

**Faculdade São Francisco de Assis – FSFA**

**APOSTILA DISCIPLINA**

**Disciplina EAD**

**Algoritmos e Programação**

**Everaldo Daronco**

**Dezembro de 2020.**

## Sumário

1	INTRODUÇÃO .....	5
1.1	ALGORITMOS .....	7
1.2	ATIVIDADE DE PESQUISA 1 .....	7
1.3	LINGUAGEM DE PROGRAMAÇÃO .....	7
1.4	ATIVIDADE DE PESQUISA 2 .....	9
1.5	JOGO DAS GAVETAS.....	9
1.6	ALGORITMOS COMPUTACIONAIS.....	10
2	FORMAS DE REPRESENTAÇÃO DE ALGORITMOS COMPUTACIONAIS .....	11
2.1	FLUXOGRAMA.....	11
2.2	PSEUDOCÓDIGO.....	12
2.2.1	ESTRUTURA BÁSICA PSEUDOCÓDIGO.....	14
2.2.2	DECLARAÇÃO DE VARIÁVEIS .....	14
2.2.3	COMANDOS DE ENTRADA E SAÍDA .....	15
2.2.4	COMANDOS DE ATRIBUIÇÃO .....	15
2.2.5	OPERADORES .....	16
2.2.6	FUNÇÕES MATEMÁTICAS E PRECEDÊNCIA DAS OPERAÇÕES .....	16
2.2.7	ESTRUTURAS DE SELEÇÃO (CONDICIONAL) .....	17
2.2.8	ALGORITMO DE EXEMPLO E TESTE DE MESA .....	18
2.3	LISTA DE EXERCÍCIOS.....	20
2.4	DESAFIOS (INSTRUÇÕES SEQUENCIAIS E CONDICIONAIS).....	21
3	LINGUAGEM C.....	23
3.1	CARACTERÍSTICAS E ESTRUTURA BÁSICAS.....	23
3.2	BIBLIOTECAS DE FUNÇÕES.....	24
3.3	DECLARAÇÃO DE VARIÁVEIS .....	24
3.4	DESAFIO .....	27
4	OPERADORES EM C.....	29
4.1	Operadores Aritméticos.....	29
4.1.1	OUTRAS FUNÇÕES.....	30
4.2	OPERADORES RELACIONAIS.....	32
4.3	OPERADORES LÓGICOS .....	33
4.3.1	OPERADORES LÓGICOS NA LINGUAGEM C.....	34
4.4	PRIORIDADE ENTRE OS OPERADORES .....	35

---

4.5	OUTROS EXEMPLOS DE OPERADORES EM.....	35
4.6	FUNÇÕES MATEMÁTICAS.....	35
4.7	OPERADOR CAST EM C (TYPE CASTING) .....	37
4.8	LISTA DE EXERCÍCIOS.....	37
4.9	DESAFIOS.....	38
5	FUNÇÕES DE ENTRADA E SAÍDA .....	39
5.1	FUNÇÃO PRINTF().....	39
5.2	FUNÇÃO SCANF() .....	43
6	ESTRUTURAS CONDICIONAIS .....	45
6.1	ESTRUTURA CONDICIONAL SIMPLES .....	45
6.2	ESTRUTURA CONDICIONAL COMPOSTA .....	45
6.3	ESTRUTURA CONDICIONAL ENCADEADA.....	46
6.4	OPERADOR “?” .....	48
6.5	ESTRUTURA CONDICIONAL MULTIPLA ESCOLHA (SWITCH) .....	49
6.6	LISTA DE EXERCÍCIOS.....	50
6.7	DESAFIOS.....	51
7	GERAÇÃO DE NÚMEROS ALEATÓRIOS.....	54
7.1	<b>Mão na Massa: JOGO DE ADIVINHAR NÚMEROS.....</b>	55
7.2	DESAFIOS.....	55
8	LAÇOS DE REPETIÇÃO.....	57
8.1	INSTRUÇÃO FOR() .....	57
8.2	FOR Encadeado .....	60
8.3	LISTA DE EXERCÍCIOS.....	60
8.4	DESAFIOS.....	61
9	INSTRUÇÃO WHILE() .....	62
9.1	LISTA DE EXERCÍCIOS.....	65
9.2	DESAFIO .....	65
10	ESTRUTURA DE DADOS HOMOGÊNEAS .....	66
10.1	VETORES.....	66
10.2	Mão na Massa.....	68
10.3	LISTA DE EXERCÍCIOS.....	68
10.4	DESAFIOS.....	69
11	MATRIZES.....	72

---

---

11.1	String .....	73
11.2	LISTA DE EXERCÍCIOS.....	73
11.3	DESAFIO .....	74
12	ESTRUTURA DE DADOS HETEROGÊNEAS .....	76
12.1	LISTA DE EXERCÍCIOS.....	77
13	REFERENCIAS.....	78

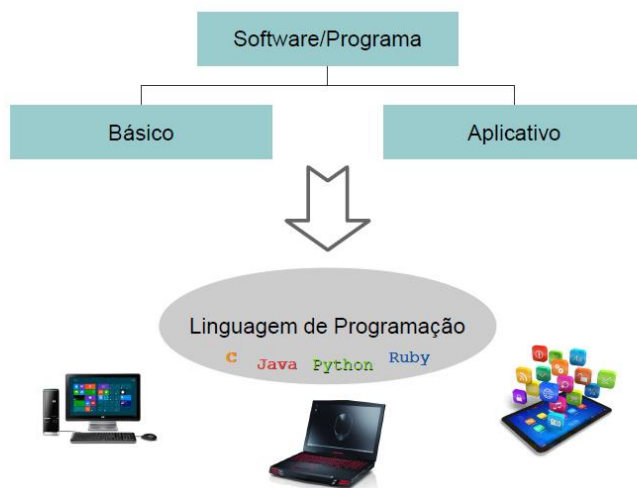
## 1 INTRODUÇÃO

Neste capítulo serão introduzidos os principais conceitos sobre algoritmos e suas aplicações, bem como os conceitos que envolvem a programação de computadores.

Assim sendo o principal objetivo da disciplina é apresentar aos alunos os princípios e fundamentos da programação de computadores para o bom desenvolvimento de programas de computador por meio dos estudos de algoritmos básicos e sua correção utilizando uma linguagem de programação.

Uma das primeiras questões que temos que analisar são os tipos de softwares que um profissional da computação pode programar. A figura 1 apresenta uma tipologia básica dividindo os programas em dois grandes grupos. O primeiro são os softwares básicos que compreendem aqueles que realizam o gerenciamento do hardware e são considerados aqueles que são essenciais para o funcionamento do computador. Aqui podemos citar como exemplo os Sistemas Operacionais. Outra categoria são os softwares aplicativos que são aqueles programas voltadas para resolver os problemas do usuário final, podendo ser aqueles de uso geral, como por exemplo: editores de texto, planilhas eletrônicas; ou aqueles de uso específico, como por exemplo: sistemas de controle de estoque, folha de pagamento, ERP, etc. Conforme a Figura 1 todos estes tipos de softwares podem ser escritos a partir de diversas linguagens de programação, algumas destas linguagens são mais adequadas para softwares básicos e outras para aplicativos.

**Figura 1: Tipologia Softwares**

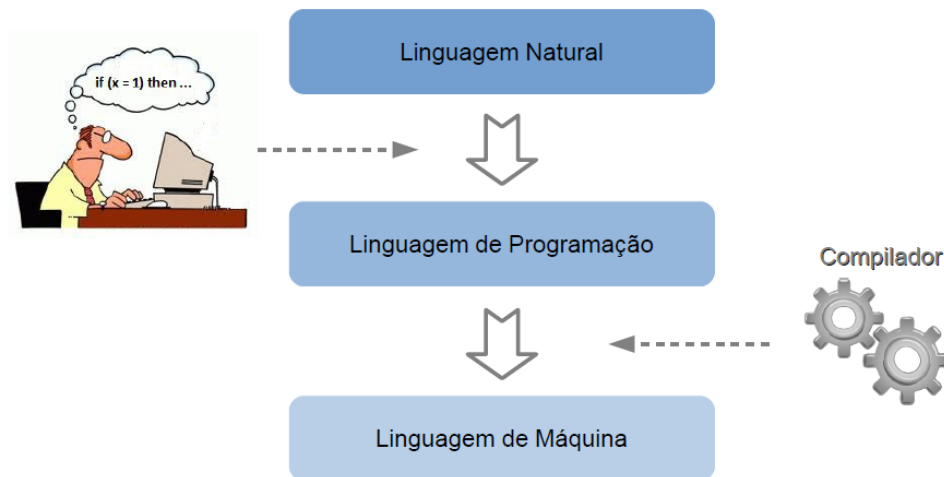


*Figura 1: Tipologia Softwares*

A figura 2 apresenta de forma simples o processo de codificação de um problema do mundo real para transformar uma linguagem natural (aquela linguagem que os seres humanos entendem) para uma linguagem de programação. Isso envolve uma série de habilidades do programador em abstrair o problema e convertê-lo dentro da sintaxe da linguagem de programação. Após está conversar entre o mundo real e a linguagem de programação o programa deve ser “compilado” por meio de um software chamado compilador. Em termos gerais o compilador realiza uma verificação do código escrito com o

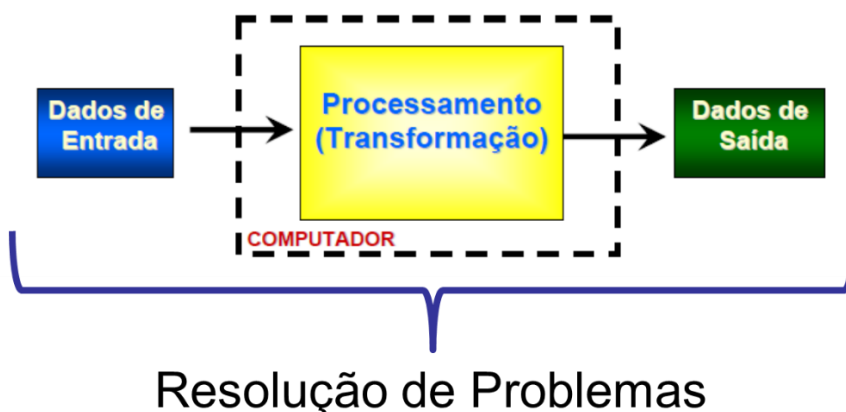
objetivo de analisar instrução por instrução e verificar se tais instruções foram escritas dentro da sintaxe da linguagem, bem como fazer a tradução das instruções para uma linguagem que o computador entenda. Esse processo então faz a tradução de uma linguagem de alto nível para uma linguagem de baixo nível.

**Figura 2: Processo de Tradução Alto Nível a Baixo Nível**



*Figura 2: Processo de tradução Alto nível → Baixo Nível*

Neste contexto, devemos entender que a habilidade de traduzir um problema do mundo real para uma solução em programa de computador constitui em uma abordagem de resolução de problemas que deve seguir o fluxo proposto na figura 3. A primeira fase do fluxo constitui na entrada de dados, neste momento todo e qualquer problema deverá ter como entrada algumas informações pertinentes para que possa ser resolvido. A próxima fase é a transformação das informações coletadas, ou o processamento, onde deve ocorrer as operações de análise e comparação das informações de entrada. No caso da computação estas operações serão realizadas pelos computadores. A última fase é a saída das informações processadas.



*Figura 3: Resolução e problemas*

Diante desse processo de resolução de problemas, podemos determinar alguns passos lógicos a serem seguidos, a saber:

1. Entendimento do Problema
2. Criação de uma sequência de operações para solução do problema
3. Execução desta sequência (executada pelo computador)
4. Verificação da adequação da solução

## 1.1 ALGORITMOS

Neste capítulo iremos abordar o tema relacionado aos algoritmos e os principais conceitos que estão relacionados nesta temática. Leia atentamente e busque complementar seus estudos com pesquisas na bibliografia indicada.

Algoritmo é uma sequência finita e bem definida de passos que, quando executados, realizam uma tarefa específica ou resolvem um problema ou indicam que a solução não pode ser obtida ou resumidamente como uma sequência de passos que visam atingir um objetivo bem definido (FORBELLONE, 2005).

Ex: Receita de bolo, manuais com instruções de uso, demonstrações de cálculo de juros, contas de luz, água, etc....

Qual o algoritmo que seguimos quando resolvemos uma equação de 2º grau? Tente pensar como resolver esta equação. Muito provavelmente você pensará na fórmula de Bhaskara representada da seguinte forma:

$$x = \frac{-b \pm \sqrt{b^2 - 4.a.c}}{2.a}$$

Esta fórmula determina a forma de resolver uma equação de 2º Grau, dando a resolução do problema em específico, basta substituir os termos a, b e c e teremos a solução.

Assim, podemos dizer que os algoritmos possuem algumas Características, quais sejam elas:

1. Ter início e fim;
2. Sequência ordenada de ações;
3. Não pode haver ambiguidade (dupla interpretação);
4. Gerar informações de saída;
5. Ser efetivo (todas as etapas devem ser alcançáveis em um tempo finito);
6. Base para codificação em qualquer linguagem de programação.

## 1.2 ATIVIDADE DE PESQUISA 1

Pesquisar na bibliografia da disciplina e em sites acadêmicos conceitos de Algoritmos e suas características/propriedades. Faça a pesquisa e analise os conceitos pesquisados para determinar os contrastes e semelhanças.

## 1.3 LINGUAGEM DE PROGRAMAÇÃO

É importante ter verificamos a diferença entre algoritmos e uma linguagem de programação (LP). Linguagem de Programação são ambientes de programação que disponibilizam instruções

computacionais de alto nível para traduzir os algoritmos em linguagem de máquina (compilação e interpretação) e consiste na Sintaxe (gramática) e semântica (significado) utilizados para escrever (codificar) um programa (FARRER, 2010).

As linguagens de programação possuem algumas características básicas, a saber:

- 1) Rigidez Sintaxe: padrões de construções das instruções; Instruções são limitadas; Palavras reservadas;
- 2) Rigidez Semântica: O computador apenas executa o que foi programado; Os passos devem ser exatos e não ambíguos para chegarmos ao resultado esperado; Linguagem Natural não é adequada.

Assim, podemos depreender que:

### Algoritmos + LP = Programa de Computadores

As linguagens de programação podem ser classificadas segundo a sua proximidade com a linguagem de máquina. Quanto maior a semelhança com a linguagem de máquina, mais baixo é o nível da linguagem (Linguagens de Baixo Nível). Analogamente, linguagens de programação “distantes” da linguagem de máquina são conhecidas como linguagens de programação de alto nível. Alguns exemplos de linguagens de programação são: Pascal, C, C++, C#, Java, PHP, Python, entre outras.

A Figura 4 ilustra um programa na linguagem C com o objetivo de multiplicar dois números quaisquer, sendo que podemos identificar claramente as fases do processo de resolução e problemas (Figura 3). Nas linhas 7 e 8 temos a entrada de dados, na 10 o processamento e na linha 12 a saída.

```
1 #include <stdio.h>
2
3 int main() {
4     int n1,n2,m;
5
6     /*ler n1 e n2*/
7     scanf("%d",&n1);
8     scanf("%d",&n2);
9     /*multiplicar n1 e n2 e armazenar em m*/
10    m = n1*n2;
11    /*Escrever o resultado da multiplicação na tela*/
12    printf("%d\n",m);
13    return 0;
14 }
15
```

Figura 4: Programa em C (Exemplo)

Esta disciplina irá abordar o paradigma da programação Estruturada que é a técnica de construir e formular algoritmos de uma forma sistemática. Consiste numa metodologia de projeto de programas visando facilitar a escrita, a leitura, a verificação a priori, a manutenção e a modificação de programas. Os algoritmos estruturados utilizam em sua sintaxe um número muito limitado de instruções e estruturas básicas (sequência, seleção e repetição), que correspondem a formas de raciocínio intuitivamente óbvias. Solução implementada através de ações, executadas sequencialmente (FARRER, 2010).



#### 1.4 ATIVIDADE DE PESQUISA 2

Pesquisar na bibliografia da disciplina o que é um programa de computador e como este é executado pelo computador, identificando as etapas para que o computador possa executar um programa em linguagem de alto nível: compilação, interpretação, programas em Java (bytecode, JVM).

