Escola do Futuro – EFG – Paulo Renato Sousa

Professor – Roberto Carlos

Definição de Algoritmos

Algoritmos são uma sequência ordenada de instruções (passos) que devem ser seguidos para atingir um objetivo. Eles são fundamentais em diversas áreas, não apenas na computação.

Exemplos comuns de algoritmos incluem:

- Manuais
- Receitas
- Roteiros para realização de tarefas específicas

Estes exemplos demonstram como os algoritmos estão presentes em nosso dia a dia, fornecendo instruções claras e ordenadas para alcançar um resultado desejado.

Especificação de Algoritmos

Você sabe o que é um Algoritmo? Um Algoritmo é...

- A especificação de uma sequência ordenada de instruções, finitas e não-ambíguas, que deve ser seguida para a solução de um determinado problema, garantindo a sua repetibilidade.
- Uma linguagem intermediária entre a linguagem humana e as linguagens de programação.
- Utilizado para representar a solução de um problema.
- Descreve instruções a serem executadas pelos computadores.

Utilidade dos Algoritmos

Solução de Problemas

O algoritmo é uma sequência de passos lógicos e finitos que permite solucionar problemas. Ao criarmos um algoritmo, indicamos uma dentre várias possíveis sequências de passos para solucionar o problema.

Base para Programação

O objetivo de aprender a criar algoritmos é que este é a base de conhecimentos para as linguagens de programação. Dominar a criação de algoritmos é fundamental para o desenvolvimento de habilidades de programação.

Múltiplas Soluções

Em geral, existem muitas maneiras de resolver o mesmo problema. Ou seja, podem ser criados vários algoritmos diferentes para resolver o mesmo problema. Isso permite flexibilidade e criatividade na resolução de problemas.

Algoritmo Computacional

Para que um computador possa desempenhar uma tarefa é necessário que esta seja detalhada, passo a passo, em uma linguagem compreensível pela máquina, por meio de um... Programa.

Um programa de computador é um algoritmo escrito em um formato compreensível pelo computador.

Na elaboração de um algoritmo devem ser especificadas ações claras e precisas que resultem na solução do problema proposto;

A lógica está na correta sequência de passos que deve ser seguida para alcançar um objetivo específico;

O grau de detalhe do algoritmo dependerá da situação em que o programador se encontra.

Propriedades Essenciais de um Algoritmo

Completo

Todas as ações precisam ser descritas e devem ser únicas.

