

# Algoritmos e Programação

**Sarah de Oliveira Alcântara**

**[sarah.alcantara01@etec.sp.gov.br](mailto:sarah.alcantara01@etec.sp.gov.br)**

# Conteúdo do Curso

## Competências:

- ✓ Implementar algoritmos de programação.
- ✓ Utilizar linguagem de programação em ambiente de desenvolvimento.

## Habilidades:

- ✓ Elaborar algoritmos.
- ✓ Codificar programas, utilizando técnica de programação estruturada.

## Bases Tecnológicas:

<ul style="list-style-type: none"><li>1. Comandos da linguagem de programação:<ul style="list-style-type: none"><li>▪ Memória, tipos de dados e variáveis</li><li>▪ Entrada, saída e conversão de tipos</li><li>▪ Tratamento de erros e exceções</li><li>▪ Operadores aritméticos, relacionais e lógicos</li><li>▪ Expressões e tabela da verdade</li><li>▪ Funções pré-definidas</li></ul></li><li>2. Programação estruturada:<ul style="list-style-type: none"><li>▪ Decisão simples</li><li>▪ Decisão múltipla</li><li>▪ Iteração</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ Laços</li><li>▪ Teste de mesa</li><li>3. Programação modular:<ul style="list-style-type: none"><li>▪ Sub-rotinas</li><li>▪ Procedimentos e funções</li><li>▪ Argumentos e escopo de identificadores</li><li>▪ Recursividade</li></ul></li><li>4. Tipos de dados estruturados:<ul style="list-style-type: none"><li>▪ Vetores</li><li>▪ Matrizes</li><li>▪ Arquivos binários e de texto</li></ul></li></ul>
--	--

# Objetivos

## ✓ Objetivo Geral:

- Capacitar o aluno a visualizar soluções computacionais para problemas através da aplicação dos conceitos da lógica de programação e dotá-los da capacidade de construção de programas básicos em linguagem de alto nível estruturada (**linguagem C**).

# Objetivos

## ✓ Objetivos Específicos:

- Desenvolver o raciocínio lógico e abstrato do aluno;
- Familiarizar o aluno com o modelo sequencial de computação;
- Apresentar técnicas e linguagens para representação e construção de algoritmos simples;
- Apresentar conceitos básicos de linguagens de programação;
- Capacitar o aluno no uso da linguagem C;
- Treinar o aluno no processo básico de desenvolvimento de software (concepção, edição, execução e teste de programas de computador).

# Bibliografia

## ✓ Básica.

- ASCENCIO, A.F.G.; CAMPOS, E.A.V. Fundamentos da programação de computadores. 2<sup>a</sup> ed. Pearson Prentice Hall.
- FORBELLONE, V.L.A.; Lógica de Programação – A construção de Algoritmos e Estrutura de Dados. Pearson Education, 2000.

## ✓ Complementar.

- CARBONI, I.F. Lógica de programação. Thomson.
- CORMEN, T.H. et al. Algoritmos, teoria e prática. Campus, 2002.



# Introdução ao Ambiente Computacional

- ✓ **Computador** → É uma máquina capaz de possibilitar variados tipos de tratamento automático de informações ou processamento de dados.
- ✓ O que deve ser feito para que um determinado tratamento automático de informações ocorra?
  - Deve-se instruir o computador para que o mesmo utilizando-se de sua estrutura execute determinada tarefa.
  - Como?
    - Software (programas).

# Introdução ao Ambiente Computacional

- ✓ **Nosso objetivo** → Aprender conceitos básicos para desenvolver programas para computadores.
  - **Exemplos:** sistemas bancários, sistemas de restaurantes, cálculos avançados entre outros.
- ✓ Roteiro para desenvolver programas:

**Problema → Solução → Algoritmo → Programa → Resultado**

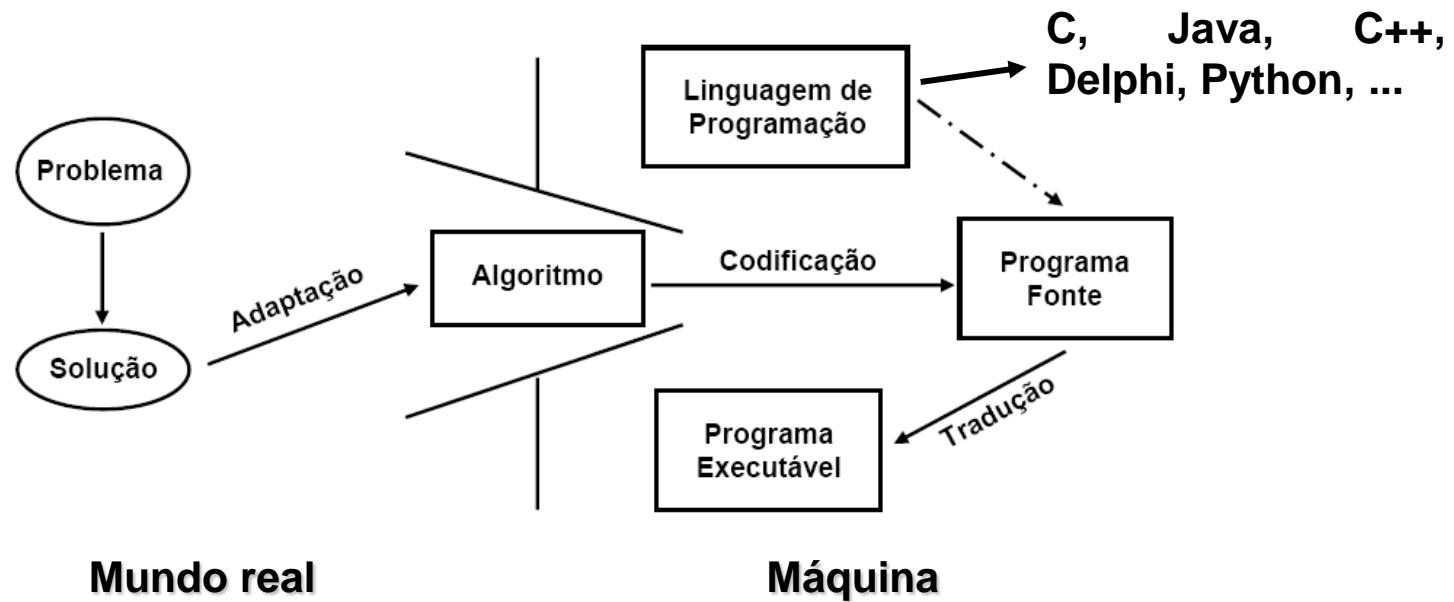
# Algoritmo e Programação

## ✓ Definições:

- **Algoritmo** → Conjunto de regras e operações bem definidas e ordenadas, destinadas à solução de um problema, ou de uma classe de problemas, em um número finito de etapas → **Representação de uma solução para um problema.**
- **Programa** → Sequencia completa de instruções a serem executadas por um computador → **De acordo com um algoritmo.**

# Algoritmo e Programação

- ✓ O **algoritmo**, do ponto de vista computacional, tem um papel fundamental por ser o elo de ligação entre dois mundos (real e computacional).
- ✓ A atividade de programação começa com a construção do algoritmo.



# Algoritmo e Programação

- ✓ Exemplos de algoritmos.

## Algoritmo: trocar lâmpada

**Passo 1:** pegar a lâmpada nova.

**Passo 2:** pegar a escada.

**Passo 3:** posicionar a escada embaixo da lâmpada queimada.

**Passo 4:** subir na escada com a lâmpada nova.

**Passo 5:** Retirar a lâmpada queimada.

**Passo 6:** Colocar a lâmpada nova.

**Passo 7:** Descer da escada.

**Passo 8:** Ligar o interruptor.

**Passo 9:** Guardar a escada.

**Passo 10:** Jogar a lâmpada velha no lixo.

## Algoritmo: sacar dinheiro

**Passo 1:** ir até o caixa eletrônico.

**Passo 2:** colocar o cartão.

**Passo 3:** digitar a senha.

**Passo 4:** solicitar o saldo.

**Passo 5:** se o saldo for maior ou igual à quantia desejada, sacar a quantia desejada; caso contrário sacar o valor do saldo.

**Passo 6:** retirar dinheiro e cartão.

**Passo 7:** sair do caixa eletrônico.

# Métodos de Representação de Algoritmos

- ✓ Existem duas formas de representação de algoritmos:
  - **Fluxograma** → Representação gráfica.
  - **Pseudocódigo (Português estruturado)** → Representação textual.

# Métodos de Representação de Algoritmos

## ✓ Características.

### – Fluxograma.

- A representação gráfica é mais concisa que a representação textual.
- É necessário aprender a simbologia dos fluxogramas.

### – Pseudocódigo.

- A transcrição para qualquer linguagem de programação é quase direta.
- É necessário aprender as regras do pseudocódigo.

