



Programação de Computadores I

UNIDADE 1 – Conceitos introdutórios

O QUE VAMOS ESTUDAR:

- **Abstração e Lógica.**
- **Lógica estruturada: padrões.**
- **Algoritmos: conceitos e representações.**
- **Conceitos: Pseudocódigos, Linguagens de Programação e Codificação.**

➤ **Abstração e Lógica**

A lógica é um termo comumente utilizado no cotidiano nas formas mais simples sem denotar conceitos (Generalização – Encadeamento – Inferência). Utilizamos de lógica cotidiana em sentenças:

- **Generalização** (Todo – Partes)

Todo mamífero é um animal.

Todo cavalo é um mamífero.

Portanto, todo cavalo é um animal.

- **Encadeamento**

A gaveta está fechada.

A caneta está dentro da gaveta.

Precisamos primeiro abrir a gaveta para depois pegar a caneta.

- **Inferência**

Anacleto é mais velho que Felisberto.

Felisberto é mais velho que Marivaldo.

Portanto, Anacleto é mais velho que Marivaldo.

Prof. Alessandro Bertolani Oliveira

Entretanto, a lógica cotidiana não representa com fidelidade esta ciência que é linha de pesquisa da filosofia e pode ser estendida a outras disciplinas.

E ainda, percebemos que a ilógica leva a desorganização (caos) e a falta de padronização.

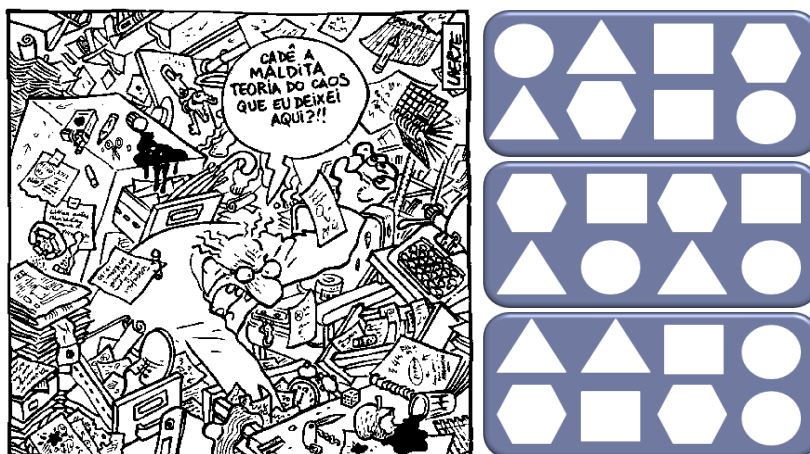


Figura 1 – O pensamento “sem lógica” leva a desordem e ao caos.

LÓGICA: É a arte de “**bem pensar**”. Pensar neste contexto significa abstrair (Imaginar – Generalizar – Modelar) dados e fatos encontrados no mundo real. Com isso, a lógica é uma ciência que objetiva colocar “**ordem nos pensamentos**” – dados e fatos – generalizados (abstraídos) do mundo real.

Outro fato é que na arte de “**bem pensar**”, a forma mais complexa do pensamento é o **raciocínio**. O raciocínio livre é devaneio que, não educado e treinado, facilmente se “perde”. Portanto, a lógica também visa “**corrigir o raciocínio**”, realinhando idéias e estabelecendo padrões.

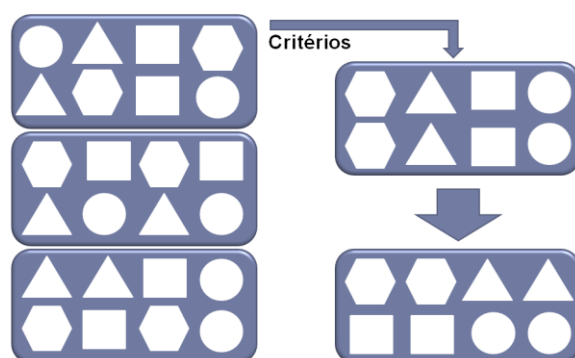
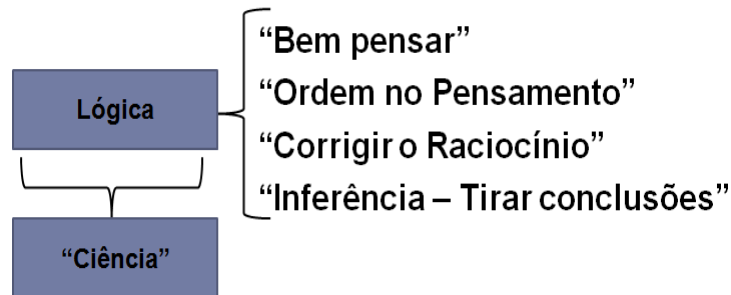


Figura 2 – A lógica estabelece critérios corrigindo e realinhando o raciocínio.

Portanto, a lógica visa:



Então, o que é **LÓGICA DE PROGRAMAÇÃO**?

O raciocínio é algo abstrato, intangível. Os seres humanos têm a capacidade de expressá-lo através da palavra falada ou escrita (idioma). Um mesmo raciocínio (“pensamento”) pode ser expresso em qualquer um dos inúmeros idiomas existentes, mas continuará representando o mesmo raciocínio, usando apenas outra convenção.

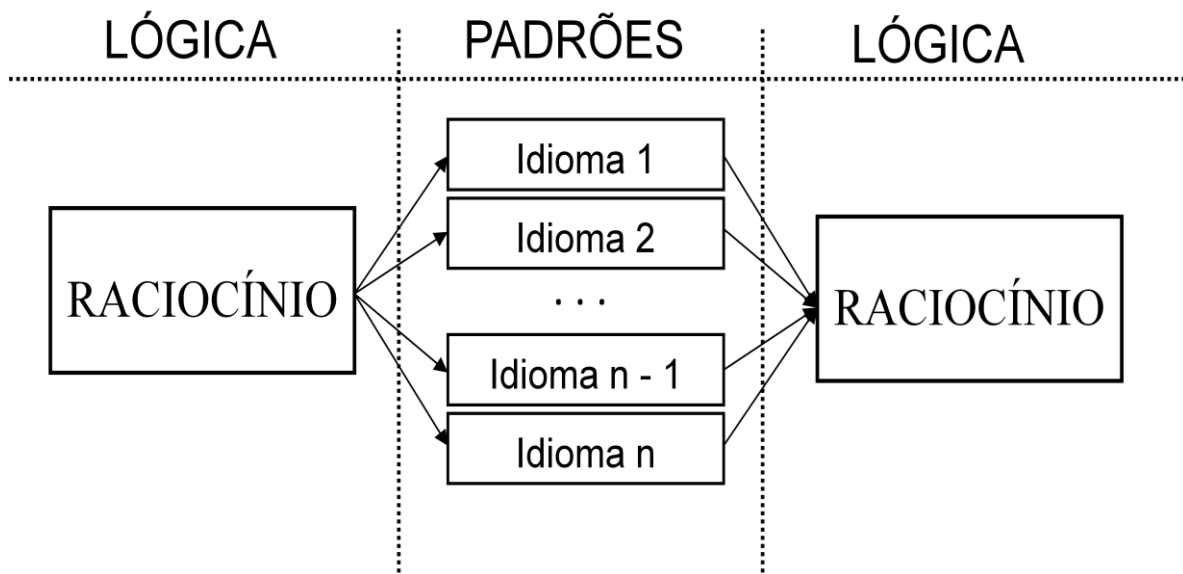


Figura 3 – O raciocínio (“Pensamento”) não depende de padrões.

À semelhança da lógica humana, a **Lógica de Programação** visa fornecer uma série de técnicas que possam “ordenar o pensamento” e “corrigir o raciocínio” na realização de uma atividade (objetivo). Para isso, a idéia ou pretensão inicial seria fugir dos padrões (Detalhe: **Torre de Babel**) e representar fielmente o raciocínio (“Pensamento”).

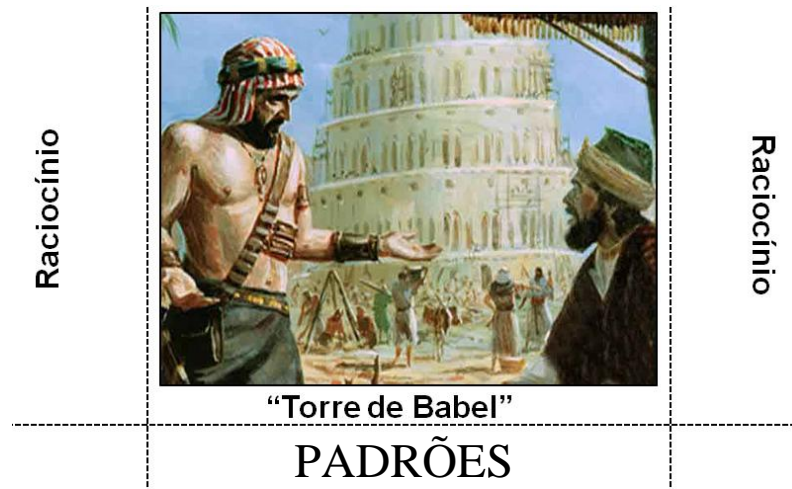


Figura 4 – O mesmo pensamento – objetivo (construir a torre) e muitos idiomas.

Entretanto; na lógica de programação esbarramos novamente na limitação em relação à **“Forma de Expressão”** do raciocínio (“Pensamento”). Com isso, no intuito de fugir dos padrões (“Torre de Babel”) e representar mais fielmente o raciocínio foram idealizados os **Algoritmos**.

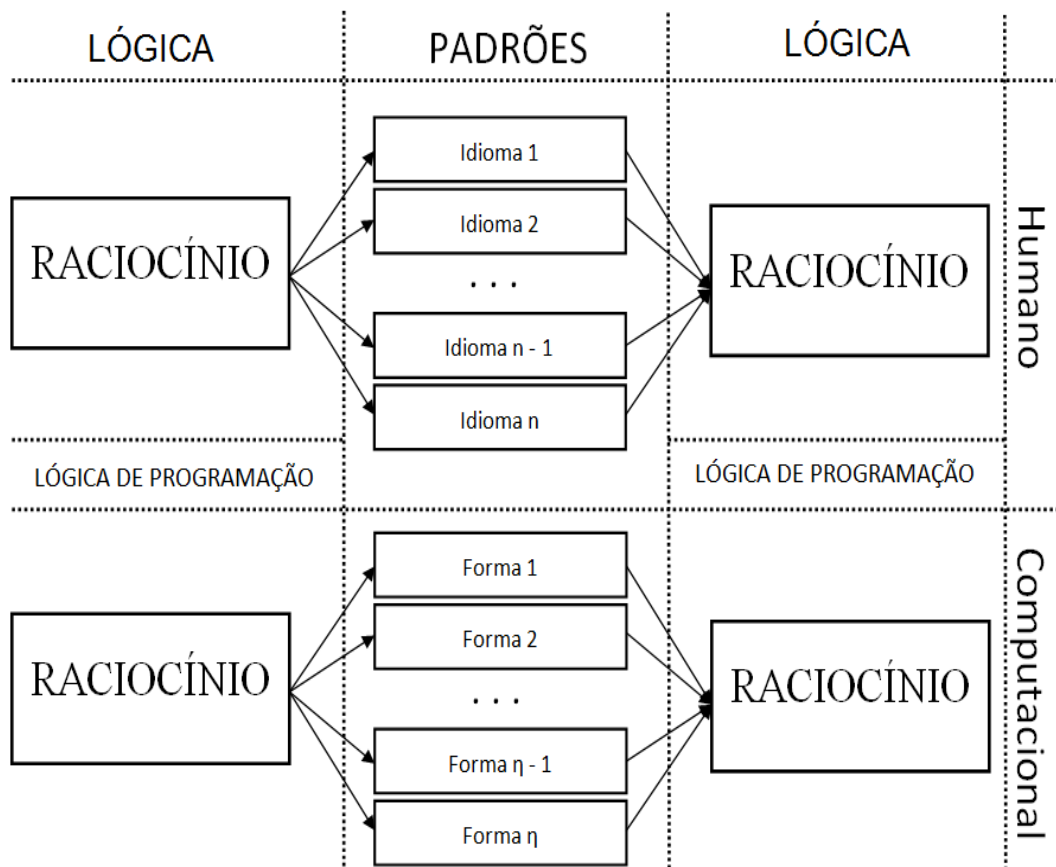
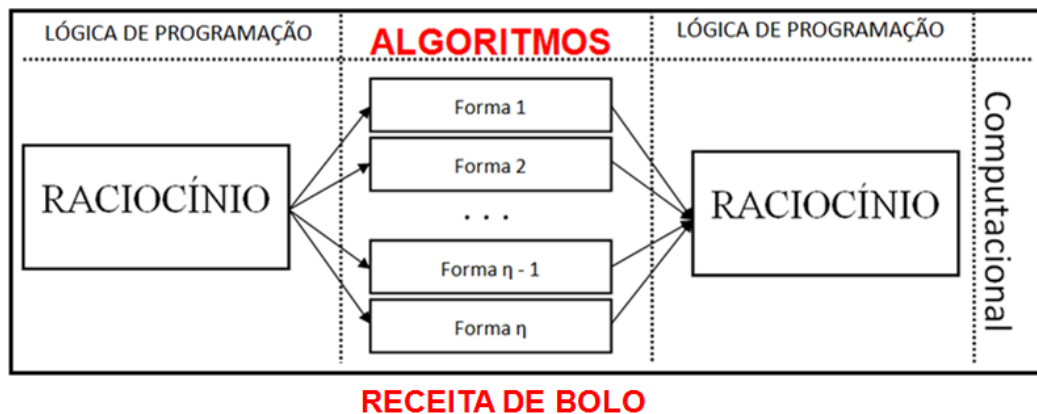


Figura 5 – A lógica humana e computacional com seus muitos padrões de representação.

ALGORITMOS: É uma **seqüência de passos** que visa atingir um **objetivo bem definido**. Para definir uma seqüência de passos, precisamos utilizar ordem, ou seja, “pensar em ordem”, com isso utilizamos “lógica” (de programação).

Exemplo de algoritmo: RECEITA DE BOLO.



Seqüência de passos: RECEITA

Objetivo: BOLO

Figura 6 – Exemplo de algoritmo e muitas formas de expressão.

A combinação **Raciocínio – Padrão (Formas de Expressão) – Raciocínio** é de complexa dissociação. Essa dissociação somada à idéia de fugir dos padrões realmente nos poupa de uma série de detalhes computacionais pertencentes à lógica de programação que poderíamos incluir posteriormente.

Contudo, assim como a forma de expressão do raciocínio humano passa pela forma falada ou escrita, a forma de expressão da lógica de programação, através dos algoritmos (“Raciocínio computacional”), passa por formas de expressão **gráfica** ou **textual**.