Sem redundância

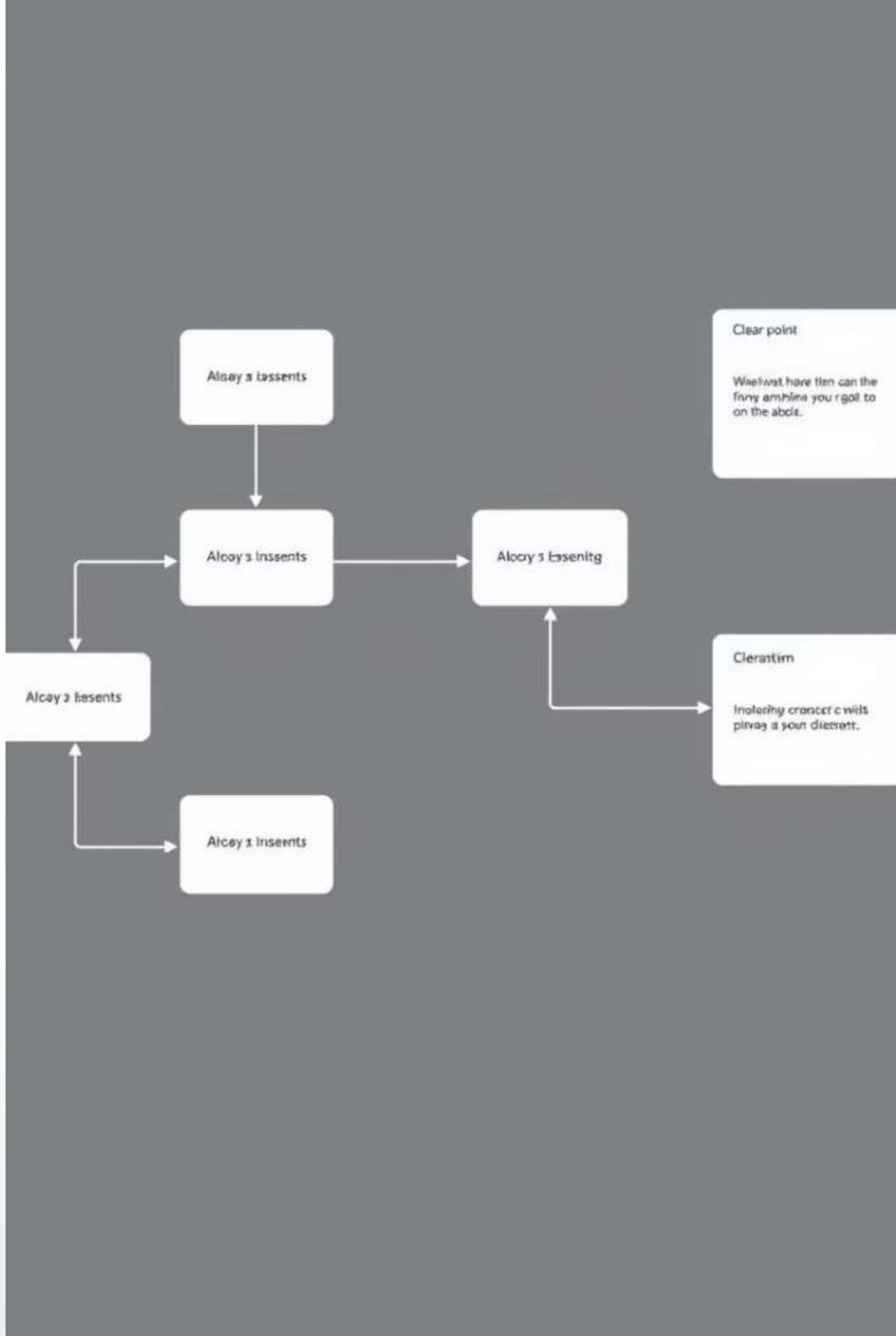
Um conjunto de instruções só pode ter uma única forma de ser interpretada.

Determinístico

Se as instruções forem executadas, o resultado esperado será sempre atingido.

Finito

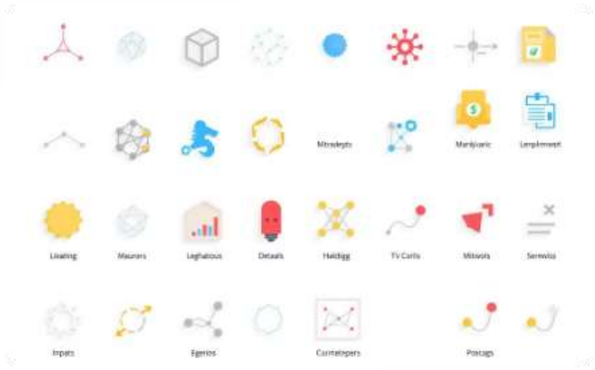
As instruções precisam terminar após um número limitado de passos.



Formas de Representação de Algoritmos

Você conhece alguma forma de representação (escrita) dos algoritmos?

Existem diversas formas de representação de algoritmos, mas não há uma forma considerada a melhor. Entre as principais diferenças está o maior ou menor nível de detalhamento (grau de abstração).



Formas mais conhecidas de representação

Descrição narrativa; Fluxograma; Pseudocódigo (Linguagem estruturada ou Portugol).

