



# **ALGORITMOS E LÓGICA DE PROGRAMAÇÃO**

Carlinho Viana de Sousa  
Organizador

© by Carlinho Viana de Sousa, 2016.

1ª edição: 2011

3ª edição rev., atual. e ampl.: set. 2016

## **Universidade do Estado de Mato Grosso – UNEMAT**

Reitora: Profa. Dra. Ana Maria Di Renzo

### **Curso de Graduação Bacharelado em Ciência da Computação**

Campus Universitário de Alto Araguaia-MT

Departamento de Computação

Coord. do curso: Prof. Dr. Fernando Yoití Obana

---

Algoritmos e Lógica de Programação/ [organizado por] Carlinho Viana de Sousa;

coordenado pelo Departamento de Computação e pelo Curso de Graduação em Bacharelado de Ciência da Computação da Unemat – Campus Universitário de Alto Araguaia. Alto Araguaia-MT: UNEMAT, 2016.

105 p.

Inclui figuras, tabelas e quadros.

Inclui referências.

1. Algoritmos. 2. Lógica de programação. 3. Linguagens de programação. 4. Computador Simplificado. 5. Variáveis, constantes, operadores, tipos de dados, memória e processamento de dados. 6. Diagrama de Blocos. 7. Pseudocódigo. 8. Estruturas de controle condicionais e de seleção. Estruturas de controle de repetição. 9. Vetores e matrizes. 10. Introdução à linguagem de programação C. Universidade do Estado de Mato Grosso. Curso de Graduação em Bacharelado em Ciência da Computação.

---

**UNIDADE I**

<b>ALGORITMOS</b>	01
1.1 Conceitos iniciais	01
1.2 Qualidades de um bom algoritmo	02
1.3 Estratégias na construção de algoritmos	02
1.4 Visão esquemática da construção de algoritmos	03
1.5 Exemplos de algoritmos	04

**UNIDADE II**

<b>LINGUAGENS DE PROGRAMAÇÃO</b>	08
2.1 Conceitos iniciais	08
2.2 Principais linguagens de programação	09
2.3 Linguagens de programação para internet	14
2.4 Paradigmas de linguagens de programação	16
2.4.1 <i>Paradigma imperativo</i>	16
2.4.2 <i>Paradigma declarativo</i>	17
2.5 Compiladores e interpretadores	17
2.5.1 <i>Compilador</i>	17
2.5.2 <i>Interpretador</i>	18

**UNIDADE III**

<b>COMPUTADOR SIMPLIFICADO</b>	20
3.1 Conceitos iniciais	20
3.2 Estrutura de um CS	20
3.3 Funções dos elementos de um CS	21
3.4 Lista de comandos de um CS	22
3.5 Manipulação de um CS	23
3.6 Escrita de algoritmos em um CS	23
3.7 Processamento de um algoritmo em CS	26

**UNIDADE IV**

<b>VARIÁVEIS, CONSTANTES, OPERADORES, TIPOS DE DADOS, MEMÓRIA E PROCESSAMENTO DE DADOS</b>	29
4.1 Variáveis e constantes	29

4.2 Operadores .....	30
4.2.1 Operadores aritméticos ou matemáticos .....	30
4.2.2 Operadores relacionais .....	30
4.2.3 Operadores lógicos .....	31
4.2.4 Operadores de atribuição .....	32
4.2.5 Operadores de incremento/decremento .....	33
4.2.6 Operadores especiais .....	33
4.2.7 Hierarquia de operadores .....	34
4.3 Tipos de dados .....	36
4.3.1 Tipos de dados inteiros .....	36
4.3.2 Tipos de dados reais .....	36
4.3.3 Tipos de dados caracteres .....	36
4.3.4 Tipos de dados lógicos .....	36
4.3.5 Tamanho e faixa de valores para os tipos de dados .....	37
4.4 Memória e processamento de dados .....	37

## **UNIDADE V**

<b>DIAGRAMA DE BLOCOS, PSEUDOCÓDIGO E VISUALG .....</b>	<b>41</b>
5.1 Diagrama de Blocos .....	41
5.1.1 Algoritmo e processamento de um programa em DB .....	42
5.2 Pseudocódigo e VisuAlg .....	43
5.2.1 Estrutura básica de um programa escrito em VisuAlg .....	44
5.2.2 Escrita de um algoritmo/programa no ambiente VisuAlg .....	44
5.2.3 Processamento de um algoritmo/programa no ambiente VisuAlg .....	46
5.3 DB x Pseudocódigo .....	48

## **UNIDADE VI**

<b>ESTRUTURAS DE CONTROLE CONDICIONAIS E DE SELEÇÃO .....</b>	<b>51</b>
6.1 Estruturas de controle condicionais .....	51
6.1.1 Estruturas de controle condicionais simples .....	51
6.1.2 Estruturas de controle condicionais compostas .....	52
6.1.3 Estruturas de controle condicionais encadeadas .....	53
6.1.4 Estruturas de controle condicionais com operadores .e. e .ou. ....	54
6.2 Estruturas de controle condicionais representadas no DB e em Pseudocódigo .....	55
6.3 Estruturas de controle de seleção .....	61

## **UNIDADE VII**

<b>ESTRUTURAS DE CONTROLE DE REPETIÇÃO</b> .....	65
7.1 Conceitos iniciais .....	65
7.2 Estrutura de controle de repetição ENQUANTO .....	65
7.3 Estrutura de controle de repetição REPITA .....	66
7.4 Estrutura de controle de repetição PARA .....	67
7.5 Estrutura de controle de repetição no DB, Pseudocódigo e linguagem C ....	67
7.6 Laço de repetição controlado pelo usuário .....	71

## **UNIDADE VIII**

<b>ESTRUTURAS DE DADOS HOMOGÊNEA: VETORES E MATRIZES</b> .....	76
8.1 Conceitos iniciais .....	76
8.2 Vetor ou matriz unidimensional .....	76
8.2.1 <i>Aplicação prática com vetores</i> .....	77
8.2.2 <i>Leitura de dados em um vetor</i> .....	80
8.2.3 <i>Escrita de dados em um vetor</i> .....	82
8.2.4 <i>Manipulação de dados alfanuméricos em um vetor</i> .....	83
8.3 Matriz bidimensional .....	85
8.3.1 <i>Leitura de dados em uma matriz bidimensional</i> .....	86
8.3.2 <i>Escrita de dados em uma matriz bidimensional</i> .....	88

## **UNIDADE IX**

<b>INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO C</b> .....	93
9.1 Breve histórico da linguagem C .....	93
9.2 Características da linguagem C .....	93
9.3 Bibliotecas da linguagem C .....	94
9.4 Palavras reservadas da linguagem C .....	95
9.5 Sequências de escape da linguagem C .....	95
9.6 Transposição de algoritmo escrito em pseudocódigo para linguagem C ....	96
9.7 Tipos de dados na linguagem C .....	97
9.8 Tipos de operadores na linguagem C .....	98
9.8.1 <i>Operadores aritméticos ou matemáticos</i> .....	98
9.8.2 <i>Operadores relacionais</i> .....	99
9.8.3 <i>Operadores lógicos</i> .....	100

9.8.4 Operadores de atribuição .....	101
--------------------------------------	-----

<b>REFERÊNCIAS</b> .....	105
--------------------------	-----

# UNIDADE I

## ALGORITMOS

Os objetivos específicos dessa unidade são:

- ✓ Introduzir o conceito de algoritmo;
- ✓ Indicar alguns passos e estratégias para o desenvolvimento de bons algoritmos; e,
- ✓ Fornecer exemplos de algoritmos para que os alunos aprendam a resolver problemas por meio de algoritmos.

### 1.1 Conceitos iniciais

No nosso cotidiano estamos sempre resolvendo problemas, ou seja, sempre executando ações que nos levam a cumprir um objetivo para chegar a um resultado final.

Tomemos como exemplo uma situação simples que pode ocorrer no dia a dia: uma lâmpada queimada na nossa casa e que precisa ser substituída por outra. Temos então um problema que precisa ser resolvido, precisamos de ações que nos levem à solução do problema, sem nos darmos conta estamos trabalhando de forma algorítmica.

Mas o que seria um *Algoritmo*? Temos várias definições, entre elas destacamos algumas abaixo:

- ✓ Conjunto de passos bem definidos, seguindo uma seqüência lógica, com objetivo de atingir um resultado;
- ✓ A arte de escrever programas usando pseudocódigo ou uma linguagem específica de programação;
- ✓ Algoritmo não é a solução de um problema, pois, se assim fosse, cada problema teria um único algoritmo. Algoritmo é um conjunto de passos (ações) que levam à solução de um determinado problema, ou então, é um caminho para a solução de um problema e, em geral, os caminhos que levam a uma solução são muitos (LOPES; GARCIA, 2002);
- ✓ Um algoritmo, intuitivamente, é uma seqüência finita de instruções ou operações básicas (operações definidas sem ambigüidade e executáveis em tempo finito dispondo-se apenas de lápis e papel) cuja execução, em tempo finito, resolve um problema computacional qualquer que seja sua instância (SALVETI; BARBOSA, 1998, p. 5);
- ✓ [...] o termo algoritmo, do ponto de vista computacional, pode ser entendido como um conjunto de regras formais, sequenciais e bem definidas

baseadas no entendimento lógico de um problema a ser resolvido por um programador com o objetivo de transformar o problema em um programa que possa ser tratado e executado por um computador (MANZANO; OLIVEIRA, 2011, p. 32); e,

✓ Os algoritmos fazem parte do dia-dia das pessoas. As instruções para o uso de medicamentos, as indicações de como montar um aparelho, uma receita de culinária são alguns exemplos de algoritmos. Um algoritmo pode ser visto como uma sequência de ações executáveis para a obtenção de uma solução para um determinado tipo de problema (ZIVIANI, 2004, p. 1).

## 1.2 Qualidades de um bom algoritmo

Para escrevermos um bom algoritmo precisamos seguir alguns passos importantes, desde a definição do problema até a sua resolução. Um bom algoritmo deverá de acordo com Paulo (2007?) ter:

✓ **Definição perfeita** – deve descrever exatamente quais são as instruções que devem ser executadas e em que sequência. Deve ser tornado explícito o maior número possível de informações, pois a falta de alguma informação pode levar a uma interpretação errada do algoritmo;

✓ **Ausência de ambiguidade** – não deve deixar dúvidas sobre o que deve ser feito. A ambiguidade acerca do que deve ser feito também pode levar a uma interpretação errada do algoritmo;

✓ **Eficácia** – Conseguir resolver o problema em qualquer situação. Todas as situações de exceção que possam alterar o comportamento do algoritmo devem ser especificadas e tratadas; e,

✓ **Eficiência** – Resolver o problema com o mínimo de recursos. Sempre se deve buscar aquele algoritmo que, dentre os diversos algoritmos que resolvam um mesmo problema, utilize a menor quantidade de recursos. No caso de algoritmos para processamento de dados, os recursos a serem considerados são espaços de memória (principal e auxiliar) e tempo de processamento (economia de CPU), entre outros.

## 1.3 Estratégias na construção de algoritmos

Para se construir um bom algoritmo precisamos traçar algumas metas para a resolução do problema em questão. De acordo com Paulo (2007?) é preciso:

✓ Especificar o problema claramente e entendê-lo completamente;

✓ Explicitar todos os detalhes supérfluos;

✓ Entrar no problema (envolver-se totalmente com o problema);

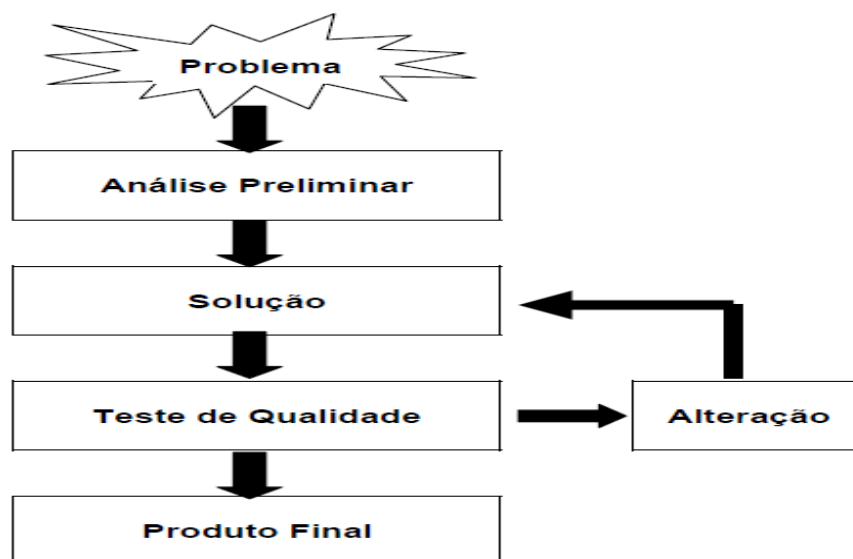
✓ Usar todas as informações disponíveis;



- ✓ Decompor o problema (**Top-Down** – Parta do problema geral até chegar aos problemas específicos); e,
- ✓ Usar o sentido inverso, se necessário (**Bottom-Up** – Parta dos problemas específicos até chegar ao geral).

#### 1.4 Visão esquemática da construção de algoritmos

Figura 1. Visão esquemática da construção de um algoritmo



Fonte: Paulo (2007?)

A figura 1 revela a visão esquemática para o desenvolvimento de algoritmos, levando-se em consideração a partir do problema, cinco fases a serem executadas. São elas:

- ✓ **Análise Preliminar** – entenda o problema com a maior precisão possível, identifique os dados; identifique os resultados desejados;
- ✓ **Solução** – desenvolva um algoritmo para resolver o problema;
- ✓ **Teste de Qualidade** – execute o algoritmo desenvolvido com dados para os quais o resultado seja conhecido. O ideal é que o universo dos dados tenha todas as combinações possíveis. Note que a qualidade de um algoritmo pode ser limitada por fatores como tempo para a sua confecção e recursos disponíveis;
- ✓ **Alteração** – se o resultado do teste de qualidade não for satisfatório, altere o algoritmo e submeta-o a um novo teste de qualidade; e,
- ✓ **Produto Final** – o algoritmo concluído e testado, pronto para ser aplicado (PAULO, 2007?).

## 1.5 Exemplos de algoritmos

Vejamos a seguir alguns exemplos de algoritmos.

### Exemplo 1 – Problema: Algoritmo para ensinar uma pessoa a cortar as unhas dos pés

Algoritmo "CORTAR\_UNHAS\_DOS\_PES"

- 1 - Início.
- 2 - Pegue o cortador de unha.
- 3 - Pegue uma bacia com água morna.
- 4 - Mergulhe os dois pés na água morna.
- 5 - Deixe os pés de molho na água morna por 5 minutos.
- 6 - Seque os pés.
- 7 - Corte as unhas.
- 8 - Fim.

### Exemplo 2 – Problema: Calcular a média de quatro notas fornecidas pelo usuário utilizando Pseudocódigo

**Algoritmo** "CALCULAR\_MEDIA\_ARTIMETICA"

**var**

N1, N2, N3, N4, MEDIA: **real**

**Início**

**escreva**("Digite a primeira nota: ")

**leia**(N1)

**escreva**("Digite a segunda nota: ")

**leia**(N2)

**escreva**("Digite a terceira nota: ")

**leia**(N3)

**escreva**("Digite a quarta nota: ")

**leia**(N4)

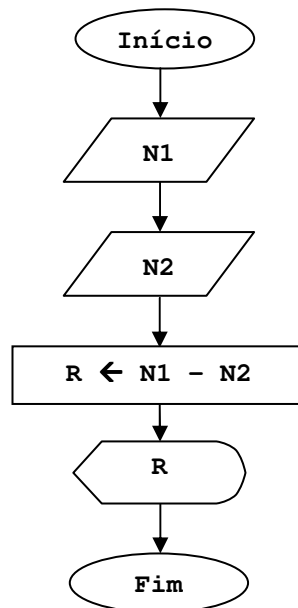
MEDIA ← (N1+N2+N3+N4)/4

**escreva**("A média das notas é igual a: ",MEDIA)

**Fimalgoritmo**

### Exemplo 3 – Problema: Realizar a subtração de dois números inteiros fornecidos pelo usuário utilizando Diagrama de Blocos (DB)

**DB "SUBTRAI\_DOIS\_NROS"**



Exemplo 4 – Problema: Realizar a soma de dois números inteiros usando o Computador Simplificado (CS)

**CS "SOMA\_DOIS\_NROS\_INTEIROS"**

E1 – Pegue um cartão na bandeja e ESCREVA o seu valor em E16.

E2 – Pegue um cartão na bandeja e ESCREVA o seu valor em E15.

E3 – Some o valor de E16 com o de E15 e ESCREVA o resultado em E14.

E4 – IMPRIMA o conteúdo de E14.

E5 – PARE.

Exemplo 5 – Problema: Realizar a divisão de dois números inteiros usando a linguagem de programação C

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
```

```
//Programa para realizar a divisão de dois números inteiros
```

```
int main(){
    int n1, n2;
    float r;

    printf("Digite o primeiro numero inteiro: ");
    scanf("%d",&n1);
    printf("Digite o segundo numero inteiro: ");
    scanf("%d",&n2);
```

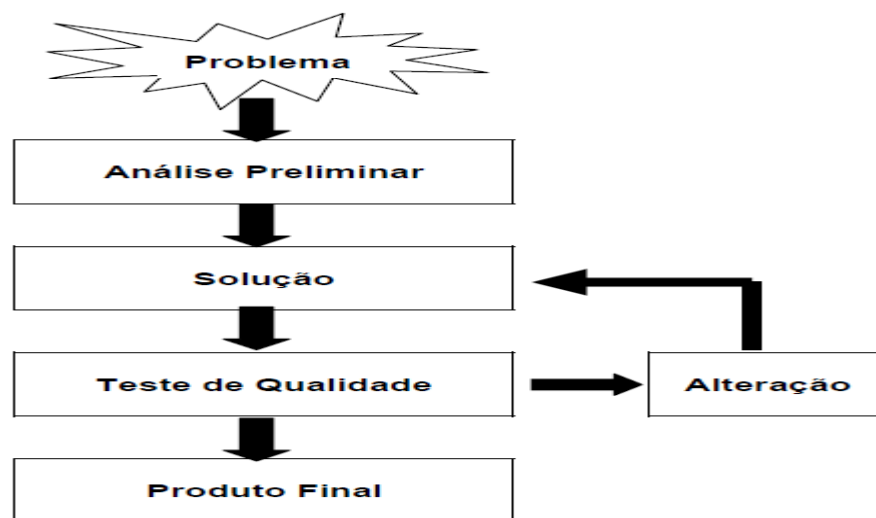
```

r = n1/n2;
printf("\nO resultado corresponde a: %.2f", r);
getche();
return(0);
}

```

### Exercícios propostos:

1. Com base no que foi estudado nessa apostila dê a sua conceituação acerca de Algoritmos.
2. Como devemos proceder na resolução de Algoritmos? Que passos devemos seguir?
3. Para termos um Algoritmo de boa qualidade, que critérios devemos obedecer?
4. Com base no que foi estudado nesta apostila, explique com suas palavras a figura abaixo:



5. Com base no Exemplo 1 demonstrado no tópico 1.5 desta apostila, resolva os problemas abaixo da mesma forma:
  - a) Construa um conjunto de passos para ensinar uma pessoa a lavar os cabelos com xampu;
  - b) Construa um conjunto de instruções para ensinar uma pessoa a fritar um ovo;
  - c) Construa um conjunto de passos para descrever a sua vinda para a universidade;

- d) Construa um conjunto de passos para ensinar uma pessoa a fazer um bolo de chocolate;
  - e) Construa um conjunto de passos que ensine uma pessoa a conquistar um namorado ou uma namorada.
6. Com base no Exemplo 3 demonstrado no tópico 1.5 desta apostila, resolva os problemas abaixo da mesma forma:
- a) Construa um conjunto de passos que realize a soma de dois números inteiros;
  - b) Construa um conjunto de passos que realize a multiplicação de dois números inteiros;
  - c) Construa um conjunto de passos que realize a divisão de dois números inteiros.
7. Problema proposto: Existe em uma margem de um rio, **três** missionários e **três** canibais. É preciso atravessar essas seis pessoas para a outra margem do rio, por meio, de uma **canoa** com capacidade para **duas pessoas**. Construa um algoritmo que:
- a) Atravesse as seis pessoas, primeiramente os missionários e depois os canibais;
  - b) Atravesse as seis pessoas, não importando a ordem, desde que em nenhuma das margens do rio o **número de canibais seja maior que o número de missionários**, pois se isso acontecer, os canibais devorarão os missionários.

**Obs.: A canoa pode ir e voltar (com uma ou duas pessoas) quantas vezes vocês acharem necessário para a resolução do algoritmo.**

## UNIDADE II

# LINGUAGENS DE PROGRAMAÇÃO

Os objetivos específicos dessa unidade são:

- ✓ Introduzir o conceito de uma linguagem de programação;
- ✓ Descrever as principais linguagens de programação;
- ✓ Mostrar exemplos de programas escritos em várias linguagens de programação; e,
- ✓ Incentivar o aluno a pesquisar.

### 2.1 Conceitos iniciais

**Linguagem de programação** é um conjunto de termos (vocabulário) e de regras (sintaxe) que permitem a formulação de instruções a serem executadas por um computador (um algoritmo ou um programa, por exemplo).

No início, foi extremamente difícil passar um programa a uma máquina porque não se concebera um meio de controlar uma barreira presente em qualquer computador: a sua limitação de só reconhecer “instruções” colocadas em sua memória sob a forma de dígitos binários (zeros e uns).

Existem três níveis de linguagens de programação, são eles:

- ✓ *Linguagem de máquina.* É a linguagem compreendida pelo computador, cujas instruções são representadas por valores binários (zeros e uns). “Utilizada a partir da década de 40 do século XX, com o surgimento do primeiro computador eletrônico (ENIAC)” (MANZANO & MANZANO, 2007, p. 163).
- ✓ *Linguagem de baixo nível.* É a linguagem que utiliza instruções próximas à compreensão da máquina. Essa linguagem exige um grande conhecimento de hardware. Exemplo: *Assembly*; e,
- ✓ *Linguagem de alto nível.* É a linguagem cujas instruções estão próximas do nível de compreensão humana (linguagem natural). Exemplo: Visual Basic, Delphi, Clipper, Java, C, entre outras.

**Software.** É o conjunto dos programas que comandam o funcionamento do hardware. Programas são feitos a partir de algoritmos – sequência de instruções/comandos para se atingir um objetivo. Depois de pronto o algoritmo é convertido para uma linguagem de programação. O produto dessa conversão é o programa.

**Programa.** É o conjunto de instruções que contém as operações necessárias para, a partir de dados inseridos obter um resultado que será disponibilizado por algum dispositivo de saída.

## 2.2 Principais linguagens de programação

**ADA.** Criada pela Honeywell Bull em 1975 para o Departamento de Defesa dos Estados Unidos. É usada para equipamentos com processadores múltiplos e grandes recursos de memória; é estruturada, aceita entrada e saída por múltiplos dispositivos e permite a execução de tarefas com controle de erros. Seu nome é homenagem àquela que foi considerada a primeira mulher programadora do mundo – Augusta Ada Byron, condessa de Lovelace.

### Exemplo de programa em ADA:

```
with Ada.Text_IO;  
  
procedure OlaMundo is  
begin  
  Ada.Text_IO.Put_Line("Olá, Mundo!");  
  Ada.Text_IO.New_Line;  
end OlaMundo;
```

**ALGOL (ALGorithmic Oriented Language – Linguagem Orientada a Algoritmos).** Voltada à expressão de algoritmos, portanto de característica científica. É pouco difundida em face do alto custo do compilador que requer.

### Exemplo de programa em ALGOL:

```
BEGIN  
FILE F (KIND=REMOTE); FILE F (tipo = REMOTE);  
EBCDIC ARRAY E [0:11]; ARRAY EBCDIC E [00:11];  
REPLACE E BY "HELLO WORLD!"; E substituir por "OLÁ MUNDO!";  
WHILE TRUE DO Enquanto verdade fazer  
  BEGIN BEGIN  
    WRITE (F, *, E); Write (F, * E);  
  END; END;  
END.
```

**APL (A Programming Language – A Linguagem de Programação).** Destinada a terminais com aplicações interativas; muito própria para operar com vetores, matrizes e funções matemáticas.

### Exemplo de programa em APL:

```
"OLÁ MUNDO!"
```

**Assembly.** É uma linguagem orientada para a máquina, cujas instruções têm geralmente uma correspondência de um-para-um com as instruções de máquina, e que pode permitir facilidades com o uso de macroinstruções. Para ser executada pelo computador necessita passar pela fase de montagem (*assembler*) através de um montador (*Assembler, Assembler Program*), que traduz suas instruções de máquina executáveis pelo computador.

Exemplo de programa em ASSEMBLY:

```
variable:
    .message    db    "Olá, Mundo!$"
code:
    mov    ah, 9
    mov    dx, offset .message
    int    0x21
    ret
```

**BASIC.** Criada por Thomas Kurtz e John Kemeny, nos EUA, em 1965, à base de um repertório de instruções simples e poderosas, com capacidade de grande desenvoltura, inclusive no tratamento de funções matemáticas. Logo tornou-se a linguagem de programação mais popular do mundo, sendo compatível (com pequenas variantes de versões) com a maioria dos microcomputadores lançados em qualquer país do mundo, ao tempo em que eles se popularizavam.

Exemplo de programa em BASIC:

```
10 PRINT "Olá, Mundo!"
20 END
```

**C.** Linguagem estruturada, criada como evolução da linguagem B (nos laboratórios Bell, EUA, 1972), que se constitui em uma ótima ferramenta para codificação de software básico. Foi idealizada por Dennis Ritchie, quando se dedicava ao trato do sistema operacional UNIX, com vistas a facilitar o trabalho de programadores experientes.

Por ser uma linguagem estruturada, facilita a programação de qualquer nível de complexidade, desde as que envolvem pequenos cálculos aritméticos até as voltadas ao desenvolvimento de complexos sistemas operacionais.

Sua versão avançada é o **C++**, que é uma linguagem de programação orientada a objetos, isto é, os programas são compostos por módulos (objetos) autossuficientes.

Exemplo de programa em C:

```
#include <stdio.h>
```

```
int main(){
```



```
printf("\nOla Mundo!\n");
getche();
return(0);
}
```

**CLIPPER.** Voltada à gerência de arquivos em microcomputadores, é a continuação da família dBase, da qual surgiu como versão compilada e evoluída. O **dBase** foi o gerenciador de arquivos que mais se popularizou a partir da versão dBase II lançada em 1981 para os micros CP/M de 8 bits e para os PCs de 16 bits. A versão dBase IV incluiu um SQL (*Structured Query Language*) para consulta a banco de dados.

Exemplo de programa em Clipper:

```
? "Alô Mundo!"
```

**COBOL** (**CO**mmon **B**usiness **O**riented **L**anguage – Linguagem Orientada a Negócios). Resultado de um esforço para estabelecer uma linguagem padrão de programação no processamento comercial, usuários e fabricantes de computadores (CODASYL) lançaram sua primeira versão em agosto de 1961.

Exemplo de programa em COBOL:

```
* Hello World Program
* GPL Copyleft Jonathan Riddell 2001

IDENTIFICATION DIVISION.
PROGRAM-ID.      hello.
ENVIRONMENT DIVISION.
DATA DIVISION.

