ESCOLA DO FUTURO

Eixo de Gestão e Negócios

Lógica de Programação

UFG UNIVERSIDADE FEDERAL DE GOIÁS

SEDI Secretaria de Estado de Desenvolvimento e Inovação

GOIÁS GOVERNO DO ESTADO

Governo de Goiás
Secretaria de Estado de Desenvolvimento e Inovação
Subsecretaria de Ciência, Tecnologia e Inovação
Superintendência de Capacitação e Formação Tecnológica

ESCOLA DO FUTURO

Governador do Estado de Goiás
Ronaldo Ramos Caiado

Secretário de Estado de Desenvolvimento e Inovação
Marcio Cesar Pereira

Subsecretária de Ciência, Tecnologia e Inovação
Sheila Oliveira Pires

Superintendente de Capacitação e Formação Tecnológica
José Teodoro Coelho

Gerência de Gestão das Escolas do Futuro
Mychelly Ferreira Carlos Simões

Diretoria Geral do CETT - UFG
Dr. Moisés Ferreira Cunha

Diretoria de Ensino CETT – UFG
Drª. Daiana Paula Pimenta

Gerência EaD – CETT - UFG
Tânia Mara Lopes Ribeiro

Coordenação Pedagógica EaD – CETT - UFG
Ana Paula Nogueira Ribeiro

Professora Autor
Ricardo André Naka

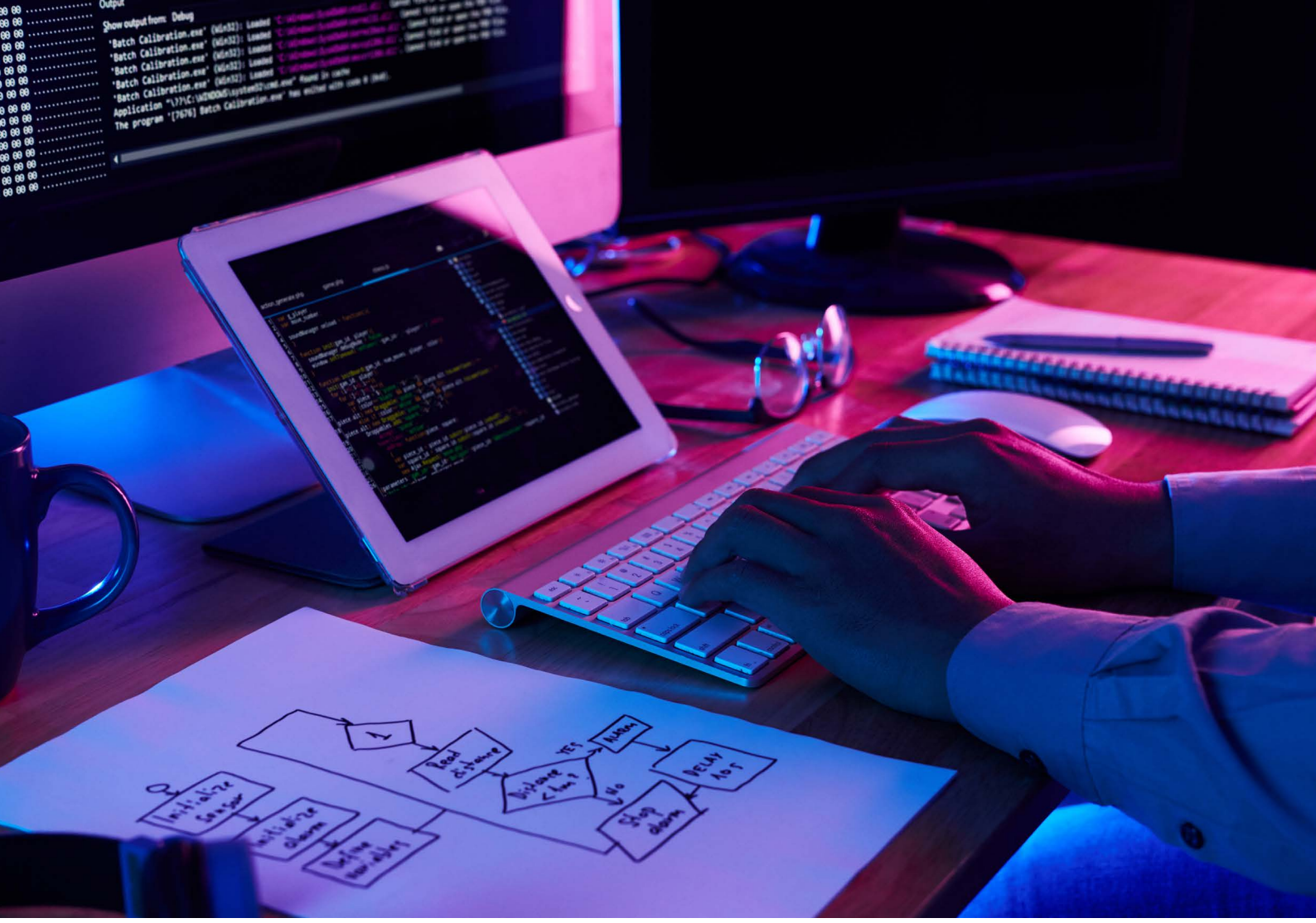
Projeto Gráfico
João Daniell Oliveira

Designer
Maykell Mendes Guimarães

Revisão da Língua Portuguesa
Cícero Manzan Corsi

Banco de Imagens
http://freepik.com

Sumário



Lógica de Programação

Ótimos e proveitosos estudos.
Muito sucesso na futura profissão!

Introdução

O Componente de **Lógica de Programação** tem como objetivo desenvolver o raciocínio lógico e computacional, proporcionando o aprendizado de introdução à computação e lógica de computadores. Nesse componente será possível desenvolver a capacidade de mapear problemas reais em problemas computacionais.

Acerca do desenvolvimento das competências previstas nesse componente curricular, atribui-se um destaque com relação ao emprego do raciocínio lógico estruturado na análise de problemas de programação, utilizando estruturas básicas de linguagens de programação.

Ao longo do Componente de Lógica de Programação, encontraremos exercícios práticos e questionários com todas as orientações necessárias para a realização das atividades para fixar o conteúdo e praticar os conhecimentos que serão abordados.

Para que você obtenha sucesso e aprendizagem satisfatória ao longo do componente procure seguir algumas dicas para o estudo dirigido no ensino a distância. Crie uma rotina diária de estudos, separe 20 minutos para estudar de forma ordenada os módulos deste material, organizando o seu ambiente de estudos sem maiores obstáculos que impeçam a sua concentração, assim como formas de distração que são as redes sociais, o seu celular ou até mesmo qualquer outro barulho ou entretenimento que tome a sua atenção. Por fim, utilize métodos de memorização e realize os exercícios para fixação do conteúdo. Bons Estudos!