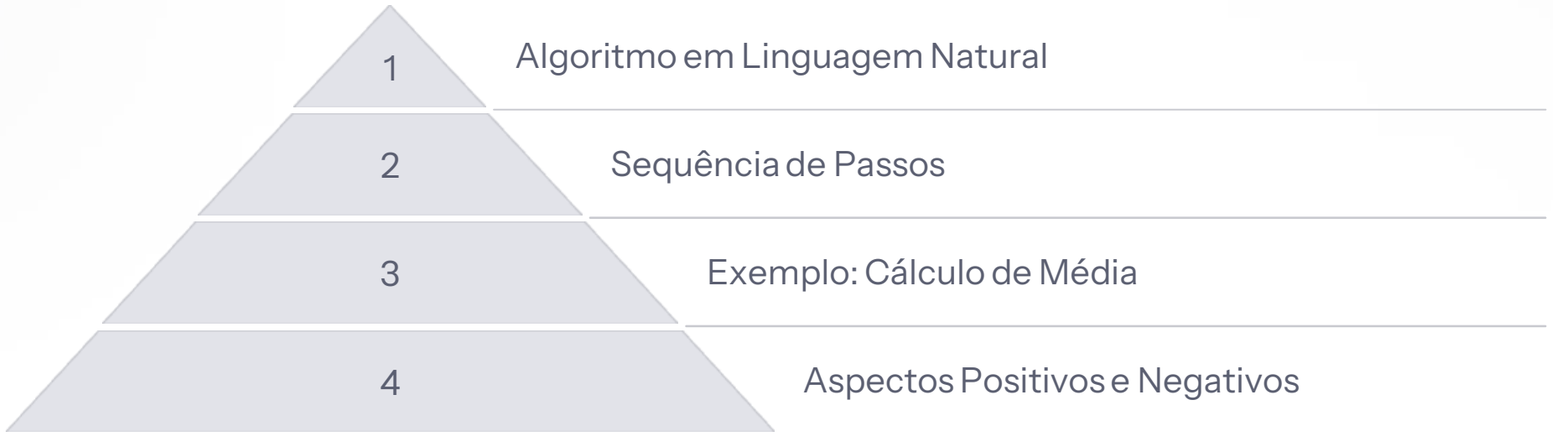
Escolha da representação

Cada uma das formas de representação possui vantagens e desvantagens. Cabe ao programador escolher qual forma oferece as melhores características de acordo com a situação/problema.

Combinação de representações

É comum a combinação das representações, principalmente quando há a necessidade do entendimento por vários tipos de pessoas.

Descrição Narrativa



Os algoritmos são expressos diretamente em linguagem natural. Ou seja, a sequência de passos é descrita em nossa língua nativa (português).