# Métodos de Representação de Algoritmos

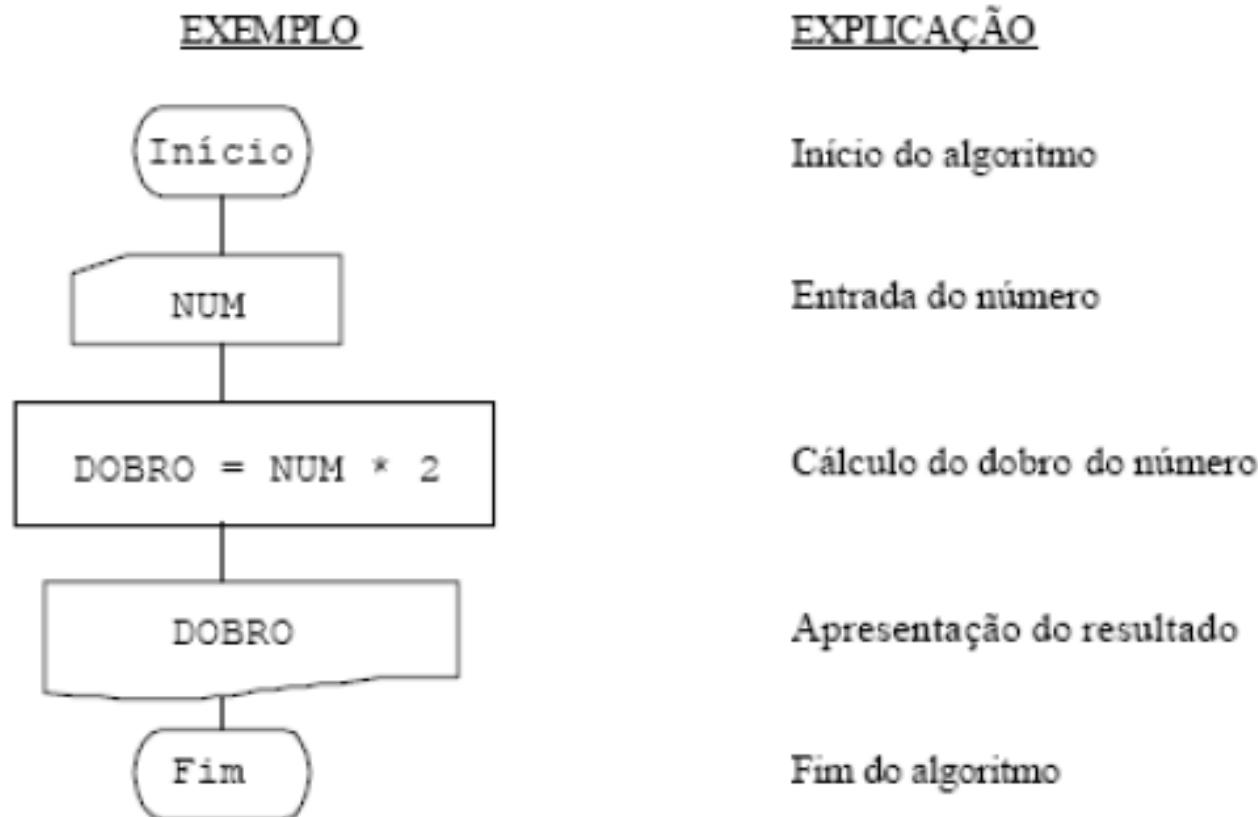
## ✓ Fluxograma.

- Representação gráfica por meio de símbolos geométricos, da solução algorítmica de um problema.



# Métodos de Representação de Algoritmos

## ✓ Exemplo - Fluxograma.



# Métodos de Representação de Algoritmos

## ✓ Pseudocódigo.

- Descrição narrativa utilizando nosso idioma para descrever o algoritmo.
- **Exemplo de uma descrição narrativa.**
  - Soma de dois números.
    1. Receber os dois números.
    2. Efetuar a soma dos dois números.
    3. Mostrar o resultado.

# Métodos de Representação de Algoritmos

## ✓ Exemplo – Descrição narrativa (Visualg).

algoritmo "soma dois numeros"

// Função :

// Autor :

// Data : 30/3/2010

// Seção de Declarações

var

n1, n2, d: inteiro

inicio

// Seção de Comandos

escreval("Digite dois numeros")

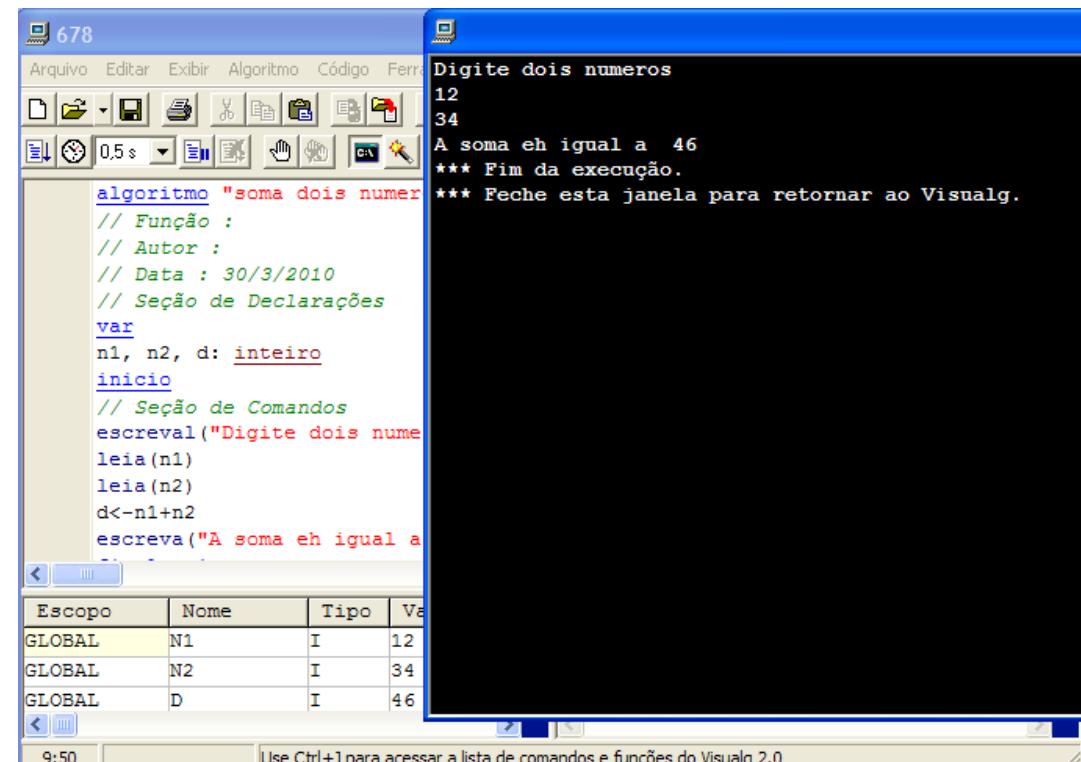
leia(n1)

leia(n2)

d<-n1+n2

escreva("A soma eh igual a ", d)

fimalgoritmo



The screenshot shows the VisualG 2.0 interface. On the left, the 'algoritmo "soma dois numeros"' window displays the pseudocode. On the right, the terminal window shows the execution results.

**algoritmo "soma dois numeros"**

// Função :  
// Autor :  
// Data : 30/3/2010  
// Seção de Declarações  
var  
n1, n2, d: inteiro  
inicio  
// Seção de Comandos  
escreval("Digite dois numeros")  
leia(n1)  
leia(n2)  
d<-n1+n2  
escreva("A soma eh igual a ", d)  
fimalgoritmo

Digite dois numeros  
12  
34  
A soma eh igual a 46  
\*\*\* Fim da execução.  
\*\*\* Feche esta janela para retornar ao Visualg.

Escopo	Nome	Tipo	Valor
GLOBAL	N1	I	12
GLOBAL	N2	I	34
GLOBAL	D	I	46

# Métodos de Representação de Algoritmos

## ✓ Resumindo.

- Escrever algoritmos e, por fim, programar, consiste em dividir qualquer problema em vários **passos** menores, usando uma ou mais formas de representação.
- Esses **passos** que compõem o algoritmo são denominados de **comandos**.

# Conceituação de Elementos Básicos para Construção de um Algoritmo

## ✓ Constante.

- Valores fixos, tais como números. Estes valores não podem ser alterados pelas instruções do algoritmo, ou seja, é um espaço de memória cujo valor não deve ser alterado durante a execução do programa.
- Exemplo:
  - Inteiro → 10, -23768, ...
  - Real → -2.34, 0.149, ...
  - Caractere → “k”, “computador”

# Conceituação de Elementos Básicos para Construção de um Algoritmo

## ✓ Variável.

- Elemento de dado cujo valor pode ser modificado ao longo de sua execução.
- Uma variável representa uma posição na memória e pode ter tipo (inteiro, caractere, real), tamanho (16, 32 bits, ...) e nome definidos.

# Conceituação de Elementos Básicos para Construção de um Algoritmo

## ✓ Identificadores.

- Nomes utilizados para referenciar variáveis, funções ou vários outros objetos definidos pelo programador.
- Exemplo:
  - letras, dígitos e sublinhado(\_);
  - Não podem começar com dígito;
  - Não podem ser iguais a uma palavra-chave e nem iguais a um nome de uma função declarada pelo programador ou pelas bibliotecas da linguagem utilizada.