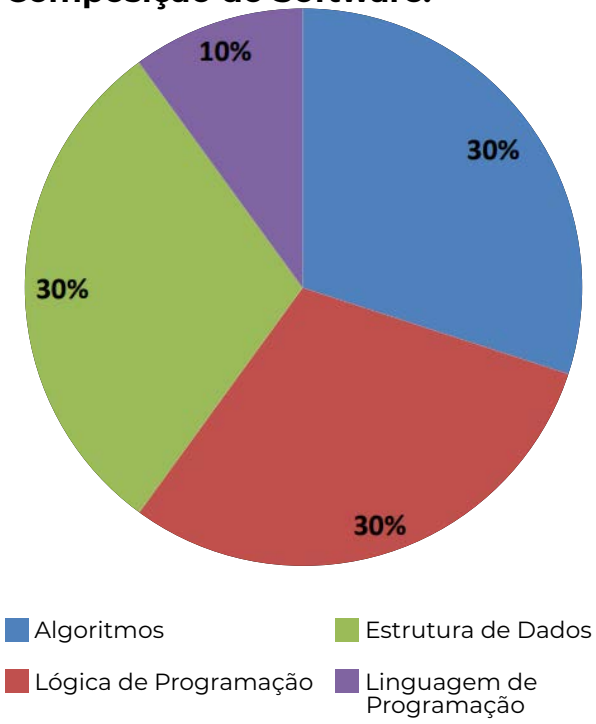
CAPÍTULO 1
Introdução à Lógica de Programação

ALGORITMO

Um Algoritmo serve para representar um conjunto finito de ações, sendo uma solução para um problema, e uma linguagem intermediária entre a humana e as de programação. Representa uma sequência de passos que o computador deve executar a fim de atingir ou obter um resultado. (MICHAELIS, 1998, p.21).

Compreende-se através de citações de Stair e Reynolds (2011), que Software que é o produto final, gerado a partir de um projeto estruturado de dados e que foi programado para processar informações ou dados, tem como base principal, trabalhar com algoritmos. As informações a serem processadas como entrada, podem ser oriundas de dispositivos físicos como Teclado, Mouse, a tela touch screen, e até mesmo de requisições da parte lógica computacional, como arquivos e banco de dados. O Software viabiliza a Saída como resultado do processamento em telas de diversos tipos de computadores e até mesmo smarthphones, arquivos, e comandos eletrônicos como resposta ao usuário daquilo que foi filtrado como requisição, interpretação da informação e resultado computacional da requisição.

Composição do Software:



Como podemos identificar as áreas e habilidades que compõem o projeto de desenvolvimento de um Software, Algoritmos, Lógica de Programação e Estrutura de Dados totalizam 90% e geram um grau de importância maior no que tange ao requisito para

se iniciar a programar, o que torna um grande diferencial entre os programadores. Reconhecer, interpretar e estruturar o problema bem como solucionar o problema.

Conforme Baldwin e Scragg (2004), os algoritmos são a preocupação central do campo, a partir da representação algorítmica conseguimos desenvolver e planejar a construção de algo.

Um **Algoritmo** Pode ser representado em forma:
➘ narrativa;
➘ fluxograma;
➘ pseudocódigo.

Narrativa: Nessa forma de representação, os algoritmos são expressos em linguagem natural.

Exemplo: Instruções Simples para "Passar Café"

1 - Água na Chaleira
2 - Esquentar Água
3 - Filtro na Jarra
4 - Pó de Café no Filtro
5 - Água no Filtro

Perceba que temos uma sequência de ações a serem realizadas e que são fundamentais para se alcançar o objetivo final. Esta sequência seguiu uma lógica que visa colocar em ordem no pensamento e até mesmo visa a correção do pensamento ou correção do raciocínio.

Sequência Lógica: São passos executados até atingir um objetivo ou solução de um problema.

Lógica de Programação: É a técnica de encadear pensamentos para atingir determinado objetivo.

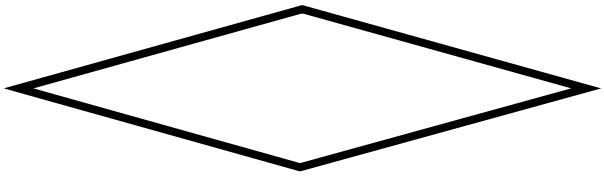
Segundo Aguilar (2011), o pseudocódigo é considerado uma ferramenta que pode auxiliar a programação, na interpretação e o desenvolvimento de um programa.

Fluxograma: O Diagrama de Chapin, do criador Ned Chapin, apresenta uma visão hierárquica e estruturada da lógica do programa, tendo como vantagem a representação das estruturas. É uma representação gráfica dos algoritmos, na qual cada figura geométrica representa diferentes ações que tem como objetivo facilitar o entendimento das ideias contidas no algoritmo.

Início e Fim de Programa

Operação de Atribuição

Operação de Entrada de Dados



Decisão

Operação de Saída de Dados

Exemplo de fluxograma:

➘ Início (dentro de uma elipse);

➘ calcular média de 4 bimestres (dentro de um retângulo com um dos cantos dobrados);

➘ a média para passar é 7 (dentro de um retângulo);

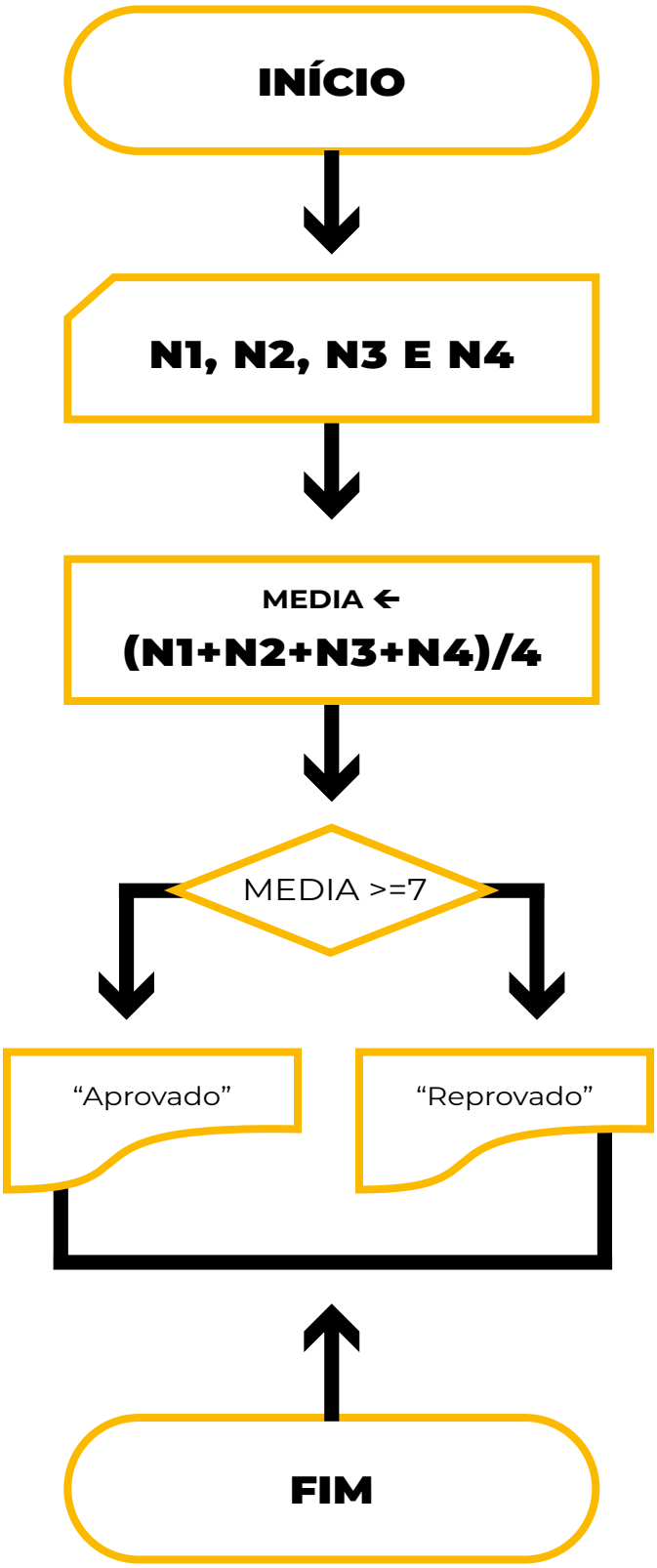
➘ indicar “Aprovado” ou “Reprovado” como saída (verifica se a média é maior ou igual a 7 dentro de um losango);

➘ se a média for maior ou igual a 7 imprime “Aprovado” dentro de um retângulo comum dos lados, recortado de maneira ondulada;

➘ se a média for menor do que 7 imprime “Reprovado” dentro de um retângulo com um dos lados recortado de maneira ondulada;

➘ Fim de programa (dentro de uma elipse).

