



# Interativa

## Linguagem e Técnicas de Programação

**Autor:** Prof. Olavo Ito

**Colaboradores:** Prof. Luciano Soares de Souza  
Prof. Marcelo Nogueira

## Professor conteudista: Olavo Ito

Mestre em Engenharia de Produção pela UNIP, bacharel em Física pelo Instituto de Física da USP, licenciado em Ciências pela Faculdade de Educação da USP e bacharel em Língua e Literatura Japonesa pela FFLCH-USP.

Os primeiros passos do autor no mundo da programação de computadores foram dados com calculadoras programáveis HP25C e Sharp Pocket Computer. Nessa mesma época, aprendeu lógica de programação, perfurando cartões para praticar seus conhecimentos em computador de grande porte (Burroughs B6700).

Possui experiência com linguagem Fortran e teve contato com um dos dois únicos computadores IBM 360 remanescentes no mundo, na época, pertencente ao laboratório do acelerador de partículas Pelletron. Também teve a oportunidade de trabalhar com um Itautec I-7000, que tinha tanto poder de processamento quanto o gigantesco IBM360 e permitia o uso de outras linguagens de programação além de Fortran: Basic, xBase, C, Pascal, Algol, COBOL e muitas outras.

A experiência com programação levou-o a sair da vida acadêmica para trabalhar na iniciativa privada, mas sem deixar de lado o compromisso com o ensino. Trabalhou em banco, consulado, holdings, indústria e comércio, especializando-se em implantação de Enterprise Resource Planning (ERP) – Sistemas de Gestão Empresarial.

Professor dos cursos de graduação tradicional de Sistemas de Informação e Ciência da Computação, bem como do curso superior tecnológico de Análise e Desenvolvimento de Sistemas. Das disciplinas que ministra, *Linguagem e Técnicas de Programação* é especial, pois engloba dois assuntos que tem especial prazer em lecionar: Algoritmos e Estrutura de Dados.

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

I96L Ivo, Olavo.  
Linguagens e técnicas de programação. / Olavo Ivo. – São Paulo, 2014.  
488 p., il.  
Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ano XX, n. 2-082/14, ISSN 1517-9230.  
1. Técnicas de programação. 2. Estrutura de dados. 3. Estrutura unidimensional. I. Título.

CDU 681.3.02

Prof. Dr. João Carlos Di Genio  
**Reitor**

Prof. Fábio Romeu de Carvalho  
**Vice-Reitor de Planejamento, Administração e Finanças**

Profa. Melânia Dalla Torre  
**Vice-Reitora de Unidades Universitárias**

Prof. Dr. Yugo Okida  
**Vice-Reitor de Pós-Graduação e Pesquisa**

Profa. Dra. Marília Ancona-Lopez  
**Vice-Reitora de Graduação**

### **Unip Interativa – EaD**

Profa. Elisabete Brihy  
Prof. Marcelo Souza  
Prof. Dr. Luiz Felipe Scabar  
Prof. Ivan Daliberto Frugoli

### **Material Didático – EaD**

Comissão editorial:

Dra. Angélica L. Carlini (UNIP)  
Dra. Divane Alves da Silva (UNIP)  
Dr. Ivan Dias da Motta (CESUMAR)  
Dra. Kátia Mosorov Alonso (UFMT)  
Dra. Valéria de Carvalho (UNIP)

Apoio:

Profa. Cláudia Regina Baptista – EaD  
Profa. Betisa Malaman – Comissão de Qualificação e Avaliação de Cursos

Projeto gráfico:

Prof. Alexandre Ponzetto

Revisão:

Juliana Maria Mendes  
Amanda Casale



# Sumário

## Linguagem e Técnicas de Programação

APRESENTAÇÃO .....	9
INTRODUÇÃO .....	10

### Unidade I

1 ALGORITMOS.....	13
1.1 Conceitos básicos.....	13
1.2 Programas.....	14
1.3 Lógica de programação.....	14
1.4 Representação lógica.....	15
1.5 Tipos de algoritmos.....	15
1.5.1 Descrição narrativa .....	15
1.5.2 Fluxograma.....	16
1.5.3 Pseudocódigo ou Portugol.....	17
1.6 Regras para construção do algoritmo .....	18
1.7 Introdução à programação.....	20
1.7.1 Tipos de informações, entrada e saída.....	20
1.7.2 Dados.....	21
1.7.3 Tipos de dados.....	21
1.8 Programação sequencial .....	33
1.8.1 Estrutura sequencial.....	34
1.9 Estrutura de um pseudocódigo .....	34
1.10 Entrada e saída de informações.....	37
1.10.1 Instrução primitiva de saída de dados.....	37
1.10.2 Instrução primitiva de entrada de dados .....	39
1.10.3 Teste de mesa.....	41

### Unidade II

2 INTRODUÇÃO À LINGUAGEM C.....	53
2.1 Histórico.....	54
2.2 Regras da linguagem.....	54
2.2.1 Estrutura de um programa em C.....	57
2.3 Tipos de informações, variáveis e constantes.....	58
2.4 Operadores.....	62
2.5 Entrada e saída de informações.....	66

## Unidade III

3 TOMADA DE DECISÕES.....	76
3.1 Teoria .....	76
3.2 Condicionais.....	76
3.2.1 Se então.....	76
3.2.2 Se então senão.....	84
3.2.3 Estrutura condicional se aninhada.....	92
3.2.4 Escolha-caso.....	121
3.3 Laboratório.....	150
3.3.1 Escolha-caso na linguagem C++ .....	157

## Unidade IV

4 LAÇOS DE REPETIÇÃO.....	165
4.1 Teoria .....	165
4.1.1 Laços condicionais.....	165
4.1.2 Laços contados .....	208
4.2 Laboratório.....	227
4.2.1 Repita até que.....	227
4.2.2 Enquanto faça.....	229
4.2.3 Para até que .....	231

## Unidade V

5 ESTRUTURA DE DADOS.....	237
5.1 Dados homogêneos.....	237
5.1.1 Vetores, <i>strings</i> e matrizes .....	237
5.1.2 Vetores em C.....	262
5.1.3 Matrizes .....	264
5.1.4 Matrizes em C .....	265
5.1.5 Cadeias de caracteres em C.....	266
5.1.6 Caracteres em C.....	267
5.1.7 Manipulação de <i>strings</i> .....	268
5.2 Dados heterogêneos.....	273
5.2.1 Estrutura em C.....	275
5.2.2 Tipo união .....	276
5.3 Modularização.....	276
5.3.1 Teoria .....	277
5.3.2 Funções e procedimentos.....	278
5.3.3 Funções e procedimentos em C.....	319
5.3.4 Variáveis locais e variáveis globais .....	321

## Unidade VI

6 ALOCAÇÃO DINÂMICA DA MEMÓRIA.....	326
6.1 O uso da memória.....	326
6.2 Ponteiro de variáveis.....	330

6.3 Passagem de valores por valor e por referência .....	334
6.4 Passagem de vetores para funções .....	347
6.5 Funções da biblioteca-padrão .....	348
6.6 Recursividade .....	353

## Unidade VII

7 ESTRUTURA UNIDIMENSIONAL, LISTAS .....	370
7.1 Listas ligadas .....	371
7.1.1 Função de inicialização .....	373
7.1.2 Função de inserção .....	374
7.1.3 Função de remoção .....	384
7.1.4 Função de busca, função de impressão e função de liberação de memória .....	395
7.2 Pilhas .....	398
7.2.1 Implementação de pilha com lista – representação encadeada .....	399
7.2.2 Criação da pilha com lista .....	400
7.2.3 Inserção na pilha com lista .....	402
7.2.4 Remoção da pilha .....	407
7.2.5 Função de impressão e função de liberação de memória .....	414
7.3 Filas .....	417
7.3.1 Implementação de fila com lista – representação encadeada .....	418
7.3.2 Criação da fila com lista .....	419
7.3.3 Inserção na pilha com lista .....	420
7.3.4 Remoção de nó da fila .....	429
7.3.5 Função de impressão e função de liberação de memória .....	432

## Unidade VIII

8 ÁRVORES .....	440
8.1 Árvores binárias .....	441
8.2 Percurso em árvores binárias .....	442
8.2.1 Percurso em pré-ordem ou prefixo (busca em profundidade) .....	442
8.2.2 Percurso em ordem ou infixado (ordem simétrica) .....	450
8.2.3 Percurso em pós-ordem ou posfixo .....	454
8.3 Árvores binárias de busca .....	459
8.3.1 Operação de busca .....	461
8.3.2 Operação de inserção .....	462
8.3.3 Operação de remoção .....	463
8.3.4 Remoção de folha .....	464
8.4 Estrutura de árvores binárias em C .....	467





## APRESENTAÇÃO

A disciplina de *Linguagem e Técnicas de Programação* tem duas partes. A primeira será de estudo da técnica de programação de computadores, a chamada programação estruturada. Nela veremos os comandos básicos que qualquer das linguagens de programação deve ter, bem como usaremos o raciocínio lógico para fazer o computador executar aquilo que queremos.

Inicialmente, apresentaremos a teoria, ou seja, o desenvolvimento da lógica da programação utilizando o chamado pseudocódigo, numa linguagem em português cuja principal função é montar a lógica do programa e testar manualmente o funcionamento deste. A seguir, abordaremos a tradução para uma linguagem de programação, que, no nosso caso, será a linguagem C++, quando efetivamente será possível fazer o computador perder sua característica de "irracional" e executar aquilo que foi programado para fazer. A Unidade I é específica para apresentar a lógica de programação, e a Unidade II é específica para fazer a introdução à linguagem C. Nas Unidades III e IV, é apresentado o controle do fluxo dos programas. Em cada uma delas será feita uma apresentação teórica e, no final de cada uma, a passagem da teoria para a linguagem de programação.

Uma vez entendida a programação básica, passaremos à análise das estruturas clássicas de programação. Serão apresentados programas básicos, simplificados ao máximo, porém com um alto grau de sofisticação. Esses programas serão desenvolvidos utilizando apenas a linguagem C++. Na Unidade V, será feita a transição da programação sequencial simples para a programação modular, bem como a apresentação de estruturas mais sofisticadas de armazenamento na memória. Com isso, será necessário conhecer o processo de gerenciamento da memória; assim, na Unidade VI, a linguagem C, por ser muito próxima à linguagem de máquina, sem perder a facilidade de entendimento da lógica, passará a ser a ferramenta principal para o restante do curso.

Utilizando a linguagem C e usando um modelo fictício, porém didático, de um mapeamento da memória de um computador, estudaremos nas Unidades VII e VIII os principais modelos de estruturas de dados: lista ligada, pilha, fila e árvores.