#### 1.5 JOGO DAS GAVETAS

A seguir é apresentada uma forma lúdica de explicar o funcionamento do processamento de um programa na memória principal de um computador.

Algumas informações básicas que temos que ter para o “conceito” de Gavetas:

- 1) São posições de memória
- 2) Possuem nome
- 3) Possuem Endereço
- 4) Tem Capacidade limitada (quantidade e tipo) - números internos ou reais, texto, etc...
- 5) O conteúdo pode ser lido e modificado

Vamos para um exemplo de soma de dois números inteiros:

- 1) Considere as gavetas a e b;
- 2) Atribua 20 para a e 30 para b;
- 3) Some a com b e coloque o resultado em b

Assim, podemos fazer uma analogia, sendo que as gavetas poderiam ser chamadas de variáveis e os passos de 1 a 3 são considerados o algoritmo que realiza a operação de soma entre dois números.

Podemos reescrever os passos anteriores para uma forma mais algorítmica:

- 1) Sejam a e b variáveis inteiras
- 2) Faça  $a \leftarrow 20$  e  $b \leftarrow 30$
- 3) Faça  $a \leftarrow a + b$

A figura 5 apresenta os passos 1 a 3 e como os valores das variáveis (gavetas) são alterados a cada execução.

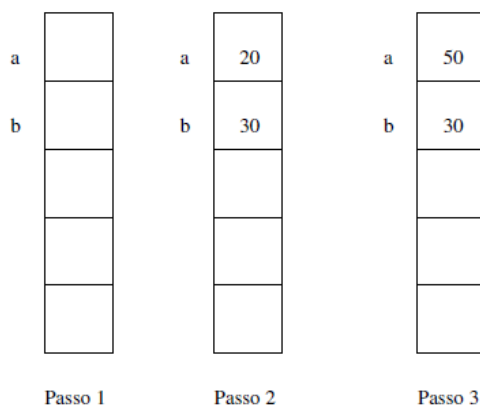


Figura 5: Jogo das Gavetas

## 1.6 ALGORITMOS COMPUTACIONAIS

Até o momento foram apresentados os passos de resolução de problemas de propósito geral. De agora em diante iremos abordar os conceitos e resolução de problemas por meio dos algoritmos computacionais (AC). AC são passos lógicos que serão executados pelo computador e que auxiliam na concepção e resolução de problemas (independente da linguagem de programação) (FARRER, 2005). Algumas limitações podem ser elencadas como por exemplo o poder de expressão que é limitado pelo conjunto de instruções que são disponibilizados.

Alguns conceitos básicos são essenciais para a construção dos algoritmos computacionais como:

- 1) Estruturas de dados (estruturas de armazenamento)
- 2) Estrutura de controle (ações)

Para a elaboração dos algoritmos computacionais devemos seguir algumas diretrizes, a saber:

- 1) Identificação do Problema: determinar o que se quer resolver ou qual objetivo a ser atingido;
- 2) Identificação das “entradas do sistema”: quais informações estarão disponíveis (serão fornecidas);
- 3) Identificação das “saídas do sistema”: quais informações deverão ser geradas/calculadas como resultado;
- 4) Definir os passos a serem realizados: determinar as sequências das ações que levem à solução do problema (transforme as entradas nas saídas):
  - a. Identificar as regras e limitações do problema;
  - b. Identificar as limitações do computador;
  - c. Determinar as ações possíveis de serem realizadas pelo computador.
- 5) Concepção do algoritmo: registrar a sequência de comandos, utilizando uma das formas de representação de algoritmos.
- 6) Teste da solução: execução manual de cada passo do algoritmo, seguindo o fluxo estabelecido, para detectar possíveis erros.

Exemplo: Calcular a média final, sendo que foram realizadas duas provas. A média final será calculada pela média aritmética simples das notas.

- Quais são os dados de entrada? R: Nota P1 e Nota P2
- Quais são os dados de saída? R: Média calculada
- Quais são os passos a serem executados para o devido cálculo?

R:

- 1) Obter as duas notas
- 2) calcular a média  $(\text{Nota P1} + \text{Nota P2}) / 2$
- 3) Apresentar o resultado do cálculo da média

## 2 FORMAS DE REPRESENTAÇÃO DE ALGORITMOS COMPUTACIONAIS

Podemos representar AC de diversas maneiras. Nesta disciplina daremos ênfase em duas formas, a saber: Fluxograma (gráfica) e pseudocódigo. Esta última daremos mais detalhes pois assemelha-se muito a uma linguagem de programação.

### 2.1 FLUXOGRAMA

Como já mencionado os fluxogramas são representações gráficas e são formados por símbolos (caixas, losangos, entre outros) e setas dão o sentido do processamento. Algumas vantagens podem ser elencadas, tais como: (i) Compreensão mais fácil e (ii) qualquer pessoa pode ter um entendimento (Mesmo para leigos). Algumas desvantagens também podem ser levantadas, tais como: (i) Alterações nas estruturas (redesenho) pode ser trabalhosas e (ii) Algoritmos complexos são trabalhosos de especificar.

Veja a Figura 6 tente responder as perguntas.

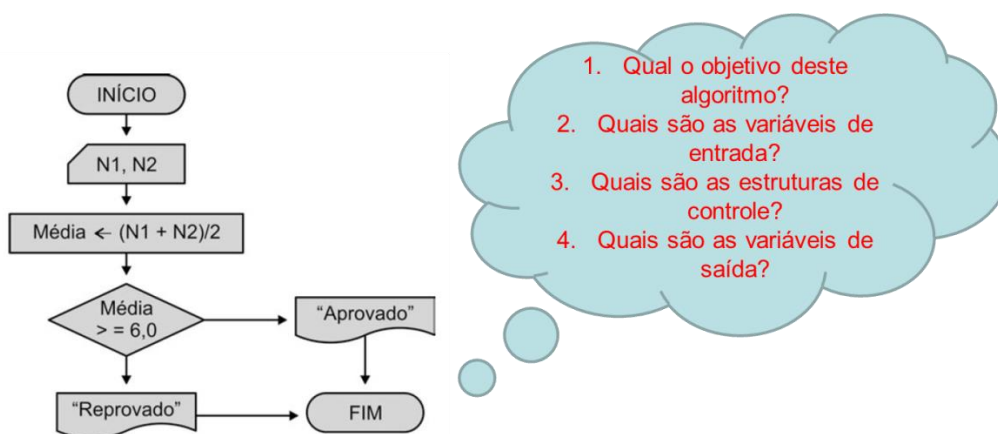


Figura 6: Fluxograma Exemplo

Podemos responder as perguntas da seguinte forma: O objetivo do algoritmo é calcular a média aritmética simples de duas notas e apresentar o resultado Aprovado ou Reprovado. As variáveis de entrada são as duas notas necessárias para o cálculo da média, sendo que as estruturas de controle são as atividades de processamento que é a atribuição (faz o cálculo da média) e o teste dos valores da média calculada. As variáveis ou informações de saída são: "Aprovado" ou "Reprovado".







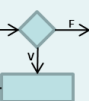
Símbolo	Significado
	Terminação (Início ou Fim)
	Entrada de Dados (leitura)
	Processamento
	Saída de Dados (escrita ou impressão)
	Tomada de decisão
	Fluxo da execução
	Laço de repetição indeterminado

Figura 7: Principais símbolos Fluxograma

A figura 7 apresenta um resumo dos principais símbolos usados para a construção de um fluxograma para algoritmos computacionais. Lembre que sempre teremos um início e fim. Você pode buscar mais exemplos na bibliografia disponibilizada na disciplina.

## 2.2 PSEUDOCÓDIGO

Pseudocódigo é uma linguagem especial para escrever algoritmos e consiste numa linguagem simplificada, linguagem textual, estruturada e rígida. Algumas características podem ser elencadas:

- 1) Utiliza palavras-reservadas;
- 2) Indentação<sup>1</sup> (reco, veja um exemplo na Figura 8);
- 3) Um comando por linha;
- 4) Utiliza ";" como finalizador de comando.

### Código em C com Indentação

```
#include <stdio.h>
int n, i, c, vlr;
main(){
    scanf("%i", &n);
    if (n>30)
        printf("Nro errado!");
    else
        for (i=0; i<n; i++){
            for (c=0; c<n; c++)
                scanf("%i", &vlr);
```

### Código em C sem Indentação

```
#include <stdio.h>
int n, i, c, vlr;
main(){
    scanf("%i", &n);
    if (n>30)
        printf("Nro errado!");
    else
        for (i=0; i<n; i++){
            for (c=0; c<n; c++)
                scanf("%i",&vlr);
```

Figura 8: Indentação (Exemplos)

<sup>1</sup> De acordo com o dicionário indentação um significa substantivo feminino Ação de indentar, de afastar o texto da sua margem, geralmente inserindo espaços entre a margem e o começo do parágrafo.

Vamos considerar o conceito de estruturas de dados aqueles elementos que estão relacionados apenas aos valores que estão armazenados em memória e que todo o processamento (processo de armazenamento e transformações dos valores) será realizado em memória principal.

Assim:

- 1) Memória = conjunto de posições
- 2) Cada posição recebe uma identificação (nome) e armazena um valor → Variável. Veja a Figura 9.

#### Abstração do Conceito de Memória

IDENTIFICADOR	IDADE	NOME	X1
VALOR	18	"João"	2.5

Figura 9: Abstração Memória

Uma premissa básica é que se armazenarmos um novo valor em uma posição o valor antigo é perdido. Todas as variáveis ou identificador devem possuir as seguintes características:

- 1) Identificador: Nome de uma posição de memória;
- 2) É definido pelo programador;
- 3) Recomenda-se o uso de nomes significativos;
- 4) Exemplo: Idade, Nome\_Aluno, Sexo\_Aluno, Vlr\_Salario;
- 5) Geralmente iniciam com uma letra;
- 6) Não usar: caracteres especiais (\*, /, ?, ., #, +, -)

OBS: Não usar as palavras reservadas da linguagem de programação: (For, While, if, then, else, etc).

O conceito de Constantes representa um valor de memória que não se altera durante a execução do algoritmo. Normalmente as constantes são declaradas no início do algoritmo e em **letras maiúsculas**. Ex: PI, NRO\_SALAS, NRO\_MAX\_ALUNOS.

Já o conceito de variáveis representa um valor de memória que se altera durante a execução do algoritmo. Alguns tipos primitivos de dados podem ser:

- 1) Inteiro (Ex: -1, -10, -100, 0, 100, 120)
- 2) Real (Ex: 1,5; 20,5; -30,666; 0)
- 3) Booleano (lógico: true or false)
- 4) Caractere (alfanumérico: "11FFEEDD"; "everaldo")

Alguns tipos de estruturas de Dados (ED) pode ser: (i) Homogêneos ou (ii) Heterogêneas. Estes tipos de dados serão detalhados nos próximos capítulos desta apostila. Exemplos de ED homogêneas são os Vetores que suportam N posições de memória de uma mesmo tipo de dado, como por exemplo, char[1..100]: string. Outro exemplo são as Matrizes que suportam NXM posições de memória de um mesmo tipo de dado, como por exemplo, int [1..10, 1..10]: matriz\_identidade.

Exemplos de ED Heterogêneos são os registros que compreendem uma estrutura que suporta K variáveis de tipos diferentes, como por exemplo:

```
Registro {  
    char[1..50]: nome;  
    int: idade;  
    real: salario;  
    int: sexo;  
    booleano: ativo }; cadastro;
```

### 2.2.1 ESTRUTURA BÁSICA PSEUDOCÓDIGO

A Figura 10 apresenta a estrutura básica de um algoritmo representado em pseudocódigo. Note que existe a seção de início e fim do algoritmo e é nesta seção que serão apresentados todos os comandos para que o problema seja resolvido. É importante salientar que todos os comandos serão executados de forma sequencial. Estes comandos serão vistos com mais detalhes a seguir.

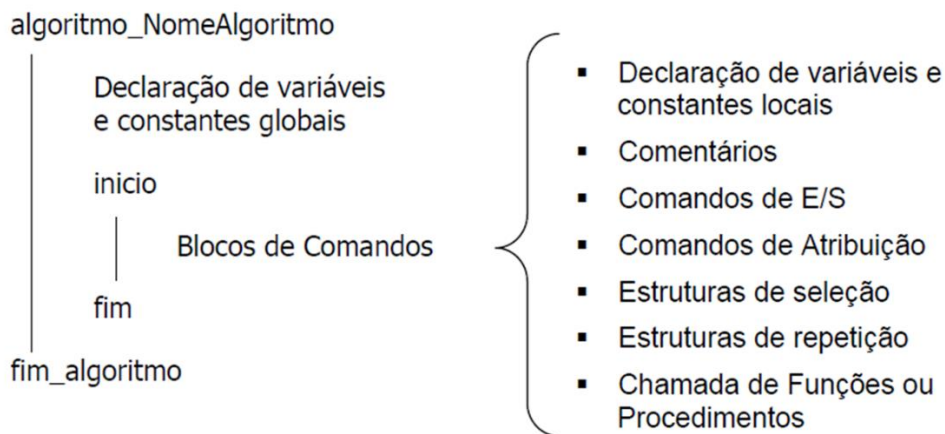


Figura 10: Estrutura Pseudocódigo

### 2.2.2 DECLARAÇÃO DE VARIÁVEIS

Para declarar variáveis é usado a instrução Declare. Abaixo temos a sintaxe de como declarar as variáveis que iremos usar em nosso algoritmo. Verifica que tipo\_do\_dado é referente aos tipos primitivos Inteiro, real, caracter ou lógico, bem como as estruturadas de dados.

Sintaxe: **declare**

**<tipo\_do\_dado>: Lista\_Variaveis;**

*Tipo\_do\_dado = tipos primitivos (inteiro, real, caracter e lógico) ou tipos estruturados (vetores, matrizes e registros).*

*Lista\_Variaveis = conjunto de identificadores.*

**Dica:** Comentários não são comandos, apenas explicações em linguagem natural no código

Para inserir comentários insira após o comando “//” para comentários de uma linha ou “/\*” para iniciar um comentário de várias linhas e ao final “\*/”. Veja exemplo abaixo:

```
/* este algoritmo tem o objetivo de calcular a média final do aluno e tem como variáveis de entrada duas notas e como saída a média aritmética simples*/
```

OBS: É uma boa prática inserir comentários ao longo do algoritmo para que tenhamos referências e explicações para que todos tenhamos um bom entendimento.

### 2.2.3 COMANDOS DE ENTRADA E SAÍDA

Os comandos de entrada e saída tem o objetivo de realizar o interfaceamento entre máquina e usuário. Os principais comandos são: leia e escreva.

O comando leia tem a função de realizar a entrada de dados e armazenar o valor nas variáveis correspondentes e o comando escreva tem a função de escrever na tela algumas informações. Abaixo segue a sintaxe para esses dois comandos.

Sintaxe: **leia** (identificador<sub>1</sub> [, identificador<sub>2</sub>, ..., identificador<sub>n</sub>]); //entrada de dados

Sintaxe: **escreva** (termo<sub>1</sub> [, termo<sub>2</sub>, ..., termo<sub>n</sub>]); //saída de dados

Onde,

Identificador: nome da variável que irá armazenar a informação de entrada

Termo: uma variável, constante, literal ou expressão

Exemplos:

```
leia (nota1, nota2, nota3);
```

```
escreva (“A média final é: “, (nota1+nota2+nota3)/3);
```

```
//realiza a concatenação entre o literal e a expressão
```

### 2.2.4 COMANDOS DE ATRIBUIÇÃO

Estes comandos têm o objetivo de atribuir valores às variáveis de um programa. Estes valores poderão ser expressões aritméticas ou valores constantes.

