Anotações sobre o curso: Curso completo de Lógica de Programação e Algoritmos.

Três definições de lógica:

1 – Maneira rigorosa de racioncinar (correção do pensamento)

2 – O encadeamentos coerente de alguma coisa que obedece a certas convenções e regras (ligação de ideias)

3 – Organização de planejamento das instruções, em um algoritmo, a fim de viabilizar a implantação de um programa.

Exemplo – Escrever é materializar um pensamento. A linguagem possue uma lógica prória em uma ordem (1 – 2 – 3).

Lógica é a capacidade de colocar os algoritmos em uma ordem para que eles façam sentido e possam ser executados.

Quatro paradigmas de programação:

1. Imperativa ou Procedural (Ligado ao tempo verbal imperativo; Sequências de comandos para o computador executar). Tipos de linguagem que usam esse paradigma: Ada, Basic, Fortan, Pascal, Algol, C e Cobol
2. Funcional (Enfatiza a aplicação de funções; Trata a computação como uma avaliação de funções matematicas. Evita estado ou dados mutáveis. Linguagem que usam: Lisp, Scheme, ML, Miranda e Haskel
3. Lógica (Utilização de sentenças lógicas como linguagem para programação. Elabora deduções com base em lista de premissas). Linguagens que usam: Popler, Conniver, QLISP, Planner, Mercurry, OZ e Frill.
4. Orientada a Objetos (Interação entre unidades de softwares, chamadas “Objetos”. Trocando mensagens entre si. Classes = Objetos + Método (Comportamento e estado de uma classe) + Atributo (Comportamento e estado de uma classe)) Cada classe possui suas responsábilidades e sempre há uma classe global que serve de referência para outras classes. Linguagens: Python, Ruby, C#, C++ e Java (Sucedeu a linguagem imperativa).

Programar = Encadear uma série de **INSTRUÇÕES** para que se possa chegar à solução de um problema.

**INSTRUÇÕES** – Ordens dadas ao computador para que ele execute. Somente uma instrução isolada, não permite que o computador realize um processo completo. É necessário que se faça uma gama de instruções de forma lógica (organizada) descrevendo como deve ser o processo.

Conjunto de instruções para imprimir um arquivo:

1. Abra o arquivo “Becode.txt”
2. Verifique se a impressora está ligada e desocupada.
3. Envie o arquivo “Becode.txt” para a impressora imprimir.
4. Feche o arquivo e avise que a impressão está finalizada.

Todas as ações que são ditas ao computador para ele executar, são chamadas de INSTRUÇÕES.

Progama = Conjunto de ALGORITMOS organizados de uma forma LÓGICA, em uma LINGUAGEM específica para que o computador possa interpreta-la.

Algoritmos 🡪 Uma sequência de INSTRUÇÕES para atingir um determinado objetivo. (Manuais eletrônicos; Receitas culinárias ou Regras de expressões numéricas)

Processamento de dados de um Algoritmo:

1. Entrada (Informação é lida ou inserida)
2. Processamento
3. Saída (Valores retornados após o processamento das informações)

Exemplo – Google tradutor.

Inicio (Problema) – Necessidade de traduzir uma frase

Entrada – Uma frase qualquer em uma lingua estrangeira

Processamento – Google procura em um dicionário cada palavra para o idioma desejado.

Saída – A frase traduzida de acordo com a lógica proposta.

É necessario formalizar um algorítimo em uma linguagem (pseudocódigo).

Pseudocódigo = Forma genérica de escrever um algorítimo. Escrever uma cadeira de instrução na linguagem nativa, sem a necessidade de aplica à uma sintaxe de programação. Não há como rodar em um programa, mas formaliza o algoritmo em uma linguagem.

Pseudocódigo 🡪 Portugol (Serve para dar uma noção da construção e da lógica do algorítimo e ser depois transcrito para uma linguagem de programação).

Dicas usando o Portugol 🡪 Seja objetivo; Se faça entender; Frases curtas; Um só verbo por frase; Evitar duplo sentido.

Código Binário = Sinais elétricos que o computador interpreta.

0 = Desligado e 1 = Ligado

0 = Bit

1 = Bit

00000000 = Byte (8)

11111111 = Byte (8)

Toda linguagem de programação é codificada de forma binária, para que desta forma, o computador interprete.

