3 Operadores relacionais

Os operadores relacionais são utilizados para comparar valores ou variáveis e o resultado da comparação serão sempre os valores lógicos **verdadeiro** e **falso**. É possível comparar valores com variáveis ou constantes. A tabela a seguir apresenta os operadores relacionais e exemplos de expressões relacionais.

**Quadro 5 – Operadores relacionais e exemplos de aplicação**

Operador	Significado	Expressões relacionais	Resultado
>	Maior	23 > 10	Verdadeiro
<	Menor	18 < 5	Falso
>=	Maior ou igual	10 >= 10 18 >= 10 20 >= 40	Verdadeiro Verdadeiro Falso
<=	Menor ou igual	20 <= 20 20 <= 40 20 <= 10	Verdadeiro Verdadeiro Falso
<>	Diferente	5 <> 3 10 <> x x <> y	Verdadeiro Falso, se x = 10 Verdadeiro, se x ≠ y
=	Igual	5 = 3 10 = x x = y	Falso Verdadeiro, se x = 10 Falso, se x ≠ y

As expressões relacionais retornam sempre verdadeiro ou falso. Portanto, uma expressão relacional pode ser atribuída a uma variável do tipo lógica, uma expressão condicional dentro de uma estrutura de decisão, uma condição dentro de uma estrutura de repetição, ou como argumento de um comando de saída. As expressões relacionais são muito importantes em processos de solução de problemas, pois definem desvios ou decisões dentro do processo.

### Exemplo

A figura a seguir demonstra um algoritmo com aplicações de expressões relacionais:

---

```
1.  Algoritmo "Operadores relacionais"
2.  Var
3.      n1, n2 : inteiro
4.      comparacao: logico
5.  Inicio
6.      //entrada
7.      escreval("Número 1: ")
8.      leia(n1)
9.      escreval("Número 2: ")
10.     leia(n2)
11.     //processamento e saida
12.     comparacao←(n1=n2)
13.     escreval("num1", n1, "é igual ", n2, " ", teste)
14.
15.     comparacao←(n1<>n2)
16.     escreval("num1", n1, "é diferente de ", n2, " ", teste)
17.
18.     comparacao←(n1>n2)
19.     escreval("num1", n1, "é maior que ", n2, teste)
20.
21.     //saida
22.     escreval("num1", n1, "é menor que ", n2, n1<n2)
23.     escreval("num1", n1, "é maior ou igual a ", n2, n1=n2)
24.     escreval("num1", n1, "é menor ou igual a ", n2, n1=n2)
25.  FimAlgoritmo
```

---

Figura 33 – Algoritmo com aplicações de expressões relacionais

No exemplo da figura anterior, a variável teste do tipo lógico armazena um valor lógico resultante de uma expressão relacional e tem o seu valor alterado nas linhas 12, 15 e 18. Os comandos de saída das linhas 13, 16 e 19 exibirão **verdadeiro** ou **falso** para cada valor que a variável teste assumir nos comandos das linhas imediatamente inferiores.

O resultado de uma expressão relacional, assim como das expressões aritméticas e lógicas, pode ser armazenado em variável do tipo lógica ou escrito num comando de saída, como é o caso das linhas 22 a 24.

## 2.2.4 Operadores lógicos

Os operadores lógicos são expressos pelos símbolos **e**, **ou** e **não**, que significam, respectivamente, a conjunção, a disjunção e a negação. Esses operadores são utilizados para comparar duas ou mais expressões relacionais e podem ser combinados com expressões aritméticas e relacionais.

Para entender os operadores lógicos, é necessário compreender a lógica matemática que os fundamenta, conhecida como **tabela verdade**.

A tabela verdade é construída a partir de proposições. Uma proposição é uma sentença declarativa que afirma ou nega um fato e o resultado é uma **verdade** ou **falsidade**. Exemplos de proposições são apresentadas a seguir.

**Proposição P:** a porta é de madeira.

Se a proposição fizer referência a uma porta de madeira, a sentença será verdadeira, mas caso a porta em questão seja de qualquer outro material que não madeira, a proposição será falsa. A tabela verdade para uma proposição como a deste exemplo está identificada pela letra **P** e será:

P
V
F

Figura 34

Uma sentença lógica é uma proposição combinada com outra proposição, e a sentença será satisfeita se sua interpretação for **verdadeira**. A combinação de duas ou mais proposições é feita com o uso dos operadores lógicos **e**, **ou** ou **não**, respectivamente: conjunção, disjunção ou negação, conforme apresentado no quadro a seguir:

**Quadro 6 – Operadores lógicos**

Operador	Como se lê	Significado
E	E lógico	Conjunção
Ou	Ou lógico	Disjunção
Não	Não lógico	Negação

## 2.2.5 Operador lógico NÃO

A negação de uma verdade é uma falsidade. Da mesma forma, ao negar uma **falsidade** obtém-se uma **verdade**. A tabela verdade da negação é:

P	NÃO P
V	F
F	V

Figura 35

Considerando que a proposição **P** afirma que a porta é de madeira, negando-se a proposição, obtém-se **NÃO P**, ou a sentença **A porta NÃO é de madeira**. A tabela verdade dessa proposição é:

P	NÃO P
A porta é de madeira	A porta <b>NÃO</b> é de madeira
V	F
F	V

Figura 36

A tabela verdade da negação define que **NÃO P** é verdadeiro se, e somente se, **P** for falso. Adicionalmente, **NÃO P** é falso se, e somente se, **P** for verdadeiro. Considere a **Proposição A: está chovendo**. **A** será verdadeira quando estiver chovendo e falsa quando não chover.

## Quadro 7 – Tabela verdade da negação – operador NÃO

A Está chovendo	NÃO A NÃO é verdade que está chovendo
V	F
F	V

### 2.2.6 Operador lógico E

O operador lógico **E** representa o resultado da conjunção de duas proposições ou duas expressões. Considere que a regra para aprovação em nossa disciplina é a descrita a seguir:

O aluno desta disciplina estará aprovado se obtiver média semestral (MS) maior ou igual a 7,0 (sete) **E** frequência igual ou superior a 75% das aulas.

