PROGRAMAÇÃO BÁSICA COM VISUALG

TEORIA E PRÁTICA

Mário Leite

Antonio Carlos Nicolodi

DEDICATÓRIAS

Dedico este livro às minhas quatro queridas, e inesquecíveis, professoras do Curso Primário do "Grupo Escolar Emílio Soares", da cidade de Tombos (MG), no período de 1956-1959, que me ensinaram a ler e escrever, para formar a base da minha Educação!

Olga de Oliveira Osken (in memorian)

Consuelo Idalina Ramos (in memorian)

Maria Passos Borba (in memoriam)

Terezinha Rezende Linhares

Mário Leite

Dedico esta obra à minha esposa Márcia e à minha filha Bianca que me deram seu apoio incondicional.

Antônio Carlos Nicolodi

CITAÇÃO

O computador só faz aquilo que você o manda fazer; e não, necessariamente, aquilo que você quer ele faça!

AGRADECIMENTOS

Agradeço primeiramente a Deus que me deu força e persistência para escrever mais este livro, depois de um bom tempo de muita pesquisa, dedicação e muitas revisões.

Agradeço a todos os meus colegas professores e professoras da Universidade de Uberaba (UNIUBE) pela força e incentivo que sempre me deram em todas as minhas obras.

Agradeço ao professor Antonio Carlos Nicolodi pelos preciosos esclarecimentos e atualizações da ferramenta Visualg, além desta sua preciosa parceria.

Agradeço ao Clube dos Autores, pela oportunidade desta publicação.

Mário Leite

Agradeço primeiramente aos meus pais por me dar a vida, à minha esposa Márcia e à minha filha Bianca, por me ajudarem nos momentos mais difíceis e nos de grandes alegrias; agradeço também aos meus professores.
Agradeço muito ao meu grande amigo e parceiro Mário Leite por me dar esta oportunidade de poder participar com ele desta obra, que deverá ajudar muita gente no futuro. Mas, principalmente, agradeço à Deus por me dar saúde e o conhecimento que pretendo compartilhar com todos.

Antonio Carlos Nicolodi

SOBRE OS AUTORES

Mário Leite é natural de Tombos (MG); estudou física durante dois anos no Instituto de Física da UFRJ; foi aluno de Iniciação Científica do Centro Brasileiro de Pesquisas Físicas (CBPF: Divisão de Física Nuclear) e do CNPq, no Rio de Janeiro. É graduado e pós-graduado em engenharia pela Pontifícia Universidade Católica do Rio de Janeiro (PUC/RJ), onde foi professor auxiliar de ensino e pesquisa do Departamento de Ciências dos Materiais e Metalurgia. É especialista em Análise de Sistemas pelo Centro Universitário de Maringá (UniCesumar/PR) e mestre em Engenharia de Produção pela Universidade Federal de Santa Catarina (UFSC).

Trabalhou durante quatro anos na Indústria e Comércio de Minérios (ICOMI), no estado do Amapá, como engenheiro de pesquisas, desenvolvendo aplicações para o setor de produção. Foi chefe do Setor de Informações Gerenciais da Mineração Caraíba S.A (BA) durante nove anos, ministrando cursos de técnicas de programação para os engenheiros da empresa, e desenvolvendo aplicações para os setores de produção e manutenção. Nesta empresa participou do projeto de mecânica das rochas para implantação do sistema de escavação da mina subterrânea, na adaptação do *software* de elementos finitos para microcomputadores. É professor de linguagem de programação, análise de sistemas e tecnologia da informação em cursos de processamento de dados, sistemas de informação e administração de empresas. Também é autor de vários livros nas áreas de linguagens e técnicas de programação e ferramenta de ambiente numérico. Atualmente é professor da Universidade de Uberaba (UNIUBE) nas áreas de ferramentas computacionais e linguagens de programação.

Antonio Carlos Nicolodi é natural de Blumenau (SC); estudou eletrônica no Centro Educacional Profissionalizante Herman Hering (CEDUPHH), estudou Mecanografia e processamentos de dados na ETEVI-Furb de Blumenau e Técnica em Informática para a Internet no Instituto Federal Santa Catarina (IFSC). É graduado em Gestão em Tecnologia da Informação e em Administração de Empresas pela Uniasselvi-Fameblu, com pós-graduado em Educação à Distância (EaD - Gestão e Tutoria) e Governança em T.I. Trabalhou como programador na empresa Apoio Informática (Blumenau), foi gerente de CPD na Escola de Contabilidade aonde desenvolveu um sistema de integrado de contabilidade. Foi sócio proprietário da empresa Domínio Informática e programador chefe, onde desenvolveu vários sistemas ERP para a comunidade do Vale do Itajaí; foi professor no Colégio Doutor Blumenau na disciplina de programação Cobol para os cursos de (Processamento de dados, Administração de empresas e Contabilidade).

É professor de informática nas prefeituras de Gaspar e Ilhota, professor de lógica de programação no CEDUPHH (Blumenau); professor de arquitetura de computadores, lógica de programação de cursos de informática, jogos de computadores, entre outros, na Uniasselvi-Fameblu (Blumenau). Criador de vários métodos de ensino de lógica de programação, entre eles o método conhecido como *portugol* (algoritmos escritos em Português nos anos 1980-83 também conhecido como português estruturado), usado por vários professores/alunos e escritores (autores em suas obras literárias técnicas). É um dos desenvolvedores do *software* conhecido como Visualg até a versão 2.5 e o criador da versão 3.0 em 2014, que é o interpretador oficial de algoritmos escritos em *portugol*. Atualmente é professor, lecionando no horário noturno, e durante o dia é funcionário público na prefeitura de Ilhota (SC) na área administrativa.

PREFÁCIO

Eis-me aqui tecendo o prefácio do livro destes ilustres profissionais - o engenheiro e programador Mário Leite, e o professor Antônio Carlos Nicolodi, administrador, programador que há décadas tem se dedicado a área da tecnologia da informação sendo um dos idealizadores do Visualg.

É com agrado e satisfação que recebi o convite, por três motivos: primeiro, porque apesar de eu ser administradora, o mundo da programação sempre me fascinou; em segundo, por saber que ambos autores estão preocupados em facilitar o aprendizado em programação, ao apresentarem inúmeros exemplos de programas que exploram as denominadas Estruturas de Controle, e em terceiro porque encontraram uma nova forma de ensinar a programar, o que torna o aprendizado mais leve e eficaz.

Atuo na gestão de cursos que apresentam em algum momento do currículo a disciplina de programação e tenho observado a barreira que o próprio aluno coloca quando participa da primeira aula de programação, por ser uma disciplina vista como "*hard*". Sempre me perguntei o porquê da resistência em aprender programação.

Eu encontrei a resposta junto a estes dois autores, quando demonstram que para ensinar programação é preciso deixar para trás métodos tradicionais ainda muito presentes nas Instituições de Ensino Superior (IES). É preciso encontrar um novo sentido de ensinar programação mais dinâmico e interessante apontaram Mário e Antônio. À vista disso, uma das características mais notáveis no trabalho desenvolvido por esses autores é a preocupação de tornar evidente "o COMO FAZER".

A obra composta trabalha, basicamente, com exemplos, esquemas e exercícios dentro do ambiente do Visualg, o que proporciona rápido aprendizado na programação com esta ferramenta. Além disto, os exercícios propostos (e resolvidos ao

final do livro) é um fator decisivo para a complementação da formação do programador, sanando as dúvidas ao longo do curso. Tenho a convicção de que esta obra proporcionará a você adentrar no universo da programação com uma visão muito mais clara de que o COMO FAZER em Visualg pode significar tudo!
Boa leitura!
#Vamos aprender com prazer!

Andreia Mileski Zuliani Santos
Professora e Gestora de cursos EaD – Maringá-PR.