Algebra Booleana 🡪 Permite fazer operações lógicas e aritiméticas usando apenas dois digitos ou estados (Sim ou Não; Verdadeiro ou Falso; Tudo ou Nada; 1 ou 0)

Códio ASCII (American Standert Code For Information Interchange) 🡪 Representar textos em computadores. Grande parte das codificações de caracteres herdaram sua base.

A linguagem de programação é mais rigida que nossa linguagem comum. É baseada em Regras, Sitaxes e Palavras Reservadas (Só podem ser utilizadas para expressar uma funcionalidade específica).

Temos que usar palavras chaves para definir o inicio, o fim e as condições em uma linguagem de programação.

Algoritmos devem ter contexto, local e objetivo definido. Isto deve ser realizado na LINGUAGEM DE PROGRAMAÇÃO.

Linguagens em evidências e onde são usadas!

1. Java 🡪 Mais usada no mundo; Plataforma de desenvolvimento; Utilizada na porra toda; Programação orientada a objetos.
2. PHP 🡪 Script open source; Usada em dev de Web; Embutida no HTML; Suporte aos bancos de dados; Pode usar em desktop; Programação procedural ou orientada a objetos.
3. C++ 🡪 Herdeiro do C; Poderosa, flexivel e abrangente; Utilizada em diversos tipos de hardware; Boa na área de automação; Programação orientada a objetos.

FIM DO PRIMEIRO MÓDULO!

**Variáveis** 🡪 Pequenas “caixas” que armazenam dados que podem mudar. Exemplo:

Dados númericos 🡪 2, 6, 10, 16

Sequência de caracteres 🡪 Nome/ frase [O nome um dia poderá mudar, portanto se armazena em uma variável].

No exemplo do Google Translate mais acima. A caixa onde está contida a frase é uma variável, porém o que for escrito dentro dela é uma **Atribuição**.

Toda variáveis precisa de um nome! O nome de uma variáveis é conhecido como **INDENTIFICADOR**

Não se pode dar qualquer identificador para uma variável, pois cada linguagem possue sua regra específica de nomear.

Contudo normalmente seguem as seguintes regras:

1. Iniciar com caractere alfabético
2. Pode ser seguido por mais caracteres (alfabeticos ou números)
3. Não pode ter caracteres especiais, espaços em braco e palavras reservadas

(Cada linguagem possuem suas palavras reservadas que não podem ser usadas como IDENTIFICADORES).

(Tem na pasta algumas figuras de palavras reservadas)

OBS 🡪 As linguagens Java, PHP e C++ tratam letras maiúsculas e minusculas de formas distintas!

Exemplo: A palavra Target (Mesclado); target (Minusculo) e TARGET (Maiúsculo) são tratadas de forma completamente diferente.

IDENTIFICADORES VÁLIDOS: idade, nome, salario (sem acento), resposta, TELEFONE, NOTA1

IDENTIFICADORES INVÁLIOS: aniversário (acento), endereço (ç) algoritmo (reservado), busca tabela (espaço), bairro/cidade (/), PARA (reservado).

PS 🡪 Reservado no Portugol (Pseudolinguagem usada no curso).

PS2 🡪 (`´^ç<>:.,/?) Acentos

As variáveis são armazenadas em uma memória volátil (quando o computador é desligado, essas memóridas deixam de existir!) chamada de RAM (Random Access Memory).

Depois de IDENTIFICAR uma variáveis, está deverá ser **DECLARADA**. Dependendo da declaração, o sistema vai tratar de forma diferente os dados atribuidos a variável.

Exemplo:

Memoria RAM = Armário de Gavetas

Variável = GAVETAS (precisamos IDENTIFICAR)

Dados de uma Variável = OBJETOS (armazena somete um Tipo de Objeto)

Cada gaveta só pode armazenar UM TIPO de objeto.

Por isso devemos DECLARAR uma variável, visto que ela armazena um dado de cada vez, sendo que, esse dado deve ser SEMPRE DO MESMO TIPO.

A DECLARAÇÃO só é obrigatória em casos de Linguagem “Fortemente Tipadas” (C#, Java, C++, Fortran)

Caso seja “Fracamente Tipadas”, não há obrigatoriedade pois são mais dinâmicas e a execução do programa pode alterar os dados da variável. Linguagens: (Javascript, Ruby e Python; PHP você pode escolher se quer Fraco ou Forte).