Pseudocódigo: Forma de representação de algoritmos rica em detalhes, o que acaba sendo uma aproximação do código final a ser escrito em uma linguagem de programação.



Exemplo de algoritmo da média das notas dos 4 bimestres em pseudocódigo:

Algoritmo Media;
Var N1, N2, MEDIA: real;

Início
Leia (N1, N2);
MEDIA ← (N1 + N2) / 2;

Se **MEDIA >= 7** então
Escreva **"Aprovado"**

Senão
Escreva **"Reprovado";**
Fim_se
Fim

ATIVIDADES DE Aprendizagem

1 - Faça um algoritmo (descreva os passos) para realizar as seguintes tarefas:
➘ Sacar dinheiro em um caixa eletrônico.
➘ Somar dois números quaisquer.

2 - Acesse os links abaixo e desenvolva o seu raciocínio lógico para resolver alguns problemas

CONTEÚDO Interativo

https://shre.ink/dNO
https://shre.ink/dNR
https://shre.ink/dNZ

3 - Identifique os dados de entrada, processamento e saída no algoritmo abaixo:
➘ Receba código da peça
➘ Receba valor da peça
➘ Receba Quantidade de peças
➘ Calcule o valor total da peça (Quantidade * Valor da peça)
➘ Mostre o código da peça e seu valor total

4 - Faça um algoritmo para "Calcular o estoque médio de uma peça", sendo que ESTOQUEMEDIO= (QUANTIDADE MÍNIMA + QUANTIDADE MAXIMA) / 2.

5 - Construa um diagrama de blocos que:
➘ Leia a cotação do dólar
➘ Leia um valor em dólares
➘ Converta esse valor para Real
➘ Mostre o Resultado

6 - Desenvolva um diagrama que:
➘ Leia 4 (quatro) números
➘ Calcule o quadrado para cada um
➘ Some todos e
➘ Mostre o resultado

7 - Construa um algoritmo para pagamento de comissão de vendedores de peças, levando-se em consideração que sua comissão será de 5% do total da venda e que você tem os seguintes dados:
➘ Identificação do vendedor
➘ Código da peça
➘ Preço unitário da peça
➘ Quantidade vendida
Não se esqueça de construir o diagrama de blocos do referido algoritmo.

CAPÍTULO 2

Estrutura de Dados: Constantes, Variáveis e Tipos de Dados

Estruturas de Dados

Nossos algoritmos trabalham com Dados. Mas o que é, ou pode ser, um "dado"?

↘ É um número;
↘ é um nome;
↘ é um endereço;
↘ é o valor de um produto;
↘ é um pixel de uma imagem;
↘ enfim, pode ser muitas coisas...
E sobre o conceito de estrutura?
↘ É algo que dá forma?
↘ É algo que sustenta?

- Segundo Houaiss: "Aquilo que dá sustentação (concreta ou abstrata) a alguma coisa; armação, arcabouço".
Dados são representados em pequenas estruturas e em geral são chamados de **variáveis ou constantes!**

Tudo que é armazenado dentro do computador, permanece em alguma memória!

Variáveis: Armazenam dados em caráter temporário e tem conteúdo dinâmico, ou seja, podem ser acessadas ou alteradas a qualquer momento;

Constantes: São dados estáticos, o seu conteúdo pode ser acessado a qualquer momento, mas será definido no momento de sua criação e não poderá ser mais alterado.

Tipos Básicos de Dados:

↘ Dados Numéricos Inteiros: São os números positivos e negativos sem casas decimais.
↘ Dados Numéricos Reais: São os números positivos e negativos que possuem casas decimais.
↘ Dados Literais: São sequências de caracteres.
↘ Dados Lógicos ou Booleanos: Podem ser verdadeiros ou falsos.

Como armazenar dados?

O armazenamento de informações pelo computador em sua memória, se dá em uma região nomeada através de uma variável, esta variável possui:

- NOME
- TIPO
- CONTEÚDO

As regras para nomes de variáveis mudam de uma linguagem para outra, sendo importante gerar um nome para a variável antes mesmo que esta seja utilizada.

Sintaxe:
Nome: TipodeDado
Atribuindo ou alterando valores dos dados:
Podemos apenas definir valores coerentes com o tipo de dado.

Sintaxe:
Nome ← Valor

Erros comuns:
Idade: inteiro

Idade ← 22.0 (22.0 é um tipo Real e não inteiro)

Regras para Nomear Variáveis:

1. Nomes de variáveis não podem ser iguais a palavras reservadas.

2. Nomes de variáveis devem possuir como primeiro caractere uma letra ou sublinhado '_' (os outros caracteres podem ser letras, números e sublinhado).

3. Nomes de variáveis não devem ser muito longos.
4. Nomes de variáveis não podem conter espaços em branco.

5. Na sintaxe do Português Estruturado, não há diferença entre letras maiúsculas de minúsculas (NOME é o mesmo que nome).

Exemplo de algoritmo para somar números:

1. X: inteiro **2.** Y: inteiro **3.** soma: inteiro	**Declarações**
4. X ↞ 10 **5.** Y ↞ 20	**Entrada**
6. soma ↞ x+y	**Processamento**
7. escreva(soma)	**Saída**

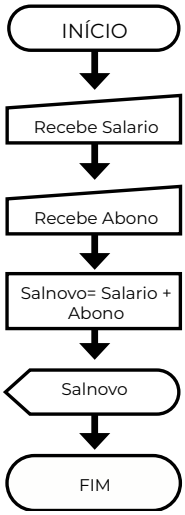
Após estas informações, e de acordo com os conceitos de Mizrahi (2006), concluímos então que estrutura de dados é um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados eficientemente. Vetores, Matrizes, Imagens e diversas outras composições de tipos de dados podem ser chamados de estruturas de dados.

As estruturas de dados permitem melhor organização para o armazenamento e manipulação de dados.

ATIVIDADES DE Aprendizagem

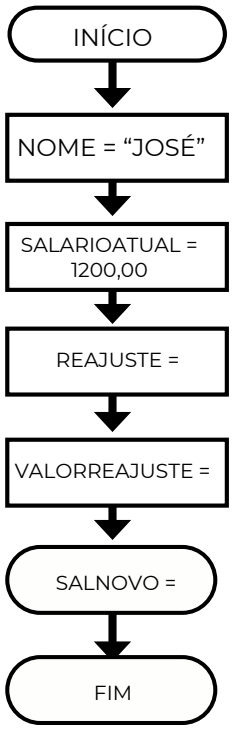
1 - Descreva com suas palavras o significado computacional de Constante e Variável. Cite 02 exemplos de cada.