PROCEDURE DIVISION.
    DISPLAY "hello ," WITH NO ADVANCING
    DISPLAY "world!"
    STOP RUN.
```

**DELPHI.** Para concorrer com o Visual Basic da Microsoft, a Borland International, Inc. criou a linguagem Delphi – baseada no Pascal. É uma linguagem de programação orientada a objetos (OOP), oferecendo aos programadores a oportunidade de utilizar/incrementar bibliotecas de objetos reutilizáveis.

O ambiente de desenvolvimento Delphi permite que se trabalhe em plataformas providas de sistemas operacionais Windows desde o 3.11 até a versão mais atual, com necessidade mínima de codificação.

Possui um compilador capaz de gerar código diretamente executável pelo Windows, proporcionando velocidade 5 a 20 vezes maior do que as linguagens

interpretadas. Além disso, traz também um gerenciador de banco de dados, um gerador de relatórios que é de fácil aprendizado. Tudo isso contribuiu para que se tornasse uma linguagem de extraordinário sucesso em todo mundo.

#### Exemplo de programa em DELPHI:

```
Program Ola_Mundo;  
  
{$APPTYPE CONSOLE}  
  
begin  
    WriteLn('Olá, Mundo!');  
end.
```

**FORTRAN** (**FOR**mula **TRAN**slation – Tradução de Fórmula). Desenvolvida na década de 50, pela IBM, com objetivo de atender às necessidades de tratamento de cálculos sobre fórmulas matemáticas, foi, durante muito tempo, uma linguagem de elevada importância. Cedeu lugar a linguagens mais simples e estruturadas.

#### Exemplo de programa em FORTRAN:

```
PROGRAM HELLO  
WRITE(*,10)  
10 FORMAT('Olá, Mundo!')  
STOP  
END
```

**JAVA.** É uma linguagem de programação orientada a objetos, independente de plataforma, desenvolvida pela Sun Microsystems, similar ao C++, porém bem mais simples, recursos tediosos deixaram de ser considerados em seu escopo.

Pequenos programas em Java, projetados para distribuição na Internet, denominado **applets**, são executados na janela do navegador e ampliam a funcionalidade de páginas na Web.

Programas maiores e independentes, ao contrário dos applets, são executados na própria janela do usuário e tem acesso integral ao sistema de arquivos do computador. Para executar uma aplicação em Java, o equipamento deve ser dotado de interpretador. Antecedendo à ação do interpretador, o programa em Java é submetido a uma espécie de compilação, cujo resultado é a transformação do código em código de byte (**byte-code**). Esse código, capaz de ser transmitido pelas redes como a Internet, é justamente o que torna a linguagem independente de plataformas.

#### Exemplo de programa em JAVA:

```
public class Hello {
```

```

    public static void main(String[] args) {
        System.out.println("Olá, Mundo!");
    }
}

```

**LISP (LISt Processing Language – Linguagem para Processamento de Listas).** Voltada prioritariamente à tradução e à documentação automática de textos. Foi desenvolvida no início da década de 60, no Instituto de Tecnologia de Massachusetts (MIT), por John McCarthy, como ferramenta para estudo da inteligência artificial e simulação do pensamento. Foi utilizada na implementação de alguns softwares de apoio.

*Exemplo de um programa em LISP:*

```

(Defun OLÁ ()
  "HELLO WORLD" "OLÁ MUNDO"
) )

```

**LOGO.** Especificamente desenvolvida para o meio educacional, foi inicialmente usada em grandes computadores; em seguida, foi implementada em microcomputadores. Presta-se, prioritariamente, para introduzir o computador como instrumento de desenvolvimento intelectual, desde a pré-escola até o ensino universitário. Foi desenvolvida no MIT, com recursos de LISP, no final da década de 60, pelo prof. Seymour Papert.

*Exemplo de programa em LOGO:*

```

print [Olá, Mundo!]

```

**PASCAL.** Desenvolvida em Zurique em 1971, por Nicklaus Whirth. É estruturada e de uso geral: aplicações comerciais e científicas. Devido a características de recursividade, facilidade de estruturação de algoritmos e liberdade de formas de E/S, além de incorporação das técnicas de programação estruturada, o PASCAL substituiu o FORTRAN nas universidades e nas aplicações científicas. Sua versão interpretada é mais difundida do que a versão compilada, por requerer menor espaço de memória e ser compatível com qualquer microcomputador.

*Exemplo de programa em PASCAL:*

```

program OlaMundo;
uses crt;
begin
  writeln('Olá mundo!');
end.

```

**PL1 (Programming Language One – Linguagem Única de Programação).** Representou, em certa época, o esforço da IBM no sentido de propiciar uma linguagem orientada a problemas de qualquer natureza, a partir dos recursos do ALGOL, do FORTRAN e do COBOL.

Exemplo de programa em PL1:

```
OLÁ: OPÇÕES DE PROCEDIMENTO (principal);

      / * Um programa para a saída * OLÁ MUNDO /
      FLAG = 0;

LOOP: DO WHILE (FLAG = 0);
      PUT ignorar os dados ("OLÁ MUNDO! ');
      END LOOP;
FIM OLÁ;
```

**SNOBOL (StrinNg Oriented SymBOLic Language – Linguagem Orientada para Cadeia Simbólica).** Foi, por algum tempo, a linguagem mais utilizada na criação de compiladores e editores de texto. Voltada ao processamento de cadeias de caracteres e de expressões matemáticas.

Exemplo de programa em SNOBOL:

```
* PROGRAMA PARA EXIBIR MENSAGEM "OLÁ MUNDO"

OUTPUT = 'Olá Mundo!'
END
```

**VISUAL BASIC (VB).** O VB foi lançado pela Microsoft em 1991, constituindo-se na primeira linguagem com características gráficas do Windows aplicadas aos ambientes de programação. Por meio dela, portanto, as notáveis facilidades da recém-chegada GUI (Graphical User Interface) passaram a ser aproveitadas efetivamente no desenvolvimento de aplicações.

Exemplo de programa em VISUAL BASIC:

```
Private Sub Form_Load()
    Print "Olá, Mundo!"
End Sub
```

## 2.3 Linguagens de programação para internet

**ASP. (Active Server Pages – Páginas de Servidor Ativas).** É um ambiente para programação por scripts no servidor, que pode ser usado para criar páginas dinâmicas, interativas e de alta performance. As páginas ASP rodam no servidor

e não no cliente. É o próprio servidor que transforma os scripts em HTML padrão, fazendo com que qualquer browser do mercado seja capaz de acessar um site que usa ASP.

#### Exemplo de programa em ASP:

```
<% Response.Write ("Olá Mundo!") %>
```

**HTML (Hyper Text Markup Language).** É uma linguagem declarativa utilizada para criação de páginas e exibição de textos. Permite que documentos acessados por programas na Web tenham suas partes mostradas em formatos específicos. A HTML é padronizada por uma DTD (Document Type Definition), pelo W3C (World Wide Web Consortium), órgão composto por pesquisadores e profissionais, que se dedicam a estabelecer padrões para a rede.

#### Exemplo de programa em HTML:

```
<HTML>
<HEAD>
  <TITLE>Olá mundo!</TITLE>
</HEAD>
<BODY>
  <P><B>Olá mundo!</B></P>
</BODY>
</HTML>
```

**JAVASCRIPT.** É uma linguagem de criação de scripts desenvolvida pela Netscape Communications, voltada para editoração na Web. Permite que programadores incorporem instruções simples de programação, semelhantes à da linguagem Java, no texto HTML de suas páginas.

#### Exemplo de programa em JAVASCRIPT:

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write ("Olá Mundo <H1> </ H1>");
</SCRIPT>
</BODY>
</HTML>
```

**PHP.** É o acrônimo de Hypertext Preprocessor (pré-processador de hipertexto), uma poderosa linguagem de programação open source, amplamente utilizada, principalmente no contexto da web. Uma das principais características do PHP é

a sua capacidade de se integrar ao HTML, facilitando o desenvolvimento de páginas dinâmicas.

#### Exemplo de programa em PHP:

```
<HTML>
<BODY>
<?PHP
ECHO "Olá Mundo!";
?>
</BODY>
</HTML>
```

## 2.4 Paradigmas de linguagens de programação

O termo paradigma foi introduzido por Kuhn (1962), designando o conjunto de crenças e preconceitos que permearia a atividade dos membros de um corpo científico em certa época. Em suma os paradigmas são um conjunto de regras que estabelecem fronteiras (domínios) e descrevem como resolver os problemas dentro dessas fronteiras.

Em programação, os paradigmas começaram a ser introduzidos e chamados como tais a partir de 1975. Nessa época já era evidente a existência de um paradigma básico de programação, o **imperativo**, que era representado por linguagens como FORTRAN, ALGOL e PASCAL. Em contraposição, LISP e linguagens derivadas davam suporte a outro paradigma, o **funcional**, cujos programas eram expressos através de composições de funções. Na década de 80 surgiu um novo paradigma, o **lógico**, baseado no reconhecimento de PROLOG e outras linguagens baseadas em lógica. Também nesta época surgiram as linguagens orientadas a objeto, que constituem o paradigma em questão.

### 2.4.1 Paradigma imperativo

Linguagens que expressam sequências de operações necessárias para os cálculos; são orientadas a comando. Dividem-se em duas subclasses:

- ✓ *Procedimental*: construções para modularizar o código na forma de procedimentos e funções; e,
- ✓ *Orientada a objeto*: computação é vista como a interação entre objetos de dados ativos; todos dados são objetos e esses são tratados com uniformidade e todo processamento é realizado pela passagem de mensagens entre objetos; cada objeto incorpora operações definidas para ele e o controle tem a forma de mensagens enviadas a objetos de dados para transformá-los; objetos similares podem ser agrupados em uma hierarquia de classes, com classes capazes de herdarem atributos de suas superclasses.

### 2.4.2 Paradigma declarativo

Dão ênfase ao que calcular, ao invés de como calcular; um programa declarativo é um *statement* (declaração) que se concentra na solução. Dividem-se em duas subclasses:

- *Lógico*: construções para definir relações atômicas entre valores através de regras, na forma de implicações na quais uma conclusão, no máximo, deriva da conjunção de zero ou mais condições; os programas são interrogados para elucidar verdades sobre indivíduos e relações; e,
- *Funcional* (aplicativo): especificam a saída como uma função; são orientadas a valor.

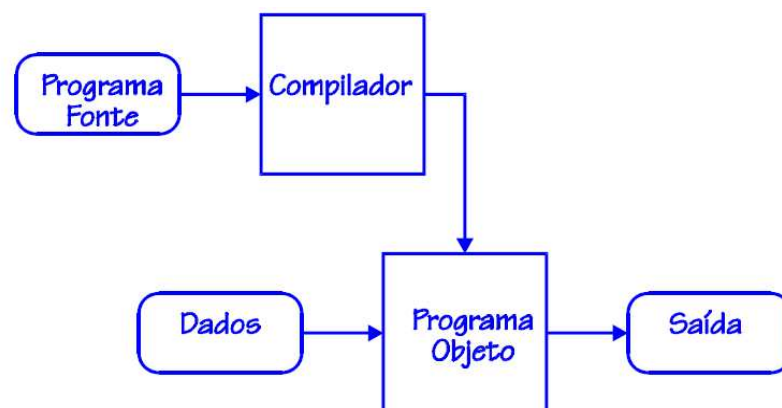
## 2.5 Compiladores e interpretadores

Compiladores e interpretadores são programas que realizam a tradução de um código, geralmente escrito em alto nível para baixo nível (linguagem de máquina), ou melhor, codificam uma linguagem abstrata para uma linguagem binária.

### 2.5.1 Compilador

Um compilador (figura 2) tem a função de converter um código-fonte em um código-objeto, ou seja, traduz as informações do programa escrito em uma dada linguagem de programação para uma linguagem que a máquina possa entender e executar. Ao realizar essa tradução, o código-fonte se torna um código-objeto (escrito em linguagem de alto nível compatível com o processador utilizado), em seguida faz a ligação do código-objeto com as rotinas de execução de programas do sistema operacional, tornando o código-fonte um programa executável (MANZANO; OLIVEIRA, 2011).

**Figura 2. Compilador**

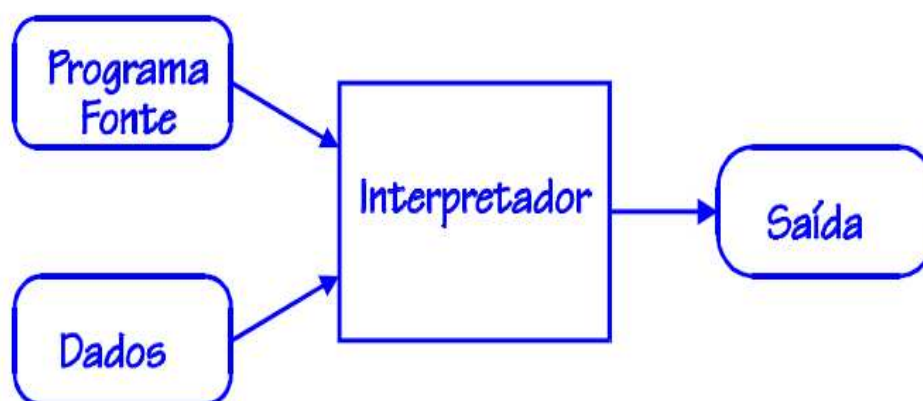


Fonte: images.google.com

### 2.5.2 Interpretador

Diferentemente do compilador o interpretador (figura 3) executa “[...] um programa-fonte escrito em uma linguagem na própria memória principal do computador, sem que o programa seja executado no processador central da máquina” (MANZANO; OLIVEIRA, 2011, p. 28). Os programas são executados de forma mais rápida, no entanto é possível ter acesso ao código-fonte já que não é gerado um código-objeto.

**Figura 3. Interpretador**



Fonte: images.google.com

No quadro 1 são mostradas as principais vantagens e desvantagens de cada um.

**Quadro 1. Compiladores x Interpretadores**

	<b>Vantagens</b>	<b>Desvantagens</b>
<b>Compiladores</b>	<ul style="list-style-type: none"><li>✓ Execução mais rápida;</li><li>✓ Permite estruturas de programação mais completas;</li><li>✓ Permite otimização do código-fonte.</li></ul>	<ul style="list-style-type: none"><li>✓ Várias etapas de tradução;</li><li>✓ Programa final é maior;</li><li>✓ Necessita mais memória para sua execução.</li></ul>
<b>Interpretadores</b>	<ul style="list-style-type: none"><li>✓ Depuração do programa é mais simples;</li><li>✓ Consome menos memória;</li><li>✓ Resultado imediato do programa ou rotina.</li></ul>	<ul style="list-style-type: none"><li>✓ Execução do programa é mais lenta;</li><li>✓ Estrutura de dados demasiado simples;</li><li>✓ Necessário fornecer o programa-fonte ao utilizador.</li></ul>

Fonte: organizado pelo autor



### **Exercícios propostos:**

1. Conceitue o que é uma linguagem de programação e qual a sua utilidade para os dispositivos eletrônicos.
2. Explique os três níveis das linguagens de programação.
3. Pesquise e escreva sobre três linguagens de programação que não constem no estudo desta unidade.
4. O que é um paradigma de linguagem de programação? Explique e dê exemplos.
5. Explique a diferença entre compilador e interpretador. Dê pelo menos um exemplo de uma linguagem compilada e outra interpretada.
6. Qual a diferença entre algoritmo e programa?
7. A partir do conhecimento adquirido sobre as linguagens de programação, pesquise e traga para a sala de aula exemplos de programas para realizar a divisão de dois números inteiros escritos nas linguagens *Pascal* e *C*.

## UNIDADE III

### COMPUTADOR SIMPLIFICADO

Os objetivos específicos dessa unidade são:

- ✓ Apresentar ao aluno uma visão geral dos elementos e do funcionamento de um computador;
- ✓ Apresentar o conceito de um programa e o seus procedimentos de execução; e,
- ✓ Introduzir o aluno no raciocínio lógico e algorítmico por meio do conceito de Computador Simplificado.

#### 3.1 Conceitos iniciais

Um **Computador Simplificado** (CS) é uma máquina hipotética que simula um computador real, ou melhor, simula os processos de entrada, armazenamento, processamento e saída de dados executados em um computador.

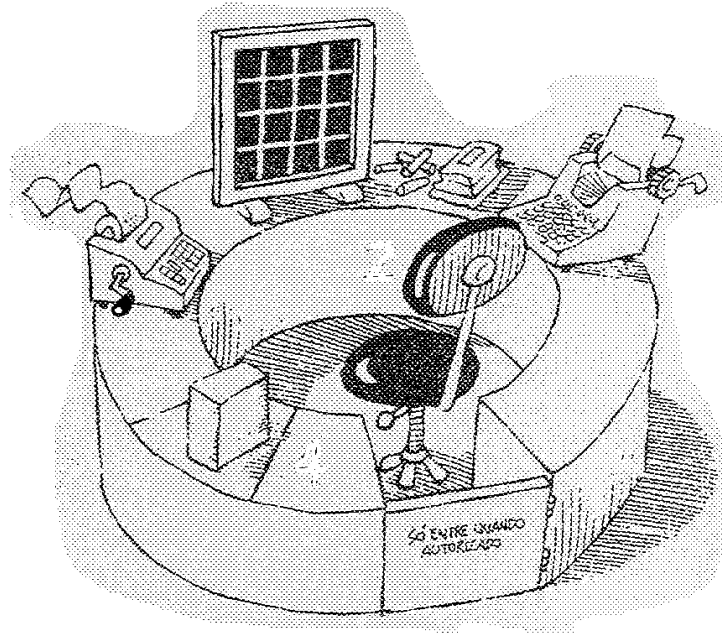
Um CS trabalha com locais que podem armazenar instruções, valores (numéricos e alfanuméricos), realizar cálculos, etc. Tais locais são conhecidos como escaninhos. "Cada escaninho tem uma identificação, formada pela letra E, seguida de um número sequencial. Assim, E1 é uma referência ao escaninho 1 e E16 ao escaninho 16" (GUIMARÃES; LAGES, 1992, p. 28).

#### 3.2 Estrutura de um CS

A estrutura de um CS consiste nos elementos apresentados abaixo:

- ✓ um conjunto de 16 (dezesesseis) escaninhos desenhados em um quadro negro ou lousa branca;
- ✓ uma cadeira onde se sentará o operador (pessoa que vai executar as instruções);
- ✓ giz e apagador;
- ✓ uma bandeja com N cartões empilhados com números escritos neles;
- ✓ uma calculadora para realizar somas, subtrações, multiplicações, divisões, etc.; e,
- ✓ uma máquina de escrever para imprimir os resultados.

**Figura 4. Representação de um CS**



Fonte: Guimarães & Lages (1992)

A figura 4 apresenta a estrutura de um CS com cinco elementos descritos acima, tais elementos podem simular um computador real como conhecemos atualmente. A estrutura do CS apresenta um giz (para escrever as instruções) e apagador (para apagar as instruções) nos escaninhos. Os escaninhos podem ainda ser representados por uma tabela com 4 linhas e 4 colunas desenhadas em qualquer editor de texto ou no caderno como representada no quadro 2.

**Quadro 2. Representação de um escaninho 4x4**

E1	E2	E3	E4
E5	E6	E7	E8
E9	E10	E11	E12
E13	E14	E15	E16

Fonte: Elaborado pelo autor

### 3.3 Funções dos elementos de um CS

**Função dos escaninhos:** neles são armazenadas todas as informações, ou seja, as instruções e os números contidos nos cartões dispostos na bandeja.

**Funções do operador:** o operador deverá ler as instruções escritas nos escaninhos e utilizar os elementos do CS para realizar os cálculos. Todas as instruções que o operador deverá executar precisam estar previamente escritas nos escaninhos.

**Funções do giz e apagador:** o giz serve para escrever as instruções nos escaninhos para o operador executar. É utilizado também para o operador escrever os números nos escaninhos. Caso o operador precise utilizar um escaninho no qual já existe um número armazenado, basta utilizar o apagador para apagar tal número e o giz para escrever outro número em seu lugar.

**Função da bandeja de cartões:** é utilizada para colocar os cartões empilhados, nesses cartões só existem escritos números que devem ser armazenados nos escaninhos, um por vez. Depois de lido o cartão ele fica virado na bandeja ao lado da pilha original.

**Função da máquina de calcular:** é utilizada para realizar todos os cálculos (soma, subtração, raiz quadrada, etc.) contidos nas instruções armazenadas nos escaninhos.

**Função da máquina de escrever:** é utilizada para imprimir os resultados numéricos ou alfanuméricos. Caso dê um erro no programa, o operador deverá datilografar a palavra ERRO na máquina de escrever e finalizar o programa.

### 3.4 Lista de comandos de um CS

O CS necessita de alguns comandos para executar o programa escrito, os principais são apresentados, por ordem alfabética, no quadro 3 a seguir.

Quadro 3. Lista de comandos de um CS

Comando	Função	Exemplo
AVANCE_PARA	Deslocar de um escaninho para outro	E8 - <b>AVANCE_PARA</b> E10
ENQUANTO	Repetir uma quantidade finita de ações nos escaninhos	E5 - <b>ENQUANTO</b> houver mais cartões na bandeja <b>RETORNE_PARA</b> E2
ESCREVA	Armazenar um valor dentro de um escaninho	E2 - Pegue um cartão na bandeja e <b>ESCREVA</b> o seu valor em E16
IMPRIMA	Imprimir os resultados dos escaninhos	E7 - <b>IMPRIMA</b> o valor de E16
PARE	Finalizar o algoritmo/programa	E8 - <b>PARE</b>

RETORNE_PARA	Voltar de um escaninho para outro	E10 - <b>RETORNE_PARA</b> E2
SE	Realizar um teste sobre um valor VERDADEIRO	E3 - <b>SE</b> o valor de E16 for maior que zero, <b>RETORNE_PARA</b> E1
SENAO	Realizar um teste sobre um valor FALSO	E4 - <b>SE</b> o valor de E16 for maior que zero, <b>RETORNE_PARA</b> E1, <b>SENAO</b> <b>AVANCE_PARA</b> E8

Fonte: Elaborado pelo autor

*Nota: Todos os comandos de um CS devem ser escritos em letras maiúsculas – caixa alta (Exemplos: PARE, AVANCE\_PARA, RETORNE\_PARA, etc.).*

### 3.5 Manipulação de um CS

Antes de escrevermos algoritmos em um CS é importante sabermos como manipular os cartões e os escaninhos (quadro 1). A seguir apresentaremos alguns passos para escrever um algoritmo em um CS, são eles:

- ✓ todo algoritmo escrito em CS precisa ter um nome antecedido da sigla CS (Exemplo: **CS Soma**);
- ✓ todas as instruções escritas nos escaninhos devem ser armazenadas do início para o final, ou seja, de E1 a E16 (Exemplo: E1 - Pegue um cartão na bandeja e ESCRIVA o seu valor em E15);
- ✓ todos os valores contidos nos cartões bem como os processamentos (cálculos, por exemplo) são armazenados do final para o começo, ou seja, de E16 a E1 (Exemplo: E5 - Pegue o valor de E15 e some com o de E16 e ESCRIVA o resultado em E16);
- ✓ somente poderá ser lida uma instrução de cada vez contida nos cartões; e,
- ✓ não é possível utilizar mais instruções/dados/cálculos superior à quantidade de espaços no escaninho, caso todo espaço for utilizado aumenta a capacidade do escaninho (Exemplo: 5x5 – 25 espaços para armazenamento).

### 3.6 Escrita de algoritmos em um CS

Sabemos que um algoritmo é um problema a ser resolvido passo a passo até que tenhamos um resultado final, pois bem, isso é possível de ser realizado em um CS, pois temos instruções contidas em cartões as quais podem ser armazenadas e processadas nos escaninhos pelo operador. De que forma isto pode ser feito? Observem abaixo alguns exemplos que demonstram o algoritmo em sua representação textual seguido de sua representação gráfica (nos escaninhos).

**Algoritmo 1:** Escreva um algoritmo em CS que imprima a mensagem: "SEJAM TODOS BEM VINDOS AO SEMESTRE QUE SE INICIA!".

**CS "Saudacao"**

E1 - ESCREVA a mensagem: "SEJAM TODOS BEM VINDOS AO SEMESTRE QUE SE INICIA!" em E16.  
E2 - IMPRIMA o valor de E16.  
E3 - PARE.

<b>E1</b> ESCREVA a mensagem: "SEJAM TODOS BEM VINDOS AO SEMESTRE QUE SE INICIA!" em E16.	<b>E2</b> IMPRIMA o valor de E16.	<b>E3</b> PARE.	<b>E4</b>
<b>E5</b>	<b>E6</b>	<b>E7</b>	<b>E8</b>
<b>E9</b>	<b>E10</b>	<b>E11</b>	<b>E12</b>
<b>E13</b>	<b>E14</b>	<b>E15</b>	<b>E16</b> "SEJAM TODOS BEM VINDOS AO SEMESTRE QUE SE INICIA!"