É importante observar que este não é um "curso de linguagem C", a qual será apenas um exemplo de linguagem de programação que o computador entende. A intenção é desenvolver a lógica de programação no chamado pseudocódigo, a fim de, posteriormente, traduzi-lo para qualquer linguagem de programação.

Existem programadores que escrevem diretamente em uma linguagem de programação, sem usar o pseudocódigo. Isso é um risco muito grande, pois a história mostra que sempre existiu uma linguagem popular que, depois, foi abandonada e substituída por outra. Aconteceu isso com diversas linguagens, como COBOL, FORTRAN, Pascal, Basic, a família xBase (Dbase, Clipper) e muitas outras, e nada indica que as novas linguagens deixarão de ser desenvolvidas. O uso preliminar do pseudocódigo permite que o foco seja a lógica, e não a linguagem. Outro problema comum em programar diretamente em uma linguagem de programação é a perda da linha de raciocínio. As linguagens de programação são muito sensíveis à sintaxe e à correta escrita de seus comandos. Um erro de sintaxe sempre causa uma falha,

impedindo o funcionamento do programa. Assim, ocorrerá uma interrupção do raciocínio lógico do programador. Um pseudocódigo nunca apresentará erro de sintaxe.

O importante é o programador ter uma sólida base de lógica de programação, e não ser um especialista em uma linguagem de programação.



### Saiba mais

Todos os programas apresentados neste livro-texto foram testados no Microsoft Visual C++ 2010 Express, que pode ser instalado gratuitamente. O *download* pode ser efetuado no *site*:

MICROSOFT VISUAL STUDIO. *Downloads do Visual Studio*. s.l., 2014. Disponível em: <<http://www.visualstudio.com/downloads/download-visual-studio-vs>>. Acesso em: 14 abr. 2014.

Em caso de mudança no *link*, os programas poderão ser procurados navegando pelo *site*: <<http://www.visualstudio.com>>. Além do Visual C++, os programas foram testados no Borland Turbo C++. Não recomendamos outras versões, pois, além de os programas não terem sido testados, podem acontecer diferenças no processo de gerenciamento da memória.

## INTRODUÇÃO

O computador é totalmente irracional, incapaz de executar qualquer tarefa se não houver algo ou alguém que o instrua quanto aos passos para efetuar-lá. Ao mesmo tempo, é extremamente fiel: a máquina obedecerá às suas ordens exatamente da maneira que foram expressas. Por um lado, essa obediência é muito louvável, mas, por outro, isso significa que uma ordem mal dada será executada fielmente, da maneira que foi instruído.

Um outro problema é o fato de o computador entender poucas palavras, significando que existe a possibilidade de uma ordem nossa não estar no vocabulário dele, portanto devemos ensinar-lhe novas palavras. Essa falta de vocabulário, a princípio, pode parecer um problema sério, porém é uma vantagem, pois, apesar da necessidade de um texto maior, evita a dupla interpretação.

Com o vocabulário limitado é necessário fazer programas que tornem amigável a experiência de uma pessoa leiga ao utilizar o computador. Nossa função ao fazer este curso é montar programas, sistemas que permitam ao usuário leigo a possibilidade de trabalhar com o computador sem a necessidade de conhecer os detalhes da máquina. Apresentar soluções computacionais sem uma intervenção nossa (especialistas).

Quem usa uma planilha eletrônica não precisa saber computação, mas "alguém" teve de programar para que a planilha fosse funcional. Quem se diverte em um jogo eletrônico não precisa saber dos detalhes da programação do computador, mas "alguém" teve de dar aos elementos do jogo instruções para o comportamento durante a partida. Quem usa um celular não precisa saber que existe uma conversa digital entre o aparelho e a operadora antes de uma ligação se completar, pois "alguém" deixou montado um processo que permite ao aparelho tomar uma série de decisões antes de ligar ou recusar uma conexão. Num mundo cada vez mais digital, existem vários programas que ficam executando instruções que diminuem as operações manuais, os chamados programas embarcados, e alguém os automatizou, por meio da programação de computador; todos esses profissionais, no mínimo, devem conhecer a lógica de programação. Dessa forma, este curso apresenta o conhecimento mínimo para o desenvolvimento de pequenos programas até grandes sistemas empresariais.



# Unidade I

Vamos pensar nas ações que praticamos em nossa rotina matinal: o despertador toca, tiramos a coberta, levantamos, procuramos o chinelo debaixo da cama, escovamos os dentes, e assim por diante. Vamos nos concentrar no ato de escovar os dentes. Mesmo sem pensar em nada, exceto por algumas variações, agimos pegando o tubo do dentífrico, abrindo a tampa, depois pegamos a escova de dente, damos uma molhadinha nas cerdas, colocamos um pouco da pasta de dentes nela, escovamos os dentes, abrimos a torneira da pia, enxaguamos a boca, limpamos a escova, fechamos a torneira, fechamos a tampa do tubo. Nesse momento, estamos com os dentes limpos e escovados, a escova limpa, o tubo tampado e a torneira fechada. Imaginemos que um dia acordemos mal-humorados e resolvamos mudar a ordem de algumas das ações. Pegamos a escova de dentes, abrimos a torneira, damos uma molhadinha nas cerdas, escovamos os dentes, pegamos o tubo do dentífrico, abrimos a tampa, colocamos um pouco da pasta na escova, fechamos a tampa, enxaguamos a boca, limpamos a escova, fechamos a torneira. No final, teremos a escova limpa, a torneira fechada, o tubo fechado e os dentes escovados, porém não estarão limpos. Será que é isso mesmo? Vamos estudar cada ação executada no segundo caso, passo a passo. Após pegar a escova de dentes, abrir a torneira, molhar as cerdas e escovar os dentes, teremos a torneira aberta, a escova suja ainda sem a pasta e os dentes escovados sem o creme dental. Depois, pegamos o tubo, abrimos a tampa, colocamos a pasta na escova e fechamos a tampa. Agora, temos o tubo novamente fechado, a escova com pasta e os dentes escovados sem o creme dental. Finalmente, enxaguamos a boca, retiramos o creme dental da escova e fechamos a torneira, chegando à conclusão anterior, com a escova limpa, a torneira fechada, o tubo fechado e os dentes escovados, porém não limpos.

A situação anterior mostra que cada coisa que fazemos em qualquer momento da vida obedece a uma sequência de ações, desde o simples ato de procurar o botão para desligar o despertador até o de apagar a lâmpada antes de dormir. As ações obedecem a uma sequência lógica, procurando chegar a um certo objetivo. Quando queremos flertar com uma pessoa, há um encadeamento, uma sequência de ações, de tomadas de decisões, escolha entre várias possibilidades, além do uso das experiências adquiridas. Existe o desenvolvimento de um algoritmo com o intuito de conquistar essa pessoa.

O computador não possui conhecimento próprio. Assim, se precisarmos de sua ajuda para atingir um objetivo, deveremos instruí-lo com um algoritmo escrito conforme uma lógica correta, para ele cumprir a tarefa que queremos. Vamos estudar como fazer o computador começar a executar aquilo que desejamos.

## **1 ALGORITMOS**

### **1.1 Conceitos básicos**

Pelo texto introdutório, podemos ter uma vaga ideia do que seja um algoritmo. Todas as ações no nosso dia a dia seguem um algoritmo, seja um ato intencional, seja um ato instintivo. Com o decorrer da vida, conforme vamos crescendo e aprendendo, novas instruções vão se incorporando ao nosso

repertório. A experiência humana é um repertório de algoritmos que foram se acumulando durante a vida. O algoritmo é um conjunto finito de instruções, de comandos, de ações que tem como objetivo a resolução de uma tarefa, ou a resolução de um problema.

Algumas outras definições sobre algoritmos:

"Algoritmo é uma sequência finita de instruções ou operações cuja execução, em tempo finito, resolve um problema computacional, qualquer que seja sua instância. [...]"

Algoritmo é uma sequência de passos que visam atingir um objetivo bem-definido" (FORBELLONE, 1993, p. 3).

"Algoritmo são regras formais para a obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas" (MANZANO, 1996, p. 6).

"Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma tarefa" (ASCENCIO; CAMPOS, 2003, p. 1).

Ação é um acontecimento que, a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e bem-definido. Portanto um algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de ações (FARRER et al., 1999, p. 14).

### 1.2 Programas

Os computadores executam as tarefas que são comandadas pelo sistema operacional. Mais especificamente, o *hardware* (equipamento, monitor, teclado, *mouse*, tela de toque, dentre vários dispositivos), para interagir com o mundo físico, necessita de uma camada intermediária, que faça a ponte para executar os *softwares* (os programas, as instruções) Essa ponte é o chamado sistema operacional.

Hoje conhecemos alguns sistemas, como Linux, Windows, Mac OS, Android, e, no passado, tivemos outros, como MS-DOS, DOS, CP/M. O sistema operacional, por sua vez, compreende as instruções-padrão que os programas mandam executar. Esses programas são uma "tradução" dos comandos escritos em linguagem computacional que resultaram dos algoritmos por nós pensados. Assim, nossos algoritmos, uma vez corretos, podem ser transcritos em linguagem computacional, ou linguagem de programação, como o C, o C++, o C#, o Delphi, o Pascal, o PHP, o COBOL e diversas outras já existentes ou que virão a existir. Além do algoritmo, os programas incluem os dados a serem utilizados no processamento.

### 1.3 Lógica de programação

Na feira, as peras-asiáticas custam R\$ 1,50 cada. Tenho R\$ 10,00. Quantas peras eu posso comprar?

Nesse momento, o cérebro busca informações aprendidas no Ensino Fundamental e começa a fazer cálculos, ou a imaginar como resolver esse problema. Algumas pessoas, mentalmente, começam a somar

e a contar no dedo: 1,50... 3,00... 4,50... 6,00... 7,50... 9,00... 10,50... Passou, então são seis. Outras já fazem a conta 10 dividido por 1,50, então 6 e sobra 1.

Esse processo, aplicado na programação de computadores, chama-se Lógica de Programação. Antes de montar um algoritmo, precisamos ter uma noção do que iremos fazer, ou, pelo menos, uma estrutura mental de como realizar a tarefa, ou de como resolver o problema proposto. Assim, é a técnica de encadear pensamentos para atingir um determinado objetivo.

O objetivo principal da Lógica de Programação é demonstrar técnicas para resolução de problemas e, conseqüentemente, para automatização de tarefas.

## 1.4 Representação lógica

Para desenvolver um programa, uma vez estruturado o raciocínio lógico, passamos a utilizar simbologias dos algoritmos para representarmos esse pensamento. Na linguagem computacional, temos três tipos de algoritmos mais utilizados: descrição narrativa, fluxograma e pseudocódigo ou português, que descreveremos a seguir.