As proposições que compõem a regra serão representadas pelas letras **A** e **B**, sendo que **A** significa **média maior ou igual a 7,0** e **B** significa **frequência igual ou superior a 75%**. O quadro a seguir apresenta a interpretação de cada linha da tabela verdade desse exemplo.

## Quadro 8 – Tabela verdade da conjunção – operador E

A Média maior ou igual a 7,0	B Frequência igual ou superior a 75%	A e B Aprovados se média maior ou igual a 7,0 E frequência igual ou superior a 75%	Interpretação
V	V	V	Esta linha representa o grupo de alunos que possui média e frequência mínima, portanto, o grupo de aprovados
V	F	F	Esta linha representa o grupo de alunos que possui média, mas não frequência mínima exigida, portanto, um grupo de reprovados



A Média maior ou igual a 7,0	B Frequência igual ou superior a 75%	A e B Aprovados se média maior ou igual a 7,0 E frequência igual ou superior a 75%	Interpretação
F	V	F	Esta linha representa o grupo de alunos que não obtiveram média mínima exigida, apesar de terem frequência mínima, portanto, um grupo de reprovados
F	F	F	Esta linha representa o grupo de alunos que não obtiveram média nem frequência mínima, portanto, um grupo de reprovados

A conjunção representa uma regra rígida cuja expressão será verdadeira se, e somente se, todas as proposições forem verdadeiras.

## 2.2.7 Operador lógico OU

O operador lógico **OU** representa o resultado da disjunção de duas ou mais proposições. A disjunção é falsa se, e somente se, todas as proposições forem falsas.

Suponha que haja um anúncio de oportunidade de emprego recrutando profissionais com formação na área de TI, conforme o seguinte.

### Vaga

Procuram-se profissionais de tecnologia com formação em Ciência da Computação ou Sistemas de Informação.

Para o anúncio do texto apresentado, são habilitados para a vaga todos os candidatos formados em qualquer um dos cursos. Contudo, candidatos com formação em qualquer curso que não Ciência da Computação ou Sistemas de Informação não serão considerados habilitados para concorrer à vaga.

Considere que a proposição **A** representa os **candidatos formados em Ciência da Computação** e a proposição **B** representa os **candidatos formados em Sistemas de Informação**. O resultado da disjunção **verdadeiro** representa as condições em que o candidato estará habilitado para a vaga e **falso** caso não esteja. A tabela verdade da disjunção e a interpretação são apresentadas no quadro a seguir.

**Quadro 9 – Tabela verdade da disjunção – operador OU**

A	B	A ou B	Interpretação
V	V	V	Esta linha representa os candidatos com formação em ambos os cursos e que, por isso, atendem aos requisitos da vaga; portanto o resultado da disjunção é <b>verdadeiro</b>
V	F	V	Esta linha representa os candidatos com formação apenas em Ciência da Computação. O resultado da disjunção <b>A OU B</b> é <b>verdadeiro</b> porque atende a um dos requisitos para a vaga
F	V	V	Esta linha representa os candidatos com formação apenas em Sistemas de Informação. O resultado da disjunção <b>A OU B</b> é <b>verdadeiro</b> porque atende a um dos requisitos para a vaga
F	F	F	Esta linha representa os candidatos que não possuem formação em nenhum dos dois cursos requeridos para a vaga. Portanto, o resultado da disjunção <b>A OU B</b> é <b>falso</b>

O quadro a seguir exemplifica os operadores lógicos e uma única tabela verdade para duas proposições. O número de linhas de uma tabela verdade é  $2^p$ , onde  $p$  é o número de proposições.

**Quadro 10 – Tabela verdade para duas proposições**

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
	A	B	Não A	Não B	A e B	A ou B	Não (A e B)	Não A ou B
(1)	V	V	F	F	V	V	F	V
(2)	V	F	F	V	F	V	V	F
(3)	F	V	V	F	F	V	V	V
(4)	F	F	V	V	F	F	V	V

Esse quadro apresenta os valores verdade para duas proposições. A seguir explicaremos pormenorizadamente o raciocínio lógico para cada coluna desse quadro.

As colunas (a) e (b) representam os valores verdade para duas proposições, ou seja, verifica-se que:

- ambas podem ser verdadeiras (linha 1);
- a primeira pode ser verdadeira e a segunda (linha 2), falsa;
- a primeira pode ser falsa e a segunda (linha 3), verdadeira;
- ambas podem ser falsas (linha 4).

A expressão lógica **não A** da coluna (c) apresenta em cada linha a interpretação da negação da proposição A. A proposição A é verdadeira nas linhas 1 e 2 e:

- negando-se uma verdade, obtém-se uma falsidade (linhas 1 e 2 da coluna (c));
- negando-se uma falsidade, obtém-se uma verdade (linhas 3 e 4 da coluna (c)).

A expressão lógica A e B apresentada na coluna (e) é a conjunção e a sua interpretação será verdadeira sempre que as proposições A e B forem simultaneamente verdadeiras, ou seja, apenas na linha 1.

A expressão lógica A ou B apresentada na coluna (f) é a disjunção e a sua interpretação será verdadeira sempre que uma das proposições A ou B for verdadeira, ou seja, linhas 1, 2 e 3 da tabela verdade. É correto dizer que a disjunção resultará em falsidade somente quando as proposições forem simultaneamente falsas.

Na coluna (g) temos a negação de uma expressão lógica de conjunção. Nesse caso, deve-se resolver a expressão lógica dentro dos parênteses (**A e B**) e depois a negação, conforme a expressão **não (A e B)**. Na tabela verdade do quadro a seguir, embaixo da expressão (**A e B**), com cores apagadas, constam os

resultados da interpretação parcial da expressão. Na linha 1 resolve-se a expressão **não V**, que é igual a F. Nas linhas 2 a 4, **não F** é igual a V.