NOTA DOS AUTORES

Este livro tem como objetivo principal ensinar os programadores a utilizar o **Visualg** de maneira correta para tirar o melhor proveito no aprendizado de Lógica de Programação com esta ferramenta, nos diversos segmentos da Educação . É um guia que orienta os usuários, tanto nos seus recursos e nas suas funcionalidades básicas, quanto na própria linguagem da ferramenta. Mostra uma maneira melhor e mais didática de como criar, interpretar, editar e testar os algoritmos escritos em português estruturado, sem a necessidade de utilizar qualquer linguagem real, ajustando às suas necessidades.

Este livro mostra que este *software*, além ser prático e objetivo para testar algoritmos, também possui todas as funcionalidades de um bom editor de textos no padrão do Windows®, para criar programas bastante funcionais dentro de um ambiente de desenvolvimento integrado muito fácil de operar e com recursos que podem ser equiparados a outros IDEs mais sofisticados.

A obra apresenta o *software* como sendo uma verdadeira ferramenta no auxílio aos seus usuários, tanto para iniciantes quanto para os mais avançados na área de programação, auxiliando e mostrando várias formas de escrever e testar os seus algoritmos, nos mais variados tipos de problemas resolvidos por computador. Mostra a história da ferramenta, os seus novos recursos, as novas funcionalidades e também os novos comandos e funções da linguagem, além de muitos esquemas explicativos e exercícios prontos e propostos para melhorar a eficiência do aprendizado na criação de programas de computador.

Agosto de 2020

ORGANIZAÇÃO DO LIVRO

Capítulo 1 - Lógica de Programação

Neste capítulo o livro focaliza a "lógica de programação" iniciando com os conceitos básicos sobre instruções, e em seguida introduz o conceito de "algoritmo" de maneira bem clara e intuitiva e com exemplos básicos de programas simples. Explica como o algoritmo define a própria solução do problema; portanto um assunto fundamental na programação. Mostra a sequência básica de solução dos problemas através da programação: algoritmo-pseudocódigo-código. Discute a importância de se criar a lógica da programação. antes da codificação, frisando que codificar (criar o código-fonte na linguagem de programação) é apenas uma questão de saber a sintaxe da linguagem; mas, não é programar, pois o fundamental é achar a solução mais eficiente e mais eficaz para o problema; isto é, o fundamental é o "Como Fazer". No final do capítulo são propostos onze exercícios para o usuário exercitar o aprendizado.

Capítulo 2 - O Visualg

Neste capítulo o *software* Visualg é apresentado com sua história, seus recursos, seu funcionamento e sua linguagem baseada no *portugol*. Mostra como esta ferramenta substitui, com vantagens, os antigos "Testes de Mesa", permitindo ao programador a checagem imediata do algoritmo da solução do problema e o aperfeiçoamento dessa solução. Explica seus comandos e funções, além de apresentar a estrutura básica dos algoritmos escritos no seu editor. Este capítulo mostra os recursos dessa ferramenta, com esquemas e exemplos bem esclarecedores: como editar e executar um programa, como verificar e corrigir os

possíveis tipos de erros, No final são sugeridos onze exercícios para praticar.

Capítulo 3 - Tipos de Dados no Visualg

Neste capítulo são apresentados os tipos de dados suportados pelo Visualg, e como declarar elementos num programa. Também são mostrados os operadores e operações permitidos pela ferramenta, com exemplos bem esclarecedores de utilização dos tipos básicos de dados na programação. Ao final do capítulo são propostos doze exercícios para o leitor verificar seu aprendizado.

Capítulo 4 - Estruturas de Controle Visualg

Neste capítulo são apresentadas as estruturas de controle no Visualg, com as regras e os conceitos aplicados aos operadores lógicos e relacionais. Este capítulo trata dos chamados "comandos compostos", que permitem desviar o fluxo do programa, como também repetir blocos de instruções. São apresentadas as estruturas de desvios condicionais, desvio selecionado e estruturas de repetição (*loops*). Explica a importância dessas estruturas, pois, em programas reais a execução das instruções nem sempre são sequenciais. Também enfatiza a importância das indentações nos programas para melhorar a legibilidade e detecção de erros. Doze exercícios são propostos para verificação do aprendizado.

Capítulo 5 - Trabalhando com Vetores

Este capítulo inicia o estudo dos *arrays*, apresentando as variáveis compostas que definem estruturas homogêneas, através de vetores. Esses elementos são abordados neste capítulo sob uma ótica bem didática, objetiva e fácil de entender; com esquemas, figuras e exemplos esclarecedores. O assunto é tratado de maneira a aumentar o nível de compreensão do leitor e baixar o nível de dúvidas que muitas vezes aparecem em materiais de

programação. Com um formato bem didático nos programas, o assunto é explicado nos exemplos de modo bem contundente e sem sofisticações São estudadas aplicações práticas com vetores, além de operações, pesquisas e ordenações. Onze exercícios são propostos para verificar o aprendizado.

Capítulo 6 - Trabalhando com Matrizes

Este capítulo apresenta as variáveis compostas que definem estruturas homogêneas, através de matrizes. Complementando o assunto "*arrays*", este capítulo trata desses elementos, como uma generalização dos vetores, de maneira bem fácil de entender: começando bem tranquilo e aumentando o grau de complexidade bem devagar. Declaração, inicialização, leitura e impressão dos elementos de uma matriz são tratados de maneira bem simples, e enfatizando as características dos *arrays*. Operações com matrizes e cálculos de determinantes também são assuntos bem discutidos e estudados neste capítulo de maneira bem didática e prática. São propostos onze exercícios para o usuário trabalhar seu aprendizado.

Capítulo 7 - Trabalhando com Registros

Neste capítulo é apresentado o conceito de registro para dados heterogêneos, ao contrário dos vetores e matrizes que só trabalham com dados homogêneos. Este oitavo capítulo complementa os estudos sobre os tipos de estruturas de dados, como uma extensão dos *arrays*. O objetivo é mostrar como criar uma estrutura heterogênea em contraste com as estruturas homogêneas representadas pelos *arrays*. É explicado como criar estruturas - tal como uma tabela num banco de dados - para conter dados de tipos diferentes para uma mesma entidade. Para testar o aprendizado, sete exercícios são propostos ao final do capítulo.

Capítulo 8 - Modularização

Este capítulo aborda um tema importantíssimo para uma boa programação: a "modularização de um programa". O objetivo é mostrar que esta técnica é fundamental para que o programa tenha uma melhor legibilidade, além de ter sua manutenção bastante facilitada. Explica como criar as sub-rotinas (*procedimentos* e *funções*) nos programas, e como a rotina principal as chama e as executa. Explica com bastante clareza as diferenças entre *procedimento* e *função*, quanto ao modo de chamar, executar e sobre o resultado. Também discute os tipos de passagem de parâmetros: "por valor" e "por referência" através de esquemas práticos. Doze exercícios são propostos ao final deste capítulo para o usuário praticar.

Anexo - Soluções dos Exercícios Propostos

Neste anexo são apresentadas as soluções de todos os exercícios propostos ao longo dos nove capítulos estudados.