## 1.5 Tipos de algoritmos

### 1.5.1 Descrição narrativa

Para fazer uma descrição narrativa é necessário entender o problema proposto e narrar a sua solução por meio da linguagem normalmente utilizada no dia a dia, ou seja, a língua portuguesa.

Exemplo: você está perdido em um lugar de uma cidade desconhecida e procura a rodoviária; encontra um policial e pede auxílio:

– Por favor, como eu chego até a rodoviária?

O policial responde usando a descrição narrativa:

– Siga em frente e, na quarta esquina, vire à direita; siga em frente e, de lá, você já vai vê-la.

### Quadro 1 – Vantagem e desvantagem de usar a descrição narrativa

Vantagem	Desvantagem
Não é necessário aprender nenhum conceito novo. Basta escrever da maneira como se fala	Essa linguagem dá margem para vários tipos de interpretação. Portanto, se você não for claro o suficiente, isso poderá dificultar a transição desse algoritmo para a programação



## Saiba mais

Assista ao filme:







UMA MENTE brilhante. Direção: Ron Howard. Produção: Brian Grazer e Ron Howard. EUA: Universal Pictures, 2001. 1 DVD (135 min).

Nesse filme, a personagem principal, o professor John Nash, utiliza algoritmos em diversas ocasiões. É interessante acompanhar o filme para compreender o processo de relacionar eventos cotidianos com as suas teorias. A tentativa de criar um algoritmo para os passos do pombo e, principalmente, como a conquista de um grupo de garotas por um grupo de rapazes leva à derrubada de uma teoria clássica.

## 1.5.2 Fluxograma

No fluxograma, os passos para a resolução de um problema são demonstrados por meio de um conjunto de símbolos-padrão (Quadro 2).

**Quadro 2 – Principais símbolos de um fluxograma**

Símbolo	Função
	Início e fim do algoritmo
	Indica cálculo e atribuições de valores
	Indica a entrada de dados
	Indica uma decisão com possibilidades de desvios
	Indica saída de dados
	Indica o fluxo de dados. Serve também para conectar os blocos ou símbolos existentes



Voltando ao exemplo do tópico anterior, se o guarda respondesse com um fluxograma, ele o faria conforme a Figura 1:

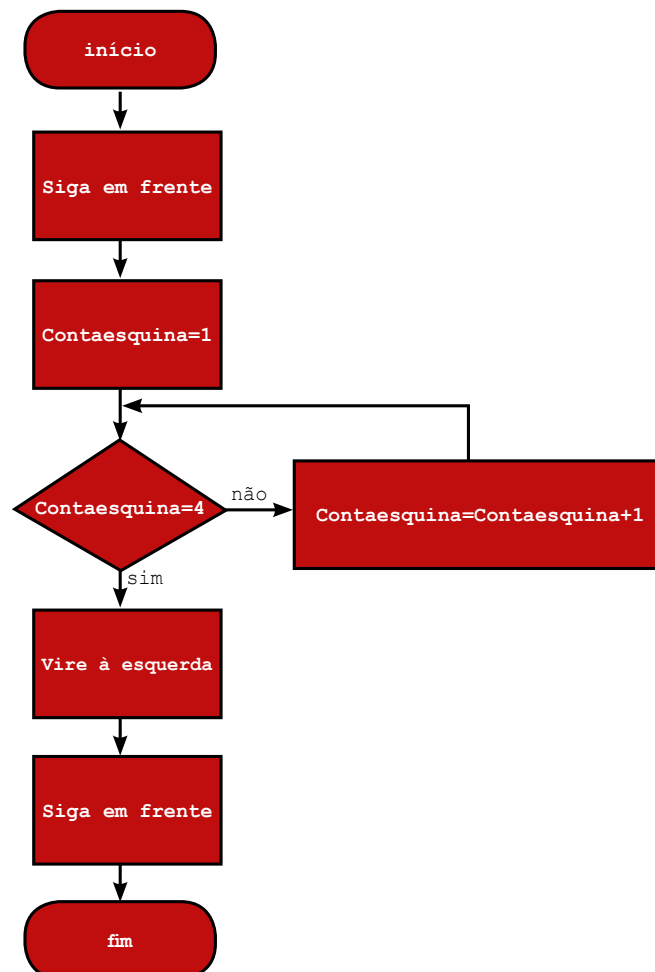


Figura 1 – Fluxograma para chegar à rodoviária

## Quadro 3 – Vantagem e desvantagens de usar o fluxograma

Vantagem	Desvantagens
O entendimento é bem mais simples	É necessário que se tenha conhecimento dos símbolos Modificações e edições difíceis

### 1.5.3 Pseudocódigo ou Portugol

Essa estrutura é escrita em português, muito próxima à linguagem natural, utilizando regras predefinidas para escrevê-la. Essas regras não são tão rígidas quanto as de uma linguagem de programação, portanto o foco do pseudocódigo é a lógica, e não a rigidez sintática. Utilizando a situação anterior, do pedido de auxílio ao guarda, um exemplo de pseudocódigo (ainda sem as regras bem-definidas, como veremos mais adiante) seria:

**Algoritmo** procura\_rodoviária

**Variáveis**

contaesquina: inteiro

**Início**

Siga("em frente")

contaesquina  $\leftarrow$  1

Enquanto contaesquina  $\neq$  4 faça

contaesquina  $\leftarrow$  contaesquina + 1

fim Enquanto

Vire("direita")

Siga("em frente")

**Fim.**

## Quadro 4 – Vantagens e desvantagem de usar o pseudocódigo

Vantagens	Desvantagem
O entendimento é bem mais simples	É necessário que se tenha conhecimento de pseudocódigo, que veremos mais adiante
Não ocorrem erros de compilação	

## 1.6 Regras para construção do algoritmo

Para construir um algoritmo, existem algumas regras práticas, que podem ser aperfeiçoadas conforme a experiência do programador. Deve-se ter em mente que um mau desenvolvedor de algoritmos dificilmente será um bom programador. Portanto, antes de ficar diante do computador para desenvolver um programa, é preciso definir as metas.

Assim, diante de um problema, devem-se seguir alguns passos.

- Leia todo o problema: faça uma leitura de todo o problema até o final, para formar a primeira impressão. Releia o problema e faça anotações sobre os pontos principais.
- Entendimento: o problema foi bem-entendido? Questione, se preciso, o autor da especificação sobre suas dúvidas. Releia o problema quantas vezes for preciso para entendê-lo.
- Resultados esperados: extraia do problema todas as suas **saídas**.
- Informações disponíveis: extraia do problema todas as suas **entradas**.
- Como funciona: identifique qual é o processamento principal.
- Outras informações: verifique se será necessário algum valor intermediário que auxilie na transformação das entradas em saídas. Essa etapa pode parecer obscura no início, mas, com certeza, no desenrolar do algoritmo, esses valores aparecerão naturalmente.

- Verificação: teste cada passo do algoritmo, com todos os seus caminhos, para verificar se o processamento está gerando os resultados esperados. Crie valores de teste para submeter ao algoritmo.
- Normatização: reveja o algoritmo, checando as boas normas de criação.

Além disso, existem algumas recomendações para escrever um algoritmo legível: basta ser simples e objetivo. Para isso, descreva a sequência de instruções conforme as indicações que seguem:

- seja objetivo, usando apenas um verbo por frase;
- o algoritmo deve ser simples o suficiente para qualquer pessoa que não trabalhe com informática entender;
- não use frases complexas e confusas, o importante é ser simples e claro no seu objetivo;
- não utilize palavras que tenham duplas interpretações ou deixem margem a dúvidas.

Por exemplo: desde que os celulares e os *smartphones* estão no mercado, os telefones públicos estão quase esquecidos, e nem sempre as pessoas sabem mais como utilizá-los. Existem sempre as emergências, principalmente, quando você percebe que se esqueceu de carregar o celular, está longe de casa e precisa fazer uma ligação urgente. Pensando nessa situação, faça um algoritmo não computacional cujo objetivo é usar um telefone público.

- Tirar o fone do gancho.
- Ouvir o sinal de linha.
- Introduzir o cartão.
- Teclar o número desejado.
- Se der o sinal de chamar:
  - conversar;
  - desligar;
  - retirar o cartão.
- Se não der o sinal de chamar:
  - colocar o fone no gancho;
  - repetir.



### Observação

Pense em algumas situações do dia a dia nas quais esse raciocínio pode ser aplicado, como dar partida no carro, tomar café, ficar com dor de barriga no meio da rua etc.



### Saiba mais

Existem programas que ajudam a organizar o pensamento e que se chamam mapas mentais. Existem vários programas desse tipo, como Freemind e Open Mind, que podem ser baixados e instalados gratuitamente.

Alguns *sites* oferecem programas para o uso *on-line*, como os seguintes:

<<http://www.mindmeister.com/pt>>.

<<http://www.mindmup.com/>>.

## 1.7 Introdução à programação

Uma vez tendo a noção geral de um algoritmo, veremos agora as regras para a montagem de programas.

É importante frisar que iremos aprender o pseudocódigo, mas a nomenclatura, os comandos e a estrutura sintática podem mudar, conforme a literatura. O importante não é a rigidez das regras formais, mas a ideia contida, ou seja, o que um comando faz, para que serve e como o faz. O pseudocódigo serve para desenvolver o raciocínio que será utilizado para criar um programa, e não para nos preocuparmos se estamos escrevendo dentro de normas rígidas como as exigidas por uma linguagem de programação.

Inicialmente, veremos os recursos: com que tipo de informação o computador trabalha, como fazemos a entrada e a saída de dados e quais operações é possível realizar.

### 1.7.1 Tipos de informações, entrada e saída

Podemos classificar os tipos de informações a serem processadas pelo computador, de modo geral, em dados e instruções. Os dados são as informações a serem processadas, e as instruções são os comandos que orientam o processamento feito por um computador.

Nesta primeira unidade focalizaremos, principalmente, os dados.

## 1.7.2 Dados

Desde o Ensino Fundamental, trabalhamos com unidades de medida (kg, km, L, m etc.) e com tipos de número (natural, inteiro, real). Os algoritmos e as linguagens de programação trabalham com dados bem-definidos. Esses dados são classificados em tipos.

## 1.7.3 Tipos de dados

Conforme o tipo de dado, o computador trabalha de uma forma diferente no tratamento destes durante o processamento.

### 1.7.3.1 Variáveis, constantes e operadores

#### 1.7.3.1.1 Tipos primitivos de dados

Em princípio, os computadores trabalham com quatro tipos de dados. Partindo destes, as linguagens de programação derivam outros mais específicos, ou com uma precisão maior. O tratamento lógico entre esses tipos derivados é o mesmo. Assim, em Algoritmo trataremos apenas dos fundamentais: Números Inteiros; Números Reais; Caracteres e Cadeias; e Lógicos, definidos conforme a seguir.