Na coluna (h) são apresentadas as interpretações da expressão **não A ou B**, sem o uso dos parênteses. Neste caso, resolve-se a negação da proposição A, e depois a disjunção. Uma forma prática de resolver expressões lógicas complexas é rascunhar os resultados das expressões parciais embaixo delas, conforme ilustrado a seguir, e colocar em destaque o resultado final da expressão.

**Quadro 11 – Negação da conjunção e negação de uma proposição da conjunção**

Negação da conjunção		Negação de uma proposição da conjunção		
Não (A e B)		Não A ou B		
(1)	F	V	V	V
(2)	V	F	F	F
(3)	V	V	V	V
(4)	V	V	V	F

Repare: primeiro resolvemos os parênteses internos e depois o operador lógico que está fora. Os valores lógicos verdadeiro (V) e falso (F) em verde e roxo, respectivamente, correspondem ao resultado da expressão dentro do parênteses. Os valores lógicos verdadeiro (V) e falso (F) em letras destacadas em vermelho e azul são o resultado da expressão final.

Por exemplo, na linha 1 da negação da conjunção, a expressão **não (A e B)** significa que conjunção (A e B) está sendo negada, portanto, se A e B é uma verdadeira, a negação obtém uma falsidade. Já na linha 1 da negação de uma proposição da conjunção, a expressão **(não A ou B)**, a proposição A é negada. Veja que A é verdadeiro, logo **não A** é falso porque se uma verdade é negada o resultado é uma falsidade. Dessa forma, para avaliar o resultado da expressão **(não A ou B)**, deve-se primeiro considerar a negação da proposição A, e o resultado analisa a disjunção com B.

## Exemplo 1

Para exemplificar o que dissemos até agora, vamos observar o seguinte exemplo para um operador lógico de negação (**NÃO** lógico):

**A:** O professor ministra aulas de Lógica de Programação de Algoritmos.

**B:** O professor ministra aulas de Lógica Matemática.

**NÃO A:** O professor não ministra aulas de Lógica de Programação de Algoritmos.

**NÃO B:** O professor não ministra aulas de Lógica Matemática.

**Quadro 12 – Tabela verdade da negação com duas proposições**

A	B	não A	não B
V	V	F	F
V	F	F	V
F	V	V	F
F	F	V	V

O quadro anterior corresponde a todas as combinações de possibilidades para as duas proposições A e B, a saber:

- A e B serem verdadeiras;
- A ser verdadeira e B, falsa;
- A ser falsa e B ser verdadeira;
- ambas serem falsas.

Sempre que **A** for verdadeira, **não A** será falsa, ou seja, a negação de uma verdade é uma falsidade. Sempre que **A** for falsa, **não A** será verdadeira pois, negando-se uma falsidade, obtém-se uma verdade.

Podemos exemplificar com a seguinte proposição: **o ser humano é mortal**. Essa proposição é uma sentença afirmativa verdadeira. A negação dessa sentença pode ser apresentada das seguintes formas:

- **não é verdade que o ser humano é mortal;**
- **o ser humano não é mortal.**

As sentenças negativas são formas diferentes de negar a proposição original e resultam numa falsidade, pois é verdade que todo ser humano é mortal.

## Exemplo 2

A situação a seguir exemplifica um operador lógico de conjunção (**E** lógico):

**Sentença:** João vai à praia sempre que tem folga no sábado e o dia está ensolarado.

**Proposição A:** João está de folga.

**Proposição B:** É sábado.

**Proposição C:** O dia está ensolarado.

**Quadro 13 – Tabela verdade da conjunção com três proposições**

A	B	C	A e B e C	Interpretação
V	V	V	V	Esta linha representa que João está de folga, é sábado e o dia está ensolarado. Todas as condições para João ir à praia estão satisfeitas e o resultado da expressão <b>verdadeiro</b> indica que João vai à praia
V	V	F	F	Esta linha representa que João está de folga, é sábado, mas o dia não está ensolarado. Como João vai à praia apenas em dias ensolarados, o resultado da expressão <b>falso</b> indica que João não vai à praia
V	F	V	F	Esta linha representa que João está de folga, o dia está ensolarado, mas <b>não é sábado</b> . Como João vai à praia apenas aos sábados, o resultado da expressão <b>falso</b> indica que João não vai à praia
V	F	F	F	Esta linha representa que João está de folga, mas não é sábado e o dia não está ensolarado e o resultado da expressão <b>falso</b> indica que João não vai à praia
F	V	V	F	Esta linha representa que João não está de folga, é sábado e o dia está ensolarado. Na conjunção, basta que uma proposição seja <b>falsa</b> para que a expressão seja <b>falsa</b> . Não importa que seja sábado ou que o dia esteja ensolarado, pois João não está com tempo para ir à praia
F	V	F	F	Esta linha representa que é sábado, mas João não está de folga e o dia não está ensolarado. O resultado da expressão <b>falso</b> indica que João não vai à praia
F	F	V	F	Esta linha representa que João não está de folga e não é sábado. O resultado da expressão <b>falso</b> indica que João não vai à praia
F	F	F	F	Esta linha representa que João não está de folga, não é sábado e o dia não está ensolarado. O resultado da expressão <b>falso</b> indica que João não vai à praia

## Exemplo 3

A situação a seguir exemplifica um operador lógico de disjunção (ou lógico).

Maria deseja casar com uma pessoa bonita OU inteligente. Os requisitos de Maria são expressos na expressão (**B ou I**), onde **B** significa que **a pessoa é bonita** e **I** significa que **a pessoa é inteligente**.

**Quadro 14 – Tabela verdade da disjunção com duas proposições**

	B	I	(B ou I)	Maria aceita casar com uma pessoa bonita OU inteligente
Grupo de candidatos 1	V	V	V	Os candidatos deste grupo são bonitos e inteligentes e Maria aceita casar-se com eles
Grupo de candidatos 2	V	F	V	Os candidatos deste grupo são bonitos, mas não inteligentes, e Maria aceita casar-se com eles
Grupo de candidatos 3	F	V	V	Os candidatos deste grupo não são bonitos, mas são inteligentes, e Maria aceita casar-se com eles
Grupo de candidatos 4	F	F	F	Os candidatos deste grupo não são bonitos nem inteligentes e Maria não aceita se casar com nenhum deles

## 2.3 Prioridades entre as expressões aritméticas, lógicas e relacionais

As prioridades indicam as precedências das operações quando as expressões combinam operadores aritméticos, lógicos e relacionais. Os operadores aritméticos têm precedência em relação aos operadores relacionais e os relacionais têm precedência em relação aos lógicos.