Bibliografia e Referências Bibliográficas

SUMÁRIO

Capítulo 1 - Lógica de Programação 1

1.1 - Introdução ao Capítulo..... 1
1.2 - Conceitos básicos de Lógica de Programação 1
1.3 - Algoritmo e Pseudocódigo 5
1.3.1 - Algoritmo 5
1.3.2 - Pseudocódigo 8
1.4 - Comando e Funções Internos de Pseudocódigo 10
1.5 - Etapas da Solução de um Problema..... 11
1.6 - Variáveis e Constantes..... 14
1.6.1 - Identificação de Variáveis 17
1.6.2 - Escopo de Variáveis..... 20
1.7 - Constantes..... 21
1.8 - Programação *x* Codificação 22
1.9 - Exercícios Propostos..... 26

Capítulo 2 - O Visualg 29

2.1 - Introdução ao Capítulo 29
2.2 - Básico sobre Aprendizagem em Programação 29
2.3 - Histórico do Visualg..... 31
2.4 - O Visualg no auxílio à Programação 34
2.4.1 - A Tela Principal do Visualg..... 35
2.4.2 - Características Básicas da Ferramenta 37
2.5 - Pseudocódigo *x* Visualg..... 39
2.6 - Funcionalidades Práticas do Visualg 43
2.7 - Itens do Menu Principal..... 44
2.7.1 - Opção “Arquivo”..... 44
2.7.2 - Opção “Editar” 48
2.7.3 - Opção “Run” 51
2.7.4 - Opção “Exportar”..... 53
2.7.5 - Opção “Manutenção” 53

2.7.6 - Opção “Help” 59
2.8 - Áreas Funcionais do Visualg 64
2.8.1 - Área do Programa..... 65
2.8.2 - Área das Variáveis de Memória 66
2.8.3 - Área de Visualização dos Resultados..... 72
2.9 - Funcionalidade dos Botões da Barra de Ícones 73
2.10 - A “Barra de Status” 79
2.11 - Palavras Reservadas 79
2.12 - Funções, Comandos Internos e Palavras Chaves..... 81
2.13 - Recursos de Cálculos Numéricos 84
2.14 - Manipulação de Caracteres..... 85
2.15 - Operações de Entrada e Saída..... 88
2.16 - Arredondando Valores de Saída 93
2.17 - Mensagens de Erros ao Rodar o Programa 95
2.17.1 - Erro de Sintaxe..... 95
2.17.2 - Erro de Computação..... 101
2.17.3 - Erro de Lógica (*bug*) 103
2.17.4 - Erro de Incompatibilidade de Tipos 105
2.17.5 - Erro de Instrução fora de área 107
2.17.6 - Erro de Falta de Terminadores 112
2.18 - Medindo o Tempo de Processamento 117
2.19 - Depurando Programas 118
2.20 - Comentários..... 120
2.21 - Exercícios Propostos..... 123

Capítulo 3 - Tipos de Dados no Visualg 127

3.1 - Introdução ao Capítulo..... 127
3.2 - Tipos Primitivos de Dados..... 127
3.3 - Tipos Complexos 130
3.4 - Os Tipos de Dados Considerados no Visualg..... 130
3.5 - Declaração de Variáveis no Visualg 131
3.6 - Operadores e Operações 137
3.7 - Exercícios Propostos 147

Capítulo 4 - Estruturas de Controle Visualg 151

4.1 - Introdução ao Capítulo 151
4.2 - Estruturas de Controle 152
4.2.1 - Estruturas de Decisão 154
4.2.1.1 - Estrutura de Decisão Simples 154
4.2.1.2 - Estrutura de Decisão Composta 158
4.2.2 - Estrutura de Decisão Selecionada 163
4.2.3 - Estruturas de Repetição (*loops*) 167
4.2.3.1 - Loop Lógico com Teste no Início ... 168
4.2.3.2 - Loop Lógico com Teste no Final 175
4.2.3.3 - Loop Numérico 179
4.3 - Aninhamento de Decisões usando Se's 184
4.4 - Indentações 185
4.5 - Exercícios Propostos 187

Capítulo 5 - Trabalhando com Vetores 193

5.1 - Introdução ao Capítulo 193
5.2 - Conceitos básicos 193
5.3 - Operações Elementares com Vetores 196
5.4 - Declaração de Vetores no Visualg 198
5.5 - Atribuição Direta 199
5.6 - Atribuição por Leituras Individualizadas 200
5.7 - Atribuição por Leituras Através de *Loop* 202
5.8 - Operações Matemáticas Simples com Vetores 206
5.8.1 - Soma de Vetores 206
5.8.2 - Subtração de Vetores 208
5.9 - Aplicações de Vetores 211
5.9.1 - Ordenação 211
5.9.2 - Pesquisas 215
5.9.2.1 - Pesquisa Sequencial 216
5.9.2.2 - Pesquisa Binária 219
5.9.3 - Verificando Maior e Menor elemento 222
5.9.4 - Soma e Média dos elementos 224

5.9.5 - Invertendo a Ordem dos elementos 227
5.9.6 - Verificando Característica de elemento..... 229
5.9.6.1 - Verificando elemento Múltiplo.....229
5.9.6.2 - Verificando elemento Divisor 231
5.9.6.3 - Verificando Paridade e Posição . . 233
5.9.7 - Incluindo/Excluindo/Alterando elemento ..235
5.9.7.1- Incluindo um novo elemento 235
5.9.7.2 - Excluindo um elemento 238
5.9.7.3 - Alterando o valor do elemento 240
5.10 - Erros na utilização de Vetores 245
5.11 - Exercícios Propostos..... 247

Capítulo 6 - Trabalhando com Matrizes 253

6.1 - Introdução ao Capítulo 253
6.2 - Conceitos Básicos 253
6.3 - Declaração de Matrizes no Visualg 256
6.4 - Leitura e Escrita de uma Matriz 258
6.5 - Operações com Matrizes 261
6.5.1 - Adição de Matrizes 261
6.5.2 - Subtração de Matrizes 265
6.5.3 - Multiplicação de Matriz por Escalar 268
6.5.4 - Multiplicação de Matriz por Vetor 271
6.5.5 - Multiplicação de Matriz por Matriz 273
6.6 - Determinantes de Matrizes 278
6.6.1 - Determinante de Matrizes de Ordem 2 278
6.6.2 - Determinante de Matrizes de Ordem 3 280
6.6.3 - Determinante de Matrizes de Ordem > 3 282
6.7 - Matriz Transposta 286
6.8 - Matriz Identidade 288
6.9 - Elementos das Diagonais de uma Matriz..... 290
6.10 - Exercícios Propostos..... 298

Capítulo 7 - Trabalhando com Registros 303

7.1 - Introdução ao Capítulo 303
7.2 - Conceitos Básicos 303
7.3 - Manipulação de Registros 305
7.4 - Conjunto de Registros 309
7.5 - Leitura de um Conjunto de Registros 311
7.6 - Pesquisa em Conjunto de Registros 318
7.7 - Exercícios Propostos 326

Capítulo 8 - Modularização 327

8.1 - Introdução ao Capítulo 327
8.2 - Conceitos Básicos 328
8.3 - Dividir para Unificar 329
8.4 - Programação Modular 329
8.5 - Sub-rotinas 333
8.5.1 - Procedimentos 334
8.5.2 - Funções 336
8.6 - Parâmetros 342
8.6.1 - Passagem de Parâmetros 344
8.6.1.1 - Passagem Por Valor 349
8.6.1.2 - Passagem Por referência 350
8.7 - Programas com Sub-rotinas 354
8.8 - Acesso a Arquivos 369
8.9 - Exercícios Propostos 373

Anexo - Solução dos Exercícios Propostos 379

Capítulo 1 379
Capítulo 2 383
Capítulo 3 388
Capítulo 4 395
Capítulo 5 405
Capítulo 6 415

Capítulo 7 424
Capítulo 8 428

Bibliografia e Referências Bibliográficas 437

Capítulo 1 - Lógica de Programação

1.1 - Introdução ao Capítulo

O processo de criação de um programa passa, necessariamente, pela análise do problema a ser resolvido, e em seguida sua solução deve ser criada com uma sistemática que possa resolver o problema. Essa sistemática é baseada numa lógica que deve implementar uma solução. E, embora uma linguagem de programação seja importante para automatizar a solução do problema, o programa tem que ser elaborado, antes, de modo que essa solução seja a melhor possível. Ao final, a tradução do código-fonte numa linguagem de programação deve ser "introduzida" no computador para dar a informação desejado ao usuário. A etapa que antecede a criação do código-fonte em linguagem de programação, é o que implementa a solução; por isto, é importante frisar que *codificar* (criar o código-fonte na linguagem de programação) é apenas uma questão de saber a sintaxe da linguagem; mas, isto não cria a solução do problema, pois o fundamental é achar a solução mais eficiente e mais eficaz,

1.2 - Conceitos Básicos de Lógica de Programação

Lógica de Programação pode ser definida como uma "*técnica de desenvolver sequências de ações para atingir um objetivo bem determinado*". Essas sequências são adaptadas para uma linguagem de computador a fim de produzir o *programa*.