Sintaxe: variável ← expressão;

Variável: nome da variável que irá armazenar o resultado da expressão

Expressão: pode ser uma variável, uma constante, uma expressão/função matemática ou expressão lógica

Exemplos:

```
nome ← “fulano de tal”;
```

```
soma ← nota1 + nota2;
```

```
media ← (nota1*2 + nota2*3)/5;
```

```
raiz ← sqrt(media);
```

### 2.2.5 OPERADORES

Usaremos três tipos de operadores para testes lógicos, relacionais ou para realizar operações aritméticas em nossos algoritmos. A primeira classe são os **operadores relacionais** que servem para realizarmos comparações entre valores. São eles:

- = (realiza o teste de igualdade entre dois valores)
- <> (realiza o teste de diferença entre dois valores)
- > (realiza o teste de maior que entre dois valores)
- >= (realiza o teste de maior ou igual que entre dois valores)
- < (realiza o teste de menor que entre dois valores)
- <= (realiza o teste de menor ou igual que entre dois valores)

A segunda classe são os **operadores lógicos** que têm o objetivo de conectar por meio de seus operadores expressões relacionais. Os operadores lógicos sempre retornam valores lógicos (verdadeiro ou Falso). São eles:

- E** (função AND): retorna Verdade se todas as sentenças são verdadeiras
- OU** (função OR): retorna Verdade se alguma das sentenças for verdadeira
- Não** (função NOT): faz a inversão da sentença

A terceira classe são os **operadores aritméticos** que têm o objetivo de realizar as operações básicas da matemática. São eles:

- +
  - 
  - \*
  - /
  - ^
- (soma)  
(subtração)  
(multiplicação)  
(divisão)  
(exponenciação)

### 2.2.6 FUNÇÕES MATEMÁTICAS E PRECEDÊNCIA DAS OPERAÇÕES

Podemos utilizar algumas funções matemáticas para resolver determinados tipos de problemas mais complexos. A seguir algumas funções:



- ***abs(X)***: obtém o valor absoluto de X;
- ***sqrt(X)***: calcula a raiz quadrada de X;
- ***log(X)***: calcula o logaritmo de X;
- ***mod(X,Y)***: obtém o resto da divisão de X por Y;
- ***trunca(X)***: obtém a parte inteira de X;
- ***round(X)***: arredonda o valor de X;
- ***sen(X)***: calcula o valor do seno de X;
- ***cos(X)***: calcula o valor do cosseno de X;
- ***tan(X)***: calcula o valor da tangente de X.

Dica: deve-se observar que existe uma precedência de operações entre as operações aritméticas, relacionais, lógicas e funções. A Figura 11 apresenta a sequência correta de execução quando as expressões contêm mais de um tipo de operador.

<b><i>Operadores</i></b>
Parênteses e Funções
Exponenciação e Radiciação
Multiplicação e Divisão
Soma e Subtração
Operadores Relacionais
Operadores Lógicos

Figura 11: Precedência das operações

### 2.2.7 ESTRUTURAS DE SELEÇÃO (CONDICIONAL)

Este comando tem a função de realizar teste condicional de uma condição, podendo o fluxo de execução dos comandos ser desviado para um bloco de comandos se o teste for verdadeiro e para outro bloco caso seja falso. Note que este comando pode testar apenas se a condição for verdadeira, omitindo o bloco de comandos para o caso Falso.

Sintaxe:      **se <condição> então**  
                  |  
                  |      Bloco de comandos (caso Verdadeira);  
                  |  
                  | **[senão**  
                  |      Bloco de comandos (caso Falso);]  
                  |  
                  **fim-se**



Exemplo:

```
se sexo = "F" então
|   escreva ("Sra. Fulana de Tal");
senão
|   escreva ("Sr. Fulano de Tal");
fim-se
```

*Dica: utilizar as indentações para facilitar a leitura.*

Ainda podemos ter estruturas condicionais aninhadas. Isso significa que podemos inserir dentro dos blocos de comandos outras estruturas condicionais. Veja a Figura 12, três condições são testadas. A <condição1> é a primeira a ser testada e, se e somente se, está for verdadeira a <condição2> será testada, senão (caso contrário) a <condição3> será testada e os blocos de comando serão executados conforme o resultado dos testes.

➤ **Exemplo:**

```
se <condição1> então
|
|   se <condição2> então
|   |   bloco de comandos (condição2 Verdadeira);
|   fim-se
|
|   senão
|   |   se <condição3> então
|   |   |   bloco de comandos (condição3 verdadeira);
|   |   senão
|   |   |   bloco de comandos (condição3 falsa);
|   |   fim-se
|   fim-se
fim-se
```

## 2.2.8 ALGORITMO DE EXEMPLO E TESTE DE MESA

Uma técnica interessante para testar se o algoritmo está respondendo à solução proposta é realizar um tipo de teste (depuração). Existem diferentes formas para fazer esse tipo de teste: manual ou automático. Nesta fase iremos abordar os testes manuais.

O teste de mesa consiste no acompanhamento manual (executar linha a linha) do algoritmo. Este teste tem como objetivo: (i) avaliar se os resultados obtidos correspondem àqueles esperados/desejados e (ii) detectar se existem erros de comandos e/ou fluxo de execução.

Durante os testes, deve-se definir os valores de entrada, visando a avaliar as seguintes situações: (i) casos extremos (valores limítrofes da validade) e (ii) exceções do problema (valores inválidos).

Para realizarmos um exemplo do teste de mesa (Martins, 2010), suponha que desejamos obter o conceito (aprovado ou reprovado) de um aluno a partir do cálculo da média aritmética simples entre quatro notas de suas provas. O algoritmo deve calcular a média, apresentá-la e informar ao aluno se ele teve aprovação, sabendo que a média de aprovação deve ser maior ou igual a 6. Dado este problema, o seguinte algoritmo foi elaborado em pseudocódigo e o seu referido teste de mesa.

algoritmo\_Obtem\_Conceito\_Aluno

```

declare
  int: P1, P2, P3, P4;
  real: media;
inicio
  leia(P1,P2,P3,P4);
  media ← (P1+P2+P3+P4) / 4;
  escreva(media);
  se media < 6 entao
    escreva("REPROVADO");
  senao
    escreva("APROVADO");
  fim-se
fim
fim-algoritmo

```

Valores das Entradas: P1 = -5, P2 = 11, P3 = 12 e P4 = 15

Teste2	P1	P2	P3	P4	media	media < 6
<b>Declare</b>	-	-	-	-	-	-
<b>int: P1,P2,P3,P4;</b>	*	*	*	*	-	-
<b>real: media;</b>	*	*	*	*	*	-
<b>Início</b>	*	*	*	*	*	-
<b>leia(P1,P2,P3,P4);</b>	-5	11	12	15	*	-
<b>media ← (P1+P2+P3+P4) / 4;</b>	-5	11	12	15	8,25	-
<b>escreva(media);</b>	-5	11	12	15	8,25	-
<b>se media &lt; 6 entao</b>	-5	11	12	15	8,25	False
<b>Demais comandos</b>	-5	11	12	15	8,25	false

■ Este teste identifica duas falhas no algoritmo: aceitar valor negativo e valores maiores que 10.

Note que podemos identificar claramente as fases do processo de resolução de problemas, a saber:

Entrada: leituras das quatro notas

Processamento: cálculo da média

Saída: conceito do aluno (aprovado ou reprovado)

Verifique que a tabela acima ilustra, para cada linha do algoritmo, os valores de cada variável declarada no algoritmo e os valores de cada uma das condições das estruturas condicionais. Da mesma forma, note que o algoritmo apresenta um problema de valores limítrofes, aceitando valores negativos e maiores de 10.

**Atividade:** Realize a alteração no algoritmo apresentado para que o mesmo aceite apenas valores válidos, ou seja, notas entre o intervalo de [0 a 10].

Agora o mesmo exemplo do algoritmo, mas com valores aceitáveis dentro dos limites.



algoritmo\_Obtem\_Conceito\_Aluno

```
declare
    int: P1, P2, P3, P4;
    real: media;
inicio
    leia(P1,P2,P3,P4);
    media ← (P1+P2+P3+P4) / 4;
    escreva(media);
    se media < 6 entao
        escreva("REPROVADO");
    senao
        escreva("APROVADO");
    fim-se
fim
fim-algoritmo
```

Valores das Entradas: P1 = 5, P2 = 6, P3 = 8 e P4 = 4

Teste1	P1	P2	P3	P4	media	media < 6
Declare	-	-	-	-	-	-
int: P1,P2,P3,P4;	*	*	*	*	-	-
real: media;	*	*	*	*	*	-
Inicio	*	*	*	*	*	-
leia(P1,P2,P3,P4);	5	6	8	4	*	-
media ← (P1+P2+P3+P4) / 4;	5	6	8	4	5,75	-
escreva(media);	5	6	8	4	5,75	-
se media < 6 entao	5	6	8	4	5,75	true
Demais comandos	5	6	8	4	5,75	true

\* Representa um valor não conhecido (lixo)

## 2.3 LISTA DE EXERCÍCIOS

- 1) Faça o teste de mesa dos seguintes algoritmos.

### Algoritmo 1

```
declare
    real: A, B;
    int: C, X;
inicio
    A ← 6.0;
    B ← A/2;
    C ← 11;
    X ← trunca(C / 4);
    C ← mod(C, 2);
    B ← 5.4;
    C ← C+1;
    A ← B+2;
fim
fim-algoritmo
```

- 2) Faça o pseudocódigo, o fluxograma e o teste de mesa dos seguintes problemas:

2.1) Elabore um algoritmo que calcule a área das seguintes figuras geométricas:

- a) Círculo, sendo que:  $A_c = 2\pi r^2$ ;
- b) Quadrado, sendo  $A_q = l^2$ ;
- c) Triângulo, sendo  $A_t = (b \cdot h) / 2$

2.2) Elabore um algoritmo que dado um número qualquer, responda se ele é positivo, negativo ou nulo (zero)

3) Elabore um algoritmo que verifica se um determinado número é par ou ímpar

4) Elabore um algoritmo que tenha como objetivo o cálculo do IMC (índice da Massa Corporal) de um ser humano, sabendo que  $IMC = \text{peso (em quilogramas)} / \text{altura}^2$  (em metros). O algoritmo deve mostrar o resultado e o significado do valor, considerando a tabela abaixo:

Resultado	Situação
Abaixo de 17	Muito abaixo do peso
Entre 17 e 18,49	Abaixo do peso
Entre 18,5 e 24,99	Peso normal
Entre 25 e 29,99	Acima do peso
Entre 30 e 34,99	Obesidade I
Entre 35 e 39,99	Obesidade II (severa)
Acima de 40	Obesidade III (mórbida)

**OBS: Veja a correção dos exercícios na aula específica.**

## 2.4 DESAFIOS (INSTRUÇÕES SEQUENCIAIS E CONDICIONAIS)

1) Escreva o algoritmo em pseudocódigo para solucionar os seguintes problemas, utilizando apenas instruções sequenciais e condicionais.

a) Construa um algoritmo que, tendo como dados de entrada dois pontos quaisquer no plano,  $P(x_1, y_1)$  e  $P(x_2, y_2)$ , então escreva a distância entre eles. A fórmula que efetua tal cálculo é:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

b) Escreva um algoritmo que leia três números inteiros e positivos (A, B, C) e calcule a seguinte expressão:

$$D = \frac{R + S}{2}, \text{ onde } R = (A + B)^2 \text{ e } S = (B + C)^2$$

c) Faça um algoritmo que leia a idade de uma pessoa expressa em anos, meses e dias e mostre-a expressa apenas em dias.

d) Faça um algoritmo que leia a idade de uma pessoa expressa em dias e mostre-a expressa em anos, meses e dias.

e) Faça um algoritmo que leia as 3 notas de um aluno e calcule a média final deste aluno. Considerar que a média é ponderada e que o peso das notas é: 2,3 e 5, respectivamente.

f) Faça um algoritmo que leia o tempo de duração de um evento em uma fábrica expressa em segundos e mostre-o expresso em horas, minutos e segundos.

**Nota: Ler Seção 3.3. Livro: Como programar em C de Deitel e Deitel.**

---

### 3 LINGUAGEM C

A linguagem C foi criada pela AT&T Bell por Brian W. Kernighan e Dennis Ritchie em 1972 com o objetivo de escrever o Sistema Operacional Unix. A ANSI (*American National Standard Institute*) padronizou a linguagem devido à sua evolução. A linguagem C é dita como uma linguagem de propósitos gerais, de grande portabilidade e de facilidade no manuseio direto com o hardware (DEITEL E DEITEL, 2015). Esta linguagem também influenciou outras linguagens como: C++, Java e o C#.

#### 3.1 CARACTERÍSTICAS E ESTRUTURA BÁSICAS

Algumas características básicas de um programa escrito na linguagem C são:

- Comandos em letras minúsculas seguidos por ponto-e-virgula:
  - Ex: `printf("Hello world");`
- Podem (devem) ser incluídos comentários em meio do código
  - `//` para comentário de uma linha
  - `/*` início e `*/` para fim → comentários de mais de uma linha
- Extensão `.c` (arquivo texto)
  - Ex: `programa.c`
- *Case sensitive* (*Nome = nome?*), significando que tais variáveis serão diferentes
- Blocos de comandos são delimitados por `{ }`
- Bloco principal é o *main* (*todo programa em c deve ter um*)

Veja um exemplo de um programa em C com sua estrutura básica. A seguir iremos detalhar cada um dos blocos apresentados.

<code>#include &lt;stdio.h&gt;</code>	}	Bibliotecas de funções (diretivas para o pré-processamento)
<code>int p1, p2;</code>	}	Declaração de variáveis globais
<code>main(){</code>		
<code>float media;</code>		
<code>scanf("%i", &amp;p1);</code>		Declaração de Variáveis locais
<code>scanf("%i", &amp;p2);</code>		Comandos (instruções) para a função main()
<code>media = float (p1+p2)/2;</code>		
<code>printf("A media foi %f", media);</code>		
<code>,</code>		

### 3.2 BIBLIOTECAS DE FUNÇÕES

A primeira parte de um programa em C refere-se às diretivas de pré-processamento. Estas diretivas têm a função de incluir em nossos programas algumas funções pré-definidas como funções de entrada e saída (i/o), funções matemáticas, de sistemas, entre outras (DEITEL e DEITEL, 2015). Para incluir uma biblioteca de função deve ser utilizada a instrução `#include` e o nome da biblioteca deve estar entre `<>`.

Veja abaixo algumas bibliotecas que serão utilizadas neste curso.

Biblioteca	Descrição
stdio.h	Funções de entrada e saída
stdlib.h	Funções padrão (conversão de tipos, nro aleatórios, controle de processos, alocação de memória)
math.h	Funções matemáticas
system.h	Funções de sistema
string.h	Funções de texto

\* [https://pt.wikipedia.org/wiki/Biblioteca\\_padr%C3%A3o\\_do\\_C](https://pt.wikipedia.org/wiki/Biblioteca_padr%C3%A3o_do_C)

Segue exemplo do uso da biblioteca stdio.h que disponibiliza as funções de printf e scanf.

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

```
main() {
```

```
    printf("Hello world");
```

```
}
```

### 3.3 DECLARAÇÃO DE VARIÁVEIS

A declaração de variáveis é um passo muito importante para um programa, pois elas serão as estruturas que utilizaremos para armazenar e processar as informações. As variáveis podem ser definidas como globais ou locais. Mas para nosso entendimento inicial iremos trabalhar apenas com as declarações de variáveis globais. Portanto a declaração delas se dará logo após as declarações das bibliotecas de função.

Partimos do entendimento que todas as variáveis/constantes deverão ter um nome (identificador) e um tipo de dado associado.

Vejamos o exemplo abaixo:

(...)



```
#define PI 3.14          //constante PI

#define NRO_MAX_MESA 10  //constante NRO_MAX_MESA

int Valor1;              //variável com domínio dos inteiros

(...)

perimetro = 2*PI*5.5;    //atribuição

Valor1 = NRO_MAX_MESA--; //atribuição

(...)
```

A quadro a seguir ilustra os tipos de dados que podemos definir para as variáveis, bem como o seu tamanho (em bytes) e o intervalo possível de armazenamento. Em negrito estão os tipos de dados que iremos utilizar com maior frequência em nossos exercícios e desafios.