2 - Faça um teste de mesa no diagrama de bloco abaixo e preencha a tabela ao lado com os dados do teste:



Salário	Abono	Salnovo
600,00	60,00	
350,00		

3 - Sabendo que José tem direito a 15% de reajuste de salário, complete o diagrama ao lado:



MÍDIAS Interativas

Agora que compreendemos o que são algoritmos e como os dados são armazenados e organizados, iremos iniciar nossa prática de algoritmos. Para isso, será disponibilizado neste espaço um pequeno vídeo, de como instalar dos softwares essenciais nesta prática, o Visual G e o Portugol Webstudio online.

Orientações de como instalar e utilizar os recursos:
https://youtu.be/vavJVF2KXiM

Obs: Orientações neste vídeo acontecem a partir do tempo: 1:44:54 até 1:50:49

Link úteis: Visual G:
https://sourceforge.net/projects/visualg30/

Portugol Webstudio online:
https://portugol-webstudio.cubos.io/ide

Winrar 64 bits para extração de arquivos:
https://drive.google.com/file/d/1eu-9ZZHFYPDyJ9asU2rso6k6EPj-dAKyKt/view?usp=sharing

CAPÍTULO 3
Operadores

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador.

Os operadores são símbolos que representam atribuições, cálculos e ordem dos dados, estas operações possuem uma ordem de prioridades (alguns cálculos são processados antes de outros).

São utilizados nas expressões matemáticas, lógicas, relacionais e de atribuição.
Temos três tipos de operadores:

Aritméticos: Resultam em obter dados numéricos que derivam da adição, subtração, multiplicação e divisão, e podem utilizar também o operador para exponenciação. Veja abaixo os símbolos utilizados:

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**

Hierarquia das Operações Aritméticas:

1° ( ) Parênteses
2° Exponenciação
3° Multiplicação, divisão (o que aparecer primeiro)
4° + ou – (o que aparecer primeiro)

Exemplo:
TOTAL = PREÇO * QUANTIDADE

1 + 7 * 2 ** 2 - 1 = 28
3 * (1 – 2) + 4 * 2 = 5

Relacionais: São utilizados para comparar String de caracteres e números. Os valores a serem comparados podem ser caracteres ou variáveis. Estes operadores sempre retornam valores lógicos (verdadeiro ou falso / True ou False).

Os Operadores relacionais são:

Descrição	Símbolo
Igual a	=
Diferente de	< > ou #
Maior que	>
Menor que	<
Maior ou igual a	>=

Exemplo de Algoritmo que utilize Operador Relacional:

```
Algoritmo Pode_Tirar_Carteira_de_Motorista
Var idade: inteiro;
Início
```

Leia idade;
If idade >=18
Escreva “Pode tirar carteira de motorista.”;
Else
Escreva “Não pode tirar carteira de motorista.”;
Fim

Lógicos: Combina resultados de expressões, retornando se o resultado final é verdadeiro ou falso. Os operadores lógicos são:

E	AND
OU	OR
NÃO	NOT

Uma expressão AND (E) é verdadeira se todas as condições forem verdadeiras.

Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira.

Uma expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.

Exemplo: Suponha que temos três variáveis A=5, B=8 e C=1. Os resultados das expressões seriam:

Expressões			**Resultado**
A = B	AND	B > C	**Falso**
A < > B	OR	B < C	**Verdadeiro**
A > B	NOT		**Verdadeiro**
A < B	AND	B > C	**Verdadeiro**
A > = B	OR	B = C	**Falso**
A < = B	NOT		**Falso**

ATIVIDADES DE Aprendizagem

1 - Tendo as variáveis SALARIO, IR e SALLIQ, e considerando os valores abaixo. Informe se as expressões são verdadeiras ou falsas.

SALÁRIO	IR	SALLIQ	EXPRESSÃO	V ou F
2.826,65	211,99	2.614,66	(SALLIQ >= 2.826,65)	
3751,05	562,55	3.188,50	(SALLIQ < 3.188,50)	
4664,68	1049,55	3.615,13	SALLIQ = SALÁRIO - IR	

2 - Sabendo que A= 6, B= 14 e C= 8, informe se as expressões abaixo são verdadeiras ou falsas.

a) (A + C) > B ( )
b) B >= (A + 2) ( )
c) C = (B – A) ( )
d) (B + A) <= C ( )
e) (C + A) > B ( )

3 - Sabendo que A= 5, B= 4, C= 3 e D= 6, informe se as expressões abaixo são verdadeiras ou falsas.

a) (A > C) AND (C < = D) ( )
b) (A + B) > 10 OR (A + B) = (C + D) ( )
c) (A >= C) AND (D >= C) ( )

4) Prática de Algoritmos:
Utilize o **Visual G** ou o **Portugol Webstudio online.**

↘ Faça um algoritmo que receba dois números e exiba o resultado da sua soma.

↘ Faça um algoritmo que receba dois números e ao final mostre a soma, subtração, multiplicação e a divisão dos números lidos.

➘ Escrever um algoritmo para determinar o consumo médio de um automóvel sendo fornecida a distância total percorrida pelo automóvel e o total de combustível gasto.

➘ Escrever um algoritmo que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o seu nome, o salário fixo e salário no final do mês.

MÍDIAS
Interativas

Caso tenha alguma dificuldade de implementar os algoritmos acima, assista à videoaula gravada abaixo que orienta como trabalhar com os parâmetros de entrada e saída além de como atribuir as varáveis. As respostas e explicações de como foram estruturados os algoritmos do exercício 4 estão no link abaixo:
https://youtu.be/Bm9bxTxD8m4

CAPÍTULO 4

Estrutura de decisão e repetição

Ao aplicar as operações lógicas, entendemos que na maioria das vezes necessitamos tomar decisões no andamento do algoritmo, que interferem diretamente no andamento do programa.

Comandos de Decisão ou estruturas de seleção: Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estrutura de programas que não são totalmente sequenciais, onde existem instruções de salto ou desvio, fazendo com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores. Conheceremos as estruturas de decisão: “Se Então”, “Se então Senão” e “Caso Selecione”.

Em algoritmo condicional, a estrutura de decisão ou seleção, permite a escolha de um grupo de ações a ser executado quando determinadas condições, representadas por expressões lógicas, são ou não satisfeitas.

A estrutura de Decisão ou Seleção, permite alterar o fluxo de execução do algoritmo, de forma a selecionar qual parte deve ser executada, sendo esta "decisão" de execução tomada a partir de uma condição, que pode resultar apenas dois valores: **verdadeiro ou falso.** Esta condição é representada por expressões **relacionais ou lógicas.**

A partir das expressões e após executar as funções de validação e comparação, as estruturas de seleção ou decisão irão executar os blocos de comando, definidos de acordo com o resultado da comparação **(verdadeiro ou falso).**

Os tipos de Estruturas de Seleção ou Decisão são: If/Else (Se/Então); Switch/Case (Escolha/Caso)

Exercícios Resolvidos:
Construir um algoritmo que leia 2 números inteiros A e B. Imprimir uma mensagem informando que A é maior que B.