**Algoritmo 2:** Escreva um algoritmo em CS que realize a soma de dois números.

**CS "SomaDoisNumeros"**

E1 - Pegue um cartão na bandeja e ESCREVA o seu valor em E16.  
E2 - Pegue um cartão na bandeja e ESCREVA o seu valor em E15.  
E3 - Some o valor de E16 mais o valor de E15 e ESCREVA o resultado em E14.  
E4 - IMPRIMA o valor de E14.  
E5 - PARE.

<b>E1</b> Pegue um cartão na bandeja e ESCREVA o seu valor em E16.	<b>E2</b> Pegue um cartão na bandeja e ESCREVA o seu valor em E15.	<b>E3</b> Some o valor de E16 mais o valor de E15 e escreva o resultado em E14.	<b>E4</b> IMPRIMA o valor de E14.
<b>E5</b> PARE.	<b>E6</b>	<b>E7</b>	<b>E8</b>
<b>E9</b>	<b>E10</b>	<b>E11</b>	<b>E12</b>
<b>E13</b>	<b>E14</b>	<b>E15</b>	<b>E16</b>

**Algoritmo 3:** Escreva um algoritmo em CS que realize somente a soma de dois números positivos, caso um dos números seja negativo finalize o programa.

**CS "SomaDoisNrosPositivos"**

E1 - Pegue um cartão na bandeja e ESCREVA o seu valor em E16.  
 E2 - SE o valor de E16 for menor que zero, AVANCE\_PARA E7.  
 E3 - Pegue um cartão na bandeja e ESCREVA o seu valor em E15.  
 E4 - SE o valor de E15 for menor que zero, AVANCE\_PARA E7.  
 E5 - Some o valor de E16 mais o valor de E15 e ESCREVA o resultado em E14.  
 E6 - IMPRIMA o valor de E14.  
 E7 - PARE.

<b>E1</b> Pegue um cartão na bandeja e ESCREVA o seu valor em E16.	<b>E2</b> SE o valor de E16 for menor que zero, AVANCE_PARA E7.	<b>E3</b> Pegue um cartão na bandeja e ESCREVA o seu valor em E15.	<b>E4</b> SE o valor de E15 for menor que zero, AVANCE_PARA E7.
<b>E5</b> Some o valor de E16 mais o valor de E15 e ESCREVA o resultado em E14.	<b>E6</b> IMPRIMA o valor de E14.	<b>E7</b> PARE.	<b>E8</b>
<b>E9</b>	<b>E10</b>	<b>E11</b>	<b>E12</b>
<b>E13</b>	<b>E14</b>	<b>E15</b>	<b>E16</b>

**Algoritmo 4:** Escreva um algoritmo em CS que realize a impressão de N cartões e finalize o programa quando for encontrado um valor zero em qualquer um dos cartões.

**CS "Imprime\_N\_Cartoes"**

E1 - Pegue um cartão na bandeja e ESCREVA o seu valor em E16.  
 E2 - SE o valor de E16 for zero, AVANCE\_PARA E5.  
 E3 - IMPRIMA o valor de E16.  
 E4 - ENQUANTO houver cartões na bandeja, RETORNE\_PARA E1.  
 E5 - PARE.

**Desafio:** Quantos cartões serão impressos no algoritmo escrito acima? Faça a representação gráfica do algoritmo nos escaninhos.

### 3.7 Processamento de algoritmo em um CS

Para representarmos um algoritmo processado em CS, propomos como problema a ser resolvido o cálculo da média aritmética de quatro notas escritas nos cartões. Vejamos abaixo a representação textual do algoritmo em CS:

#### CS "CalculoMediaAritmetica"

E1 - Pegue um cartão na bandeja e ESCRIVA o seu valor em E16.  
E2 - Pegue um cartão na bandeja e ESCRIVA o seu valor em E15.  
E3 - Pegue um cartão na bandeja e ESCRIVA o seu valor em E14.  
E4 - Pegue um cartão na bandeja e ESCRIVA o seu valor em E13.  
E5 - Some o valor de E16 mais os valores de E15, E14 e E13 e ESCRIVA o resultado em E12.  
E6 - Pegue o valor de E12 e divida por quatro, e ESCRIVA o resultado em E11.  
E7 - IMPRIMA o valor de E11.  
E8 - PARE.

O processamento do algoritmo (também conhecido como "teste de mesa") escrito acima será representado no quadro 4. Para realizarmos o processamento do algoritmo vamos considerar os valores: **7, 8, 9 e 10** para serem armazenados, respectivamente nos escaninhos **E16, E15, E14 e E13**.

**Quadro 4. Processamento do algoritmo CS CalculoMediaAritmetica**

Passo	E11	E12	E13	E14	E15	E16
1	...	...	...	...	...	7
2	...	...	...	...	8	7
3	...	...	...	9	8	7
4	...	...	10	9	8	7
5	...	34	10	9	8	7
6	8,5	34	10	9	8	7
7	<b>8,5</b>	34	10	9	8	7
8	<b>8,5</b>	34	10	9	8	7

*Fonte:* Elaborado pelo autor

Analisando o quadro 4 acima temos:

Passo 1: leitura e armazenamento do primeiro valor do cartão em E16 (**7**).

Passo 2: leitura e armazenamento do segundo valor do cartão em E15 (**8**).

Passo 3: leitura e armazenamento do terceiro valor do cartão em E14 (**9**).

Passo 4: leitura e armazenamento do quarto valor do cartão em E13 (**10**).

Passo 5: soma dos valores armazenados nos escaninhos E16, E15, E14 e E13 (**7+8+9+10**). Armazenamento do valor da soma em E12 (**34**).

Passo 6: Divisão do valor de E12 por 4 e armazenamento do resultado em E11 (**8,5**).



Passo 7: Impressão do valor armazenado em E11 (**8,5**) – deve ser destacado (em negrito ou circulado) o valor a ser impresso do resultado no processamento.

Passo 8: Fim do programa/algoritmo.

### Exercícios propostos:

1. O que é um Computador Simplificado (CS) e quais os seus principais componentes?
2. Compare cada componente do CS com os componentes de um computador pessoal (PC). Explique as comparações.
3. Quais os principais comandos de um CS? Dê exemplos.
4. Quais as regras para se escrever um algoritmo ou programa em CS?
5. Escreva um algoritmo em CS que realize a multiplicação de dois números. Mostre ao final o processamento do algoritmo.
6. Escreva um algoritmo em CS que realize a divisão de dois números. Mostre ao final o processamento do algoritmo.
7. Descubra os erros do algoritmo abaixo e reescreva-o corretamente realizando o seu processamento:

E1 – Pegue um cartão na bandeja e escreva o seu valor em E16.

E2 – Pegue um cartão na bandeja e escreva o seu valor em E15.

E3 – Pegue um cartão na bandeja e escreva o seu valor em E14.

E4 – Subtraia o valor de E16 com o de E15 e escreva o resultado em E16.

E5 – Some o valor de E16 com o de E14 e escreva o resultado em E16.

E6 – Imprima o valor de E16.

8. Escreva um algoritmo em CS que realize a média ponderada de três números inteiros (lidos de três cartões) e armazenados em **E16**, **E15** e **E14**. Os pesos correspondem respectivamente a: **2**, **3**, **5** para os valores armazenados em E16, E15 e E14. Mostre ao final o processamento do algoritmo.
9. Escreva um algoritmo em CS que realize a média aritmética de dez números inteiros.
10. Sabemos que para calcular o fatorial de um número inteiro e positivo devemos:

- a. Diminuir um do número inicial (**n-1**) e multiplicar pelo número inicial (**n**), ou seja, **n\*(n-1)**;

b. Realizar as multiplicações até que o número inicial chegue em **1**, como demonstrado abaixo:

Por exemplo:  **$n = 5$** ,  $n! = n*(n-1)*(n-2)*(n-3)*(n-4)$ , ou seja,  $5! = 5*(5-1)*(5-2)*(5-3)*(5-4) = 5*4*3*2*1 = \mathbf{120}$ .

Ante o exposto escreva um algoritmo em CS que realize o cálculo do fatorial de um número inteiro e positivo. Mostre ao final o processamento do algoritmo.

11. Escreva um algoritmo em CS que a partir de uma pilha de dez cartões imprima somente os números positivos.

12. Escreva um algoritmo em CS que realize o somatório de N cartões de uma bandeja.

## UNIDADE IV

# VARIÁVEIS, CONSTANTES, OPERADORES, TIPOS DE DADOS, MEMÓRIA E PROCESSAMENTO DE DADOS

Os objetivos específicos dessa unidade são:

- ✓ Dotar o aluno de conhecimentos sobre variáveis, constantes, operadores e tipos de dados;
- ✓ Mostrar ao aluno como o computador trabalha com expressões numéricas;
- ✓ Mostrar ao aluno como o computador trabalha com expressões lógicas; e,
- ✓ Explicar o conceito de memória bem como o processamento de dados em um computador.

### 4.1 Variáveis e constantes

*Variáveis* são **dados cujos valores variam** durante a execução do programa. Uma variável pode ser definida como um local na memória do computador identificada por um **nome** pronta para receber um **valor qualquer**. Esse valor pode ser alterado durante a execução do programa.

Toda variável possui um **nome**, um **tipo**, um **tamanho** e um **valor**.

*Constantes* também são locais designados na memória do computador identificado por um **nome**. Ao contrário das variáveis os dados em uma constante permanecem com seus **valores inalterados** durante a execução do programa.

Para se criar uma variável ou constante devemos seguir algumas regras, as quais são apresentadas abaixo:

- ✓ Toda variável deverá ser identificada por um NOME que pode ser escrito em maiúsculo ou minúsculo;
- ✓ Uma variável **nunca** poderá iniciar o seu nome com um **número** ou símbolo especial. O primeiro caractere (símbolo para representar uma informação computacional na tela) de uma variável sempre será uma letra;
- ✓ Uma variável **nunca** poderá conter espaços em **branco** em seu nome. Se precisar usar espaço coloque o caracter **underline** (**\_**) entre uma palavra e outra;
- ✓ Uma variável **nunca** poderá receber nenhum tipo de **acentuação** e nem o caractere **cê-cedilha** (**ç**) em seu nome ou qualquer outro caractere especial; e,
- ✓ Uma variável **nunca** poderá receber o mesmo **nome** de uma **palavra reservada** ao ambiente de programação.

### Exemplos de nomes de variáveis ou constantes:

Nome, NomeAluno, Nome\_Aluno, NOME, NOMEALUNO, NOME\_ALUNO, PI, x123, X25, etc.

## **4.2 Operadores**

Muitas vezes dentro de um programa precisamos realizar operações matemáticas, comparações lógicas, cálculos, incrementar ou decrementar, atribuir valores, entre outros. Para essas atividades fazemos uso de operadores, vejamos a seguir quais são eles e como utilizá-los.

### **4.2.1 Operadores aritméticos ou matemáticos**

Na construção de pseudocódigos ou programas, muitas vezes necessitamos realizar operações matemáticas como: equações, funções, expressões algébricas, entre outras. Para isso necessitamos fazer uso de operadores aritméticos ou matemáticos. Vejamos abaixo os principais:

- + Adição
- Subtração
- ^, ↑ Potenciação
- / Divisão
- \* Multiplicação

### Exemplos:

- a)  $X \leftarrow 2 * 3$
- b)  $x = a^b$
- c)  $cont := cont + 1$
- d)  $R = 5 \uparrow C - A * (2 \uparrow A + (6 \uparrow 2 - 1))$

### **4.2.2 Operadores relacionais**

Esses operadores são utilizados quando precisamos realizar comparações dentro de blocos de programas. Vejamos abaixo os principais:

- ==, = Igual
- !=, ≠, <> Diferente

- >            Maior que
- <            Menor que
- >=          Maior ou igual que
- <=          Menor ou igual que

Exemplos:

- a)      T == y ou T = y
- b)      5 != 3 ou 5 <> 3
- c)      Nro <= 10
- d)      10 < 5

### 4.2.3 Operadores lógicos

Os operadores lógicos também são conhecidos como operadores **booleanos** ou operadores biestáveis, os quais podem assumir somente dois valores: *True* (Verdadeiro) ou *False* (Falso). Esses operadores são muito utilizados quando precisamos testar duas ou mais expressões dentro de um programa. Vejamos abaixo os principais:

- &&, .e., .E., and**    Operador lógico **E** //Retorna valor verdadeiro, somente se todas as sentenças forem verdadeiras
- ||, .ou., .OU., or**    Operador lógico **OU** //Retorna valor verdadeiro, se pelo menos uma das sentenças forem verdadeiras
- !, .não., .NÃO., not** Operador lógico **NÃO** //Inverte o valor lógico da sentença: V para **F** e F para **V**

Exemplo 1: Tabela para dados lógicos

**Quadro 5. Manipulação de valores lógicos**

X	Y	X.E.Y	X.OU.Y	.NÃO.X	.NÃO.Y
V	V	V	V	F	F
F	V	F	V	V	F
V	F	F	V	F	V
F	F	F	F	V	V

Fonte: Elaborado pelo autor

### Exemplo 2: Resolução passo a passo de uma expressão lógica

Para ilustrarmos a resolução de uma expressão lógica passo a passo, consideremos os valores  $A = -3$ ;  $B = 10$ ;  $C = 1$ ; e,  $D = 5$ , para a expressão lógica abaixo:

1.  $.\text{não.} (D < 3) .\text{ou.} .\text{não.} (C > 7) .\text{e.} (A > B)$
2.  $.\text{não.} (5 < 3) .\text{ou.} .\text{não.} (1 > 7) .\text{e.} (-3 > 10)$
3.  $. \text{não.} (F) .\text{ou.} .\text{não.} (F) .\text{e.} (F)$
4.  $(V) .\text{ou.} . \text{não.} (F) .\text{e.} (F)$
5.  $(V) .\text{ou.} (V) .\text{e.} (F)$
6.  $(V) .\text{e.} (F)$
7. **F**

Analisando a expressão lógica acima, notamos que o primeiro passo para resolvê-la é substituir todas as variáveis pelos seus respectivos valores (linha 2). Segundo passo (linha 3), da esquerda para direita realizamos os testes lógicos com os valores colocando dentro dos parênteses os resultados V ou F. Terceiro passo (linha 4), invertemos todos os valores lógicos (da esquerda para a direita linha a linha) pela ocorrência do operador *.não.*, V se torna F e, vice versa. Antepenúltimo e penúltimo passo (linhas 5 e 6), da esquerda para direita realizamos o teste lógico para que fique somente duas sentenças a serem julgadas para se obter o resultado final (linha 7).

### Exemplo 3: Representação de uma expressão computacional em pseudocódigo para a linguagem C

$. \text{não.} (D < 3) .\text{ou.} .\text{não.} (C > 7) .\text{e.} (A > B)$  – Pseudocódigo

$!(D < 3) || !(C > 7) \&\& (A > B)$  – Linguagem C

#### **4.2.4 Operadores de atribuição**

Esses operadores são utilizados para a atribuição de valores para variáveis ou constantes. Vejamos abaixo os principais:

$:=, =, \leftarrow$  Operadores de atribuição

#### Exemplos:

- a)  $X := X+1$
- b)  $\text{Nome} = \text{"Carlinho"}$
- c)  $\text{Saldo} \leftarrow 125.50$

#### 4.2.5 Operadores de incremento/decremento

Esses operadores são utilizados para incrementar ou decrementar uma variável de 1 em 1. Vejamos abaixo os principais

**++** Operador de incremento (+1)

**--** Operador de decremento (-1)

Exemplos:

a) `CONT++` ou `CONT = CONT+1`

b) `x--` ou `x = x-1`

#### 4.2.6 Operadores especiais

Esses operadores realizam operações que os outros operadores não podem realizar. Foram desenvolvidos para atender algumas tarefas específicas. Vejamos abaixo os principais:

**mod** Resto da divisão inteira

**div** Quociente da divisão inteira

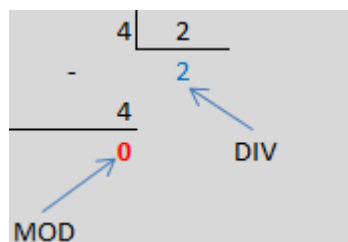
**+=, -=, \*=, /=** Funcionam como operadores matemáticos e de atribuição ao mesmo tempo

Exemplos:

a) `4mod2` //Retorna o resto da divisão, no caso **zero**.

b) `4div2` //Retorna o quociente da divisão, no caso **dois**.

**Figura 5. Aplicação dos operadores div e mod**



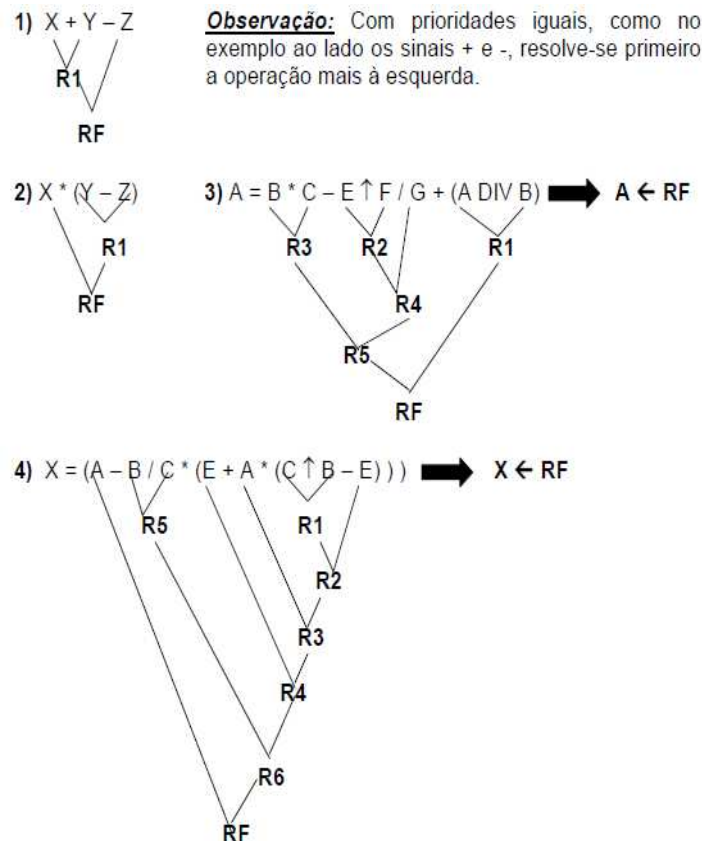
Fonte: elaborado pelo autor a partir do programa *Microsoft Excel* 2007.

c) `x += 2` //A variável `x` está recebendo o incremento de **+2**.

### 4.2.7 Hierarquia de operadores

Como nas expressões matemáticas, as expressões em computação também são resolvidas obedecendo a uma hierarquia como mostrado na figura 6.

**Figura 6. Hierarquia de operadores**



Fonte: Paulo (2007?)

Vejamos abaixo a ordem de prioridade dos operadores:

1. Parênteses (com vários níveis);
2. Funções;
3. Potenciação ( $\wedge$ ,  $\uparrow$ );
4.  $*$ ,  $/$ ,  $\text{div}$ ,  $\text{mod}$ ;
5.  $+$ ,  $-$ ;
6. Operadores relacionais; e,
7. Operadores lógicos.



Uma expressão algébrica do tipo:  $2^6 + [C \cdot (17 - 2^3 : A) + 6^2 : 9] : (10 - 3)^2$  é representada como uma expressão computacional dessa forma:  $2^6 + (C * (17 - 2^3 / A) + 6^2 / 9) / (10 - 3)^2$ .

### Exemplo 1: Resolução passo a passo de uma expressão computacional

Para resolvermos passo a passo uma expressão computacional basta respeitarmos a hierarquia de operadores demonstrada na figura 5. Para ilustrarmos consideremos a expressão matemática:  $D = \{5 \cdot [(A^2 : B \cdot 6) + 5] - [-4 - (5^2 + A \cdot B)]\}$ .

Nas linhas que seguem vamos transformar a expressão matemática em uma expressão computacional e resolver passo a passo até chegar ao resultado final. Consideremos os valores:  $A = 4$ ;  $B = 8$ ; e,  $C = 3$ .

1.  $D = \{5 \cdot [(A^2 : B \cdot 6) + 5] - [-4 - (5^2 + A \cdot B)]\}$
2.  $D \leftarrow (5 * (A^2 / B * 6) - (-4 - (5^2 + A * B)))$
3.  $D \leftarrow (5 * (4^2 / 8 * 6) - (-4 - (5^2 + 4 * 8)))$
4.  $D \leftarrow (5 * (16 / 8 * 6) - (-4 - (5^2 + 4 * 8)))$
5.  $D \leftarrow (5 * (2 * 6) - (-4 - (5^2 + 4 * 8)))$
6.  $D \leftarrow (5 * (12 - (-4 - (5^2 + 4 * 8))))$
7.  $D \leftarrow (5 * (12 - (-4 - (25 + 4 * 8))))$
8.  $D \leftarrow (5 * (12 - (-4 - (25 + 32))))$
9.  $D \leftarrow (5 * (12 - (-4 - 57)))$
10.  $D \leftarrow (5 * (12 - (-61)))$
11.  $D \leftarrow (5 * (12 + 61))$
12.  $D \leftarrow (5 * 73)$
13.  $D \leftarrow 365$

Analisando a expressão computacional acima, informamos que nas linhas 2 a 12, o destaque em azul é o nível de parêntese interno a ser resolvido; o destaque em vermelho é expressão a ser resolvida. Para realizar o cálculo da expressão é preciso primeiramente substituir todos os operadores matemáticos pelos operadores computacionais (linha 2). O segundo passo (linha 3) é substituir todas as variáveis pelos seus respectivos valores dentro dos parênteses. Feito isso começamos a realizar os cálculos pelos **parênteses mais internos**, havendo vários deles, começamos sempre a resolução da esquerda para a direita (linha 4 – destaque em vermelho), **respeitando a ordem de prioridade dos operadores**. Depois de realizados os cálculos, e o resultado do cálculo se resumir apenas a um valor dentro do parêntese, o mesmo é eliminado (linha

6, por exemplo). Realizamos os mesmos procedimentos para as linhas 5 a 12 até chegar ao resultado final (linha 13).

### Exemplo 2: Representação de uma expressão computacional em pseudocódigo para linguagem C

$D \leftarrow (5 * ((A^2 / B^6) - (-4 - (5^2 + A * B))))$  – Pseudocódigo

$D = (5 * ((\text{pow}(A, 2) / B^6) - (-4 - (\text{pow}(5, 2) + A * B))))$  – Linguagem C

## 4.3 Tipos de dados

Dados são conjuntos de informações processadas pelo compilador. Os dados são representados por quatro tipos, a saber: **dados numéricos inteiros**, **dados numéricos reais**, **dados caracteres** ou **alfanuméricos** e **dados lógicos**. São conhecidos como *Dados Primitivos*, ou seja, são os primeiros tipos reconhecidos por um computador e a partir deles são criados os outros tipos de dados. A partir de agora veremos cada tipo.

### 4.3.1 Tipos de dados inteiros

São representados pelo conjunto dos números inteiros positivos e negativos. Exemplos: 9, 10, -10, 1, -2.

### 4.3.2 Tipos de dados reais

São representados pelo conjunto dos números inteiros positivos e negativos bem como pelo conjunto dos números fracionários positivos e negativos. Exemplos: 1, 2, -3, 32.55, -1.89.

### 4.3.3 Tipos de dados caracteres

São representados por letras, números, e caracteres especiais (\*, /, %, #, entre outros). São conhecidos também como dados alfanuméricos, strings, cadeias ou literais. Devem ser escritos sempre entre aspas duplas (") e em, alguns casos, aspas simples ('). Exemplos: "Casa", "L", "\$", "(66) 3481-1857".

### 4.3.4 Tipos de dados lógicos

São representados por tipos lógicos ou booleanos. Só admitem duas representações: Verdadeiro ou Falso (True ou False). Exemplos: F = false, V = true.

### 4.3.5 Tamanho e faixa de valores para os tipos de dados

Cada dado reconhecido por um computador possui um nome, um tipo, um tamanho e um valor. Observe o tamanho e a faixa de abrangência de cada tipo de dado no quadro 6.

**Quadro 6. Tamanho e faixa de valores para tipos de dados**

Tipo	Tamanho em Byte(s)	Faixa Mínima
Caracter	1	-127 a 127
Inteiro	4	-2.147.483.648 a 2.147.483.647
Real	4	Seis dígitos de precisão
Lógico	1	Verdadeiro ou Falso

Fonte: [www.google.com](http://www.google.com)

## 4.4 Memória e processamento de dados

Memória é um local no computador que serve para armazenamento de dados. Sabemos que a memória do computador é dividida em: Principal (Memória RAM) e Secundária (HDs, CDs ROM, entre outros). A memória principal é um espaço reservado para armazenamento de programas ou grupos de instruções a serem executados pelo computador, diretamente controlada pela unidade central de processamento (CPU).

Para entendermos melhor o conceito de memória, observe o algoritmo abaixo:

Algoritmo **"SOMA\_DOIS\_NUMEROS"**

1. Início.
2. Pegue uma calculadora.
3. Se a calculadora estiver desligada, ligue-a.
4. Digite um número qualquer na calculadora.
5. Aperte a tecla com o sinal de +.
6. Digite outro número qualquer na calculadora.
7. Aperte a tecla com o sinal de =.
8. Veja o resultado no visor.
9. Fim

Pensemos no conceito de memória a partir desse momento! Nas linhas 4 e 6 do algoritmo acima, para onde vão os números digitados? E o resultado na linha 8 no visor da calculadora, onde é armazenado?

Os questionamentos acima podem ser respondidos por meio do **Conceito de Memória**. Se pensarmos bem, uma calculadora pode simular um computador, pois a mesma é dotada de estrutura básica de ENTRADA, PROCESSAMENTO, MEMÓRIA E SAÍDA.

Pense bem! Se tivéssemos digitado os números **5** e **7** na calculadora teríamos como resultado o número **12**. Ou seja,  $5 + 7 = 12$ . Certo?