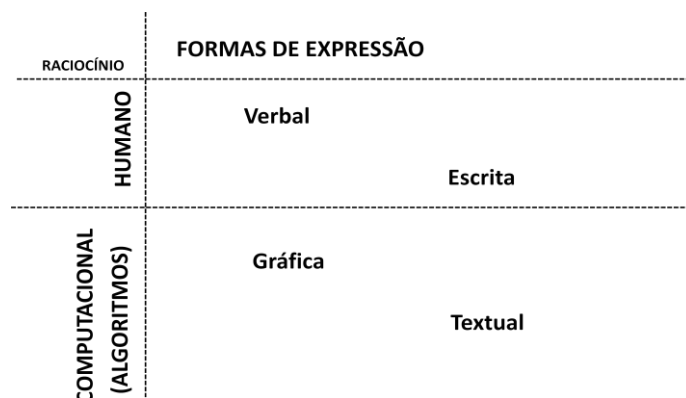


Figura 7 – O raciocínio humano e o “raciocínio computacional” através da lógica.

Nos algoritmos, as **formas de expressão do raciocínio** podem ser classificadas como:

- **Gráfica:** Fluxograma de dados e outros
- **Textual:** Linguagens “naturais” (Pseudocódigos) ou de Programação.

Os algoritmos expressos nas formas gráficas são de intuitiva percepção, pois representam mais fielmente o raciocínio original. Entretanto, são de difícil elaboração (devido à multiplicidade de símbolos e formas) e alteração (mobilidade gráfica dos elementos).

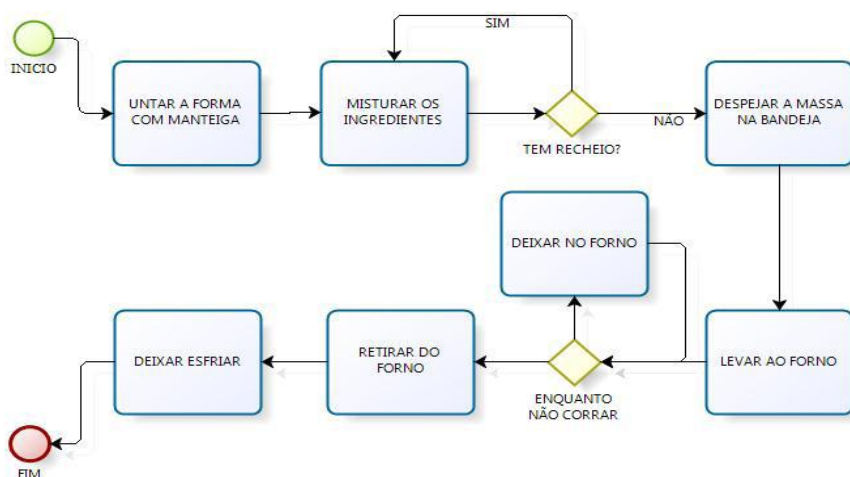


Figura 8 – **Algoritmo Gráfico:** Receita de bolo através de fluxograma de dados.

Por sua vez, a representação textual “estrutural” (a um idioma) pode ser de ambígua interpretação e de confusa padronização, podendo distorcer a linha de raciocínio originalmente proposta.

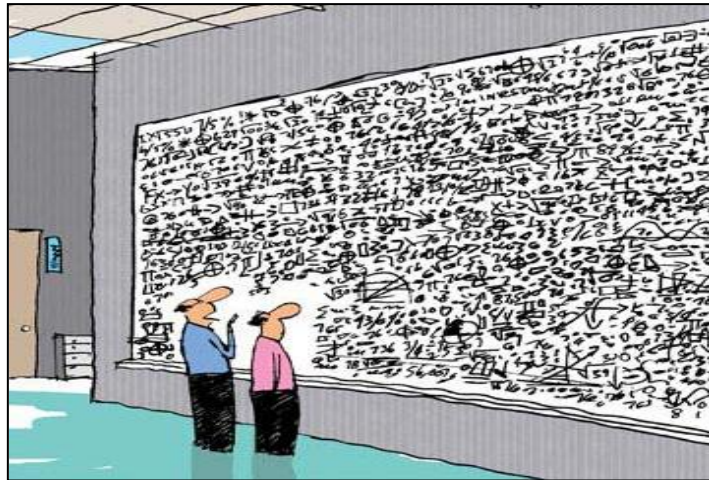


Figura 9 – Distorção da linha de raciocínio devido à falta de padrão.

Para evitarmos essa confusão, adotamos (à priori) uma **falsa codificação (Pseudocódigo)** que fixa e estrutura o padrão (textual) de elaboração do algoritmo e ao mesmo tempo o afasta de detalhes computacionais, objetivando assim o raciocínio, a lógica e não; a forma.

```

1  /* PSEUDOCÓDIGO: PORTUGUÊS ESTRUTURADO */
2  // ALGORITMO: RECEITA DE BOLO
3  Algoritmo ReceitaBolo
4  inicio
5      AÇÃO 1: UNTAR = "A forma com Manteiga";
6      AÇÃO 2: MISTURAR = "Os ingredientes";
7      DECISÃO 1: enquanto (Tem Recheio ?) faça
8          inicio
9              AÇÃO 2: MISTURAR = "Os ingredientes";
10         fim
11     AÇÃO 3: DESPEJAR = "A massa na bandeija";
12     AÇÃO 4: LEVAR = "Ao forno";
13     DECISÃO 2: enquanto (Não corar ?) faça
14         inicio
15             AÇÃO 5: Deixar = "No forno";
16         fim
17     AÇÃO 6: RETIRAR = "Bandeija do forno";
18     AÇÃO 7: DEIXAR = "Esfriar o bolo";
19 fim
  
```

Figura 10 – **Algoritmo Textual:** Receita de bolo através do Português Estruturado.

Por outro lado, em contrapartida, precisamos de uma linguagem estruturada de programação que possa desenvolver o algoritmo (e seu raciocínio) de uma forma computacionalmente válida; em verdadeiros códigos de **Linguagens de Programação**, por exemplo, **C**, **MatLab** ou **Pascal**. Esse processo de conversão de falsos códigos, para uma linguagem de programação verdadeiramente estrutura, dá-se o nome de **Codificação**.



UNIDADE 2 – Tópicos preliminares

O QUE VAMOS ESTUDAR:

- **Pseudocódigo: Português estruturado e padrão.**
- **Variável: Identificadores, Tipos, declaração e comando de atribuição.**
- **Expressões: Aritméticas, Relacionais e Lógicas.**
- **Comando de Entrada e Saída.**
- **Bloco de Instruções.**

Na unidade anterior, vimos que para **focar no raciocínio e não na “forma (Padrão) de expressão”**, devemos utilizar uma solução intermediária para desenvolvimento de algoritmos que estrutura a lógica de programação através de pseudocódigo. Com isso, vamos utilizar um padrão estabelecido nesta apostila que é o “**Português Estruturado**”.

Para isso, vamos estabelecer um **padrão** que, direciona para a **correção do raciocínio**, para o **treinamento de habilidades lógicas** e para o **desenvolvimento dos algoritmos**, sem a obrigatoriedade (a princípio) de serem computacionalmente válidos. Contudo, ao mesmo tempo, treina (ordena – Corrige) a mente (“Pensamento” – Raciocínio) para uma futura etapa de codificação.

- **Pseudocódigo: Português estruturado e padrão**

O padrão estruturado segue na apostila da seguinte forma de expressão:

	PSEUDOCÓDIGO: Português estruturado
	COMANDO:
PADRÃO	



➤ **Variável: Identificadores, Tipos, declaração e comando de atribuição**

Um dado é classificado como **Variável** quando tem a possibilidade de ser alterado em algum instante no decorrer do tempo.

Exemplos: **Cotação** do dólar, **Massa** de uma pessoa, **Índice** de inflação.

As variáveis são representadas pelos seus Identificadores. O **Identificador** é o nome que recebe o dado no algoritmo e o acompanha ao longo de toda execução. Estes Identificadores são formados pelas seguintes **regras**:

- Devem começar por um caractere alfabético:
 - **Exemplo:** Nome
 - **Contra-exemplo:** 1ºNome
- Podem ser seguidos por mais caracteres alfabéticos ou numéricos:
 - **Exemplo:** NUMEROCELULAR
- Os identificadores maiúsculos e minúsculos são distintos:
 - **Exemplo:** Altura \neq altura
- Os identificadores não podem conter espaços em braço ou hífen:
 - **Exemplo:** caixa postal ou caixa-postal
 - **Contra-exemplo:** caixa_postal
- Os identificadores não podem conter acentos ou caracteres especiais (exceto: _):
 - **Exemplo:** %Juros, @luno, Média
 - **Contra-exemplo:** _Juros

Os identificadores armazenados em memória representam o nomes das variáveis (dados) que se alteram ao longo do tempo na execução do algoritmo.

TIPOS DE DADOS – Os tipos determinam a natureza das variáveis que podem ser assim agrupadas:

- **INTEIRO:** Pertencem ao conjunto dos números inteiros – \mathbb{Z}
- **REAL:** Pertencem ao conjunto dos números reais – \mathbb{R}
- **CARACTERE:** Uma sequência de símbolos alfabéticos, representados entre aspas “”.

- **LÓGICO:** Variável que somente pode assumir duas situações (Biestável).

As variáveis (dados) dos tipos inteiros e reais seguem a notação matemática de conjunto e são assim representados:

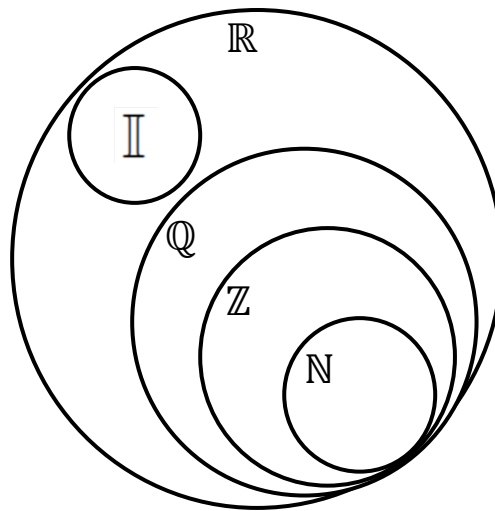


FIGURA 11 – Representação de conjunto

Exemplos de variáveis – Inteiras:

- ✓ Constam **175** CDs e **55** DVDs em **estoque**.
- ✓ O prédio tem exatos 30 **andares**.
- ✓ Tivemos que estornar negativamente **500** pares de tênis do **mostruário**.

Exemplos de variáveis – Reais:

- ✓ A **temperatura** ambiente é de **-5.3** Graus Celsius em NY.
- ✓ A **inflação** deste mês foi **0.641** por cento com certa precisão.
- ✓ Um **total** de R\$ **962.85** foi gasto no supermercado este mês.

Exemplos de variáveis – Caracteres:

- ✓ O **nome** do bebê é “**Ana**”.
- ✓ O **sexo** do animal é “**fêmea**”.
- ✓ O nome da **rua** é “**Projetada A**” provisoriamente.



Exemplos de variáveis – Lógicas: Vamos convencionar que as variáveis lógicas podem receber somente como valores Zero (**0**) ou Um (**1**). Sendo que:

- ZERO (**0**): Falso / Errado / Não / Desligado / Fechado
- UM (**1**): Verdadeiro / Certo / Sim / Ligado / Aberto

EXEMPLOS:

- A **questão** da prova está *certa* ou *errada*.
- A **porta** pode estar *aberta* ou *fechada*.
- No júri sua **resposta** apenas será *verdadeira* ou *falsa*.
- Sua **decisão** final é *sim* ou *não* para o caso.

Todas as variáveis lógicas acima (destacadas em negritos) somente podem receber como valores Zero (**0**) ou Um (**1**).

PADRÃO:

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Tipos de variáveis e Identificadores
PADRÃO	inteiro Degraus, Pessoas, Carros; real Temperatura, Inflacao, Saldo; caractere Nome, Sexo, Rua; logico Questao, Resposta, Decisao, Porta;

COMANDO DE ATRIBUIÇÃO – O comando de atribuição nos permite fornecer e alterar o valor de uma variável ao longo do algoritmo e no padrão utilizado aqui será utilizado o símbolo: **=**



PADRÃO:

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Comando de Atribuição
PADRÃO	inteiro Degraus = 75, Pessoas, Carros = 302; real Temperatura = -5.2, Inflacao, Saldo = 987.58; caractere Nome = “Fulano”, Sexo, Rua = “Projetada A”; logico Questao = 1, Resposta = 0, Decisao, Porta = 0;

➤ Expressões: Aritméticas, Relacionais e Lógicas

EXPRESSÕES ARITMÉTICAS – São cálculos cujos operadores são aritméticos e os operandos podem ser constantes ou variáveis do tipo numérico (inteiro ou real).

Operador	Função	Exemplos
+	Adição	$2 + 3$, $x + y$
-	Subtração	$4 - 1.8$, $n - 5$
*	Multiplicação	$3.76 * 9$, $5.27 * p$
/	Divisão	$9.2 / 2.8$, $N1 / N2$
%	Resto (somente inteiro)	$10 \% 3$, $X \% Y$

EXPRESSÕES RELACIONAIS – São cálculos cujos operadores estabelecem uma comparação entre valores que podem ser constantes ou variáveis. Os cálculos relacionais, assim como os lógicos, também são biestáveis, ou seja, o resultado somente pode apresentar dois estados: verdadeiro ou falso, certo ou errado.

Com isso, os resultados também podem ser representados por Zero (0) ou Um (1).



Operador	Função	Exemplos
==	Igual a	$3 == 3$, $x == -15.8$
!=	Diferente de	$5 != 3$, $x != y$
>	Maior que	$15.4 > 4.8$, $x > -1$
>=	Maior ou igual que	$15 >= 15$, $y >= x$
<	Menor que	$31 < 4$, $x < -3$
<=	Menor ou igual que	$31 <= 4$, $x <= -3.7$

OBSERVAÇÃO: Os operadores $>$ e $<$ quando em comparações de igual valor retornam sempre Falso (0). Exemplos:

- $5 > 5$ implica em **Falso** (0).
- $-15.2 < -15.2$ implica em **Falso** (0).

EXPRESSÕES LÓGICAS – São cálculos utilizando variáveis lógicas e os seguintes operadores:

Operador	Função	Exemplos
!	Negação (não)	$!x$, $!(y)$
&&	Conjunção (e)	$x \&\& y$
	Disjunção (ou)	$y x$

As operações envolvendo os operadores lógicos compõem uma estrutura denominada de tabela verdade. Portanto, **Tabela Verdade** é o conjunto de todas as possibilidades combinatórias entre os valores de diversas variáveis lógicas, as quais apresentam por valor apenas dois estados: Zero (**0**) ou Um (**1**).