- Números Inteiros
  - Toda e qualquer informação numérica que pertença ao Conjunto dos Números Inteiros (negativa, nula ou positiva). Exemplos: 1; 2; 44; 100; 0; -15; -99.
  - Usos mais comuns: contador de eventos, idade, ano, quantidade.
- Números Reais
  - Toda e qualquer informação numérica que pertença ao Conjunto dos Números Reais (negativa, nula ou positiva). Exemplos: 0,12; 3,14159; 0,000; -1,23.
  - Usos mais comuns: valores em dinheiro, como salário, preço total, preço unitário, peso, média.
- Caracteres (*char*) e cadeias (*string*)
  - Toda e qualquer informação composta por um conjunto de caracteres alfanuméricos:
    - numéricos ('0'..'9');
    - alfabéticos ('A'..'Z'; 'a'..'z');
    - especiais (por exemplo, '#', '?', '!', '@').

- São identificados por estarem entre apóstrofos (").
- Uma cadeia é um conjunto de caracteres; assim, 'a' é um caractere, e 'avião' é uma cadeia.
- Usos mais comuns: nome, endereço, descrição.
- Lógicos
  - Toda e qualquer informação que se enquadre em apenas duas situações:
    - Valores booleanos V (verdadeiro) ou F (falso).
  - Usos comuns: controle de fluxo de programa, flags.

Exemplo: identificar o tipo.

**Tabela 1 – Identificação dos tipos**

3,14159	Real
123	Inteiro
'B52'	Cadeia
V	Lógico
315,0	Real (pois o número pode assumir valores depois da vírgula)
-12	Inteiro
'F'	Caractere (note os apóstrofos)
2014	Inteiro
'Universidade Paulista'	Cadeira

### 1.7.3.1.2 Constantes

Quando um dado não sofre nenhuma variação durante a execução do programa, é chamado constante. Seu valor deverá ser o mesmo do início ao fim da execução do programa, assim como é o mesmo para execuções diferentes.

As constantes podem ser de qualquer tipo de dado e, normalmente, são definidas em parte específica de um programa.

Exemplo:

#definido PI = 3,141617 (**nunca muda**).

### 1.7.3.1.3 Variáveis

As variáveis são espaços reservados na memória principal do computador em que os dados são armazenados para reutilização posterior.

Mesmo nas calculadoras mais simples, há um botãozinho M+ que pouca gente usa. Ele serve para guardar (ou acumular) temporariamente um valor, que recuperamos apertando a tecla MR. Essa tecla é especialmente útil quando temos operações mais complexas ou fórmulas com parênteses. No computador, podemos ter uma infinidade desses botões e, para podermos reconhecer o conteúdo de cada um deles, podemos batizar com um nome. Cada um desses botões é uma variável.

### 1.7.3.1.4 Variáveis – nomenclaturas

Para batizar uma variável, é recomendado dar um nome que lembre a informação armazenada.

Os nomes das variáveis são atribuídos pelo usuário e obedecem a certas regras:

- o primeiro caractere deve ser uma letra;
- os nomes podem ser formados por letras, dígitos e pelo caractere sublinhado (*underline*);
- podem ter qualquer comprimento.

O computador, para poder trabalhar com as informações, precisa saber a localização do dado na memória. Portanto, cada variável criada possui um endereço em uma posição de memória, ou seja, um número representado por meio da notação hexadecimal que indica o endereço de memória em que está armazenada a informação.

**Tabela 2 – Modelo de tabela de memória**

Código	7000: B128	'ABC'
Letra	2000: 0007	'D'
Saldo	8900: 138D	12395

Os endereços lógicos são locais que representam o endereço físico da memória *RAM*, que armazenam momentaneamente algum tipo de informação, ou seja, não ficam o tempo todo com o mesmo valor. Na Tabela 2, os dados podem ser diversos; a caixa com o endereço 7000: B128 recebeu o texto 'ABC', mas, em outro momento, pode receber uma cadeia de caracteres qualquer. Assim, pode-se considerar que o conteúdo dos endereços lógicos pode sofrer alterações a qualquer momento. Daí vem o nome utilizado para essa representação: variáveis, pois variam durante cada execução do programa.

Ao definirmos uma variável como caractere ou string (ou cadeia de caracteres), alocaremos até 255 bytes, um byte para cada caractere da string (cadeia). Com relação a cadeias, podemos utilizar o valor entre aspas ou entre apóstrofes.

Exemplo: crie nomes para as variáveis e determine os possíveis tipos de dados:

- cargo do funcionário;

- salário;
- valor do aluguel;
- telefone;
- endereço;
- nome do cantor;
- idade;
- RG;
- CNPJ;
- título do *CD*.

Poderia ser:

**Quadro 5 – Exemplo de nomes e tipos para variáveis**

Cargo do funcionário	CFunc ou Cargo_Func...	Cadeia
Salário	Salário...	Real
Valor do aluguel	VAluguel ou Valor_Aluguel...	Real
Telefone	Tel ou fone ou Telefone...	Cadeia ou Inteiro
Endereço	End ou Endereço...	Cadeia
Nome do cantor	Cantor ou N_Cantor ...	Cadeia
Idade	Idade	Inteiro
RG	RG	Cadeia ou Inteiro
CNPJ	CNPJ	Cadeia ou Inteiro
Título do CD	Título ou Tit_Cd...	Cadeia

Exemplo: dados alguns nomes de variáveis, quais são válidos? Justifique os que não são.

**Quadro 6 – Nomes de variáveis válidos e inválidos**

<b>dia/mês</b>	Caractere '/' inválido
<b>CEP</b>	Válido
<b>1parcela</b>	Inicia com número
<b>B52</b>	Válido
<b>P{O}</b>	Caracteres '{ }' inválidos
<b>O&amp;O</b>	Caractere '&' inválido
<b>AB*C</b>	Caractere '*' inválido
<b>ASDRUBAL</b>	Válido



UYT	Válido
KM/L	Caractere '/' inválido
#55	Caractere '#' inválido
"ALUNO"	Nome entre aspas
AH!	Caractere '!' inválido
U2	X
VALOR TOTAL	Espaço entre os nomes

## 1.7.3.1.5 Variáveis – declaração

Criar uma variável significa reservar um espaço na memória do computador dando-lhe um nome e determinar qual o tipo de dado que essa memória armazenará.

O processo de criação de uma variável é chamado também de **declaração de variável**. A declaração de uma variável no pseudocódigo tem um trecho (sessão) específico identificado como var ou variável. Assim, a declaração é feita conforme segue.

Var (ou variável)

```
<nome da variavel1>, ... <nome da variaveln>: <tipo>  
<nome da variaveln1>, ... <nome da variavelnn>: <tipo>
```

Exemplo:

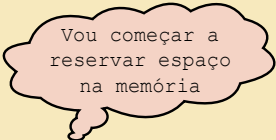
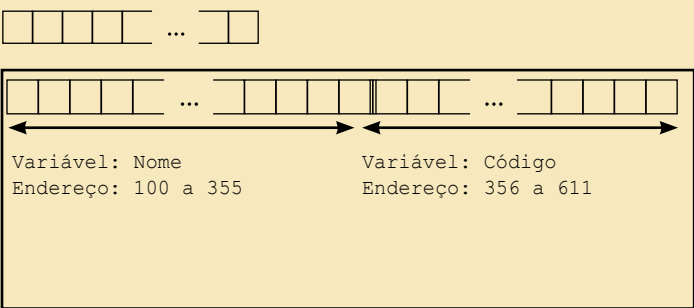
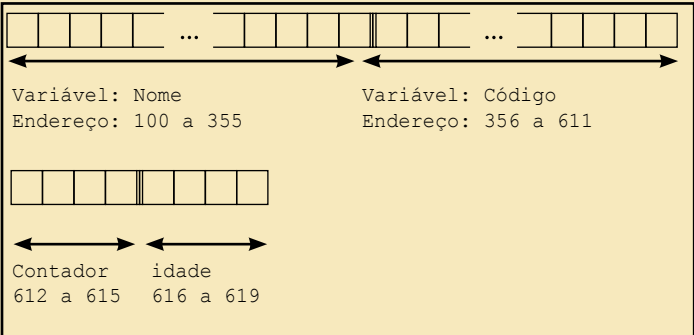
**Var**

```
Nome, Código: Cadeia  
Contador, idade: inteiro
```

Agora, vamos entender, passo a passo, como o computador entende e faz as operações do exemplo.

Ao encontrar a instrução **var**, o computador entende que se trata da seção de declaração de variáveis e prepara a memória para criar os espaços para as variáveis. Na linha seguinte, encontra dois nomes que batizarão espaços de memória, Nome e Código. Ao final da linha, há a indicação de que o espaço a ser reservado será do tipo Cadeia. Assim, para cada uma das variáveis, serão reservados 256 bytes, e o computador armazenará, por critérios técnicos, os endereços de memória no local em que foram criados; no caso, do byte do endereço 100 até o byte do endereço 355, para Nome, e do endereço 356 até o 611, para Código. A seguir, encontra dois novos nomes, Contador e Idade, que são do tipo Inteiro. O computador reserva 4 bytes (inteiros, no nosso caso, mas que podem mudar de programa para programa) para cada uma das variáveis. Assim, reserva os bytes do endereço 612 até o 615, para Contador, e de 616 até 619, para Idade.

**Quadro 7 – Passo a passo da declaração de variáveis**

Linha de execução	Computador
Var ◀ Nome, Código: Cadeia Contador, idade:inteiro	
Var Nome, Código: Cadeia◀ Contador, idade:inteiro	<p>Memória</p>  <p>Variável: Nome                      Variável: Código            Endereço: 100 a 355              Endereço: 356 a 611</p>
Var Nome, Código: Cadeia Contador, idade:inteiro◀	<p>Memória</p>  <p>Variável: Nome                      Variável: Código            Endereço: 100 a 355              Endereço: 356 a 611</p> <p>Contador      idade            612 a 615    616 a 619</p>

## 1.7.3.2 Operadores

O computador faz uma série de operações. Quando falamos em operações, não são apenas as matemáticas – soma, subtração, entre outras –, mas sim aquelas feitas para operar qualquer manipulação de dados. Assim, operadores são elementos que atuam sobre operandos para produzirem um resultado. Ao tomarmos a expressão aritmética  $2 + 3$ , temos dois operandos, os números 2 e 3, que são relacionados pelo operador soma (+), fazendo a operação de adição. Os operadores podem ser binários, quando atuam sobre dois operadores (soma, subtração etc.), ou unários, quando atuam somente sobre um operador (como o sinal negativo em um número).