**Quadro 15 – Prioridades entre os operadores aritméticos, lógicos e relacionais**

Ordem de prioridades entre operadores	
1º aritméticos	Parênteses internos Funções Operadores unários Multiplicação Divisão Adição Subtração
2º relacionais	Igual Diferente <= >= < >
3º lógicos	Não E Ou

Em todos os casos, as expressões dentro de parênteses internos são prioritárias e serão resolvidas primeiro.

A seguir, veremos alguns exemplos de expressões com operadores aritméticos.

### Exemplo 1

Calcule os resultados das seguintes **expressões aritméticas**:

- a)  $5 * 2 - 3 + 14 / 2 + 9$
- b)  $5 - \text{pot}(2,3) + 4 - 2 * \text{rad}(4)$
- c)  $\text{pot}(3-1, 2) - 7 + \text{rad}(8+1) * 2$

### Solução

a)  $5 * 2 - 3 + 14 / 2 + 9$   
 $10 - 3 + 7 + 9$   
 $23$

$$\begin{aligned} \text{b) } & 5 - \text{pot}(2,3) + 4 - 2 * \text{rad}(4) \\ & 5 - 8 + 4 - 2 * 2 \\ & 5 - 8 + 4 - 4 \\ & - 3 \end{aligned}$$

$$\begin{aligned} \text{c) } & \text{pot}(3 - 1, 2) - 7 + \text{rad}(8 + 1) * 2 \\ & \text{pot}(2,2) - 7 + \text{rad}(9) * 2 \\ & 4 - 7 + 3 * 2 \\ & 4 - 7 + 6 \\ & 3 \end{aligned}$$



## Observação

Nos exemplos b) e c), `pot()` e `rad()` são as funções de potenciação e radiciação, respectivamente, e a resolução das funções é prioritária. No VisuAlg, a potenciação é representada por `exp(base, expoente)` e a radiciação pela função `raizq(valor)`.

Agora trataremos de expressões com operadores aritméticos e relacionais.

## Exemplo 2

Calcule o resultado das seguintes expressões:

$$\text{a) } 5 * 2 = 4 + 10 / 2$$

$$\text{b) } 5 \bmod 2 + 3 < \text{pot}(3, 2) * 10$$

$$\text{c) } 5 \text{ div } 2 - 1 \geq 4 / 2 + 7$$

## Solução

$$\begin{aligned} \text{a) } & 5 * 2 = 4 + 10 / 2 \\ & 10 = 4 + 5 \\ & 10 = 9 \\ & \text{F} \end{aligned}$$

$$\begin{aligned} \text{b) } & 5 \bmod 2 + 3 < \text{pot}(3,2) * 10 \\ & 1 + 3 < 9 * 10 \\ & 4 < 90 \\ & \text{V} \end{aligned}$$

c)  $5 \text{ div } 2 - 1 \geq 4/2 + 7$

$2 - 1 \geq 2 + 7$

$1 \geq 9$

F

Nos exemplos a seguir, as expressões b) e c) fazem referência às funções mod e div, as quais são operadores aritméticos auxiliares. O operador mod possui a sintaxe  $x \text{ mod } y$  e retorna o resto da divisão de  $x$  por  $y$ . O operador div possui sintaxe similar,  $x \text{ div } y$ , e retorna o quociente da divisão de  $x$  por  $y$ . Na prática, são funções como os operadores do exemplo anterior e possuem prioridades conforme apresentadas no quadro 15.

## Exemplo 3

Calcular o resultado das seguintes expressões com operadores aritméticos, relacionais e lógicos:

a)  $2 = 10 \text{ mod } 2 \text{ e } -16 > 4$

b)  $3 < 7 \text{ e } 5 * 2 = 2 + 1$

c) Não F e  $\text{rad}(\text{pot}(3,2) < \text{pot}(5,2) \text{ ou } 5 \text{ mod } \text{rad}(49) \geq 81/9$

## Solução

a)  $2 = 10 \text{ mod } 2 \text{ e } -16 > 4$

$2 = \underline{0}$

F

e

F

e

F

F

A expressão  $2 = 10 \text{ mod } 2 \text{ e } -16 > 4$  aplica o operador unário negativo  $-16$  na comparação  $-16 > 4$ . O operador negativo multiplica o inteiro 16 por  $-1$ , invertendo o sinal.

$-16 > 4$  é falso.

b)  $3 < 7 \text{ e } 5 * 2 = 2 + 1$

$3 < 7 \text{ e } 10 = 3$

$\underline{V} \text{ e } F$

F

c) não F e  $\text{rad}(\text{pot}(3,2) < \text{pot}(5,2) \text{ ou } 5 \text{ mod } \text{rad}(49) \geq 81/9$

não F e  $\text{rad}(9) < 25 \text{ ou } 5 \text{ mod } 7 \geq 9$

não F e  $3 < 25 \text{ ou } 5 \geq 9$

$\underline{V}$

V

ou F

ou F

V



## 2.4 O ambiente de desenvolvimento integrado

O ambiente de desenvolvimento integrado (IDE, do inglês integrated development environment) é um aplicativo que contém um editor de texto e um compilador para uma linguagem de programação específica.

Os editores integrados aos compiladores oferecem facilidades para codificação, verificadores sintáticos e atalhos para as tarefas de compilação e verificação do código. É possível, a partir do IDE, executar o programa, depurar o código e dar suporte à indentação. Além disso, esses editores possuem outras facilidades, tais como colorir o código com cores específicas para destacar palavras reservadas e variáveis declaradas, além de detectar variáveis declaradas, mas não utilizadas. Nesta disciplina, o IDE adotado é o VisuAlg.