Construiremos a tabela-verdade com o objetivo de dispor de uma maneira prática os valores lógicos envolvidos em uma expressão lógica. Os operadores lógicos são:



Operação de negação

A	!A
0	1
1	0

Operação de disjunção A ou B

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

Operação de conjunção A e B

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

EXEMPLO:

- ✓ Se *chover* **e** *relampejar*, os pássaros não voam. Como é conjunção (e), os pássaros não voam se *chover* e *relampejar* forem simultaneamente condições verdadeiras (1).
- ✓ Se *chover* **ou** *relampejar*, os pássaros não voam. Como é disjunção (ou), os pássaros não voam se *chover* ou *relampejar* amplia a possibilidade dos pássaros não voarem em três circunstâncias diferentes: somente *chovendo*, somente *relampejando*, *chovendo* e *relampejando* simultaneamente.

As variáveis lógicas, por apresentarem apenas dois estados (0 ou 1), podem ser combinadas numa proporção de 2^n , onde n é o número de variáveis. **Exemplo de combinações:**

Uma variável (A): 2^1	
A	
0	
1	

Duas variáveis (A – B): 2^2	
A	B
0	0
0	1
1	0
1	1

Três variáveis (A – B – C): 2^3		
A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

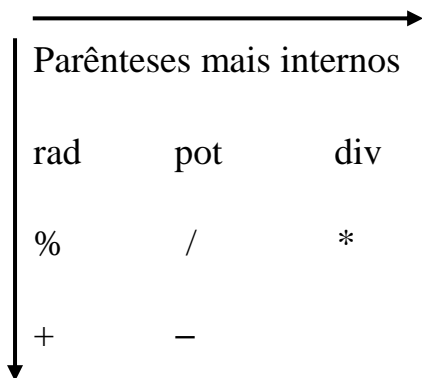


FUNÇÃO – São operações aritméticas realizadas sobre os termos envolvidos nas operações.

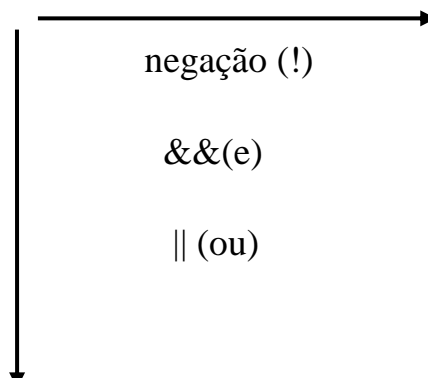
Operador	Função	Significado	Exemplos
pot (x, y)	Potenciação	x elevado a y	pot (2, 3)
rad (x)	Radiciação	Raiz quadrada de x	rad (9)
div (x, y)	Quociente	Quociente inteiro	div (9, 5)

PRIORIDADE DE OPERADORES – É a ordem como as expressões devem ser calculadas em função dos operadores envolvidos.

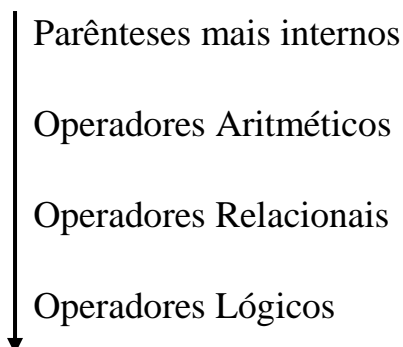
Operadores Aritméticos:



Operadores Lógicos:



E entre operadores:



Os parênteses sempre alteram e priorizam a ordem de execução das expressões.

Exemplo: $(4 - 5) * 6 \rightarrow -6$

Exemplo: $4 - 5 * 6 \rightarrow -26$

➤ COMANDOS DE ENTRADA E SAÍDA

Os algoritmos precisam ser “alimentados” com dados para efetuarem as operações e cálculos (aritméticos e lógicos) que são necessários a fim de alcançar o resultado (objetivo) desejado. Vejamos uma analogia desse processo com uma atividade que nos é corriqueira, como a respiração.

No processo respiratório, inspiramos os diversos gases que compõem a atmosfera; realizamos uma **entrada** de substância que agora são **processadas** pelo organismo, sendo que, depois de devidamente aplicados por ele, serão devolvidos, alterados, ao meio, como **saída** de substâncias.

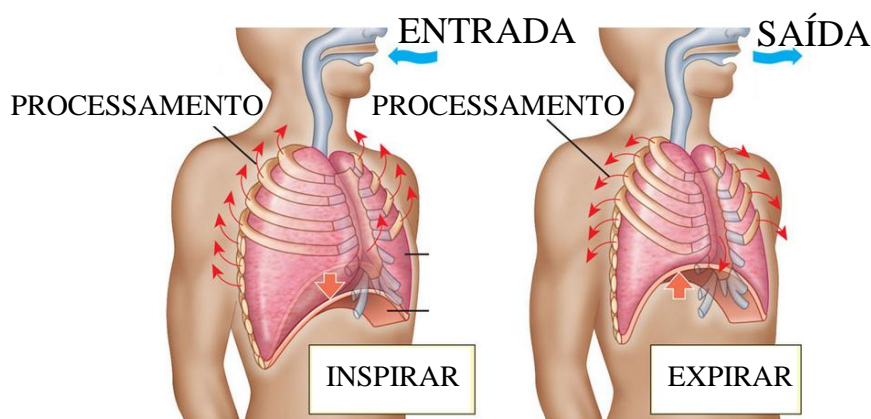


FIGURA 12 – Processo respiratório: Entrada – Processamento – Saída de ar.

Em analogia, o computador também realiza as atividades de Entrada, Processamento e Saída de dados.

ENTRADA DE DADOS – Para que o algoritmo possa receber os dados de que necessita adotaremos um comando de **entrada de dados** denominado **leia**, cuja finalidade é atribuir o dado a ser fornecido à variável identificada.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Comando de Entrada de Dados
PADRÃO	leia (Saldo); leia (Nome, CPF, Idade);



SAÍDA DE DADOS – Para que o algoritmo possa mostrar os dados que calculou, como resposta ao problema que resolveu, adotaremos um comando de **saída de dados** denominado **escreva**, cuja finalidade é exibir o conteúdo da variável identificada.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Comando de Saída de Dados
PADRÃO	escreva ("Carros em estoque: ", QuantidadeCarros); escreva ("Bom dia: ", Nome); escreva ("Você pesa: ", Peso, "quilos");

➤ BLOCO DE INSTRUÇÕES

Um bloco de instruções pode ser definido como um conjunto de ações, com uma função definida; neste caso, **um algoritmo pode ser visto como um bloco**. Ele serve também para definir os limites nos quais as variáveis declaradas em seu interior são conhecidas.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Bloco de Instruções
PADRÃO	Programa // Início do Algoritmo Início // Início do bloco // Declaração das variáveis // seqüência de ações Fim. // Fim do bloco

COMENTÁRIOS NO ALGORITMO – Com o objetivo de comentarmos certas linhas importantes nos algoritmos, estaremos utilizando os símbolos // seguido das explicações e anotações, se necessárias. **Não é obrigatório** o uso de comentários e anotações.



UNIDADE 3 – Estruturas de Controle

O QUE VAMOS ESTUDAR:

- **Estrutura de Controle do Fluxo de Execução.**
- **Estrutura Sequencial.**
- **Estrutura de Seleção.**
- **Estrutura de Repetição.**

Na criação de algoritmos, utilizamos os conceitos de **bloco de instruções, entrada e saída de dados, variáveis, constantes, atribuição, expressões aritméticas, lógicas e relacionais**, bem como, comandos que traduzem esses conceitos em um “padrão estruturado” estabelecendo uma falsa codificação como forma de expressão do algoritmo.

Todos esses comandos se relacionam formando assim a **sequencia de passos** necessária a realização do **objetivo** almejado pelo algoritmo. Essa sequência obedece rigorosamente uma ordem encadeada de ações estabelecendo assim o **fluxo de execução** do algoritmo.

Através das **estruturas básicas de controle** do fluxo de execução – **seqüenciamento, seleção e repetição** – e da combinação delas, poderemos criar algoritmos para solucionar nossos problemas.

➤ **Estrutura Sequencial**

A **estrutura sequencial** de um algoritmo corresponde ao fato de que o conjunto de ações primitivas será executado em uma seqüência linear de cima para baixo e da esquerda para direita, isto é, na mesma ordem em que foram escritas.

Convencionaremos que **as ações serão seguidas por um ponto-e-vírgula**, que objetiva separar uma ação da outra e auxiliar na organização sequencial das ações, pois após encontrar um (;) deveremos seguir para a próxima ação e assim sucessivamente até o fim do bloco de instruções.



O padrão a seguir demonstra esse seqüenciamento de ações numa forma estruturada:

	PSEUDOCÓDIGO: Português estruturado	
	COMANDO: Estrutura seqüencial	
PADRÃO	Programa	// Início do Algoritmo
	Início	// Início do bloco
	// Declaração das variáveis	
	// seqüência de ações	
	// Ação 1;	
	// Ação 2;	
	// Ação 3;	
	.	
	.	
	// Ação n;	
	Fim.	// Fim do bloco

Essa estrutura seqüencial de ações visa justamente atingir o objetivo estabelecido para resolução do problema através do algoritmo. Essas **ações correspondem aos comandos de entrada e saída, expressões aritméticas, relacionais e lógicas** assim como as estruturas de **seleção e repetição**.

O **comando de atribuição (=)** altera o valor das variáveis ao longo do seqüenciamento do bloco de instruções (Comandos e expressões matemáticas) através dos operadores aritméticos, relacionais e lógicos. Desta forma, os dados são transformados na seqüência de ações e as informações de saída podem ser exibidas como resolução do problema.

O último padrão (**estrutura seqüencial**) estabelece o **modelo básico** para desenvolvimento dos algoritmos. Ao longo da disciplina, utilizaremos o **modelo básico** como ponto de partida para representação textual do algoritmo. Esse modelo básico padroniza um falso código para elaboração dos algoritmos privilegiando seu desenvolvimento e não detalhes computacionalmente válidos pertencentes a uma verdadeira codificação.



➤ Estrutura de Seleção

Uma estrutura de **seleção** permite a escolha de um grupo de ações (Bloco) a ser executado sob certas **condições**, representadas por expressões relacionais ou lógicas, quando estas são satisfeitas ou não.

Portanto, **Condição** é uma **expressão relacional ou lógica** que, quando inspecionada, pode resultar como resposta: falso (0) ou verdadeiro (1).

SELEÇÃO SIMPLES – Este comando é utilizado quando precisamos testar uma única condição antes de executarmos **uma ou mais ações** (Bloco) dentro do algoritmo sob certas condições. A seleção simples segue como padrão.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Seleção Simples (única ação)
PADRÃO	se (Condição) então // Ação 1;

Padrão de seleção simples para um bloco de ações:

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Seleção Simples (Duas ou mais ações)
PADRÃO	se (Condição) então início // Ação 1; // Ação 2; . . . // Ação n; fim;

A ação ou ações (do bloco) é executada se a condição for **Verdadeira** e nenhuma ação é executada se a condição for **Falsa**.



SELEÇÃO COMPOSTA – Este comando é necessário quando uma situação possui duas alternativas sob a mesma condição, uma verdadeira e outra falsa usaram esta estrutura como padrão.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Seleção Composta (única ação)
PADRÃO	se (Condição) então // Condição Verdadeira // Ação 1; senão // Condição Falsa // Ação 2;

Seleção composta para bloco de ações:

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Seleção Composta (várias ações)
PADRÃO	se (Condição) então início // Condição Verdadeira //Ação 1; //Ação 2; fim ; senão // Condição Falsa //Ação 3;

SELEÇÃO ENCADEADA – Quando, devido à necessidade do algoritmo, agrupamos várias seleções, formamos uma **seleção encadeada**. Normalmente, tal formação ocorre quando uma determinada ação ou bloco deve ser executado se um grande conjunto de possibilidades ou combinações de situações for satisfeito. **Um bloco** (Inicio - Fim) é usado se **duas ou mais ações** devem ser executadas sob a mesma condição for satisfeita.



Seleção encadeada para bloco de ações:

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Seleção encadeada - (várias ações) (Duas ou mais condições)
PADRÃO	se (Condição 1) então
	início // Condição 1: Verdadeira
	//Ação 1;
	//Ação 2;
	fim ;
	senão se (Condição 2) então // Condição 1: Falsa e Condição 2: Verdadeira
	//Ação 3;
	senão // Condição 2: Falsa
	início
	//Ação 4;
	//Ação 5;
	fim ;

A **Estrutura de Seleção Encadeada** com várias ações são utilização quando **duas ou mais condições devem ser testadas**. No último padrão acima, as ações 1 e 2 são realizadas se a condição 1 for verdadeira; caso contrário, se a condição 1 for falsa, então a condição 2 é inspecionada.

Caso a condição 2 for verdadeira é executada a ação 3. Caso contrário, as ações 4 e 5 são realizadas. Relembrando que, as condições 1 e 2 são operações relacionais e / ou lógicas que apenas podem resultar em respostas do tipo **Falso (0) ou Verdadeiro (1)**.

Em qualquer estrutura, as ações que são executadas sob o mesmo bloco devem ser alinhadas no corpo do algoritmo com a mesma distância em relação à margem esquerda. A esse **alinhamento das ações que pertencem ao mesmo bloco** recebe o nome de **Indentação**.



Seleção Múltipla pode ser usada para valores unitários de uma mesma variável de escolha. A variável de escolha deve conter valores que pertençam a um grupo e não pertençam a um intervalo (contínuo ou discreto), sendo que neste caso devem ser utilizados os comandos de seleção simples, composta ou encadeada.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Seleção múltipla (Grupo de valores)
PADRÃO	<pre>escolha (Variável) início caso Valor1: //Grupo de ações; pare; caso Valor2: //Grupo de ações; pare; . . . caso contrário: //Grupo de ações; fim;</pre>

A variável pode conter o **valor1** ou **valor2** ou **valor3** ou sucessivos valores que podem ser testados, entretanto é executado o grupo de ações que pertençam ao valor escolhido. Esse grupo de ações é seguido de um comando **pare** que finaliza a execução da estrutura de seleção múltipla e faz com que os **outros casos da estrutura não sejam testados**.

No caso de nenhum valor constante pertencer aos valores válidos nos casos existentes é escolhido o “**caso contrário**” que finaliza a estrutura do comando de seleção **escolha**.



➤ Estrutura de Repetição

Uma **estrutura de repetição** permite que um grupo de ações (Bloco) possa ser executado sucessivas vezes enquanto uma determinada **condição** permanecer satisfatória. As sucessivas vezes que esse grupo de ações é repetido, denominamos esse processo de **laço** (ou *loop*).