### 1.7.3.2.1 Operador de atribuição ( $\rightarrow$ ou $\leftarrow$ )

Vimos que as variáveis armazenam valores para o uso posterior no transcorrer do programa, mas como colocar um valor em uma variável?

Para armazenar um valor em uma variável utilizamos o operador de atribuição.

O operador de atribuição é representado por uma seta apontando para a esquerda ( $\leftarrow$ ), ou, por questões gráficas, por um símbolo de "menor" e um "traço" ( $\leftarrow$ ).

Forma:

`<variável> ← valor`

Exemplo:

**Var**

Nome:**Cadeia**

Idade:**inteiro**

**Inicio**

Nome←"Aderbal Schmydt"

Idade←19

**Fim**

Estudando detalhadamente cada passo do computador:

**Var** (O computador entende que entrará em processo de reserva de memória.)

**Nome:Cadeia** (Reserva o espaço chamado Nome para o tipo *String*.)

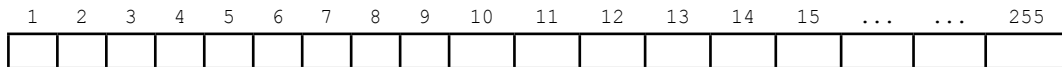


Figura 2 – Reservando a variável "nome"

**Idade:inteiro** (Reserva o espaço chamado Idade para o tipo Inteiro.)

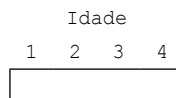


Figura 3 – Reservando a variável "idade"

**Inicio** (Início do processamento.)

**Nome←"Aderbal Schmydt"** (O Valor "Aderbal Schmydt" será armazenado no espaço chamado Nome.)

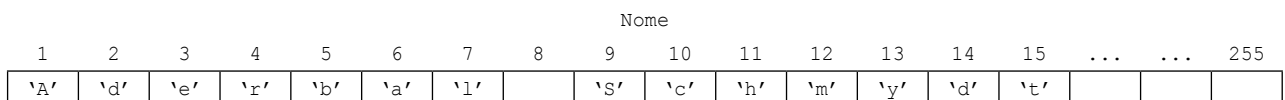


Figura 4 – Atribuindo valor

**Idade <- 19** (O Valor 19 será armazenado no espaço chamado Idade.)

Idade				
1	2	3	4	
				19

Figura 5 – Atribuindo valor numérico

**Fim** (Fim do processamento.)



## Lembrete

Você somente conseguirá atribuir um valor a uma variável se ela estiver criada na seção de declaração.

### 1.7.3.2.2 Operadores aritméticos

Como fazemos contas com os algoritmos? Da mesma forma que na matemática, utilizamos os operadores aritméticos. Como são aritméticos, as operações são feitas entre números, resultando em um outro número.

#### Quadro 8 – Principais operadores aritméticos

Operador	Referência do operador
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
DIV	Quociente
MOD	Resto da divisão
**	Exponenciação

As operações de soma, subtração, multiplicação, divisão e exponenciação são as habituais da matemática. A computação oferece duas outras que são de extrema utilidade: o DIV e o MOD.

A divisão, o MOD e o DIV envolvem a operação de divisão, mas cada uma dessas operações pode dar resultados diferentes. Vamos tomar a divisão de 9 por 2.

A divisão  $9/2$  resulta em 4,5, conforme segue:

$$\begin{array}{r}
 9 \overline{) 2} \\
 \underline{-8} \phantom{0} \\
 10 \\
 \underline{-10} \\
 0
 \end{array}
 \quad \rightarrow \quad 9/2$$

Para entender o MOD e o DIV, é importante ter em mente que as operações envolvem apenas números inteiros:

$$\begin{array}{r} 9 \overline{) 2} \\ -8 \quad 4 \\ \hline 1 \end{array}$$

Diagram illustrating the division of 9 by 2. The quotient is 4 (labeled `9 div 2`) and the remainder is 1 (labeled `9 mod 2`).

Resumindo:

9 / 2 resulta em 4,5  
9 div 2 resulta em 4  
9 mod 2 resulta em 1

Exemplos:

- Para verificar se um número é par, é bem simples: por definição, os pares são divisíveis por 2, ou seja, o resto da divisão tem de ser zero. Portanto, ao fazer a operação:

Numero mod 2

Se o resultado for zero, o conteúdo da variável número será par.

- Em muitos jogos, é necessário delimitar uma faixa de número. Num jogo de dados, os resultados sempre estão entre 1 e 6. A melhor forma de conseguir é utilizando o princípio de que o resto de uma divisão sempre estará entre 0 e o valor do divisor menos 1; assim, o resto da divisão de qualquer número inteiro por 6 resultará em um número entre 0 e 5. Como o resultado tem de estar entre 1 e 6 e o resto está entre 0 e 5, basta somar 1.

Numero mod 6 + 1

Qualquer que seja o conteúdo da variável número, o resultado sempre será 1, 2, 3, 4, 5 ou 6.

- Para separar os algarismos de um número de dois dígitos, como usamos um sistema de numeração decimal, base dez, ao utilizarmos o quociente da divisão inteira teremos o dígito das dezenas, e o resto, o dígito das unidades.

Numero div 10

Numero mod 10

Caso a variável número contenha um valor entre 0 e 99 (dois dígitos), ao fazer a operação DIV, ela resultará num valor entre 0 e 9, e o MOD a unidade que resta dos múltiplos de dez.

## 1.7.3.2.3 Operadores relacionais

Quando queremos comparar duas informações, são usados os operadores relacionais. Como são comparações, os resultados possíveis serão o verdadeiro ou o falso. Mas o que são essas comparações?

Ocorrem quando um mesmo tipo de informação pode ser avaliado quanto à sua grandeza, ou à sua igualdade. Assim, um número pode ser maior ou menor que outro, ou, ainda, igual a esse número.

**Quadro 9 – Operadores relacionais**

Operador	Referência do operador
=	Igual
<>	Diferente
>=	Maior ou igual
<=	Menor ou igual
>	Maior
<	Menor

Exemplo: avaliar o resultado das operações relacionais.

**Tabela 3 – Exemplo de resultado dos operadores relacionais**

9 > 8	Verdadeiro	
15 <=15	Verdadeiro	A operação é menor ou igual; portanto, nesse caso, a igualdade valida a comparação
15 > 15	Falso	
15 <> 15	Falso	Atenção: os números são iguais, e a comparação é diferente
15 = 15	Verdadeiro	
17 >= 20	Falso	
"rata"< "rato"	Verdadeiro	No caso de cadeias, compara-se dígito a dígito, até encontrar um caractere diferente, com o qual será feita a comparação

## 1.7.3.2.4 Operadores lógicos

Para fazer cálculos com os valores verdadeiro e falso, são utilizados os operadores lógicos. Os mais comuns são os operadores 'e', 'ou' e 'não'; e, menos comumente, o 'xor'. Cada um dos operadores obedece a uma lei de formação.

- Operador 'e': o resultado do operador **e** somente será verdadeiro quando ambos os operandos forem verdadeiros. Assim, a chamada tabela-verdade para o **e** será como segue.

**Quadro 10 – Tabela-verdade da operação e**

Operando 1		Operando 2		Resultado
VERDADEIRO	E	VERDADEIRO	→	VERDADEIRO
VERDADEIRO	E	FALSO	→	FALSO
FALSO	E	VERDADEIRO	→	FALSO
FALSO	E	FALSO	→	FALSO

- Operador 'ou': o resultado do operador **ou** somente será verdadeiro quando pelo menos um dos operandos for verdadeiro. Assim, a chamada tabela-verdade para o **ou** será como segue.

**Quadro 11 – Tabela-verdade da operação ou**

Operando 1		Operando 2		Resultado
VERDADEIRO	OU	VERDADEIRO	→	VERDADEIRO
VERDADEIRO	OU	FALSO	→	VERDADEIRO
FALSO	OU	VERDADEIRO	→	VERDADEIRO
FALSO	OU	FALSO	→	FALSO

- Operador 'não': o operador **não** é unário, ou seja, tem apenas um operando e inverte o valor do resultado desse operando. Assim, a chamada tabela-verdade para o **não** será como segue.

**Quadro 12 – Tabela-verdade da operação não**

	Operando		Resultado
NÃO	VERDADEIRO	→	FALSO
NÃO	FALSO	→	VERDADEIRO

- Operador XOR: o operador XOR é menos utilizado; seu resultado somente será verdadeiro quando os dois operandos forem diferentes. O **XOR** é conhecido também como **ou exclusivo**. Assim, a chamada tabela-verdade para o **XOR** será como segue.

**Quadro 13 – Tabela-verdade da operação XOR**

Operando 1		Operando 2		Resultado
VERDADEIRO	XOR	VERDADEIRO	→	FALSO
VERDADEIRO	XOR	FALSO	→	VERDADEIRO
FALSO	XOR	VERDADEIRO	→	VERDADEIRO
FALSO	XOR	FALSO	→	FALSO

## Exemplo de aplicação

Um caso típico do uso do XOR é para analisar a diferença de valores binários, por exemplo:

a) 0101 (5 decimal).

b) 0100 (4 decimal).

Ao aplicamos o operador XOR nos dois valores, teremos o seguinte resultado:

```
0101
0100
----
0001
```

Assim, vemos facilmente que a diferença está no quarto bit.

### 1.7.3.2.5 Prioridade na avaliação de expressões

Encontre o resultado de:

$$2 + 3 \times 5$$

A resposta não é 25. O correto é 17. Voltando à aritmética do Ensino Fundamental, vamos nos lembrar de que a multiplicação é efetuada antes da soma.

O mesmo acontece com a programação: existe uma sequência para realizar as operações. Portanto, deve-se obedecer à seguinte ordem:

- parênteses e funções (resolvidos da esquerda para a direita);
- multiplicação (\*), divisão (/ e div) e resto (Mod) (resolvidos da esquerda para a direita);
- soma e subtração;
- operadores relacionais: >, <, >=, <=, =, <>;
- operador lógico **não**;
- operador lógico **e**;
- operador lógico **ou**.

Exemplo: resolver a seguinte expressão.

```
X=((1+ 2 * 3) <= (4 + 6 / 2)) e (não((6+1)<>(10-3)))
X=((1+ 6 ) <= (4 + 3 )) e (não( 7 <> 7))
X=( 7 <= 7 ) e (não( Falso ))
X=( Verdadeiro ) e ( Verdadeiro )
X=Verdadeiro
```

Exemplo: sabendo que A = 5, B = 4, C = 3 e D = 6, avaliar as expressões a seguir.