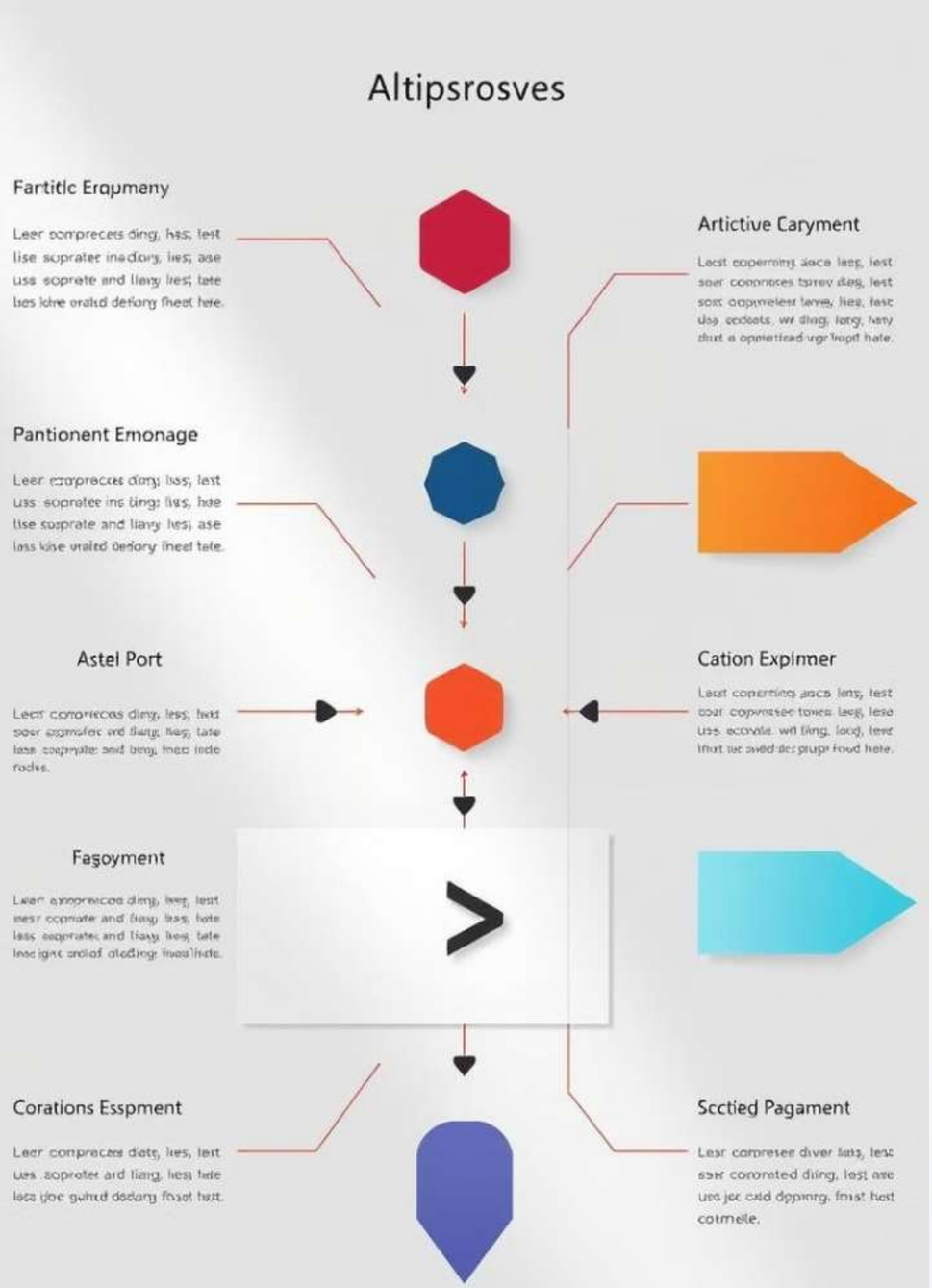
Exemplo:

Cálculo da média de um aluno:

1. Obter as suas 2 notas de provas;
2. Calcular a média aritmética;
3. Se a média for maior ou igual a 7, o aluno foi aprovado;
4. Senão o aluno foi reprovado.

Aspecto positivo - Não é necessário aprender novos conceitos, pois a língua natural já é bem conhecida.

Aspecto negativo - A língua natural dá oportunidade para várias interpretações e ambiguidades, dificultando a transcrição desse algoritmo para programa.



Fluxograma

É uma representação gráfica em que formas geométricas diferentes implicam ações (instruções, comandos) distintos.

É mais precisa que a Descrição Narrativa, porém não se preocupa com detalhes de implementação do programa, como o tipo das variáveis utilizadas. O fluxograma utiliza símbolos específicos para a representação gráfica dos algoritmos.

Os símbolos sofrem algumas variações de acordo com o autor ou ferramenta em uso.

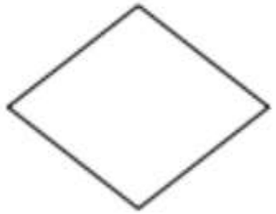
Aspecto positivo

O entendimento de elementos gráficos é mais simples que o entendimento de textos.