Condição: É uma operação relacional e / ou lógica, pois estas sempre resultam como resposta o estado **FALSO (0)** ou **VERDADEIRO (1)**.

As estruturas de repetição podem ser de três tipos, a saber:

- “**enquanto**” – Repetição com teste no início.
- “**faça**” – Repetição com teste no final.
- “**para**” – Repetição com variável de controle.

Qualquer uma das estruturas pode ser utilizada para um mesmo objetivo e sua aplicabilidade depende da natureza do problema (Algoritmo) a ser resolvido. Seguem os padrões e algumas recomendações de aplicabilidade e comentários sobre suas estruturas.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Repetição com teste no início – “ enquanto ”
PADRÃO	enquanto (Condição) faça início //Ação 1; //Ação 2; ... //Ação n; fim;

Lembrando que as ações 1, 2, 3,..., **n** pertencem ao mesmo bloco e por isso devem sempre estar delimitadas pelas palavras **início** e **fim** do bloco nos três casos.



	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Repetição com teste no fim – “ faça ”
PADRÃO	faça início //Ação 1; //Ação 2; ... //Ação n; fim; enquanto (condição);

As estruturas de repetição com teste no início ou no final são bem semelhantes e podem ser utilizadas principalmente onde se queira executar sucessivas vezes o grupo de ações sendo que **não se é conhecido previamente o número de vezes do laço**.

Entretanto, não é impedimento que seja também utilizada uma variável de controle para delimitar o número de **iterações**. Sendo que, **Iteração** é a repetição sucessiva em passos regulares dado de forma a **incrementar ou decrementar a variável de controle da condição**.

	PSEUDOCÓDIGO: Português estruturado
	COMANDO: Repetição com variável de controle – “ para ”
PADRÃO	para (Início; Condição; Iteração) início //Ação 1; //Ação 2; ... //Ação n; fim;

A estrutura de repetição com **variável de controle** possui três “parâmetros” que devem ser bem definidos na sua utilização. Esses parâmetros são:

- **Início:** Determina o valor inicial da variável de controle.
- **Condição:** Determina o teste (relacional e/ou lógico) de parada do laço através da variável de controle.
- **Iteração:** Aumenta ou diminui a variável de controle em **passos regulares**.

Como previamente dito, cada estrutura de repetição pode ser utilizada nos mais diversos casos não tendo restrições específicas em relação a sua utilização. Entretanto, algumas recomendações são pertinentes e podem nortear a escolha da estrutura mais adequada para a resolução do algoritmo.

Essas recomendações são:

1. Se a quantidade de vezes do laço de iteração for previamente conhecida, a estrutura mais adequada é a com variável de controle (“**para**”);
2. Se a condição de teste for mais previamente conhecida do que o número de iterações é mais adequado a estrutura de repetição com teste no início (“**enquanto**”);
3. Além do item anterior, se também é necessário que pelo menos uma vez o grupo de ações seja executado, a estrutura mais adequada é a com teste no final (“**faça**”);
4. As estruturas de repetição com teste no início (“**enquanto**”) ou no final (“**faça**”), além da condição previamente conhecida, também podem ser utilizadas com uma variável de controle em seu teste relacional e/ou lógico.

Todas as três estruturas de repetição devem ser utilizadas com suas ações delimitadas pelo bloco de controle através das palavras **início** e **fim** do bloco. Essas palavras determinam quais são as ações que pertencem ao bloco e devem ser obrigatoriamente executadas até o final.

UNIDADE 4 – Codificação

O QUE VAMOS ESTUDAR:

- **Codificação: Utilização de uma Linguagem de Programação.**
- **Ambiente de programação: Linguagem de Programação, Editor e Compilador.**

O estudo de um pseudocódigo nos possibilita a princípio o treinamento da lógica de programação, correção do raciocínio e o desenvolvimento de algoritmos sem compromisso de se tornarem verdadeiramente programas de computadores. Entretanto, esse estudo deve aprofundar-se para uma estrutura de programação realmente padronizada desfazendo **a falsa impressão causada de que o padrão não é significativo** no desenvolvimento (textual) do algoritmo e conseqüentemente do raciocínio.

A utilização de uma linguagem de programação corrige esta impressão inicial causada pelos falsos códigos e nos coloca diante de uma estrutura de programação consolidada ao longo de anos de estudo, desenvolvimento e aprimoramento com o objetivo de estabelecer uma verdadeira **codificação do raciocínio** numa lógica computacionalmente válida. Isso é possível através do estudo de uma **Linguagem de Programação**.

Entretanto, existem inúmeras linguagens de programação. Neste estudo vamos focar numa linguagem que representa a consolidação de anos de desenvolvimento e uma gama de programas e projetos implementados com sucesso. Em virtude disso, nos estudos da lógica de programação desta disciplina será utilizada a **Linguagem de Programação C**.

➤ **Ambiente de programação: linguagem de programação, editor e compilador**

Uma linguagem de programação estrutura estabelece rígidos padrões matemáticos: **Códigos** que devem ser seguidos na elaboração dos algoritmos. Por sua vez, esta linguagem é suportada por uma ferramenta computacional que transforma o **Algoritmo** em um **Programa** de computador, essa ferramenta é o **Compilador**.

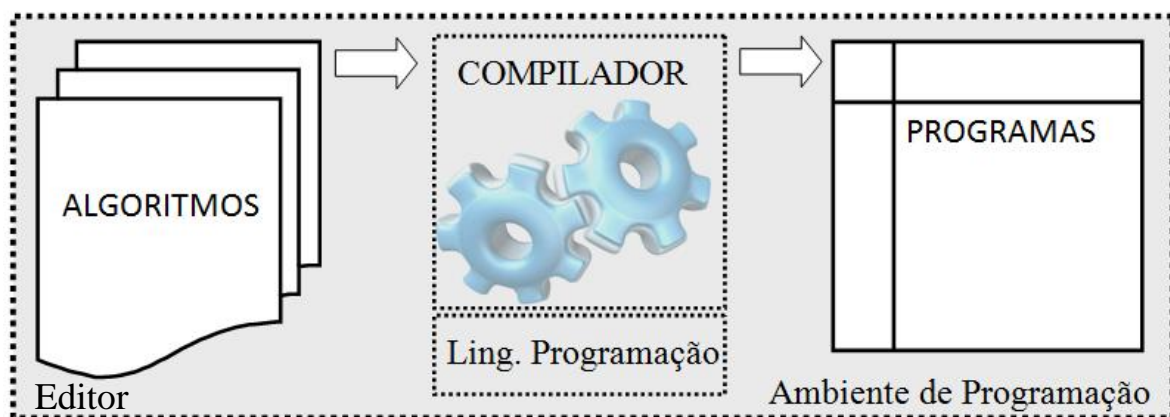


FIGURA 13 – Ambiente de programação (IDE: *Integrated Development Environment*).

O **Ambiente de Programação** é formado pelo **Compilador**, por um **Editor** para desenvolvimento (textual) do algoritmo e por uma **Linguagem de Programação**. Os algoritmos desenvolvidos no formato textual são submetidos ao compilador que **analisa o código** conforme os padrões estabelecidos na linguagem de programação (**Análise Léxica** do Código) e aponta os possíveis erros na escrita e na sequência dos comandos e estruturas (**Análise Sintática**) do Código.

Caso o algoritmo escrito não contenha erros conforme o padrão da **Linguagem de programação C**, o compilador o transforma o arquivo texto do algoritmo (**Arquivo: .C**) em um **Programa** de Computador (**Executável: .EXE**). Existem inúmeros ambientes de programação e seus compiladores, nesta disciplina será utilizado o **IDE: CodeBlocks** (<http://www.codeblocks.org>).



UNIDADE 5 – Linguagem de Programação C

O QUE VAMOS ESTUDAR:

- **Linguagem C: Breve histórico.**
- **Variável: Tipos e declaração.**
- **Operadores matemáticos em C.**
- **Entrada e saída de dados.**
- **Estrutura de seleção de dados.**
- **Estrutura de repetição.**

A Linguagem de Programação C é uma das mais utilizadas nos mais diversos casos práticos de programação, tanto na engenharia quanto na informática. Sua origem se confunde com a própria criação dos computadores e realmente compôs sua formação e desenvolvimento. Cronologicamente, podemos citar alguns breves fatos e pessoas marcantes desta história:

- ✓ **1965:** O Sistema Operacional Unix tem suas raízes no projeto MULTICS (Multiplexed Information and Computing Service), desenvolvido por grandes instituições da época: AT&T, GE (*General Electric*) e o MIT (*Massachusetts Institute of Technology*).
- ✓ **1966:** **BCPL** é uma linguagem de programação, criada por Martin Richards, da Universidade de Cambridge. O nome **BCPL** é um acrônimo para **Basic Combined Programming Language** (Linguagem de Programação Básica Combinada).
- ✓ **1969 – 1973:** O desenvolvimento inicial de C ocorreu no AT&T Bell Labs. Não se sabe se o nome "C" foi dado à linguagem porque muitas de suas características derivaram da linguagem B e C é a letra conseqüente no alfabeto, ou porque "C" é a segunda letra do nome da linguagem BCPL, da qual se derivou a linguagem B.
- ✓ **1973 – 1983:** Com sua filosofia de simplicidade, padrões abertos e seu licenciamento facilitado pela AT&T, o Unix se espalhou e se desenvolveu rapidamente pelas universidades. Várias versões de Unix foram surgindo neste período.
- ✓ **1980 – ATUAL:** Outras versões comerciais também foram surgindo, tais como: Irix pela **SGI** em 1982, XENIX pela **SCO (Microsoft)** em 1983, **HP-UX** pela **HP** em



1986, SunOS pela **Sun** em 1987 e AIX pela **IBM** em 1990 e até o Mac OS X (**APPLE**) (baseado em um núcleo Mach BSD chamado Darwin).



FIGURA 14 – Criadores da **Linguagem C de Programação** (Fonte: <https://pt.wikipedia.org>).

Com mais de 50 anos de pesquisa, desenvolvimento e aprimoramento, a linguagem de programação C é uma das mais bem consolidadas e estruturadas formas de programação. Responsável também pelo surgimento de diversas outras linguagens: C, C++, C# e JAVA que tomam por base seus códigos (**Análise Léxica**) e estruturas (**Análise Sintática**).

➤ Variável: Tipos e declaração.

Variáveis são espaços em memória caracterizados pelos seus identificadores e seus valores são alteráveis ao longo do tempo. As variáveis podem ser de três **tipos básicos**:

NOME	DESCRIÇÃO	ESPAÇO DE MEMÓRIA	INTERVALO
char	Caracteres ou inteiros*	1 byte	Com sinal: [-128 127] Sem sinal: [0 255]
int	Inteiros	4 byte	Com sinal: [-2147483648 2147483647] Sem sinal: [0 4294967295]
float	Real	4 byte	+ / - 3.4 e + / - 38 (~ 7 dígitos)

1. **Tipo de Dados Inteiros (int):** é a variável que representa os valores do conjunto dos números inteiros.
 - a. \mathbb{Z} : { . . . , -3, -2, -1 , 0, 1, 2 , 3, . . . }
 - b. Exemplos:
 - i. Constam 175 CDs e 55 DVDs em estoque.
 - ii. Ele chegou na 56ª posição no Triátlon.
 - iii. Tivemos que estornar negativamente 500 pares de tênis do mostruário.
2. **Tipo de Dados Real (float):** é a variável que representa os valores do conjunto dos números reais.
 - a. \mathbb{R} :]-∞ ∞[
 - b. Exemplos:
 - i. A temperatura ambiente é de -5.3 Graus Celsius em NY.
 - ii. A constante π é 3.14159 com certa precisão.
 - iii. Um total de R\$ 162.81 foi gasto no supermercado este mês.
3. **Tipo de Dados Caractere (char):** Um caractere, no contexto da informática, é o nome que se dá a cada um dos símbolos que se pode usar para produzir um programa de computador.
 - i. Modo de escrita como caractere: Deve ser usado com aspas simples ' '.
 - ii. Exemplos: 'A' , 'B' , 'C' , 'D' , 'F' , 'G' , ..., 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , ...
'@' , '\$' , '£' , '¢' , '#' , '|' , '\$' , ..., '°' ⇒Graus Celsius
'M' ⇒Masculino
'F' ⇒Feminino



- iii. A **variável caractere (char)** pode assumir valores *inteiros compreendidos no intervalo fechado [0 255]. Esses valores correspondem aos caracteres (códigos-símbolos) estabelecidos internamente ao Sistema Operacional pela **TABELA ASCII**.

código	caracter	código	caracter	código	caracter	código	caracter
000	CTRL-@	032	(BRANCO)	064	@	096	`
001	CTRL-A	033	!	065	A	097	a
002	CTRL-B	034	"	066	B	098	b
003	CTRL-C	035	#	067	C	099	c
004	CTRL-D	036	\$	068	D	100	d
005	CTRL-E	037	%	069	E	101	e
006	CTRL-F	038	&	070	F	102	f
007	CTRL-G	039	'	071	G	103	g
008	CTRL-H	040	(072	H	104	h
009	CTRL-I	041)	073	I	105	i
010	CTRL-J	042	*	074	J	106	j
011	CTRL-K	043	+	075	K	107	k
012	CTRL-L	044	,	076	L	108	l
013	CTRL-M	045	-	077	M	109	m
014	CTRL-N	046	.	078	N	110	n
015	CTRL-O	047	/	079	O	111	o
016	CTRL-P	048	0	080	P	112	p
017	CTRL-Q	049	1	081	Q	113	q
018	CTRL-R	050	2	082	R	114	r
019	CTRL-S	051	3	083	S	115	s
020	CTRL-T	052	4	084	T	116	t
021	CTRL-U	053	5	085	U	117	u
022	CTRL-V	054	6	086	V	118	v
023	CTRL-W	055	7	087	W	119	w
024	CTRL-X	056	8	088	X	120	x
025	CTRL-Y	057	9	089	Y	121	y
026	CTRL-Z	058	:	090	Z	122	z
027	CTRL-[059	;	091	[123	{
028	CTRL-\	060	<	092	\	124	
029	CTRL-]	061	=	093]	125	}
030	CTRL-^	062	>	094	^	126	~
031	CTRL-_	063	?	095	_	127	DEL