Tipos de IDENTIFICADORES para DECLARAR uma VARIÁVEL:

1. Inteiro
2. Real
3. Lógico
4. Caractere

Tipo Inteiro 🡪 Específico para números

Pode ser usada para cálculos

Armazena somente números inteiros (sem casas decimais)

Podem ser negativos ou positivos

Tipo Real ou Numérico🡪 Específico para números

Pode ser usada para cálculos

Armazena números Racionais e Irracionais

Podem ser negativos ou positivos

(Decimais, Frações, Pi [3,1415926...] ou raiz de 2 [1,414213562...] Ou seja, números que não podem ser representados por uma divisão de números inteiros).

Tipo Caractere ou Literal 🡪 Armazenamento de todo tipo de dados (Caracteres ou números

Não pode ser usada para cálculos

Exemplo 🡪 “Para mais detalhes, veja página 245”

Tipo Lógico 🡪 Específico para armazenar dois valores: Verdadeiro ou Falso.

Representa variáveis que apresentam respostas

Não pode ser usada para cálculos.

Exemplo 🡪 temcarro = verdadeiro; desempregado = falso.

Exemplo de variáveis com seus tipos declarados:

VARIÁVEL 🡪 TIPO DECLARADO

idade 🡪 inteiro

nome 🡪 caractere

salario 🡪 real

resposta 🡪 logico

Variáveis com o mesmo tipo devem ser declaradas na mesma linha:

VARIAVEL1, VARIAVEL2: TIPO DECLARADO.

idade, numerofilhos: inteiro

ATRIBUIÇÃO 🡪 Dado ATRIBUÍDO a uma variável

No exemplo do armário (ver acima), os objetos dentro das gavetas são os dados.

Na maioria das linguagens a ATRIBUIÇÃO é representada pelo sinal de igual ( = )

VARIAVEL = “ATRIBUICAO”

No Portugol é representado por uma Seta ( <- Menor seguido de hifen)

VARIAVEL <- “ATRIBUICAO”

Exemplo em Algoritmo:

VARIÁVEL: TIPO DECLARADO

a: logico [verdadeiro ou falso]

b: logico

x: inteiro [operações matemáticas do tipo N]

VARIÁVEL <- ATRIBUIÇÃO

a <- verdadeiro

b <- 5 = 3 // será FALSO, pois 5 é diferente de 3

x <- 8 + 3

No caso da atribuição de valores por lógica ou aritimetica, as expressões devem ser resolvidas primeiro para depois o valor ser atribuido. No caso de x <- 8 + 3 o valor que deve ser atribuido é “11”.

Em b <- 5 = 3 temos uma operação lógica, no caso dessa expressão o valor de deve ser atribuido é “Falso”, pois sabemos que 5 ≠ 3.

Esse “diferente” na expressão b do algoritmo passado é um Comentário!

O comentário não será lido pelo computador.

Comentário 🡪 Trechos de caracteres (frases), escritos dentro do algoritmo. Servem para aumentar a clareza do código.

Comentários devem ser: Objetivos e esclarecedores; Sem textos e frases longas; Pode ser em qualquer lugar do código conforma a necessidade.

Em Java, PHP e C++ o operador de cometário é representado por:

Comentário em linha //

Comentário em trecho (multiplas linhas) /\* no inicio do trecho e no final \*/

BULINDO NO VISUALG

Algoritmo = Organizar 3 gavetas de um armário.

Passo a passo:

1. Identificar as gavetas;
2. Abrir as gavetas;
3. Informar quais objetos vamos inserir nas gavetas;
4. Identificar a quantidade de objetos;
5. Inserir objetos em cada gaveta;
6. Verificar se os objetos foram colocados nas gavetas corretas e se suas quantidades estão certas;
7. Finalizar.

Algoritmo = organizar\_gavetas

Variáveis = gaveta1; gaveta2; gaveta3 (g1, g2, g3)

Objetos = objeto1; objeto2; objeto3 (obj1, obj2, obj3)

Declaração = g1, g2, g3: caracter

obj1, obj2, obj3: inteiro

Atribuição =

Manual Por interface:

g1 <- Utilizando commandos de entrada e saída de dados

g2 <-

g3 <-

Comando de entrada de dados

No VisuAlg o comando é **LEIA** (Lê valores digitados no teclado e armazena na RAM)

Sintaxe: LEIA (variavel1, variavel2, variavel3...)