Tipo Dado	Significado	Tamanho (byte)	Intervalo
<b>Char</b>	<b>Caractere</b>	<b>1</b>	<b>de -128 a 127</b>
unsigned char	Caracter sem sinal	1	de 0 a 255
short int	Inteiro curto	2 (16 bits)	de -32 768 a 32 767
unsigned short int	Inteiro curto sem sinal	2 (16 bits)	de 0 a 65 535
<b>Int</b>	<b>Inteiro</b>	<b>2 (no processador de 16 bits)</b> <b>4 (no processador de 32 bits)</b>	<b>de -32 768 a 32 767</b> <b>de -2 147 483 648 a 2 147 483 647</b>
unsigned int	Inteiro sem sinal	2 (no processador de 16 bits) 4 (no processador de 32 bits)	de 0 a 65 535 de 0 a 4 294 967 295
long int	Inteiro Longo	4 (32 bits)	de -2 147 483 648 a 2 147 483 647
unsigned long int	Inteiro Longo sem sinal	4 (32 bits)	de 0 a 4 294 967 295
<b>Float</b>	<b>Flutuante (real)</b>	<b>4 (32 bits)</b>	<b>de -3.4*10<sup>-38</sup> à 3.4*10<sup>38</sup></b>

double	Flutuante Duplo	8 (64 bits)	de $-1.7 \times 10^{-308}$ a $1.7 \times 10^{308}$
long double	Flutuante duplo longo	10 (80 bits)	de $-3.4 \times 10^{-4932}$ a $3.4 \times 10^{4932}$

Seguem mais alguns exemplos de declarações de variáveis. Note que podemos ter uma lista de nomes (identificadores) para um mesmo tipo de dado. Também é recomendável o uso de nomes de variáveis que tenham algum sentido, evitando identificadores com apenas uma letra. Lembre-se que na linguagem C nomes são case sensitive! Da mesma forma é recomendável o uso de comentários para as declarações das variáveis.

```
(...)
int idade;           //comentários
char conceito;       //comentários
float nota1, nota2, nota3; //comentários
(...)
```

**ATIVIDADE 1:** Complete o código C abaixo declarando todas as variáveis e/ou constantes:

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

//insira aqui a declaração das variáveis utilizadas neste programa. Note que temos uma variável que aceita string, portanto pesquise como fazer a declaração de uma string em C.

```
int main(){

    printf("Cálculo da Média Final. Informe o Nome e notas do aluno com precisão de 0,1 (1 décimo).
    P.e. 5,5; 6,5; 7,1 \n\n\n");

    printf("Informe o Nome.....: ");
    scanf("%s", nome_aluno);
    printf("Informe a Nota 1.....: ");
    scanf("%f", &nota1);
    printf("Informe a Nota 2.....: ");
    scanf("%f", &nota2);
    printf("A Média final do aluno %s é.....: %f", nome_aluno, (nota1 + nota2)/2);
```

}

**ATIVIDADE 2:** Faça a declaração das variáveis abaixo, dando um identificador e tipo de dado:

- a) Armazenar a informação "Fulano de Tal": \_\_\_\_\_
- b) Armazenar a informação 10,5: \_\_\_\_\_
- c) Armazenar a informação Verdadeiro ou Falso: \_\_\_\_\_
- d) Armazenar a informação 60.550: \_\_\_\_\_
- e) Armazenar a informação 333.455,369: \_\_\_\_\_
- f) Armazenar a informação -500: \_\_\_\_\_
- g) Armazenar a informação -1/2: \_\_\_\_\_
- h) Armazenar a informação -5555889/200: \_\_\_\_\_
- i) Armazenar a informação  $3 \times 10^{20}$ : \_\_\_\_\_
- j) Armazenar a informação  $0,5 \times 10^6$ : \_\_\_\_\_
- k) Armazenar a informação  $1,66 \times 10^{-9}$ : \_\_\_\_\_

OBS: As respostas para estas atividades encontram-se na vídeo aula correspondente.

### 3.4 DESAFIO

Faça a tradução dos algoritmos elaborados em pseudocódigo para programas em C. Participe dos fóruns de discussão dos desafios para troca de experiências com os colegas.

Dica1: Para compilar um programa em C na linha de comando:

**Linux:**

`gcc programa.c -o executável`

**Windows:** achar o programa gcc.exe e incluir o caminho na variável de ambiente PATH

`Set path=%path%+C:\Program Files (x86)\Dev-Cpp\MinGW64\bin`

Dica2: Para ajustar a linguagem de entrada insira em seu código as seguintes instruções:

....

```
#include <locale.h>    // biblioteca para a determinação da linguagem padrão
```

```
main() {
```

```
    setlocale(LC_ALL, "PORTUGUESE"); // configura a entrada
```

```
...
```

```
...
```

}

**LEITURA COMPLEMENTAR:** Capítulo 2 Livro Deitel e Deitel 6ª. Edição, 2015.

---

---

## 4 OPERADORES EM C

Os operadores em C aparecem nas expressões de manipulação de dados (processamento) e podem ser: (i) aritméticos, (ii) relacionais e (iii) lógicos. A seguir serão apresentados exemplos para cada operador e sua forma de funcionamento.

### 4.1 Operadores Aritméticos

Estes operadores geralmente são binários (necessitam de dois operandos) ou unários (apenas um operando) e seu resultado (da operação) é um valor numérico. Os seguintes operadores são listados na tabela abaixo:

Operador	Operação
=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
-	Sinal Negativo (operador unário)

Temos que ter um cuidado, pois existe uma prioridade de operação entre os operadores aritméticos, sendo que a leitura deve ocorrer da esquerda para a direita e a prioridade será eliminar inicialmente:

- Parênteses internos
- Funções matemáticas
- Multiplicação
- Divisão
- Resto
- Adição
- subtração

Exemplo:



...

```
float nota1, nota2, media;  
char conceito;  
nota1 = 6.0;  
nota2 = 7.5;  
media = (nota1+nota2*4)/5;           //faça o cálculo da  
expressão....media=7.2  
conceito = 'C'
```

...

#### ATIVIDADE:

Dados os valores inteiros das variáveis A=10; B=3 C=9 e D=7, determine o resultado das seguintes expressões aritméticas:

- a)  $(A+B)/C =$  \_\_\_\_\_
- b)  $A * C / B + (D+A) =$  \_\_\_\_\_
- c)  $A + C / B + D =$  \_\_\_\_\_
- d)  $(A + B) \% D =$  \_\_\_\_\_
- e)  $-(A*C) / A =$  \_\_\_\_\_
- f)  $3 * (C \% 3) * ((A + B) / (C * D)) =$  \_\_\_\_\_
- g)  $A\%(C/B) + C*D =$  \_\_\_\_\_

Respostas nos slides da Aula.

#### 4.1.1 OUTRAS FUNÇÕES

Também utilizaremos outras funções para algumas expressões aritméticas. Verificaremos os operadores de incremento, decremento e acúmulo. Veja a tabela abaixo:

Operador	Operação
++	Incremento de 1 (pré ou pós)
--	Decremento de 1 (pré ou pós)
+=	Acúmulo por adição
-=	Acúmulo por subtração
*=	Acúmulo por multiplicação



---

/=	Acúmulo por divisão
%=	Acúmulo por módulo (resto)

Veja abaixo alguns exemplos desses operadores. Veja que incremento funciona apenas para incrementar 1 de uma determinada variável. Operadores de decremento, da mesma forma, decrementam o valor de uma determinada variável. Os operadores de acúmulo, como o próprio nome indica, realizam a adição do valor da própria variável a outro valor (constante ou outra variável).

Dica: realize vários testes em um programa em C com todos esses operadores.

(...)

```
int x, y, z;
y=0;
x=1;
y+=x;           // mesmo que y = y + x
y++;            // mesmo que y = y + 1
z = 10%y;       // resto da divisão de z por y
```

Uma classe especial de operador de incremento é chamada de PÓS-INCREMENTO. Veja o exemplo abaixo e verifique o comportamento do operador ++ quando está **APÓS** uma variável. Verifique que a variável Y receberá o valor de X e logo após (pós-incremento) o valor de X será aumentado em 1.

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
    int x=0;
    int y=1;
    y = x++;    /* pós incremento =
                y = x;
                x = x+1;
                assim, ao final: x=1 e y=0 */
    printf("x = %i\n", x);
    printf("y = %i\n", y);
}
```

Outra classe especial de operador de incremento é chamada de PRÉ-INCREMENTO. Veja o exemplo abaixo e verifique o comportamento do operador ++ quando está **ANTES** de uma variável. Verifique que a variável Y receberá o valor de X já incrementado em 1. Assim as duas variáveis terão o mesmo valor ao final do programa.

Dica: Copie os dois programas e faça diversos testes

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
    int x=0;
    int y=1;
    y = ++x;    /* pré incremento =
                x = x+ 1;
```

```
        y = x;  
        assim, ao final: x=1 e y=1    */  
    printf("x = %i\n", x);  
    printf("y = %i\n", y);  
}
```

## 4.2 OPERADORES RELACIONAIS

Os operadores relacionais são utilizados para comparar operandos e seu resultado sempre será um valor booleano: Verdadeiro ou Falso (True or False).

Toda expressão relacional é composta por operadores relacionais, tais como: maior, menor, igual ou negação, podendo ser simples ou composto. Expressões simples usam apenas operadores relacionais e compostos podem usar operadores relacionais e lógicos. O quadro a seguir apresenta os operadores relacionais:

Operador	Operação
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual a
!=	Não igual a (diferente de)

Para ilustrar o funcionamento das expressões relacionais, a tabela abaixo apresenta alguns exemplos com seu resultado.

Expressões Lógicas	Resultado
5>=2	Verdadeiro
5+4<7	Falso
A > C	Será verdadeiro se o valor de A for maior que o valor de C, caso contrário será falso



$A * B < 10$	Será verdadeiro se a multiplicação das variáveis A e B for menor que 10, caso contrário será falso
"Maria" != "João"	Verdadeiro

### 4.3 OPERADORES LÓGICOS

Usaremos três operadores lógicos em nosso curso a saber: E, OU e NÃO (AND, OR e NOT). Operadores lógico do tipo AND são utilizados quando dois ou mais relacionamentos lógicos de uma determinada instrução deve ser verdadeiros. Operadores lógicos OR são utilizados quando pelo menos um dos relacionamentos lógicos de uma determinada instrução é verdadeira e operadores NOT são utilizados quando uma determinada condição deve ser "não verdadeira" ou "não falsa".

A seguir é apresentado a tabela-verdade do operador NOT, bem como seus exemplos de uso.

Expressão A	Não A
Verdadeiro	Falso
Falso	Verdadeiro

Exemplos:

NÃO ( $5 > 2$ ); // falso  
NÃO ("Maria" == "João"); // verdadeiro

A seguir é apresentado a tabela-verdade do operador AND, bem como seus exemplos de uso.

Expressão A	Expressão B	A e B
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

Exemplo:

$(5 > 2)$  E  $(3 > 1)$ ; // verdadeiro

("Maria" != "João") e (5 < 2); // falso

A seguir é apresentado a tabela-verdade do operador OR, bem como seus exemplos de uso.

Expressão A	Expressão B	A e B
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Exemplo:

(5 > 2) ou (3 > 1); // verdadeiro

("Maria" != "João") ou (5 > 2); // verdadeiro

#### 4.3.1 OPERADORES LÓGICOS NA LINGUAGEM C

A tabela abaixo apresenta os operadores lógicos na linguagem C, bem como exemplos de uso.

Operador	Operação
&&	E (AND)
	OU (OR)
!	NEGAÇÃO – NÃO (NOT)

Exemplos de uso:

Expressão Lógica Composta	Linguagem C	Resultado
(5 > 2 E 5 + 4 < 7)	(5 > 2 && 5 + 4 < 7)	Falso
(NÃO(5 > 2))	(!(5 > 2))	Falso

("Maria" != "João") OU 5\*2<8)

("Maria" != "João") || 5\*2<8)

Verdadeiro

#### 4.4 PRIORIDADE ENTRE OS OPERADORES

Abaixo segue a prioridade que deve ser seguida na execução de instruções que apresentarem os diversos tipos de operadores, a saber:

- Parênteses mais internos
- Operadores aritméticos
- Operadores relacionais
- Operadores lógicos (NÃO, E, OU)

#### 4.5 OUTROS EXEMPLOS DE OPERADORES EM

O quadro abaixo mostra como escrever expressões aritméticas expressas em álgebra para a linguagem C.

Expressão aritmética em texto de álgebra	Em C	Análise
$\frac{(a+b+c+d)}{4}$	<code>(a+b+c+d) / 4 ;</code>	Parênteses são obrigatórios para efetivar a soma dos 4 valores antes da divisão. Se não usados, d será dividido (/) por 4 e depois serão efetivadas a soma de d/4 com a, b, c.
$\frac{b^2}{4ac}$	<code>b*2 / (4*a*c) ;</code>	Todos os operadores devem ser explicitamente escritos, senão b2 é entendido como um identificador, enquanto que b*2 é uma expressão aritmética.
$(a+b)^2$	<code>(a+b) * (a+b) ;</code>	Não existe operador potência, então aplica-se o conceito, multiplicando-se 2 vezes.
$\frac{1+C}{A-B}$	<code>(1+C) / (A-B) ;</code>	Os identificadores devem ser transcritos para formato em minúsculo, ou caixa baixa. Observar a necessidade dos parênteses
$4 * \pi * r^3$	<code>4*3.1416*r*r*r ;</code>	A constante Pi é um caractere grego, sem significado para C. Neste caso, deve-se colocar o valor aproximado.

#### 4.6 FUNÇÕES MATEMÁTICAS

Alguns problemas podem exigir o uso de algumas funções matemáticas como raiz quadrada, logaritmos, arredondamento, entre outras. A biblioteca math.h apresenta uma lista de várias funções que podem ser utilizadas. Veja abaixo alguma delas:

Dica: lembre-se de incluir `#include <math.h>` em seu programa.



Função	Descrição	Exemplo
sqrt(x)	Raiz quadrada de x	sqrt(900.0) é 30 sqrt(9.0) é 10
exp(x)	Função exponencial de e^x	exp(1.0) é 2.718282 exp(2.0) é 7.389056
log(x)	Logaritmo natural de x (base e)	log (2.718282) é 1.0 log (7.389056) é 2.0
log10(x)	Log de x (base 10)	log10(1.0) é 0.0 log10(10.0) é 1.0 log10(100.0) é 2.0
fabs(x)	Valor absoluto de x	Se x > 0 então fabs(x) é x Se x = 0 então fabs(x) é 0.0 Se x < 0 então fabs(x) é -x
ceil(x)	Arredonda x para o menor inteiro maior que x	ceil(9.2) é 10 ceil(-9.8) é -9
floor (x)	Arredonda x para o maior inteiro menor que x	floor(9.2) é 9 floor(-9.8) é -10
pow(x,y)	x elevado a potencia y	pow(2,7) é 128.0 pow(9, .5) é 3.0
fmod(x,y)	Resto de x/y, como numero de ponto flutuante	fmod(13.657, 2.333_) é 1.992
sin(x)	Seno trigonométrico de x em rad	sin(0.0) é 0
cos(x)	Cosseno trigonométrico de x em rad	cos(0.0) é 1
tan(x)	Tangente trigonométrica de x em rad	tan(0.0) é 0

Veja abaixo um exemplo de um programa que inclui várias funções matemáticas.

```
/* Para usar as funções matemáticas precisamos incluir suas
definições, que estão em math.h. */

#include <math.h>
#define PI 3.1415926536
int main()
{
    double a,b;
    int c,d,e;
    a = 1.0;
    b = exp(a);           // atribui 2.718282 para b ou número
    neperiano e
    a = 4.0;
    a = pow(a,3.0);       /* atribui 64.0 para a */
    b = log10(100);       /* atribui 2.000000 para b */
    a = sin(PI/4.0);      /* atribui 0.707107 para a */
    c = 5;
    d = 3;
    e = c/d;              /* atribui 1 para c - divisão inteira */
    e = c%d;              /* atribui 2 para e - resto da divisão inteira
*/
}
```

#### 4.7 OPERADOR CAST EM C (TYPE CASTING)

Às vezes é necessário realizar operações aritméticas com números inteiros e reais. A Linguagem C possui um tratamento especial para realizar a conversão de tipos em tempo de execução. Esta operação chama-se Casting. Tente executar a operação entre duas variáveis inteiras que retorne um valor fracionário. Veja o exemplo abaixo:

```
#include <stdio.h>
int v1, v2;
float r;
int main(){
    v1=3;
    v2=10;
    r=v2/v1;
    printf("%f", r);
}
```

Você esperaria que o resultado da divisão de V2 por V1 seja um valor 3,33333. Mas ao executar o código o retorno foi 3.000000 sem a parte fracionária. Isso ocorre, pois, a linguagem C verifica que a operação é realizada com variáveis inteiras e, portanto, o resultado deve ser um valor inteiro. Para resolver esta questão usamos uma conversão de tipo CAST. Veja agora o código abaixo.

```
#include <stdio.h>
int v1, v2;
float r;
int main(){
    v1=3;
    v2=10;
    r=(float)v2/v1;
    printf("%f", r);
}
```

---

OBS: Veja uma demonstração deste comportamento no vídeo aula da Aula 4 – Operadores.