O VisuAlg é um interpretador e editor de algoritmos, ou seja, um software para editar, interpretar e executar algoritmos escritos exclusivamente em pseudocódigo. Por isso é indicado para iniciantes em programação. A maioria dos exemplos deste livro-texto podem ser executados no VisuAlg.



### Saiba mais

O VisuAlg pode ser obtido gratuitamente no site do produtor:

Disponível em: <https://bit.ly/3zseTju>. Acesso em: 20 set. 2021.

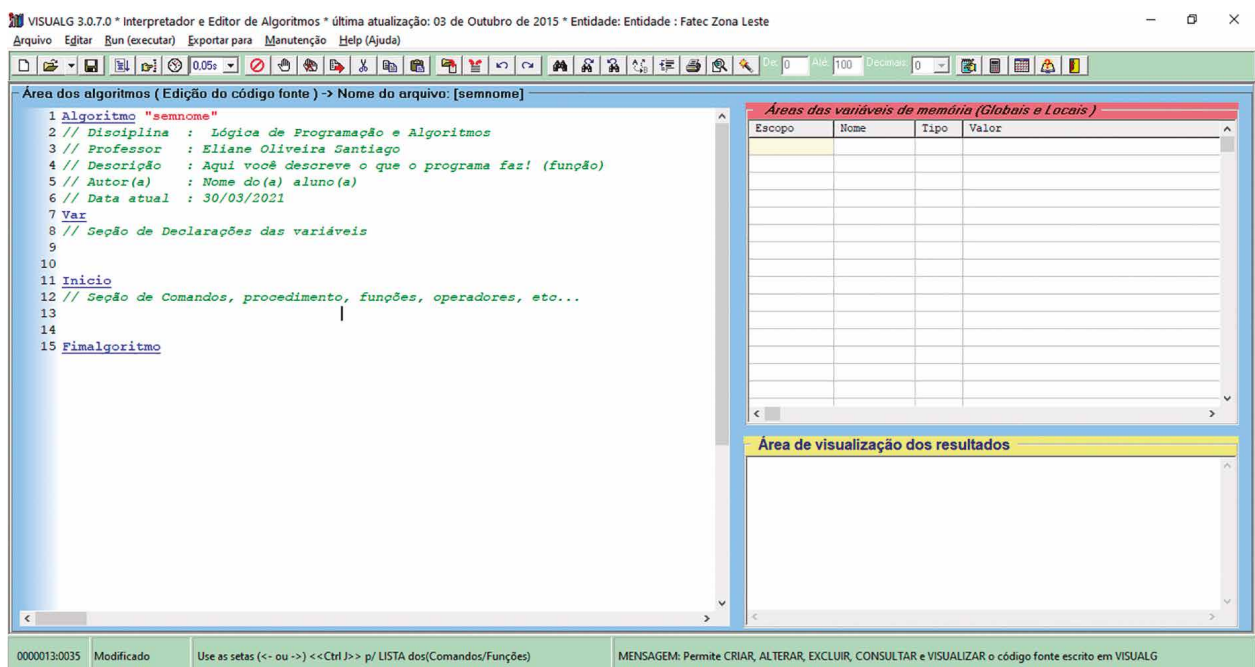


Figura 37 – Interface do ambiente integrado de desenvolvimento de algoritmos em pseudocódigo (português estruturado) do VisuAlg

Os algoritmos codificados nessa interface podem ser salvos (CTRL+S) e executados (F9) a partir do IDE. Existem muitos IDEs disponíveis que suportam várias linguagens de programação, mas não muitos para pseudocódigo. À medida que o estudante evoluir seus conhecimentos em lógica de programação, os algoritmos poderão ser escritos diretamente em linguagem de programação.

### 2.5 Comentários no corpo do programa

A lógica de programação pode ser complexa e, por vezes, faz-se necessário um texto escrito em linguagem natural para explicar a ideia codificada. Esse tipo de texto deve ser ignorado pelo compilador, mas é útil para o programador entender alguma regra ou propósito de um determinado bloco (ou trecho) de programação.

A dupla barra de divisão `//` (que aparece em verde nos trechos de código) é usada como um marcador de comentários de uma única linha. Significa que apenas a linha marcada será ignorada. Os pseudocódigos apresentados até aqui possuem comentários de uma linha, cujo propósito é apresentar quais comandos correspondem à entrada, ao processamento e à saída do programa.

No entanto, por vezes, é necessário um texto maior, tal como um cabeçalho de início de programa, ou uma explicação sobre uma regra. Nesse caso, os marcadores que delimitam o início e o fim do comentário são, respectivamente, `/*` e `*/`.

Os IDEs modificam as cores das linhas comentadas para verde ou cinza-claro, destacando linhas de código comentadas ou a serem ignoradas. Veja a seguir dois exemplos de cabeçalhos sugeridos para serem inseridos no início de cada algoritmo ou programa.

```
/*  
 *  
 * Programa.....: Calcular área do Retângulo  
 * Autor(a).....: Eliane Oliveira Santiago  
 * Data de criação...: 26/01/2021  
 * Última alteração..: 05/02/2021  
 * Responsável.....: a autora  
 */
```

Figura 38 – Exemplo de cabeçalho

Os delimitadores de blocos `/*` e `*/` representam, respectivamente, o início e o fim do bloco. Os blocos de comentários no início do código são úteis para apresentar um cabeçalho para o programa, conforme os exemplos da figura anterior e da seguinte, ou para explicações mais detalhadas no corpo do código.

```
/      *
      * Programa.....: Lab01 – Sistema de Cálculo do Seguro Desemprego
      * Aluno(a).....: Ricardo Júnior (RA A9989-1)
      * Disciplina.....: LPA
      * Professor(a).....: Eliane Oliveira Santiago
*      /
```

Figura 39 – Exemplo de cabeçalho

Os delimitadores de blocos de comentários não são reconhecidos pelo VisuAlg, mas são reconhecidos pelas linguagens de programação derivadas da linguagem C, tais como C++ e Java.