O processamento para resolução do Algoritmo "SOMA\_DOIS\_NUMEROS" poderia ser descrito na MEMÓRIA DA CALCULADORA conforme demonstrado na tabela 1.

**Tabela 1. Processamento do algoritmo para soma de dois números em uma calculadora**

Passo	Primeiro Número Digitado	Segundo Número Digitado	Resultado no Visor
1	...	...	...
2	...	...	...
3	...	...	...
4	5	...	5
5	5	...	5+
6	5	7	5+7
6	5	7	5+7
7	5	7	12
8	5	7	<b>12</b>

Fonte: Elaborado pelo autor

O processamento na MEMÓRIA DO COMPUTADOR do Algoritmo "SOMA\_DOIS\_NUMEROS" se daria de forma semelhante ao exposto acima, a única diferença é que veremos agora o conceito de memória por meio de uma identificação chamada de *Variável*. Vejamos abaixo o resultado do processamento na tabela 2.

**Tabela 2. Processamento do algoritmo para soma de dois números em um computador**

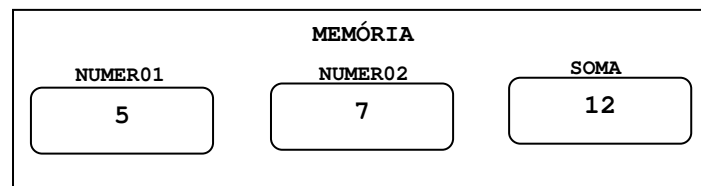
Passo	NUMERO1	NUMERO2	SOMA
1	...	...	...
2	...	...	...
3	...	...	...
4	5	...	...
5	5	...	...
6	5	7	...
6	5	7	...
7	5	7	12
8	5	7	<b>12</b>

Fonte: Elaborado pelo autor

Percebam que na segunda, terceira e quarta colunas da tabela 2 existem as Variáveis NUMERO1, NUMERO2 e SOMA.

Na figura 7 é representado:  $SOMA \leftarrow NUMERO1 + NUMERO2$ .

**Figura 7. Dados representados na memória do computador**



Fonte: Elaborado pelo autor

### Exercícios propostos:

1. Conceitue Variáveis e Constantes? Dê exemplos.
2. O que são operadores? Quais os principais usados em programação? Dê exemplos.
3. Com base na tabela-verdade, determine o resultado lógico (V ou F), resolvendo passo a passo as expressões lógicas abaixo. Considere para as respostas os valores:  $X = 2$ ;  $A = 4$ ;  $B = -5$ ;  $C = 3$  e  $D = 7$ .

- a.  $\neg (X > 3)$
- b.  $(X > 1) \wedge \neg (B > D)$
- c.  $\neg (D > 0) \wedge (C > 5)$
- d.  $\neg (X > 3) \vee (C > 7)$
- e.  $(A > B) \vee (C > B)$
- f.  $(X \leq 2) \wedge (B > 10) \vee (A = 3)$
- g.  $(X < 1) \wedge (B = D) \wedge (D = 5)$
- h.  $(D < 0) \vee (C > 5)$
- i.  $\neg (D > 3) \vee \neg (B > 7)$
- j.  $\neg ((X > C) \wedge (A > B)) \vee ((B < C) \vee \neg (D < A) \vee (B < 10))$

4. Realize o cálculo, passo a passo, das expressões computacionais abaixo, atribuindo o valor final para a variável correspondente à expressão. Considere  $A = 4$ ;  $B = 8$  e  $C = 3$ .

- a)  $A \leftarrow 2 * A + 7 * B$
- b)  $B \leftarrow (3 * C + 4) - 5$
- c)  $C \leftarrow (A * B) + 5 * (8 / 2 * C)$
- d)  $D \leftarrow B \bmod A$
- e)  $E \leftarrow B \div C$

- f)  $F \leftarrow (5 * ((A^2 / B * 6) + 5) - (-4 - (5^2 + A * B)))$
- g)  $G \leftarrow (A^2 * (-10 + (B^2 + C^2) + 50 \bmod 25 * (A * B) + C))$
- h)  $H \leftarrow (5^C - A * (2^A + 3^2 - 1)) \text{div} 7$
- i)  $I \leftarrow 200 - C * ((72 - 65 + 10) * (87 - 7 * B))$
- j)  $J \leftarrow 2^6 + (C * (17 - 2^3 / A) + 6^2 / 9) / (5 - 3)^2$

5. Quais os tipos de dados reconhecidos por um computador? Explique cada um e dê exemplos.
6. Transforme as expressões computacionais do exercício 4 em expressões que a linguagem de programação C possa interpretar. Realize uma pesquisa na Internet para resolver os exercícios.
7. Pesquise sobre Diagrama de Blocos (DB). Em seguida mostre um exemplo de Algoritmo em DB.

## UNIDADE V

### DIAGRAMA DE BLOCOS, PSEUDOCÓDIGO E VISUALG




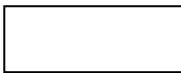
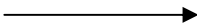

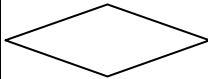
Os objetivos específicos dessa unidade são:

- ✓ Dotar o aluno de conhecimento sobre *Diagrama de Blocos*;
- ✓ Dotar o aluno de conhecimento sobre *Pseudocódigo*; e,
- ✓ Realizar a transposição de algoritmos em DB para Pseudocódigo utilizando o ambiente de programação *VisuAlg*.

#### 5.1 Diagrama de blocos

**Diagrama de Blocos (DB)** é uma metodologia proposta por Manzano & Oliveira (1997), é baseada em figuras geométricas organizadas em fluxogramas, as quais representam ações realizadas por meio de uma estrutura chamada Diagrama de Blocos (DB). É muito eficiente para projetar um algoritmo. Seus principais símbolos são mostrados e descritos no quadro 7.

**Quadro 7. Principais símbolos de um DB**

	Elipse – Símbolo terminal que indica o <b>início</b> e o <b>fim</b> do algoritmo.
	Display – Símbolo exibir, o qual é usado para <b>mostrar/imprimir</b> um conteúdo na <b>tela</b> .
	Paralelogramo – Símbolo utilizado para a entrada e saída de dados, usualmente utilizado para <b>armazenamento</b> de dados na memória.
	Retângulo – Símbolo utilizado para o <b>processamento</b> dos dados.
	Seta – Símbolo utilizado para indicar a direção do <b>fluxo</b> de dados.
	Conector – Símbolo utilizado para escrever a <b>continuação</b> do fluxo de dados na mesma página ou em outra página.
	Losango – Símbolo usado para a tomada de <b>decisões</b> ou <b>repetição</b> de um processamento.

Fonte: Manzano & Oliveira (2011)

Além dos símbolos apresentados acima, existem outros, mas para o estudo de algoritmos usaremos somente os do quadro 5.

### 5.1.1 Algoritmo e processamento de um programa em DB

**Algoritmo:** Solicitar dois números inteiros ao usuário, realizar a soma dos números e, em seguida apresentar o resultado na tela.

**Solução:**

1. Criar duas variáveis: *n1* e *n2* para armazenar os números fornecidos pelo usuário;
2. Criar uma terceira variável chamada *Soma* para armazenar o resultado de  $n1 + n2$ ;
3. Apresentar o resultado da variável *Soma* na tela.

DB "SomaDoisNumeros"

**Linha 1:**

Início

**Linha 2:**

*n1*, *n2*, *Soma*:

**Linha 3:**

"Digite o primeiro  
número inteiro: "

**Linha 4:**

*n1*

**Linha 5:**

"Digite o segundo  
número inteiro: "

**Linha 6:**

*n2*

**Linha 7:**

$Soma \leftarrow n1 + n2$

**Linha 8:**

"A soma dos números é  
igual a: ", *Soma*

**Linha 9:**

Fim

Analisando o algoritmo DB "SomaDoisNumeros" temos:

**Linha 1:** Início do algoritmo;

**Linha 2:** Declaração de variáveis com seus respectivos tipos;



**Linha 3:** Solicitação da primeira entrada ao usuário (primeira mensagem na tela);

**Linha 4:** Armazenamento da primeira entrada do usuário na memória por meio da variável *n1*;

**Linha 5:** Solicitação da segunda entrada ao usuário (segunda mensagem na tela);

**Linha 6:** Armazenamento da segunda entrada do usuário na memória por meio da variável *n2*;

**Linha 7:** Processamento dos dados (somatório dos dois valores fornecidos pelo usuário e armazenamento na variável *Soma*);

**Linha 8:** Apresentação do resultado da variável *Soma* na tela.

**Linha 9:** Fim do algoritmo.

Agora vamos realizar o processamento do algoritmo DB “SomaDoisNumeros” tendo como entrada os valores 3 e 6 (informados pelo usuário). Veja como é feito na tabela 3.

**Tabela 3. Processamento do algoritmo DB “SomaDoisNumeros”**

<b>Linha</b>	<b>n1</b>	<b>n2</b>	<b>Soma</b>
1	...	...	...
2	...	...	...
3	3	...	...
4	3	...	...
5	3	6	...
6	3	6	...
7	3	6	9
8	3	6	<b>9</b>
9	3	6	<b>9</b>

*Fonte:* Elaborado pelo autor.

## 5.2 Pseudocódigo e VisuAlg

**Pseudocódigo** é uma técnica textual de representação de um algoritmo. Também é conhecida como Português Estruturado. A técnica é baseada em uma PDL (*Program Design Language*), que é uma linguagem genérica na qual é possível representar um algoritmo de forma semelhante à das linguagens de programação.

A estrutura de um algoritmo em *pseudocódigo* pode variar um pouco de acordo com o autor ou com base na linguagem de programação que será utilizada posteriormente, mas essas variações ocorrem apenas na sintaxe, pois a semântica deve ser exatamente a mesma.

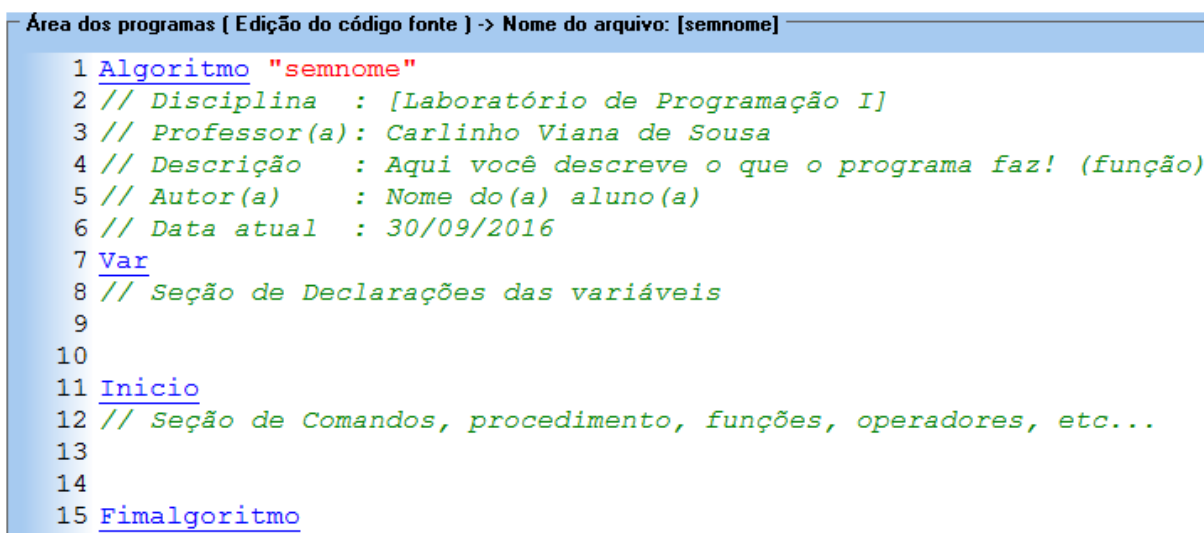
Para o estudo de *pseudocódigos* utilizaremos o ambiente *VisuAlg* que é um editor e interpretador de algoritmos escritos em *Pseudocódigo* idealizado por Cláudio Morgado de Souza. De acordo com seu criador o *software* nasceu da necessidade de se obter uma ferramenta que pudesse interpretar códigos escritos em Portugol, ou seja, códigos escritos em Língua Portuguesa.

O ambiente pode ser baixado no site <http://baixaki.com.br> por meio do *link* <http://www.baixaki.com.br/download/visualg.htm> ou no site do próprio fabricante por meio do *link* <http://www.apoioinformatica.inf.br/produtos/visualg>. Nesse site você também encontrará todas as informações sobre o ambiente *VisuAlg* e sobre seu idealizador.

### 5.2.1 Estrutura básica de um programa escrito em VisuAlg

Todo programa escrito no ambiente *VisuAlg* tem a estrutura como demonstrada na figura 8.

**Figura 8. Estrutura de um programa no VisuAlg**



```
1 Algoritmo "semnome"
2 // Disciplina : [Laboratório de Programação I]
3 // Professor(a): Carlinho Viana de Sousa
4 // Descrição : Aqui você descreve o que o programa faz! (função)
5 // Autor(a) : Nome do(a) aluno(a)
6 // Data atual : 30/09/2016
7 Var
8 // Seção de Declarações das variáveis
9
10
11 Inicio
12 // Seção de Comandos, procedimento, funções, operadores, etc...
13
14
15 Fimalgoritmo
```

Fonte: Software VisuAlg 3.0

Analisando a figura 8 temos: a) na linha 1 fica o espaço reservado para o nome do programa; b) nas linhas 2 a 6 temos o cabeçalho do programa em forma de comentários (//); c) nas linhas 7 a 10 temos o espaço reservado para a criação das variáveis que serão utilizadas no programa; e, c) nas linhas 11 a 15 temos o bloco (**Início/Fimalgoritmo**), no qual o programa será escrito com a utilização de comandos, operadores, funções, variáveis, etc.

### 5.2.2 Escrita de um algoritmo/programa no ambiente VisuAlg

A figura 9 apresenta um exemplo de programa escrito no ambiente VisuAlg.

Figura 9. Programa escrito no VisuAlg

```
1 Algoritmo "SomaDoisNumeros"  
2 //Função: Realizar a soma de dois números inteiros  
3 //Autor: Carlinho Viana de Sousa  
4 //Data: 30/09/2016  
5 //Seção de Declarações  
6 Var  
7   x, y, Soma: inteiro  
8 Inicio  
9 //Seção de Comandos  
10  escreva("Digite o primeiro número inteiro: ")  
11  leia(x)  
12  escreva("Digite o segundo número inteiro: ")  
13  leia(y)  
14  Soma <- x + y  
15  escreva("A soma dos números é igual a: ", Soma)  
16 Fimalgoritmo
```

Fonte: Software VisuAlg 3.0

Analisando o Algoritmo "SomaDoisNumeros" no ambiente *VisuAlg* (figura 9), temos:

- ✓ na linha 1 o nome do algoritmo ou programa, sempre entre abre e fecha aspas duplas ("") precedido da palavra Algoritmo;
- ✓ nas linhas 2, 3, 4, 5 e 9 o símbolo // (comentários). Os comentários não serão considerados pelo compilador na hora da execução do programa, só servem para documentá-lo;
- ✓ na linha 6 a palavra var que indica um espaço reservado para a declaração/criação de variáveis;
- ✓ na linha 7 abaixo da palavra var podemos declarar as nossas variáveis com seus respectivos tipos;
- ✓ na linha 8 temos o comando Inicio, o que significa que daremos início ao algoritmo/programa;
- ✓ na linha 10 temos a função escreva, a qual emite uma mensagem na tela para o usuário;
- ✓ na linha 11 temos a função leia, a qual armazena um valor digitado pelo usuário na memória por meio de uma variável;
- ✓ nas linhas 12 e 13 temos novamente as funções leia e escreva, as quais já foram explicadas acima;

✓ na linha 14 temos o processamento das informações digitadas pelo usuário. Temos o somatório das variáveis  $x$  e  $y$  e o armazenamento na variável *Soma* por meio do operador de atribuição  $<-;$ ;

✓ na linha 15 temos o resultado da variável *Soma* apresentado na tela por meio da função escreva; e,

✓ na linha 16 temos o comando Fimalgoritmo, o qual determina o fim do algoritmo/programa.

### 5.2.3 Processamento de um algoritmo/programa no ambiente VisuAlg

A tabela 4 apresenta o processamento do Algoritmo “SomaDoisNumeros” no ambiente *VisuAlg*.

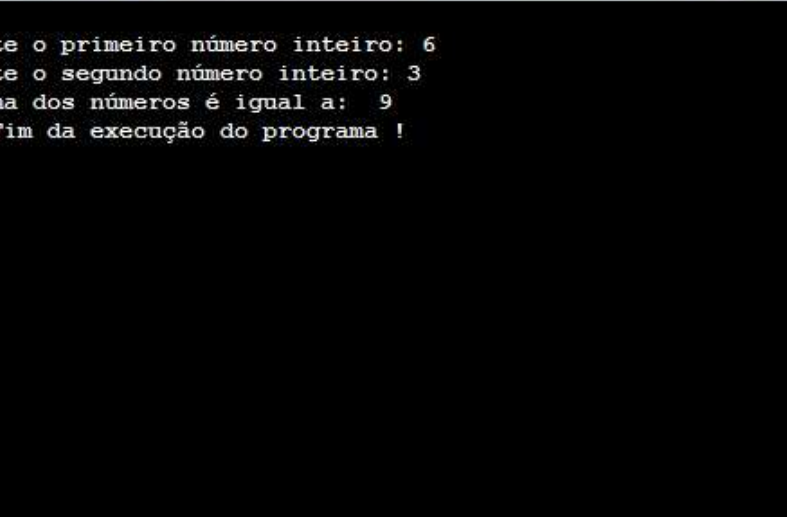
**Tabela 4. Processamento do algoritmo “SomaDoisNumeros” no VisuAlg**

<b>Linha</b>	<b>x</b>	<b>y</b>	<b>Soma</b>
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...
7	...	...	...
8	...	...	...
9	...	...	...
10	3	...	...
11	3	...	...
12	3	6	...
13	3	6	...
14	3	6	9
15	3	6	<b>9</b>
16	3	6	<b>9</b>

Fonte: Elaborado pelo autor

A figura 10 apresenta o Algoritmo “SomaDoisNumeros” compilado no ambiente *VisuAlg*. O Algoritmo/Programa é apresentado no console do sistema operacional (SO), nesse caso no MS-DOS, já que SO utilizado foi o *Windows*. O algoritmo/programa é processado linha a linha até que seja dado o resultado final: >>> Fim da execução do programa !.

Se ocorrer algum erro durante a execução do algoritmo/programa o usuário será direcionado ao ambiente de programação do *VisuAlg*, onde o compilador emitirá a mensagem de erro e indicará a linha de programação onde o erro ocorreu.



```
C:\ Console simulando o modo texto do MS-DOS
Digite o primeiro número inteiro: 6
Digite o segundo número inteiro: 3
A soma dos números é igual a: 9
>>> Fim da execução do programa !
```

A figura 11 apresenta o relatório do Algoritmo “SomaDoisNumeros” no ambiente VisuAlg. É apresentado um histórico do algoritmo/programa depois de compilado.

[illegible]

Algoritmos e Lógica de Programação  
Prof. Me. Carlinho Viana de Sousa – profcarlinho@gmail.com

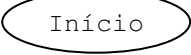
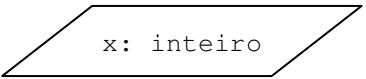
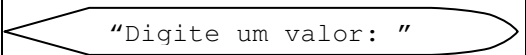
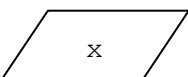
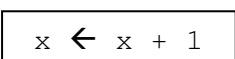
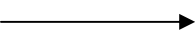

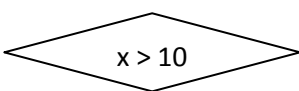

Página 47/105

Observando a figura 11 notamos que é apresentado o comportamento das variáveis durante a execução do algoritmo/programa. São listadas as variáveis, seus respectivos tipos e seus valores. Também é apresentada uma área para visualização dos resultados idêntica à da figura 10.

### 5.3 DB x Pseudocódigo

Os símbolos utilizados no DB podem ser representados em *Pseudocódigo* e, vice-versa, a única diferença de um algoritmo escrito em *DB* para um algoritmo escrito em *Pseudocódigo* está na sintaxe. Pois, no *DB* fazemos uso de figuras, símbolos para representar as estruturas de um algoritmo, enquanto que no *Pseudocódigo* fazemos uso de texto. Veja a comparação no quadro 8.

**Quadro 8. Comparativo entre DB e Pseudocódigo**

DB	PSEUDOCÓDIGO
	<u>início</u>
	<u>Var</u> x: <u>inteiro</u>
	<u>escreva</u> ("Digite um valor: ")
	<u>leia</u> (x)
	x ← x + 1
	Linhas do algoritmo: 1, 2, 3...
	O algoritmo pode ser escrito utilizando quantas linhas forem necessárias em um mesmo documento.
	<u>se</u> (x > 10) <u>então</u> <u>fimse</u>  <u>enquanto</u> (x > 10) <u>faca</u> <u>fimenquanto</u>
	<u>Fimalgoritmo</u>

Fonte: Elaborado pelo autor

Exemplo 1 – Programa para somar dois números em linguagem de programação C:

```
1 //Programa para somar dois números
2 #include <stdio.h>
```

```

3 #include <conio.h>
4 int main(){
5     int n1, n2, soma;
6     printf("Digite o primeiro numero: ");
7     scanf("%d",&n1);
8     printf("Digite o segundo numero: ");
9     scanf("%d",&n2);
10    soma = n1 + n2;
11    printf("A soma e igual a: %d",soma);
12    getch();
13    return(0);
14 }

```

### Exercícios propostos:

1. O que é e para que serve o Diagrama de Blocos (DB). Dê exemplo.
2. O que é e para que serve um *Pseudocódigo*. Dê exemplo.
3. Escreva um conjunto de instruções em *DB* que realize:
  - a. A subtração de dois números inteiros;
  - b. A multiplicação de dois números inteiros;
  - c. A divisão de dois números reais.
4. Escreva um conjunto de instruções em *DB* para realizar a soma e a subtração de dois números inteiros digitados pelo usuário. Ao resultado da soma deverá ser acrescentado +5. Ao resultado da subtração deverá ser multiplicado 5.
5. Escreva um conjunto de instruções em *DB* para realizar a impressão do nome e sobrenome de uma pessoa digitados pelo usuário.
6. Escreva um conjunto de instruções em *DB* para realizar a média aritmética de quatro notas digitadas pelo usuário.
7. Escreva um conjunto de instruções em *DB* para realizar a soma e a multiplicação de três números inteiros usando o conceito de propriedade distributiva. Imprimir os resultados de dois em dois.

Exemplo: números digitados: 1, 2, 3

Resultados:

$$1 + 2 = 3$$

$$1 * 2 = 2$$

Imprimir(3,2)

$1 + 3 = 4$

$1 * 3 = 3$

Imprimir(4,3)

$2 + 3 = 5$

$2 * 3 = 6$

Imprimir(5,6)

8. Escreva um conjunto de instruções em *DB* para calcular o quadrado de um número inteiro digitado pelo usuário.

9. Escreva um conjunto de instruções em *DB* que, a partir de dois números inteiros ( $n_1$ ,  $n_2$ ) calcule a soma do cubo de  $n_1$  mais o quadrado de  $n_2$ .

10. Construa um conjunto de instruções em *Pseudocódigo* que imprima o resto e o quociente da divisão de dois números inteiros digitados pelo usuário.



## UNIDADE VI

# ESTRUTURAS DE CONTROLE CONDICIONAIS E DE SELEÇÃO

Os objetivos específicos dessa unidade são:

- ✓ Dotar o aluno de conhecimentos sobre estruturas de condição e seleção;
- ✓ Ensinar o aluno a realizar testes em um programa; e,
- ✓ Realizar a transposição de algoritmos em DB para Pseudocódigo e, vice-versa.

### 6.1 Estruturas de controle condicionais

Como o próprio nome já indica essas estruturas fazem restrições no programa por meio de decisões/condições lógicas do tipo FALSO ou VERDADEIRO. São representadas pelos comandos se...entao...senao...fimse.

As estruturas de controle condicionais são divididas em três categorias: Estruturas de Controle Condicionais **Simples**, Estruturas de Controle Condicionais **Compostas** e Estruturas de Controle Condicionais **Encadeadas**. A seguir estudaremos cada uma delas.

#### 6.1.1 Estruturas de controle condicionais simples

Utilizadas para a tomada de decisão simples, considerando somente um valor do tipo VERDADEIRO para a sentença. São representadas pelo comando se...entao...fimse.

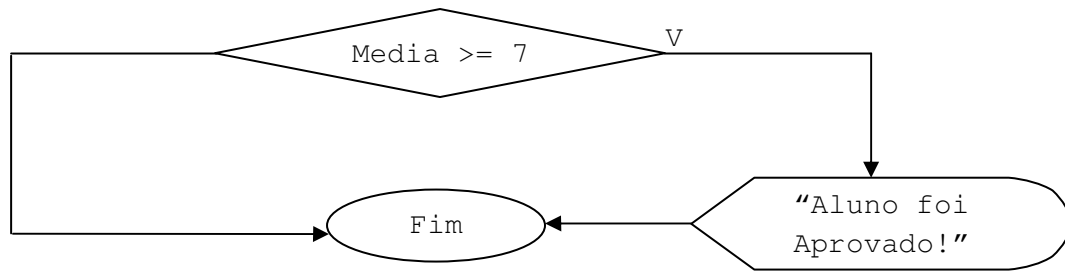
*Sintaxe:*

```
se (teste lógico VERDADEIRO) entao  
    <bloco de comandos>  
fimse
```

*Exemplo em Pseudocódigo:*

```
se (Media >= 7) entao  
    escreva("Aluno foi aprovado!")  
fimse
```

Exemplo em DB:



Exemplo em linguagem de programação C:

```
if (Media >= 7)
    printf("\nAluno foi aprovado!");
```

### 6.1.2 Estruturas de controle condicionais compostas

Utilizadas para a tomada de duas decisões lógicas, considerando um valor do tipo VERDADEIRO (se...entao) e outro do tipo FALSO (senao...fimse) para a sentença.