TABELA ASCII – ORIGINAL



código	caracter	código	caracter	código	caracter	código	caracter
128	Ç	160	á	192	+	224	Ó
129	ü	161	í	193	-	225	β
130	é	162	ó	194	-	226	Ô
131	â	163	ú	195	+	227	Ò
132	ä	164	ñ	196	-	228	õ
133	à	165	Ñ	197	+	229	Õ
134	å	166	a	198	ã	230	μ
135	ç	167	o	199	Ã	231	þ
136	ê	168	ı	200	+	232	Ɔ
137	ë	169	®	201	+	233	Ú
138	è	170	¬	202	-	234	Û
139	ï	171	½	203	-	235	Ù
140	î	172	¼	204	ı	236	Ý
141	ì	173	i	205	-	237	Ý
142	Ä	174	«	206	+	238	—
143	Å	175	»	207	×	239	'
144	É	176	_	208	ð	240	
145	æ	177	_	209	Ð	241	±
146	Æ	178	_	210	Ê	242	_
147	ô	179	ı	211	Ë	243	¾
148	ö	180	ı	212	È	244	¶
149	ò	181	Á	213	i	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	ı	217	+	249	¨
154	Ü	186	ı	218	+	250	·
155	ø	187	+	219	_	251	¹
156	£	188	+	220	_	252	³
157	Ø	189	¢	221	ı	253	²
158	×	190	¥	222	Ì	254	_
159	f	191	+	223	_	255	

TABELA ASCII – ESTENDIDA

iv. Caracteres especiais precedidos do contrabarra ou barra inversa: '\n' e '\t'

Sequência	Equivale a
\n	Nova linha
\t	Tabulação



➤ Declaração de variáveis

Em linguagem de programação C, as variáveis devem ser declaradas no ambiente de programação conforme o formato a seguir:

= ;

TIPO DA VARIÁVEL NOME DA VARIÁVEL VALOR INICIAL

- i. **Tipo de variável:** Os tipos mais comuns aqui apresentados são: Caractere (**char**), Inteiro (**int**) e Reais (**float**).
- ii. **Nome da variável:** Descreve o nome da variável utilizada ao longo do programa. Possuem por características:
 - ✓ Não podem conter caracteres especiais (Ex: #nota, &resultado1);
 - ✓ Não podem iniciar com números (Ex: 1A, -5B);
 - ✓ Não podem ser maiores que 32 caracteres;
 - ✓ Não podem conter acentos. (Ex: área, média);
 - ✓ A Linguagem de Programação C faz distinção entre variáveis escritas com caracteres maiúsculos e minúsculos, ou seja, representam variáveis diferentes. Essa característica denomina-se de “*case sensitive*”, ou seja, do inglês, sensível ao tamanho, sendo que esse tamanho refere-se a maiúsculas e minúsculas na definição de variáveis. (Exemplo de **variáveis diferentes: nota ≠ Nota**).
- iii. **Atribuição de valores:** o caractere reservado = atribui um valor inicial a variável declarada.

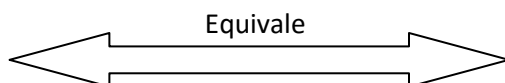
Exemplos de variáveis declaradas **NÃO INICIALIZADAS**:

- ✓ Exemplo de declaração de variável inteira a, b e c:

int a;

int b;

int c;



int a, b, c;



- ✓ Exemplo de declaração de variável inteira (**int**) **a**, **b** do tipo real (**float**) e **c** do tipo caractere (**char**):

```
int a;
```

```
float b;
```

```
char c;
```

- ✓ Exemplos de variáveis declaradas INICIALIZADAS: Na declaração da variável é atribuído um VALOR INICIAL, ou seja, esta variável é dita como inicializada. Exemplos:

```
int a = -10252;
```

```
int Teste = 1;      // Comentário: verdadeiro (true) = 1
```

```
int Questao = 0;    // Comentário: falso(false) = 0
```

```
float s = 0;
```

```
const float PI = 3.14159; // Declaração de constantes
```

```
char Opcao = 'F';
```

```
char Letra = 65;    /* Comentário: O conteúdo da variável Opcao equivale ao caractere 'A' (código: 65) da TABELA ASCII */
```

- ✓ Quando as variáveis declaradas devem ser INICIALIZADAS?
 - a. As variáveis lidas do usuário e calculadas (objetivos) NÃO são inicializadas na declaração;
 - b. As variáveis de controle e as variáveis acumulativas de valores DEVEM ser inicializadas na declaração.

➤ Operadores Matemáticos na Linguagem C

Os operadores matemáticos contidos na Linguagem de Programação C estão agrupados em três categorias:

1. OPERADOR ARITMÉTICO
2. OPERADOR RELACIONAL
3. OPERADOR LÓGICO

- i. **Operador Aritmético:** São responsáveis pelas operações matemáticas elementares como: **Adição, Subtração, Multiplicação, Divisão e Resto Inteiro** (da divisão).

OPERAÇÃO	SÍMBOLO		
ADIÇÃO	+	++	+=
SUBTRAÇÃO	-	--	-=
MULTIPLICAÇÃO	*		*=
DIVISÃO	/		/=
RESTO (Divisão)	%		%=
EXPONENCIAÇÃO	NÃO EXISTE		

No ambiente de programação C devem ser utilizados conforme os exemplos:

EXEMPLOS	COMENTÁRIO
<pre>int a = 11, b, c; c = 2 * a; b = a % 3; b++; c = b * 2;</pre>	<pre>// a = 11, b = ?, c = ? // a = 11, b = ?, c = 22 // a = 11, b = 2, c = 22 // a = 11, b = 3, c = 22 // a = 11, b = 3, c = 6</pre>
<pre>int a = 9, b, c; a *= 2; b = a % 6; -b; c = b * 2; c -= 3;</pre>	<pre>// a = 9, b = ?, c = ? // a = 18, b = ?, c = ? // a = 18, b = 0, c = ? // a = 18, b = -1, c = ? // a = 18, b = -1, c = -2 // a = 18, b = -1, c = -5</pre>



✓ Propriedades:

- O operador “++” incrementa uma unidade (1) a variável;
- O operador “--” decrementa uma unidade (1) a variável;
- O operador “%” retorna o resto da divisão. **SOMENTE** pode ser utilizada com variáveis e números inteiros (tipo **int**);
- Os operadores “+=”, “-=”, “*=”, “/=”, “%=” executam sua respectiva operação aritmética e atribuem o resultado a mesma variável contida na operação.

ii. **Operador Relacional:** Utilizado nas operações de comparação relacionais de valores. Essas operações retornam como resultado **ZERO** (false) ou **UM** (true); ou seja, **SOMENTE** valores lógicos.

SÍMBOLO	OPERAÇÃO
==	IGUAL
!=	DIFERENTE
>	MAIOR
>=	MAIOR OU IGUAL
<	MENOR
<=	MENOR OU IGUAL

✓ Exemplos:

LINHAS DE CÓDIGOS	COMENTÁRIO
<code>int a = 2, b = 5, c = -3;</code>	<code>//a = 2, b = 5, c = -3</code>
<code>float resposta;</code>	<code>//a = 2, b = 5, c = -3, resposta = ?</code>
<code>resposta = a == b;</code>	<code>//a = 2, b = 5, c = -3, resposta = 0</code>
<code>resposta = b != c;</code>	<code>//a = 2, b = 5, c = -3, resposta = 1</code>
<code>resposta = (b + c) != a;</code>	<code>//a = 2, b = 5, c = -3, resposta = 0</code>
<code>resposta = (b % a) >= c;</code>	<code>//a = 2, b = 5, c = -3, resposta = 1</code>

Observação: Condições com < e > são FALSOS para operações de iguais valores.

- iii. **Operador Lógico:** Utilizado nas operações envolvendo Álgebra Booleana. Essas operações retornam como resultado **ZERO** (false) ou **UM** (true); ou seja, **SOMENTE** valores lógicos, conforme a natureza da sentença (Linha de código) executada no programa.

SÍMBOLO	OPERAÇÃO
!	NOT (Negação)
&&	AND (e)
	OR (ou)

✓ Operadores algébricos booleanos:

- OPERADOR DE NEGAÇÃO (not): Símbolo na linguagem C: !

NOT			NOT	
a	!a		a	!a
FALSE	TRUE	↔	0	1
TRUE	FALSE		1	0

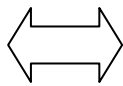
- OPERADOR DE DISJUNÇÃO (or): Símbolo na linguagem C: ||

OR				OR		
a	b	a + b		a	b	a + b
		a b				a b
FALSE	FALSE	FALSE	↔	0	0	0
FALSE	TRUE	TRUE		0	1	1
TRUE	FALSE	TRUE		1	0	1
TRUE	TRUE	TRUE		1	1	1

➤ OPERADOR DE CONJUNÇÃO (and): Símbolo na linguagem C: &&

AND

a	b	a * b
		a && b
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE



AND

a	b	a * b
		a && b
0	0	0
0	1	0
1	0	0
1	1	1

✓ Exemplos:

LINHAS DE CÓDIGOS

```
int a = 0, b = 1, c = 0;
int resposta;
resposta = !a;
resposta = b && c;
resposta = (!a && b) || c;
resposta = (a || b) && !c;
```

COMENTÁRIO

```
//a = 0, b = 1, c = 0
//a = 0, b = 1, c = 0, resposta = ?
//a = 0, b = 1, c = 0, resposta = 1
//a = 0, b = 1, c = 0, resposta = 0
//a = 0, b = 1, c = 0, resposta = 1
//a = 0, b = 1, c = 0, resposta = 1
```

✓ **OBSERVAÇÃO:** Variáveis reais ou inteiras podem ser utilizadas como variáveis lógicas, da seguinte forma:

$\{\forall \eta \in \mathbb{R} / \text{se } \eta = 0: \{\text{Falso}\}; \text{se não } \forall \eta \neq 0: \{\text{Verdadeiro}\}\}$

LINHAS DE CÓDIGOS

```
float a = 0, b = -5.8, c = 10;
float resposta;
resposta = !a;
resposta = b && c;
resposta = (!a && b) || c;
resposta = (a || b) && !c;
```

COMENTÁRIO

```
//a = 0, b = -5.8, c = 10
//a = 0, b = -5.8, c = 10, resposta = ?
//a = 0, b = -5.8, c = 10, resposta = 1
//a = 0, b = -5.8, c = 10, resposta = 1
//a = 0, b = -5.8, c = 10, resposta = 1
//a = 0, b = -5.8, c = 10, resposta = 0
```



➤ **Entrada e saída de dados.**

Os comandos básicos de Entrada (**Portugol: leia**) e Saída (**Portugol: escreva**) de dados nos programas podem ser executados com diversas formas e dispositivos, como:

- **ENTRADA:** Teclado, mouse, leitor óptico ou biométrico, webcam, microfones, scanner, entre outros
- **SAÍDA:** Monitor, impressora, caixa de som, entre outros.

Na linguagem de programação C, podem ser utilizados os comandos básicos de entrada (**scanf**) e saída (**printf**) no desenvolvimento dos algoritmos em substituição aos anteriormente utilizados, tendo as mesmas funcionalidades.

Os comandos de Entrada e Saída são determinados pela biblioteca (**stdio.h**) que dita seu funcionamento e suas formas de utilização. O comando de saída de dados – **printf** – pode ser utilizado da seguinte forma:

```
printf ("Mensagem"); // ou
```

```
printf("Mensagem %Tipo_Variável1 %Tipo_variável2", Variavel1, Variavel2);
```

Sendo que:

% – Demarca o local exato na tela onde aparece o valor da respectiva variável.

" " – Demarca o limite da mensagem.

Tipo – Determina o tipo da variável (**int** / **float** / **char**) a ser exibida. Ou seja:

Caractere especial - %	Tipo	Variável
%d ou %i	int	Inteiro
%f	float	Real
%c	char	Caractere



Segue alguns exemplos de como utilizar o comando de saída: **printf**.

// Comandos: Exemplos	// Tela
printf ("Linguagem de Programacao C");	Linguagem de Programacao C
printf ("Minha idade: %i anos", 22);	Minha idade: 22 anos
int Idade = 22; printf ("Minha idade: %i anos", Idade);	Minha idade: 22 anos
int CD = 85, DVD = 15; printf ("Quantidade de CD: %i e DVD: %i", CD, DVD);	Quantidade de CD: 85 e DVD: 15
float Temperatura = 21.4; printf ("Temperatura da sala: %.1f °", Temperatura);	Temperatura da sala: 21.4 °
char Simbolo = '@'; printf ("Simbolo escolhido: %c", Simbolo);	Simbolo escolhido: @
char Simbolo = 64; printf ("Simbolo escolhido: %c", Simbolo);	Simbolo escolhido: @

O comando de entrada de dados – **scanf** – pode seu utilizado da seguinte forma:

scanf ("%tipo", &Variavel); // ou

scanf ("%tipo %tipo", &Variavel1, &Variavel2);

Os valores digitados pelo usuário são armazenados na variável, conforme exemplos:

// Comandos: Exemplos	// Tela
int QuantidadeDeEntrevistados; printf ("Quantidade de Entrevistados: "); scanf ("%i", &QuantidadeDeEntrevistados);	Quantidade de Entrevistados: 1000
float Temperatura; printf ("Temperatura do dia: "); scanf ("%f", &Temperatura);	Temperatura do dia: 39.5
char Opcao; printf ("Sua opcao (Feminino-'F' ou Masculino-'M'): "); scanf ("%c", &Opcao);	Sua opcao (Feminino-'F' ou Masculino-'M'): M

➤ Estrutura de seleção de dados

As Estruturas de Seleção são blocos estruturados de comandos no ambiente de linguagem de programação aplicáveis na solução de problemas de natureza seletiva ou optativa de eventos (dados) reais. Nestes casos, a Estrutura de Seleção estabelece distinções bem definidas em relação aos dados ou fatos dos diferentes subconjuntos pertencentes ao mesmo problema.

As Estruturas de Seleção são de dois tipos:

- Estrutura de Seleção (para Intervalos de Dados): *if-else*
- Estrutura de Seleção (Para Valores Unitários) : *switch*

A estrutura de seleção *if – else* se destina a segmentar a faixa de dados em intervalos distintos de acordo com o problema. Nestes casos, onde **os valores são selecionados em intervalos (contínuos)** deve ser utilizada a estrutura do tipo *if-else*.