## 2.6 Blocos de programação e indentação

Os fluxogramas ajudam a descrever procedimentos delimitados entre os objetos **início** e **fim**. Da mesma forma, um bloco de programação é um conjunto de procedimentos descritos em forma de comandos delimitados pelos marcadores **início** e **fim**. A indentação é uma boa prática que ajuda a alinhar o código dentro dos blocos com os devidos recuos ou avanços.

O VisuAlg possui o atalho CTRL+G, que faz a indentação automática. No entanto, é importante, como programador, escrever os códigos e ao mesmo tempo já ir formatando o texto para que o código fique limpo e compreensível.

```
Início
comando1
    comando2
    :
    :
    comandoN
Fim
```

Figura 40 – Exemplo de programa indentado

Cada bloco pode ser identificado pela indentação. Níveis de indentação ajudam na leitura e compreensão do código, pois cada nível permite ao programador identificar os desvios, início e fim dos laços de repetição ou módulos do programa. Essas estruturas serão temas de nossas discussões mais adiante em nosso percurso de estudos. A cada bloco, um novo deslocamento à esquerda deve ser dado para que os blocos fiquem alinhados.



## Lembrete

Não confunda **programação em blocos** com **blocos de programação** (ou **blocos de programas** ou simplesmente **blocos**).

Um **bloco de programação** é um trecho de código a ser executado de maneira sequencial, indentado e delimitado pelos marcadores início e fim. Em algumas linguagens de programação, os delimitadores não são explícitos, mas apenas implícitos pela indentação. Já a **programação em blocos** é uma metodologia para ensinar os conceitos iniciais da programação de computadores. Ela não será alvo dos nossos estudos nesta disciplina.

## 2.7 Instâncias do problema

Um programa de computador é "correto" quando gera saídas corretas para diferentes instâncias do problema que se propõe a resolver. Até este ponto de nosso livro-texto, dois algoritmos foram apresentados: o primeiro calcula a área de retângulos e o segundo calcula a média aritmética de quatro notas.

Cada retângulo da figura seguinte corresponde a uma instância do problema. Cada um contém uma área específica. O algoritmo que calcula a área do retângulo estará correto se para quaisquer valores de entrada, a saída for sempre correta.

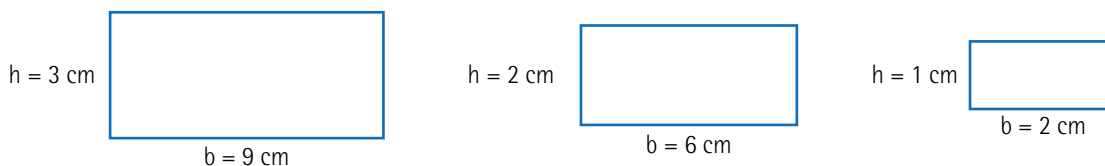


Figura 41 – Instâncias do problema

## 2.8 Algoritmos sequenciais

Algoritmos são comandos executados numa sequência lógica visando à solução de um problema. Cada comando é executado na ordem em que é escrito, por isso o algoritmo é chamado de sequencial.



## Lembrete

Os algoritmos definem uma sequência lógica para execução de comandos a fim de resolver um problema. A sequência de comandos de entrada, processamento e saída pode sofrer desvios ou ser repetida. As estruturas de decisão selecionam um bloco de algoritmo a ser executado conforme o resultado do teste condicional, enquanto as estruturas de repetição iteram repetidas vezes pelo bloco de código até que a condição seja satisfeita.

A partir de agora serão apresentados exemplos de algoritmos sequenciais que aplicam e combinam todos os conceitos estudados até agora.

## Exemplo 1

Escreva um algoritmo que receba três números inteiros, calcule e escreva o valor da equação de primeiro grau dada pela forma  $y = ax + b$ , a variável  $x$  é o fator de multiplicação.

```
1.  Algoritmo "Equação de primeiro grau"
2.  Var
3.    x, y, a, b: inteiro
4.  Inicio
5.    //entrada
6.    escreva("Digite o valor de A.: ")
7.    leia(a)
8.
9.    escreva("Digite o valor de B.: ")
10.   leia(b)
11.
12.   escreva("Digite o valor de X.: ")
13.   leia(x)
14.   //processamento
15.   y <- a*x+b
16.   //saida
17.   escreva( "O valor de y é:", y)
18.  Fimalgoritmo
```

Figura 42 – Algoritmo para calcular uma função de primeiro grau

Nesse exemplo de solução para calcular uma equação de primeiro grau, na linha 3 estão declaradas quatro variáveis do tipo inteiro (x, y, a, b), sendo que três delas serão usadas como entrada (x, a, b) e a última como atribuição do resultado a ser calculado.

As linhas 6, 8 e 10 são mensagens para interação com o usuário, para que ele entenda o que o sistema espera que ele faça, ou seja, digitar um valor para atribuir a cada uma das variáveis para resolver a equação proposta no problema.

Nas linhas 7, 9 e 11, o valor digitado pelo usuário é capturado e atribuído à variável especificada. Na linha 14, o comando para execução do processamento é efetuado e seu resultado, atribuído à variável y. A linha 17 é um comando de saída, e o resultado do valor calculado é apresentado.

### Exemplo 2

Escreva um algoritmo para calcular a média aritmética de quatro notas bimestrais. O algoritmo deve receber quatro números reais, calcular e escrever a média.

```
1.  Algoritmo "Média aritmética"
2.  Var
3.    nota1, nota2, nota3, nota4, media : real
4.
5.  Inicio
6.    //entradas
7.    escreva("Digite a nota do 1o bimestre: ")
8.    leia(nota1)
9.
10.   escreva("Digite a nota do 2o bimestre: ")
11.   leia(nota2)
12.
13.   escreva("Digite a nota do 3o bimestre: ")
14.   leia(nota3)
15.
16.   escreva("Digite a nota do 4o bimestre: ")
17.   leia(nota4)
18.
19.   //processamento
20.   media <- (nota1+nota2+nota3+nota4)/4
21.
22.   //saida
23.   escreva("A média é ", media)
24. Fimalgoritmo
```

Figura 43 – Algoritmo para calcular a média de quatro notas

Nesse algoritmo, a média aritmética é obtida a partir das quatro notas digitadas pelo usuário. Na linha 3, estão declaradas as variáveis que comporão o armazenamento de valores ora digitados pelo usuário, ora atribuídos pelo cálculo da média, bem como a tipificação dessas variáveis, que são do tipo **real**.