# Conceituação de Elementos Básicos para Construção de um Algoritmo

- ✓ **Palavras-reservadas (palavras-chave).**
  - São identificadores predefinidos que possuem significados especiais para o interpretador do algoritmo.

```
inicio    senao    para    enquanto
var       logico   se      ate
faca      inteiro  real
```

# Conceituação de Elementos Básicos para Construção de um Algoritmo

## ✓ Tipos primitivos.

- Palavra-reservada: **logico** - define variáveis do tipo booleano, ou seja, com valor VERDADEIRO ou FALSO.
- Palavra-reservada: **caractere** – define variáveis do tipo string, ou seja, cadeia de caracteres.
- Palavra-reservada: **inteiro** - define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais.
- Palavra-reservada: **real** - define variáveis numéricas do tipo real, ou seja, com casas decimais.

# Declaração de Variáveis

- ✓ Palavra-reservada: **var** - utilizada para iniciar a seção de declaração de variáveis.
  - Exemplo:

```
var a: inteiro  
      nome_do_aluno: caractere  
      sinalizador: logico  
      valor1, valor2: real
```

# Declaração de Variáveis

## ✓ Regra para criar nomes de variáveis.

- Os nomes das variáveis devem representar o que será guardado dentro dela.
- O primeiro caractere de um nome deverá ser sempre alfabético.
- Não podem ser colocados espaços em branco no nome de variáveis, usar o UNDERSCORE “\_”.
- A declaração de uma variável é feita no algoritmo informando o seu nome, seguido por : e terminado com o seu tipo.

# Operadores e Hierarquia nas Operações

<-	Atribuição. x <- 2. A variável x recebeu o valor 2. Logo x = 2
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
a\b	Retorna o quociente da divisão inteira de a por b
a%b	Retorna o resto da divisão inteira de a por b
a^b	Retorna o valor de a elevado a b
a^1/b	Retorna a raiz b de a
aleatorio (a)	Retorna um número aleatório, em intervalo fechado, entre 0 e a

Hierarquia	Operação
1	Parênteses
2	Função
3	-, + (unários)
4	^
5	*, /, \, %
6	+, -

## Exemplos:

$3/4+5 = 5.75$ $3\backslash 2 * 9 = 9$ $11\% (3^2) = 2$ $3\backslash 2 + (65-40)^{(1/2)} = 6$	$3/(4+5) = 0.33333333$ $11\% 3^2 = 2$ $(11\% 3)^2 = 4$
--	--

# Operadores Relacionais e Lógicos

Operador	Ação
>	maior que
$\geq$	maior ou igual a
<	menor que
$\leq$	menor ou igual
=	igual a
$\neq$	diferente de

Operador
e
ou
nao
xou

## Exemplos:

$3 > 7 = \text{FALSO}$

“A” = “a” = VERDADEIRO

“a” > “B” = FALSO

$(3 \geq 13 \setminus 4) \text{ xou } (\text{nao } (5 \% 2 = 0)) = \text{FALSO}$

# Criando um Algoritmo

## ✓ Os passos necessários para a construção de um algoritmo:

- ler atentamente o enunciado do problema, compreendendo-o e destacando os pontos mais importantes;
- definir os dados de entrada, ou seja, quais dados serão fornecidos;
- definir os dados de saída, ou seja, quais dados serão gerados depois do processamento;
- definir o processamento, ou seja, quais cálculos serão efetuados e quais as restrições para esses cálculos. O processamento é responsável pela obtenção dos dados de saída com base nos dados de entrada;
- definir as variáveis necessárias para armazenar as entradas e efetuar o processamento;
- elaborar o algoritmo;

# Pseudocódigo

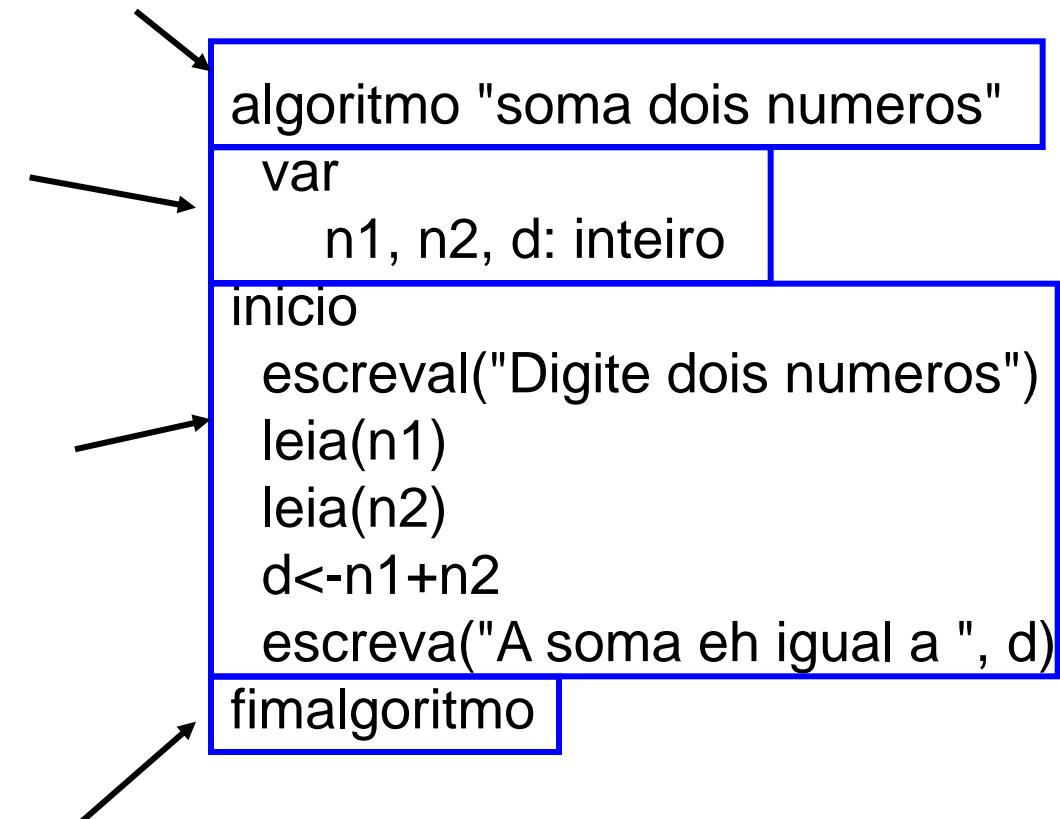
- ✓ **Método de representação de algoritmos** → Utilizaremos pseudocódigo.
- ✓ **Pseudocódigo** → Técnica textual de representação de um algoritmo - Também conhecida como Português Estruturado ou **Portugol**.
- ✓ Técnica é baseada em uma PDL (*Program Design Language*), que é uma linguagem genérica na qual é possível representar um algoritmo de forma semelhante à das linguagens de programação.

# Estrutura de um Algoritmo

**NOME DO ALGORITMO**

**VAR**

declaração de variáveis



**INICIO DO ALGORITMO**

bloco de comandos

**FIM DO ALGORITMO**

# Estruturas de Controle de Fluxo

- ✓ Os algoritmos desenvolvidos até o momento constituem uma seqüência de ações que sempre são executadas em sua totalidade indiferente do valor da entrada de dados.
  
- ✓ Para a resolução de determinados problemas ou para a execução de determinadas tarefas é necessária a realização de um conjunto distinto de ações e este conjunto é definido com base na análise da entrada de dados.

# Estruturas de Controle de Fluxo

- ✓ **Exemplo:** um algoritmo capaz de efetuar o cálculo do imposto de renda devido por um determinado contribuinte. Neste caso dependendo da quantidade de dependentes, do valor de sua renda e outros fatores o cálculo será feito de formas distintas.

# InSTRUÇÃO CONDICIONAL

- ✓ Considere um problema que exija uma decisão.
  - Tomemos como exemplo uma divisão, onde haja a necessidade de que o algoritmo verifique se o divisor é igual ou diferente de zero. Se for igual não é possível dividir. Se for diferente é possível dividir.