➤ Sem Bloco de Ações:

1º CASO:	2º CASO:	3º CASO:
<pre>if (condição) linha 1;</pre>	<pre>if (condição) linha 1; else linha 2;</pre>	<pre>if (condição A) linha 1; else if (condição B) linha 2; else linha 3;</pre>

➤ Com Bloco de Ações:

<pre>if (condição) { linha 1; linha 2; }</pre>	<pre>if (condição) linha 1; else { linha 2; linha 3; }</pre>	<pre>if (condição A) linha 1; else if (condição B) { linha 2; linha 3; } else linha 4;</pre>
--	--	--

Exemplo 1: Escrever um algoritmo em C que leia um número qualquer e exiba na tela se o dobro deste número é maior, menor ou igual a π . Declarar π (PI = 3.14) como uma constante no algoritmo.



```
// Apostila: Exemplo1

//Bibliotecas
#include <stdio.h> //printf - scanf
#include <stdlib.h> // Operadores: Aritméticos/Relacionais/Lógicos

//Programa Principal
int main()
{
    //Declaração das variáveis
    float Numero;
    const float PI = 3.14;
    printf("Digite um numero real qualquer: ");
    scanf("%f", &Numero);
    // estrutura de seleção dos dados:
    if (2 * Numero > PI)
        printf("O dobro de %.2f eh MAIOR que PI.", Numero);
    else if (2 * Numero < PI)
        printf("O dobro de %.2f eh MENOR que PI.", Numero);
    else
        printf("O dobro de %.2f eh IGUAL a constante PI.", Numero);
    printf("\n\n\n"); // Pula 3 linhas.
    return 0; // Esta linha finaliza o programa.
}
```

Exemplo 2: Um site de compras coletivas estabelece o seu critério de promoções conforme a seguir:

- Até 50 clientes cadastrados: Sem desconto;
- Mais que 50 até 100 clientes cadastrados: 15% de desconto;
- Mais que 100 clientes cadastrados: 25% de desconto.

Escrever um algoritmo em C que determine o total arrecadado e total arrecadado com desconto baseado na quantidade ($\in \mathbb{N}^*$) de clientes que entraram na promoção e no preço ($\in \mathbb{R}_+^*$) da mercadoria.



```
// Apostila: Exemplo2

//Bibliotecas
#include <stdio.h> //printf - scanf
#include <stdlib.h> // Operadores: Aritméticos/Relacionais/Lógicos

//Programa Principal
int main()
{
    //Declaração das variáveis
    float Preco, Total, Total_Desconto;
    int Quantidade;
    printf("Digite a quantidade de clientes cadastrados: ");
    scanf("%i", &Quantidade);
    printf("Digite o preco da mercadoria (em reais): ");
    scanf("%f", &Preco);
    // estrutura de seleção dos dados:
    if (Quantidade <= 0 || Preco <= 0)
        printf("Os valores devem ser positivos.\n");
    else
    {
        Total = Quantidade * Preco;
        printf("Total arrecadado: %.2f reais.\n", Total);
        if (Quantidade < 50)
            printf("Total arrecadado com desconto: Sem desconto\n");
        else if (Quantidade >= 50 && Quantidade < 100)
        {
            Total_Desconto = Total * 0.85; //15% de desconto.
            printf("Total arrecadado com desconto: %.2f reais.\n", Total_Desconto);
        }
        else
        {
            Total_Desconto = Total * 0.75; //25% de desconto.
            printf("Total arrecadado com desconto: %.2f reais.\n", Total_Desconto);
        }
    }
    printf("\n\n\n"); // Pula 3 linhas.
    return 0; // Esta linha finaliza o programa.
}
```

A Estrutura de Seleção Múltipla: **switch** se destina a segmentar os dados em valores unitários (discretos) de acordo com o problema. Pode substituir a **Seleção Encadeada** de dados somente para valores do tipo: **Inteiro (int)** e **Caractere (char)**.

FORMA ESTRUTURAL

```
switch (variavel)
{
    case Constante1:
        Grupo de comando;
        break;
    case Constante2:
        Grupo de comando;
        break;
    .
    .
    .
    default:
        Grupo de comando;
}
```

OBSERVAÇÃO: A variável da Estrutura de seleção **switch** aceita SOMENTE como tipo de dados Caractere (char) e Inteiro (int).

Nestes casos, onde **os valores são selecionados de forma unitária (discretos)** pode ser utilizada a estrutura do tipo **switch** (Tradução da Estrutura: ESCOLHA do Português Estruturado).



Comparação entre as estruturas de seleção: *if - else if - else* e *switch*.

<pre>//Estrutura de seleção: if - else #include <stdio.h> #include <stdlib.h> int main() { // Declaração das variáveis int Opcao; printf("Digite sua opcao (1 ou 2 ou 3): "); scanf("%i", &Opcao); if (Opcao == 1) printf("Sua opcao de escolha: %i\n", Opcao); else if (Opcao == 2) printf("Sua opcao de escolha: %i\n", Opcao); else if (Opcao == 3) printf("Sua opcao de escolha: %i\n", Opcao); else printf("Sua opcao de escolha eh invalida.\n"); printf("\n\n"); //Pula duas linhas return 0; }</pre>	<pre>//Estrutura de seleção: switch #include <stdio.h> #include <stdlib.h> int main() { // Declaração das variáveis int Opcao; printf("Digite sua opcao (1 ou 2 ou 3): "); scanf("%i", &Opcao); switch (Opcao) { case 1: printf("Sua opcao de escolha: %i\n", Opcao); break; case 2: printf("Sua opcao de escolha: %i\n", Opcao); break; case 3: printf("Sua opcao de escolha: %i\n", Opcao); break; default: printf("Sua opcao de escolha eh invalida.\n"); } printf("\n\n"); //Pula duas linhas return 0; }</pre>
--	---

A estrutura de seleção encadeada: *if - else if - else* pode ser utilizada para segmentar os dados em valores discretos e contínuos conforme a comparação anterior. No caso do comando *switch*, **somente** pode ser utilizado com valores discretos.

O comando ***break*** finaliza a estrutura de **Seleção Múltipla**: *switch* no caso do valor encontrado conforme a seleção da variável de escolha.

➤ Estrutura de Repetição

As Estruturas de Repetição são blocos estruturados no ambiente de linguagem de programação aplicáveis na solução de problemas onde é necessária a iteração sucessiva de comando. Nestes casos, a **Iteração** é o processo pelo qual uma única linha ou um bloco de instruções é sucessivamente executado até que uma **condição: expressão lógica e / ou relacional** tenha seu estado alterado de **Verdadeiro para Falso**.

As estruturas de repetição são de três tipos:

- Estrutura de Repetição: *for*
- Estrutura de Repetição: *while*
- Estrutura de Repetição: *do-while*



Qual estrutura de repetição utilizar? A escolha da estrutura de repetição depende das características do problema. De modo geral, **qualquer uma das três estruturas pode ser utilizada na resolução do algoritmo**, sendo que, a única estrutura que depende necessariamente de uma variável de controle é a estrutura de repetição *for*. Os **exemplos (1 ao 5)** a seguir ajudam a responder esta pergunta.

➤ **Estrutura de repetição: *for***

	Linguagem de programação C
	COMANDO: Repetição com variável de controle – <i>FOR</i>
	for (Início; Condição; Iteração) { //Ação 1; //Ação 2; ... //Ação n; }

INÍCIO: Valor inicial da contagem. A variável de controle pode ser do tipo caractere (*char*), inteiro (*int*) ou real (*float*).

CONDIÇÃO: Operação lógica e / ou relacional.

ITERAÇÃO: Repetição em etapas regulares enquanto a condição for **Verdadeira** (1). A **variável de controle** pode executar um incremento ou decremento de seu valor.

Exemplo 1: Criar um algoritmo em C (usando a estrutura de repetição *for*) que exiba na saída (Tela) uma contagem progressiva do 1 até 10 finalizando assim o programa com uma mensagem: “FIM DA CONTAGEM!”.

```
//Estrutura de repetição: for
```

```
//Exemplo1
```

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

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

```
int main()
```

```
{
```

```
    int Contador;
```

```
    char Caractere = 248; //Caractere: '°' da Tabela
```

```
    printf("Contagem progressiva do 1 ate 10: \n");
```

```
    for (Contador = 1; Contador <= 10; Contador++)
```

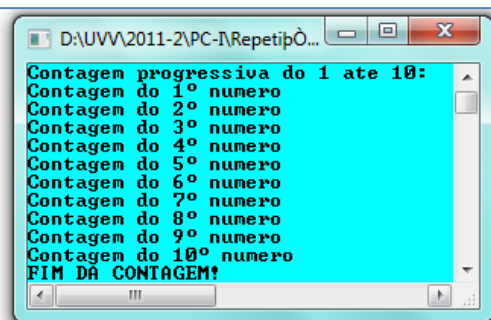
```
        printf("Contagem do %i%c numero\n", Contador, Caractere);
```

```
    printf("FIM DA CONTAGEM!\n");
```

```
    return 0;
```

```
}
```

TELA:



Exemplo 2: Criar um algoritmo em C (usando a estrutura de repetição *for*) que exiba na saída (Tela) uma contagem regressiva do 10 até 1 finalizando assim o programa com uma mensagem: “FIM DA CONTAGEM!”.

```
//Estrutura de repetição: for
```

```
//Exemplo2
```

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

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

```
int main()
```

```
{
```

```
    int Contador;
```

```
    char Caractere = 248; //Caractere: '°' da Tabela
```

```
    printf("Contagem regressiva do 10 ate 1: \n");
```

```
    for (Contador = 10; Contador >= 1; Contador--)
```

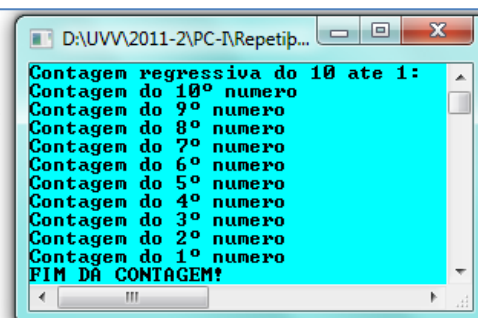
```
        printf("Contagem do %i%c numero\n", Contador, Caractere);
```

```
    printf("FIM DA CONTAGEM!\n");
```

```
    return 0;
```

```
}
```

TELA:



Nestes dois exemplos, a **variável de controle** Contador determina o final do laço através da condição (relacional) estabelecida na estrutura de repetição *for*. A repetição acontece enquanto a condição permanece Verdadeira (1), saindo da estrutura do *for* quando esta se tornar Falsa (0);

A variável Caractere (**char**), nos exemplos 1 e 2, exibe na tela o símbolo especial ° (Código = 248) da Tabela de códigos ASCII (Página 32).



➤ Estrutura de repetição: *while*

	Linguagem de programação C
	COMANDO: Repetição com teste no início – <i>while</i>
PADRÃO	<pre>while (Condição) { //Ação 1; //Ação 2; ... //Ação n; }</pre>

Exemplo 3:

Criar um algoritmo em C (usando a estrutura de repetição *while*) que exiba na saída (Tela) uma contagem progressiva do 1 até 10 finalizando assim o programa com uma mensagem: “FIM DA CONTAGEM!”.

```
//Estrutura de repetição: while
//Exemplo3
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int Contador = 1;
    char Caractere = 248; //Caractere: '°' da Tabela
    printf("Contagem progressiva do 1 ate 10: \n");
    while (Contador <= 10)
    {
        printf("Contagem do %i%c numero\n", Contador, Caractere);
        Contador++;
    }
    printf("FIM DA CONTAGEM!\n");
    return 0;
}
```

TELA:



➤ Estrutura de repetição: *do-while*

	Linguagem de programação C
	COMANDO: Repetição com teste no final – <i>do-while</i>
PADRÃO	<pre>do { //Ação 1; //Ação 2; ... //Ação n; } while (condição);</pre>

Exemplo 4:

Criar um algoritmo em C (usando a estrutura de repetição *do-while*) que exiba na saída (Tela) uma contagem progressiva do 1 até 10 finalizando assim o programa com uma mensagem: “FIM DA CONTAGEM!”.

```
//Estrutura de repetição: do-while
//Exemplo4
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int Contador = 1;
    char Caractere = 248; //Caractere: '°' da Tabela
    printf("Contagem progressiva do 1 ate 10: \n");
    do
    {
        printf("Contagem do %i%c numero\n", Contador, Caractere);
        Contador++;
    }while (Contador <= 10);
    printf("FIM DA CONTAGEM!\n");
    return 0;
}
```

TELA:

```
Contagem progressiva do 1 ate 10:
Contagem do 1° numero
Contagem do 2° numero
Contagem do 3° numero
Contagem do 4° numero
Contagem do 5° numero
Contagem do 6° numero
Contagem do 7° numero
Contagem do 8° numero
Contagem do 9° numero
Contagem do 10° numero
FIM DA CONTAGEM!
```

Nos exemplos 3 e 4 também foi utilizado uma variável de controle (Contador) para finalizar o laço.

Entretanto, nas estruturas de repetição **while** e **do-while** não necessariamente é utilizada uma variável de controle do laço, sendo que nestes casos, a condição de parada geralmente depende de uma expressão lógica.

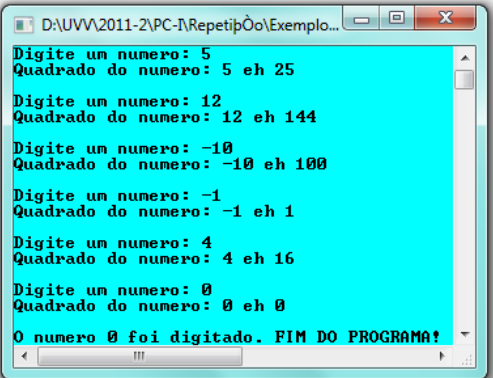
Exemplo 5:

Criar um algoritmo em C que leia um número do usuário e exiba o quadrado deste número na tela. Pare de ler mais números quando for digitado o número zero.

```

//Estrutura de repetição: do-while
//Exemplo5
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Numero;
    do
    {
        printf("Digite um numero: ");
        scanf("%i", &Numero);
        printf("Quadrado do numero: %i eh %i\n\n", Numero, Numero * Numero);
    }while(Numero);
    printf("O numero %i foi digitado. FIM DO PROGRAMA!", Numero);
    return 0;
}
        
```

TELA:

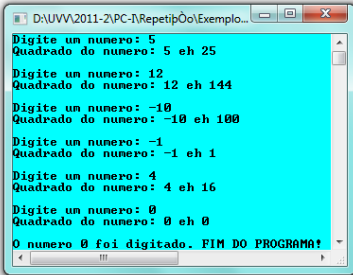


No exemplo 5, o **teste da condição** é feito sobre a variável inteira Número (Condição: **Número != 0**). Como é necessário ler o número antes de testá-lo, a estrutura de repetição **do-while** é a mais adequada nestes casos.

Como **valores positivos e negativos na Linguagem de Programação C são verdadeiros (1)**, a condição somente se tornou falsa quando o Número digitado foi o zero (0). Se este mesmo algoritmo fosse escrito utilizando a estrutura de repetição **while** seria obrigatório inicializar a variável Número com um valor verdadeiro, por exemplo, o número 1 (um).

```

//Estrutura de repetição: usando o while
//Exemplo5
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Numero = 1; //Inicializar a variável
    while(Numero)
    {
        printf("Digite um numero: ");
        scanf("%i", &Numero);
        printf("Quadrado do numero: %i eh %i\n\n", Numero, Numero * Numero);
    }
    printf("O numero %i foi digitado. FIM DO PROGRAMA!", Numero);
    return 0;
}
        
```





UNIDADE 6 – Estruturas de dados na Linguagem C

O QUE VAMOS ESTUDAR:

- **Estrutura de dados homogênea unidimensional – Vetor**



➤ Estrutura de dados homogênea unidimensional – Vetor

Vetores são Estruturas de Dados homogêneas formadas por um conjunto de variáveis de **mesmo tipo** acessadas pelo mesmo identificador. Os vetores podem ser declarados com qualquer um dos tipos de variáveis primitivas da linguagem C.