As linhas 7, 10, 13 e 16 são mensagens de interação com o usuário. Já as linhas 8, 11, 14 e 17 capturam o valor digitado e o atribuem à respectiva variável. Na linha 20, o cálculo é efetuado e atribuído à variável média para que, na linha 23, seja exibido o resultado. Os comandos de saída são apresentados para o usuário na tela.

## Exemplo 3

Escreva um algoritmo que concatene duas palavras. O algoritmo deve receber duas palavras, concatená-las e escrevê-las juntas.

```
1. Algoritmo "Concatenar Nome e Sobrenome"  
2. Var  
3.   nome, sobrenome, nome_completo : caractere  
4.  
5. Inicio  
6.   //entrada  
7.   escreva("Nome: ")  
8.   leia(nome)  
9.  
10.  escreva("Sobrenome: ")  
11.  leia(sobrenome)  
12.  
13.  //processamento  
14.  nome_completo <- nome + " " + sobrenome  
15.  
16.  //saida  
17.  escreva("Nome completo : ", nome_completo)  
18. Fimalgoritmo
```

Figura 44 – Algoritmo para concatenar duas palavras

Nessa solução, a concatenação entre dois valores é feita com o operador "+". Ele age de forma diferente entre palavras e números. Números são somados e palavras são concatenadas, isto é, juntadas. A variável **nome\_completo** recebe a junção de nome e sobrenome.

As linhas 7 e 10 da figura anterior são mensagens de interação com o usuário, cujo propósito é o usuário saber o que o programa espera que ele digite. Os programas de computador executam os comandos na sequência em que são digitados e fazem desvios de acordo com condições aplicadas na lógica de programação. As mensagens de interação estabelecem um diálogo entre o programa de computador e o usuário e, nessa interação, o programa é resolvido.

## Exemplo 4

Escreva um algoritmo que calcule o volume de um paralelepípedo. Esse algoritmo deve receber três valores reais, um para comprimento, um para largura e um para altura, depois deve apresentar o resultado calculado.

```
1.  Algoritmo "Volume do paralelepípedo"
2.  Var
3.    comprimento, largura, altura : real
4.
5.  Inicio
6.    //entrada
7.    escreva("Comprimento do paralelepípedo... (C): ")
8.    leia(comprimento)
9.
10.   escreva("Largura do paralelepípedo..... (L): ")
11.   leia(largura)
12.
13.   escreva("Altura do paralelepípedo..... (A): ")
14.   leia(altura)
15.
16.   //processamento e saída
17.   escreva("O Volume do Paralelepípedo é: ", comprimento*largura*altura)
18. Fimalgoritmo
```

Figura 45 – Algoritmo para calcular volume de um paralelepípedo

Poderia ter sido declarada uma variável do tipo real denominada volume para armazenar o resultado da expressão aritmética da forma a seguir:

```
volume <- comprimento*largura*altura
```

O objetivo desse exemplo é mostrar que é possível armazenar ou simplesmente calcular o volume e mostrar o resultado na interface do usuário. O comando de saída **escreva()** da linha 17 sintetiza o processamento e a saída do algoritmo. Isso é possível porque a saída do comando **escreva** irá concatenar o que está entre aspas (a mensagem) como também o que está descrito em sua sequência (neste caso, a multiplicação entre os valores das variáveis de entrada).

A desvantagem desse tipo de especificação de comando se dá pelo não armazenamento do resultado, que é o valor do volume calculado. Não há alocação de espaço de memória para o armazenamento, mas também não é possível reusar a informação sem calculá-la novamente.

Nesse exemplo foram declarados:

- na linha 3, três variáveis do tipo real,
- nas linhas 7, 10 e 13, os comandos de saída para a interação com o usuário;



- nas linhas 8, 11 e 14, os comandos de entrada para captura e atribuição às respectivas variáveis;
- na linha 17, o comando de saída para exibir na interface do usuário a mensagem e o resultado da expressão aritmética.



## Resumo

Definimos os principais conceitos básicos para começar a escrever algoritmos sequenciais, com exemplos para resolver problemas simples.

Programar é projetar soluções de problemas. O desafio do projeto de algoritmo é ser compreensível para dois atores, o usuário e o programador.

O usuário interage com o sistema, cadastrando dados com o uso de um teclado, acionando comandos usando o mouse, ou capturando imagens com sua webcam. Já o programa interage com o usuário por meio da interface projetada na tela do computador, apresentando mensagens ao usuário que servirão de estímulos para as ações de resposta.

O projeto de solução de problemas estrutura o processo de resolução em termos das operações e das interações do usuário com o programa, estabelecendo uma comunicação entre usuário e programa de forma a possibilitar ao usuário informar os dados de entrada necessários para a resolução.

O segundo ator é o próprio programador ou outro programador que será responsável por dar manutenção ou implementar uma nova funcionalidade, atualizá-la ou corrigi-la. Entender a lógica escrita no computador requer códigos simples, com identificadores de variáveis, constantes e módulos significativos, sem repetições desnecessárias e com comentários mínimos, necessários e suficientes.

Códigos muito comentados também atrapalham na leitura e compreensão da lógica da programação. Como boa prática de programação, evite códigos duplicados ou comentários excessivos.

A qualidade do algoritmo pode ser avaliada pela eficácia e eficiência. Um algoritmo é eficaz se ele resolve o problema para o qual foi desenvolvido, é eficiente quando o resolve da melhor forma, ou seja, com o menor número de comandos, a maior velocidade de processamento, ou o menor tempo de resposta.