---

#### 4.8 LISTA DE EXERCÍCIOS

Escreva um programa em C para cada uma das sentenças abaixo. Veja a correção desses exercícios no vídeo correspondente.

- Dado um número inteiro obter o último algarismo desse número: Ex. se digitado “799” escrever apenas o “9”.
- Dados o primeiro termo e a razão de uma progressão aritmética, determinar a soma dos seus primeiros CINCO termos, sabendo que:

$$a_n = a_1 + (n - 1) * r$$

$$S_n = \frac{(a_1 + a_n) * n}{2}$$

- c) Faça um algoritmo que: i) Obtenha o valor para a variável HT (horas trabalhadas no mês); ii) Obtenha o valor para a variável VH (valor hora trabalhada); iii) Obtenha o valor para a variável PD (percentual de desconto); iv) Calcule o salário bruto:  $SB = HT * VH$ ; v) Calcule o total de desconto:  $TD = (PD/100) * SB$ ; vi) Calcule o salário líquido:  $SL = SB - TD$ ; vii) Apresente os valores de: Horas trabalhadas, Salário Bruto, Desconto, Salário Líquido.

#### 4.9 DESAFIOS

**Faça pelo menos três dos problemas a seguir. Participe do fórum de discussão relativo aos desafios para troca de experiências com os colegas.**

- a) Dado um número de 3 algarismos, inverter a ordem de seus algarismos. Os três algarismos do número dado devem ser diferentes de Zero
- b) Dado um número de 3 algarismos construir outro número de quatro algarismos de acordo com as seguintes regras: (i) os três primeiros algarismos, contados da esquerda para a direita são iguais aos do número dado; (ii) o quarto algarismo é um dígito verificador calculado da seguinte forma: primeiro algarismo + segundo algarismo x 3 + terceiro algarismo x 5; o dígito verificador é igual ao resto da divisão dessa soma por 7.
- c) Dado um número inteiro que representa um número binário de cinco dígitos, determinar o seu equivalente em decimal.
- d) Faça um algoritmo que calcule a quantidade de litros de combustível gasta em uma viagem. Para obter o cálculo, o usuário deve fornecer a média de Km por litro, o tempo gasto na viagem e a velocidade média durante ela. Desta forma, será possível obter a distância percorrida com a fórmula  $DISTÂNCIA = TEMPO * VELOCIDADE$ . Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula:  $LITROS\_USADOS = DISTÂNCIA / Valor\ KM\ por\ LITRO$ . O programa deve apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem.
- e) Faça um algoritmo que leia a velocidade de um veículo em km/h e calcule e imprima a velocidade em m/s (metros por segundo).

## 5 FUNÇÕES DE ENTRADA E SAÍDA

As funções de entrada e saída possibilitam a comunicação entre usuários e computador. Os principais comandos vistos em pseudocódigo foram: Ler (entrada) e Escrever (saída). A função ler possibilita que uma informação digitada no teclado seja capturada e armazenada em uma variável **previamente definida**. A função escrever possibilita apresentar o conteúdo de uma variável (**previamente definida**) ou expressão na tela. Na linguagem C não existe comando interno de E/S, portanto faça-se o uso da biblioteca de função `<stdio.h>` para os comandos mais comuns que são.

- `printf` (escrita)
- `scanf` (leitura)

### 5.1 FUNÇÃO `PRINTF()`

A função `printf()` envia os dados para a saída padrão (*stdout*). A função pode receber dois argumentos:

1. Cadeia de caracteres: uma string de especificação de conversão ou *string* de formato (As especificações de conversão em geral descrevem como escrever os demais argumentos da função)
2. Argumento de saída – variáveis e/ou expressões

Observe que uma característica interessante destas funções é a possibilidade de serem invocadas com 1, 2, ..., n argumentos.

**Exemplo:**

Sem conversão:

```
printf("exemplo de formato sem conversão\n");
```

Com conversão:

```
#include <stdio.h>
(...)
float nota1, nota2, media;
nota1 = 6.0;
nota2 = 8.5;
media = (nota1 + nota2*2)/3;
printf("Nota 1: %f\n", nota1); // exibe a mensagem Nota 1:
6.000000
printf("Nota 2: %f\n", nota2); // exibe a mensagem Nota 2:
8.500000
printf("Média: %f\n", media); //exibe a mensagem Média:
7.666666
(...)
```

Note que nos exemplos acima aparecem os caracteres de SCAPE que não são imprimíveis na tela, apenas sinalizam algum comportamento e necessariamente iniciam com “\”. Veja no quadro a seguir alguns exemplos:

Sequência	Descrição
\n	Nova linha. Posiciona o cursor no início da nova linha.
\t	Tabulação horizontal. Move o cursor para a próxima marca parada de tabulação
\r	Carriage return ( <b>CR</b> ). Posiciona o cursor no início da linha atual; não avança para a próxima linha.
\a	Alerta. Faz soar a campainha (Bell) do sistema.
\\	Barra invertida (backslash). Imprime um caractere de barra invertida em uma instrução <b>printf</b> .
\"	Aspas duplas. Imprime um caractere de aspas duplas em uma instrução <b>printf</b> .

Note que no exemplo anterior a precisão foi de 6 casas decimais para os cálculos da média:

```
printf("Média: %f\n", media); //exibe Média: 7.666666
```

Podemos definir a precisão das casas decimais que serão mostradas com a ajuda de Flags na string de conversão. Veja abaixo:

```
printf("Média: %.2f\n", media); //exibe Média: 7.67
```

No quadro abaixo são mostrados alguns formatos de conversão:





Character	Argument type; Printed As
d, i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x, X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *: print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-] m.dddddd, where the number of d's is given by the precision (default 6).
e, E	double; [-] m.ddddde+/-xx or [-] m.dddddE+/-xx, where the number of d's is given by the precision (default 6).
g, G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *: pointer (implementation-dependent representation).
%	no argument is converted; print a %

Veja alguns exemplos de programas. Copie e cole e execute os programas para verificar o funcionamento da função printf().

**Programa 1:**

```
#include <stdio.h>
int main(){
    float d;
    d=0.00314;
    printf("%e\n", d);
}
```

**Programa 2:**

```
#include <stdio.h>
int main(){
    int i; float d;
    i=123; d=3.14;
    printf("%i %d\n", i, d);
}
```

**Programa 3:**

```
#include <stdio.h>
int main(){
    int i;
    i=123;
    printf("%i\n", i);
}
```

**Programa 4:**

```
#include <stdio.h>
int main(void) {
    int a, b;
```



```
float media;  
scanf( "%i %i", &a, &b);  
media = (float)(a + b)/2.0;  
printf( "A média de %d e %d é %.2f\n", a, b, media);
```

Abaixo segue a sintaxe para o uso de flags na string de especificação de conversão da função printf():

**%[flags][width][.prec][h|l]type**

\*Os campos opcionais estão entre colchetes, mas devem estar na ordem especificada acima.

Flag	Significado
-	Posicione o valor à esquerda
+	O valor deve ser precedido de + ou -
espaço/branco	O valor positivo deve ser precedido de 1 caractere em branco
0	Preencha o valor com zeros
#	Preceder octais com 0, hex com 0x; mostrar ponto decimal...

Exemplo:

```
printf("o valor é %+d", a);
```

**%[flags][width][.prec][h|l]type**

A tabela abaixo descreve "width.prec":

número	Tamanho mínimo da largura ou precisão
*	O próximo argumento da printf é a largura
.numero	Número mínimo de dígitos para int; Número de casas decimais para e ou f Número max de dígitos significativos para g Número max de caracteres para s
.*	O próximo argumento da printf é o valor da precisão (interpretado como acima)

Um h especifica short int, Um l (letra l minúscula) especifica long int.

Exemplos:

```
printf("%05.2f\n", 1.5); // exibe 00001.50  
printf("%05d\n", 10); // exibe 00010  
printf("%.7f", 7, 2, 98.736); // Usa 7 para largura e 2 para  
precisão
```

Veja o código abaixo, copie e cole e execute para analisar as diferenças de alinhamento.

```
/* Alinhando valores pela esquerda e pela direita */

#include <stdio.h>

main() {
    printf("%10s%10d%10s%10.2f\n\n",    "Ola...", 10, "A", 1.5);
    //direita
    printf("%10s%10d%10s%10.2f\n\n",    "Tche...", 20, "B", 1.7);
    //direita
    printf("%10s%10d%10s%10.2f\n\n\n\n", "mas bah..", 30, "C",
    2.5);    //direita
    printf("%-10s%-10d%-10s%-10.2f\n\n", "Ola...", 10, "A", 1.5);
    //esquerda
    printf("%-10s%-10d%-10s%-10.2f\n\n", "Tche...", 20, "B", 1.7);
    //esquerda
    printf("%-10s%-10d%-10s%-10.2f\n\n", "mas bah..", 30, "C",
    2.5);    //esquerda
}
```

## 5.2 FUNÇÃO SCANF()

A função `scanf()` realiza a leitura de dados da entrada padrão (*stdin*). Atua de forma análoga ao `printf()`. A função recebe dois argumentos, sendo:

1. Argumento 1: função de conversão (%d %f, etc..)
2. Argumento 2: lista de variáveis que receberão os valores digitados no teclado, para isso deve-se usar *&Var para referência ao endereço de memória da variável (conceito de ponteiros – mais adiante no curso)*

Devemos ter um cuidado com a entrada de string por meio da função `scanf()`. Alguns compiladores em C podem apresentar problemas com o uso do ***scanf*** para ler uma sequência de caracteres. Entretanto, recomendamos que testem seus programas com o uso do conversor “%s” para uma ***String*** e “%c” para apenas um caractere.

Exemplo:

```
scanf("%s", nome_aluno);    // note que não é usado o &Var
```

Podemos utilizar outra função para a leitura de strings e de caracteres. Para isso vamos incluir a biblioteca *conio.h* e usar os comandos:

- `getch()` //não escreve o caractere na tela
- `getche()` //escreve o caractere na tela
- `gets()` //leitura de uma string do teclado

Exemplo:

```
scanf("%s", nome_aluno); // note que não é usado o &Var
gets(nome_aluno);       // lê string do teclado e armazena em
nome_aluno
conceito = getche();     // lê um caractere do teclado e
armazena em conceito
```

Faça um teste:

Verifique a diferença nos valores da variável nome\_aluno e n\_aluno das seguintes instruções:

- 1) scanf("%s", n\_aluno)
- 2) gets(nome\_aluno),
- 3) atribuir a string "Everaldo Luis Daronco"
- 4) Imprima na tela as variáveis n\_aluno e nome\_aluno

Nota: Você irá verificar que a função scanf() armazena a string até o primeiro espaço, no caso, ela armazenou apenas "Everaldo" na variável n\_aluno. Já a função gets() armazena toda a string "Everaldo Luis Daronco" na variável nome\_aluno.

Mais alguns exemplos do uso das funções com conversão entre tipos:

```
scanf ("%i%i%i%i%i", &a, &b, &c, &d, &e); //realiza leitura de 5 números

printf("%d %d %d %d %d \n", a, b, c, d, e); //imprime 5 nros inteiros (4 na base 10 e 1 na base
octal → conversão entre tipos)
```

**Maiores informações Capítulo 9: Deitel, P.J; Deitel, H.M. Como programar em C.**

---

---

## 6 ESTRUTURAS CONDICIONAIS

No capítulo 2 estudamos pseudocódigo e usamos a instrução SE, com a seguinte sintaxe:

```
Se (<expressão lógica>) então
    <bloco de instruções>
Senão
    <bloco de instruções>
Fim se
```

Na linguagem C podendo reescrever esta mesma instrução das seguintes formas: (i) Estrutura condicional simples; (ii) Estrutura condicional composta; (iii) Estrutura condicional encadeada e (iv) Estrutura condicional de múltipla escolha. Vejamos as diferenças entre elas.

### 6.1 ESTRUTURA CONDICIONAL SIMPLES

Estrutura condicional simples é quando usamos IF sem a cláusula ELSE (senão), podendo ser uma estrutura condicional de uma única instrução e um conjunto de instruções.

Vejam os exemplos:

#### Uma única instrução

```
If (<expressão lógica>)
    <instrução>;
```

#### Um bloco de instruções

```
if (<expressão lógica>) {
    < bloco de instrução>;
}
```

\* Note que a diferença são as chaves que delimitam o conjunto de instruções.

### 6.2 ESTRUTURA CONDICIONAL COMPOSTA

Temos uma estrutura condicional composta quando usamos IF com a cláusula ELSE (senão), podendo ser de uma única instrução e um conjunto de instruções.

Vejam os exemplos:

#### Uma única instrução

```
if (<expressão lógica>)
    <instrução>;
else
    <instrução>;
```

#### Um bloco de instruções

```
if (<expressão lógica>) {
```

```
        < bloco de instrução>;  
    } else {  
        < bloco de instrução>;  
    }
```

\* da mesma forma que na anterior a diferença encontra-se na { }

### 6.3 ESTRUTURA CONDICIONAL ENCADEADA

Em muitos casos podemos usar estruturas condicionais dentro de outras estruturas condicionais. Isso chamamos de encadeamento de IFs.

#### Uma única instrução

```
if (<expressão lógica>  
    <instrução>  
else  
    If (<expressão lógica>  
        <instrução>  
    else  
        <instrução>
```

*Dica: use a indentação para que fique mais legível e de fácil entendimento*

**Desafio:** Tente escrever a sintaxe para IF encadeados com um conjunto de instruções ou pesquisa nos livros.

Veja como funciona os seguintes exemplos. Copie e cole e execute os programas abaixo.

O exemplo abaixo faz o teste na variável idade para determinar as condições de voto obrigatório, facultativo ou não voto de uma determinada idade digitada pelo Usuário.

```
include <stdio.h>  
#include <locale.h>  
main() {  
    setlocale(LC_ALL, "PORTUGUESE");  
    int idade;  
    printf("Informe sua idade: ");  
    scanf("%d", &idade);  
    if (idade <= 15)  
        printf("\n\nnão Vota");  
    else  
        if (idade == 16 || idade == 17 || idade > 70)  
            printf("\n\nVoto facultativo\n");  
        else  
            printf("\n\nvoto obrigatório\n");  
  
    printf("\n\nO voto é um direito do cidadão! Escolha bem nas  
próximas eleições!!!!\n\n\n");  
}
```

Este exemplo executa uma série de entradas conforme interação com o usuário e ao final calcula o valor total do pedido.

```
#include <stdio.h>
#include <locale.h>
#include <conio.h>
    float vlanche, vrefri, vfinal, vdesc;
    int qlanche, qrefri;
    char resp;
main(){
    setlocale(LC_ALL, "PORTUGUESE");
    printf("\n\n Informe a quantidade do Lanche: ");
    scanf("%i", &qlanche);
    printf("\n\n Informe o Valor do Lanche: ");
    scanf("%f", &vlanche);printf("\n\n Deseja refrigerante (S |
N): ");
    resp = getche();
    vfinal=0;
    if (resp == 's' || resp == 'S'){
        printf("\n\n Informe o valor do Refrigerante: ");
        scanf("%f", &vrefri);
        printf("\n\n Informe a quantidade de Refrigerante: ");
        scanf("%d", &qrefri);
        vfinal = float (vlanche*qlanche + vrefri*qrefri);
    } else
        if (resp=='n' || resp=='N')
            vfinal = float (vlanche*qlanche);
    if (vfinal>20.0){
        printf("\n\n ***Vc ganhou desconto de 10%% ***\n\n");
        vdesc = vfinal *0.1;
        vfinal-=vdesc;
    }
    printf("\n\n *** O Valor final é %.2f", vfinal);
}
```

Este exemplo é uma nova versão do programa anterior com a inclusão da função toupper() da biblioteca ctype.h para testar apenas caracteres em maiúsculo. Em negrito as alterações no código anterior.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <locale.h>
main(){
    setlocale(LC_ALL, "PORTUGUESE");
    ....
    printf("\n\n Deseja refrigerante (S | N): ");
    resp = getche();
    resp = toupper(resp);          // converte o caractere digitado
em maiúscula
                                // existe o tolower() converte em minúsculo
    if (resp == 'S'){
```



```
        printf("\nInforme o valor do Refrigerante: ");
        scanf("%f", &vrefri);
        printf("\nInforme a quantidade de Refrigerante: ");
        scanf("%d", &qrefri);
        vfinal = vlanche*qlanche + vrefri * qrefri;
    }
    else
        if (resp == 'N')
            vfinal = vlanche*qlanche;
    if ( vfinal>200.0){
        printf("\n\n ***Vc ganhou desconto de 10%!***\n\n");
        vdesc = vfinal *0.1;
        vfinal-=vdesc;
    }
}
```

## 6.4 OPERADOR “?”

Podemos utilizar o operador? para uma estrutura condicional de apenas uma instrução. Veja a sintaxe e os exemplos a seguir.

***expressão lógica*** > ? <***expressão aritmética1***> : <***expressão aritmética2***>

***Verdadeiro***

***Falso***

O exemplo abaixo apresenta o uso do operador “?” realizando o teste se X>Y, se verdade irá atribuir a variável “resultado” o valor da variável “X”, caso contrário irá atribuir a soma dos valores de X e Y a variável “resultado”.