Exemplo: Vetores com **Termos (ou Elementos)** inteiros (**int**), reais (**float**) e caracteres (**char**) todos com cinco **posições**, respectivamente:

5	0	-1	3	20
---	---	----	---	----

0.3	-10.5	-8.0	33.7	12.8
-----	-------	------	------	------

@	A	d	&	#
---	---	---	---	---

Os vetores são acessados através do seu **Índice**. O **Índice** do vetor é determinado de **ZERO** até **TAMANHO DO VETOR – 1** na linguagem C por definição. Portanto, posição e índice são defasados de uma unidade.

Ou seja:

5	0	-1	3	20
POSIÇÃO 1	POSIÇÃO 2	POSIÇÃO 3	POSIÇÃO 4	POSIÇÃO 5
ÍNDICE 0	ÍNDICE 1	ÍNDICE 2	ÍNDICE 3	ÍNDICE 4

➤ Declaração de Vetores

--

--

 [TAMANHO];

TIPO DA VARIÁVEL NOME DA VARIÁVEL



Inicialização de Vetores

Vetores de Inteiros:

```
int Vetor [5] = { 1, 5, -1, 8, 0};
```

Vetores de Reais:

```
float Vetor [3] = { 1.3, 0.5, -3.7};
```

Vetores de Caracteres:

```
char Vetor [5] = { 'A', 'B', 'C', 'D', 'E'};
```

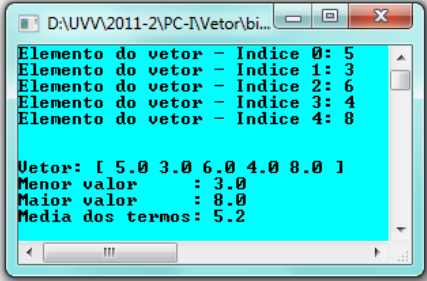
```
char Vetor [5] = { 65, 66, 'C', 68, 69}; // 'A', 'B', 'C', 'D', 'E'
```

```
char Vetor [11] = { 'L', 'i', 'n', 'g', 'u', 'a', 'g', 'e', 'm', ' ', 'C'};
```

PROGRAMA EXEMPLO: Escrever um programa em C que leia cinco valores reais quaisquer e os guarde em um vetor. Após isso, determine o **menor**, o **maior** e a **média** dos termos deste vetor.

```
#include <stdio.h>
#include <stdlib.h>
#define TamanhoVetor 5

int main()
{
    float Vetor[TamanhoVetor]; // Vetor de reais
    int i = 0; // Índice do vetor
    float Menor, Maior, Soma; // Menor, Maior e Soma dos termos
    while(i < TamanhoVetor)
    {
        printf("Elemento do vetor - Índice %i: ", i);
        scanf("%f", &Vetor[i]);
        i++; // Indexação próximo índice
    }
    /* Após a leitura dos termos do vetor */
    /* Inicializar as variáveis Menor, Maior e Soma com 1º Termo do vetor */
    Menor = Vetor[0];
    Maior = Vetor[0];
    Soma = Vetor[0];
    // A partir do 2º Termo do vetor
    for(i = 1; i < TamanhoVetor; i++)
    {
        if(Vetor[i] < Menor)
            Menor = Vetor[i];
        if(Vetor[i] > Maior)
            Maior = Vetor[i];
        Soma = Soma + Vetor[i];
    }
    /* Exibir o vetor e os resultados */
    printf("\n\nVetor: [ ");
    for(i = 0; i < TamanhoVetor; i++)
        printf("%.1f ", Vetor[i]);
    printf("]\n");
    printf("Menor valor : %.1f\n", Menor);
    printf("Maior valor : %.1f\n", Maior);
    printf("Media dos termos: %.1f\n", Soma / TamanhoVetor);
    return 0;
}
```



Elemento do vetor - Índice 0: 5
Elemento do vetor - Índice 1: 3
Elemento do vetor - Índice 2: 6
Elemento do vetor - Índice 3: 4
Elemento do vetor - Índice 4: 8

Vetor: [5.0 3.0 6.0 4.0 8.0]
Menor valor : 3.0
Maior valor : 8.0
Media dos termos: 5.2



UNIDADE 7 – Pesquisa e Ordenação de Dados

O QUE VAMOS ESTUDAR:

➤ Métodos de Pesquisa e Ordenação – Vetor

Neste capítulo vamos fazer uma introdução aos algoritmos de **Pesquisa e Ordenação de dados (P.O)** utilizando da estrutura de dados homogênea: Vetor. Esse estudo é uma Linha de Pesquisa que objetiva desenvolver algoritmos eficientes em relação a dois parâmetros específicos que são o Tempo e o Espaço de Memória utilizados neste processo.

Neste contexto, os algoritmos desenvolvidos são matematicamente pensados para que possam realizar o mínimo possível de operações sobre os dados do vetor e atingir o objetivo de forma eficiente. Taís características tornam esses algoritmos matematicamente organizados em **Métodos**. A seguir temos alguns exemplos de Métodos:

➤ MÉTODOS DE PESQUISA DE DADOS:

- ✓ Pesquisa Sequencial.
- ✓ Pesquisa Binária.
- ✓ Pesquisa por Interpolação.
- ✓ entre outros Métodos...

➤ MÉTODOS DE ORDENAÇÃO DE DADOS:

- ✓ Ordenação por Troca: Método Bolha.
- ✓ Ordenação por Inserção.
- ✓ Ordenação por Seleção.
- ✓ Método *Shellsort*.
- ✓ Método *Quicksort*.
- ✓ entre outros Métodos...

Vamos abordar neste capítulo somente os mais simples métodos de P.O de dados com a intenção de introduzir os conceitos básicos desta Linha de Pesquisa dentro do estudo da Lógica Estruturada de Programação de Computadores. A seguir vamos implementar os mais básicos métodos e abordar as suas principais características.



➤ Métodos de Ordenação por Troca: Método Bolha

Imagine a seguinte estrutura de dados homogênea Vetor com os elementos do tipo inteiro abaixo:

3	5	1	2	4
---	---	---	---	---

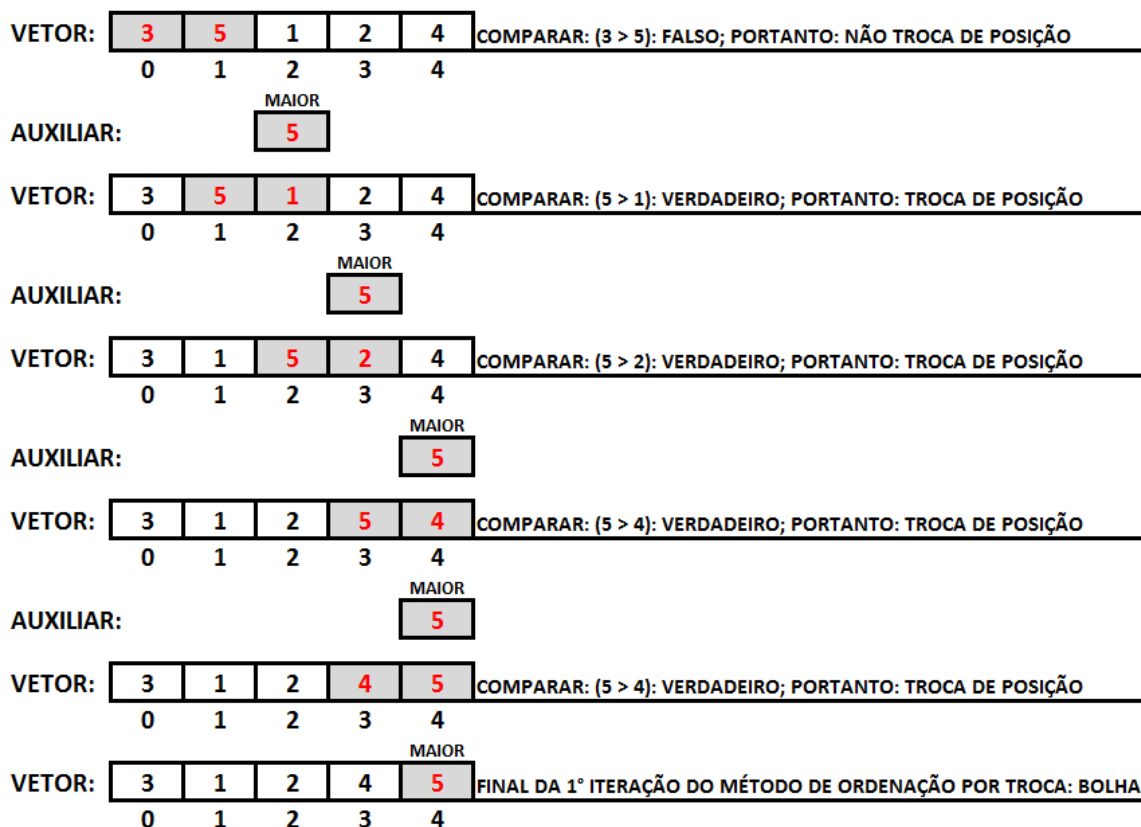
Como desenvolver um algoritmo que possa realizar a ordenação dos termos deste vetor em uma ordem ascendente de valores ?

Visando responder a esta pergunta de forma eficiente muitos algoritmos foram matematicamente organizados de tal modo que passaram a configurar um Método de resolução deste tipo de problema:

Problema de Ordenação de Dados.

ORDENAÇÃO POR TROCA DE DADOS:

Esse Método consiste em ordenar os dados utilizando por princípio a comparação entre os termos (de dois a dois) e a troca de lugar no vetor de forma que a cada iteração do algoritmo o Maior dos elementos do vetor (Ordem Ascendente de Valores) seja levado para a última posição da estrutura. A seguir um rastreamento, passo a passo, das ideias contidas em uma (1) iteração deste Método:





A Figura acima, demonstra a 1ª iteração do Método Bolha sobre os dados do vetor. Nesta iteração, os termos são comparados, dois a dois, a partir do primeiro termo do vetor. Se o resultado da comparação for falso, os termos permanecem na mesma posição do vetor. O índice é incrementado, e a comparação é feita novamente resultando em verdadeiro. Isso ocorrendo; é feita a troca de posição dos dois termos e a variável AUXILIAR memoriza temporariamente um dos termos antes de ser sobrescrito no vetor.

Esse processo é repetido até a penúltima posição do vetor; de modo que, no final da 1ª iteração o MAIOR termo do vetor ficará armazenado na última posição. O algoritmo a seguir demonstra o Método Bolha sobre os dados do vetor:

```
1  /* MÉTODO DE ORDENAÇÃO DE DADOS POR TROCA */
2  /* MÉTODO BOLHA */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #define TAMANHO 5
6
7  int main()
8  {
9      int Vetor[TAMANHO] = {3, 5, 1, 2, 4};
10     int Indice, Trocar, LimiteSuperior, Auxiliar;
11
12     /* Exibir o Vetor original */
13     printf("\t***MÉTODO BOLHA***\n\n");
14     printf("Vetor Original: [");
15     for (Indice = 0; Indice < TAMANHO; Indice++)
16         printf(" %i ", Vetor[Indice]);
17     printf("]\n\n");
18
19     /* MÉTODO BOLHA */
20     LimiteSuperior = TAMANHO - 1;           // Última posição do Vetor.
21     do
22     {
23         Trocar = 0;                        // Verifica se o Vetor já está Ordenado.
24         for (Indice = 0; Indice < LimiteSuperior; Indice++)
25         {
26             if (Vetor[Indice] > Vetor[Indice + 1])
27             {
28                 Auxiliar = Vetor[Indice];
29                 Vetor[Indice] = Vetor[Indice + 1];
30                 Vetor[Indice + 1] = Auxiliar;
31                 Trocar = 1;                // Verifica se existem termos fora de ordem.
32             }
33         }
34         LimiteSuperior--;                  // Reduz uma posição do vetor: última.
35     } while (Trocar);
36     /* FIM DO MÉTODO BOLHA */
37
38     /* Exibir o vetor ordenado: Ordem ascendente */
39     printf("Vetor Ordenado: [");
40     for (Indice = 0; Indice < TAMANHO; Indice++)
41         printf(" %i ", Vetor[Indice]);
42     printf("]\n\n");
43     return 0;
44 }
```

TELA

```
C:\Users\Alessandro\Desktop\Bolha\bin\Debug\Bolha.exe
***MÉTODO BOLHA***
Vetor Original: [ 3  5  1  2  4 ]
Vetor Ordenado: [ 1  2  3  4  5 ]
```

O Método Bolha possui algumas características que podem ser destacadas, conforme a implementação sugerida acima:

- ✓ A variável **Índice** sempre começa do zero, a cada nova Iteração;
- ✓ O incremento da variável **Índice** conta até o **Limite Superior** (excluindo) do vetor;
- ✓ Os termos são comparados, dois a dois, consecutivamente: **Índice** e **Índice + 1** conforme a Linha 26 do algoritmo;
- ✓ Se o resultado da comparação for Falso (0); não é necessário **Trocar** os termos de posição;
- ✓ Se o resultado da comparação for Verdadeiro (1); os termos são trocados de posição com a variável **Auxiliar** que memoriza temporariamente o valor a ser trocado de posição do vetor;
- ✓ No final da execução da estrutura de repetição **for**; o Maior termo do vetor é levado para a última posição do vetor;
- ✓ Este processo é repetido; sendo que, o segundo Maior termo é levado para o final do vetor e assim sucessivamente até o segundo termo do vetor;
- ✓ A cada iteração da estrutura de repetição **for**; podemos reduzir uma posição do **Limite Superior**: Linha 34, pois este termo já está na posição correta do vetor;
- ✓ Se a ordenação desejada for descendente, simplesmente devemos alterar a operação de comparação: Linha 26, da operação de Maior para Menor;
- ✓ A estrutura de repetição **do-while** é executada enquanto a variável **Trocar**: Linha 31 identifica, de forma lógica, que os termos continuam desorganizados no vetor;
- ✓ A estrutura de repetição **do-while** é finalizada quando a variável **Trocar**: Linha 23 permanecer Falsa, sinalizando que o vetor está ordenado. Se o vetor já estiver ordenado, o método é finalizado de forma eficiente já na primeira execução do programa;
- ✓ Ao longo do número de iterações do Método: **TAMANHO - 1** e a cada passo, a variável **Trocar**: Linha 23 consegue identificar se o vetor está ordenado, tornando este processo mais eficiente;
- ✓ Se a variável **Trocar** não alterar de valor na: Linha 31, significa que o vetor já está ordenado (ordem ascendente), podendo assim finalizar o Método.