*Sintaxe:*

```
se (teste lógico VERDADEIRO) entao
    <bloco de comandos para o teste lógico VERDADEIRO>
```

```
senao
    <bloco de comandos para o teste lógico FALSO>
```

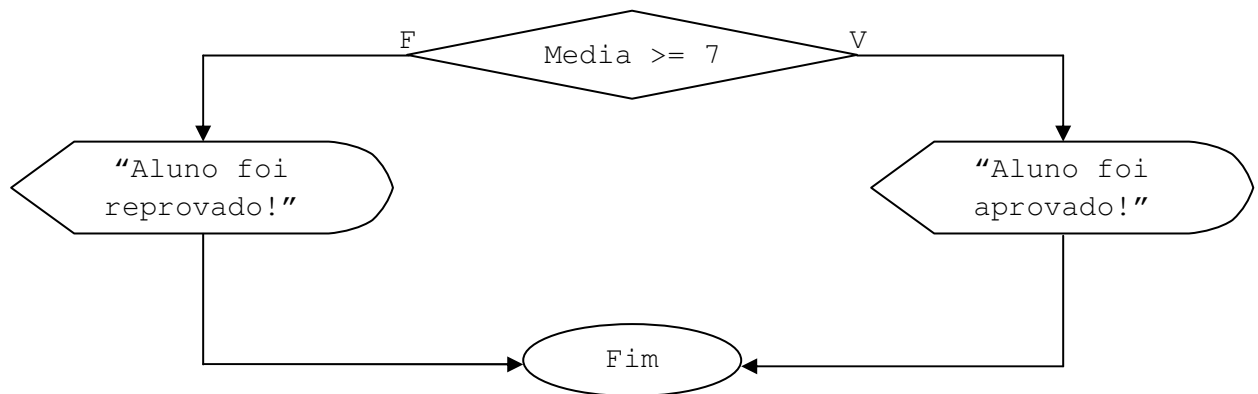
```
fimse
```

Exemplo em Pseudocódigo:

```
se (Media >= 7) entao
    escreva("Aluno foi aprovado!")
senao
    escreva("Aluno foi reprovado!")
fimse
```

*Nota: Apesar de existir dois blocos de programa no algoritmo acima, somente um será executado. Tudo vai depender do valor da variável *Media*.*

Exemplo em DB:



Exemplo em linguagem de programação C:

```
if (Media >= 7)
    printf("\nAluno foi aprovado!");
else
    printf("\nAluno foi reprovado!");
```

### 6.1.3 Estruturas de controle condicionais encadeadas

Utilizadas para a tomada de mais de duas decisões lógicas, considerando valores do tipo VERDADEIRO (se...entao) e FALSO (senao...fimse) para as sentenças.

*Sintaxe:*

```
se (teste lógico VERDADEIRO1) entao
    <bloco de comandos para o teste lógico VERDADEIRO1>
senao //teste lógico FALSO1
    se (teste lógico VERDADEIRO2) entao
        <bloco de comandos para o teste lógico VERDADEIRO2>
    senao
        <bloco de comandos para o teste lógico FALSO2>
    fimse
fimse
```

*Exemplo em Pseudocódigo:*

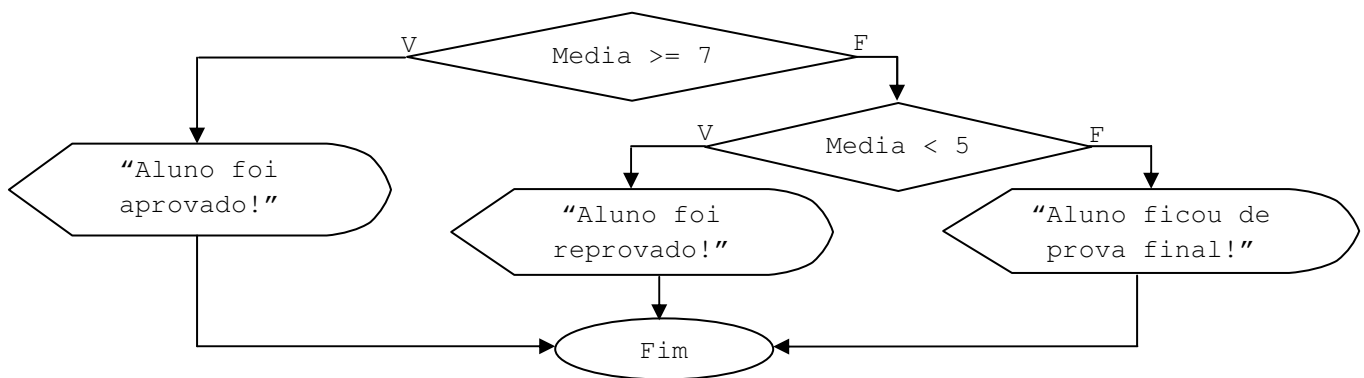
```
se (Media >= 7) entao
    escreva("Aluno foi aprovado!")
senao
    se (Media < 5) entao
        escreva("Aluno foi reprovado!")
```

```

senao
    escreva ("Aluno ficou de prova final!")
fimse
fimse

```

*Exemplo em DB:*



*Exemplo em linguagem de programação C:*

```

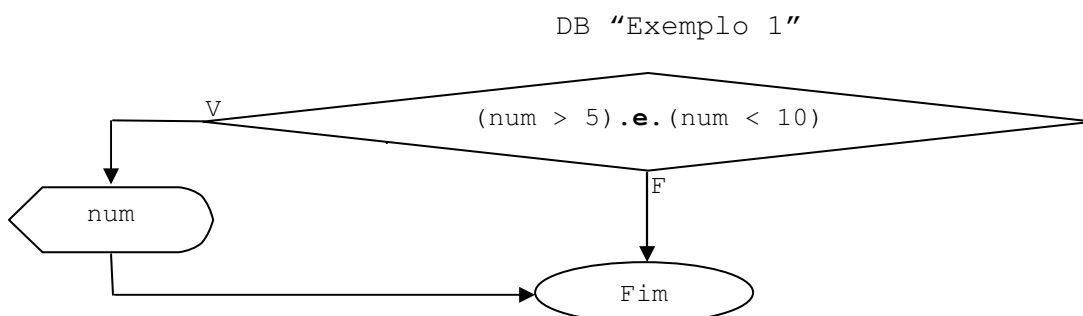
if (Media >= 7)
    printf("\nAluno foi aprovado!");
else
    if (Media < 5)
        printf("\nAluno foi reprovado!");
    else
        printf("\nAluno ficou de prova final!");

```

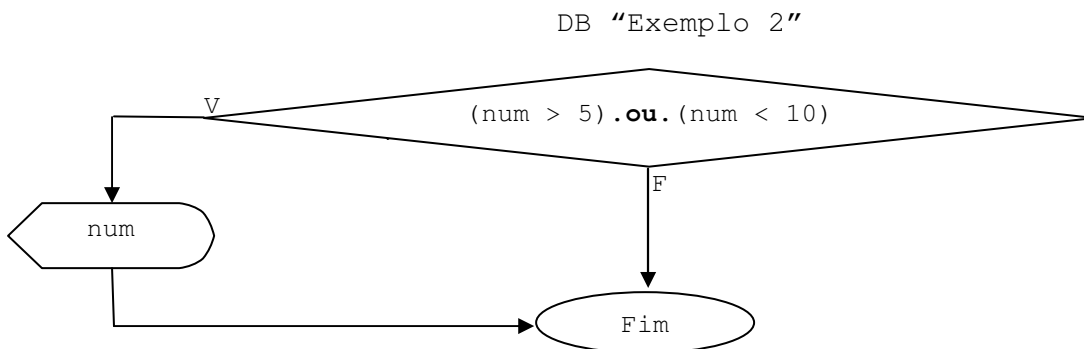
#### **6.1.4 Estruturas de controle condicionais com operadores .e. e .ou.**

É possível combinar as estruturas de controle condicionais com os operadores lógicos **.e.** e **.ou.**, ou melhor é possível realizar mais de um teste em uma única linha de programação. Para ilustrarmos o que foi exposto, consideremos os exemplos 1 e 2 apresentados a seguir.

**Exemplo 1** - Imprimir um número inteiro digitado pelo usuário, somente se o mesmo for maior que 5 e menor que 10:



**Exemplo 2** - Imprimir um número inteiro digitado pelo usuário, somente se o mesmo for maior que 5 **ou** menor que 10:



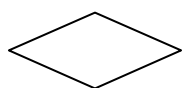
Analisando os dois exemplos acima, que diferenças podemos notar?

Obviamente no primeiro algoritmo (DB "Exemplo 1") só serão impressos números no intervalo de 6 a 9, pois, o operador **.e.** faz com que as duas condições ( $>5$  e  $<10$ ) sejam atendidas ao mesmo tempo.

Já no segundo algoritmo (DB "Exemplo 2") qualquer número inteiro digitado pelo usuário será impresso, uma vez que o operador **.ou.** faz com que pelo menos uma das condições seja atendida.

## 6.2 Estruturas de controle condicionais representadas no DB e em Pseudocódigo

No DB as estruturas de controle condicionais são representadas pela figura geométrica chamada losango.



➔ Símbolo usado para a tomada de decisões lógicas do tipo VERDADEIRO (V) ou FALSO (F).

A partir de agora faremos a transposição de algoritmos escritos em DB para Pseudocódigo e, vice-versa.

Consideraremos como exercício um algoritmo que fará a leitura de quatro notas de um aluno calculando a média das notas bem como a situação do aluno ("Aprovado", "Reprovado" ou de "Prova Final").

Para representarmos as três estruturas de controle condicionais (simples, compostas e encadeadas) o algoritmo será executado por partes. Faremos, primeiramente, o algoritmo para a situação de "Aprovado", em seguida para as situações de "Aprovado ou Reprovado" e, por último, para as situações de "Aprovado, Reprovado ou de Prova Final". Vejamos a seguir os resultados.

Exemplo 1: Transposição de algoritmo em Pseudocódigo para DB (situação: "Aprovado")

**Algoritmo** "AlunoAprovado"

**Var**

n1, n2, n3, n4, Media: **real**

**Início**

**escreva**("Digite a primeira nota: ")

**leia**(n1)

**escreva**("Digite a segunda nota: ")

**leia**(n2)

**escreva**("Digite a terceira nota: ")

**leia**(n3)

**escreva**("Digite a quarta nota: ")

**leia**(n4)

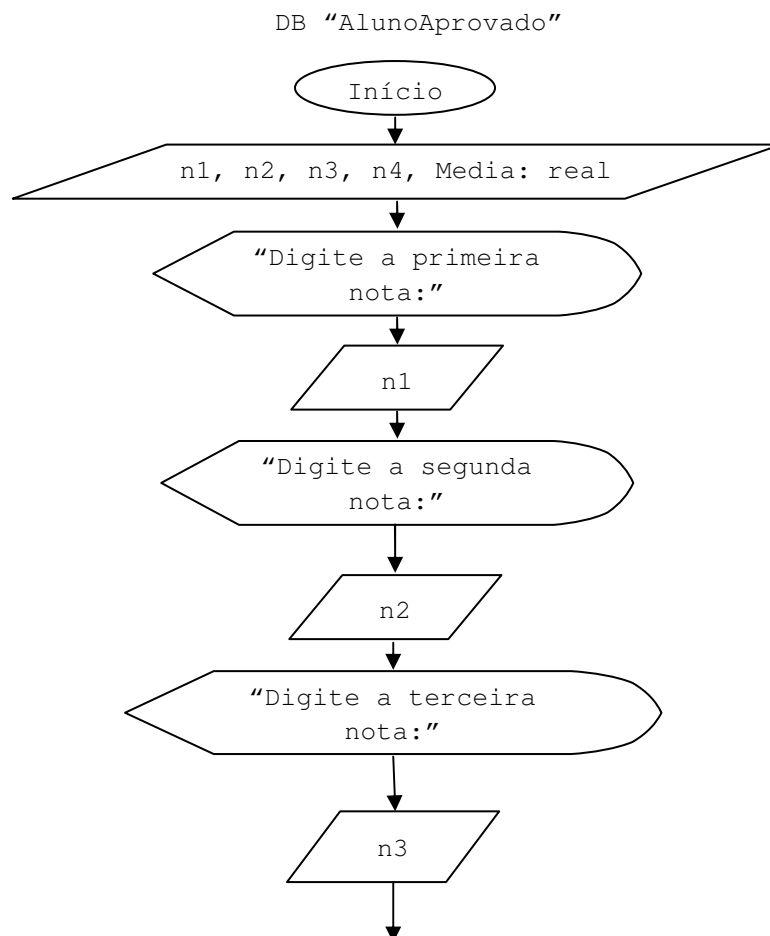
Media  $\leftarrow$  (n1+n2+n3+n4)/4

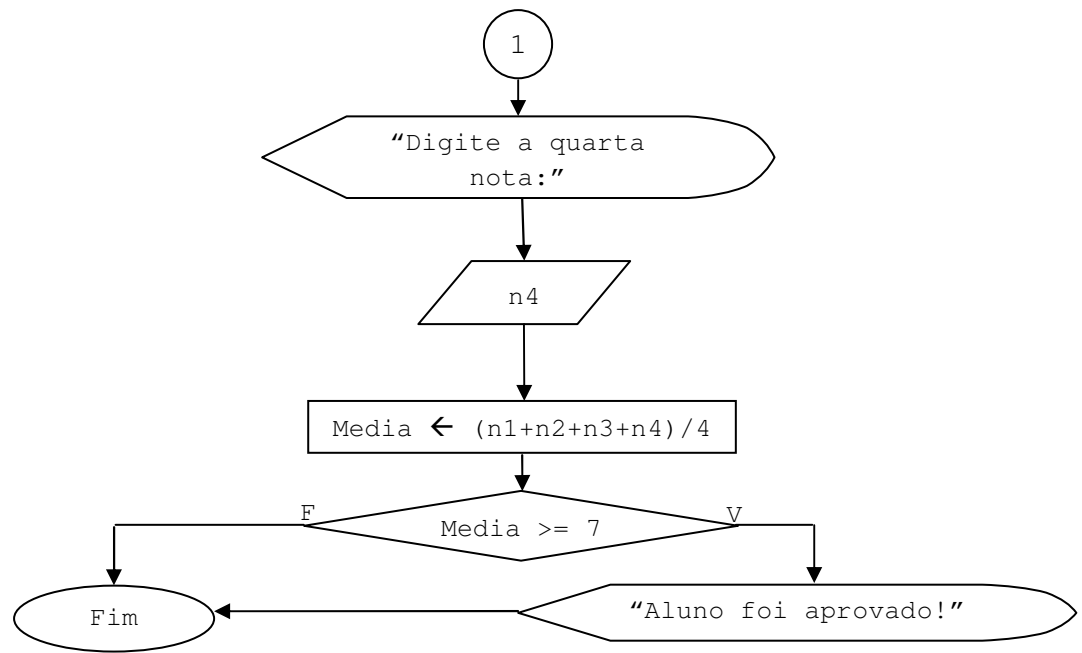
**se** (Media >= 7) **entao**

**escreva**("Aluno foi aprovado!")

**fimse**

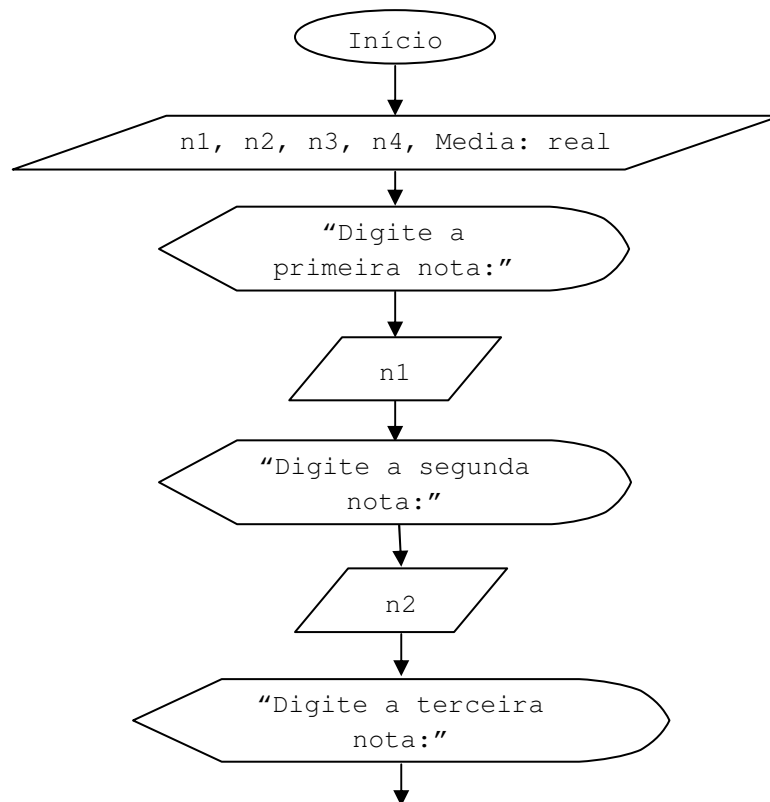
**Fimalgoritmo**

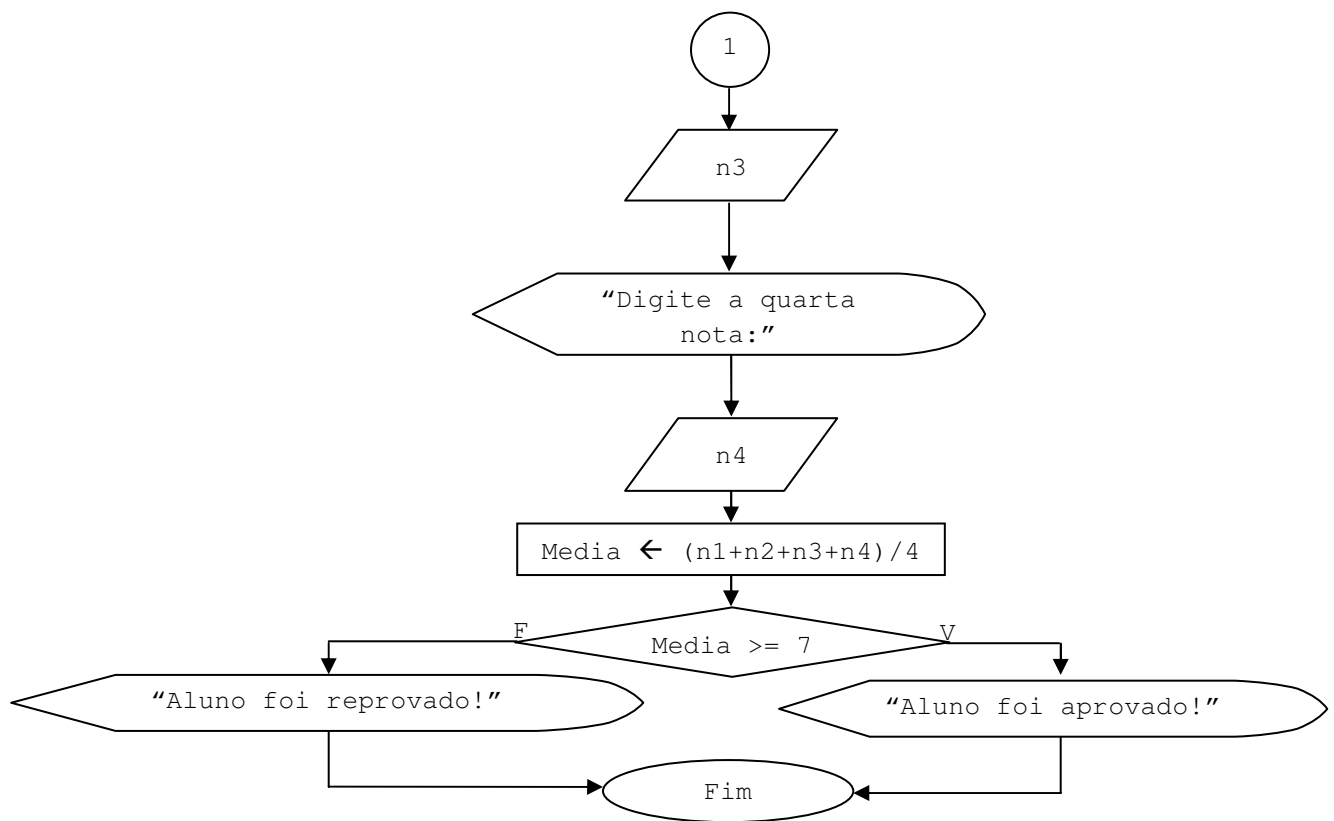




Exemplo 2: Transposição de algoritmo em DB para Pseudocódigo (situação: "Aprovado ou Reprovado")

DB "AlunoAprovadoOuReprovado"





**Algoritmo** "AlunoAprovadoOuReprovado"

**Var**

n1, n2, n3, n4, Media: real

**Inicio**

**escreva**("Digite a primeira nota: ")

**leia**(n1)

**escreva**("Digite a segunda nota: ")

**leia**(n2)

**escreva**("Digite a terceira nota: ")

**leia**(n3)

**escreva**("Digite a quarta nota: ")

**leia**(n4)

Media ← (n1+n2+n3+n4)/4

**se** (Media >= 7) **entao**

**escreva**("Aluno foi aprovado!")

**senao**

**escreva**("Aluno foi reprovado!")

**fimse**

**Fimalgoritmo**

***Exemplo 3: Transposição de algoritmo Pseudocódigo para DB***  
***(situação: "Aprovado", "Reprovado" ou de "Prova Final")***

**Algoritmo** "AlunoAprovadoOuReprovadoOuProvaFinal"

**Var**

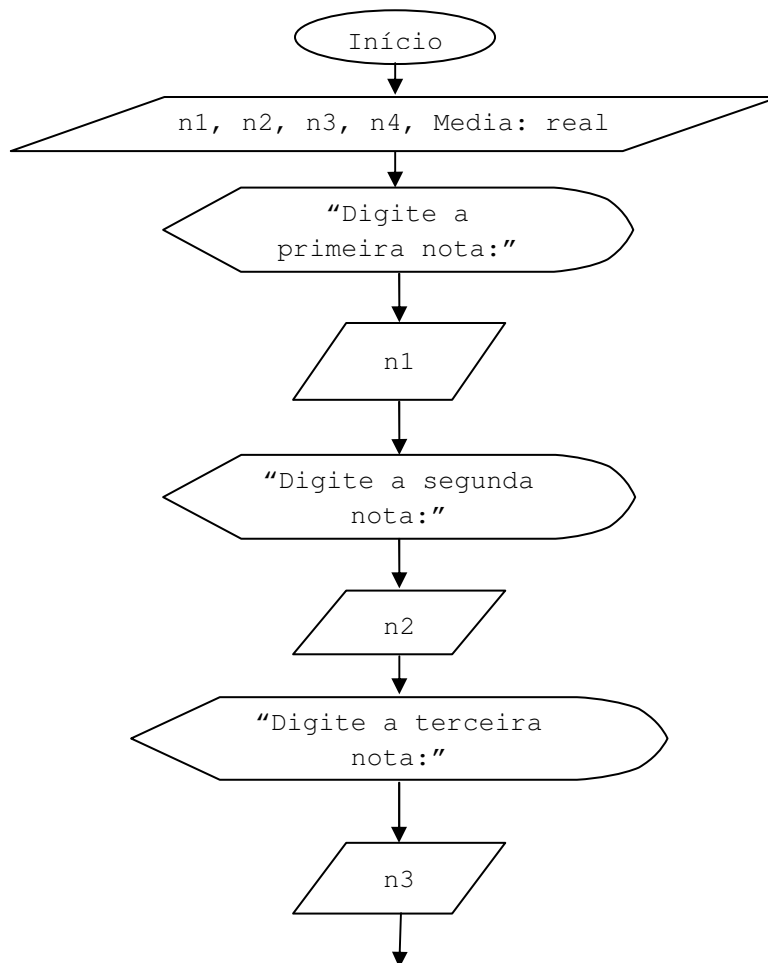


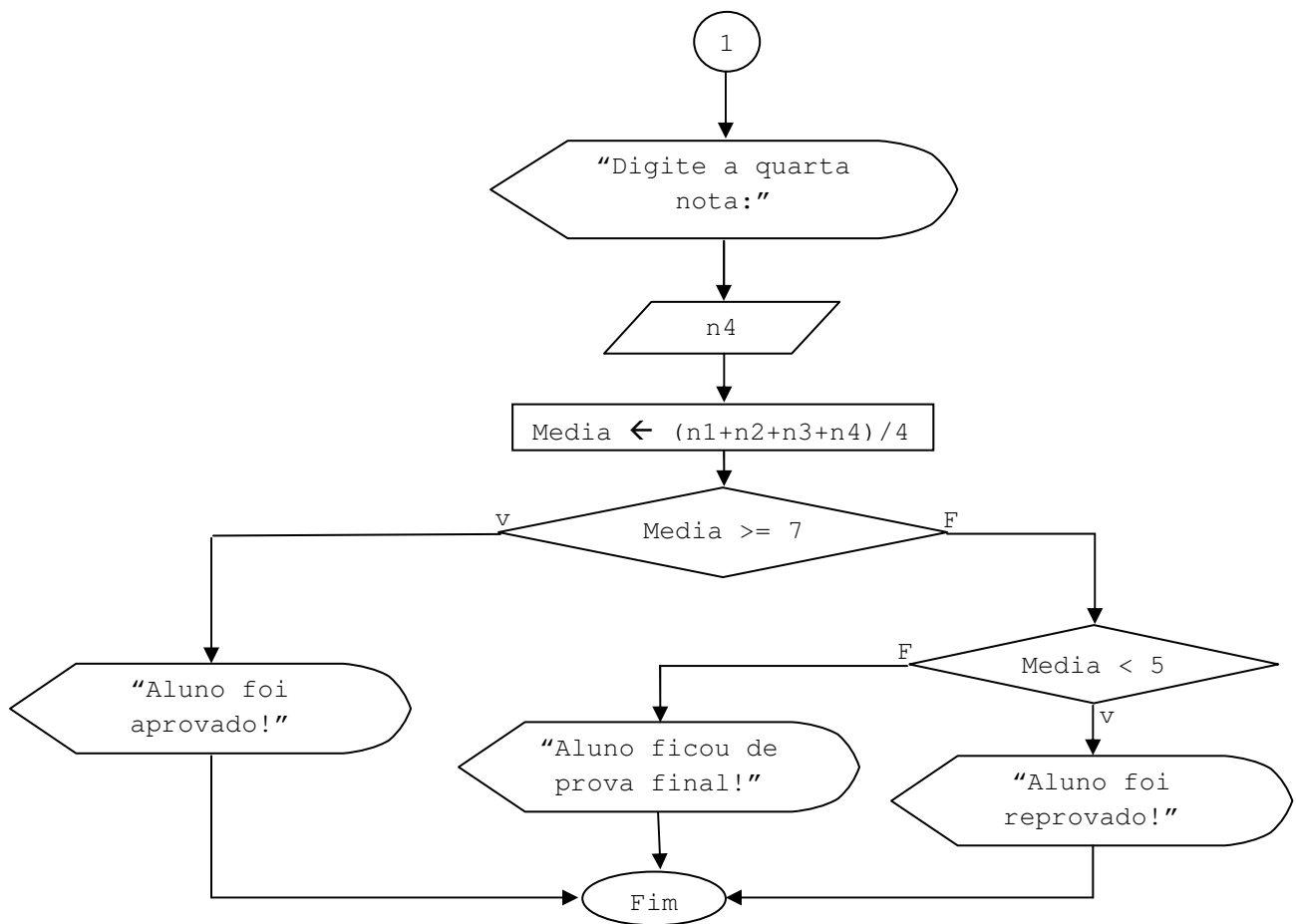
```

n1, n2, n3, n4, Media: real
Início
  escreva("Digite a primeira nota: ")
  leia(n1)
  escreva("Digite a segunda nota: ")
  leia(n2)
  escreva("Digite a terceira nota: ")
  leia(n3)
  escreva("Digite a quarta nota: ")
  leia(n4)
  Media  $\leftarrow$  (n1+n2+n3+n4)/4
  se (Media >= 7) entao
    escreva("Aluno foi aprovado!")
  senao
    se (Media < 5) entao
      escreva("Aluno foi reprovado!")
    senao
      escreva("Aluno ficou de prova final!")
  fimse
fimse
Fimalgoritmo