If/Else (Se/Então):

início inteiro: A, B
escreva ("Digite um número inteiro: ")
leia (A)
escreva ("Digite outro número inteiro: ")
leia (B)
se A > B
então escreva ("A é maior que B")
fim se
fim

Entendendo o código: Temos duas variáveis A e B do tipo inteiro, onde temos o parâmetro de entrada solicitando um número inteiro para A e B e na sequência estes números informados são guardados respectivamente em suas variáveis e dentro da estrutura de decisão é feito o comparativo a partir da estrutura SE, na qual irá informar ao usuário a mensagem: "A é maior que B" em caso de teste verdadeiro, em caso falso o programa será finalizado.

Podemos ainda utilizar o SENÃO para acrescentar opções de mensagens de retorno ao usuário, verifique no exemplo a seguir:

Algoritmo Media
Var
nota1, nota2, media: real /* 3 variáveis do tipo real*/
Inicio
Escreva ("Digite nota1:")
Leia (nota1)
Escreva ("Digite nota2:")
Leia (nota2)
Media <– (nota 1 + nota 2) / 2 /*Atribui-se o resultado da expressão que faz a média das notas */

```
Se media >= 60 /* Verifica-se se a média é maior que 60 */
Então
Escreva (“Parabéns! Aprovado!”)
Senão
Escreva (“Aluno Reprovado!”)
Fim se
Fim.
```

Estrutura Switch/Case – Escolha/Caso: É utilizada quando é necessário testar a mesma variável com uma série de valores (várias vezes). Esta variável deve ser sempre do tipo **inteiro** ou **literal,** possibilitando ao usuário escolher um valor dentre vários.

Possui como vantagem, a questão de se evitar uma série de testes com o comando **IF/SE,** funciona de maneira semelhante ao IF/SE encadeado de maneira que o valor que será comparado em cada **CASE/CASO** será passado como parâmetro pelo usuário antes.

Se o valor do parâmetro informado for mesmo (igual) do **CASE/CASO,** será executado o trecho de código dentro do respectivo **CASE/CASO,** em caso do parâmetro informado for diferente do CASE/CASO, será testada a condição de próximo **CASE/CASO.**

O comando **BREAK/PARE** é utilizado para forçar a saída do **SWITCH/ESCOLHA** ao se entrar em um **CASE/CASO,** sem este comando todos os **CASE/CASO** serão testados, mesmo que algum **CASE/CASO** já tenha atendido a condição. Já o comando **DEFAULT/SENÃO** é opcional e define um fluxo alternativo para as opções não atendidas por nenhum **CASE/CASO,** sendo este trecho executado apenas quando o valor de nenhum **CASE/CASO** for igual ao valor do parâmetro informado.

Exemplo: Desenvolva um algoritmo como uma calculadora, onde se digita o primeiro número, depois a operação (subtração, soma, divisão ou multiplicação), em seguida o segundo número. O algoritmo executará um cálculo diferente, dependendo de qual operador o usuário escolher – Utilize o comando caso.

```
algoritmo “Calculadora Basica”
var
num1,num2,resultado : REAL
operacao:caracter
inicio
ESCREVAl (“Digite o primeiro número: “)
LEIA (num1)
ESCREVAl (“Digite a operação: “)
LEIA (operacao)
ESCREVAl (“Digite o segundo número: “)
LEIA (num2)
ESCOLHA operacao
CASO “+”
resultado <- num1 + num2
CASO “-”
resultado <- num1 – num2
CASO “*”
resultado <- num1 * num2
CASO “/”
resultado <- num1 / num2
FIMESCOLHA
ESCREVAl (“Resultado: “, resultado)
fimalgoritmo
```

MÍDIAS
Interativas

Em caso de dúvida, consulte material complementar sobre as estruturas de decisão ou seleção com aplicação de operadores aritméticos e operadores relacionais, consulte a videoaula gravada com aplicação de exercícios.

1 - https://youtu.be/YpA5MQcQJKg
2 - https://youtu.be/SkzHy3lyhiY
3 - https://youtu.be/1gLwsjbgp7g
4 - https://youtu.be/L52_wqfG8Yo
5 - https://youtu.be/-d04ZA8p5oo
6 - https://youtu.be/O_pBF4orSWc

Estrutura de Repetição:

A estrutura de repetição é utilizada quando desejamos que um determinado conjunto de instruções ou comando seja executado em um número definido ou indefinido de vezes, ou enquanto um determinado estado de situações prevalecer ou até que seja alcançado.

Para que servem? Servem para repetir um conjunto de instruções sem que seja necessário escrevê-las várias vezes, sendo sendo este algoritmo pode ser repetido em um número determinado ou indeterminado de vezes. Estas estruturas de repetição também são chamadas de **Laços** ou **Loops.**

Funcionamento: As estruturas de repetição envolvem a avaliação de uma condição (teste) o que resulta em valores **Verdadeiros** ou **Falsos.** Se o resultado da condição é **Falso,** não é iniciada a repetição ou caso esteja em execução, é encerrada a repetição. No caso da condição ser **Verdadeira,** é iniciada a repetição ou, caso esteja em execução, é reiniciada a execução das instruções dentro da Estrutura de Repetição.

A avaliação da condição é sempre novamente realizada após a execução da última instrução dentro da estrutura de repetição, com exceção do Do/While (Faça/Enquanto), garantindo assim que todas as instruções dentro da Estrutura de Repetição do Do/While serão executadas pelo menos uma vez.

Tipos de Estruturas de Repetição:

- For (Para/Faça);
- While (Enquanto/Faça);
- Do/While (Faça/Enquanto).

Estrutura de Repetição FOR (Para/Faça).

É usada quando o número exato de repetições é conhecido, sendo a variável de controle do tipo Inteiro ou Literal.

O significado da palavra para (FOR), indica o início da estrutura de repetição. Após essa palavra vem o nome de uma variável que irá controlar a quantidade de repetições do laço, o tipo desta variável deve ser do tipo inteiro e deve ter sido declarado no blo-

co de declarações de variáveis do seu algoritmo. A palavra reservada de, define o valor inicial que será atribuído à variável de controle, seguido por ate que define o valor que irá encerrar o laço, ou seja, as instruções serão repetidas enquanto a **variável de controle** tiver valor menor ou igual ao valor final.

Por fim, é definido o valor de incremento através da palavra reservada passo. Esse valor é utilizado para incrementar a variável de controle após cada ciclo de execução.

Exemplo:

```
Início
        Inteiro: I
        Real: SOMA, NOTA, MEDIA
        SOMA ← 0
        Para I de 1 até 20 faça
                Escreva ("Informe uma nota:")
                Leia (NOTA)
                SOMA <- SOMA + NOTA
        Fim para
        MEDIA ← SOMA / 40
        Escreva ("Média da turma: ", MEDIA)
        FIM
```

Estrutura de Repetição WHILE (Enquanto/Faça).

Tem como característica ser a estrutura de repetição mais simples, sendo ideal para situações em que não se sabe o número exato de vezes em que o bloco de instruções deve ser repetido. Seu funcionamento faz com que a condição seja validada antes de cada repetição do laço e enquanto a condição for Verdadeira, o bloco de instruções dentro do laço é executado, quando a condição se torna Falsa, o laço é finalizado.