**Tabela 4 – Exemplos de expressões**

(A > C) e (C<=D)	Verdadeiro
(A+B) > 10 ou (A+B) = (C+D)	Verdadeiro
(A>=C) e não(D>=C)	Falso





## Lembrete

O operador aritmético atua entre dois números, resultando em outro número.

O operador relacional atua entre duas grandezas, resultando num valor lógico.

O operador lógico atua apenas em valores lógicos, resultando em um valor lógico.

### 1.7.3.2.6 Linearização

O computador não reconhece as fórmulas matemáticas armadas; por exemplo, para calcular a área de um triângulo, aprendemos que:

$$\text{Área} = \frac{\text{base} \times \text{altura}}{2}$$

Para que o computador entenda a fórmula, precisamos, manualmente, fazer o processo de linearização. Esse processo consiste em remontar a fórmula utilizando parênteses e escrevendo-a em apenas uma linha. Assim, a fórmula da área do triângulo fica:

```
Área <- (base * altura) / 2
```

Exemplo: no manual do aluno, consta que o cálculo da média final é:

$$MF = \frac{4 \times NP1 + 4 \times NP2 + 2 \times PIM}{10}$$

Linearizando, fica:

```
NF <- (4 * NP1 + 4 * NP2 + 2 * PIM) / 10
```

## 1.8 Programação sequencial

Agora que sabemos as operações, vamos começar a programar. Como você deve ter notado, os eventos que vimos com relação à execução de um algoritmo são feitos linha a linha; temos uma instrução e, uma vez realizada, passamos para a linha seguinte. Na programação ocorre o mesmo. O computador lê cada linha e executa o comando escrito pelo programador; feito isso, passa para a linha seguinte, e assim sucessivamente, até o final do programa.

### 1.8.1 Estrutura sequencial

Essa maneira de programar, em que as instruções são executadas uma após a outra, na ordem em que foram escritas, chama-se **programação sequencial**, ou podemos dizer que o programa foi escrito numa **estrutura sequencial**. Portanto, a estrutura sequencial é um conjunto de comandos que serão executados numa sequência linear de cima para baixo, linha a linha. Pode aparecer em qualquer estrutura de controle, agrupada ou não por blocos.

```
comando 1;  
comando 2;  
início comando 3 ...  
    meio comando 3 ...  
fim comando 3 ; ...  
comando n;  
.  
.  
.
```

Quando é feito o uso de fluxogramas, a estrutura sequencial é caracterizada por um único fluxo de execução (um único caminho orientado) no diagrama (Figura 2). Em pseudocódigos, a estrutura sequencial caracteriza-se por um conjunto de comandos dispostos ordenadamente.

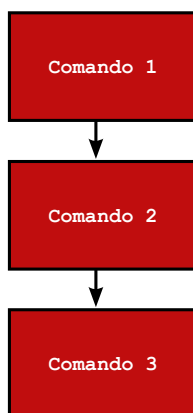


Figura 6 – Operações sequenciais

### 1.9 Estrutura de um pseudocódigo

Não existe uma estrutura rígida de escrita de um pseudocódigo; conforme a literatura, tal estrutura pode mudar, mas o importante é que transmita a ideia, com vistas à posterior passagem para a linguagem computacional. Exigir regras fixas e pouco flexíveis é tornar o pseudocódigo uma linguagem de programação. Isso não quer dizer que um pseudocódigo possa ser escrito de qualquer forma. Um mínimo de formalidade é necessário para um bom entendimento.

Com base nas formas mais comuns encontradas na literatura, um pseudocódigo tem três partes:

- cabeçalho com o nome do programa;
- uma seção para a declaração das constantes e das variáveis;
- o corpo do programa, em que é feita a programação sequencial.

Segue um algoritmo completo: em azul-claro, temos o cabeçalho; em bege, a declaração das variáveis; e, em verde, o corpo do programa.

```
algoritmo "Calc_Preco_final"
var
    parcelas,i:inteiro
    preco,taxa,juros, prestacao, precofinal,parcelamensal:real
inicio
    escreva("Entre com o preço do produto")
    leia(preco)
    escreva("Entre com a quantidade de parcelas")
    leia(parcelas)
    se parcelas > 1 entao
        venda parcelada
        escreva("Entre com a taxa de juro % mensal")
        leia(taxa)
        prestacao<-preco/parcelas
        inicializando o valor total
        precofinal<-0
        acumulando as prestações
        para i de 1 ate parcelas faca
            parcelamensal<-prestacao + (prestacao * taxa / 100)
            escreva("prestacao ",i," valor ", parcelamensal)
            precofinal<-precofinal + parcelamensal
        fimpara
    senao
        venda a vista
        precofinal<-preco
    fimse
    escreva("Preço final",precofinal)
finalgoritmo
```

Temos também algumas regras de etiqueta, ou boa educação. Essas regras não estão diretamente ligadas ao processamento em si, mas facilitam a compreensão do programa, tanto pelo autor quanto pelas pessoas que, posteriormente, tentarão entender o programa. Usando o mesmo código, veremos dois conceitos importantíssimos: o comentário e a indentação (são válidos tanto o termo **indentação**, um anglicismo vindo de *to indent*, quanto o oficialmente adotado **endentação**).

- Comentário são as linhas marcadas em azul (a seguir). Serve para lembrar ou explicar o que está acontecendo em um determinado local do código. Deve ser feito com critério, para ser bem objetivo, e conter a ideia daquele trecho, e não especificamente dizer o que uma linha está executando.

```

algoritmo "Calc_Preco_final"
var
    parcelas,i:inteiro
    preco,taxa,juros, prestacao, precofinal,parcelamensal:real
inicio
    escreva("Entre com o preço do produto")
    leia(preco)
    escreva("Entre com a quantidade de parcelas")
    leia(parcelas)
    se parcelas > 1 entao
//      venda parcelada
        escreva("Entre com a taxa de juro % mensal")
        leia(taxa)
        prestacao<-preco/parcelas
//      inicializando o valor total
        precofinal<-0
//      acumulando as prestações
        para i de 1 ate parcelas faca
            parcelamensal<-prestacao + (prestacao * taxa / 100)
            escreva("prestacao ",i," valor ", parcelamensal)
            precofinal<-precofinal + parcelamensal
        fimpara
    senao
//      venda a vista
        precofinal<-preco
    fimse
    escreva("Preço final",precofinal)
fimalgoritmo

```

- A indentação é útil para separar blocos de programação, afastando e realinhando as linhas da margem esquerda. A indentação na confecção do pseudocódigo é fundamental, e, muitas vezes, os códigos são considerados inválidos caso esta não seja feita. No código a seguir, as linhas verticais são os blocos, e as setas horizontais, as indentações.

```

algoritmo "Calc_Preco_final"
var
    parcelas,i:inteiro
    preco,taxa,juros, prestacao, precofinal,parcelamensal:real
inicio
    escreva("Entre com o preço do produto")
    leia(preco)
    escreva("Entre com a quantidade de parcelas")
    leia(parcelas)
    se parcelas > 1 entao
//      venda parcelada
        escreva("Entre com a taxa de juro % mensal")
        leia(taxa)
        prestacao<-preco/parcelas
//      inicializando o valor total
        precofinal<-0
//      acumulando as prestações
        para i de 1 ate parcelas faca
            parcelamensal<-prestacao + (prestacao * taxa / 100)
            escreva("prestacao ",i," valor ", parcelamensal)
            precofinal<-precofinal + parcelamensal
        fimpara
    senao
//      venda a vista
        precofinal<-preco
    fimse
    escreva("Preço final",precofinal)
fimalgoritmo

```

Ao tirarmos a indentação, o mesmo código fica da seguinte forma:

```
algoritmo "Calc_Preco_final"
var
parcelas,i:inteiro
preco,taxa,juros, prestacao, precofinal,parcelamensal:real
inicio
escreva("Entre com o preço do produto")
leia(preco)
escreva("Entre com a quantidade de parcelas")
leia(parcelas)
se parcelas > 1 entao
//venda parcelada
escreva("Entre com a taxa de juro % mensal")
leia(taxa)
prestacao<-preco/parcelas
//inicializando o valor total
precofinal<-0
//acumulando as prestações
para i de 1 ate parcelas faca
parcelamensal<-prestacao + (prestacao * taxa / 100)
escreva("prestacao ",i," valor ", parcelamensal)
precofinal<-precofinal + parcelamensal
fimpara
senao
//          venda a vista
precofinal<-preco
fimse
escreva("Preço final",precofinal)
finalgoritmo
```

Notam-se claramente o desconforto e a perda total das diversas seções do código, bem como do início e do fim dos blocos de execução.

## 1.10 Entrada e saída de informações

Nos primeiros passos da programação vistos até agora, os programas efetuaram operações, mas não interagiram com o ser humano, o usuário.

### 1.10.1 Instrução primitiva de saída de dados

Para podermos ver no monitor, ou em uma impressora (os chamados dispositivos de saída), precisamos ter uma instrução específica. Esta é chamada de instrução primitiva de saída de dados. Primitiva, pois é uma instrução nativa, que não requer programação e já vem pronta para ser utilizada. Essa instrução é a

maneira que temos para mostrar (colocar no dispositivo de saída) as informações contidas na memória dos computadores, para que o usuário possa visualizá-las. A sintaxe é:

```
escreva(<lista_de_variáveis>)  
escreva("Texto")  
Ou  
escreva("Texto", <variáveis>, "Texto", <variáveis>...)
```

Uma <lista\_de\_variáveis> é um conjunto de nomes de variáveis separados por vírgulas:

Exemplo:

```
escreva(i, j, total)  
escreva("Entre com a taxa de juro % mensal")  
escreva("prestacao ", i, " valor ", parcelamensal))
```



### Observação

Muitas obras utilizam as instruções **escrever** ou **imprimir**, em vez de **escreva**. Todas estão corretas e podem ser utilizadas no curso. O importante é a compreensão do comando que está sendo utilizado. Essa é a grande vantagem de utilizar o português estruturado: ele não fica limitado a uma regra restrita de linguagem de programação.

Escreva (escrever), por ser considerada uma palavra reservada, não mais poderá ser utilizada como nome de variável. Assim, toda vez que ela for encontrada dentro dos pseudocódigos, será identificada como um comando de saída de dados.

No fluxograma, uma instrução de saída de dados é representada como na Figura 7.