```
#include <stdio.h>
#include <locale.h>
main(){
    setlocale(LC_ALL, "PORTUGUESE");
    int x, y, resultado;
    printf("informe o valor de x: \n");
    scanf("%d", &x);
    printf("informe o valor de y: \n");
    scanf("%d", &y);
    resultado = (x>y) ? x : x+y;    //mesmo que:
    /* Se (x>y)
    resultado = x;
    else
    resultado = x+y;*/
    printf("o resultado é %d: \n", resultado);
}
```



### 6.5 ESTRUTURA CONDICIONAL MULTIPLA ESCOLHA (SWITCH)

Analise o programa abaixo e verifique que o mesmo implementa uma calculadora com quatro operações básicas e usa a estrutura condicional **switch()**. Esta função recebe como parâmetro uma variável que após será testada pelas cláusulas **CASE**. Verifique que ao final de cada Cláusula CASE existe uma instrução **BREAK**. Faça um teste retirando as instruções Break do programa e veja o comportamento do programa. A cláusula **DEFAULT** é executada caso nenhuma das cláusulas CASE tenha sido executada.

```
#include <stdio.h>
#include <conio.h>
#include <locale.h>
main() {
    setlocale(LC_ALL, "PORTUGUESE");
    float valor1, valor2, resultado;
    char op;
    printf("Informe o valor 1: \n");
    scanf("%f", &valor1);
    printf("Informe o valor 2: \n");
    scanf("%f", &valor2);
    printf("Informe a operação (+ | - | * | /): \n");
    op = getche();
    switch(op) {
        case '+':
            resultado = valor1 + valor2;
            break;
        case '-':
            resultado = valor1 - valor2;
            break;
        case '*':
            resultado = valor1 * valor2;
            break;
        case '/':
            resultado = valor1 / valor2;
            break;
        default:
            printf("\n Operação Inválida!\n");
    }
    printf("\n\nResultado: %.2f", resultado);
}
```

Veja que na instrução de Múltipla Escolha switch() não existem testes de expressões lógicas e nem testes de intervalos. Abaixo verificamos a sintaxe desta instrução.

#### Sintaxe:

```
Switch <variável>{
    case <valor1>:
        <bloco de instruções>
        break;
    case <valor2>:
```



```
        <bloco de instruções>
        break;
    case <valorn>
        <bloco de instruções>
        break;
    default:
        <bloco de instruções>
}
```

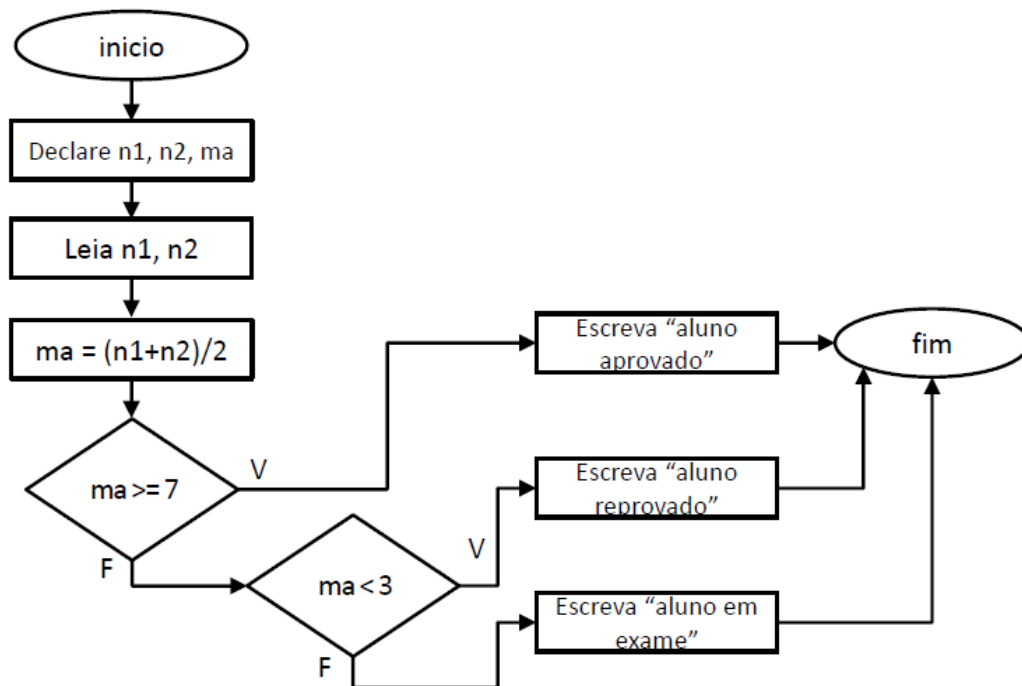
O Exemplo abaixo tem o objetivo de identificar o conceito de notas inteiras, fazendo trocas de valores interiores para letras. Note que na linha 9 existe uma série de valores, isso é uma simulação de operador lógico “OR”, pois se nora for: 1 ou 2 ou 3 ou 4 ou 5 ou 6 o conceito correspondente será D.

```
#include <stdio.h>
#include <locale.h>
main() {
    setlocale(LC_ALL, "PORTUGUESE");
    int nota;
    char conceito;
    printf("Informe a nota final: \n");
    scanf("%d", &nota);
    switch(nota) {
        case 0: case 1: case 2: case 3: case 4: case 5: case
6:
            printf("\n Conceito D!\n");
            break;
        case 7:
            printf("\n Conceito C!\n");
            break;
        case 8:
            printf("\n Conceito B!\n");
            break;
        case 9: case 10:
            printf("\n Conceito A!\n");;
            break;
        default:
            printf("\n Valor Inválido!\n");
    }
}
```

## 6.6 LISTA DE EXERCÍCIOS

Escreva em linguagem C os seguintes problemas.

- 1) Leia o fluxograma abaixo e escreva um programa em C com todos os testes.



- 2) Escreva um programa que leia o ano de nascimento de uma pessoa e calcule e escreva sua idade. Informar também se ele já tem idade para votar (idade  $\geq 16$ ) com a frase "apto a votar" e também se já tem idade para conseguir a CNH (idade  $\geq 18$ ) com a frase "apto a dirigir"
- 3) Escreva um programa que lê um caractere e testa se é vogal. Se for vogal, escrever o caractere lido e a mensagem correspondente; caso contrário, escrever não é vogal
- 4) Escrever um programa para uma calculadora de 4 operações. Fazer uma versão usando IF.

## 6.7 DESAFIOS

Faça pelo menos três dos programas abaixo e participe do fórum de discussão sobre os desafios no ambiente virtual.

- a) Escreva um programa que leia 4 valores (op, a, b, c) e teste: i) se op for igual a 1, calcular a média aritmética de A, B e C e escrever esta média; ii) se op for igual a 2, somar os 3 valores atribuindo esta soma a uma variável qualquer e escreva o resultado; iii) se op for igual a 3, fazer um teste para saber se B é par, se for par, escrever a mensagem e o valor, caso contrário, escrever ímpar.
- b) Dado um número inteiro qualquer, fazer o programa para imprimir a raiz quadrada desse número se ele for positivo, se ele for negativo imprimir o seu quadrado.

- c) Sabendo-se que se uma coordenada X de um ponto no plano (X, Y) for igual a zero e a coordenada Y for diferente de Zero, o ponto está sobre o eixo dos Y, caso contrário está sobre o eixo dos X. Escrever um programa que imprima uma mensagem indicando onde se encontra o ponto no plano, a partir das coordenadas desse ponto no plano (X, Y).
- d) Um supermercado deseja reajustar os preços de seus produtos, para mais ou para menos, de acordo com os critérios demonstrados na tabela a seguir. Escrever um programa que leia o preço atual e a venda mensal média do produto e calcule e escreva o seu preço reajustado.

Venda Média Mensal	Preço Atual	% Aumento	% de Diminuição
< 500	< R\$ 30,00	10	
$\geq 500$ e < 1200	$\geq$ R\$ 30,00 e < R\$ 80,00	15	
$\geq 1200$	$\leq$ R\$ 80,00	-	20

- e) Sabendo que A, B e C são valores dos lados de um triângulo, e se  $A > B + C$  não será possível formar um triângulo. Fazer um programa para ler os valores de A, B e C e se eles o podem formar um triângulo informando ao usuário o resultado, positivo ou negativo.
- f) Uma empresa de eletrodomésticos gratifica seus vendedores com uma comissão proporcional às vendas efetuadas no mês. A comissão é de 2,5% se o total de vendas for de até R\$ 500,00 e de 4% se as vendas forem maiores que esse valor. Fazer o programa para ler o código do vendedor, nome, seu salário básico, o total de vendas no mês e calcular o salário recebido. Imprimir o nome, o salário básico, a comissão e o salário a ser recebido.
- g) Elaborar um programa que, dada a idade de um nadador, classifique-o em uma das seguintes categorias:
- Infantil A: 5-7 anos
  - Infantil B: 8-10 anos
  - Juvenil A: 11-13 anos
  - Juvenil B: 14-17 anos
  - Adulto: Maiores de 18 anos
- h) Um banco concederá um crédito especial a seus clientes de acordo com seu saldo médio nos últimos dois anos. Escrever um programa que leia o saldo médio nos últimos 2 anos e calcule o valor do crédito utilizando a tabela abaixo:



Saldo Médio (2 anos)	Percentual de Crédito
Até 200,00	Zero
De 201,00 a 400,00	20% do Saldo médio
401,00 a 600,00	30% do saldo médio
Acima de 600,00	40% do saldo médio

## 7 GERAÇÃO DE NÚMEROS ALEATÓRIOS

A linguagem C disponibiliza, por meio da biblioteca `stdlib.h`, uma função que gera números aleatórios. A função chama-se `rand()` e gera um número aleatório entre 0 e `RAND_MAX` (constante simbólica do C) < 32767, sendo que os valores gerados terão mesma probabilidade (DEITEL e DEITEL, 2015).

Para gerar um número em um determinado intervalo podemos usar o seguinte comando:

`X = rand() % valor desejado = Ajuste de escala`

Onde:

Valor desejado = fator de escala

Escalas geradas entre 0 e fator de escala -1

Abaixo segue alguns exemplos para que possamos entender melhor o funcionamento da função `rand()`.

```
/* Gerando três números aleatórios */
#include <stdlib.h>
#include <stdio.h>
main() {
    printf ("%i\n", (rand() % 6));
    printf ("%i\n", (rand() % 6));
    printf ("%i\n", (rand() % 6));
    return 0;
}
```

Tente executar várias vezes e veja o que acontece com os resultados. Note que em todas as execuções os números gerados se repetem N vezes. Isso ocorre pois a “semente” utilizada é a mesma. Para resolver este problema temos que usar a **função `srand()`** para introduzir uma nova semente e assim gerar números diferentes de forma aleatória. Isso é feito com o uso da função `time()` da biblioteca `time.h`. Esta função retorna um valor do relógio do sistema. Assim teremos a seguinte instrução:

`srand(time(NULL));`

Assim o novo código será alterado para que possamos gerar números aleatórios diferentes em cada execução. Veja o código alterado abaixo:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
main() {
    srand(time(NULL));
    printf ("%i\n", (rand() % 6));
    printf ("%i\n", (rand() % 6));
    printf ("%i\n", (rand() % 6));
}
```

Dica: Pesquise mais sobre semente aleatória.

**Atividade: Responda quais são as faixas numéricas geradas nas duas últimas linhas do programa abaixo. As respostas estão na vídeo aula da aula correspondente.**

```
/* Gerando três números aleatórios */
#include <stdlib.h>
#include <stdio.h>
main() {
    printf ("%i\n", (1+rand() % 6)); //gera um nro entre 1 e 6
    printf ("%i\n", (10+rand() % 100)); //gera um nro entre 10 e
109
    printf ("%i\n", (10+rand() % 101)); //gera um nro entre
.....complete
    printf ("%i\n", (rand() % 21-10)); //gera um nro entre
.....complete
    return 0;
}
```

## 7.1 Mão na Massa: JOGO DE ADIVINHAR NÚMEROS

Vamos elaborar um programa para gerar um número entre 1 e 10 e o usuário deverá ter três chances para adivinhar o número gerado. Use apenas as instruções vistas até o momento.

OBS: Veja a solução no vídeo aula correspondente.

## 7.2 DESAFIOS

### 1) NOVA VERSÃO DO PRGRAMA DE ADVINHAR NÚMEROS

Faça uma alteração no programa de adivinhar números da aula de nros Aleatórios. Para cada tentativa que o usuário errar o número dê uma dica, informando se o nro digitado é maior ou menor que o nro que o computador gerou. Por exemplo, se o computador gerou o nro 5 (cinco) e o usuário digitou 9 (nove), informe ao usuário, por meio de uma mensagem, que o número gerado é menor e assim por diante até completar as três tentativas.

### 2) JOGO DE PEDRA-PAPEL-TESOURA

Faça um programa em C para o jogo “pedra-papel-tesoura”. O jogo deve imprimir vitória, empate ou derrota conforme a opção que o jogador escolher e a opção que for sorteada aleatoriamente pelo computador.

Regra geral - Pedra ganha de tesoura; Tesoura ganha de papel; Papel ganha de pedra.

**Dica 1** - Mostrar as opções para o jogador escolher, e após a escolha, mostre a opção gerada pelo computador.

Exemplo: Usuário escolheu Tesoura e Computador escolheu Pedra. Pedra ganha de tesoura. Ganhador Computador.

**Dica 2** - Utilize a geração de números aleatórios com intervalo definido, com o uso da função `rand()` da biblioteca `stdlib`.

**Participe do fórum de discussão dos desafios e compartilhe sua solução e discuta com os colegas sobre suas estratégias e como solucionou este problema.**



## 8 LAÇOS DE REPETIÇÃO

As instruções de laços de repetição (Loopings) permitem repetições de instruções várias vezes, até que uma condição seja satisfeita e evitam escritas repetitivas de comandos (E/S, instruções condicionais, operações, etc). As instruções são:

for

while

do-while

A seguir vamos detalhar cada uma dessas instruções com suas características e exemplos para um melhor aprendizagem.