Exemplo:

Um algoritmo para calcular a média de um conjunto de 50 valores inteiros fornecidos em uma unidade de entrada qualquer.

```
Inicio
    Inteiro: NUM, I, SOMA
    real: MEDIA
    I ← 1
    SOMA ← 0
    Enquanto I <= 50 faça
      escreva ("Digite um número")
      Leia (NUM)
      SOMA ← SOMA + NUM
      I ← I + 1
      fim enquanto
      MEDIA ← SOMA / 50
      Escreva ("Média = ", MEDIA)
  Fim
```

Entendo o código: Declara-se as variáveis que serão utilizadas neste algoritmo, em seguida faz-se a inicialização dessas variáveis. Aplica-se o laço de repetição, na qual o objetivo deste algoritmo é somar todos os números e trazer como resultado final a média. Este processo é embutido em um laço de repetição que irá receber até 50 números, guardar na variável ao término, somar estes números e mostrar a média destes números informados pelo usuário, este procedimento será incrementado 50 vezes

até que atinja a condição 51, retornando como Falo o parâmetro principal e o comando de repetição então será abandonado, calcula-se a média aritmética e a imprime em seguida como resultado do algoritmo.

Estrutura de Repetição DO/WHILE (Para/Faça).

A estrutura DO/WHILE, testa a condição de validação do laço apenas no final do comando, sendo assegurado assim que as instruções dentro do laço serão executadas pelo menos uma vez.

A diferença para a estrutura WHILE é que na DO/WHILE a condição de validação é verificada após a execução do bloco de instruções do laço.

Quando este laço for executado todas as instruções dentro deste serão executadas, independente da condição estabelecida e somente após a primeira execução das instruções do laço é que a expressão será testada. Depois da primeira execução, as instruções dentro do laço só são executadas novamente se a condição de validação for **Verdadeira.**

Exemplo:

```
Inicio
Inteiro: I
Real: SOMA, NOTA, MEDIA
SOMA ← 0
Para I de 1 até 20 faça
    Escreva ("Digite uma nota: ")
    Leia (NOTA)
    SOMA ← SOMA + NOTA
    Fim para
    MEDIA ← SOMA / 20
    Escreva ("Média da turma: ", MEDIA)
Fim
```

Entendendo o código: Neste algoritmo serão somadas as notas para depois calcularmos a média, veja que não é necessário inicializar a variável de controle I, sendo feito de forma automática implementado na instrução para. Saindo então, do laço de repetição, calcula-se a média aritmética e a imprime em seguida.

MÍDIAS Interativas

Em caso de dúvida, consulte material complementar sobre as estruturas de repetição, consulte a videoaula gravada com aplicação de exercícios.

1 - https://youtu.be/2jFCOxtj5Us
2 - https://youtu.be/O_pBF4orSWc
3 - https://youtu.be/VElnTAjr6nY
4 - https://youtu.be/PhSML3ChhAA
5 - https://youtu.be/PhAJDguglNA
6 - https://youtu.be/36wR0D4UOg4

CAPÍTULO 5

Funções e Procedimentos

Também chamados de subalgoritmos que efetuam um ou mais cálculos determinados, tendo como objetivo evitar a escrita de um código grande e modularizando então em algoritmos menores. É muito utilizado quando precisamos utilizar uma mesma tarefa em diversos lugares, evitando assim a reescrita do trecho diversas vezes.

Características:

- Reduzem o tamanho do algoritmo
- Facilitam a compreensão e visualização do algoritmo
- São declarados no início do algoritmo e podem ser chamados em quaisquer pontos após sua declaração.

Funções: Bloco de comandos que têm por objetivo retornar um valor ou uma informação. A chamada de uma função é feita através da citação do seu nome seguido, opcionalmente, de seus argumentos iniciais entre parênteses. Estas funções podem ser **predefinidas** pela linguagem ou criadas pelo programador.

Funções predefinidas: A linguagem do Visualg possui diversas funções predefinidas que podem ser usadas na construção de algoritmos.

Exemplo: Criar um algoritmo que calcule o valor da raiz quadrada de um número.

O Visualg possui uma função predefinida que recebe como parâmetro uma variável (do tipo real) e retorna um valor (também do tipo real) referente à raiz quadrada do número.

Funções predefinidas: Assinatura da função para cálculo da Raiz Quadrada.

Raizq (valor : real) : real
Nome da Função Tipo do parâmetro
Tipo de Retorno

Como utilizar:

```
Algoritmo "Uso_de_Função_Predefinida"
Var
    numero, raiz: real
inicio
```

```
escreva (“Digite um número:”)
leia (numero)
raiz ← Raizq(numero)
escreva (“A raiz quadrada do número digitado é: “, raiz)
fimalgoritmo
```

Atribuindo à variável raiz (do tipo real) o retorno da função Raizq

Chamada da função passando a variável numero (do tipo real) como parâmetro da função Raizq

Lista das Funções Predefinidas do Visualg.

Funções numéricas, algébricas e trigonométricas:

➘ **Abs(expressão)** - Retorna o valor absoluto de uma expressão do tipo inteiro ou real. Equivale a | expressão | na álgebra.
➘ **ArcCos(expressão)** - Retorna o ângulo (em radianos) cujo co-seno é representado por expressão.
➘ **ArcSen(expressão)** - Retorna o ângulo (em radianos) cujo seno é representado por expressão.
➘ **ArcTan(expressão)** - Retorna o ângulo (em radianos) cuja tangente é representada por expressão.
➘ **Cos(expressão)** - Retorna o co-seno do ângulo (em radianos) representado por expressão.
➘ **CoTan(expressão)** - Retorna a co-tangente do ângulo (em radianos) representado por expressão.
➘ **Exp(base, expoente)** - Retorna o valor de base elevado a expoente, sendo ambas expressões do tipo real.
➘ **GraupRad(expressão)** - Retorna o valor em radianos correspondente ao valor em graus representado por expressão.
➘ **Int(expressão)** - Retorna a parte inteira do valor representado por expressão.
➘ **Log(expressão)** - Retorna o logaritmo na base 10 do valor representado por expressão.
➘ **LogN(expressão)** - Retorna o logaritmo neperiano (base e) do valor representado por expressão.
➘ **Pi -** Retorna o valor 3.141592.
➘ **Quad(expressão)** - Retorna quadrado do valor representado por expressão.
➘ **RadpGrau(expressão)** - Retorna o valor em graus correspondente ao valor em radianos representado por expressão.
➘ **RaizQ(expressão)** - Retorna a raiz quadrada do valor representado por expressão.
➘ **Rand -** Retorna um número real gerado aleatoriamente, maior ou igual a zero e menor que um.
➘ **RandI(limite)** - Retorna um número inteiro gerado aleatoriamente, maior ou igual a zero e menor que limite.
➘ **Sen(expressão)** - Retorna o seno do ângulo (em radianos) representado por expressão.
➘ **Tan(expressão)** - Retorna a tangente do ângulo (em radianos) representado por expressão.

Funções para manipular cadeias de caracteres:

➘ **Asc (s : caracter) :** Retorna um inteiro com o código ASCII do primeiro caracter da expressão.

➘ **Carac (c : inteiro) :** Retorna o caracter cujo código ASCII corresponde à expressão.

➘ **Caracpnum (c : caracter) :** Retorna o inteiro ou real representado pela expressão. Corresponde a StrToInt() ou StrToFloat() do Delphi, Val() do Basic ou Clipper, etc.

➘ **Compr (c : caracter) :** Retorna um inteiro contendo o comprimento (quantidade de caracteres) da expressão.