Comando de saida de dados

ESCREVA (Imprime dados na tela do computador, sendo que estes dados podem ou não estar armazenados em variáveis. Pode ser Texto, Números, Variáveis, Conjunto de dados + Variáveis).

Sintaxe: ESCREVA (variavel1, variavel2, variavel3...)

Comando de saida de dados2

ESCREVAL (Parece com o ESCREVA, mas quebra o resultado para a próxima linha).

Combinação 🡪 Olhar arquivo organizar\_gavetas\_combinação

Concatenação 🡪 UNIR as variaveis de gavetas (g1,g2,g3) com suas respectivas variáveis de quantidade (obj1,obj2,obj3).

Concatenação é normalmente usado para UNIR o nome e o sobrenome de uma pessoa, formando um nome completo.

A união na concatenação é feito manualmente (atribuição)!

Ex: nomecompleto <- nome + “ “ + sobrenome

Ver exemplo de Concatenação em organizar\_gavetas\_completo.

Operadores aritméticos 🡪 Ver PNG na pasta

Operandos 🡪 Números

**Soma +** 🡪 Somar dois operandos 1+1 = 2

**Subtração -** 🡪 Diminuir dois operandos 2-1 = 1

**Multiplicação \*** 🡪 Multiplicar dois operandos (2\*4) = 8

**Dividir /** 🡪Dividir dois operandos (10/2) = 5

**Módulo % ou mod** 🡪 Apresenta o resto da divisão de um operando por outro. Caso zero, a divisão é exata. 13 mod 2 = 1 (Divisão irracional); 150 mod 5 = 0 (Divisão racional).

**Potenciação ^** 🡪 Eleva à potencia (10^2) = 100

[10²]

Expressões Aritméticas 🡪 VisuAlg (Expressoes\_aritmeticas)

Funções 🡪 Normalmente já está implementada na linguagem de programação.

As funções servem para dar mais flexibilidade e tempo.

Ex:

Log (Expressão – Retorna o logaritmo na base 10 do valor representado)

RaizQ (Expressão – Retorna a raiz quadrada do valor representado)

Quad (Expressão – Retorna o quadrado do valor)

Sen (Expressão – Retorna o Seno do angulo)

Tan (Expressão – Retorna a Tangente do angulo)

Exp (Retorna o valor Base elevado ao Expoente)

Pode ser usado em qualquer lugar que uma variável. Porém, não antes de um comando de atribuição. Pois a função retorna um valor e não recebe.

Sintaxe 🡪 Ver VisuALG (Raiz\_quadrada)

Operadores relacionais 🡪 Olhar JPG na pasta

Símbolo utilizado para representar operações com relações entre dois objetos.

O resultado sempre será do tipo lógico (verdadeiro ou falso).

**Operador =** 🡪 Relação entre dois objetos e verifica se são iguais

**Operador <>** 🡪 Relação entre dois objetos e verifica se são diferentes

**Operador >** 🡪 Relação entre dois objetos e verifica se o primeiro objeto é maior que o segundo

**Operador <** 🡪 Relação entre dois objetos e verifica se o primeiro objeto é menor que o segundo

**Operador >=** 🡪 Relação entre dois objetos e verifica se o primeiro objeto é maior ou igual que o segundo

**Operador <=** 🡪 Relação entre dois objetos e verifica se o primeiro objeto é menor ou igual que o segundo

Operadores lógicos 🡪 Utilizados para conectar expressões

E (And)

Ou (Or)

Não (Not)

E (And) [Só resulta em verdadeiro caso todas a proposições sejam VERDADEIRAS]

|  |  |  |
| --- | --- | --- |
| Condição A | Condição B | Resultado |
| Falso | Falso | Falso |
| Verdadeiro | Falso | Falso |
| Falso | Verdadeiro | Falso |
| Verdadeiro | Verdadeiro | Verdadeiro |

Ou (Or) [Só resulta em falso caso todas a proposições sejam FALSAS]

|  |  |  |
| --- | --- | --- |
| Condição A | Condição B | Resultado |
| Falso | Falso | Falso |
| Verdadeiro | Falso | Verdadeiro |
| Falso | Verdadeiro | Verdadeiro |
| Verdadeiro | Verdadeiro | Verdadeiro |