Atualmente toda as comunicações dos usuários com os computadores são feitas através de *bytes*, que trazem no seu contexto as ordens para que o computador resolva o problema

apresentado pelo usuário, na forma de um programa. Na prática, o problema é analisado, estudado e manipulado por um outro personagem no ambiente da computação: o **programador**. O programador é quem cria e escreve as ordens que serão dadas ao computador para que apresente uma solução para o problema. A **figura 1.1** mostra um esquema de como fica a interação: *usuário-programador-computador*.

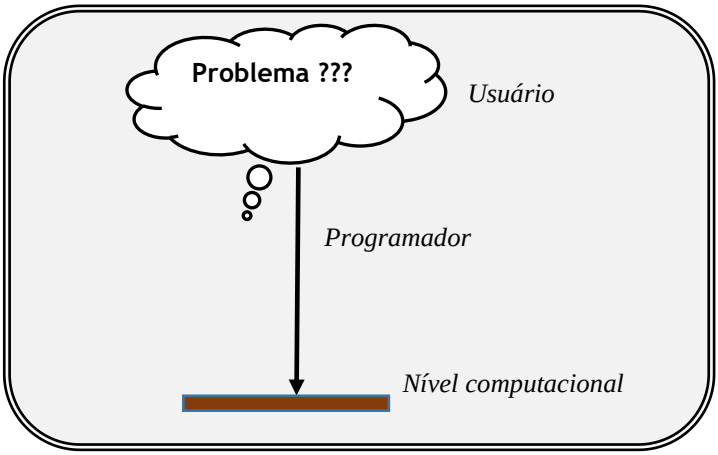


Figura 1.1 - Trazendo o problema para o nível computacional

O usuário apresenta o problema ao programador, que o refina e traduz para o computador executar as ordens embutidas nesses refinamentos. E para que essas ordens sejam executadas corretamente, elas devem ser escritas como texto, e numa sequência lógica, tal que a solução apresentada pelo computador seja correta e satisfaça o usuário. Essas ordens são traduzidas (codificadas) num texto chamado de **código-fonte**.

Existem várias definições para "Lógica", além daquela mencionada no início deste item, mas, de um modo geral, é possível resumir essas definições em uma só: "*Logica é a forma ordenada do pensamento*". Do ponto de vista prático, pode-se dizer que lógica de programação é a *forma ordenada de escrever*

as ações para resolver um problema. Para os programadores, é muito importante saber distinguir três definições:

- Linguagem de programação é um conjunto de regras que definem como os comandos e funções devem se comportar dentro dos programas de computador.
- Programas são as ordens dadas ao computador para que execute as tarefas exigidas no processamento, a partir de um conjunto de comandos e funções, seguindo uma sequência estabelecida.
- Algoritmo é o conjunto de instruções na forma de texto descritivo, usando um método de montagem conforme a lógica, para resolver o problema,

Os métodos de ensino de disciplinas de "Lógica de Programação" têm nomes variados, mas com o mesmo propósito, como:

- Técnicas de Programação.
- Algoritmos.
- Lógica de Programação.

Essas disciplinas apresentam em sua ementa vários métodos de ensino da lógica; entre os quais podem ser destacados:

1) **Método do Algoritmo Descritivo**: que literalmente descreve passo a passo cada etapa de um processo da solução do problema.

2) **Método dos Diagramas Gráficos** (fluxograma e diagramas de blocos): onde, por meio de desenhos gráficos é descrito o fluxo do processo, mostrando como funciona o raciocínio lógico e as estruturas de controle.

3) **Método da Pseudolinguagem** (ou **Pseudocódigo**), que pode ser apresentado através de uma pseudolinguagem de programação, como o *portugol* (português estruturado).

O *portugol* (que é a junção de **Portu**guês+al**go**l) é uma pseudolinguagem de programação; ou seja, é um método para ensinar lógica de programação, muito importante para os iniciantes no assunto "programação de computadores". Ele permite ao aluno entender o que cada comando (ordem) significa, antes mesmo de saber como será o programa real; por isto é um método muito poderoso. Um algoritmo não é um programa, mas ele exprime a lógica do processo para solucionar o problema; o aluno deve construir um algoritmo antes de codificar em linguagem de programação real.

Todas as linguagens de programação têm seus códigos-fonte escritos em Inglês; isto por que esse idioma é o que mais se adapta ao ambiente computacional, devido ao seu poder de compactação dos termos. ENTRETANTO, é preciso distinguir BEM o termo "**programação**" do termo "**codificação**"!!!

"*Programação é o ato intelectual de escrever a solução do programa em função do algoritmo (solução descritiva do problema), e que pode ser aprimorada com uma solução mais formal (pseudocódigo)*."

"*Codificação é o ato técnico de converter a solução escrita pseudocódigo em texto (o chamado código-fonte), baseado na sintaxe da linguagem escolhida*".

Embora, na prática possa ser dito "programar em Java", na verdade, o correto seria: *"codificar em Java*", pois a programação para resolver o problema é feita através da seguinte sequência:

1. Análise do problema;
2. Criação do algoritmo para resolver o problema, em nível informal;
3. Criação do pseudocódigo para formalizar a solução em nível mais técnico;
4. Codificação do programa.

É esta a sequência - em uma explicação bem simples - que deve ser executada para resolver um problema num ambiente computacional.

1.3 - Algoritmo e Pseudocódigo

Embora muitos autores tratem os dois termos como sendo a mesma coisa, existe uma diferença que pode ser bem marcante entre algoritmo e pseudocódigo.

1.3.1 - Algoritmo

Existem muitas definições para "algoritmo"; uma delas é dada da seguinte forma: "*Algoritmo é uma sequência de passos factíveis de execução, com início e fim, que quando executados produzem a solução de um problema*", Leite (2006, p. 54). Um exemplo clássico de algoritmo é o famoso problema "Trocar uma lâmpada", resumido nos seguintes passos na descrição a seguir...

Início

1 - Pegue uma lâmpada nova.
2 - Teste a lâmpada.
3 - **Se** *lâmpada passar no teste Passo 4*; **Senão** Passo 1
4 - Pegue uma escada.
5 - Posicione a escada no local correto.
6 - Suba os degraus da escada.
7 - Retire a lâmpada queimada.
8 - Coloque a nova lâmpada.
9 - Desça a escada.
11 - Descarte adequadamente a lâmpada queimada.

Fim

Embora os termos **Início** e **Fim** não sejam obrigatórios, é importante colocá-los para frisar que um algoritmo sempre deve ter um início e um fim bem determinados, como citado por Leite (2006, p. 54). Pelo esquema da solução do problema "Trocar uma lâmpada", pode ser observado que as ações dadas a cada passo sempre devem estar colocadas em verbos no modo Imperativo Afirmativo: *Pegue*, *Teste*, *Suba*, etc, para dar sentido de **ordenar, exigir**! Então, pelo que foi visto, pode-se afirmar que: "*Algoritmo é uma descrição textual e informal da solução do problema*". Em outras palavras, o algoritmo é a própria solução do problema; se o algoritmo estiver correto, o problema está resolvido. Simples, assim!