### 8.1 INSTRUÇÃO FOR()

Analise o seguinte problema. Escrever um programa que imprima os 10 primeiros números ímpares. Uma alternativa de solução seria como segue:

- 1) escrever um comando `printf("1, 3, 5, 7, 9, 11, 13, 15, 17, 19");`

Agora se o problema fosse, escrever um programa que imprima os 100 primeiros números ímpares. Da mesma forma uma solução alternativa seria a seguinte:

- 1) escrever um `printf("1, 3, 5, .... 199");`

Ainda, se precisássemos escrever um programa que imprima os N primeiros números ímpares. Talvez escrever um `printf` não seria suficiente, pois não saberíamos que número era e então diversas instruções IF seriam necessárias. Isso não é razoável em programação, assim podemos lançar mão do uso dos laços de repetição. Veja o programa abaixo para resolver o problema:

```
#include <stdio.h>
int main(void){
    int n;
    for(n = 1; n < 6; n++)
        printf("O %d número é: %d\n", n, 2*n-1);
}
```

Nesta solução usamos o laço de repetição **for** que irá executar a função de I/O **printf**, neste caso 5x num intervalo de n [1 até 5]. Abaixo segue a sintaxe da instrução **for**.

**Sintaxe:**

```
for (<variável contadora>=<valor inicial>; (<expressão lógica>);
contagem/incremento da variável contadora)
    <instrução>           //comando for para uma instrução
```

```
for (<variável contadora>=<valor inicial>; (<expressão lógica>);  
contagem/incremento da variável contadora){  
    <bloco de instruções>           //comando for para um bloco de  
    instruções  
}
```

Note que {} são os delimitadores de Bloco

O programa a seguir inclui 10 notas, verifica o número de alunos aprovados, reprovados e calcula a média da turma. Veja a sintaxe da instrução FOR

```
#include <stdio.h>  
main(){  
    int contador, conta_aprov, conta_rec;  
    float nota1, nota2, media, media_geral, soma;  
    conta_aprov = conta_rec = soma = 0;  
  
    for(contador = 0; contador < 10; contador++){  
        printf("Digite a nota 1: \n");  
        scanf("%f", &nota1);  
        printf("Digite a nota 2: \n");  
        scanf("%f", &nota2);  
        media = (float)(nota1+nota2*2)/3;  
        printf("Média: %.2f\n\n", media);  
        printf("*****");  
        soma+=media;  
        (media>6.0) ? conta_aprov++ : conta_rec++;  
    }  
    media_geral = (float)(soma/10);  
    printf("Nro de alunos aprovados: %d\n", conta_aprov);  
    printf("Nro de alunos reprovados: %d\n", conta_rec);  
    printf("Média geral da turma: %.2f\n", media_geral);  
}
```

O exemplo abaixo faz uso do Laço FOR com o comando Break. Veja que o comando Break força a saída do laço FOR.

```
#include <stdio.h>  
#include <conio.h>  
#include <ctype.h>  
main(){  
    int contador, conta_aprov, conta_rec;  
    float nota1, nota2, media, media_geral, soma;  
    conta_aprov = conta_rec = soma = 0;  
    for(contador = 1; contador < 51; contador++){  
        printf("Digite a nota 1: \n");  
        scanf("%f", &nota1);  
        printf("Digite a nota 2: \n");
```



```
scanf("%f", &nota2);
media = (float)(nota1+nota2*2)/3;
printf("Média: %.2f\n", media);
soma+=media;
(media>6.0) ? conta_aprov++ : conta_rec++;
printf("Deseja continuar? <S | N>");
if (toupper(getche()) == 'N')           //sentinela
    break;
}
media_geral = soma/contador;           //faz a média apenas do
que foi digitado, pois existe o break dentro do for
printf("Nro de alunos aprovados: %d\n", conta_aprov);
printf("Nro de alunos reprovados: %d\n", conta_rec);
printf("Média geral da turma: %.2f\n", media_geral);
}
```

O código abaixo faz uso do Laço FOR com o comando **Continue**. Veja que o comando continue realiza um desvio para o início do laço FOR e incrementa a variável de controle.

```
#include <stdio.h>
int main(){
    int contador, conta_aprov, conta_rec, conta_invalidos;
    float nota1, nota2, media, media_geral, soma;
    conta_aprov = conta_rec = soma = conta_invalidos = 0;
    for(contador = 1; contador < 51; contador++){ //controle
por contador
        printf("Digite a nota 1: \n");
        scanf("%f", &nota1);
        printf("Digite a nota 2: \n");
        scanf("%f", &nota2);
        if (nota1 == 0 || nota2 ==0 || nota1 > 10 || nota2 >
10){
            conta_invalidos++;
            continue; //força uma nova iteração e
incrementa a variável de controle
        }
        media = (float)(nota1+nota2*2)/3;
        printf("Média: %.2f\n", media);
        soma+=media;
        (media>6.0) ? conta_aprov++ : conta_rec++;
    }
    media_geral = soma/(contador - conta_invalidos); //desconta
as notas invalidas
    printf("Nro de alunos aprovados: %d\n", conta_aprov);
    printf("Nro de alunos reprovados: %d\n", conta_rec);
    printf("Média geral da turma: %.2f\n", media_geral);
}
```

## 8.2 FOR Encadeado

O Laço FOR encadeado é utilizado quando desejamos ter várias instruções FOR uma dentro da outra. Veja o exemplo abaixo:

Faremos um programa que imprime a tabuada de números de 1 a 9. Para que isso seja feito, teremos que ter um FOR para os números (regras) e outro laço para multiplicar o número de 1 até 10. Veja código abaixo. Verifique que para cada laço FOR existe uma variável de controle.

```
for (nro=1; nro<10; nro++){           //laço de controle dos nros
de 1 a 9
    printf("Tabuada do %d\n\n", nro);
    for (linha=1; linha<11; linha++) //laço de controle das
linhas
        printf("%d x %d = %d\n", nro, linha, nro*linha);
    printf("*****\n\n");
}
```

Veja que temos dois laços FOR e controlados por variáveis diferentes. No primeiro temos o FOR para a variável nro e o segundo para a variável linha. Veja como funciona na explicação da vídeo aula.

Ainda, caso tenhamos que implementar um laço FOR indeterminado, podemos utilizar a seguinte instrução abaixo. Verifique que deverá existir uma alternativa de saída do laço com o comando Break.

....

```
for(;;) {           //for indeterminado
    i++;
    printf("Estou dentro do laço indeterminado! Em %d
veze(s)\n", i);
    if (toupper(getche()) == 'N')
        break;      //para sair do laço for
}
```

## 8.3 LISTA DE EXERCÍCIOS

- 1) Alterar o programa exemplo de cálculo das médias das notas, que aparece a instrução **continue**, para identificar a menor e maior média dos alunos.
- 2) Imprimir os n primeiros números ímpares em ordem oposta

OBS: VEJA A CORREÇÃO NA AULA CORRESPONDENTE.

#### 8.4 DESAFIOS

- 1) Alterar o programa do cálculo das médias dos alunos para que seja um nro de entradas indeterminadas, fazendo o cálculo das médias gerais e nro de aprovados e reprovados com a apresentação da maior e menor nota e controle da digitação de notas inválidas.
  - 2) Refazer os programas das notas para que o usuário determine antecipadamente o número de alunos para digitação das notas.
  - 3) Escrever um programa que calcule o número de números ímpares de um determinado intervalo (N e M).
-

## 9 INSTRUÇÃO WHILE()

O uso da instrução **while** é muito semelhante ao FOR, uma vez que um trecho de código será executado enquanto a condição (expressão lógica) for verdadeira. Veja o exemplo abaixo. O bloco de instruções será executado até que o contador seja igual ou maior que 10, ou ainda, será executado enquanto a variável contador seja menor que 10.

```
#include <stdio.h>
main() {
    int contador, conta_aprov, conta_rec;
    float nota1, nota2, media, media_geral, soma;
    conta_aprov = conta_rec = soma = 0;
    contador=0;
    while (contador<10) {          // controle do laço de repetição
por contador
        contador++;
        printf("Aluno nro %d: \n", contador);
        printf("Digite a nota 1: \n");
        scanf("%f", &nota1);
        printf("Digite a nota 2: \n");
        scanf("%f", &nota2);
        media = (float)(nota1+nota2*2)/3;
        printf("Média: %.2f\n\n", media);
        printf("*****\n");
        soma+=media;
        (media>6.0) ? conta_aprov++ : conta_rec++;
    }
    media_geral = (float)(soma/contador);
    printf("Nro de alunos aprovados: %d\n", conta_aprov);
    printf("Nro de alunos reprovados: %d\n", conta_rec);
    printf("Média geral da turma: %.2f\n", media_geral);
}
```

Veja abaixo a sintaxe da instrução while, que deve receber como parâmetro uma expressão lógica. Verifique que existe uma variação para a instrução. Temos **DO .. WHILE**.

### Sintaxe:

```
while (<expressão lógica>)
    <instrução> // única instrução

while (<expressão lógica>){
    <bloco de instruções> //bloco de instruções
}

do
    <instrução> // única instrução
while (<expressão lógica>);
```

```
do {  
    <bloco de instruções>           //bloco de instruções  
} while (<expressão lógica>;
```

Note mais uma vez que o delimitador de bloco usamos {}

**Lembre-se os comandos Break e Continue também funcionam com os comandos while e do ..while**

Veja algumas perguntas que normalmente geram dúvidas quando do uso dessas instruções.

1. Quando usar o laço while? Quando não se sabe o número de vezes em que um bloco de instruções necessita ser repetido.
2. Qual a diferença entre while e do .. while? No while o bloco de instrução **pode não ser executado** se a condição lógica for falsa, e no do..while **pelo menos um vez o bloco** de instrução será executado.

Veja abaixo um exemplo do uso da instrução do..while, para ficar mais claro o seu comportamento em tempo de execução.

```
#include <stdio.h>  
main(){  
    int contador, conta_aprov, conta_rec;  
    float nota1, nota2, media, media_geral, soma;  
    conta_aprov = conta_rec = soma = 0;  
    contador=1;  
    do {  
        printf("Aluno nro %d: \n", contador);  
        printf("Digite a nota 1: \n");  
        scanf("%f", &nota1);  
        printf("Digite a nota 2: \n");  
        scanf("%f", &nota2);  
        media = (float)(nota1+nota2*2)/3;  
        printf("Média: %.2f\n\n", media);  
        printf("*****\n");  
        soma+=media;  
        (media>6.0) ? conta_aprov++ : conta_rec++;  
        contador++;  
    }while (contador<=10);           // controle do laço de  
    repetição por contador  
  
    media_geral = (float)(soma/contador);  
    printf("Nro de alunos aprovados: %d\n", conta_aprov);  
    printf("Nro de alunos reprovados: %d\n", conta_rec);  
    printf("Média geral da turma: %.2f\n", media_geral);  
}
```

Mais um exemplo, agora alterando para que as instruções sejam executadas dentro da instrução **While** até que o usuário digite qualquer caractere diferente de "S". Veja que não é necessário incluir uma instrução IF com a instrução Break.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
    int contador, conta_aprov, conta_rec;
    float nota1, nota2, media, media_geral, soma;

main(){
    conta_aprov = conta_rec = soma = 0;
    contador=0;

    do{ //laço de 1 a 10 (10 iterações)
        contador++;
        printf("\nDigite a nota 1: ");
        scanf("%f", &nota1);
        printf("Digite a nota 2: ");
        scanf("%f", &nota2);
        media = (float)(nota1+nota2*2)/3;
        printf("Media: %.2f\n", media);
        printf("*****\n");
        soma+=media;
        (media>6.0) ? conta_aprov++ : conta_rec++;
        printf("\nDeseja continuar? <S | N>");

        }while (toupper(getche()) != 'S' );

    media_geral = (float)(soma/contador);
    printf("\nNro de alunos aprovados: %d", conta_aprov);
    printf("\nNro de alunos reprovados: %d", conta_rec);
    printf("\nMedia geral da turma: %.2f", media_geral);
}
```

Resumindo....

- A instruções **FOR** é mais usada quando sabemos o número de repetições
- A instruções **WHILE** é mais usada para quando não sabemos o número exato de repetições
- Quando usamos a instrução **DO-WHILE** garantimos que, pelo menos uma vez, as instruções do laço serão executadas
- Lembre-se que, para cada Instrução **FOR**, deve haver uma variável de controle (contador)
- As instruções de repetição podem ser controladas por contador ou controladas por sentinela (variáveis que são testadas a cada laço)

**Ex. de sentinela:**

.....



```
printf("Entre com o grau, -1 para finalizar:");  
scanf("%d", &grau);  
while (grau != -1) {  
    total = total + grau; contador = contador + 1;  
    printf("Entre com o grau, -1 para finalizar:");  
    scanf("%d", &grau);  
}  
....
```

---

#### **Leitura Adicional:**

Capítulos 3 e 4, Paul J. Deitel e Harvey M. Deitel. **Como Programar em C.**

---

### **9.1 LISTA DE EXERCÍCIOS**

- 1) Escrever um programa que leia o nome, sexo e altura até que o usuário deseje sair. Após fazer o resumo com a quantidade de pessoas do sexo masculino e feminino, bem como as médias das alturas para cada sexo.
- 2) Escrever um programa que leia um número inteiro até que a soma de todos os números digitados for maior que 500.
- 3) Escrever a um programa que leia o salário inicial e faça aumentos sucessivos de 10% até que o valor do salário atualizado seja maior que o dobro do salário inicial.

### **9.2 DESAFIO**

Altere o programa do desafio do jogo Pedra-Papel-Tesoura para que o usuário possa jogar um número de vezes que desejar. Ao final o programa deve relatar o número de vezes que o jogador e o computador ganharam, ou seja, gerar um placar.

## 10 ESTRUTURA DE DADOS HOMOGÊNEAS

Até agora vimos a declaração de variáveis simples (primitivas), em que cada declaração possui um identificador, Tipo de dados e as informações são monovaloradas (aceitam apenas um valor).

### Problematização:

Armazenar notas finais de 10 alunos e compará-las com a média geral informando se a nota final do aluno está acima ou abaixo da média geral.

### Como resolver?

1. Uma solução possível é criar 10 variáveis de notas, entretanto esta solução torna o código mais complexo e extenso e de difícil manutenção e evolução.
2. Uma solução alternativa mais viável é usar uma estrutura de dados que possa armazenar um conjunto de valores

Desta forma, usaremos as estruturas de dados para armazenar mais de um valor em uma variável. Os tipos de Estruturas de Dados Homogêneas que iremos usar são:

- Vetores (arrays unidimensionais)
- Matrizes (arrays bidimensionais)

É importante salientar que as estruturas de dados envolvem, além da estrutura de armazenamento, a lógica das regras de formação, de manipulação e de acesso de dados. Um tipo de estrutura de dados são ditas homogêneas, pois armazenam um conjunto de dados de mesmo tipo (int, float, char, ....). Elas são acessadas por um índice que indica a posição dentro desta estrutura e todas essas estruturas são manipuladas em memória. Existem outras estruturas de dados com características dinâmicas, tais como: Pilhas, Filas, listas, árvores, grafos, mas não veremos nesta disciplina.

### 10.1 VETORES

Veja abaixo como declaramos uma estrutura de dados do tipo vetor. Sempre associamos um tipo e o número de elementos que desejamos.

#### Declaração

<tipo de dados> identificador[nro elementos];

Veja um exemplo que cria uma sequência de 10 valores na memória do tipo float:.

```
float notas[10];
```

Podemos entender um vetor como uma representação lógica, representado por uma lista de valores do mesmo tipo e indexado por um índice em que o primeiro elemento tem índice 0 e o último terá índice N-1.

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
V1	V2	V3	V4	V5	V6	V7	V8	V9	V10

Analise abaixo o programa que tem declarado um vetor de notas com 10 elementos do tipo float. O programa realiza a entrada de 10 notas e armazena no vetor chamado notas, faz o calculo da média geral da turma e após compara todas as notas digitas com a média geral e mostra aquelas que estão acima da média geral da turma.

```
#include <stdio.h>
int i;
float notas[10], media_geral, soma_notas;

main(){
    soma_notas=0;
    for (i=0; i<10; i++){
        printf("Informe a nota: ");
        scanf("%f", &notas[i]);
        soma_notas+=notas[i];
    }
    media_geral = (float) soma_notas / 10;
    printf("\n\nA média geral da turma foi %f\n\n",
media_geral);

    for (i=0; i<10; i++)
        if (notas[i]>media_geral)
            printf("Nota do aluno de indice %d = %f foi maior
que a media da turma\n", i, notas[i]);
}
```

O programa abaixo mostra uma solução para uma pesquisa nos elementos do vetor produtos e utiliza uma variável FLAG para determinar se o código foi encontrado.