Não (Not) [Inverte o valor lógico de uma variável ou expressão]

|  |  |
| --- | --- |
| Entrada | Saída |
| Falso | Verdadeiro |
| Verdadeiro | Falso |

Prioridade dos operadores lógicos 🡪

1 – Não (Not)

2 – E (And)

3 – Ou (Or)

Caso quisermos mudar essa hierarquia, é necessário usar os parenteses

Prioridade em expressões com vários tipos de operadores 🡪

1 – Expressões com ( ) mais internos

2 – Funções aritméticas

3 – Operadores aritiméticos

4 – Operadores relacionais

5 – Operadores lógicos

Estruturas de Tomada de Decisão 🡪 Operadores condicionais, possuem a função de auxiliar nas tomadas de decisões.

Ex: Login de um usuário 🡪 Caso esteja certo entra, se estiver errado aparece uma mensagem de erro.

Blocos lógicos 🡪 Trechos de algoritmos, onde temos uma palavra que indica o INICIO e outra que indica o FIM. Esse trecho é SEQUENCIAL e o computador vai interpretar na mesma ordem que foi escrito.

Ex:

|  |
| --- |
| Algoritmo “bloco”  Var  a:real  **inicio** 🡪 (marca o inicio do algoritmo)  //instruções (bloco lógico  **finalalgoritmo** 🡪 (marca o fim) |

Sendo assim, a estrutura de tomada de decisão é a forma como instruções são organizadas para tratar determinada decisão. Lembrando que só existem duas respostas para o teste lógico: V ou F.

São os seguintes operadores lógicos condicionais:

1. Condicionais simples
2. Condicionais compostas
3. Condicionais encadeadas

3.1- Homogêneas

3.2- Heterogêneas

Condicional Simples 🡪 Define o caminho a ser seguido a ser seguido dependendo da respostado teste lógico (V ou F). Quando utilizada, ela teste se o resultado é V ou F e vai executar instruções de acordo com o restultado.

Representada pela palavra Se (If)

Sintaxe: SE (condição) ENTAO

//instruções

FIMSE

As instruções são executadas apenas se o resultado do teste lógico por verdadeiro. Caso o teste dê falso, o programa não faz nada.

A condicional IF se chama simples, pois só executa se o resultado for verdadeiro.

VisuALG 🡪 Contexto (Emprego.alg)

Um gerente de RH recebe muitos currículos, então ele solicita à equipe de TI a criação de um programa para auxiliar no recrutamento. Para isso cria-se uma condicional!

Salário <- pretenção salarial <= 2000,00

Condicional composta 🡪 Conjunto de instrução para quando o teste lógico for verdadeiro e outro para quando for falso.

Sintaxe: SE (condição) ENTAO

//instruções

SENAO

//instruções

FIMSE

Condição encadeada 🡪 Sequência encadeada de testes lógicos, executados conforme o resultado de cada condição.

Normalmente, utilizada quando necessário verificar DUAS (OU MAIS) condições sucessivas.

Ex:

Condição1 IF True 🡪 Condição 2

IF False 🡪 Conjunto de instruções C

Condição2 IF True 🡪 Conjunto de instruções A

IF False 🡪 Conjunto de instruções B

No exemplo do Emprego.alg, acrescentamos mais uma condição:

1. Pretensaosalarial <- salario <= 2000
2. Horastrabalhadas <- horas disponiveis >= 7

Condição encadeada heterogênea 🡪 Quando não for possível idenficiar um padrão lógico nas condicionais, chamamos de condicionais encadeadas heterogêneas.

Não existe um padrão organizacional no algoritmo. Estão espalhados dentro do algoritmo.

Ex:

|  |
| --- |
| Se (condicao1) entao  Se (condicao2) entao  Instrucao n  Fimse  Senao  Se (condicao3) entao  Instrucao n  Senao  Se (condicao4) entao  Se (condicao5) entao  Instrucao n  Fimse  Fimse  Fimse  Fimse |

A condição 1 recebe as demais condições

A condição 2 e 3 estão no mesmo nível de identação

Dentro da condição 3 está a 4

Por fim, dentro da condição 4 tem a 5

Condição encadeada homogêneas 🡪 Diferente das heterogêneas, estas seguem um padrão lógico.