```

DB "AlunoAprovadoOuReprovadoOuProvaFinal"





**Exemplo 5: Algoritmo em linguagem de programação C (situação: "Aprovado", "Reprovado" ou de "Prova Final")**

//Programa para calcular média de quatro notas e apresentar situação final

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
```

```
int main(){
    float n1, n2, n3, n4, Media;

    printf("Digite a primeira nota: ");
    scanf("%f",&n1);
    printf("Digite a segunda nota: ");
    scanf("%f",&n2);
    printf("Digite a terceira nota: ");
    scanf("%f",&n3);
    printf("Digite a quarta nota: ");
    scanf("%f",&n4);
```

```
    Media = (n1+n2+n3+n4)/4;
```

```

if (Media >= 7)
    printf("\nAluno foi aprovado!");
else
    if (media < 5)
        printf("\nAluno foi reprovado!");
    else
        printf("\nAluno ficou de prova final!");
getche();
return(0);
}

```

### 6.3 Estruturas de controle de seleção

Como o próprio nome já diz, essas estruturas servem para criar uma lista de seleção em um programa. Muito utilizada quando queremos criar um *menu* de opções. É representada pelos comandos (*escolha...caso...outrocaso...fimescolha*).

*Sintaxe:*

```

escolha <expressão de seleção>
    caso <exp1>, <exp2>, ..., <expn>
        <bloco de comandos>
    caso <exp1>, <exp2>, ..., <expn>
        <bloco de comandos>
    ...
    outrocaso
        <bloco de comandos>
fimescolha

```

*Nota: Não faremos uso das estruturas de controle de seleção utilizando DB.*

Exemplo 1 em Pseudocódigo:

Algoritmo "Times"

Var

Time: caractere

Inicio

escreva("Entre com o nome de um time de futebol: ")

leia (Time)

escolha Time

**caso** "Flamengo", "Fluminense", "Vasco", "Botafogo"

escreval ("É um time carioca.")

**caso** "São Paulo", "Palmeiras", "Santos", "Corinthians"

escreval ("É um time paulista.")

**outrocaso**

escreval ("É de outro estado.")

fimescolha  
fimalgoritmo

*Exemplo 2 em Pseudocódigo:*

```
algoritmo "Opcoes"  
var  
    Opcao: caractere  
inicio  
    escreval("1 - Estudar")  
    escreval("2 - Festar")  
    escreval("3 - Namorar")  
    escreval("4 - Brincar")  
    escreva("Escolha uma opção: ")  
    leia(Opcao)  
    escolha Opcao  
    caso "1"  
        escreval("Parabéns você fez a melhor escolha: Estudar!")  
    caso "2"  
        escreval("Você escolheu Festar!")  
    caso "3"  
        escreval("Você escolheu Namorar!")  
    caso "4"  
        escreval("Você escolheu Brincar!")  
    outrocaso  
        escreval ("Opção inválida!")  
    fimescolha  
fimalgoritmo
```

*Exemplo 3 em linguagem de programação C:*

```
#include <conio.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
int main(){  
    int opcao;  
    printf("\n1 - Estudar");  
    printf("\n2 - Festar");  
    printf("\n3 - Namorar");  
    printf("\n4 - Brincar");  
    printf("\nEscolha uma opcao: ");  
    scanf("%d", &opcao);  
    switch(opcao)  
    {  
        case 1: printf("\nParabens voce fez a melhor escolha"); break;  
        case 2: printf("\nVoce escolheu festar."); break;  
        case 3: printf("\nVoce escolheu namorar."); break;  
        case 4: printf("\nVoce escolheu brincar."); break;  
        default: printf("\nOpcao invalida!!!"); break;  
    }  
    printf("\n\n");  
}
```

```
getche();  
return(0);  
}
```

## Exercícios propostos:

Resolver os exercícios abaixo utilizando *Pseudocódigo*:

1. Faça um algoritmo que descubra se um número inteiro digitado pelo usuário é divisível por 2 e 5 (ao mesmo tempo). Usar o operador **mod** em conjunto com o operador **.e.** para resolver este exercício.
2. Construa um conjunto de instruções que descubra se uma letra digitada pelo usuário é consoante ou vogal. Utilize a estrutura de seleção **escolha...caso...outrocaso...fimescolha** para resolver o algoritmo.
3. Faça um algoritmo que a partir da leitura do valor de um salário faça:
  - a) Acréscimo de 5% ao valor do salário, caso o mesmo seja maior ou igual a R\$ 1000,00;
  - b) Acréscimo de 10% ao valor do salário, caso o mesmo seja maior que R\$ 500,00 e menor que R\$ 1000,00; e,
  - c) Acréscimo de 20% ao valor do salário, caso o mesmo seja menor que R\$ 500,00.
4. Faça um algoritmo para ler os coeficiente de uma equação do segundo grau e calcular as suas raízes, na forma  $Ax^2 + Bx + C$ , levando em consideração a existência de raízes reais ( $\Delta > 0$ , a equação possui duas raízes reais e distintas;  $\Delta = 0$ , a equação possui raízes reais iguais;  $\Delta < 0$ , a equação não possui raízes reais).
5. Construa um conjunto de instruções que a partir da leitura de três números inteiros (a, b, c) apresente o **maior** entre os três.
6. Construa um conjunto de instruções que a partir da leitura de três números inteiros (n1, n2, n3) apresente o **menor** entre os três.
7. Construa um conjunto de instruções que a partir da leitura de dois números inteiros (n1, n2) imprima os valores trocados (o valor de n1 deverá ser impresso em n2 e, vice versa).
8. Construa um conjunto de instruções que a partir da leitura de três números inteiros (a, b, c) apresente-os em **ordem crescente**.

9. Construa um conjunto de instruções que a partir da leitura de três números inteiros (a, b, c) apresente-os em **ordem decrescente**.

10. Construa um conjunto de instruções que a partir da leitura de três lados (a, b, c) de um triângulo, imprima se o mesmo é equilátero (**três lados iguais**), isósceles (**dois lados iguais e um diferente**) ou escaleno (**três lados diferentes**). Verificar se os valores informados para os três lados caracterizam um triângulo, ou seja,  $(a < b+c)$  e  $(b < a+c)$  e  $(c < a+b)$ .

## UNIDADE VII

### ESTRUTURAS DE CONTROLE DE REPETIÇÃO

Os objetivos específicos dessa unidade são:

- ✓ Dotar o aluno de conhecimentos sobre estruturas de controle de repetição;
- ✓ Ensinar o aluno a realizar *loops* em um algoritmo/programa; e,
- ✓ Trabalhar com variáveis de controle em um algoritmo/programa.

#### 7.1 Conceitos iniciais

Essas estruturas são usadas para a repetição de uma quantidade de ações (processamento) dentro de um bloco de programa. São controladas por *variáveis de controle*, as quais determinam o início e o fim do laço de repetição (*loop*).

As principais estruturas de controle de repetição são: enquanto...faca...fimenquanto, repita...ate e para...ate...passo...faca...fimpara. Vejamos a seguir como funciona cada estrutura.

#### 7.2 Estrutura de controle de repetição ENQUANTO

Essa estrutura caracteriza-se por realizar o teste lógico no início do laço de repetição. Executa o bloco de programação enquanto o teste lógico for VERDADEIRO. O bloco de programação pode não ser executado nenhuma vez, caso o teste lógico for FALSO.

É representada pelo comando: enquanto...faca...fimenquanto.

*Sintaxe:*

```
enquanto (<teste_logico_VERDADEIRO>) faca  
    <bloco de instruções>  
fimenquanto
```

*Exemplo em Pseudocódigo:*

```
cont ← 1  
enquanto (cont <= 10) faca  
    escreva(cont)  
    cont ← cont+1  
fimenquanto
```

Exemplo em linguagem de programação C:

```
cont = 1;
while (cont <= 10){
    printf("\n%d", cont);
    cont++;
}
```

### 7.3 Estrutura de controle de repetição REPITA

Essa estrutura caracteriza-se por realizar o teste lógico no final do laço de repetição, no entanto, o laço é executado pelo menos uma vez. Executa o processamento enquanto o teste lógico for FALSO, e finaliza o laço quando o teste lógico for VERDADEIRO. Essa estrutura é o inverso do laço *enquanto...faca...fimenquanto*.

É representada pelo comando: repita...ate.

*Sintaxe*

```
repita
    <bloco de instruções>
ate (<teste_logico_FALSO>)
```

Exemplo em Pseudocódigo:

```
cont ← 1
repita
    escreva(cont)
    cont ← cont+1
ate (cont > 10)
```

Exemplo em linguagem de programação C:

```
cont = 1;
do{
    printf("\n%d", cont);
    cont++;
}
while (cont <= 10);
```



## 7.4 Estrutura de controle de repetição PARA

Essa estrutura caracteriza-se por trabalhar com valores finitos, ou melhor, é utilizada quando já conhecemos o início e o fim do laço. É muito utilizada para intervalos (por exemplo, 1 a 10, 1 a 100, 10 a 100, 3 a 30, etc.).

Essa estrutura é controlada por uma variável que recebe um valor inicial que pode ser incrementado de um em um, dois em dois, etc. até chegar a um valor final.

É representada pelo comando: para...ate...passo...faca...fimpara.

Nesse tipo de estrutura a variável controladora se autoincrementa, ou seja, não é preciso atribuir +1 dentro do laço para a variável.

*Sintaxe:*

```
para (<valor_inicial>) ate (<valor_final>) passo (<n>) faca  
    <bloco de instruções>  
fimpara
```

*Exemplo em Pseudocódigo:*

```
para cont de 1 ate passo 1 faca  
    escreva(cont)  
fimpara
```

O passo 1 descrito no algoritmo acima significa que a variável `cont` será incrementada de 1 em 1 até o final do laço.

*Exemplo em linguagem de programação C:*

```
for (cont = 1; cont <= 10; cont++)  
    printf("\n%d", cont);
```

## 7.5 Estruturas de controle de repetição no DB, Pseudocódigo e linguagem C

Para exemplificarmos as três estruturas de controle de repetição usaremos o problema proposto abaixo:

*Problema:* Realizar a impressão dos números inteiros de 1 a 10.

*Resolução:*

1. Criar uma variável para controlar o laço (`cont`, por exemplo);
2. Iniciar a variável `cont` com valor 1;

3. Incrementar a variável *cont* de 1 em 1 e imprimi-la até o seu valor chegar a 10.

Exemplo 1: Algoritmo utilizando a estrutura de repetição ENQUANTO

**Algoritmo** "Imprime\_01\_A\_10\_ENQUANTO"

**Var**

cont: inteiro

**Inicio**

cont ← 1

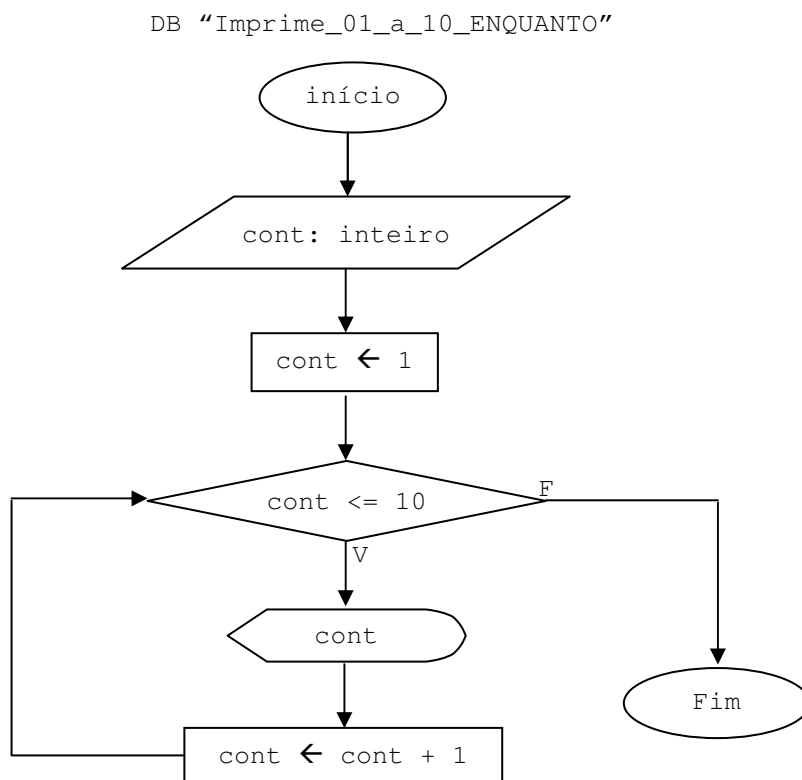
**enquanto** (cont ≤ 10) **faca**

**escreva**(cont)

    cont ← cont + 1

**fimenquanto**

**Fimalgoritmo**



Exemplo 2: Estrutura de repetição ENQUANTO em linguagem de programação C

```
//Programa para ilustrar o uso do laço de repetição ENQUANTO
```

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
```

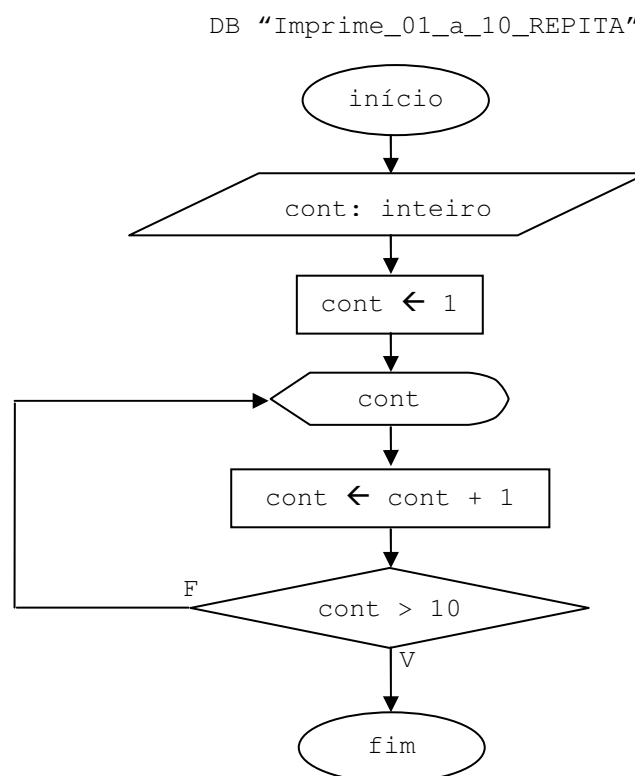
```

int main(){
    int cont = 1;

    while (cont <= 10){
        printf("\n%d", cont);
        cont++;
    }
    getch();
    return(0);
}

```

**Exemplo 3: Algoritmo utilizando a estrutura de repetição REPITA**



**Algoritmo** "Imprime\_01\_a\_10\_REPITA"

**Var**

cont: inteiro

**Inicio**

cont ← 1

**repita**

**escreva**(cont)

cont ← cont+1

**ate** (cont > 10)

**Fimalgoritmo**

#### Exemplo 4: Estrutura de repetição REPITA em linguagem de programação C

```
//Programa para ilustrar o uso do laço de repetição REPITA

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

int main(){
    int cont = 1;
    do{
        printf("\n%d", cont);
        cont++;
    }
    while (cont <= 10){

    getch();
    return(0);
}
```

#### Exemplo 5: Algoritmo utilizando a estrutura de repetição PARA

**Algoritmo** "Imprime\_01\_a\_10\_PARA"

**Var**

cont: inteiro

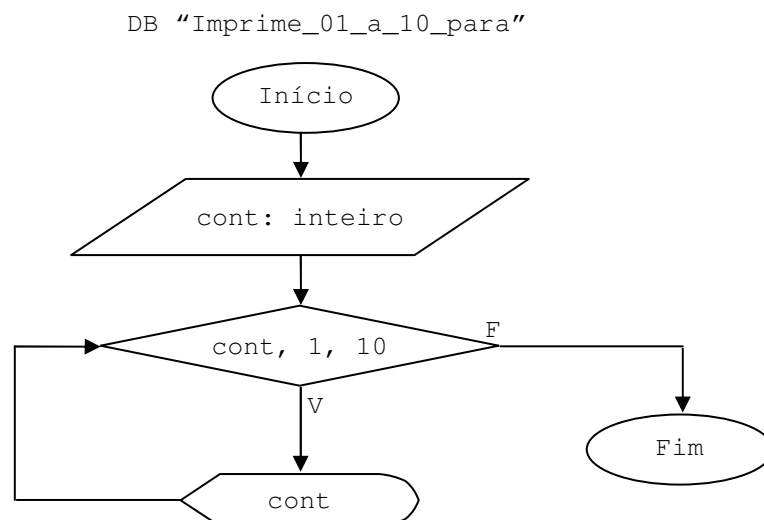
**Início**

**para** cont **de** 1 **ate** 10 **passo** 1 **faca**

**escreva**(cont)

**fimpara**

**Fimalgoritmo**



#### Exemplo 6: Estrutura de repetição PARA em linguagem de programação C

```
//Programa para ilustrar o uso do laço de repetição PARA
```

```

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

int main(){
    int cont;

    for (cont = 1; cont <= 10; cont++)
        printf("\n%d", cont);

    getch();
    return(0);
}

```

## 7.6 Laço de repetição controlado pelo usuário

Os laços apresentados acima são controlados pelo programador (com início e fim determinados por ele), no entanto, às vezes é preciso que o laço seja controlado pelo usuário do programa, ou melhor, por respostas emitidas pelo usuário ao programa.

Para exemplificarmos este tipo de laço usaremos o problema proposto a seguir.

*Problema proposto:* Imprimir números digitados pelo usuário. Dar a opção ao usuário em **continuar ou não** o programa.

Solução:

1. Pedir números para o usuário digitar os números (criar a variável *num*);
2. Imprimir esses números dentro do laço;
3. Perguntar ao usuário se ele deseja continuar (criar uma variável de *Resposta*).

*Exemplo1: Laço controlado pelo usuário com a estrutura de repetição ENQUANTO*

**Algoritmo** "Laco\_Controlado\_Pelo\_Usuario\_ENQUANTO"

**Var**

num: real

resp: caracter

**Inicio**

resp ← "S"

**enquanto** (resp = "S") **faca**

**escreva**("Digite um número: ")

**leia**(num)

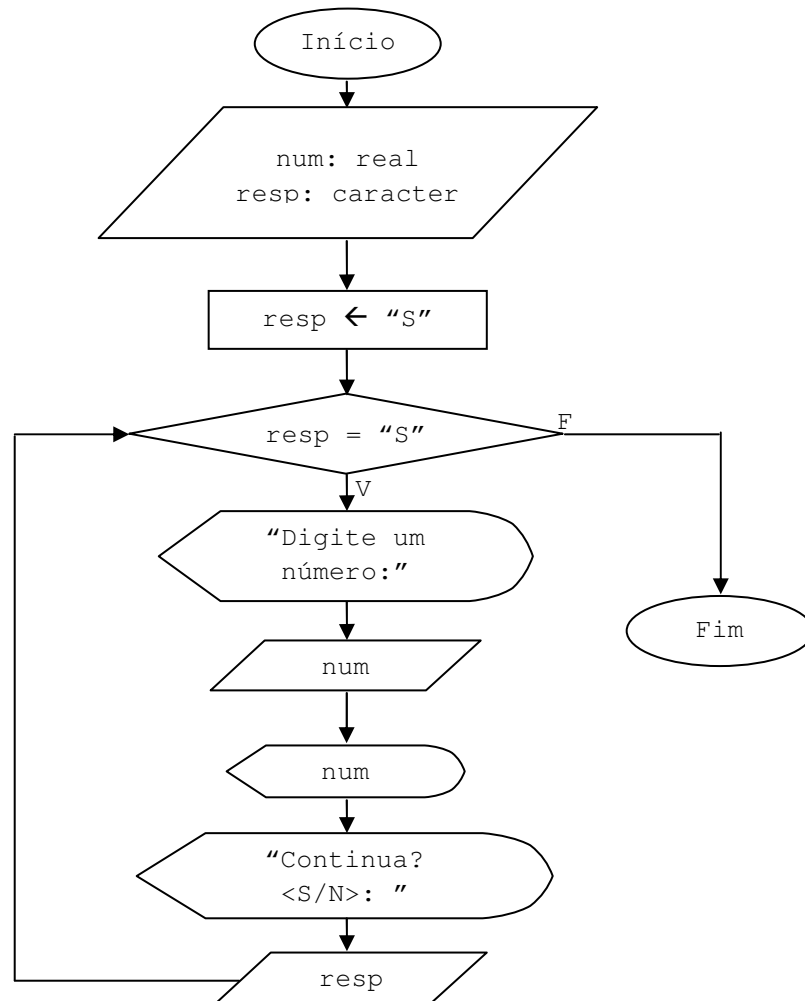
**escreva**(num)

```

    escreva("Deseja continuar? <S/N>: ")
    leia(resp)
    fimenquanto
Fimalgoritmo

```

DB "Laco\_Controlado\_Pelo\_Usuario\_ENQUANTO"



Exemplo2: Laço controlado pelo usuário com a estrutura de repetição REPITA

Algoritmo "Laco\_Controlado\_Pelo\_Usuario\_REPITA"

Var

```

    num: real
    resp: caracter

```

Inicio

repita

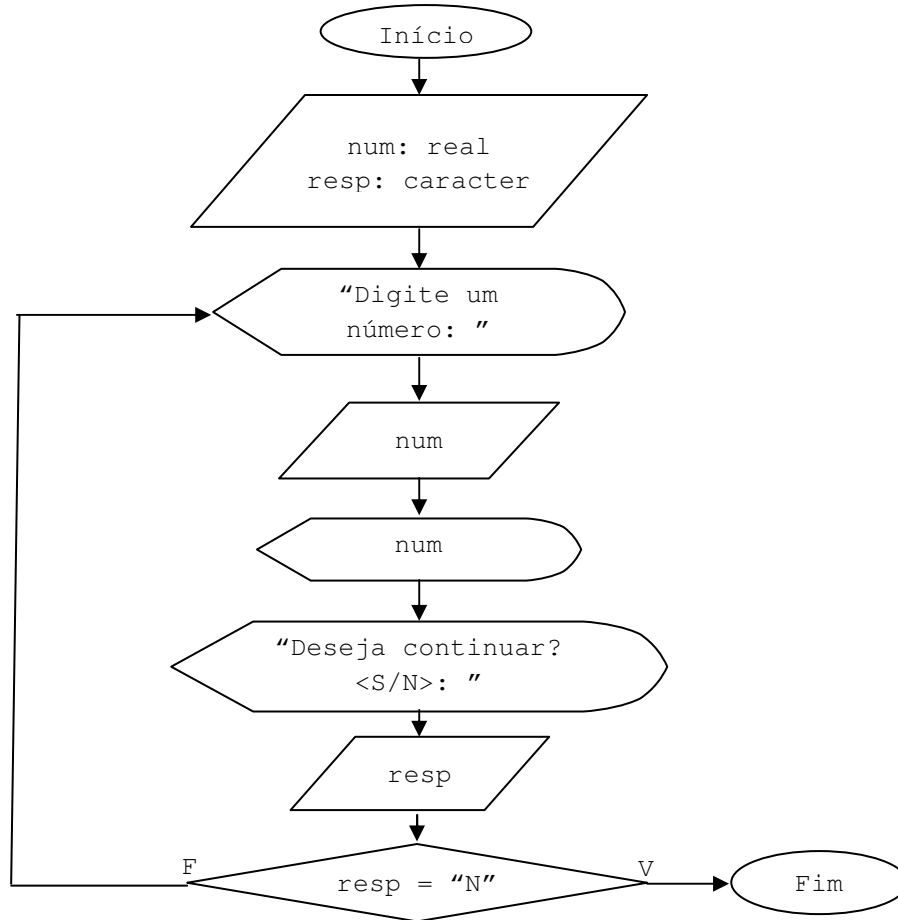
escreva("Digite um número: ")

```

    leia(num)
    escreva(num)
    escreva("Deseja continuar? <S/N>: ")
    leia(resp)
    ate(resp = "N")
Fimalgoritmo

```

DB "Laco\_Controlado\_Pelo\_Usuario\_REPITA"



### Exemplo 3: Laço controlado pelo usuário em linguagem de programação C

```

//Programa para ilustrar laço controlado pelo usuário com ENQUANTO

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

int main(){
    char resp = 'S';
    int nro;

```

```

while (resp == 'S' || resp == 's'){
    printf("Digite um numero qualquer: ");
    scanf("%d", &nro);
    printf("\n%d", nro);
    printf("\n\nContinuar? Digite S p/<Sim> ou N p/<Nao>: ");
    fflush(stdin);
    scanf("%c", &resp);
}
getche();
return(0);
}

```

## Exercícios propostos:

Os exercícios abaixo deverão ser resolvidos utilizando Pseudocódigo.