Aspecto negativo

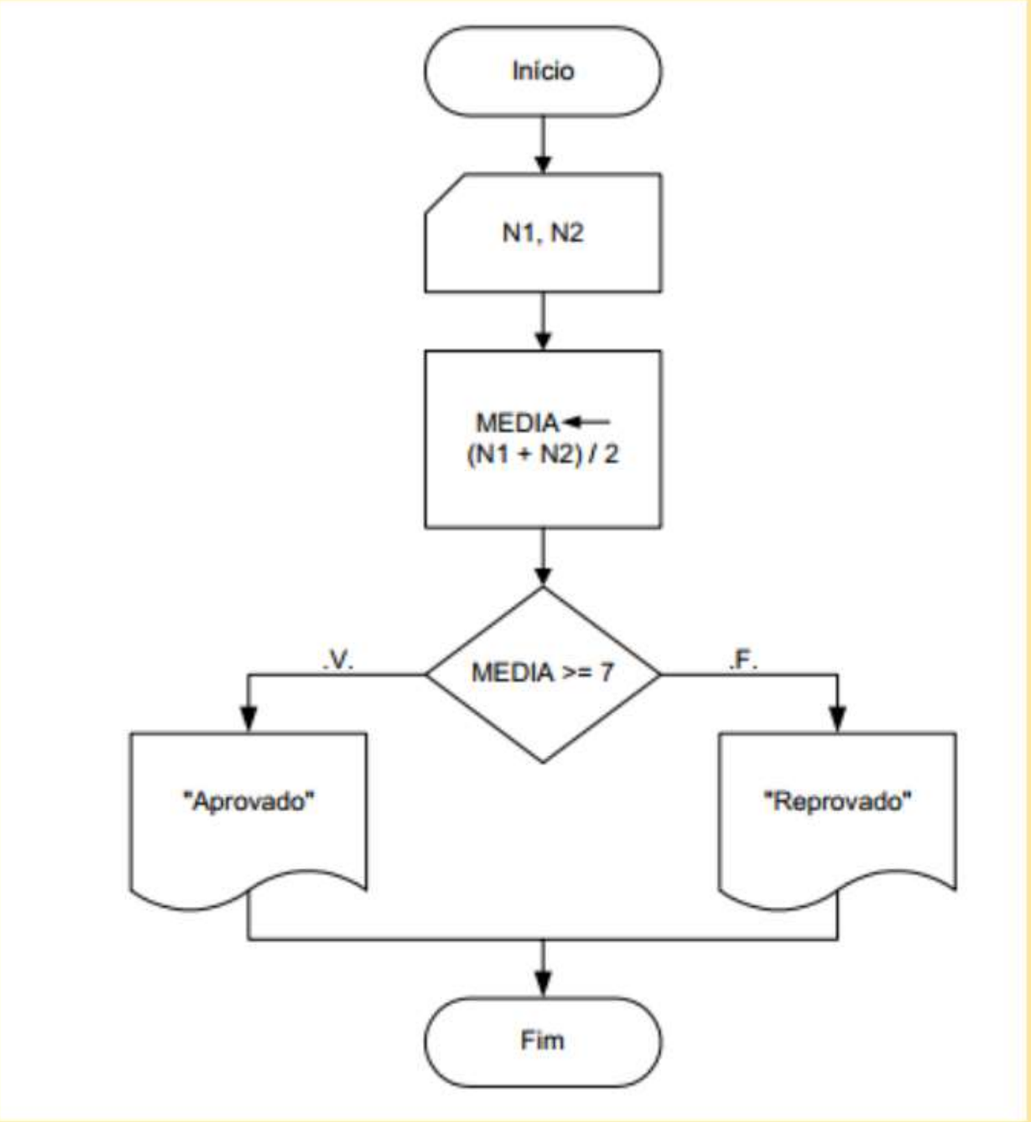
Os fluxogramas devem ser entendidos e o algoritmo resultante não é detalhado, dificultando sua transcrição para um programa.