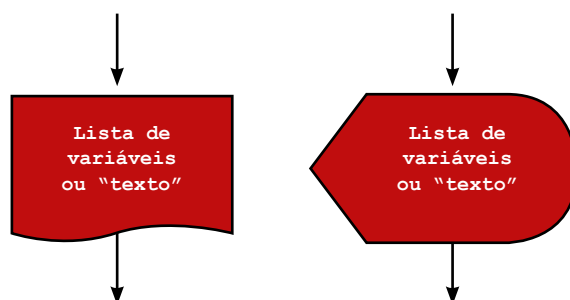


Figura 7 – Símbolo de saída

No fluxograma, não é necessário o uso do comando **escreva**, uma vez que a forma geométrica da figura determina a ação.

Exemplo: o algoritmo a seguir mostra o uso da instrução de saída. O programa faz o cálculo do preço total de cinco produtos que custam cinco reais cada, apresentando na tela o resultado.

```
algoritmo "ExemploSaida"  
var  
PR_UNIT, PR_TOT: Real  
QTD: Inteiro  
inicio  
    PR_UNIT <- 5.0  
    QTD <- 10  
    PR_TOT <- PR_UNIT * QTD  
    escreva (PR_TOT)  
finalgoritmo
```

Esse mesmo programa, escrito em forma de fluxograma (Figura 8), fica:

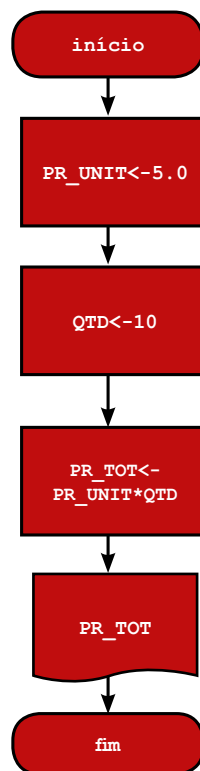


Figura 8 – Algoritmo do exemplo de saída

## 1.10.2 Instrução primitiva de entrada de dados

No algoritmo do exemplo anterior, a interação com o computador ainda não está completa. O programa faz o cálculo e mostra o resultado na tela. Os valores do preço unitário (5.0) e da quantidade (10) são fixos. O ideal seria poder alterar esses valores a cada execução do programa. Para isso, teríamos de fazer a interação de introduzir um valor no programa, durante a sua execução, por meio do teclado.

A instrução primitiva, no caso, é o **leia**. Essa instrução interrompe a execução e espera a interação do usuário para introduzir uma informação, que será armazenada na variável indicada pelo comando e que será utilizada no transcorrer do programa.

```
leia(<variável>)  
leia(<lista de variáveis>)
```



### Observação

O **leia** pode aparecer como **ler**, conforme o autor.

No fluxograma (Figura 9), a representação fica:

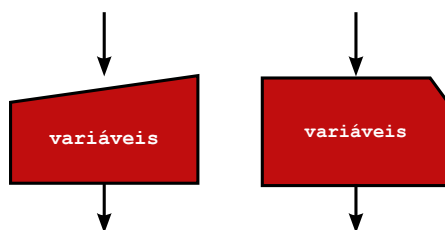


Figura 9 – Símbolo para a entrada de informações

Exemplo: voltando ao programa do exemplo anterior, agora será alterado para entrar com os valores da quantidade e do preço do produto.

```
algoritmo "ExemploSaida"  
var  
PR_UNIT, PR_TOT: Real  
QTD: Inteiro  
inicio  
    leia (PR_UNIT, QTD<-10)  
    PR_TOT<-PR_UNIT*QTD  
    escreva (PR_TOT)  
fimalgoritmo
```

Assim, o fluxograma (Figura 10) será alterado também, para receber a entrada de informações.



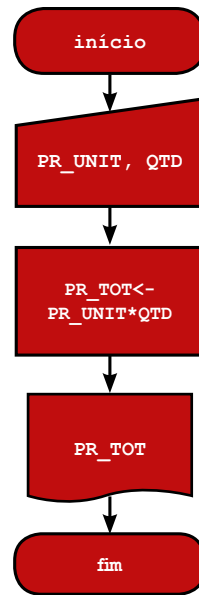


Figura 10 – Programa com entrada e saída

## 1.10.3 Teste de mesa

Um algoritmo precisa ser validado pela utilização do teste de mesa. Nesse teste, você assume o papel do processador e fica responsável por executar cada linha do código. O teste permite verificar se o código está realmente executando a função para a qual foi criado e se existe algum erro de lógica nos passos apresentados.

Esse teste é de uma época em que programar um computador era extremamente complicado, e cada processamento era muito caro. Processar um código incerto era um desperdício de tempo, e, portanto, era necessário ter certeza de que aquele programa funcionaria. Nos testes manuais, os programadores faziam simulações de execução e corrigiam erros encontrados antes mesmo de executar o código na máquina real. Essa prática se perdeu com o tempo, com o advento da programação interativa (na qual o programador tem acesso à máquina e pode testar alterações).

Para fazer um teste de mesa, inicialmente, vamos estudar um código.

O programa foi escrito para calcular o valor de um termo de uma progressão aritmética (PA) que fica determinada pela sua razão ( $r$ ) e pelo primeiro termo ( $a_1$ ). Esta é dada pela fórmula:

$$a_n = a_1 + (n - 1) \times r$$

```
algoritmo "PA"  
// Seção de Declarações  
Var  
razao, n, a1, an:inteiro  
Inicio  
// Seção de Comandos
```

```
escreva("Entre com a razao:")
leia(razao)
escreva("Entre com o primeiro termo")
leia(a1)
escreva("Entre com o elemento N")
leia(n)
an<-a1+(n-1)*razao
escreva("O elemento",n," é ",an)
finalgoritmo
```

Vamos fazer o teste de mesa verificando o resultado para  $r = 5$ ,  $a_1 = 12$  e  $n = 4$ .

Inicialmente, vamos verificar qual o valor esperado efetuando matematicamente a fórmula:

$$a_4 = 12 + (4 - 1) \times 5$$
$$a_4 = 27$$

Portanto, o valor esperado é 27.

Como estamos assumindo o papel do computador, executaremos sequencialmente cada linha do programa.

- Iniciando a execução, o computador está com a memória totalmente vazia.

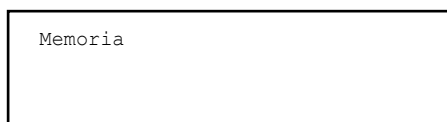


Figura 11 – Memória no início da execução do programa

- Na primeira linha, encontramos a identificação do programa.

```
algoritmo "PA"
// Seção de Declarações
Var
razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
escreva("Entre com a razao:")
leia(razao)
escreva("Entre com o primeiro termo")
leia(a1)
escreva("Entre com o elemento N")
leia(n)
an<-a1+(n-1)*razao
escreva("O elemento",n," é ",an)
finalgoritmo
```

Assim, reservamos a memória para o programa PA.

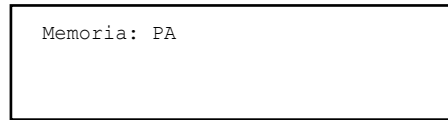


Figura 12 – Memória reservada para o programa PA

- Na linha seguinte, encontramos um comentário, portanto este será ignorado.

```
algoritmo "PA"
// Seção de Declarações
Var
  razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
  escreva("Entre com a razao:")
  leia(razao)
  escreva("Entre com o primeiro termo")
  leia(a1)
  escreva("Entre com o elemento N")
  leia(n)
  an<-a1+(n-1)*razao
  escreva("O elemento",n," é ",an)
finalgoritmo
```

A memória continua sem alteração.

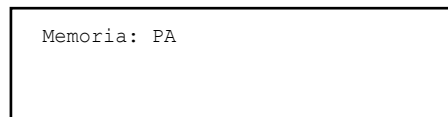


Figura 13 – Memória sem mudanças

- A seguir, encontramos a seção de declaração das variáveis.

```
algoritmo "PA"
// Seção de Declarações
Var
razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
  escreva("Entre com a razao:")
  leia(razao)
  escreva("Entre com o primeiro termo")
  leia(a1)
  escreva("Entre com o elemento N")
```

```
leia(n)
an<-a1+(n-1)*razao
escreva("O elemento",n," é ",an)
Fimalgoritmo
```

Nesse ponto, o computador criará os espaços na memória para as variáveis. Em nossa memória simulada, cada variável será representada em uma coluna.

Memória: PA			
razao	n	a1	an

Figura 14 – Espaços criados na memória pelo computador

- Montadas as variáveis, é iniciado o processamento do programa, que encontra mais um comentário e encontrará um comando de saída.

```
algoritmo "PA"
// Seção de Declarações
Var
razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
escreva("Entre com a razão:")
leia(razao)
escreva("Entre com o primeiro termo")
leia(a1)
escreva("Entre com o elemento N")
leia(n)
an<-a1+(n-1)*razao
escreva("O elemento",n," é ",an)
fimalgoritmo
```

Para simular a saída, criaremos uma nova coluna, chamada tela, em que serão simuladas as informações de entrada e saída (no caso, a instrução **escreva**).

Memória: PA				
razao	n	a1	an	Tela
				Entre com a razão

Figura 15 – Memória com tela para as informações de entrada e saída

- No passo seguinte, o programa esperará que o usuário digite um valor, que será atribuído à variável **razão**.

```
algoritmo "PA"  
// Seção de Declarações  
Var  
razao,n,a1,an:inteiro  
Inicio  
// Seção de Comandos  
    escreva("Entre com a razao:")  
    leia(razao)  
    escreva("Entre com o primeiro termo")  
    leia(a1)  
    escreva("Entre com o elemento N")  
    leia(n)  
     $an \leftarrow a1 + (n-1) * razão$   
    escreva("O elemento",n," é ",an)  
Fimalgoritmo
```

Entramos com o valor 5, pois este foi definido como condição para a simulação. Assim, a informação será armazenada na variável razão.

Memoria: PA				
<b>razao</b>	<b>n</b>	<b>a1</b>	<b>an</b>	<b>Tela</b>
5				Entre com a razão 5

Figura 16 – Memória com tela mostrando o valor 5

- O computador encontra, no passo seguinte, mais uma instrução de saída.

```
algoritmo "PA"  
// Seção de Declarações  
Var  
razao,n,a1,an:inteiro  
Inicio  
// Seção de Comandos  
    escreva("Entre com a razao:")  
    leia(razao)  
    escreva("Entre com o primeiro termo")  
    leia(a1)  
    escreva("Entre com o elemento N")  
    leia(n)  
     $an \leftarrow a1 + (n-1) * razão$   
    escreva("O elemento",n," é ",an)  
Fimalgoritmo
```

O texto será mostrado na tela.