➘ **Copia (c : caracter ; p, n : inteiro) :** Retorna um valor do tipo caracter contendo uma cópia parcial da expressão, a partir do caracter p, contendo n caracteres. Os caracteres são numerados da esquerda para a direita, começando de 1. Corresponde a Copy() do Delphi, Mid$() do Basic ou Substr() do Clipper.

➘ **Maiusc (c : caracter) :** Retorna um valor caracter contendo a expressão em maiúsculas.

➘ **Minusc (c : caracter) :** Retorna um valor caracter contendo a expressão em minúsculas. Numpcarac (n : inteiro ou real) : Retorna um valor caracter contendo a representação de n como uma cadeia de caracteres. Corresponde a IntToStr() ou FloatToStr() do Delphi, Str() do Basic ou Clipper. Pos (subc, c : caracter): Retorna um inteiro que indica a posição em que a cadeia subc se encontra em c, ou zero se subc não estiver contida em c. Corresponde funcionalmente a Pos() do Delphi, Instr() do Basic ou At() do Clipper, embora a ordem dos parâmetros possa ser diferente em algumas destas linguagens.

Criando Funções

A criação de uma função deve ser realizada dentro da seção de variáveis. Esse tipo de subalgoritmos sempre retorna apenas um valor para o algoritmo que o chamou. As funções possuem um tipo de retorno associado e pode possuir 0, 1 ou mais parâmetros.

Sintaxe:

Algoritmo <nome do algoritmo>
var
    <declaração das variáveis globais>
    <definição das funções>i
inicio
    <lista de comandos>
fimalgoritmo

Variáveis Locais: As variáveis locais são declaradas dentro dos subalgoritmos (funções ou procedimentos), podem ser usadas apenas dentro das funções, sendo que o algoritmo que chamou a função ou procedimento não tem acesso a estas funções.

Variáveis Globais: São variáveis declaradas na seção var do algoritmo. Qualquer função ou procedimento pode alterar o valor ou utilizá-la durante o seu procedimento.

Exemplo: Criando Funções para calcular o dobro de um número passado como parâmetro.

inicio

Comandos
escreva (“Digite um número para calcular o dobro:”)
leia (numero)
resultado ← Dobro(numero)
Comandos
escreva(“O dobro é: “, resultado)

fimalgoritmo

Os algoritmos podem possuir várias funções:

```
Algoritmo “FuncoesPersonalizadas”
var
numero: real

função Dobro (valor: real) : real
var
    total: real
inicio
    total ← valor * 2
    retorne total
fimfuncao
função Triplo (valor: real): real
var
    total: real
inicio
total ← valor * 3
    retorne total
    fimfuncao

inicio
escreva(“Digite um número:”)
leia (numero)
escreval(“O dobro é: “, Dobro(numero) )
escreval(“O triplo é: “, Triplo (numero) )
fimalgoritmo
```

Procedimentos

Em Visualg, os procedimentos diferem das funções apenas por não retornarem valor nenhum, sendo sua sintaxe descrita da seguinte forma:

```
procedimento <nome do procedimento> (<parâmetros>)
var
<declaração das variáveis locais>
inicio
        <lista de comandos>
fimprocedimento
```

Exemplo: Crie um procedimento que receba um valor como parâmetro e escreva o dobro desse número.

```
algoritmo “Procedimento”
var
        numero: inteiro
```

Não possui tipo de retorno e não retorna nada. Apenas executa o que está na seção de comandos.

Não possui tipo de retorno e não retorna nada. Apenas executa o que está na seção de comandos.

```
procedimento Dobro (valor: inteiro)
    var
    total: inteiro
inicio
    total ← valor * 2
    escreva (“O dobro é: “, total)
fimprocedimento
```

```
inicio
    escreva (“Digite um número: “)
    leia (numero)
    Dobro (numero)
fimalgoritmo
```

É feita apenas a chamada do procedimento, sem precisar atribuir a nenhuma variável

Declarações Globais e Locais

- **Variáveis Globais** – São vistas por toda a aplicação.

- **Variáveis Locais –** São vistos apenas dentro do procedimento que o criou.

- **Variáveis Locais** - podem possuir o mesmo nome das variáveis Globais.

```
algoritmo exemplo2
var nomeGlobal:literal   \\ Declaração de variável Global \\
procedimento imprimir \\ Declaração de variável Local \\
var nomeLocal:literal
inicio
nomeLocal<-”algoritmos”
escreva(“Esta é a variável Local: ”, nomeLocal)
fimprocecimento
inicio
nomeGlobal<- “ALGOL”
escreva(“Esta e a variavel Global:”,nomeGlobal)
imprimir()
fimalgoritmo
```

Passagem de Parâmetros: São canais por onde os dados são transferidos pelo algoritmo chamador a um subalgoritmos.

Parâmetros Formais: São os nomes simbólicos usados na definição dos parâmetros de um subalgoritmos.

```
Procedimento soma (a, b: inteiro) \\ Parâmetros Formais\\
var
        calculo: inteiro
```

Parâmetros Reais: São aqueles que substituem os parâmetros formais quando da chamada de um subalgoritmos.

```
Inicio
        X← 10
        Y← 2
        Soma (x, y) \\ Parâmetros reais \\
fimalgoritmo
```

Mecanismos de passagem de parâmetros:

A substituição dos parâmetros formais pelos parâmetros reais no ato da invocação de um subalgoritmos é denominada de passagem de parâmetros. Pode ocorrer por dois mecanismos distintos:

- **Passagem por valor (ou por cópia)**

Na passagem por valor, é criada uma cópia dos parâmetros reais. As modificações no parâmetro formal não afetam o parâmetro real, pois trabalha-se

apenas com uma cópia.

```
algoritmo “PassagemPorValor”
var
        x, y: inteiro
        procedimento soma (a, b: inteiro)
        var
                calculo: inteiro
        inicio
                calculo ← a + b
                escreval(“O valor da soma é: “, calculo)
                a←3
                b←4
        fimprocedimento
        inicio
                x←1
                y←2
                soma (x, y)
                escreval
(“Os valores de x e y são: “, x, y)
        fimalgoritmo
```

- **Passagem por referência**

O espaço de memória ocupado pelos parâmetros reais é compartilhado pelos parâmetros formais correspondentes. As modificações efetuadas nos parâmetros formais também afetarão os parâmetros reais. Na linguagem do Visualg, utiliza-se a palavra var antes do nome do parâmetro na declaração da função para informar que a passagem será por referência

```
Algoritmo “PassagemPorReferencia”
Var
x, y: inteiro
procedimento soma(var a, b : inteiro)
var
calculo: inteiro
inicio
    calculo ← a + b
    escreval(“O valor da soma é: “, calculo)
    a ← 3
    b ← 4
fimprocedimento
inicio
    x ← 1
    y ← 2
    soma (x, y)
    escreval
    (“Os valores de x e y são: “, x, y)
    fimalgoritmo.
```

Mesmo alterando os valores de a e b, os valores de x e y continuam os mesmo após a chamada do subalgoritmo.

Funções e procedimentos são utilizados com muita frequência em desenvolvimento de softwares. São vários benefícios como: evita duplicação de código quando precisamos executar a mesma operação várias vezes, deixa o entendimento do algoritmo mais intuitivo, pois tiramos a parte complexa do código do fluxo principal do algoritmo, etc.

Em caso de dúvida, consulte material complementar **sobre funções e procedimentos,** consulte a vídeo aula gravada com aplicação de exercícios.