- ✓ Sintaxe:

```
se (<expressão-lógica>) então  
    <sequência de comandos>  
senão  
    <seqüência de comandos>  
fimse
```

# InSTRUÇÃO CONDICIONAL

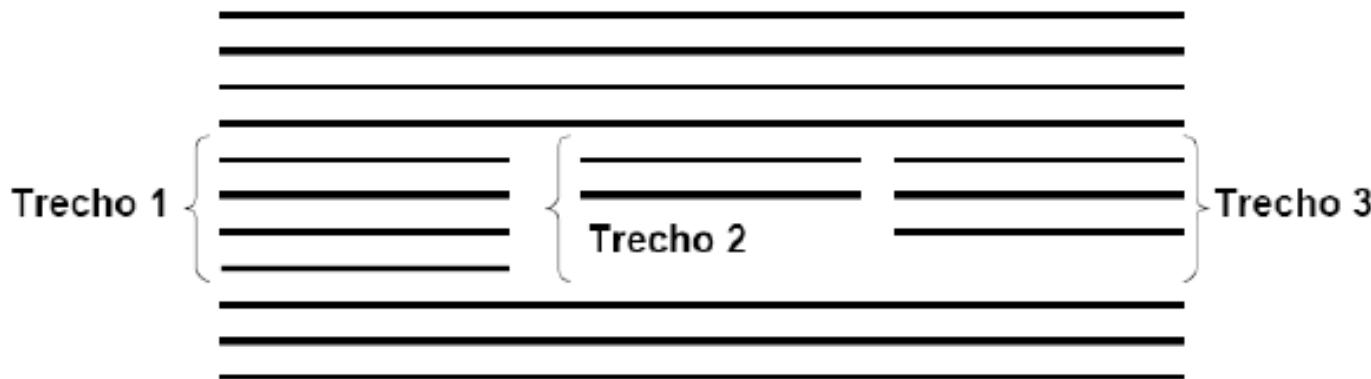
## ✓ Exemplo:

algoritmo "Divisão"

```
var
    n1, n2: inteiro
    resultado: real
inicio
    escreva ("Digite o dividendo ==> ")
    leia (n1)
    escreva ("Digite o divisor ==> ")
    leia (n2)
    se (n2=0) entao
        escreva ("impossivel dividir por 0")
    senao
        resultado <- n1/n2
        escreva ("O resultado eh ==>", resultado)
    fimse
fimalgoritmo
```

# Comando de Seleção Múltipla

- ✓ Em algumas situações ao chegarmos a uma determinada instrução de um algoritmo devemos selecionar um dentre alguns trechos a seguir, tendo como base para esta escolha um conjunto de valores.



- ✓ Para lidar com casos deste tipo foi criado o comando de seleção múltipla.

# Comando de Seleção Múltipla

✓ Sintaxe:

**escolha (<variável>)**

**caso <valor11>, <valor12>, ..., <valor1n>**

**<seqüência-de-comandos-1>**

**caso <valor21>, <valor22>, ..., <valor2m>**

**<seqüência-de-comandos-2>**

...

**outrocaso**

**<seqüência-de-comandos-extra>**

**fimescolha**

# Comando de Seleção Múltipla

algoritmo " Exemplo Seleção Múltipla"

var

    time: caractere

inicio

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

    leia (time)

    escolha (time)

        caso "Sport", "Santa Cruz", "Nautico", "Petrolina"

            escreval ("É um time pernambucano.")

        caso "Vitoria da Conquista", "Bahia de Feira",  
        "Camaçari", "Feirense"

            escreval ("É um time baiano.")

        outrocaso

            escreval ("É de outro estado.")

    fimescolha

fimalgoritmo

# Estrutura de Controle de Fluxo

- ✓ Em alguns algoritmos, é necessário executar uma mesma tarefa por um número determinado ou indeterminado de vezes.
- ✓ **Exemplos:**
  - Calcular a raiz quadrada dos números 1 à 10. Observe que para cada número, o mesmo cálculo será realizado. Neste caso, o cálculo é repetido 10 vezes.
  - Calcular a raiz quadrada de um número sempre que este número for menor que 15.
- ✓ Este fato gerou a criação das **estruturas de repetição** as quais veremos a seguir.

# Estrutura de Repetição - Enquanto

- ✓ Neste caso, uma dada tarefa será repetida enquanto uma determinada condição for verdadeira.

- ✓ **Sintaxe:**

```
enquanto (<expressão lógica ou relacional>) faça  
    <sequência de comandos>
```

**Fim enquanto**

- ✓ **Obs:** <expressão lógicae relacional> é avaliada antes de cada repetição do laço. Quando seu resultado for **VERDADEIRO**, <seqüência-de-comandos> é executada.

# Estrutura de Repetição - Enquanto

**Exemplo:**

algoritmo "Exemplo 1 enquanto"

var

    r: real

inicio

    leia (r)

    enquanto ( $r < 100$ ) faca

$r \leftarrow r^{0.5}$

        escreval (r)

        leia (r)

    fimenquanto

fimalgoritmo

*E se a condição  
for  $50 < r < 100$ ?*

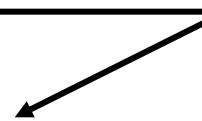
# Estruturas de Repetição – Repita...ate

- ✓ Sintaxe:

**repita**

**<seqüência de comandos>**  
**ate (<expressão lógica ou relacional>)**

**<seqüência de comandos>** será executada sempre que o resultado da **<expressão lógica ou relacional>** resultar em FALSO.



- ✓ **<seqüência de comandos>** é executada ao menos uma vez, visto que a avaliação da **<expressão lógica ou relacional>** encontrar-se no final da estrutura de repetição.

# Estruturas de Repetição – Repita...ate

## ✓ Exemplo:

algoritmo "Repita...ate"

var

    a: inteiro

inicio

    escreval("Digite um numero inteiro menor ou igual a 10")

    leia(a)

    repita

        a<-a+1

        escreval (a)

    ate (a>10)

fimalgoritmo

Sempre que a condição  $a > 10$  for FALSA, a seqüência de comandos será executada.

# Estruturas de Repetição - Para

## ✓ Sintaxe:

```
para <variável> de <valor-inicial> ate <valor limite> passo<incremento> faça  
    <sequência de comandos>  
fimpara
```

Quando o programa chega neste ponto, a variável contadora é incrementada e comparada com o valor limite.

Conta o número de repetições  
(deve ser necessariamente  
uma variável do tipo inteiro)

Especifica o valor de  
inicialização da variável  
contadora.

Especifica o valor  
máximo que a  
variável  
contadora pode  
alcançar.

Indica o valor do incremento  
que será acrescentado à  
variável contadora em cada  
repetição do laço. É opcional.

# Estruturas de Repetição - Para

## ✓ Exemplo:

algoritmo "Exemplo Para"

var

j:inteiro

inicio

para j de 0 ate 10 faca

escreval (j)

fimpara

fimalgoritmo

Se passo for omitido, o valor default do incremento é 1.

Acrescente no programa uma variável de incremento, onde o valor desta é digitada pelo usuário.

# Introdução a Liguagem C



# Material de apoio

- Download Dev C++
  - <http://www.baixaki.com.br/download/dev-c-.htm>
- Video Aulas – Curso Completo de Algoritmos e Programação em C no Dev C++.
  - [https://www.youtube.com/watch?v=joiwqS2wd1A&list=PL9PzDKD\\_B1nNpjxJ9kKF EWtN7Uzk6RpFa](https://www.youtube.com/watch?v=joiwqS2wd1A&list=PL9PzDKD_B1nNpjxJ9kKF EWtN7Uzk6RpFa)

# Linguagem C

- ✓ Criada em 1972, por Dennis Ritchie;
- ✓ Centro de Pesquisas da Bell Laboratories;
- ✓ Para utilização no S.O. UNIX;
- ✓ C é uma linguagem de propósito geral;
- ✓ Em 1989 o **Instituto Norte-Americano de Padrões (ANSI)** padronizou a linguagem C.

# Linguagem C

- ✓ Case sensitive;
- ✓ Tipos de dados primitivos: caractere, inteiro e real;
- ✓ Possui estruturas de controle de fluxo;
- ✓ Operadores aritméticos, lógicos, relacionais e condicional;
- ✓ Todo programa tem uma função principal chamada **main()**;
- ✓ Todo linha de instrução em um programa é finalizada com um “;”;

# Estrutura de um programa em C

# Palavras-reservadas

## Palavras chaves em C (padrão ANSI)

<b>auto</b>	<b>Double</b>	<b>int</b>	<b>Struct</b>
<b>break</b>	<b>Else</b>	<b>long</b>	<b>Switch</b>
<b>case</b>	<b>Enum</b>	<b>register</b>	<b>typedef</b>
<b>char</b>	<b>Extern</b>	<b>return</b>	<b>union</b>
<b>const</b>	<b>Float</b>	<b>short</b>	<b>unsigned</b>
<b>continue</b>	<b>For</b>	<b>signed</b>	<b>void</b>
<b>default</b>	<b>Goto</b>	<b>sizeof</b>	<b>volatile</b>
<b>do</b>	<b>If</b>	<b>static</b>	<b>while</b>

# Definição de Variáveis

- ✓ Devem ser declaradas no **início** do programa ou do sub bloco;
- ✓ Podem ser classificadas como **Locais** ou **Globais**.

## – Locais

- Declaradas dentro de funções;
- Utilizada apenas dentro do escopo da função;
- O escopo de uma função é determinado por abre-chaves “{“ e termina em fecha-chaves “}”;
- **Só existem** no momento que sua função está em execução.

## – Globais

- Declaradas **fora** de todas as funções;
- Podem ser **acessadas** de qualquer parte do programa;
- **Existem durante toda a execução** do programa.

# Nomes de Variáveis

- ✓ Deve conter um ou mais caracteres;
- ✓ O primeiro caractere **sempre** deve ser uma **letra**;
- ✓ Os caracteres **subseqüentes** podem ser **letras, números** ou “\_”;
- ✓ Não pode ser igual às **palavras-chaves**;
- ✓ Não pode ter o **mesmo nome de funções**;

<b>Correto</b>	<b>Incorreto</b>
Soma1	1soma
soma	soma!
area_triangulo	area...triangulo

**Obs:** as variáveis “**soma**” e “**Soma**” são **distintas**

# Declarando variáveis

## Sintaxe

- ✓ <Tipo de dados> Nome\_variável;

Ex:

```
char nome;  
int idade;  
int total;
```

## Atribuindo valor

- ✓ Nome\_da\_variavel = expressão;

Ex:

```
nome = 'Joao';  
idade = 18;  
total = 10 + 20;
```

# Operadores aritméticos

Operador Binário	Descrição
=	Atribuição
+	Soma
-	Subtração
/	Divisão
%	Modulo ( <b>resto</b> da divisão)

# Operadores aritméticos Unários e Binários

✓ **Unários** (+, -, ++, --) agem sobre uma variável apenas, modificando ou não o seu valor, e retornam o valor final da variável.