Memoria: PA				
razao	n	a1	an	Tela
5				Entre com a razão 5
				Entre com o primeiro termo

Figura 17 – Memória com tela mostrando o texto

- A seguir, é feita a leitura, para atribuir valor à variável a1.

```

algoritmo "PA"
// Seção de Declarações
Var
razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
escreva("Entre com a razao:")
leia(razao)
escreva("Entre com o primeiro termo")
leia(a1)
escreva("Entre com o elemento N")
leia(n)
an<-a1+(n-1)*razao
escreva("O elemento",n," é ",an)
Fimalgoritmo
    
```

A variável a1 receberá 12.

Memoria: PA				
razao	n	a1	an	Tela
5		12		Entre com a razão 5
				Entre com o primeiro termo 12

Figura 18 – Atribuição do valor 12 à variável a1

- Novamente, é executada a saída de um texto.

```

algoritmo "PA"
// Seção de Declarações
Var
razao,n,a1,an:inteiro
    
```

```
Inicio
// Seção de Comandos
  escreva("Entre com a razao:")
  leia(razao)
  escreva("Entre com o primeiro termo")
  leia(a1)
escreva("Entre com o elemento N")
  leia(n)
   $an \leftarrow a1 + (n-1) * razão$ 
  escreva("O elemento", n, " é ", an)
Fimalgoritmo
```

O texto será exibido na tela.

Memoria: PA				
razao	n	a1	an	Tela
5		12		Entre com a razão
				5
				Entre com o primeiro termo
				12
				<b>Entre com elemento N</b>

Figura 19 – Memória mostrando o texto na tela

- Finalmente, será feita a leitura do valor do termo que queremos saber na PA.

```
algoritmo "PA"
// Seção de Declarações
Var
  razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
  escreva("Entre com a razao:")
  leia(razao)
  escreva("Entre com o primeiro termo")
  leia(a1)
  escreva("Entre com o elemento N")
leia(n)
   $an \leftarrow a1 + (n-1) * razão$ 
  escreva("O elemento", n, " é ", an)
Fimalgoritmo
```

Nesse momento, a variável razão contém o valor 5, a variável n contém 4, a variável a1 contém 12 e a variável an continua vazia.

Memória: PA				
razao	n	a1	an	Tela
5	4	12		Entre com a razão 5 Entre com o primeiro termo 12 Entre com o elemento N 4

Figura 20 – Memória exibindo somente a variável an sem valor atribuído

- É efetuado o cálculo substituindo-se os nomes das variáveis pelo seu conteúdo e atribuindo o resultado à variável an.

```

algoritmo "PA"
// Seção de Declarações
Var
razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
escreva("Entre com a razao:")
leia(razao)
escreva("Entre com o primeiro termo")
leia(a1)
escreva("Entre com o elemento N")
leia(n)
an<-a1+(n-1)*razão
escreva("O elemento",n," é ",an)
Fimalgoritmo
  
```

O resultado 27 será armazenado na variável an.

Memória: PA				
razao	n	a1	an	Tela
5	4	12	27	Entre com a razão 5 Entre com o primeiro termo 12 Entre com o elemento N 4

Figura 21 – Memória com o resultado 27 armazenado na variável an

- O passo seguinte é mostrar o resultado.

```

algoritmo "PA"
// Seção de Declarações
  
```



```
Var
razao,n,a1,an:inteiro
Inicio
// Seção de Comandos
  escreva("Entre com a razao:")
  leia(razao)
  escreva("Entre com o primeiro termo")
  leia(a1)
  escreva("Entre com o elemento N")
  leia(n)
  an<-a1+(n-1)*razão
  escreva("O elemento",n," é ",an)
Fimalgoritmo
```

Como n vale 4, an vale 27, e a saída será **"O elemento 4 é 27"**.

Memória: PA				
razao	n	a1	an	Tela
5	4	12	27	Entre com a razão
				5
				Entre com o primeiro termo
				12
				Entre com o elemento N
				4
				<b>O elemento 4 é 27</b>

Figura 22 – Memória com tela mostrando o resultado

- A última linha é apenas uma indicação de encerramento do programa.

Ao final da simulação, o valor foi 27; correspondendo ao calculado matematicamente, podemos considerar o programa correto.

Para fazer um bom teste de mesa, há algumas sugestões. Existirão termos ainda não estudados, mas devemos nos lembrar deles no futuro.

- Quando o programa for muito extenso e existir um trecho a ser estudado, identifique as variáveis que serão necessárias, pois algumas poderão ser inúteis naquele trecho.
- Sempre fique atento aos limites dos laços (que veremos mais adiante), pois a repetição pode levar ao engano no controle; faça cada iteração, pois pode existir um ponto de saída sem o laço estar concluído.
- Preste atenção nas funções chamadas durante a iteração dos laços.
- Como vimos, execute cada linha do programa, e na sequência correta.



### Resumo

Nesta unidade, vimos os princípios básicos da programação: o processo da montagem de um programa, desde a apresentação de um problema, passando pelo entendimento, até a transformação em um algoritmo.

Aprendemos os conceitos de lógica de programação, aquele raciocínio lógico para a resolução de um problema, e de algoritmo, a sequência de instruções montadas de forma que facilmente sejam convertidas em qualquer linguagem de programação, que é o algoritmo escrito numa linguagem que o computador (ou o compilador) entende.

Para a montagem do algoritmo, vimos que existem três possibilidades de representar e desenvolver a lógica: a descrição narrativa, que é a narração em português do dia a dia, que tem a vantagem de não precisar de conhecimento prévio de programação, mas abre a possibilidade de dupla interpretação; o fluxograma, que faz uso de símbolos gráficos de fácil visualização, mas de difícil construção; e o Portugol, que exige um prévio conhecimento de estruturas de programação, mas facilita o desenvolvimento lógico e a tradução para a linguagem de máquina.

Uma vez apresentados os fundamentos, passamos a aprender os conceitos básicos para a programação. Inicialmente, vimos quais informações um programa manipula, os dados, os tipos primitivos, que são os números inteiros e reais, os lógicos (falso e verdadeiro) e os caracteres. Uma vez vistos os tipos de dados, abordamos como estes podem ser trabalhados no programa e como são armazenados em memórias chamadas variáveis.

Com os dados na memória, passamos a ver como trabalhar com eles por meio de operações, tais como a operação de atribuição para movimentar os valores, as operações aritméticas (+, -, \*, /, MOD e DIV), os operadores relacionais que comparam grandezas (>, >=, <=, =, <>) e os operadores lógicos (e, ou, não, XOR). Lembrando que todos esses operadores podem ser colocados em uma única expressão e calculados obedecendo a uma hierarquia que determina quais operações devem ser efetuadas antes e quais devem ser efetuadas depois.

Vimos, portanto, que um programa é uma sequência de comandos e operações, e aprendemos como essa sequência deve ser construída, a chamada estrutura sequencial.

Os primeiros comandos que vimos foram os de entrada e saída de informações, o **leia ()** e o **escreva ()**.

Um programa, uma vez escrito, precisa ser testado, bem como ter verificados a sua funcionalidade e o correto funcionamento lógico. Para isso, a ferramenta que nos acompanhará até o final do curso é o teste de mesa, em que nós assumimos o papel do computador e simulamos o processamento.



### Exercícios

**Questão 1.** Em uma sequência de ação, procura-se atingir certo objetivo com recursos de tomadas de decisões e escolhas entre várias alternativas, segundo as experiências adquiridas. No entanto, um computador para atingir seu objetivo precisa ser instruído através de algoritmos, utilizando-se lógicas corretas para que suas tarefas possam ser executadas. Com base nessas informações, escolha a alternativa que contém a afirmação correta:

- A) O algoritmo é um conjunto infinito de instruções, comandos e ações que têm como objetivo a resolução de uma tarefa, ou a solução de um problema.
- B) Algoritmo é uma sequência finita de instruções ou operações cuja execução, em tempo finito, resolve um problema computacional, qualquer que seja sua instância.
- C) Algoritmo é uma sequência de passos que visam atingir um objetivo bem definido. Portanto, um algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão infinita de ações.
- D) Algoritmo são regras formais para a obtenção de um resultado ou da solução de vários problemas, englobando fórmulas de expressões aritméticas.
- E) Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma ou mais tarefas.

Resposta correta: alternativa B.

### Análise das alternativas

A) Alternativa incorreta.

Justificativa: uma vez que tem como objetivo a solução de um problema, um algoritmo deve conter um conjunto finito de instruções, além de permitir que estas sejam executadas pelo computador.

B) Alternativa correta.

Justificativa: uma solução computacional deve ser apresentada através de um algoritmo que possua sequências de instruções, operações e lógicas corretas para que possam ser executadas por uma máquina.

C) Alternativa incorreta.

Justificativa: um algoritmo deve conter um conjunto de comandos de forma sequencial para que suas ações possam ser executadas por um computador e atingir seu objetivo.

D) Alternativa incorreta.

Justificativa: uma regra deve ser obedecida para que seja possível a realização de uma tarefa. A sua solução envolve fórmulas e expressões matemáticas para solucionar um problema.

E) Alternativa incorreta.

Justificativa: a descrição da solução de um problema deve ser realizada através da descrição de um algoritmo, de forma sequencial, para a solução de apenas uma tarefa.

**Questão 2.** Utilizamos o pseudocódigo para desenvolver o raciocínio que será utilizado para criar um programa, e não nos preocupamos se estamos escrevendo dentro das normas rígidas como as exigidas por uma linguagem de programação. Podemos classificar os tipos de informações a serem processadas pelo computador em dados e instruções. Os dados são as informações a serem processadas por um computador, e as instruções são os comandos que orientam o processamento feito por um computador. Conforme as afirmações descritas, escolha a alternativa que contém a afirmação correta:

A) Os algoritmos e as linguagens de programação trabalham com dados bem definidos. Estes dados são classificados em tipos. Conforme o tipo de dado, o computador trabalha de forma igual no tratamento deles durante o processamento.

B) Por princípio, os computadores trabalham com quatro tipos de dados. Partindo destes tipos, as linguagens de programação derivam em outros tipos mais genéricos, ou com precisão maior, tais como: inteiros, reais, caracteres e lógicos.

C) Quando um dado não sofre nenhuma variação durante a execução do programa é chamado constante. O seu valor deverá ser o mesmo do início ao fim da execução do programa, assim como para execuções iguais.

D) As variáveis são espaços reservados na memória principal do computador, na qual os dados são armazenados para uma posterior reutilização. Elas são especialmente úteis quando temos operações mais complexas ou expressões com parênteses.

E) Para batizar uma variável é recomendável dar-lhe um nome que lembre a informação que ela irá armazenar. Os nomes das variáveis são atribuídos pelo usuário e obedecem a certas regras, como variáveis – são as nomenclaturas.

**Resolução desta questão na plataforma.**