❖ **Exemplo 1.1** - Fazer um algoritmo para verificar se um número **A** é divisível por um número **B**.

Início

1 - Leia o número A.

2 - Leia o número B.

3 - **Se** B diferente de 0, **Então** divida A por B e siga em frente, **Senão** vá ao Passo 5.

4 - Mostre o resultado.

5 - Pare.

Fim

A solução do **Exemplo 1.1** já apresenta uma ação que é muito comum (fundamental) na lógica de programação: **Leia**. Ela ordena ao processador a "pegar" o valor digitado pelo usuário e armazená-lo na memória RAM, definindo o que se chama de *entrada* de dados através de leitura. A ação (comando) **Se** define uma ordem condicional, analisando uma situação que pode ter um dos dois valores lógicos possíveis: ***Verdadeiro*** ou ***Falso***. **Então**, se o número **B** for um número diferente de **0** (a Matemática não define divisão por 0), a operação será efetuada e o resultado mostrado, no **passo 4**;. **Senão** (caso contrário), se **B** for igual a zero, o **passo 4** não será executado e o fluxo do algoritmo será desviado para o passo 5 (último passo do algoritmo), encerrando a solução.

Nota: As ações (comandos) **Se**, **Então** e **Senão** são empregados em "situações" em que haja necessidade de se tomar uma decisão; são termos que definem comandos condicionais dentro num algoritmo. Estes serão vistas com mais detalhes no **Capítulo 4**.

❖ **Exemplo 1.2** - Criar um algoritmo para verificar se um determinado número é par ou ímpar.

Início

1 - Leia o número.

2 - Divida o número por 2.

3 - **Se** o resto da divisão for 0**, Então** siga em frente, **Senão** vá para o Passo 6.

4 - Escreva “O número é par”.

5 - Vá para o passo 7.

6 - Escreva “O número é ímpar”

7 - Pare

Fim

1.3.2 - Pseudocódigo

O pseudocódigo é o algoritmo escrito numa forma mais formal, mais técnica, utilizando as ordens convertidas em comandos mais rigorosos. Deste modo, as ações passam a ter um carácter mais computacional, o que torna a solução do problema menos descritiva e mais compacta. O esquema abaixo mostra o *layout* de um pseudocódigo, que aqui será adotado:

```
Programa "nome-do-programa"
    Declare <lista de variáveis do tipo 1>
    Declare <lista de variáveis do tipo 2>
    ...
Início
    Aqui são escritas as instruções do algoritmo
    Instrução1
    Instrução2
    Instrução3
    ...
FimPrograma
```

Observe como ficam as soluções dos dois exemplos anteriores em pseudocódigo.

```
Programa "Divide"
    Declare A, B, C: real
Início
    Leia(A)
    Leia(B)
     Se(B<>0) Então
        C <- A/b
    FimSe
    EscrevaLn(C)
FimPrograma
```

```
Programa "VerificaNumero"
    Declare A, R: inteiro
Início
    Leia(A)
    R <- Resto(A/2) //resto da divisão de A por 2
    Se(R=0)  Então
       Escreva("O número é par")
    Senão
       Escreva("O número é impar")
    FimSe
FimPrograma
```

Note que as soluções em pseudocódigo ficaram bem mais compactas, e com as ordens dadas de maneira mais técnica. Por exemplo, o comando **FimSe** para indicar a finalização de uma decisão, e **Escreva** para mostrar o resultado.

1.4 - Comandos e Funções Internos de Pseudocódigo

Nos programas escritos em pseudocódigo, às vezes, é necessário empregar comandos e/ou funções para agilizar o processo, como, por exemplo, "*pegar a parte inteira de uma divisão entre dois números"*. Se fosse utilizar uma descrição formal, isto seria muito complicado e ficaria mais ou menos assim:

```
...
Leia(N1)
Leia(N2)
Se(N2<>0) Então
    X <- Pega parte inteira da divisão de N1 por N2
    Escreva(X)
FimSe
...
```

Observe que a linha de código onde o resultado da divisão é atribuído à variável **X** ficou muito estranha; parecendo uma linha de algoritmo e não de pseudocódigo. A linha de instrução em pseudocódigo ficaria assim: **`X <- Int(N1/N2)`**, muito mais racional. Por isto, existem algumas *ordens* (comandos e funções) internas, utilizadas para melhorar a legibilidade do programa. A **tabela 1.1** mostra alguns comandos e funções que são utilizados nos pseudocódigos.

Ordem	**Tipo**	**O que Faz**
Declare v1, v2, v3	Comando	Cria variáveis de memória.
Escreva(x)	Comando	Exibe o conteúdo da variável **x** na tela e fica na mesma linha.
EscrevaLn(x)	Comando	Exibe o conteúdo da variável **x** na tela e salta para nova linha.
EscrevaLn("")	Comando	Apenas salta uma linha.

Leia(x)	Comando	Lê o conteúdo da variável **x** para a memória.
Pare(n)	Comando	Suspende o processamento por **n** milissegundos.
LimpaTela	Comando	Limpa a tela do monitor de vídeo.
Abandone	Comando	Sai do *loop* incondicionalmente.
Int(A/B)	Função	Retorna a parte inteira da divisão de **A** por **B**.
Resto(A/ B)	Função	Retorna o resto inteiro da divisão de **A** por **B**.
RaizQ(x)	Função	Retorna raiz quadrada de um número **x** não negativo.
Abs(x)	Função	Retorna o valor positivo de um número **x**.
Tamanho("s")	Função	Retorna o número de caracteres de uma *string* "s".

Tabela 1.1 - Alguns Comandos e Funções utilizados em pseudocódigos

1.5 - Etapas da solução de um Problema

Nas situações reais a solução de um problema pode ser mais complexa, exigindo técnicas de análise de sistemas e alguma complexidade. Mas, de qualquer forma, a sequência básica para resolver um problema através de um programa de computador, pode ser resumida no esquema da **figura 1.2**.

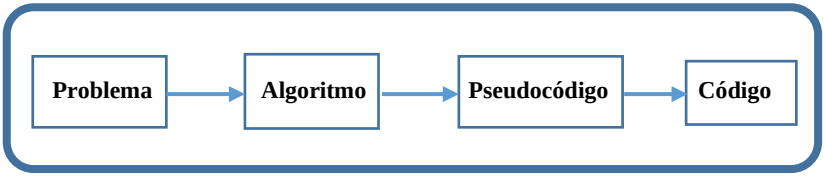


Figura 1.2 - Esquema básico da criação de um programa

❖ **Exemplo 1.3** - Ler dois números, fazer a troca de valores entre eles, e depois mostrar esses dois números com seus valores trocados.

Algoritmo:

Início

1 - Leia o primeiro número

2 - Leia o segundo número

3 - Troque os valores entre os dois números

4 - Mostre os dois números com seus novos valores

Fim

Pseudocódigo:

```
Programa "Trocas"
//Lê dois números, faz a troca entre eles.
//Autor: Mário Leite
//-------------------------------------------------------
    Declare A, B, X: inteiro
Início
    Escreva("Digite o primeiro número A: ")
    Leia(A)
    Escreva("Digite o segundo número B: ")
    Leia(B)
    X ← A  //preserva o valor do primeiro número
    A ← B  //faz o primeiro número igual ao segundo
    B ← X //faz o segundo número receber o preservado
    EscrevaLn("")  //salta linha
    EscrevaLn("Valor final de A: ", A)
    EscrevaLn("Valor final de B: ", B)
FimPrograma
```

Observe que, embora o algoritmo não mencione esse detalhe, no pseudocódigo a lógica mandou preservar o valor do primeiro número para, só no final, colocá-lo no segundo número; caso isto não fosse feito, os dois ficariam com um mesmo valor e a troca não seria efetivamente conseguida.