1. Crie um conjunto de instruções que a partir da leitura de um número inteiro digitado pelo usuário realize o cálculo de uma tabuada de multiplicação de 1 a 10.
2. Crie um conjunto de instruções que apresente todos os valores numéricos inteiros pares situados na faixa de 100 a 200.
3. Crie um conjunto de instruções que apresente todos os valores numéricos inteiros ímpares situados na faixa de 1 a 20.
4. Crie um conjunto de instruções que realize a leitura de **N números** digitados pelo usuário e imprima somente os números positivos. Dê a opção ao usuário de **continuar ou parar** o programa.
5. Crie um conjunto de instruções que realize a leitura de **N números** digitados pelo usuário e imprima somente os números negativos. Dê a opção ao usuário de **continuar ou parar** o programa.
6. Crie um conjunto de instruções que realize a leitura de **N números** inteiros digitados pelo usuário e calcule a média dos números inteiros positivos e a soma dos números inteiros pares. Dê a opção ao usuário de **continuar ou parar** o programa.
7. Crie um conjunto de instruções que a partir de um número inteiro digitado pelo usuário apresente todos os números que são divisíveis por ele (considerar divisão inteira, com resto 0). Exemplo: Se o usuário digitar o número **10**, o programa deverá verificar se o número é divisível por **(1, 2, 3, 4, 5, 6, 7, 8, 9 e 10)**.



8. Crie um conjunto de instruções que calcule a potência de uma base e um expoente digitado pelo usuário. Faça o cálculo considerando a **base elevada ao expoente**. Dê opção ao usuário em continuar ou parar o programa.
9. Crie um conjunto de instruções que calcule a potência de uma base e um expoente digitado pelo usuário. No entanto não é para ser usada a **base elevada ao expoente**. Faça da seguinte maneira: por exemplo,  $5^3 = 5 \times 5 \times 5 = 125$ , ou seja a base multiplicada por ela mesma três vezes (número do expoente).
10. Crie um conjunto de instruções que calcule o fatorial de um número inteiro e positivo digitado pelo usuário. Dê a opção ao usuário de **continuar ou parar** o programa.

## UNIDADE VIII

# ESTRUTURA DE DADOS HOMOGÊNEA: VETORES E MATRIZES

Os objetivos específicos dessa unidade são:

- ✓ Dotar o aluno de conhecimento sobre vetores e matrizes;
- ✓ Apresentar tipos de variáveis indexadas;
- ✓ Trabalhar com tabelas em memória; e,
- ✓ Exemplificar o uso de vetores e matrizes.

### 8.1 Conceitos iniciais

**Vetores** e **matrizes** são utilizados em programação quando queremos trabalhar com uma grande quantidade de dados em apenas uma **variável indexada**, isto quer dizer que consideramos apenas uma variável com vários índices que podem agrupar diversos dados, os quais são acessados via *loop* (laço de repetição). “Vale salientar que o agrupamento obedece sempre ao mesmo tipo de dado, e por esta razão denomina-se estrutura de dados homogênea [...]” (MANZANO & OLIVEIRA, 2011, p. 100).

### 8.2 Vetor ou matriz unidimensional

*Vetor* ou *matriz de uma dimensão* é uma tabela em memória com apenas **uma linha** e **várias colunas de dados**, seu tamanho é definido por um valor constante, ou melhor, uma *dimensão* com início e fim (conhecido como intervalo da matriz).

*Sintaxe:*

<nome\_variável>: **vetor** [<dimensão>] **de** <tipo>

Exemplo em Pseudocódigo:

medias: **vetor** [1..10] **de** **real**

Exemplo em linguagem C:

```
float medias[10];
```

### 8.2.1 Aplicação prática com vetores

Para fazermos uso de vetores tomemos como parâmetro a tabela de médias escolares abaixo:

**Tabela 5. Médias escolares**

<b>Matrícula</b>	<b>Nota 1</b>	<b>Nota 2</b>	<b>Nota 3</b>	<b>Nota 4</b>	<b>Média</b>
001	4,00	6,00	6,00	3,00	<b>4,75</b>
002	6,00	8,00	6,00	8,00	<b>7,00</b>
003	9,00	5,00	7,00	6,00	<b>6,75</b>
004	3,00	7,00	8,00	2,00	<b>5,00</b>
005	4,00	4,00	9,00	8,00	<b>6,25</b>
006	7,00	9,00	9,00	5,00	<b>7,50</b>
007	8,00	8,00	2,00	5,00	<b>5,75</b>
008	6,00	3,00	4,00	9,00	<b>5,50</b>
009	6,00	5,00	3,00	6,00	<b>5,00</b>
010	9,00	10,00	5,00	6,00	<b>7,50</b>

Fonte: Elaborado pelo autor

A partir da tabela 5, elaborar um algoritmo que calcule a média geral da turma. Com o conhecimento adquirido até aqui deverá ser elaborado um programa que faça a leitura das dez médias, em seguida a soma delas e, por último a divisão do valor da soma por 10.

#### Exemplo 1 - Algoritmo em Pseudocódigo sem vetor:

```
1 Algoritmo "MediaGeralTurma"
2 Var
3   md1, md2, md3, md4, md5, md6, md7, md8, md9, md10: real
4   MediaGeral: real
5 Inicio
6   escreva ("Digite a primeira média: ")
7   leia (md1)
8   escreva ("Digite a segunda média: ")
9   leia (md2)
10  escreva ("Digite a terceira média: ")
11  leia (md3)
```

```

12  escreva ("Digite a quarta média: ")
13  leia (md4)
14  escreva ("Digite a quinta média: ")
15  leia (md5)
16  escreva ("Digite a sexta média: ")
17  leia (md6)
18  escreva ("Digite a sétima média: ")
19  leia (md7)
20  escreva ("Digite a oitava média: ")
21  leia (md8)
22  escreva ("Digite a nona média: ")
23  leia (md9)
24  escreva ("Digite a décima média: ")
25  leia (md10)
26  MediaGeral <- (md1+md2+md3+md4+md5+md6+md7+md8+md9+md10)/10
27  escreva ("A média geral da turma é: ",MediaGeral)
28  Fimalgoritmo

```

Analizando o algoritmo acima percebamos que foram criadas dez variáveis para armazenar as médias dos alunos (md1, md2, md3 ..., md10). E se fossem cem médias? Como poderíamos melhorar o algoritmo acima utilizando a estrutura de vetores?

A resposta é bem simples, basta criarmos uma variável do tipo vetor para armazenar todas as médias em um *loop*. Vejamos a seguir como o algoritmo pode ser melhorado.

### Exemplo 2 - Algoritmo em Pseudocódigo com vetor:

```

1  Algoritmo "MediaGeralTurmaVet"
2  Var
3      medias: vetor [1..10] de real
4      MediaGeral, soma: real
5      i: inteiro
6  Inicio
7      soma <- 0
8      para i de 1 ate 10 passo 1 faca
9          escreva ("Digite a média: ")
10         leia (medias[i])
11         soma <- soma + medias[i]
12  fimpara

```

```

13   MediaGeral <- soma/10
14   escreva ("A média geral da turma é: ",MediaGeral)
15   Fimalgoritmo

```

Analisando o algoritmo acima percebemos o quanto ele foi melhorado com a criação de um variável do tipo vetor (linha 3), em relação ao primeiro algoritmo. O número de linhas foi reduzido de 28 para 15, o número de variáveis foi reduzido de 11 para 5. O programa foi otimizado economizando memória e processamento.

Na linha 3 do algoritmo com a criação de `medias: vetor [1..10] de real` foi destinado um espaço com 1 linha e dez colunas para armazenamento das médias dos alunos. O vetor `media` pode ser representado da seguinte forma:

`medias`

1	2	3	4	5	6	7	8	9	10
4,75	7,00	6,75	5,00	6,25	7,50	5,75	5,50	5,00	7,50

Observando o vetor `medias` acima, percebemos que temos os índices do vetor que vão de 1 até 10 e os elementos do vetor que são as médias dos alunos. Não devemos confundir índice com elemento. "Índice é o endereço de alocação de uma unidade da matriz, enquanto elemento é o conteúdo armazenado em um determinado endereço" (MANZANO & OLIVEIRA, 2011, p. 104).

No **algoritmo** "MediaGeralTurmaVet" os índices do vetor são representados pela variável `i` (linha 5). Na sequência entenderemos melhor como manipular os índices e os elementos de um vetor.

### Exemplo 3 - Algoritmo em linguagem C com vetor:

```

//Programa para calcular a média geral da turma

#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

int main(){
    float medias[10], MediaGeral, soma = 0;
    int i;

    for (i = 0; i < 10; i++){
        printf("Digite a %dª média da turma: ",i+1,166,130);
        scanf("%f",&medias[i]);
        soma = soma + medias[i];
    }
    MediaGeral = soma/10;

    printf("\n\nMédia geral da turma %c: %2.2f",130,130,MediaGeral);

```

```

    getch();
    return(0);
}

```

### 8.2.2 Leitura de dados em um vetor

Para armazenarmos dados em um vetor basta atribuímos dados ao vetor ou solicitar ao usuário que forneça os dados via teclado. Para ilustrar, propomos a solução do algoritmo abaixo:

*Algoritmo:* Solicitar dez números inteiros ao usuário e armazená-los em um vetor.

*Solução:*

1. Criar uma variável do tipo vetor: `NumerosInt`, por exemplo;
2. Criar uma variável de índice: `i`, por exemplo;
3. Criar um laço de repetição de 1 até 10 (utilizando a variável `i`) e solicitar os números inteiros para o usuário; e,
4. Armazenar os números inteiros na variável do tipo vetor.

Exemplo 1 – Leitura de dados de um vetor em Pseudocódigo:

```

1  Algoritmo "ArmazenaNumerosInteiros"
2  Var
3      NumerosInt: vetor [1..10] de inteiro
4      i: inteiro
5  Inicio
6      para i de 1 ate 10 passo 1 faca
7          escreva ("Digite um número inteiro: ")
8          leia (NumerosInt[i])
9      fimpara
10 Fimalgoritmo

```

Analisando o algoritmo acima temos na linha 3 a criação da variável `NumerosInt` para armazenar os dez números inteiros fornecidos pelo usuário. Na linha 4 temos a criação da variável `i` para controlar os índices de 1 a 10 (linha 6). Na linha 8 temos o armazenamento dos elementos no vetor, os quais serão

armazenados da seguinte maneira: `NumerosInt[1]`, `NumerosInt[2]`, `NumerosInt[3]`, ..., `NumerosInt[10]`.

Suponhamos que o usuário tenha digitado os números inteiros: 2, -9, 1, 0, 3, -7, 10, 8, 21 e 13. O processamento do algoritmo nas linhas 6 a 8 seria representado conforme descrito abaixo:

```
NumerosInt[1] = 2
NumerosInt[2] = -9
NumerosInt[3] = 1
NumerosInt[4] = 0
NumerosInt[5] = 3
NumerosInt[6] = -7
NumerosInt[7] = 10
NumerosInt[8] = 8
NumerosInt[9] = 21
NumerosInt[10] = 13
```

Os números ficariam dispostos no vetor da seguinte maneira:

NumerosInt									
1	2	3	4	5	6	7	8	9	10
2	-9	1	0	3	-7	10	8	21	13

### Exemplo 2 – Leitura de dados de um vetor em linguagem C:

```
//Programa para armazenar dez números inteiros em um vetor

#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

int main(){
    int NumerosInt[10], MediaGeral, soma = 0;
    int i;

    for (i = 0; i < 10; i++){
        printf("Digite o %dº número inteiro: ", i+1, 167, 163);
        scanf("%d", &NumerosInt[i]);
    }
    getch();
    return(0);
}
```

### 8.2.3 Escrita de dados em um vetor

Para lermos os dados de um vetor e apresentá-los na tela basta acessarmos os índices do vetor e apresentar o elemento de cada índice por meio de um *loop*.

Tomemos como exemplo o algoritmo “ArmazenaNumerosInteiros”, vamos modificar o algoritmo e acrescentar a apresentação dos elementos do vetor na tela. Vejamos no exemplo 1.

#### Exemplo 1 – Impressão de dados de um vetor em Pseudocódigo

```
1 Algoritmo “Armazena_e_MostraNumerosInt”
2 Var
3   NumerosInt: vetor [1..10] de inteiro
4   i: inteiro
5 Inicio
6   para i de 1 ate 10 passo 1 faca
7     escreva (“Digite um número inteiro: ”)
8     leia (NumerosInt[i])
9   fimpara
10  para i de 1 ate 10 passo 1 faca
11    escreval (NumerosInt[i])
12  fimpara
13 Fimalgoritmo
```

Para apresentar os dados do vetor bastou criar no programa mais um laço de repetição (linha 10) para acessar os elementos de cada índice e apresentá-los na tela por meio da variável `NumerosInt[i]`.

**Tabela 6. Dados armazenados no vetor NumerosInt**

Índice	Elemento
1	10
2	20
3	30
4	40
5	50
6	60
7	70
8	80
9	90
10	100

Fonte: Elaborado pelo autor

Suponhamos que o usuário do programa digitou os números: 10, 20, 30, 40, 50, 60, 70, 80, 90 e 100. O processamento do algoritmo seria representado como demonstrado na tabela 7.



Percebam na tabela 6 que os elementos do vetor `NumerosInt` são apresentados em dez linhas, isso se deve pelo fato de que na linha 11 do algoritmo “Armazena\_e\_MostraNumerosInt” utilizamos a função escreval que tem a função de deslocar os dados do vetor para outra linha, apresentando-os linha a linha.

### Exemplo 2 – Impressão de dados de um vetor em linguagem C

```
/*Programa para armazenar e imprimir dez números inteiros de um
vetor*/

#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

int main(){
    int NumerosInt[10], MediaGeral, soma = 0;
    int i;

    for (i = 0; i < 10; i++){
        printf("Digite o %d%c n°cmerno inteiro: ",i+1,167,163);
        scanf("%d",&NumerosInt[i]);
    }

    for (i = 0; i < 10; i++)
        printf("\n%d%c n°cmerno inteiro: %d",i+1,167,163,NumerosInt[i]);

    getche();
    return(0);
}
```

*Nota: Um vetor também pode ser representado por uma coluna e várias linhas de dados, tudo vai depender de como os dados serão apresentados na tela do computador pelo programa.*

### **8.2.4 Manipulação de dados alfanuméricos em um vetor**

Além de dados numéricos, podemos também trabalhar com dados alfanuméricos (letras, símbolos, caracteres especiais, etc.) em uma matriz do tipo vetor. Para tanto devemos seguir os mesmos procedimentos adotados anteriormente, apenas com uma mudança na criação da variável que será do tipo `caractere`.

*Sintaxe:*

<nome\_variável>: **vetor** [<dimensão>] **de** <**tipo**>

### Exemplo em Pseudocódigo:

nomes: vetor [1..5] de caractere

Para melhor entendimento propomos a resolução do algoritmo a seguir.

*Algoritmo:* Solicitar ao usuário a entrada de cinco nomes de pessoas e, em seguida imprimi-los na tela do computador.

### Solução:

1. Criar uma variável do tipo vetor para armazenar os cinco nomes, por exemplo: `nomes`;
2. Criar uma variável para os índices, por exemplo: `i`.
3. Criar um laço de repetição para solicitar a entrada dos nomes; e,
4. Criar um laço de repetição para mostrar os nomes na tela do computador.

Vejam abaixo o algoritmo em Pseudocódigo para representar um vetor de dados alfanuméricos.

```
1 Algoritmo "Armazena_e_ImprimeNomes"
2 Var
3   nomes: vetor [1..5] de caractere
4   i: inteiro
5 Inicio
6   para i de 1 ate 5 passo 1 faca
7       escreva ("Digite o nome da pessoa: ")
8       leia (nomes[i])
9   fimpara
10  para i de 1 ate 5 passo 1 faca
11      escreval (nomes[i])
12  fimpara
13 Fimalgoritmo
```

Analisando o algoritmo acima, podemos perceber que a estrutura do programa é exatamente igual à de manipulação de dados numéricos, a única diferença é na

criação da variável na linha 3 que é do tipo `caractere`, o que caracteriza que estamos trabalhando com dados alfanuméricos.

Suponha que o usuário tenha digitado os nomes: "João", "Maria", "Jesus", "Moisés" e "Isaac". O armazenamento na matriz nomes ficaria como apresentado na tabela 7.

**Tabela 7. Dados armazenados no vetor nomes**

Índice	Elemento
1	João
2	Maria
3	Jesus
4	Moisés
5	Isaac

Fonte: Elaborado pelo autor

### 8.3 Matriz bidimensional

Uma matriz bidimensional como o próprio nome já indica se trata de uma variável indexada para representar uma **tabela com linhas e colunas**. Para manipular os dados de um vetor utilizamos somente um *loop* (um laço de repetição), ao contrário para a manipulação de matrizes com mais de uma dimensão é necessário utilizar a quantidade de laços relativos ao tamanho da dimensão da matriz. Para uma matriz bidimensional (dois laços de repetição), para uma matriz tridimensional (três laços de repetição) e, assim sucessivamente.

Assim como nos vetores os elementos de uma matriz bidimensional são acessados por meio dos índices (i e j, por exemplo). Assim, em uma Tabela[1,2] estamos fazendo referência ao elemento armazenado na linha 1 coluna 2.

*Sintaxe:*

<nome\_variavel>: **vetor** [<dimensao1>, <dimensao2>] **de** <**tipo**>

*Exemplo em Pseudocódigo:*

tabela: **vetor** [1..8, 1..5] **de** **inteiro**

*Exemplo em linguagem C:*

```
int tabela[8][5];
```

A matriz `tabela` tem a capacidade de armazenar 40 elementos, pois foi definida como uma tabela de 8 linhas e 5 colunas como demonstrado na figura 12.

**Figura 12. Matriz tabela 8x5**

Matriz: tabela

		Colunas				
		1	2	3	4	5
Linhas	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

Fonte: Adaptado de Manzano & Oliveira (2011)

### **8.3.1 Leitura de dados em uma matriz bidimensional**

Para armazenarmos dados em uma matriz bidimensional basta criarmos a estrutura e utilizarmos dois índices (**i** - linha, **j** - coluna) que serão manipulados em dois laços de repetição.

Para facilitar a compreensão, propomos que seja realizado um algoritmo que solicite ao usuário quatro notas bimestrais de cinco alunos de uma escola.

*Solução:*

1. Criar a matriz para armazenar as notas bimestrais (por exemplo, matriz notas);
2. Definir a dimensão da matriz. Visto que são cinco alunos e quatro notas, teremos uma matriz  $5 \times 4 = 20$  elementos; e,
3. Utilizar dois laços para o armazenamento dos dados.

Exemplo em Pseudocódigo:

```
1 Algoritmo "NotasAlunos"
2 Var
3   notas: vetor [1..5, 1..4] de real
4   i, j: inteiro
5 Inicio
```

```

6   para i de 1 ate 5 passo 1 faca
7       para j de 1 ate 4 passo 1 faca
8           escreva("Digite a nota do aluno: ")
9           leia(notas[i,j])
10      fimpara
11  fimpara
12  Fimalgoritmo

```

Analisando o algoritmo acima temos dois laços de repetição (linhas 6 e 7) para caracterizar uma matriz bidimensional, ou melhor, uma matriz com 5 linhas e 4 colunas, com capacidade de armazenamento de 20 elementos.

As variáveis *i* e *j* controlam os índices linha e coluna da matriz, nesse sentido na primeira execução do loop os elementos serão armazenados nas posições [1,1], [1,2], [1,3] e [1,4] da matriz, na segunda execução do loop os elementos serão armazenados nas posições [2,1], [2,2], [2,3] e [2,4] e, assim sucessivamente.

Para melhor entendimento imaginemos que o usuário digitou as seguintes notas bimestrais para cada aluno:

```

Aluno 1: 5,00; 7,00; 8,00 e 9,00
Aluno 2: 9,00; 10,00; 7,50 e 5,00
Aluno 3: 4,00; 3,00; 2,00 e 9,00
Aluno 4: 5,00; 6,00; 7,00 e 8,00
Aluno 5: 10,00; 9,00; 10,00 e 8,50

```

A partir das notas bimestrais digitadas o comportamento do algoritmo "NotasAlunos" nas linhas 6 e 7 ficaria da seguinte maneira:

```

notas[1,1] = 5,00
notas[1,2] = 7,00
notas[1,3] = 8,00
notas[1,4] = 9,00
notas[2,1] = 9,00
notas[2,2] = 10,00
notas[2,3] = 7,50
notas[2,4] = 5,00

```

```
notas[3,1] = 4,00
notas[3,2] = 3,00
notas[3,3] = 2,00
notas[3,4] = 9,00
notas[4,1] = 5,00
notas[4,2] = 6,00
notas[4,3] = 7,00
notas[4,4] = 8,00
notas[5,1] = 10,00
notas[5,2] = 9,00
notas[5,3] = 10,00
notas[5,4] = 8,50
```

#### Exemplo em linguagem C:

```
//Programa para armazenar 4 notas de 5 alunos

#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

int main(){
    float notas[5][4];
    int i,j;

    for (i = 0; i < 5; i++){
        system("cls");
        for (j = 0; j < 4; j++){
            printf("Digite a %d%c nota do %d%c aluno: ",j+1,166,i+1,167);
            scanf("%f",&notas[i][j]);
        }
    }
    getch();
    return(0);
}
```

### **8.3.2 Escrita de dados em uma matriz bidimensional**

Para lermos os dados de uma matriz bidimensional e apresentá-los na tela basta acessarmos os índices (*i*, *j*) da matriz e apresentar o elemento de cada índice por meio de dois *loops*.

Tomemos como exemplo o algoritmo “NotasAlunos”, vamos modificá-lo e acrescentar a apresentação dos elementos da matriz bidimensional na tela. Vejamos abaixo o algoritmo modificado.

Exemplo em Pseudocódigo:

```

1 Algoritmo “NotasAlunosVisualizadas”
2 var
3   notas: vetor [1..5, 1..4] de real
4   i, j: inteiro
5 Inicio
6   para i de 1 ate 5 passo 1 faca
7     para j de 1 ate 4 passo 1 faca
8       escreva("Digite a nota do aluno: ")
9       leia(notas[i,j])
10    fimpara
11  fimpara
12  para i de 1 ate 5 passo 1 faca
13    para j de 1 ate 4 passo 1 faca
14      escreva(notas[i,j])
15    fimpara
16  fimpara
17 Fimalgoritmo

```

**Figura 13. Matriz notas 5x4**

Matriz: notas

		Colunas			
		1	2	3	4
Linhas	1	5,00	7,00	8,00	9,00
	2	9,00	10,00	7,50	5,00
	3	4,00	3,00	2,00	9,00
	4	5,00	6,00	7,00	8,00
	5	10,00	9,00	10,00	8,50

Fonte: Elaborado pelo autor

Ao final do processamento do algoritmo “NotasAlunosVisualizadas” podemos representar a matriz `notas` como demonstrado na figura 13.

### Exemplo em linguagem C:

```
//Programa para armazenar e imprimir quatro notas de cinco alunos

#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

int main(){
    float notas[5][4];
    int i,j;

    for (i = 0; i < 5; i++){
        system("cls");
        for (j = 0; j < 4; j++){
            printf("Digite a %d%c nota do %d%c aluno: ",j+1,166,i+1,167);
            scanf("%f",&notas[i][j]);
        }
    }

    for (i = 0; i < 5; i++){
        printf("\n\nNotas do %d%c aluno:\n ",i+1,167);
        for (j = 0; j < 4; j++){
            printf("\n%d%c nota: %2.2f",j+1,166,notas[i][j]);
        }

        getch();
    }
    return(0);
}
```

### **Exercícios propostos:**

Resolver os exercícios abaixo utilizando Pseudocódigo.

1. Escreva um algoritmo que faça a leitura de dez elementos de uma matriz do tipo vetor e apresente os valores lidos na tela do computador.
2. Escreva um algoritmo que faça a leitura de oito elementos de uma matriz A do tipo vetor. Construa a matriz B de mesma dimensão com os elementos da matriz A multiplicados por 3 (O elemento B[1] deve ser formado pelo elemento A[1]\*3, o elemento B[2] deve ser formado pelo elemento A[2]\*3 e, assim sucessivamente até chegar em 8). Apresentar na tela do computador os elementos da matriz B.
3. Escreva um algoritmo que faça a leitura de uma matriz A do tipo vetor com 15 elementos. Construir uma matriz B de mesmo tipo, sendo que cada



elemento da matriz B seja o fatorial do elemento correspondente da matriz A. Apresentar a matriz B na tela do computador.

4. Escreva um algoritmo que faça a leitura de duas matrizes A e B do tipo vetor com 15 elementos cada. Construir uma matriz C, sendo esta a junção das matrizes A e B. Desta forma, a matriz C terá o dobro elementos, ou seja, 30. Apresentar a matriz C na tela do computador.

5. Escreva um algoritmo que faça a leitura de 20 elementos de uma matriz A do tipo vetor e construa uma matriz B de mesma dimensão com os mesmos elementos de A, os quais devem estar invertidos, ou seja, o primeiro elemento A passa a ser o último de B, o segundo elemento de A passa a ser o penúltimo de B, e assim sucessivamente. Apresentar na tela do computador as matrizes A e B.