MÍDIAS Interativas

1 - https://youtu.be/YoHSbpoZxAs

CAPÍTULO 6

Vetores e Matrizes

Vetores

Iniciamos o assunto com a problemática acerca da declaração de variáveis para receber por exemplo 50 variáveis para nome e depois para as médias de cada aluno em um sistema escolar. Para esta situação, utilizaremos dentro da estrutura de dados, o recurso conhecido como **vetor,** que é uma espécie de caixa com várias divisórias para armazenar dados.

Os vetores são definidos pelo tipo de dados que eles devem armazenar e a quantidade de posições.

Exemplo:

- Vetor de 8 posições para armazenar números reais.

- Vetor de 40 posições para armazenar caracteres.

Os vetores são estruturas homogêneas, ou seja, só armazena dados do tipo inteiro.

Sintaxe no Visualg.

<nome_variavel>: vetor [posInicial.. posFinal] de <tipo>

Exemplo:

```
Algoritmo “exemplo_vetores”
var
    nome_alunos: vetor [1..50] de caractere
    media_alunos: vetor [1..50] de real
inicio
    nome_alunos[1] ← “Pedro”
    leia (nome_alunos[2])
    nome_alunos[3] : = “Joana”
    media_alunos[1] := 8.5
```

Preenchendo um vetor.

Podemos utilizar um laço de repetição para facilitar o preenchimento dos dados em vetores.

Exemplo:

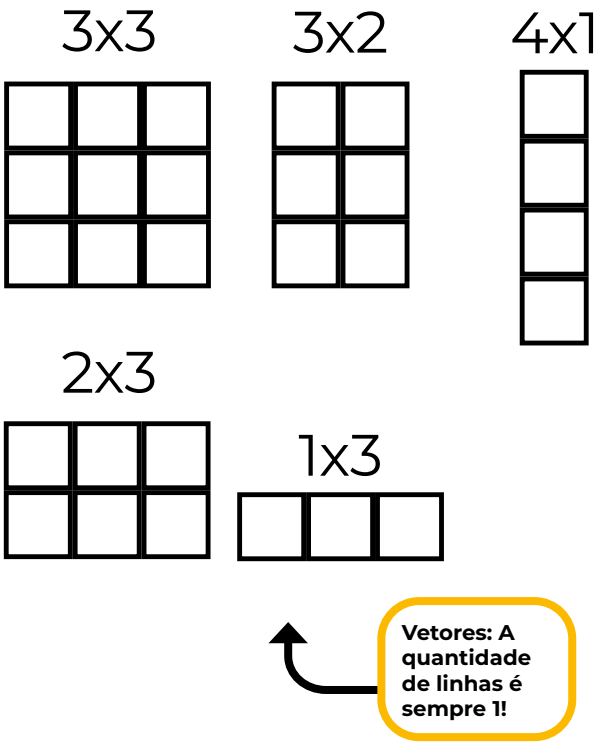
```
Algoritmo “exemplo_vetores”
Var
        numeros: vetor [1..10] de inteiro
        i: inteiro
inicio
        para i de 1 ate 10 faca
```

```
        escreva(“Digite um valor para ser adicionado ao vetor”, i, “do vetor:”)
        leia(numeros [i])
        fimpara
        para i de 1 ate 5 faca
        escreva(“O valor que está na posição”, i, “é:”, numeros[i])
        fimpara
fimpara
```

Matriz

É uma estrutura de dados que contém várias variáveis do mesmo tipo, são comumente referenciadas através de suas dimensões (quantidade de linhas e colunas).

A notação comum é: MxN, onde M é a dimensão vertical (quantidade de linhas) e N é a dimensão horizontal (quantidade de colunas)



Notação:
Como referenciar um elemento específico da matriz?

Exemplo: Matriz 3x2 (três linhas e duas colunas)

	1	2
1	1,1	1,2
2	2,1	2,1
3	3,1	3,1

Sintaxe no VisualG
Declaração:
<nome_variavel>: vetor[li..lf, ci..cf] de <tipo>
Onde:

- li e lf representam respectivamente o índice inicial e final das linhas e
- ci e cf representam, respectivamente o índice inicial e final das colunas.

Para declarar uma matriz 3x2 de inteiro

```
Algoritmo “exemplo_matriz”
var
        exMatriz: vetor [1..3, 1..2] de inteiro

inicio
        exMatriz[1,1] ←10
        leia (exMatriz [1,2])
        exMatriz[3,1] := 4
fimalgoritmo
```

Preenchendo uma matriz: Se quiser-

mos atribuir valores a todas as posições da matriz, podemos fazer:

```
algoritmo “preencher_matrizes”
var
        numeros: vetor[1..3, 1..2] de inteiro
        i: inteiro
inicio
        para i de 1 ate 3 faca \\ fazer o laço para as linhas\\
        escreva (“Digite o valor para a posição”, i, “, 1”:)
        leia (numeros[i,1])
        escreva (“Digite o valor para a posição”, i, “, 2”:)
        leia (numeros[i,2])
        fimpara
fimalgoritmo
```

Podemos ainda incluir um laço de repetição para variar pelas colunas também, por exemplo:

```
algoritmo “preencher”
var
        numeros: vetor[1..3, 1..2] de inteiro
        i, j: inteiro
inicio
        para i de 1 ate 3 faca
                para j de 1 ate 2 faca
                        escreva(“Digite um valor para a posição [“, i, “,”, j, “]: “)
                        leia(numeros [i, j])
                fimpara
                para i de 1 ate 3 faca
                        escreva(“O valor que está na posição [“, i, “, 1] [ é:”, numeros[i,1])
                        escreva(“O valor que está na posição [“, i, “, 2] [ é:”, numeros[i,2])
                fimpara
        fimpara
fimalgoritmo
```

Em caso de dúvida, consulte material complementar sobre vetores e matrizes, consulte a videoaula gravada com aplicação de exercícios.

O Componente de **Lógica de Programação** tem como objetivo desenvolver o raciocínio lógico e computacional do aluno, visando propiciar o aprendizado de introdução à computação e lógica de programação de computadores.

O aluno capacitado por este curso terá a capacidade de mapear problemas reais em problemas computacionais e apresentar os conceitos gerais das linguagens de programação e suas estruturas básicas.

MÍDIAS Interativas

1 - https://youtu.be/4OGUynjqLPM

Referências

ASCENCIO, A. F. G.; CAMPOS C, E.A. V. **Fundamentos da Programação de Computadores.** 3. ed. Editora Pearson, 2010.

FOBERLONE, A.L.V.; EBERSPACHER, H.F., **Lógica de Programação -** A construção de algoritmos e estruturas de dados. 3. ed., São Paulo: Prentice Hall, 2005.

MIZRAHI, V. V. Treinamento em Linguagem C++: Módulos 1 e 2. 2ª ed. São Paulo: Prentice Hall, 2006.

NAPRO – Núcleo de Apoio Aprendizagem de Programação. Disponível em: http://www.guanabara.info/logica/Apostilas/VisuAlg_Ref.pdf.

262588213843476. **Programação em C++ - Algoritmo de Dijkstra.** Gist (em inglês). Consultado em 31 de março de 2022.

STAIR, Ralph M.; REYNOLDS, George W. **Princípios de sistemas de informação.** 9ª edição. São Paulo: Cengage Learning, 2011.