MAS... como saber se os resultados estão corretos, ANTES DA CODIFICAÇÃO!!??

Para testar a correção de um programa são utilizados os chamados "Testadores de Algoritmos", ou "Interpretadores de Algoritmos". E um desses, talvez o mais famoso e mais popular é o Visualg (**Visu**alizador de **Alg**oritmos) que será visto no próximo capítulo. Essa ferramenta - uma pseudolinguagem de programação - é muito importante para os iniciantes no assunto, pois permite testar o programa, antes mesmo de codificá-lo numa linguagem real e mais potente; pois o que interessa é verificar se o algoritmo está correto. E, embora o Visualg não seja uma linguagem formal de programação, ele é muito importante no aprendizado de "Lógica de Programação", pois permite ao programador ter uma visão mais clara da solução já em nível de pseudocódigo. Esta ferramenta pode ser classificada como "um interpretador de algoritmos", que facilita em muito o aprendizado, com um editor onde é escrito o pseudocódigo do programa, em *portugol*, e é distribuída pela Internet, gratuitamente.

Mas, qual é, então, a diferença entre Visualg e o Basic, por exemplo?! A diferença básica está nos termos empregados para uma mesma instrução; por exemplo, para exibir o valor da variável **X** no monitor de vídeo: em Visualg é **Escreva**(X) e no Basic é **Print**(X); em Visualg utiliza-se o idioma Português adaptado, e nas outras linguagens é sempre no idioma Inglês.

De tudo que foi visto até agora, pode ser concluído que só depois de passar pelo estudo do algoritmo/pseudocódigo e um interpretador de algoritmos é que se deve adotar uma primeira

linguagem de programação real, que poderá ser o Pascal, com qualquer um dos compiladores oferecidos no mercado; entre eles o **Pascal Zim!** que é distribuído gratuitamente na Internet. Portanto, codificar um programa numa linguagem real SEM ANTES passar pela sua lógica da solução, não é uma boa atitude de programação.

1.6 - Variáveis e Constantes

Nos exemplos anteriores de programas em pseudocódigo, foram empregadas muitas linhas de códigos com instruções em que uma letra (ou conjunto de letras e números) foi colocado à esquerda do símbolo ← como por exemplo, no programa "**Trocas**",

```
Aux ← A
A ← B
B ← Aux
```

Aux, **A1** e **B** são as "VARIÁVEIS" utilizadas para manipulação/armazenamento dos dados que a lógica do programa exigiu para resolver o problema proposto.

Mas... o que são variáveis; para que servem?!
A memória RAM de um computador pode ser comparada a um armário com várias gavetas, onde cada gaveta representa um local de armazenamento. Cada local desses é um endereço de memória, com uma identificação própria denominada endereço real e que só a CPU pode" acessá-lo diretamente. Então, como o programador não conhece o endereço real, ele tem que acessá-lo simbolicamente através do conceito de "variável de memória". De acordo com Leite (2006) "*Variáveis de memória, ou simplesmente variáveis, são endereços simbólicos da memória principal definidos pelo programador, onde são armazenados temporariamente valores para serem processados, ou resultado de algum processamento*". Quer dizer: *variável* pode ser

considerada uma unidade de armazenamento em nível de programação; e é importante saber que esses locais" são voláteis; os valores ali alocados podem ser alterados e/ou descartados depois de certo tempo, dependendo das circunstâncias. Por isto, o resultado de um processamento, que originalmente estava em uma variável, deve ser salvo de forma definitiva em uma mídia: fita, disco, *pen drive* etc; senão o valor será perdido definitivamente.

❖ **Exemplo 1.4** - Criar um programa para obter o salário líquido de um funcionário a partir de seu salário bruto, após descontar a contribuição para o INSS:

O Algoritmo poderia ser assim:

1 - Entre com o valor do Salário Bruto
2 - Entre com o valor do percentual de desconto
3 - Calcule o Desconto: DC = SB*(PD/100)
4 - Calcule o Salário Líquido: SL = SB - DC
5 - Mostre o resultado armazenado em SL

Note que cinco passos foram usados para calcular o salário líquido de apenas um empregado. Na verdade, uma empresa - mesmo sendo uma microempresa - não possui apenas um empregado; assim o processamento deve ser usado para todos eles. Deste modo, a cada processamento os valores de SB, PD, DC e SL se alteram; e se não forem salvos o valor de cada SL em locais permanentes, ao final do processamento de todos os funcionários da empresa só restará o valor do salário líquido do último empregado. Os termos SB e PD são variáveis utilizadas para armazenar os dados de entrada; DC é uma variável que armazena o valor calculado para o desconto, e SL uma variável onde é guardado o resultado do salário líquido de um funcionário;

mas todos eles são locais simbólicos da memória principal. A **figura 1.3** apresenta um esquema desta situação.

Nota: SB, PD, DC e SL são os nomes das variáveis empregadas no problema apresentado no **Exemplo 1.4**. Entretanto, conforme será explicado mais adiante, esses nomes deveriam ser mais explícitos para traduzir melhor o que cada um deles significa no contexto. São como "caixas" rotuladas para guardar algo que o programador deseja.

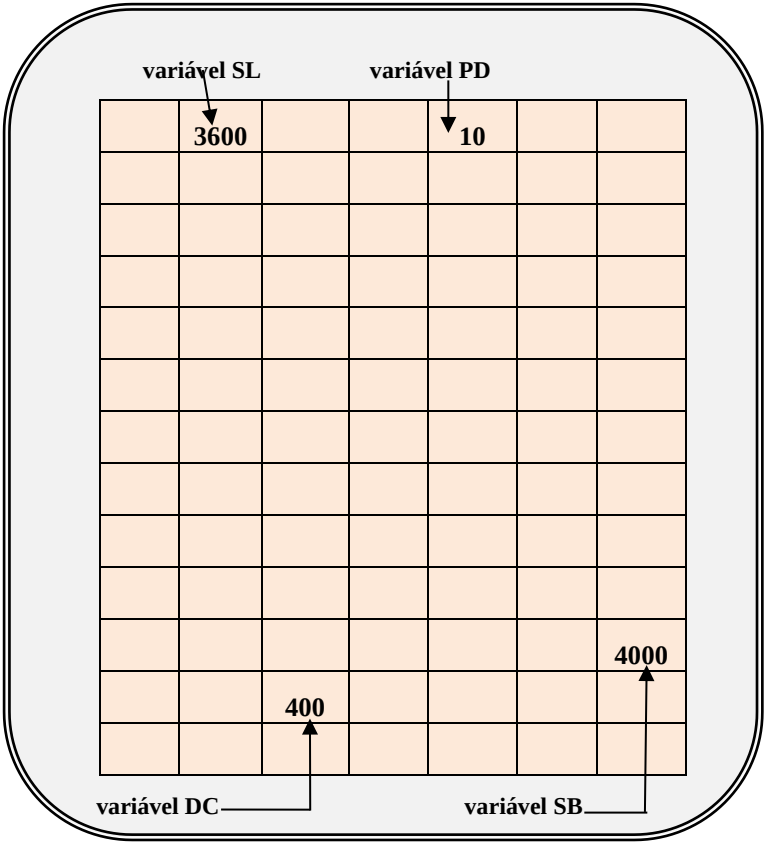


Figura 1.3 - Armazenamento de valores em variáveis de memória

Além do endereço e valor, uma variável também necessita ter um tipo de dado que qualifique o valor armazenado (tipos de dados serão vistos no **Capítulo 3**); em resumo: uma variável deve ser definida em função das seguintes características:

- Identificador ==> endereço (nome simbólico na memória).
- Valor ==> quantidade armazenada no momento.
- Tipo de Dado ==> qualidade do valor armazenado.
- Escopo ==> visibilidade (*abrangência*) dentro do programa.

```
Programa "ControleSalario"
//Controla o salário de um empregado
//Autor: Mário Leite
-----------------------------------------------------
   Declare SB, SL, PD, DC: real
Início
   Escreva("Digite o valor do salário bruto: ")
   Leia(SB)
   Escreva("Digite o percentual do desconto: ")
   Leia(PD)
   DC <- SB*(PD/100)
   SL <- SB - DC
   EscrevaLn("Salário líquido: ", SL)
FimPrograma
```

1.6.1 - Identificação de Variáveis

Nas literaturas sobre programação, dependendo até da linguagem, são sugeridas algumas regras para criar identificadores de uma variável; entretanto, a regra mais importante é esta: "*o*

identificador de uma variável deve traduzir o significado do que ela armazena". Um exemplo seria o de identificar uma variável que armazenasse o valor da devolução do imposto de renda do ano de 2020. Neste caso poder-se-ia optar por qualquer um dos nomes a seguir:

- IR
- DevIR
- DevIR20
- Devolucao_do_Imposto_de_Renda_de_2020

Qual destes identificadores você utilizaria?

O primeiro parece muito curto; o segundo parece mais prático que o primeiro, pois sugere uma devolução com o emprego do prefixo **Dev**. O terceiro é mais explícito, pois indica que a devolução é do ano de 2020. E o quarto identificador, também poderia ser utilizado? Sim poderia; mas não deveria, pois apesar de ser o mais explícito de todos é muito longo! Então, entre as quatro opções apresentadas o identificador que mais se aproxima do ideal seria o terceiro**: DevIR20**, pois além de traduzir bem o que é armazenado na memória não é muito curto e nem demasiadamente longo. De qualquer forma, a escolha do nome (*identificador*) de uma variável vai depender muito do bom senso do programador; não existe uma regra rígida para isto, apenas que ele explique o que vai ser armazenado naquele endereço, porém sem exageros; mas, existem algumas regras básicas:

1. Deve começar sempre com uma letra (**A-Z, a-z)** ou com *underline* (**_**).
2. Não pode ser utilizado caractere especial ou pontuação (**? > < @ &*, . # ; -**).
3. Não pode haver espaços em branco entre os caracteres.
4. O tamanho deve ser de no máximo 255 caracteres[5].

5. Não podem ser utilizadas letras acentuadas (**á ó ê ã**,) e nem o cedilha (**ç**).
6. Não pode utilizar palavras reservadas de linguagens de programação[6] ou de pseudocódigo.
7. Devem ser evitadas variáveis com identificadores contendo as letras "I", e "O" pois, podem ser confundidas com os dígitos 1 e 0, respecivamente.

Nota: Embora existam linguagens de programação que tratam de modo sensível os identificadores de variáveis (por exemplo, no C DevIr20 é diferente de DeveIR20) em pseudocódigo não existe *case sensitive* em relação a nome de variáveis. Portanto aqui DevIr20, DEVIR20, devIR20, Devir20, devir20, DeViR20, etc, são uma mesma variável.

Identificador	O que poderia significar
DevIR20	Devolução do Imposto de Renda de 2020.
TotAluFem	Total de alunos do sexo feminino.
TotGeral	Total geral.
Soma	Soma de alguns valores num programa.
Fat	Fatorial de um número.
Nota1Alu	Primeira nota de um aluno.
Nota1Qui	Primeira nota em Química.
NomeCli	Nome de cliente.
Num	Um número lido.
EndCli	Endereço de cliente.
ArqCli	Arquivo de clientes.
TotVendas	Total de vendas.
TotVendDez	Total de vendas em Dezembro.
TotVend20	Total de vendas em 2020.

Tabela 1.2 - Exemplos de identificadores válidos

Identificador	O que deveria significar	Por que não é válido
20DevIR	Devolução do Imposto de Renda de 2020.	Começa com dígito
Tot AluFem	Total de alunos do sexo feminino.	Tem espaço entre as letras **t** e **A**.
Tot&Geral	Total geral.	Tem caractere especial (&).
Valor$	Valor monetário.	Tem caractere especial ($).
PreçoUnit	Preço unitário de um produto	Possui **ç** no identificador.
1ªNotaAlu	Primeira nota.	Possui letra sobrescrita.
maçã	Nome de fruta.	Possui letra acentuada e cedilha.
Next	Próximo	Comando da linguagem VB.
Read	Ler	Comando da linguagem Pascal
Resto	Resto de uma divisão	Função em pseudocódigo.
Var	Valor de uma variável	Palavra reservada de linguagem.
int	Inteiro	Tipo de dado da linguagem C.
EndereçoCli	Endereço de um cliente.	Possui **ç** no identificador.

Tabela 1.3 - Exemplos de identificadores inválidos

1.6.2 - Escopos de Variáveis

Além do identificador, do tipo de dado e do valor contido as variáveis possuem também *escopo*, que quer dizer "visibilidade" ou "abrangência" dentro de um programa. O escopo define em que partes do programa uma variável poderá ser acessada ou referenciada. Existem dois tipos de escopos: **local** e **global**. Variáveis *locais* só podem ser acessadas dentro da rotina que as definiram, ao passo que as *globais* podem ser acessadas em qualquer parte do programa. Para programas compostos de módulos, ou *subprogramas*, cada subprograma desses poderá ter

suas próprias variáveis *locais* que poderão ser acessadas somente dentro de cada um deles (este assunto será visto no **Capítulo 8**).

1.7 - Constantes

O valor de PI - razão entre o comprimento de uma circunferência e seu diâmetro - é, normalmente, usado como 3.14 (com apenas duas casas decimais). Entretanto, acredite: o PI já foi calculado com oito quatrilhões de casas decimais! Esta constante é mostrada abaixo com cinquenta decimais...
3**.**14159265358979323846264338327950288419716939937511
De qualquer forma, as constantes também representam, simbolicamente, endereços da memória RAM; entretanto, ao contrário de uma variável, o valor armazenado em uma constante não pode ser alterado durante o processamento. Isto quer dizer que ao declarar uma constante deve-se definir imediatamente seu valor, o qual permanecerá inalterável durante todo o processamento do programa. Quanto às outras características (*identificador*, *tipo de dado* e *escopo*) elas funcionam de forma semelhante às variáveis. Na prática de programação, embora não seja uma regra, os identificadores das constantes criadas são, normalmente, indicados com letras em caixa alta; todas maiúsculas; eis alguns exemplos:

- PI
- TECLA
- GRAVIDADE
- DESCONTO
- TAXA
- FATOR

Nos programas em que determinado valor permanece constante, é sempre mais interessante empregar constante, e com escopo ***global***. Por exemplo, para a constante PI:

Const PI=3**.**141592653589793 //**ANTES** das declarações das variáveis

Nota: Sempre que possível deve-se empregar variáveis *locais* nos programas. As variáveis *globais*, devido à exposição em todo o programa, estão sujeitas a interferências e acessos externos indevidos, o que pode comprometer todo o código. Além disto, em algumas linguagens de programação, as variáveis *globais* fragmentam o código executável, tornando-o maior e além do necessário, fazendo com que haja perda de performance.

1.8 - Programação *x* Codificação

Existem dezenas de linguagens de programação nas quais um mesmo programa pode ser codificado (ou, implementado, como preferem alguns). Mas, não se deve confundir Programação com Codificação; rigorosamente falando, o termo "Linguagem de Programação" deveria ser alterado para "Linguagem de Codificação".