- $a = -b;$
- $a++;$  (ou seja)  $a = a + 1;$
- $a--;$  (ou seja)  $a = a - 1;$

Obs: operador “-” como troca de sinal é um operador unário que não altera a variável sobre a qual é aplicado, pois ele retorna o valor da variável multiplicado por -1.

✓ **Binários** (+, -, \*, /, %) usam duas variáveis e retornam um terceiro valor, sem modificar as variáveis originais.

# Operadores aritméticos - Hierarquia

Hierarquia	Operação
1	Parênteses
2	Função
3	<code>++</code> , <code>--</code>
4	- (menos unário)
5	<code>*</code> , <code>/</code> , <code>%</code>
6	<code>+</code> , <code>-</code>

# Operadores de Atribuição =, +=, -=, \*=, /=, %=

Instrução normal	Instrução reduzida
<code>var = var + expr;</code>	<code>Var += expr;</code>
<code>var = var - expr;</code>	<code>Var -= expr;</code>
<code>var = var / expr;</code>	<code>Var /= expr;</code>
<code>var = var * expr;</code>	<code>Var *= expr;</code>

## ✓ Exemplos:

- `a = 5;`
- `a += 5;` (*ou seja*) `a = (a + 5);`
- `a -= 5;` (*ou seja*) `a = (a - 5);`

# Comentários

- ✓ `//` Meu comentário em uma linha
- ✓ `/*` Meu comentário através de um bloco de texto que pode estar em *n* linhas `*/`

# Tipos Primitivos

## Caractere

- ✓ Definido pela palavra reservada **char**;
- ✓ Ocupa 8 bits (1 byte)
- ✓ Faixa de valores: -128 à 127
- ✓ Exemplo:
  - **char** letra;
  - letra = 'A';

# Tipos Primitivos

## Inteiro

- ✓ Definido pela palavra reservada **int**;
- ✓ Ocupa 16 bits (2 bytes)
- ✓ Faixa de valores: -32768 à 32767
- ✓ Exemplo:
  - `int num;`
  - `num = -73;`

# Tipos Primitivos

## Ponto flutuante

- ✓ Definido pela palavra reservada **float**
- ✓ Ocupa 4 bytes
- ✓ Exemplo:
  - **float** a,b,c=2.34;

## Ponto flutuante de precisão dupla

- ✓ Definido pela palavra reservada **double**
- ✓ Ocupa 8 bytes
- ✓ Exemplo:
  - **double** x=2.38, y=3.1415;

# Tipos de dados - padrão ANSI

Tipo	Tamanho aproximado em bits	Faixa mínima
char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
int	16	-32.767 a 32.767
unsigned int	16	0 a 65.535
signed int	16	O mesmo que int
short int	16	O mesmo que int
unsigned short int	16	0 a 65.535
signed short int	16	O mesmo que short int
long int	32	-2.147.483.647 a 2.147.483.647
signed long int	32	O mesmo que long int
unsigned long int	32	0 a 4.294.967.295
float	32	Seis dígitos de precisão
double	64	Dez dígitos de precisão
long double	80	Dez dígitos de precisão

# Estrutura básica de um programa em C

```
void main()    // Primeira função a ser executada
{
    /*Aqui ficaria as declarações locais e as instruções
     do programa */
}
```

# Conversão de Tipos de Dados

# Conversão de Tipos - Implícita

**Implícita** → tipos menores para tipos maiores

Exemplos:

- char → int
- int → longint
- float → double

# Conversão de Tipos - Explícita (cast)

**Explícita** → de qualquer tipo, para outros tipos maiores utilizando (cast)

Exemplos:

- char → int
- int → longint
- float → double

Exemplo:

```
int numero;  
char letra;  
numero = 555;  
letra = 'w';  
numero = (int) letra; //O tipo maior recebe o tipo menor
```

# Operadores

# Operadores Relacionais

Operador	Ação
>	maior que
$\geq$	maior ou igual a
<	menor que
$\leq$	menor ou igual
$\equiv$	igual a
$\neq$	diferente de

**IMPORTANTE!**

# Operadores Lógicos

Operador	Ação
<b>&amp;&amp;</b>	E
<b>  </b>	OU
<b>!</b>	Não

		<b>Não p</b>	<b>Não q</b>	<b>p E q</b>	<b>p OU q</b>
<b>p</b>	<b>q</b>	<b>! p</b>	<b>! q</b>	<b>p &amp;&amp; q</b>	<b>p    q</b>
falso	falso	verdadeiro	verdadeiro	falso	falso
falso	verdadeiro	verdadeiro	falso	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro	falso	verdadeiro
verdadeiro	verdadeiro	falso	falso	verdadeiro	verdadeiro

# Hierarquia dos operadores Relacionais e Lógicos

- Hierarquia ou Precedência – prioridade com que os operadores são executados pelo compilador;
- Operadores com mesmo nível hierárquico são executados da esquerda para a direita;
- Podem ser alterada utilizando “( ” “) ”.

Hierarquia	Operação
1	!
2	>, $\geq$ , <, $\leq$
3	$\equiv$ , $\neq$
4	$\&\&$
5	$\parallel$

# Operadores bit a bit

✓ Permite a manipulação de bits (baixo nível)

Operador	Descrição
&	AND
	OR
^	XOR (OR exclusivo)
<<	Deslocamento para esquerda
>>	Deslocamento para direita

Não confundir com Operadores Lógicos

# Operadores bit a bit

- ✓ O número é representado por sua forma binária;
- ✓ Poderemos fazer operações em cada um dos bits deste número;
- ✓ Programação de "baixo nível";
- ✓ Só podem ser usadas nos tipos **char**, **int** e **long int**.

## Exemplo

```
int i = 2;  
// Conteúdo em binário: 0000000000000010  
~i;  
// Ou seja, não i, resulta em binário: 1111111111111101
```

# Operadores bit a bit

## Operadores de deslocamento

### Sintaxe

- ✓ valor >> número\_de\_deslocamentos
- ✓ valor << número\_de\_deslocamentos

- ✓ Exemplo:

```
i == 0000000000000010; //2 decimal
```

```
i << 3;
```

```
i == 000000000010000; //16 decimal
```

# Funções de Entrada e Saída

# Funções de Entrada e Saída Formatada

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

io → input/output

Forma geral:

- `printf(string_de_controle, <lista_de_argumentos>);`
  
- Indica as variáveis com suas respectivas posições através dos códigos de formato “%” mostrado a seguir.

# Funções de Entrada e Saída Formatada

Código	Formato
%c	Um caractere (char)
%d	Um número inteiro decimal (int)
%i	O mesmo que %d
%e	Número em notação científica com o "e"minúsculo
%E	Número em notação científica com o "e"maiúsculo
%f	Ponto flutuante decimal
%g	Escolhe automaticamente o melhor entre %f e %e
%G	Escolhe automaticamente o melhor entre %f e %E
%o	Número octal
%s	String
%u	Decimal "unsigned" (sem sinal)
%x	Hexadecimal com letras minúsculas
%X	Hexadecimal com letras maiúsculas
%%	Imprime um %

# Funções de Entrada e Saída Formatada

Código	Imprime
printf ("Um %%%c indica %s",'c',"char");	Um %c indica char
printf ("%X %f %e",107,49.67,49.67);	6B 49.670000 4.967000e+001
printf ("%d %o",10,10);	10 12

# Constantes de barra invertida

<b>Constante</b>	<b>Significado</b>
\n	new line
\"	Aspas
\'	Apóstrofo
\0	Nulo (0 decimal)
\\"	Barra Invertida
\t	Tabulação Horizontal (TAB)

# Funções de Entrada e Saída - **scanf()**

scanf() - Leitura de dados;

**Sintaxe:**

**scanf(string\_de\_controle, lista\_de\_argumentos);**

- ✓ **string\_de\_controle** → descrição de todas as variáveis que serão lidas, com informações de seus tipos e da ordem em que serão lidas;
- ✓ **lista\_de\_argumentos** → lista com os identificadores das variáveis que serão lidas.