```
#include <stdio.h>
main(){
    int i, flag, codigo;
    int produtos[10];

    for (i=0; i<10; i++){
        printf("Informe o codigo do produto: \n");
        scanf("%d", &produtos[i]);
    }
    flag=0;    //utilizado para sinalizar se achou ou não achou
    printf("Informe o código para a pesquisa\n");
    scanf("%d", &codigo);
    for (i=0; i<10; i++)
        if (produtos[i]==codigo) {
            flag=1;
            printf("O Código foi encontrado\n\n");
        }
    if (flag==0)
        printf("Código do produto não encontrado");
}
```

O programa abaixo ilustra a declaração de um vetor de números do tipo DOUBLE com uma constante e o preenche com números de 1 a 9 em ordem decrescente.

```
#include <stdio.h>

#define TAM_MAX 10
int i;
double vetReais[TAM_MAX];

main(){
    for (i=0; i<TAM_MAX; i++)
        vetReais[i]=TAM_MAX - i; //inseri 10, 9, ... 1

    for (i=0; i<TAM_MAX; i++)
        printf("%.0f-", vetReais[i]);

}
```

## 10.2 Mão na Massa

Vamos fazer um programa que leia uma string de tamanho 100 caracteres e contar quantas vogais existem. O programa deve mostrar uma saída formatada como segue:

```
Vogal A: 01
Vogal E: 03
Vogal I: 01
Vogal O: 00
Vogal U: 00
--
Total: 05
```

**Dica:** Usar FOR, ler até encontrar “\0”, usar *toupper* da biblioteca ctype.h e /t no printf para as tabulações.

Ex: for (caractere=0; texto[caractere]!='\0'; caractere++) ...

OBS: Veja o programa que resolve esta questão no vídeo aula correspondente

## 10.3 LISTA DE EXERCÍCIOS

1) Escreva um programa em C para solucionar os seguintes problemas:

- Escrever um programa que leia um vetor de 10 números quaisquer e após imprima todos os valores iguais ou maiores a 10 e suas respectivas posições no vetor (índice).
- Escrever um programa que leia um vetor de números inteiros positivos e negativos e após imprima quantos números negativos e positivos existem dentro do vetor.

- c) Escrever um programa que leia um vetor de números inteiros positivos e após imprima o maior valor positivo dentro do vetor e sua posição.

## 10.4 DESAFIOS

**Desafio 1:** Escreva um programa que faça uma cópia de um vetor para outro vetor com seus valores em ordem inversa (ex. num vetor de 10 posições o valor do índice 9 deve estar no valor do índice 0).

**Desafio 2:** Substring: Programa que lê uma string e separa substring de tamanho N. (Erro). Ache o erro no programa e corrija-o.

```
#include <stdio.h>
main()
{
    char texto[101] = {};
    int n = 0;
    int caractere = 0;
    int contador = 0;
    int posini = 0;
    int j=0;
    printf("Digite um texto: \n");
    gets(texto); // lê a string com espaços (scanf não armazena
    espaços)
    printf("\nDigite um numero inteiro positivo: \n\n"); //Lê
    um nro que será o tamanho da substring
    scanf("%d", &n);

    for (caractere=0; texto[caractere]!='\0'; caractere++) {
        //Leitura dos caracteres da string \0 Fim da string
        if (texto[caractere] == ' ') {
            if (contador>=n) {
                for (j=posini; j<caractere; j++) printf("%c",
                texto[j]);

                printf("\n");
                contador=-1;
            }
            posini=caractere+1;
        }
        contador++;
    }
    if (contador>=n) {
        for (j=posini; j<caractere;j++) printf("%c",
        texto[j]);
        printf("\n");
    }
}
```

}

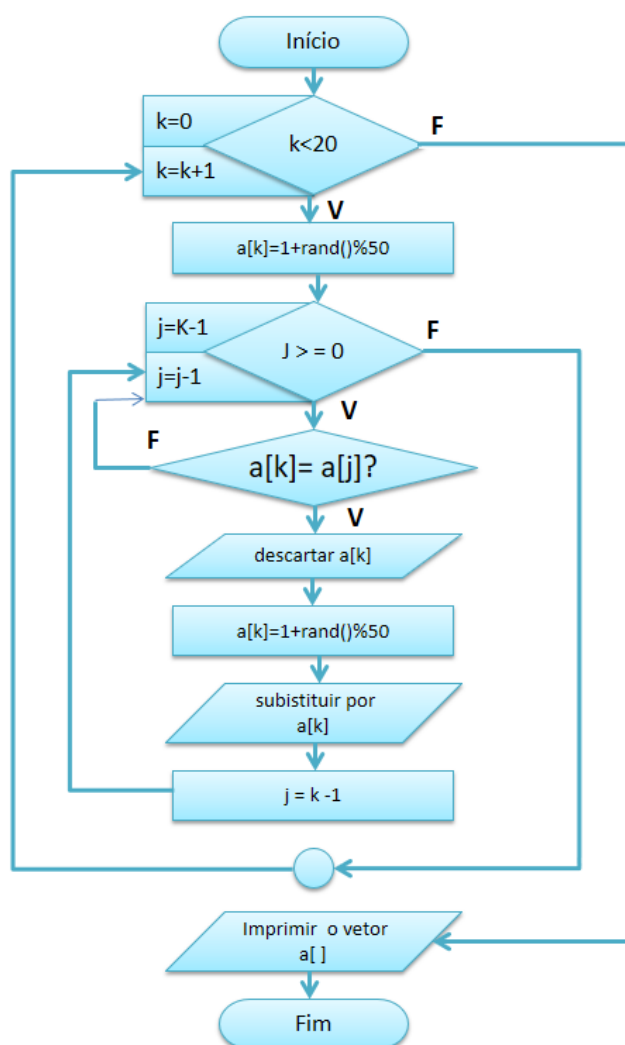
**Desafio 3:** Preencher um vetor de X elementos com números aleatórios de intervalo N a M. O Algoritmo está disponibilizado abaixo, analise o fluxograma e depois escreva o programa.

Primeiro: Com repetição (barbada)

Segundo: Sem repetição (ver ao fluxograma com o algoritmo ou crie sua própria solução)

**Dica:** usar pelo menos três variáveis para controlar o número de elementos e o intervalo dos números aleatórios

OBS: pode fazer dentro de um intervalo definido, por exemplo: números aleatórios entre 1 e 50, num vetor de 30 números inteiros.



**Leitura Adicional:**

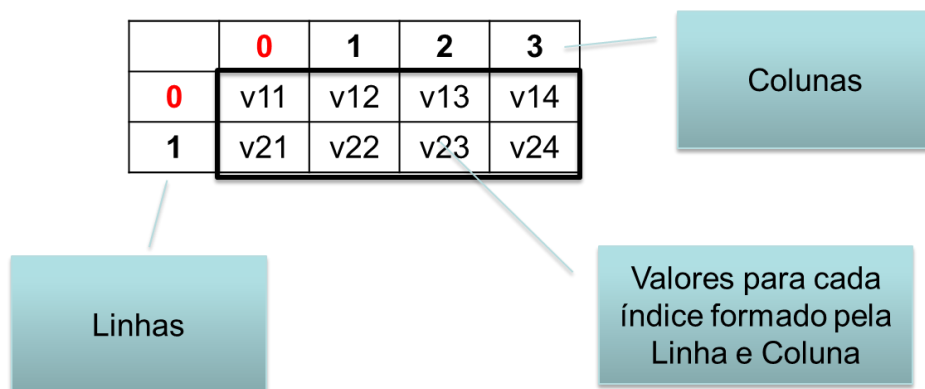
---

Capítulo 8 do Livro: DEITEL, P; DEITEL, H. **Como Programar em C.** São Paulo: Pearson, 2015.

---

## 11 MATRIZES

Matrizes são arrays bidimensionais que possuem acesso por meio de dois índices (linha e coluna) e permitem armazenamentos na forma de tabelas de valores de um mesmo tipo. Veja a figura abaixo que representa uma matriz de forma lógica. Verifique que agora existem dois índices, um para linha e outro para coluna. Da mesma forma que no caso dos vetores, os índices iniciam em 0 (Zero) e terminam em N-1.



### Declaração

<tipo de dados> identificador[nro linhas][nro de colunas];

No exemplo abaixo é criada uma sequência de 100 valores na memória do tipo float, sendo que serão 10 linhas com 10 colunas cada.

```
float notas[10][10];
```

Os índices podem ser quaisquer valores numéricos inteiros ou resultados de expressões aritméticas, desde que não extrapolem os limites estabelecidos na declaração da matriz (vale também para os vetores).

O programa abaixo declara uma matriz de notas 5 por 5 e a preenche com valores do tipo float. Após o preenchimento o programa calcula a média de cada aluno (5 notas para cada).

```
#include <stdio.h>
#define TAM_MAX_LIN 5           //cinco alunos
#define TAM_MAX_COL 5           //cinco notas
int lin, col;
float notas[TAM_MAX_LIN][TAM_MAX_COL], media;
main(){
    //entrada de dados
    for (lin=0; lin<TAM_MAX_LIN; lin++)           //controle
do aluno
        for (col=0; col<TAM_MAX_COL; col++){ //entrada das
notas (no exemplo 10 notas por aluno
```



```
        printf("Informe a nota %d do aluno %d \n", col+1,
lin+1);
        scanf("%f", &notas[lin][col]);
    }

    for (lin=0; lin<TAM_MAX_LIN; lin++){ //passo para calcular
média de cada aluno
        media=0;
        for (col=0; col<TAM_MAX_COL; col++) //soma
todas as notas do aluno
            media+=notas[lin][col];
        printf("A media do aluno %d foi de %.2f\n", lin+1,
(float)media/TAM_MAX_COL);
    }
}
```

### 11.1 String

Analise o caso das declarações de uma string em C. Na maioria das vezes uma string é declarada como um vetor de char. Veja abaixo.

```
char nome[50];
```

Na verdade, o que existe é uma matriz de string, sendo que as linhas representam o número da string e as colunas o número de caracteres de cada string. Sendo que o acesso às strings se dá apenas pelo uso do índice de linha. Veja exemplo abaixo, um fragmento de um programa:

```
char nomes[10][50];          //cria uma matriz de 10 nomes com 50 caracteres
....
scanf("%s", &nomes[i]);
...
```

### 11.2 LISTA DE EXERCÍCIOS

1) Escreva um programa em C para solucionar os seguintes problemas:

- Escrever um programa que leia uma matriz 3x2 de números inteiros e calcule a soma de todos os elementos (valores) da matriz.
- Escrever um programa que leia uma matriz 4x4 e calcule a soma de todos os elementos, a soma dos elementos da diagonal principal.
- Escrever um programa que leia uma matriz 30x2. As linhas representam os dias e as colunas as entradas (receitas) e despesas (saídas) e calcule o saldo atual (entradas – saídas).

OBS: Veja a demonstração na videoaula de como fazer a entrada de dados de forma automática por meio de um arquivo .txt e pelo prompt de comando do sistema operacional. Uma forma mais rápida para fazer os testes quando envolvemos muitas informações.

### 11.3 DESAFIO

Uma matriz quadrada é dita triangular inferior se todos os elementos situados acima de sua diagonal principal são nulos. Uma matriz é dita triangular superior se todos os elementos situados abaixo da diagonal principal são nulos.

Escreva um programa C que a partir de uma dimensão N de uma matriz quadrada NxN (N máximo = 30) descubra se a matriz é triangular inferior ou superior ou apenas quadrada

Considere nulos os valores com Zero.

Para os testes deste programa use as informações de entrada a seguir. Conforme demonstrado no vídeo aula, você pode fazer a entrada de dados por meio de um arquivo texto (.txt). Assim, poderá criar três arquivos textos com as informações a serem testadas.

Informações de Entrada 1, com N=5:

```
5
1 1 1 1 1
0 1 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

Resultado Esperado: Matriz triangular Superior

Informações de Entrada 2, com N=5:

```
5
1 0 0 0 0
1 1 0 0 0
1 1 1 0 0
1 1 1 1 0
1 1 1 1 1
```

Resultado Esperado: Matriz Triangular Inferior

Informações de Entrada 3, com N=5:

5

1 0 1 1 1

1 1 0 1 1

1 1 1 0 1

1 0 1 1 1

1 1 0 1 1

Resultado Esperado: Matriz quadrada

## 12 ESTRUTURA DE DADOS HETEROGÊNEAS

As estruturas de Dados Heterogêneas permitem agrupamentos de variáveis de tipos diferentes. Tente responder esta questão: Qual a diferença das estruturas heterogêneas e os vetores e matrizes? Vimos que as estruturas de dados homogêneas criam sequências de dados do mesmo tipo e as heterogêneas podem criar sequências de dados de tipos diferentes. Isso é muito útil para solucionar problemas, como por exemplo, cadastros, etc.

Tais estruturas são referenciadas como campos e são tratados como um tipo de dado definido pelo programador. Após a declaração deve-se associar uma variável ao tipo criado (identificador) e utilizamos o ponto (.) para identificar cada elemento do tipo definido pelo usuário.

### Declaração

```
struct
    identificador {
        tipodado1 identificador1;
        tipodado2 identificador2;
        tipodadon identificadorn;
    };
```

Veja um Exemplo em C, que declara um tipo construído chamado RegistroAluno que é composto por três campos: "id" de tipo inteiro; "nome" de tipo string com 50 caracteres e "endereço" também de tipo string com 50 caracteres. Após este tipo é associado a uma variável aluno que a partir de sua criação assume a estrutura do tipo registroAluno. Veja como são acessados os campos de registroAluno.

```
#include <stdio.h>
struct
    registroAluno {          //tipo construído
        int id;
        char nome[50];
        char endereco[50];
    };
registroAluno aluno;

main() {
    scanf("%d", &aluno.id);
    scanf("%s", aluno.nome);    //qual o problema do scanf no
tratamento de string
    scanf("%s", aluno.endereco);

    printf("Id do ALuno.....: %i\n", aluno.id);
    printf("Nome do ALuno.....: %s\n", aluno.nome);
    printf("Endereco do ALuno...: %s\n", aluno.endereco);
}
```

Uma dúvida que geralmente ocorre é como criar um vetor de um tipo Struct. Veja o trecho do código abaixo em que é declarado um vetor com 10 (dez) elementos de registroAluno.

```
#include <stdio.h>
struct
    registroAluno {          //tipo construído
        int id;
        char nome[50];
        char endereco[50];
    };
    registroAluno aluno[10];

main() {

    for (int i=0; i<10; i++) {
        printf("Digite as informacoes de ID, Nome e Endereco
do Aluno: ");
        scanf("%d %s %s", &aluno[i].id, aluno[i].nome,
aluno[i].endereco);
    }

    for (int i=0; i<10; i++) {
        printf("Aluno %i - %s - %s\n", aluno[i].id,
aluno[i].nome, aluno[i].endereco);
    }
}
```

## 12.1 LISTA DE EXERCÍCIOS

Escreva um programa em C para solucionar os seguintes problemas:

1. O sistema deve apresentar ao usuário um menu com as seguintes opções: 1-Incluir; 2-Alterar; 3-Excluir e 4-Imprimir na tela, o cadastro de N produtos de um determinado departamento de uma loja. Para o cadastro deve ser utilizado uma estrutura de dados tipo registro com as seguintes informações: Código do Produto, Nome do Produto, Valor de Custo, Valor de Venda, Quantidade em Estoque.

## 13 REFERENCIAS

DEITEL, P; DEITEL, H. C Como Programar. São Paulo: Pearson, 2015.

FORBELLONE, André Luiz Villar. Lógica de Programação: a construção de algoritmos e estruturas de dados. São Paulo: Makron Books, 2005.

MARTINS, L.G.A. Apostila de Introdução a Algoritmos. Uberlândia: UFU Universidade Federal de Uberlândia, faculdade de Computação, 2010. Acesso em: nov/2018. Disponível em: [http://www.facom.ufu.br/~gustavo/IC/Programacao/Apostila\\_Algoritmos.pdf](http://www.facom.ufu.br/~gustavo/IC/Programacao/Apostila_Algoritmos.pdf).

Notas de aula INF01040 Introdução à Programação. UFRGS, Instituto de Informática.