6. Escreva um algoritmo que faça a leitura de duas matrizes A e B, cada uma com duas dimensões com 5 linhas e 3 colunas. Construir uma matriz C de mesma dimensão, a qual é formada pela soma dos elementos da matriz A com os elementos da matriz B. Apresentar na tela do computador a matriz C.

7. Escreva um algoritmo que faça a leitura de duas matrizes A e B, cada uma com uma dimensão para 7 elementos. Construir uma matriz C de duas dimensões. A primeira coluna deve ser formada pelos elementos da matriz A e a segunda coluna pelos elementos da matriz B. Apresentar a matriz C na tela do computador.

8. Escreva um algoritmo que faça a leitura de 20 elementos de uma matriz qualquer, considerando que ela tenha o tamanho de quatro linhas por cinco colunas. Apresentar os valores da matriz na tela do computador.

9. Escreva um algoritmo que faça a leitura de uma matriz A de uma dimensão com cinco elementos. Construir uma matriz B de duas dimensões com três colunas, sendo a primeira coluna formada pelos elementos da matriz A somados com 5, a segunda coluna formada pelo valor do cálculo do fatorial de cada elemento correspondente da matriz A e a terceira e última coluna pelos quadrados dos elementos correspondentes da matriz A. Apresentar a matriz B na tela do computador.

10. Escreva um algoritmo que realize as tarefas abaixo:

- a. Crie um vetor para armazenar o nome dez alunos de uma turma;
- b. Crie uma matriz 10x4 para armazenar as quatro notas bimestrais de cada aluno;
- c. Calcule a média semestral de cada aluno e armazene em um vetor; e,
- d. A partir da média semestral indique a situação de cada aluno: Aprovado (Média > 7); Reprovado (Média < 5); Prova Final (Média < 7 e Média >= 5).

Apresente os dados na tela do computador como mostrados na tabela abaixo:

Nome	Nota 1	Nota 2	Nota 3	Nota 4	Média	Situação
1. Carlinho V. de Sousa	10,00	8,00	9,00	10,00	9,25	AP
2. Leliane V. de Souza	7,00	6,00	5,00	4,00	5,50	PF
3. Cristiano Moia	4,00	4,00	4,00	4,00	4,00	RP
4. Maria Moia	7,00	7,00	7,00	9,00	7,50	AP
5. Marcelo Berigo	4,00	6,50	5,00	3,00	4,62	RP
6. Renata Fun	8,00	8,00	8,00	8,00	8,00	AP
7. Denilson Show	5,00	5,00	5,00	5,00	5,00	PF
8. Vicente Júnior	1,00	1,00	1,00	1,00	1,00	RP
9. Tomas Khun	10,00	10,00	10,00	10,00	10,00	AP
10. Joana Dar'c de Souza	7,00	7,00	7,00	7,00	7,00	AP

## UNIDADE IX

# INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO C

Os objetivos específicos dessa unidade são:

- ✓ Dotar o aluno de conhecimentos iniciais acerca da linguagem de programação C (bibliotecas, palavras reservadas, funções etc.);
- ✓ Realizar a transposição de um algoritmo escrito em pseudocódigo para C;
- ✓ Apresentar exemplos de algoritmos em C;
- ✓ Escrever programas no ambiente *Dev C++*.

### 9.1 Breve histórico da linguagem C

Em 1972, no Laboratório da Bell, foi projetada a linguagem C por Dennis M. Ritchie (que, em 1973, escreveu uma versão do sistema operacional UNIX utilizando a linguagem C) e Brian W. Kernighan.

A linguagem C deriva de ALGOL 68 e foi baseada na estrutura da linguagem B de Ken Thompson que era uma evolução da BCPL. Alguns autores achavam que a próxima linguagem na estrutura C seria chamada de P, mas isso não aconteceu até o momento, pois a linguagem que sucedeu a C foi chamada de C++ (C Orienta a Objetos).

A linguagem C nasceu da necessidade de escrever programas que utilizassem os recursos internos de máquina de forma mais fácil que a linguagem de montagem *Assembly*.

A grande aceitação da linguagem decorre da elegância em conciliar seu poder de programação em baixo nível com seu alto grau de portabilidade, ou seja, um programa escrito em C pode ser rodado em muitas plataformas computacionais. Microcomputadores a computadores de grande porte possuem um compilador de linguagem C.

Outro ponto a ser considerado em relação à linguagem C é o fato de ter influenciado direta ou indiretamente criação de outras linguagens de programação, tais como C++, Java, JavaScript/JScript, PHP, Ajax, entre outras.

### 9.2 Características da linguagem C

Entre as principais características do C, podemos citar:

- ✓ A linguagem C é “*case sensitive*”, ou seja, difere maiúsculas de minúsculas, portanto para a linguagem o identificador `Soma` é diferente de `soma`;

- ✓ O C é uma linguagem de alto nível com uma sintaxe bastante estruturada e flexível tornando sua programação bastante simplificada;
- ✓ Programas em C são compilados, gerando programas executáveis;
- ✓ O C compartilha recursos tanto de alto quanto de baixo nível, pois permite acesso e programação direta do microprocessador. Com isto, rotinas cuja dependência do tempo é crítica, podem ser facilmente implementadas usando instruções em *Assembly*. Por esta razão o C é a linguagem preferida dos programadores de aplicativos;
- ✓ O C é uma linguagem estruturalmente simples e de grande portabilidade. O compilador C gera códigos mais enxutos e velozes do que muitas outras linguagens; e,
- ✓ Embora estruturalmente simples (poucas funções intrínsecas) o C não perde funcionalidade, pois permite a inclusão de uma farta quantidade de rotinas do usuário. Os fabricantes de compiladores fornecem uma ampla variedade de rotinas pré-compiladas em bibliotecas.

### 9.3 Bibliotecas da linguagem C

Uma biblioteca é um conjunto de rotinas prontas para serem usadas pelo programador. No quadro 9 são apresentadas as principais bibliotecas da linguagem C.

**Quadro 9. Principais bibliotecas da linguagem C**

Biblioteca	Descrição
<code>stdio.h</code>	Essa biblioteca é a mais utilizada na programação em linguagem C, pois se caracteriza por ser a biblioteca padrão. É nela que são embutidas as funções <code>printf()</code> , <code>puts()</code> , <code>gets()</code> , <code>scanf()</code> , entre outras.
<code>math.h</code>	Essa biblioteca possui as funções matemáticas usadas pela linguagem. Encontram-se funções trigonométricas, hiperbólicas, exponenciais, logarítmicas, entre outras.
<code>string.h</code>	Essa biblioteca possui as rotinas de tratamento de strings e caracteres. Nela se encontram as funções <code>strcmp()</code> e <code>strcpy()</code> , entre outras.
<code>time.h</code>	Essa biblioteca possui as funções de manipulação de data e hora do sistema.
<code>stdlib.h</code>	Essa biblioteca possui um conjunto de funções que não se enquadra em outras categorias. As funções dessa biblioteca são conhecidas como miscelânea.

ctype.h	Essa biblioteca possui várias funções para verificar caracteres individuais. Verifica se um caracter é maiúsculo ou minúsculo, se é um dígito, se é um espaço, entre outros. Nela se encontram as funções <code>isdigit()</code> , <code>islower()</code> , <code>isupper()</code> , entre outras.
conio.h	Essa biblioteca contém várias funções para manipulação da tela, ou seja, cor de fundo, cor de letra, teclado, entre outros.

Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)

Além das bibliotecas descritas no quadro 9 temos ainda outras, as quais são: `alloc.h`, `assert.h`, `bcd.h`, `bios.h`, `complex.h`, `dir.h`, `dirent.h`, `dos.h`, `errno.h`, `fcntl.h`, `float.h`, `fstream.h`, `generic.h`, `graphics.h`, `io.h`, `iomanip.h`, `iostream.h`, `limits.h`, `locale.h`, `malloc.h`, `mem.h`, `process.h`, `setjmp.h`, `share.h`, `signal.h`, `stdarg.h`, `stddef.h`, `stdiostr.h`, `stream.h`, `strstrea.h`, `sys\stat.h`, `sys\timeb.h`, `sys\types.h` e `values.h`, as quais serão usadas dependendo da necessidade do programador.

## 9.4 Palavras reservadas da linguagem C

Todas as linguagens de programação têm palavras reservadas. As palavras reservadas não podem ser usadas a não ser nos seus propósitos originais, isto é, não podemos declarar funções ou variáveis com os mesmos nomes. Como o C é "case sensitive" podemos declarar uma variável `For`, apesar de haver uma palavra reservada `for`, mas isto não é uma coisa recomendável de se fazer pois pode gerar confusão.

Apresentamos a seguir as palavras reservadas do ANSI C. Veremos o significado destas palavras à medida que elas forem utilizadas no decorrer dos estudos:

`auto`, `break`, `casechar`, `const`, `continue`, `default`, `define`, `do`, `double`, `else`, `enum`, `extern`, `float`, `for`, `goto`, `if`, `int`, `long`, `register`, `return`, `short`, `signed`, `sizeof`, `static`, `struct`, `switch`, `typedef`, `union`, `unsigned`, `void`, `volatile`, `while`.

## 9.5 Sequências de escape da linguagem C

Certos códigos de controle da tabela ASCII (como o *line feed*) ou caracteres especiais como (') possuem representação especial no C. Esta representação chama-se *escape sequence* (sequência de escape) representada por uma barra invertida (\) e um caracter. Sequências de escape são interpretadas como caracteres simples. No quadro 2 segue uma lista das principais sequências de escape usadas no C.

**Quadro 10. Sequencias de escape**

Controle/caracter	Sequência de escape	Valor ASCII
Nulo (null)	\0	00
Campainha (bell)	\a	07
Retrocesso (backspace)	\b	08
Tabulação horizontal	\t	09
Nova linha (new line)	\n	10
Tabulação vertical	\v	11
Alimentação de folha (form feed)	\f	12
Retorno de carro (carriage return)	\r	13
Aspas (")	\"	34
Apóstrofo (')	\'	39
Interrogação (?)	\?	63
Barra invertida (\)	\\	92

Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)

## 9.6 Transposição de algoritmo escrito em pseudocódigo para a linguagem C

O quadro 11 apresenta a transposição de um algoritmo escrito em pseudocódigo para a linguagem C.

**Quadro 11. Pseudocódigo para C**

Pseudocódigo	Linguagem C
<pre> 1 <b>algoritmo</b> "DivideDoisNumeros" 2 <b>var</b> 3   n1, n2, d: <u>real</u> 4 <b>inicio</b> 5   <b>escreva</b>("Digite o primeiro número: ") 6   <b>leia</b>(n1) 7   <b>escreva</b>("Digite o segundo número: ") 8   <b>leia</b>(n2) 9   d &lt;- n1/n2 10  <b>escreva</b>("A divisão é igual a: ",d) 11 <b>fimalgoritmo</b> </pre>	<pre> 1 //Programa para dividir dois números 2 #include &lt;stdio.h&gt; 3 #include &lt;conio.h&gt; 4 <b>int</b> main(){ 5   <b>float</b> n1, n2, d; 6   printf("Digite o primeiro numero: "); 7   scanf("%d",&amp;n1); 8   printf("Digite o segundo numero: "); 9   scanf("%d",&amp;n2); 10  d = n1/n2; 11  printf("A divisao e igual a: %f",d); 12  getch(); 13  <b>return</b>(0); 14 } </pre>

Fonte: Elaborado pelo autor

Analisando o algoritmo do quadro 3 em linguagem C, temos:

- Linha 1 – (//) Comentário de uma linha especificando o que o programa irá realizar;
- Linhas 2 e 3 – Chamada das bibliotecas padrão para que os comandos e as funções existentes no programa sejam reconhecidos;
- Linha 4 – `int main(){` Abertura da função principal (método principal);
- Linha 5 – Declaração das variáveis que serão utilizadas no programa;
- Linhas 6 e 8 – Chamada da função `printf()` para solicitar ao usuário a digitação dos números;
- Linha 7 e 9 – Chamada da função `scanf()` para armazenar na memória os números digitados pelo usuário;
- Linha 10 – Divisão dos dois números informados pelo usuário;
- Linha 11 – Visualização da divisão na tela;
- Linha 12 – Chamada da função `getche()` para forçar a apresentação do resultado na tela;
- Linha 13 – Chamada da função `return` para retornar zero para o método principal `int main();`
- Linha 14 – Fim do método principal.

## 9.7 Tipos de dados na linguagem C

Em C, como na maioria das linguagens, os dados são divididos em tipos: inteiro, real, caracter, etc. Esta divisão se deve basicamente ao número de bytes reservados para cada dado. Cada tipo de dado possui um intervalo de valores permitidos. Os tipos de dados estão representados no quadro 12.

**Quadro 12. Tipos de dados da linguagem C**

Tipo	Quant. de bytes	Configuração	Abrangência
<code>char</code>	1	<code>%c</code>	-128 a 127
<code>unsigned char</code>	1	<code>%c</code>	0 a 255
<code>Int</code>	2	<code>%i</code> ou <code>%d</code>	-32.768 a 32.767
<code>unsigned int</code>	2	<code>%u</code>	0 a 65.535
<code>signed int</code>	2	<code>%i</code> ou <code>%d</code>	-32.768 a 32.767
<code>short int</code>	2	<code>%hi</code>	-32.768 a 32.767
<code>unsigned short int</code>	2	<code>%hu</code>	0 a 65.535
<code>signed short int</code>	2	<code>%hi</code>	-32.768 a 32.767
<code>long int</code>	4	<code>%li</code>	-2.147.483.648 a

			2.147.483.648
<b>signed long int</b>	4	%li	-2.147.483.648 a 2.147.483.648
<b>unsigned long int</b>	4	%lu	0 a 4.294.967.295
<b>float</b>	4	%f	3,4E-38 a 3,4E+38 (seis algarismos significativos)
<b>double</b>	8	%lf	1,7E-308 a 1,7E+308 (dezoito algarismos significativos)
<b>long double</b>	10	%Lf	3,4E-4932 a 3,4E+4932

Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)

## 9.8 Tipos de operadores na linguagem C

Na linguagem C temos os operadores matemáticos ou aritméticos, os operadores de atribuição, os operadores de incremento/decremento, os operadores relacionais e os operadores lógicos.

### 9.8.1 Operadores aritméticos ou matemáticos

Na construção de programas, muitas vezes necessitamos realizar operações matemáticas como trabalhar com: equações, funções, expressões algébricas, entre outras. Para isso necessitamos fazer uso de operadores aritméticos ou matemáticos. O quadro 13 apresenta os principais operadores matemáticos ou aritméticos na linguagem C.

**Quadro 13. Operadores aritméticos ou matemáticos**

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto da divisão inteira
div(x, y)	Função da biblioteca stdlib.h para retornar o quociente da divisão inteira
pow(x, y)	Função da biblioteca math.h para cálculo da potência de uma base elevada ao expoente

Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)

O programa a seguir ilustra o uso dos operadores aritméticos. Execute-o e veja os resultados.

```
//Programa para ilustrar o uso dos operadores aritméticos.
```

```
#include <stdio.h>
```



```

#include <conio.h>
#include <stdlib.h>
#include <math.h>

int main(){

    int a=10, b=5, soma, sub, mult, divisao;
    int resto, pot;

    //Operador soma
    soma = a + b;
    printf("Soma -> %d + %d = %d",a,b,soma);

    //Operador subtração
    sub = a - b;
    printf("\n\nSubtracao -> %d - %d = %d",a,b,sub);

    //Operador multiplicação
    mult = a * b;
    printf("\n\nMultiplicacao -> %d * %d = %d",a,b,mult);

    //Divisão
    divisao = a/b;
    printf("\n\nDivisao -> %d/%d = %d",a,b,divisao);

    //Resto
    resto = a%b;
    printf("\n\nResto -> %d/%d = %d",a,b,resto);

    //Quociente
    printf("\n\nQuociente -> %d/%d = %d",a,b,div(a,b));

    //Potenciação
    pot = pow(a,b);
    printf("\n\nPotenciacao -> %d ^ %d = %d",a,b,pot);

    getch();
    return(0);
}

```

### 9.8.2 Operadores relacionais

Esses operadores são utilizados quando precisamos realizar comparações lógicas dentro de blocos de programas retornando 0 para falso ou 1 para verdadeiro. O quadro 14 apresenta os principais operadores relacionais da linguagem C.

**Quadro 14. Operadores relacionais**

Operador	Ação
>	Maior do que
<	Menor do que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual a
!=	Diferente

Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)

//Este programa ilustra o funcionamento dos operadores relacionais

```
#include <stdio.h>
```

```
int main(){
    int i, j;

    printf("\nEntre com dois numeros inteiros: ");
    scanf("%d%d", &i, &j);
    printf("\n%d == %d é %d\n", i, j, i==j);
    printf("\n%d != %d é %d\n", i, j, i!=j);
    printf("\n%d <= %d é %d\n", i, j, i<=j);
    printf("\n%d >= %d é %d\n", i, j, i>=j);
    printf("\n%d < %d é %d\n", i, j, i<j);
    printf("\n%d > %d é %d\n", i, j, i>j);
    return(0);
}
```

Execute o programa acima para ver o funcionamento dos operadores relacionais na linguagem C.

### 9.8.3 Operadores lógicos

Os operadores lógicos também são conhecidos como operadores booleanos ou operadores biestáveis, os quais podem assumir somente dois valores: *True* (Verdadeiro, no caso 1) ou *False* (Falso, no caso 0). Esses operadores são muito utilizados quando precisamos testar duas ou mais expressões dentro de um programa. O quadro 15 apresenta os principais operadores lógicos da linguagem C.

**Quadro 15. Operadores lógicos**

Operador	Ação
&&	AND (E) – Retorna valor verdadeiro, somente se todas as sentenças forem verdadeiras
	OR (OU) – Retorna valor verdadeiro se, pelo menos uma das sentenças for verdadeira.
!	NOT (NÃO) – Inverte o valor lógico da sentença, verdadeiro para falso, e vice-versa.

Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)

Usando os operadores relacionais e lógicos podemos realizar uma grande gama de testes. A tabela-verdade destes operadores é mostrada no quadro 16.

**Quadro 16. Tabela-verdade**

X	Y	X&&Y	X  Y	!X	!Y
V	V	V	V	F	F
F	V	F	V	V	F
V	F	F	V	F	V
F	F	F	F	V	V

Fonte: Elaborado pelo autor

O programa a seguir ilustra o funcionamento dos operadores lógicos. Compile-o e faça testes com valores (0 ou 1) para os identificadores i e j.

```
//Programa para ilustrar o uso dos operadores lógicos.

#include <stdio.h>

int main(){

    int i, j;

    printf("informe dois numeros(cada um sendo 0 ou 1): ");
    scanf("%d%d", &i, &j);
    printf("%d AND %d corresponde a %d\n", i, j, i&j);
    printf("%d OR %d corresponde a %d\n", i, j, i||j);
    printf("NOT %d corresponde a %d\n", i, !i);
    printf("NOT %d corresponde a %d\n", j, !j);
    return(0);
}
```

#### 9.8.4 Operadores de atribuição

Esses operadores são utilizados para a atribuição de valores para variáveis ou constantes. O quadro 17 apresenta os principais operadores de atribuição na linguagem C.

**Quadro 17. Operadores de atribuição**

Operador	Ação
=	Operador de atribuição (atribui um valor a variável).
+=	Operador de atribuição e soma ao mesmo tempo.
-=	Operador de atribuição e subtração ao mesmo tempo.
/=	Operador de atribuição e divisão ao mesmo tempo.
*=	Operador de atribuição e multiplicação ao mesmo tempo
%=	Operador de atribuição e resto de divisão inteira

	ao mesmo tempo.
++	Operador de atribuição e incremento ao mesmo tempo.
--	Operador de atribuição e decremento ao mesmo tempo.

*Fonte: Guilherme & Zago (2003); Godoy (2007?); Manzano (2008)*

O programa a seguir ilustra o uso dos operadores de atribuição execute e veja o funcionamento desses operadores.

```
//Programa para ilustrar o uso dos operadores de atribuição.
#include <stdio.h>
#include <iostream.h>

int main(){

    int a=100, b=20, c=10, d=2;

    //Uso do operador de atribuição
    printf("a = %d e b = %d", a, b);
    printf("\na = b -> a recebe o valor de b -> a = %d\n", b);
    a = b;

    //Uso do operador de atribuição +=
    printf("\nc = %d e d = %d", c,d);
    printf("\nc += d -> c = c + d -> c = %d\n", c+=d);

    //Uso do operador -=
    printf("\nc = %d e d = %d", c,d);
    printf("\nc -= d -> c = c - d -> c = %d\n", c-=d);

    //Uso do operador /=
    printf("\nc = %d e d = %d", c,d);
    printf("\nc /= d -> c = c/d -> c = %d\n", c/=d);

    //Uso do operador *=
    printf("\nc = %d e d = %d", c,d);
    printf("\nc *= d -> c = c * d -> c = %d\n", c*=d);

    //Uso do operador %=
    printf("\nc = %d e d = %d", c,d);
    printf("\nc %= d -> c = c%cd -> c = %d\n", 37, c%=d);

    //Uso do operador ++
    printf("\nc = %d e d = %d", c,d);
    c++;
```

```

d++;
printf("\nc++ = c + 1 -> c = %d", c);
printf("\nd++ = d + 1 -> d = %d\n", d);

//Uso do operador --
printf("\nc = %d e d = %d", c,d);
c--;
d--;
printf("\nc-- = c - 1 -> c = %d", c);
printf("\nd-- = d - 1 -> d = %d", d);

printf("\n\n");
system("pause");
return(0); }

```

## Exercícios propostos

1. Escreva um programa em C para realizar a média aritmética de quatro notas digitadas pelo usuário.
2. Escreva um programa em C que realize a soma e a multiplicação de três números inteiros digitados pelo usuário utilizando o conceito de propriedade distributiva. Imprimir os resultados de dois em dois.

Exemplo: números digitados: 1, 2, 3

Resultados:

$1 + 2 = 3$

$1 * 2 = 2$

**Imprimir (3,2)**

$1 + 3 = 4$

$1 * 3 = 3$

**Imprimir (4,3)**

$2 + 3 = 5$

$2 * 3 = 6$

**Imprimir (5,6)**

3. Crie um programa em C que a partir da entrada de quatro números calcule e imprima a média ponderada, sabendo que os pesos são, respectivamente: 2, , 3, 5 e 7.

4. Crie um programa em C que imprima o nome completo de cinco pessoas.
5. Crie um programa em C que solicite uma temperatura em graus Fahrenheit e apresente-a convertida em graus Celsius. A forma de conversão de temperatura a ser utilizada é  $C = (F - 32) * 5/9$ , em que a variável F representa a temperatura em graus Fahrenheit e a variável C, a temperatura em graus Celsius.
6. Crie um programa em C que a partir da leitura de um número inteiro de três casas imprima somente o algarismo da casa das dezenas. Exemplo: se for digitado o número 135, deverá ser impresso o número 3 da casa das dezenas
7. Antes do racionamento de energia ser decretado, quase ninguém falava em *quilowatts*; mas, agora, todos incorporaram esta palavra em seu vocabulário. Sabendo-se que 100 *quilowatts* de energia custam um sétimo do salário mínimo, fazer um programa em C que receba o valor do salário mínimo e a quantidade de *quilowatts* gasta por uma residência. O programa deverá calcular e imprimir:
  - O valor em reais de cada *quilowatt*;
  - O valor em reais a ser pago pela residência; e,
  - O novo valor a ser pago por essa residência com desconto de 10%.
8. Crie um programa em C que a partir da entrada de dois números reais fornecidos pelo usuário ofereça as opções abaixo:
  - Soma;
  - Subtração;
  - Multiplicação;
  - Divisão;
  - Potenciação.

Mostrar o resultado da opção escolhida pelo usuário na tela.

## REFERÊNCIAS

- GUILHERME, G.; ZAGO, R. L. **Apostila da linguagem C**. UNIOESTE - Universidade Estadual do Oeste do Paraná, SEMINC 2003. Disponível em: <<http://200.201.81.50/~guilherme/cursos/c.pdf>>. Acesso em: 13 ago. 2016.
- GUIMARÃES, A. de M.; LAGES, N. A. de C. **Introdução à ciência da computação**. Rio de Janeiro: LTC – Livros Técnicos e Científicos Ed., 1992.
- GODOY, D. L. **Fundamentos de Linguagem C**. Disponível em: <[http://www.dca.ufrn.br/~xamd/dca0800/apostila\\_C.pdf](http://www.dca.ufrn.br/~xamd/dca0800/apostila_C.pdf)>. Acesso em: 13 ago. 2016.
- KUHN, T. S. **A estrutura das revoluções científicas**. trad. Beatriz Vianna Boeira e Nelson Boeira. 12. ed. São Paulo: Perspectiva, 2013.
- LOPES, A.; GARCIA, G. **Introdução à programação**: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- MANZANO, A. L. N. G.; MANZANO, M. I. N. G. **Estudo dirigido de informática básica**. 7. ed. rev. atual. e ampl. São Paulo: Érica, 2007.
- MANZANO, J. A. N.; OLIVEIRA, J. F. **Estudo dirigido de algoritmos**. 14. ed. rev. São Paulo: Érica, 2011.
- MANZANO, J. A. N. G. **Estudo dirigido de linguagem C**. 12. ed. São Paulo: Érica, 2008.
- PAULO, D. A. de C. **Algoritmos I**. [S.l.: s.n.], [2007?].
- RANGEL, J. L. **Processadores de linguagens**: compiladores, interpretadores, máquinas virtuais. Disponível em: <<http://www-di.inf.puc-rio.br/~rangel/lp/LP2.PDF>>. Acesso em: 23 de março de 2011.
- SALVETTI, D. D.; BARBOSA, M. L. **Algoritmos**. São Paulo: Pearson Makron Books, 1998.
- SOUZA, C. M. **VisuAlg** – ferramenta de apoio ao ensino de programação. Revista TECEN, v. 2., nº 2, pp. 1-9, Set, 2009.
- VELLOSO, F. C. **Informática**: conceitos básicos. 7. ed. rev. e atualizada. Rio de Janeiro: Elsevier, 2004.
- ZIVIANI, N. **Projeto de algoritmos com implementações em Pascal e C**. 2. ed. São Paulo: Pioneira Thomson, 2004.