Elementos gráficos

	Início e final do fluxograma
	Operação de entrada de dados
	Operação de saída de dados
	Operação de atribuição
	Decisão

Exemplo:

- Cálculo da média de um aluno:



Pseudocódigo

- É rico em detalhes, como a definição dos tipos das variáveis usadas no algoritmo.
- **Algoritmo:** Palavra que indica o início da definição de um algoritmo em forma de pseudocódigo.
- Nome simbólico dado ao algoritmo com a finalidade de distingui-lo dos demais.
- Parte opcional onde são declaradas as variáveis globais usadas no algoritmo.
- **Início e Fim:** Palavras que delimitam o início e o término, respectivamente, do conjunto de instruções do corpo do algoritmo.
- Aspecto positivo: Representação clara sem as especificações de linguagem de programação.
- Aspecto negativo: As regras do pseudocódigo devem ser aprendidas.

Pseudocódigo

Estrutura básica do pseudocódigo

Algoritmo <nome_do_algoritmo>

<declaração_de_variáveis>

Início

<corpo do algoritmo>

Fim

Pseudocódigo

Exemplo:

- Cálculo da média de um aluno:

```
Algoritmo Calculo_Media
Var Nota1, Nota2, MEDIA: real;
Início
   Leia Nota1, Nota2;
   MEDIA ← (Nota1 + Nota2) / 2;
   Se MEDIA >= 7 então
      Escreva “Aprovado”;
   Senão
      Escreva “Reprovado”;
   Fim_se
Fim
```

Tipos de Dados

Instruções e Dados

As informações manipuladas pelo computador podem ser classificadas em duas categorias:

Instruções: Coordenam o funcionamento do computador, determinando a maneira como os dados devem ser tratados.

Dados: São as informações a serem processadas pelo computador.

Tipos de Dados

Os dados podem ser do tipo:

Numérico:

Os dados numéricos representáveis em um computador são divididos em duas classes:

Inteiros e Reais

Dados Numéricos

Inteiros

Os números inteiros são aqueles que não possuem componentes decimais ou fracionários, podendo ser positivos ou negativos.

Reais

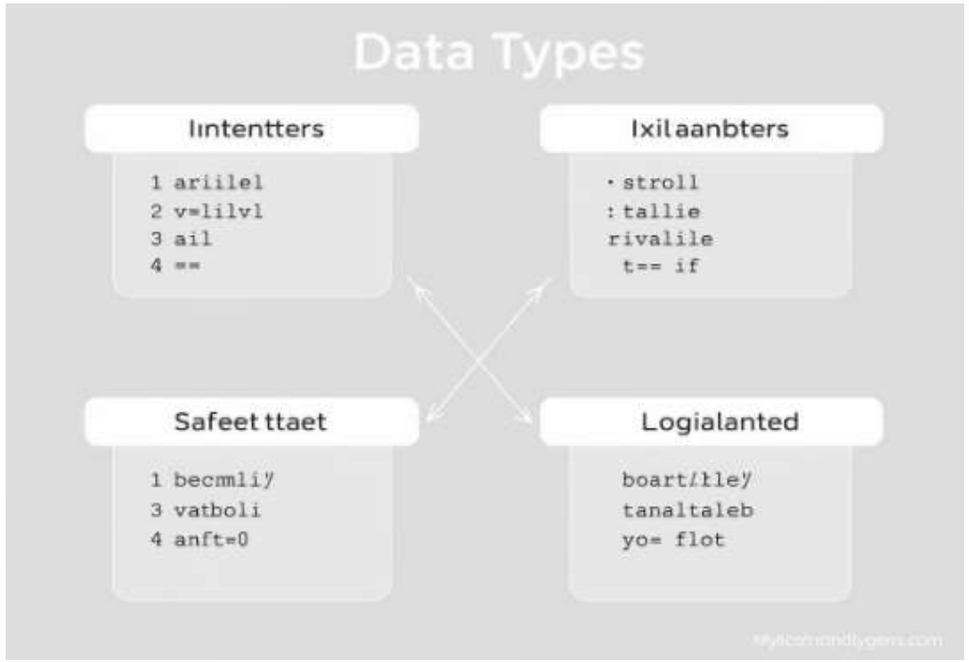
Os números reais são aqueles que podem possuir componentes decimais ou fracionários, positivos ou negativos.