Importante: colocar antes de cada variável da lista\_de\_argumentos o caractere ‘&’

**Exemplo:**

**char letra;** //Declarando a variável “letra”

**scanf(“%c”, &letra);** //Lendo dados digitados pelo usuário

# Funções de Entrada e Saída

- ✓ Tabela de códigos de tipos de dados

Código	Formato
%c	Um caracter (char)
%d	Um número inteiro decimal (int)
%f	Ponto flutuante decimal
%s	String

Exemplos:

```
char letra;  
float nota;  
int quantDeFilhos;  
scanf("%c", &letra);  
scanf("%f", &nota);  
scanf("%d", &quantDeFilhos);
```

# Estruturas de Controle de Fluxo

# Estruturas de Controle de Fluxo - if

## Instrução condicional

### Sintaxe

```
if(<condição>){  
    <instrução>  
}
```

# Estruturas de Controle de Fluxo – if ... else

## Instrução condicional

### Sintaxe

```
if(<condição>)
{
    <instrução 1>
    ...
    <instrução N>
} else {
    <instrução A>
    ...
    <instrução Z>
}
```

# Estruturas de Controle de Fluxo – if ... else

## Exemplo

```
#include <stdio.h>

int main ()
{
    char letra = 'a';
    if (letra == 'a'){
        printf("Letra digitada eh: 'A'");
    }else{
        printf("Letra digitada DESCONHECIDA!");
    }
    getchar();
    return(0);
}
```

# Estruturas de Controle de Fluxo – if ... else if ... else

```
if (condição_1){  
    instrução 1;  
}else if (condição_2){  
    instrução 2;  
}else if (condição_3){  
    instrução 3;  
}else if (condição_n){  
    instrução n;  
}  
else{  
    instrução padrão;  
}
```

```
int varNumero;  
printf ("Digite um numero: ");  
scanf ("%d", &varNumero);  
  
if (varNumero > 10){  
    printf ("Numero MAIOR que 10");  
}else if (varNumero == 10){  
    printf ("Voce digitou: 10");  
}else if (varNumero < 10){  
    printf ("Numero MENOR que 10");  
}
```

//substituir o último **else if** faz sentido?

```
else {  
    printf ("Numero MENOR que 10");  
}
```

# Estruturas de Controle de Fluxo – if ... else if ... else

Os códigos funcionam? Existe diferença?

```
int varNumero;  
printf ("Digite um numero: ");  
scanf ("%d", &varNumero);  
  
if (varNumero > 10){  
    printf ("Numero MAIOR que 10");  
}else if (varNumero == 10){  
    printf ("Voce digitou: 10");  
}else if (varNumero < 10){  
    printf ("Numero MENOR que 10");  
}
```

```
int varNumero;  
printf ("Digite um numero: ");  
scanf ("%d", &varNumero);  
  
if (varNumero > 10){  
    printf ("Numero MAIOR que 10");  
}  
if (varNumero == 10){  
    printf ("Voce digitou: 10");  
}  
if (varNumero < 10){  
    printf ("Numero MENOR que 10");  
}
```

# Estruturas de Controle de Fluxo – ifs aninhados

```
float nota;  
nota = 9.5;  
if (nota >= 7){  
    printf("Aprovado.");  
    if (nota >= 9){  
        printf("Aprovado com  
louvor!!!");  
    }  
}else{  
    printf("Aluno REPROVADO!");  
}
```

# Estruturas de Controle de Fluxo – operador ?

**if (condição){**

*instrução 1;*

**}else{**

*instrução 2;*

**}**

✓ É equivalente a:

**condição?instrução1:instrução2;**

# Estruturas de Controle de Fluxo – switch

- ✓ Próprio para se **testar** uma **variável** em relação a **valores pré-estabelecidos**.
- ✓ Testa o conteúdo da variável e **executa a instrução** correspondente ao case;
- ✓ **break**, faz com que o switch seja **interrompido**;
- ✓ **default** é opcional;
- ✓ Não aceita expressões.

```
switch (variável)
{
    case constante_1:
        instrução 1;
        break;
    case constante_2:
        instrução 2;
        break;
    ...
    default
        instrução_padrão;
}
```

# Estruturas de Controle de Fluxo – **switch**

```
switch (varNumero)
{
    case 9:
        printf ("O numero e igual a 9.");
        break;
    case 10:
        printf ("O numero e igual a 10.");
        break;
    default:
        printf ("O numero nao e nem 9 nem 10.");
}
```

# Loops de Repetição

# Loops de Repetição - **while**

**Estrutura de repetição**

**Sintaxe**

**while(<condição>)**

{

*<instrução 1>*

...

*<instrução n>*

}

# Loops de Repetição - **while**

## Exemplo

```
#include <stdio.h>

int main ()
{
    int numero;
    printf("Digite um numero: ");
    printf("\nDigite '0' para finalizar.\n\n");
    while (numero != 0){
        scanf("%d", &numero);
        printf("\n Voce digitou: %d \n Digite um novo numero: ",
            numero);
    }
    getchar();
    return(0);
}
```

# Loops de Repetição – do ... while...

```
do{  
    <instrução 1>  
    ....  
    <instrução n>  
}while(<condição>);
```

```
int i;  
do {  
    printf ("Escolha a fruta pelo  
    numero:");  
    printf("(1) Mamao");  
    printf("(2) Abacaxi");  
    printf("(3) Laranja");  
    scanf("%d", &i);  
} while ((i<1)|| (i>3));
```

# Loops de Repetição – **for**

## Sintaxe

```
for (inicialização; condição; incremento) {  
    instrução;  
}
```

- ✓ Podemos omitir qualquer um dos elementos do for: (**inicialização**; **condição**; **incremento**).

```
// int numero;  
for (int numero=1; numero<=100; numero++) {  
    printf ("%d ", numero);  
}
```

# Comando - **break**

- ✓ Faz com que a execução do programa continue na **primeira linha seguinte** ao **loop** ou **bloco** que está sendo interrompido.
- ✓ Utilizados para interromper os comandos: “**switch**”, “**for**”, “**while**” e “**do while**”.

## Exemplo

```
for(;;) {  
    printf("%d", count);  
    count++;  
    if(count==10) break;  
}
```

# Comando - **continue**

- ✓ Funciona **apenas** dentro de um loop;
- ✓ Quando o comando **continue** é encontrado, o loop pula para a próxima iteração, sem o abandono do loop;

# Comando - goto

- ✓ realiza um salto para um local determinado por um rótulo.
- ✓ tende a tornar o código confuso

## Sintaxe

**nome\_do\_rótulo:**

....

**goto nome\_do\_rótulo;**

## Exemplo

**íncio\_do\_loop:**

```
if (condição)
{
    intrução;
    incremento;
    goto íncio_do_loop;
}
```

# Estruturas de Controle de Fluxo – switch

- ✓ Testa o conteúdo de uma **variável** em relação a **valores pré-estabelecidos**.
- ✓ Executa a **instrução** correspondente ao case;
- ✓ **Break** - faz com que o switch seja **interrompido**;
- ✓ **default** é opcional;
- ✓ Não aceita expressões.

```
switch (variável)
{
    case constante_1:
        instrução 1;
        break;
    case constante_2:
        instrução 2;
        break;
    ...
    default
        instrução_padrão;
}
```

# Estruturas de Controle de Fluxo – Switch

```
switch (varNumero)
{
    case 9:
        printf ("O numero e igual a 9.");
        break;
    case 10:
        printf ("O numero e igual a 10.");
        break;
    default:
        printf ("O numero nao e nem 9 nem 10.");
}
```

# Loops de Repetição

# Loops de Repetição - **while**

Estrutura de repetição

**Sintaxe**

```
while(<condição>)
{
    <instrução 1>
    ...
    <instrução n>
}
```

# Loops de Repetição - **while**

## Exemplo

```
#include <stdio.h>
int main ()
{
    int numero;
    printf("Digite um numero: ");
    printf("\nDigite '0' para finalizar.\n\n");
    while (numero != 0 ){
        scanf("%d", &numero);
        printf("\n Voce digitou: %d \n Digite um novo numero: ", numero);
    }
    getchar();
    return(0);
}
```

# Loops de Repetição – do ... while...

```
do{  
    <instrução 1>  
    ....  
    <instrução n>  
}while(<condição>);
```

```
int i;  
do {  
    printf ("Escolha a fruta pelo  
    numero:");  
    printf("(1) Mamao");  
    printf("(2) Abacaxi");  
    printf("(3) Laranja");  
    scanf("%d", &i);  
} while ((i<1)|| (i>3));
```

# Loops de Repetição – **for**

## Sintaxe

```
for (inicialização; condição; incremento) {  
    instrução;  
}
```

- ✓ Podemos omitir qualquer um dos elementos do for: **(inicialização; condição; incremento)**.

```
// int numero;  
for (int numero=1; numero<=100; numero++) {  
    printf ("%d ", numero);  
}
```

# Comando - **break**

- ✓ Faz com que a execução do programa continue na **primeira linha seguinte ao loop ou bloco** que está sendo interrompido.
- ✓ Utilizados para interromper os comandos: “**switch**”, “**for**”, “**while**” e “**do while**”.

## Exemplo

```
for(;;) {  
    printf("%d", count);  
    count++;  
    if(count==10) break;  
}
```

# Comando - **continue**

- ✓ Funciona **apenas** dentro de um loop;
- ✓ Quando o comando **continue** é encontrado, o loop pula para a próxima iteração, sem o abandono do loop;

```
while (x>100)
```

```
{
```

```
    x-=b*3;
```

```
    if (x<y)
```

```
        continue; ——————
```

```
        x-=y*3;
```

```
}
```

# Comando - goto

- ✓ realiza um salto para um local determinado por um rótulo.
- ✓ tende a tornar o código confuso

## Sintaxe

**nome\_do\_rótulo:**

....