As características descritas acima devem ser implementadas, de forma que, a eficiência do algoritmo possa representar um Método de ordenação de dados por troca de valores. Com isso; as ideias lógicas implementadas pelas variáveis do algoritmo: **Índice**, **Limite Superior**, **Auxiliar** e **Trocar** podem executar com eficiência a ordenação dos termos do vetor.



➤ Métodos de Pesquisa de Dados: Pesquisa Sequencial

Imagine a seguinte estrutura de dados homogênea: Vetor com os elementos do tipo inteiro abaixo:

3	5	1	2	4
---	---	---	---	---

Como desenvolver um algoritmo que possa realizar a pesquisa de um Valor escolhido pelo usuário?

Visando responder a esta pergunta de forma eficiente muitos algoritmos foram matematicamente organizados de tal modo que passaram a configurar um Método de resolução deste tipo de problema:

Problema de Pesquisa de Dados.

PESQUISA SEQUENCIAL DE DADOS:

Esse Método consiste em pesquisar um valor escolhido pelo usuário e verificar se o mesmo encontra-se armazenado no vetor. Para isso, o algoritmo percorre desde o primeiro termo, com iterações que vão termo por termo, até encontrá-lo ou até que se chegue no último termo armazenado no vetor. Com isso, a resposta de saída do Método de Pesquisa Sequencial é:

- **Valor encontrado:** Uma variável de saída indica a localização do **índice** onde está o valor desejado. Neste caso, o Índice de Saída do Método é um valor inteiro entre **[0, TAMANHO DO VETOR - 1]**;
- **Valor NÃO encontrado:** A mesma variável de saída indica uma localização inexistente no vetor, ou seja, um valor de **índice** que NÃO existe, por exemplo: **Índice de Saída = -1**.

A seguir um rastreamento, passo a passo, das ideias contidas neste Método:

Suponha que o Valor Procurado seja: 2

Índice De Saída = -1 // Inicializa a variável com um índice inexistente, ou seja, Valor NÃO encontrado.

VETOR:	3	5	1	2	4	NÃO ENCONTRADO
	0	1	2	3	4	
VETOR:	3	5	1	2	4	NÃO ENCONTRADO
	0	1	2	3	4	
VETOR:	3	5	1	2	4	NÃO ENCONTRADO
	0	1	2	3	4	
VETOR:	3	5	1	2	4	ENCONTRADO
	0	1	2	3	4	



A Figura acima demonstra, passo a passo, o rastreamento do Método na pesquisa de um valor escolhido pelo usuário. É feita a comparação de cada termo, sequencialmente começando pelo primeiro termo (Índice = 0), com o valor escolhido pelo usuário, até que se encontre o valor desejado ou se chegue no final do vetor, significando neste caso, que o valor procurado NÃO consta no vetor.

No exemplo onde o valor procurado é Valor = 2, observe que o método é encerrado antes do final do vetor, uma vez que, o valor já foi encontrado neste índice. Não é necessário continuar a pesquisa, até o final do vetor; tornando assim, o Método mais eficiente em relação ao objetivo. O algoritmo a seguir demonstra o Método de Pesquisa Sequencial sobre os dados do vetor:

```
1  /* PESQUISA SEQUENCIAL */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #define TAMANHO 5
5
6  int main()
7  {
8      int Vetor[TAMANHO] = {3, 5, 1, 2, 4};
9      int Indice = 0;
10     int Valor;
11     int IndiceDeSaida = -1;
12     printf("\t***METODO PESQUISA SEQUENCIAL***\n\n");
13     printf("Digite o valor procurado: "); scanf("%i", &Valor);
14     while(Indice < TAMANHO && IndiceDeSaida == -1)
15     {
16         if(Vetor[Indice] == Valor)
17             IndiceDeSaida = Indice;
18         else
19             Indice++;
20     }
21     if (IndiceDeSaida == -1)
22         printf("\nO valor: %i NAO foi encontrado no Vetor.\n\n", Valor);
23     else
24         printf("\nO valor: %i encontrado no Indice: %i do Vetor.\n\n", Vetor[IndiceDeSaida], IndiceDeSaida);
25     return 0;
26 }
27
```

TELA

```
C:\Users\Alessandro\Desktop\PesquisaSequencial\bin\Debug\PesquisaSequencial.exe
***METODO PESQUISA SEQUENCIAL***
Digite o valor procurado: 2
0 valor: 2 encontrado no Indice: 3 do Vetor.
```

O **Método de Pesquisa Sequencial** de dados possui algumas características que podem ser destacadas, conforme a implementação sugerida acima:

- ✓ A variável **Índice** (**Linha 9**) sempre começa do zero, como é de característica da Linguagem C de programação;
- ✓ O valor escolhido pelo usuário é armazenado justamente na variável **Valor**, conforme a **Linha 13** do algoritmo;
- ✓ A resposta do Método de Pesquisa é obtido na variável **ÍndiceDeSaída**, conforme a **Linha 11** do algoritmo. Neste caso, a variável inicia com um índice inválido, ou seja, negativo, partindo do pré-suposto que o valor procurado ainda NÃO foi encontrado no vetor;



- ✓ A estrutura de repetição **while** é executada enquanto a variável **Índice** não chegar no final do vetor: **TAMANHO** e a variável **ÍndiceDeSaída** estiver com índice = -1, sinalizando que o valor NÃO foi encontrado ainda no vetor;
- ✓ Quando o teste de comparação entre o termo do **Vetor[Índice]** e o **Valor** for verdadeiro (1), na **Linha 16**, a variável **ÍndiceDeSaída** (**Linha 17**) recebe justamente o índice onde foi encontrado o valor escolhido pelo usuário.

As características descritas acima devem ser implementadas, de forma que, a eficiência do algoritmo possa representar um Método de Pesquisa de dados Sequencial. Com isso; as ideias lógicas implementadas pelas variáveis do algoritmo: **Índice**, **Valor** e **ÍndiceDeSaída** podem executar com eficiência a pesquisa dos termos do vetor.

PESQUISA BINÁRIA DE DADOS:

Esse Método consiste em pesquisar um valor escolhido pelo usuário e verificar se o mesmo encontra-se armazenado no vetor. Para isso, o algoritmo percorre o vetor de forma bem diferente do Método da Pesquisa Sequencial. Neste Método, o índice **Meio** é percorrido de acordo com os limites inferior e superior do vetor, conforme a fórmula:

$$Meio = \frac{(Limite_{Inferior} + Limite_{Superior})}{2}$$

O Método da Pesquisa Binária dos dados parte do princípio que o valor escolhido pelo usuário está sempre localizado na **metade do vetor**, por esse motivo, o cálculo da variável **Meio** como visto acima. A variável índice **Meio** possibilita verificar se o valor desejado encontra-se na metade do vetor. Caso afirmativo, a saída do Método é semelhante ao Método de Pesquisa Sequencial, sendo que:

- **Valor encontrado:** Uma variável de saída indica a localização do **índice** onde está o valor desejado. Neste caso, o Índice de Saída do Método é um valor inteiro entre **[0, TAMANHO DO VETOR - 1]**;
- **Valor NÃO encontrado:** A mesma variável de saída indica uma localização inexistente no vetor, ou seja, um valor de **índice** que NÃO existe, por exemplo: **Índice de Saída = -1**.

Caso essa comparação for falsa (0), ou seja, valor NÃO encontrado no índice **Meio**, a pesquisa é feita novamente atualizando o cálculo da variável índice **Meio** com o valor ajustado do limite do vetor, sendo que, este ajuste depende da comparação com o valor escolhido pelo usuário, ou seja:

- Se o **Valor** procurado for **Maior** que o valor localizado na variável índice **Meio**, o $Limite_{Inferior}$ do vetor deve ser atualizado para o seguinte índice: $Limite_{Inferior} = Meio + 1$. Com isso, todos os termos



antes da variável índice **Meio** podem ser descartados, pois são menores que o Valor desejado pelo usuário;

- Se o **Valor** procurado for **Menor** que o valor localizado na variável índice **Meio**, o $Limite_{Superior}$ do vetor deve ser atualizado para o seguinte índice: $Limite_{Superior} = Meio - 1$. Com isso, todos os termos depois da variável índice **Meio** podem ser descartados, pois são maiores que o Valor desejado pelo usuário;

Para que estes cálculos acima descritos possam funcionar corretamente para o Método de Pesquisa Binária dos Dados, uma característica básica para o uso do algoritmo deve ser observada na sua execução. Essa característica consiste no fato de que; para utilizar o Método de Pesquisa Binária dos dados, os valores devem obrigatoriamente estar armazenados em **ordem ascendente** no vetor. O algoritmo a seguir demonstra o **Método de Pesquisa Binária** sobre os dados do vetor em **ordem ascendente (Linha: 8)**:

TELA

```
1  /* PESQUISA BINÁRIA */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #define TAMANHO 15
5  int main()
6  {
7      /* CONDIÇÃO: O Vetor deve estar ordenado para a Pesquisa Binária */
8      int Vetor[TAMANHO] = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
9      int Valor;
10     int IndiceDeSaida = -1;
11     int Li = 0;
12     int Ls = TAMANHO - 1;
13     int Meio;
14     printf("\t***METODO PESQUISA BINARIA***\n\n");
15     printf("Digite o valor procurado: "); scanf("%i", &Valor);
16     while(Li <= Ls && IndiceDeSaida == -1)
17     {
18         Meio = (Li + Ls) / 2;
19         if(Vetor[Meio] == Valor)
20             IndiceDeSaida = Meio;
21         else if(Vetor[Meio] > Valor)
22             Ls = Meio - 1;
23         else
24             Li = Meio + 1;
25     }
26     if (IndiceDeSaida == -1)
27         printf("\nValor: %i NAO foi encontrado.\n\n", Valor);
28     else
29         printf("\nValor: %i encontrado no Indice: %i.\n\n", Vetor[IndiceDeSaida], IndiceDeSaida);
30     return 0;
31 }
```

C:\Users\Alessandro\Desktop\PesquisaBinaria\bin\Debug\PesquisaBinaria.exe
METODO PESQUISA BINARIA
Digite o valor procurado: 14
Valor: 14 encontrado no Indice: 9.

A seguir um rastreamento, passo a passo, das ideias contidas neste Método:

Suponha que o Valor Procurado seja: 14

Índice De Saída = -1 // Inicializa a variável com um índice inexistente, ou seja, Valor NÃO encontrado.



Li:	0															Li		Meio				Ls	
Meio:	$(Li + Ls) / 2$	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19							
Ls:	Tam - 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14							

Li:	Meio + 1														Li		Meio		Ls										
Meio:	(Li + Ls) / 2														5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ls:	Tam - 1														0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Li: Meio + 1															Li	Meio	Ls
Meio: (Li + Ls) / 2	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19		
Ls: Meio - 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14		

O **Método de Pesquisa Binária** de dados possui algumas características que podem ser destacadas, conforme a implementação sugerida acima:

- ✓ A variável **Limite Inferior: Li (Linha 11)** sempre começa do **ZERO**, como é de característica da Linguagem C de programação;
- ✓ A variável **Limite Superior: Ls (Linha 12)** deve começa em **TAMANHO VETOR - 1**, como é de característica da Linguagem C de programação;
- ✓ O valor escolhido pelo usuário é armazenado justamente na variável **Valor**, conforme a **Linha 9** do algoritmo;
- ✓ A resposta do Método de Pesquisa é obtido na variável **ÍndiceDeSaída**, conforme a **Linha 10** do algoritmo. Neste caso, a variável inicia com um índice inválido, ou seja, negativo, partindo do pré-suposto que o valor procurado ainda **NÃO** foi encontrado no vetor;
- ✓ A estrutura de repetição **while** é executada enquanto o limite **Li** **NÃO** ultrapassar o limite **Ls** (ou vice-versa) e a variável **ÍndiceDeSaída** estiver com índice = -1, sinalizando que o valor **NÃO** foi encontrado ainda no vetor;
- ✓ Se o teste da **Linha 19** for verdadeiro (1), a variável **ÍndiceDeSaída** recebe o cálculo da variável índice **Meio**, conforme foi executado na **Linha 18** do algoritmo;
- ✓ Caso contrário, se o **Valor** procurado for **Menor** que o valor localizado na variável índice **Meio**, o $Limite_{Superior} = Meio - 1$, conforme a **Linha 22** do algoritmo;
- ✓ Caso contrário, se o **Valor** procurado for **Maior** que o valor localizado na variável índice **Meio**, o $Limite_{Inferior} = Meio + 1$, conforme a **Linha 24** do algoritmo.

As características descritas acima devem ser implementadas, de forma que, a eficiência do algoritmo possa representar um Método de Pesquisa Binária de dados. Com isso; as ideias lógicas implementadas pelas

variáveis do algoritmo: **Limite Inferior**, **Limite Superior**, **Meio**, **Valor** e **ÍndiceDeSaída** podem executar com eficiência a pesquisa dos termos do vetor.

➤ REFERÊNCIA BIBLIOGRÁFICA:

BÁSICA:

FORBELLONE, André Luiz Villar; **EBERSPACHER**, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2006.

DAMAS, Luís. Linguagem C. 10. ed. Rio de Janeiro: LTC, 2011 e edições anteriores.

MIZRAHI, Victorine Viviane. Treinamento em linguagem C: módulo 1. São Paulo: Pearson Makron Books, 2005 e edições anteriores.

COMPLEMENTAR:

CELES, Waldemar; **CERQUEIRA**, Renato; **RANGEL**, José Lucas. Introdução a estruturas de dados: com técnicas de programação em C. Rio de Janeiro: Elsevier: 2004.

SCHILDT, Herbert. C completo e total. 3. ed. rev. e atual. São Paulo: Makron Books, c1997.

MANZANO, José Augusto Navarro Garcia; **OLIVEIRA**, Jayr Figueiredo de. Algoritmos: lógica para desenvolvimento de programação. 7. ed. São Paulo: Érica, 1999.

MANZANO, José Augusto Navarro Garcia; **OLIVEIRA**, Jayr Figueiredo de. Estudo dirigido algoritmos. 3. ed. São Paulo: Érica, 1998.

SENNE, Edson Luiz França. Primeiro curso de programação em C. 2. ed. Florianópolis: Visual Books, 2006.

JAMSA, Kris; **KLANDER**, Lars. Programando em C/C++: a Bíblia. São Paulo: Makron Books, 1999.