Duas formas de estruturação:

1. Se (condicao) entao

//instrucao

Se (condicao) entao

//instrucao

Fimse

Fimse

1. Se (condicao) entao

//instrucao

Senao

//instrucao

Se (condicao) entao

//instrucao

Senao

//instrucao

Fimse

Fimse

Na primeira forma, há um seguimento de padrão lógico. Há ausência da instrução SENAO. Por fim, o computador só executara as instruções SE TODAS AS CONDIÇÕES forem VERDADEIRAS.

Ex – Se (condicao1) entao

Se (condicao2) entao

Se (condicao3) entao

Se (condicao4) entao

**COMANDO1**

Fimse

Fimse

Fimse

Fimse

A segunda forma é normalmente usada quando desejamos checar o valor de uma variável, para então executar uma ação.

Vamos supor que uma variável X pode assumir 4 valores (v1, v2, v3 e v4).

Ex – Se (x = v1) entao

(comando1)

Senao

Se (x = v2) entao

(comando2)

Senao

Se (x = v3) entao

(comando3)

Senao

Se (x = v4) entao

(comando4)

Fimse

Fimse

Fimse

Fimse

Essa é uma forma otimizada, pois vai utilizar o poder computacional necessário e parar no momento em que o teste for verdadeiro. Se, por exemplo, x = v2, então o teste pararia por ai sem precisar calcular v3 e v4.

Esta segunda forma de condicional é conhecida como (Se senao se)

Estrutura de Repetição (Loops ou Laços de Repetições) 🡪 Repetir um trecho de um programa, código ou pedaço de algorímo. Ajuda a realizar uma tarefa várias vezes mantendo o controle.

Usar sempre que necessário, repetir instruções N vezes ou até que seja informado “**finalizar a execução**”.

Tipos: ENQUANTO (While); REPITA (Repeat) e PARA (For)

Qual tipo usar?

Se sabemos o número de vezes que iremos repetir ou testar um bloco de instruções, usamos **FOR** e **REPEAT**.

Caso não soubermos o número de vezes, usamos **WHILE**.

ENQUANTO 🡪 Se o teste lógico for VERDADEIRO, os comandos associados são executados. Repetindo-se enquanto for VERDADEIRO. Caso dê FALSO, o programa encerra a estrutura e volta para o fluxo do programa.

Sintaxe: ENQUANTO [WHILE] (condition) FACA [DO]

//instruções

FIMENQUANTO [ENDWHILE]

É necessário que a variável já tenha um VALOR CADASTRADO. Já que sua primeira verificação é o teste lógico.

REPEAT (Repita) 🡪 Realiza teste lógico no final da estrutura, diferente do WHILE. Sempre executará pelo menos uma vez as instruções antes de verificar o teste lógico.

Repeat continuará a repetir enquanto a condição for FALSA! Se o teste der VERDADEIRO ela sai da estrutura de repetição.

Exemplo 🡪 Site de dados obrigatório. Vai repetir o mesmo algorítmo até que todas as informações estejam corretas.

Sintaxe: REPITA [REPEAT]

//Instruções (Só executa quando for falso).

ATE [UNTIL] (condition)

Exemplo: VisuAlg 🡪 Emprego\_2

PARA (For) 🡪 Usada quando há um termino determinado ou limites fixos.

Exemplo: Programas de rotina em LOGIN e SENHA. Bloqueando o acesso caso tenha um numero X de tentativas.

Sintaxe:

FORMA1:

PARA (variavel) DE (Valor inicial) ATE (valor final) FACA

//instruções executadas repetidamente até a variável atingir o seu valor final

FORMA2:

PARA (variavel) DE (Valor inicial) ATE (Valor final) PASSO (valor incremental) FACA

//instruções executadas repetidamente até a variável atingir o seu valor final

Exemplo: VisuALG 🡪 Emprego\_3

**Estruturas de Dados** 🡪 Forma de estruturar ou organizar dados na memória RAM do computador, com o objetivo de utilizar estes dados de forma mais eficiente.

Os algotirmos manipulam os dados e, quando estes dados estão organizados, eles caracterizam uma estrutura de dados.