**goto nome\_do\_rótulo;**

## Exemplo

**início\_do\_loop:**

```
if (condição)
{
    intrução;
    incremento;
    goto início_do_loop;
}
```

## String

- ✓ Em C **String** é um **vetor** de **caractere** termina com um caracter **nulo** ('\0');

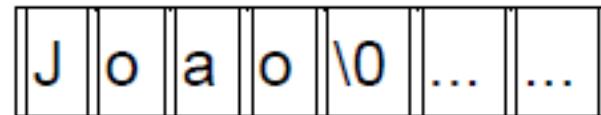
### Sintaxe

*char nome\_da\_string[tamanho];*

*Se declararmos uma string com tamanho = 7;*

*Armazenarmos nessa string o nome “joao”;*

**As duas células não usadas têm valores indeterminados.**



# String

- ✓ A função **gets()**
  - Utilizada para ler strings;
- ✓ Insere o terminador nulo na string, quando o usuário aperta a tecla "Enter".

## Sintaxe

```
gets (nomeDaVariavel);
```

# String

- ✓ A função **scanf()**;
  - Também pode ser utilizada para ler strings;

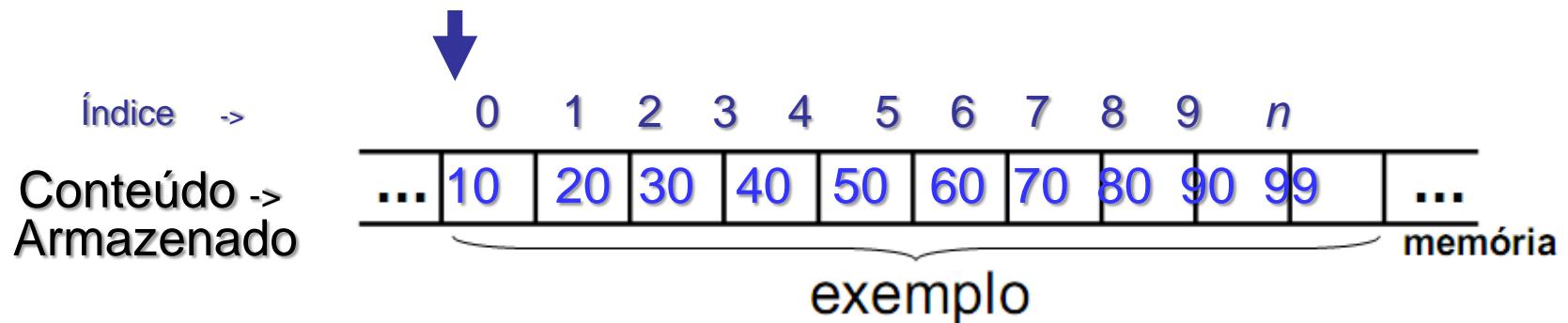
## Sintaxe

```
scanf("%s", &minha_string_de_Nomes);
```

# Vetores

## Vetores

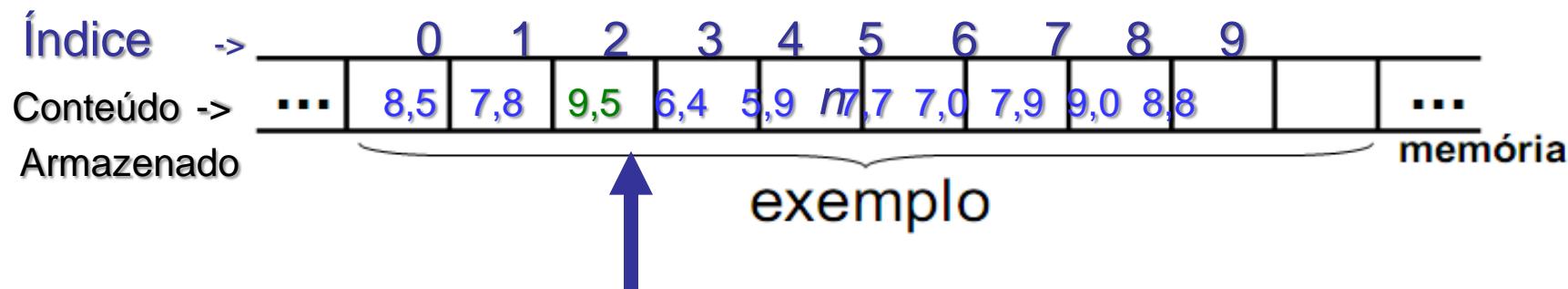
- ✓ Todos os elementos pertencentes ao mesmo tipo de dado;
- ✓ Índices (iniciam em “0”, até “n”);
- ✓ Índices utilizados para Recuperar/Inserir valores.



# Vetores

✓ Exemplo

```
float vetor_de_notas [10];
```



```
vetor_de_notas[2] = 9,5;
```

```
scanf("%f", &vetor_de_notas[x]); //Leitura do teclado  
printf("A nota eh: %f", vetor_de_notas[x]); //Escreve na Tela
```

# Vetores

- ✓ Inserindo valores na declaração de um Vetor:

```
int vetor[10]={0,1,2,3,4,5,6,7,8,9};
```

# Strings e Vetores

# String - REVISÃO

- ✓ Em C **String** é um **vetor** de **caractere** termina com um caracter **nulo** ('\0');

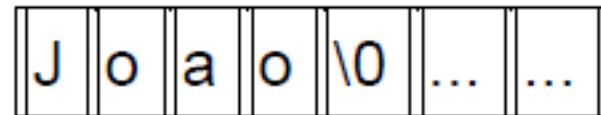
## Sintaxe

`char nome_da_string[tamanho];`

Se declararmos uma string com tamanho = 7;

Armazenarmos nessa string o nome “joao”;

As duas **células não usadas** têm **valores indeterminados**.



# String - REVISÃO

✓ Funções de Strings (#include <string.h>)

- **gets (string);**
  - Lê a entrada do teclado.
- **strcmp(string1, string2);**
  - Compara as duas strings retornando 0 se igual, ou 1 se diferente.
- **strcpy (string1, string2);**
  - Copia o conteúdo de uma string em outra
- **strcat (string1, string2);**
  - concatenar duas strings
- **strlen (string);**
  - retorna o tamanho (quantidade de letras) de uma string
  - despreza o caractere nulo final (\0).
- **puts (string);**
  - Imprime uma string na tela.

# String - REVISÃO

- ✓ A função **gets()**;
  - Utilizada para ler strings;
- ✓ Insere o terminador nulo na string, quando o usuário aperta a tecla "Enter".

## Sintaxe

- gets (nomeDaString);

- A função **scanf()**;

- Também pode ser utilizada para ler strings;

## Sintaxe

- **scanf("%s", & nomeDaString);**

# Vetores de Strings

- ✓ Matriz bidimensional de char's

## Sintaxe:

```
char nomeDoVetor[qt_de_strings][tamanho_das_strings];
```

Acessar um vetor de Strings:

```
nomeDoVetor[índice];
```

# FUNÇÕES

# Funções

## O que é? Para que serve?

- ✓ **Funções são estruturas que permitem ao usuário separar seus programas em blocos reutilizáveis.**
- ✓ **Facilita**
  - **Manutenção;**
  - **Leitura / entendimento;**
  - **Reuso de código, etc.**

# Funções

- ✓ A função **main()** é uma função que retorna um inteiro.
- ✓ Esse retorno é muito utilizado para detectar se a função **main()** **terminou** seu processamento **normalmente (return zero)**.
- ✓ Ou, se ocorreu **algo anormal** durante sua execução (**return diferente de zero**).

# Funções

**Para trabalhar com funções é necessário:**

- **Declarar a função** que deseja criar.
  - Antes do Método main ou **dentro** do método main;
  - **Ex:** `tipo_de_retorno nomeDaFuncao (tipo parametro);`
- **Implementar a função.**
  - Antes do Método main ou **depois** do método main;
  - *Ex: próximo slide.*
- **Executar/Chamar a função.**
  - **Dentro** do método main;
  - ou
  - **Dentro** de outra Função.
  - **Ex:** `nomeDaFuncao (valor);`

# Funções - Implementar a função.

- ✓ Ao implementar uma função, podemos declarar variáveis;
- ✓ Estas variáveis **apenas** podem ser utilizadas no **escopo local**, ou seja, apenas dentro desta função;
- ✓ Existe um outro tipo de variável chamada de **Global**, falaremos sobre ela na **próxima aula**.