## Exercícios

**Questão 1.** Analise atentamente o algoritmo apresentado a seguir, escrito em Portugol com a sintaxe do VisuAlg.

```
Algoritmo "Pergunta1"
Var
    N1, N2, resposta: inteiro
Inicio
    escreva("Digite o primeiro número: ")
    leia(N1)
    escreva("Digite o segundo número: ")
    leia(N2)
    N2 <- 2*N1
    resposta <- N1 + N2
    escreva("O resultado é : ", resposta)
FimAlgoritmo
```

Figura 46

Com base no código e nos seus conhecimentos, avalie as asserções e a relação entre elas.

I – Independentemente da segunda entrada do usuário, o número apresentado como resultado ao final da execução do algoritmo sempre será igual a três vezes o valor da primeira entrada do usuário.

porque

II – Na linha **N2<-2\*N1**, o valor da segunda variável (N2) inserido pelo usuário é sobrescrito. O novo valor atribuído é igual a duas vezes o primeiro valor inserido pelo usuário (N1). Na linha seguinte, esse valor é somado a N1 e atribuído à variável **resposta**.

Assinale a alternativa correta.

- A) As asserções I e II são verdadeiras, e a asserção II justifica a asserção I.
- B) As asserções I e II são verdadeiras, e a asserção II não justifica a asserção I.
- C) A asserção I é verdadeira, e a asserção II é falsa.
- D) A asserção I é falsa, e a asserção II é verdadeira.
- E) As asserções I e II são falsas.

Resposta correta: alternativa A.

## Análise das asserções

I – Asserção verdadeira.

Justificativa: se executarmos o algoritmo no VisuAlg ou simularmos manualmente a sua execução, veremos que o resultado apresentado na saída é sempre igual ao triplo da primeira entrada do usuário, independentemente do valor que colocarmos na segunda entrada do programa. Observe que a última saída corresponde à impressão do valor armazenado na variável "resposta", cujo valor é atribuído na linha imediatamente anterior ao último comando "escreva". Nessa linha, o valor atualizado de N2, que é igual a duas vezes N1, é somado a N1, resultando em 3 vezes N1.

II – Asserção verdadeira.

Justificativa: a atribuição **N2←-2\*N1** faz com que o programa perca o segundo número entrado pelo usuário, que estava armazenado na variável N2. Esse valor é substituído pelo dobro do valor armazenado na variável N1. Na linha seguinte, **resposta←-N1+N2**, esse novo valor de N2 é somado à variável N1, efetivamente é feita a operação **resposta ←-N1+ (2\*N1)**, que é equivalente à **resposta ←- (3\*N1)**, ou seja, é o triplo da primeira entrada do usuário. Note que o segundo valor entrado pelo usuário é descartado.

Devemos observar também que a segunda asserção, além de estar correta, é a justificativa do comportamento descrito na primeira asserção. Dessa forma, podemos afirmar que ambas as asserções estão corretas e que a segunda justifica a primeira.

**Questão 2.** Considere o fluxograma a seguir.

Suponha que um usuário execute um programa que implemente um algoritmo com a lógica exposta no fluxograma a seguir:

Considere o seguinte exemplo de execução: um usuário, ao interagir com o programa, entra com os seguintes valores: 2 na primeira interação, 5 na segunda interação e 7 na terceira e última interação. Nesse contexto, avalie as afirmativas:

I – O resultado da execução do programa vai mostrar na tela os valores 2, 5 e 7, anteriormente inseridos pelo usuário.

II – O resultado da execução do programa vai mostrar na tela os valores 10, 20 e 30, definidos internamente no programa.

III – O resultado da execução do programa vai mostrar na tela apenas o número 7, inserido pelo usuário na sua última interação.

IV – O resultado da execução do programa vai mostrar na tela apenas o número 30, definido internamente no programa.

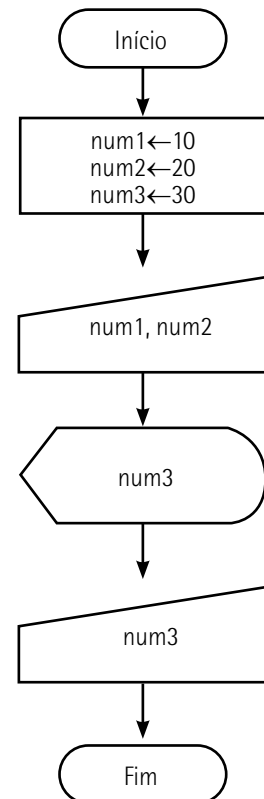


Figura 47

É correto o que se afirma apenas em:

- A) I.
- B) II.
- C) III.
- D) IV.
- E) I e II.

Resposta correta: alternativa D.

### Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: ao observarmos com atenção o fluxograma, notaremos que apenas a variável **num3** é mostrada na tela para o usuário. Dessa forma, mesmo sem levarmos em consideração outros aspectos da lógica do algoritmo, podemos afirmar que os dois primeiros números inseridos pelo usuário certamente não serão exibidos.

II – Afirmativa incorreta.

Justificativa: além do raciocínio utilizado na afirmativa I, podemos observar que os valores atribuídos às variáveis **num1** e **num2** são alterados pelo usuário antes de qualquer impressão, o que significa que o programa não poderia mostrar os valores originais dessas variáveis.

III – Afirmativa incorreta.

Justificativa: aqui, é necessário tomarmos cuidado com a ordem das operações no algoritmo: a variável **num3** é mostrada antes de o valor ser inserido pelo usuário. Isso significa que, nesse caso, é o seu valor original (definido internamente no programa) que vai ser mostrado ao usuário na tela.

IV – Afirmativa correta.

Justificativa: como foi explicado na justificativa da afirmativa III, o valor a ser mostrado para o usuário é o conteúdo inicial da variável **num3**, que é igual a 30. Logo após esse valor ser mostrado para o usuário, o programa deve permitir que um novo valor seja atribuído a essa variável. Contudo, logo após essa atribuição, o programa se encerra sem mostrar mais nada para o usuário.