A organização e os metodos para manipular as estruturas é o que otimizam e diminuem o espaço utilizado na RAM, além de tornar o código mais simples.

Organização dos dados 🡪 Estrutura composta de dados:

1. Heterogêneas (Registros)
2. Homogêneas (Arrays/Vetores/Matrizes)

Heterogêneas 🡪 Conjunto de dados formados por tipos de dados distintos, em uma mesma estrutura.

Ex: inteiro, real, literal e caracteres (todos juntos)

Homogêneas 🡪 Grupos de dados, do mesmo tipo, armazenados em uma única variável, na memória do computador.

Ex: somente do tipo inteiro, real, caracter. Sem misturas!

Estruturas de dados clássicas 🡪 Arrays, Lista, Fila, Pilha e Árvore (as mais utilizadas).

As estruturas Lista, Fila, Pilha e Árvore são mais complexas e podem ser implementadas por Arrays.

Arrays 🡪 Estrutura de dados homogênea (Só do mesmo tipo).

São lineares e estáticas 🡪 Possuem um número fixo de elementos de um determinado tipo de dado. Esses dados podem ser acessados individualmente por meio de um índice, onde geralmente a posição inicial é ZERO.

Array de tipo VETOR 🡪 Unidimensional (Forma de linha e só com 1 índice)

Array de tipo MATRÍZ 🡪 Multimensional (Similar a uma planilha com dois índices)

Vetor 🡪 Dados do mesmo tipo, armazenados na memória do computador de forma LINEAR e ESTÁTICA (Um após o outro).

Caracteristicas do VETOR:

1. Indentificador único: Nome que segue as mesmas regras de uma variável.
2. Elementos ou Valores: Conteúdo, dado, valor inserido/lido (pelo usuário) ou que a variável recebe.
3. Índice ou Posição: Garantia de independência de cada valor armazenado na estrutura.

Ver foto na pasta 🡪 Vetor

Utilizamos o Vetor quando precisamos armazenar muitos dados do mesmo tipo

Ex: CRM (Customer Relationship Management)

Contexto: Dez vendedores em um dpt pessoal. Como devo armazenar o nome destes 10 vendedores?

Forma1 🡪 Utilizando variáveis: neste caso, seria necessário criar dez variáveis diferentes para armazenar dez nomes

Forma2 🡪 Utilizando Vetores: Desta forma, seria necessário apenas uma variável que armazenaria os dez nomes.

(Palavra Reservada em Portugol) (Reservada)

Sintaxe: identificador: VETOR [posição inicial .. posição final] DE tipo\_de\_dado (Nome do Vetor) (Tamanho do Vetor) (Tipo de dado: inteiro, real, literal)

Ver exemplo em: Cadastro\_de\_Vendedores (VisuALG).

Matriz 🡪 Pode ser considerada a extensão de um Vetor, mas possue LINHAS E COLUNAS, diferente do Vetor.

Composição da Matriz:

1. Identificador único: nome único
2. Elementos ou Valores: Conteúdo, dado, valor inserido/lido (pelo usuário)
3. Índice ou Posição: Definida por dois números e deve ser acessada por um par numérico (LINHA X COLUNA).

REGRA: Sempre se identifica primeiro o número da linha e depois o da coluna.

Ver figura na pasta 🡪 Matriz.

Sintaxe: identificador VETOR [linha-inicial... linha-final, coluna-inicial... coluna-final] DE tipo\_de\_dado.

Se muda as posições na sintaxe, o resto é identico ao Vetor.

Ex: (Matriz de 6 colunas)

MATRIZ: VETOR[0..4, 0..5] DE INTEIRO

(Matriz de 5 linhas)

Atribuindo valores:

Forma1: (Operador)

MATRIZ: [0,0] <- 22 (Valor atribuido)

(Posição)

Forma2:

Leia (MATRIZ: [0,0])

Acessando valores:

ESCREVA (MATRIZ [0,0])

(Neste exemplo o resultado seria 22, já que atribuimos esse valor no outro exemplo)

Contexto 🡪 Há um CRM que precisa:

1. Nome de 5 vendedores
2. Valores vendidos na semana 1
3. E na semana 2
4. Calcular a comissão de 2% em cima dos valores vendidos

Resolução 🡪 VisuALG “Vendedores\_e\_comissao”.