# Funções - Implementar a função.

## Sintaxe:

**Tipo\_de\_retorno Nome\_da\_função (Declaração\_de\_parâmetros)**

{

**<Corpo\_da\_função>**

**return [valor\_do\_retorno];**

}

**Tipo\_de\_retorno:** é o tipo da variável que a função vai retornar. O padrão é: int;

**Nome\_da\_função:** utilizado para identificar e executar a função;

**Declaração\_de\_parâmetros:** informa ao compilador quais serão as variáveis de entrada da função através da seguinte forma geral:

**(tipo parametro1, tipo parametro2, ... )**

**Corpo\_da\_função:** onde ocorre o processamento dos dados de entrada da função.

**return:** o valor de retorno deve ser compatível com o tipo de retorno declarado para a função.

# Funções - Implementar a função.

## Sintaxe:

**Tipo\_de\_retorno Nome\_da\_função (Declaração\_de\_parâmetros)**

{

**<Corpo\_da\_função>**

**return [valor\_do\_retorno];**

}

## Exemplo:

```
float soma(float num1, float num2){  
    float resultado; //Variável Local  
    resultado = (num1 + num2);  
    return resultado;  
}
```

# Exemplo de Função

```
#include <stdio.h>
float soma(float num1, float num2); //Declaração da Função "soma"

int main (){
    float A, B;
    printf("Digite um numero: ");
    scanf("%f", &A);
    printf("Digite um OUTRO: ");
    scanf("%f", &B);
    printf("RESULTADO: %.2f", soma(A, B)); //Chamada da função
    getchar(); getchar();
    return(0);
}

float soma(float num1, float num2){
    float resultado; //Variável Local
    resultado = (num1 + num2);
    return resultado;
}
```

# Funções - **VOID**

- ✓ O tipo “void” quer dizer vazio;
- ✓ Permite fazer funções que não retornam um valor;
- ✓ O comando **return** não é necessário;
- ✓ podemos utilizar **return** para finalizar o processamento de uma função em pontos estratégicos.

**Sintaxe:**

**void nome\_da\_função (declaração\_de\_parâmetros);**

# Exemplo de Função

```
#include <stdio.h>

void minhaMensagem(int numeroDigitado); //Declaração da Função

int main (){
    int A;
    printf("Digite um numero: ");
    scanf("%d", &A);

    minhaMensagem(A); //Chamada da função “minhaMensagem”

    getchar(); getchar();
    return(0);
}
```

```
void minhaMensagem(int X){

    printf("Oi, voce digitou: %d", X);
}
```

## Variáveis

## Locais x Globais

# Variáveis Locais e Globais

- ✓ Até este aula, utilizamos apenas **variáveis locais**, ou seja, aquelas declaradas dentro da função, sempre no início.

```
float soma(float num1, float num2){  
    float resultado; //Variável LOCAL  
    resultado = (num1 + num2 + numx);  
    return resultado;  
}
```

- ✓ **Variáveis Globais:**

- estão **fora de qualquer função**, usualmente no início do programa.
- são acessível em todos os escopos;
- Mais utilizada em *programação concorrente*.

*Exemplo no próximo slide.*

# Variáveis Locais e Globais

```
#include <stdio.h>
float num1Global; //Variável GLOBAL
float num2Global; //Variável GLOBAL
float resultado; //Variável GLOBAL (não recomendado, uso exclusivo de soma())
float soma(float num1, float num2);

int main () { //Não foram declaradas variáveis nesta função
    printf("Digite um numero: ");
    scanf("%f", &num1Global);
    printf("Digite um OUTRO: ");
    scanf("%f", &num2Global);
    printf("RESULTADO: %.2f", soma(num1Global, num2Global));
    getchar(); getchar(); return(0);
}

float soma(float num1, float num2){ //Não foram declaradas variáveis nesta função
    resultado = (num1Global + num2Global);
    return resultado;
}
```

# Variáveis Locais e Globais

```
#include <stdio.h>

float num1Global; //Variável GLOBAL (não recomendado neste caso)
float num2Global; //Variável GLOBAL (não recomendado neste caso)
float resultado; //Variável GLOBAL (não recomendado, uso exclusivo de soma())
float soma(float num1, float num2);
void qualquer(); //E agora?

int main () {
    printf("Digite um numero: ");
    scanf("%f", &num1Global);
    printf("Digite um OUTRO: ");
    scanf("%f", &num2Global);
    qualquer(); //E agora?
    printf("RESULTADO: %.2f", soma(num1Global, num2Global));
    getchar(); getchar(); return(0);
}

float soma(float num1, float num2){
    resultado = (num1Global + num2Global);
    return resultado; }

void qualquer(){//E agora?
    printf("Voce executou uma funcao qualquer!\n");
    num2Global = 100;
}
```

# Variáveis Locais e Globais

## ✓ Boas Práticas

- Não se deve declarar uma variável como **GLOBAL** se ela é de uso **exclusivo** de um **bloco de código específico**. Neste caso, a declaramos dentro da função que a utiliza (**LOCAL**);
- Declarar variáveis Globais apenas quando necessário e para uso **em mais de uma função**;
- Variáveis Globais podem **confundir a leitura do código** se não bem utilizada;
- Variáveis Globais podem ser um ponto de falha do sistema, pois, **qualquer função** pode **alterar seu valor**;
- **É preciso saber a real necessidade de usá-las!**

# Exercícios de Fixação - 01

- 1) Escreva um programa que receba dois números e calcule a média deles.
- 2) Escreva um programa que receba três números e depois peça o valor dos pesos e depois calcule a média ponderada dos números.
- 3) Faça um programa que leia uma temperatura em graus Celsius e calcule o correspondente em Fahrenheit. Sabendo que:  
$$F = (180 * (C + 32)) / 100$$

# Exercício de Fixação - 02

- 1) Escreva um programa que requisita dois números e faz a soma deles e depois pergunta se o usuário quer fazer o cálculo novamente.
- 2) Escreva um programa que recebe um número e conta a partir deste número até 100.
- 3) Faça um programa que requisita dois números, os compara e depois mostra qual deles é o maior.

# Exercícios de Fixação - 03

- 1) Escreva um problema que receba as três notas do aluno e mostra a média ponderada do mesmo, sabendo que os pesos de cada unidade são 4, 5 e 6. Faça o programa de modo que seja possível calcular a nota de no mínimo três alunos. Como você faria se fossem 100 alunos? Para resolver tal problema utilize os conceitos de laços de repetição.
- 2) Escreva um programa que realize arredondamentos de números utilizando a regra geral da matemática: se a parte fracionária for maior do que ou igual a 0,5, o número é arredondado para o inteiro imediatamente superior; caso contrário, é arredondado para o inteiro imediatamente anterior.
- 3) Escreva um programa para determinar o maior entre três números dados.

## Exercício de Fixação - 04

- ✓ Desenvolva uma rotina em **C** que seja capaz de:
- ✓ Ao cadastrar um colaborador, o sistema deverá verificar se sua idade é igual ou maior de 18 anos. Caso seja maior, seu bônus salarial será de R\$ 200,00. Caso seja menor, adicionar bônus de R\$ 80,00.
- ✓ Se o colaborador for casado e tiver filhos, receberá de salário família R\$ 50,00 por cada filho.
- ✓ Imprima o salário total do colaborador.
- ✓ O cadastro só será finalizado se o nome do usuário digitado for igual a ‘fim’.

## Exercício de Fixação - 05

- ✓ Faça uma rotina em C que permita armazenar 10 números inteiros em um vetor.
- ✓ Após armazenados os 10 números, imprima na tela cada número e seu respectivo índice.

# Exercício de Fixação - 06

- ✓ Utilizando estruturas de repetição, crie um vetor capaz de armazenar 3 nomes. Após armazená-los, imprima os 3 nomes digitados.

# Exercício de Fixação - 07

- ✓ Utilizando estruturas de repetição e vetor, crie uma rotina capaz de armazenar 5 nomes de alunos e suas respectivas notas. Após armazenados, imprima todos os nomes e as respectivas notas.

# Exercício de Fixação - 08

- ✓ Faça uma rotina em C que permita armazenar 10 números inteiros em um vetor.
- ✓ Após armazenados os 10 números, imprima na tela cada número e seu respectivo índice.

# Exercício de Fixação - 09

- ✓ Faça uma rotina em C que permita armazenar 5 **NOMES** em um vetor.
- ✓ Após armazenados os 5 nomes, imprima na tela cada **nome e seu respectivo índice**.

## Exercício de Fixação - 10

- ✓ Desenvolva uma calculadora em linguagem C capaz de efetuar as operações de soma, subtração, multiplicação e divisão de dois números reais. Os números e a operação escolhida devem ser digitados pelo usuário. Após cada resultado, o programa deverá permitir que o usuário realize uma nova operação ou feche o programa.
- ✓ Obs.: na operação divisão, o divisor deverá ser diferente de zero.
- ✓ Coloque cada uma das operações dentro de uma função específica.
- ✓ Exemplo:
  - `soma()`
  - `subtracao()`
  - `divisao()`
  - `multiplicacao()`

# Exercício de Fixação - 11

- ✓ Adicione a calculadora desenvolvida na aula anterior, uma variável capaz de armazenar quantas vezes as operações foram executadas.
- ✓ Lembre-se que cada operação (+, - e \*) deve estar em uma função específica.
- ✓ Após exibir cada resultado, verifique se o usuário deseja saber quantas operações realizou. Caso afirmativo, exiba o resultado e finalize o programa.

# Projeto Prático a ser Desenvolvido nas Aulas

- Projeto prático em C para um sistema de Cadastro de uma loja de roupas que:**
- Realize o Cadastro de produtos, clientes e vendas
- Exiba os dados cadastrados
- Altere os dados cadastrados
- Exclua os dados cadastrados
- Forneça informações do sistema
- Permita que o usuário faça Login e Logon