Programar é criar uma rotina, seguindo uma lógica, para executar um trabalho qualquer. Esta definição, certamente levará o leitor a pensar que programar é criar uma receita de bolo; e é isto mesmo! Afinal, escrever um programa nada mais é do que criar uma receita para resolver um problema qualquer. O trabalho da LP é apenas a tradução das ações da rotina a serem implementadas, em códigos na sintaxe exigida pela linguagem; é claro que a Codificação é importante, mas não é a parte mais fundamental na criação do programa; pois, na hora da codificação o problema já deve estar resolvido.

❖ **Exemplo 1.5** - Ler dois números, inteiros e positivos e criar um pseudocódigo para calcular e mostrar o MDC (**M**áximo **D**ivisor **C**omum) desses dois números.

Embora seja um assunto bem popular e ensinado no Curso Fundamental, o MDC (maior número inteiro positivo que divide, simultaneamente, dois ou mais números) é muito importante em estudos de Matemática Superior, como, por exemplo, na proteção de dados através da criptografia de chave assimétrica, em matrizes, e até na computação quântica (algoritmo de Shor). Assim, baseando na sua definição formal, existem várias maneiras de calcular o MDC; vamos descrever duas, considerando os números **36** e **90**.

1 - Através dos divisores dos dois números, na seguinte sequência:

Passo 1 - Leia o primeiro número (36)
Passo 2 - Leia o segundo número (90)
Passo 3 - Mostre divisores de 36 (1, 2, 3, 4, 6, 9, 12,18 36)
Passo 4 -Mostre divisores de 90(1,2,3,5,6,9,10,15,**18,**30,45 90)
Passo 5 - Verifique qual o maior divisor comum a 36 e 90 (18)
Passo 6 - Mostre o MDC

2 - Através do "Algoritmo De Euclides" [4]

Passo 1 - Leia o primeiro número
Passo 2 - Leia o segundo número
Passo 3 - Divida o maior número pelo menor
Passo 4 - **Se**(resto da divisão for 0) **Então** MDC é o menor; **Vá** para o Passo 12, **Senão**: Siga em frente
Passo 5 - Divida o dividendo pelo resto
Passo 6 - **Se**(resto da divisão for 0) **Então** Vá para o Passo 11
Senão: Siga em frente
Passo 7 - Considere o resto como o novo dividendo
Passo 8 - Considere o divisor como o resto anterior

Passo 9 - Divida o dividendo pelo divisor do resto anterior
Passo 10 - Volte ao Passo 4
Passo 11 - Considere o penúltimo resto como o MDC

Passo 12 - Mostre o MDC

Então, para o caso de calcular o MDC(90,36) o esquema abaixo mostra como seria a aplicação do "Algoritmo de Euclides"

90	36
resto	18

Em seguida o resto 18 é transportado para o lado direito de 36 e divide-se o 36 por 18, posicionando o novo resto (0) abaixo do 18:

90	36	18
Resto	**18**	0

Repete-se este procedimento até encontrar o resto 0 (zero); o penúltimo resto obtido (anterior ao primeiro resto 0) é o MDC dos dois números dados.

Baseado na segunda forma de calcular o MDC, através no "Algoritmo de Euclides" descrito acima, a solução do **Exemplo 1.5** é dada pelo pseudocódigo a seguir, e a codificação poderia ser feita em qualquer das inúmeras linguagens e/ou ambientes de programação, mas, a base SEMPRE é o algoritmo/pseudocódigo que, realmente, é a solução do problema.

```
Programa "CalculaMDC"
//Calcula o MDC de dois números:"Algoritmo de Euclides".
//Em Pseudocódigo.
//Autor: Mário Leite.
//-----------------------------------------------------
   Declare Num1, Num2, N1, N2, Aux, MDC: inteiro
Início
   {Inicializações convenientes para entrar no loop de
   Num1 ← -1
   Num2 ← -1
   Enquanto ((Num1<=0) ou (Num2<=0)) Faça
      Escreva("Entre com o primeiro número: ")
      Leia(Num1)
      Escreva ("Entre com o segundo número: ")
      Leia(Num2)
   FimEnquanto //fim do loop de leitura dos números
   EscrevaLn("") //salta uma linha
   N1 ← Num1
   N2 ← Num2
   {Calcula o MDC com o "Algoritmo de Euclides"}
   FimEnquanto (N2<>0) Faça
      Aux ← N1
      N1  ← N2
      N2  ← (Resto(Aux/N2)
   FimEnquanto  //fim do loop de cálculo do MDC
   MDC ← N1
   EscrevaLn("MDC(",Num1,",",Num2,")"," = ", MDC)
FimPrograma
```

A implementação (codificação) do problema "Calcular o MDC de dois números" pode ser feita em várias linguagens de programação: compiladas ou interpretadas, mas seguindo uma lógica de programação para alcançar esse objetivo, baseada num único algoritmo/pseudocódigo. E, embora fossem considerados dois números bem determinados (90 e 36), poderia, também,

solicitar as entradas para quaisquer dois números inteiros. Mas, o importante é frisar que, em princípio, não importa em qual linguagem de programação o programa será codificado: o MAIS IMPORTANTE é saber qual algoritmo a ser seguido para obter o resultado esperado.

1.9 - Exercícios Propostos

1 - Explique, com suas próprias palavras, o que é “Lógica de Programação”.

2 - Por que não se deve partir direto para a codificação de um programa sem, antes, criar o algoritmo/pseudocódigo.

3 - Qual é a vantagem de se declarar um endereço de memória RAM como *constante*, em vez de *variável*? Dê um exemplo.

4 - Qual é a vantagem de fazer o pseudocódigo depois do algoritmo de um programa?.

5 - Explique por que o conceito de variável é fundamental na programação,

6 - Se você tivesse que nomear uma variável para armazenar o nome do Campeonato Mundial de Futebol da FIFA para 2024, qual dos identificadores é o mais apropriado?

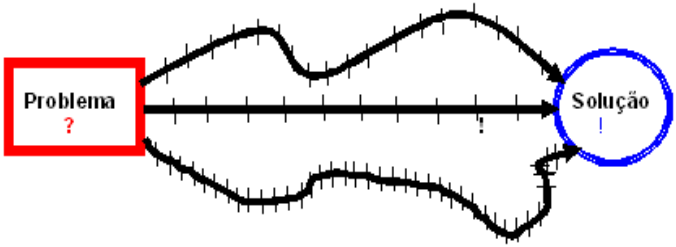
A ( ) CampeonatoMundialDeFutebol2024
B ( ) CampeonatoMundialFIFA2024
C ( ) FIFA2024
D ( ) FutebolFIFA2024
E ( ) CampeonatoFIFA2024

7 - Explique, com um exemplo, a diferença entre Algoritmo e Pseudocódigo.

8 - De acordo com as teorias de Mark A. Changizi[7], se os vulcanianos - planeta natal do Dr. Spock - tiverem três dedos em cada mão (tal como a criaturinha do filme *ET - O Extraterrestre*), então, provavelmente, o sistema de numeração deles seria o da base hexa (base 6). Considerando essa teoria, marque nas alternativas abaixo, a que aparece um número (representado na Base Decimal) que jamais poderia ser escrito na Base Vulcaniana, considerando que a Matemática vale para todo o Universo.
Explique, o porquê da sua resposta.

A ( ) 31234
B ( ) 4356
C ( ) 234100
D ( ) -1
E ( ) 0

9 - Sabemos que para realizar uma tarefa é necessário executar alguns passos (poucos ou muitos, dependendo da solução adotada). Suponha que um cliente deseja que você resolva um problema para ele, através de um programa; e, considerando o esquema abaixo, qual dos três caminhos você escolheria para solucionar esse problema?



A ( ) O caminho de cima.
B ( ) O caminho do meio.
C ( ) O caminho de baixo.
D ( ) Aquele que produzisse uma solução mais eficaz.
E ( ) Qualquer um serviria.