Literal

Dados Literais são sequência de caracteres que podem ser letras, dígitos e símbolos especiais. São representados nos algoritmos, pelo delimitador aspas (") no seu início e término.

Lógico

Dados Lógicos são usados para representar os dois únicos valores lógicos possíveis: Verdadeiro e Falso. Seus pares valores podem representados por meio de outros tipos, como: sim/não, 1/0, true/false.



Exemplos:

- Dados Numéricos Inteiros:

10 - número inteiro positivo

0 - número inteiro

-10 - número inteiro negativo

- Dados Numéricos Reais:

20.05 - número real positivo com duas casas decimais

110. - número real positivo com zero casas decimais

-15.2 - número real negativo com uma casa decimal

0. - número real com zero casas decimais

Exemplos:

V - valor lógico verdadeiro

F - valor lógico falso

Exemplos:

"AbCdefGHi" - literal de comprimento 9

"1.2" - literal de comprimento 3

“0” - literal de comprimento 1

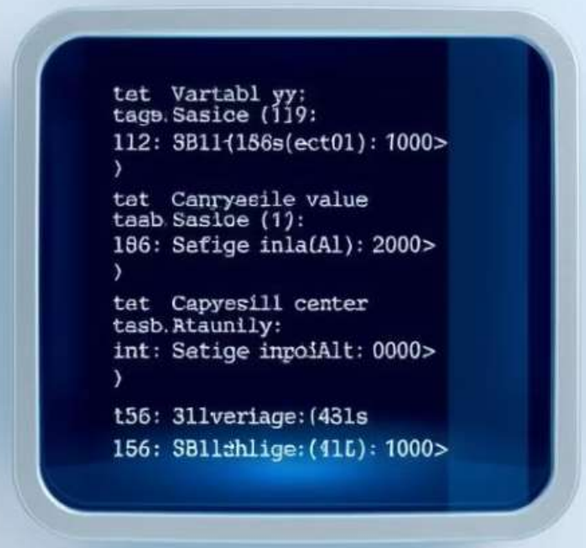
***Note que, "1.2" representa um dado do tipo literal, diferindo de 1.2 que é um dado do tipo real, devido às aspas.**

Constantes

Você sabe o que é uma Constante?

Em programação, uma constante armazena um valor fixo, que **NÃO** mudará com o tempo de execução do programa. Ou seja, o valor será definido uma única vez e jamais será alterado durante a execução da aplicação.

- Uma constante deve ser utilizada quando uma informação **NÃO** tem qualquer possibilidade de alteração
- O valor de uma constante não varia no decorrer da execução do algoritmo (programa)



Variáveis

O que é uma Variável? Agora que você sabe o que é uma Constante, vamos explorar o conceito de Variável. Uma variável é uma entidade destinada a guardar uma informação. Chama-se variável, pois o valor contido nesta varia com o tempo, ou seja, não é um valor fixo.

O conteúdo de uma variável pode ser alterado, consultado ou apagado quantas vezes forem necessárias no algoritmo. Ao alterar o conteúdo de uma variável, a informação anterior é perdida. Ou seja, a variável armazena sempre a última informação recebida.

Em geral, uma variável possui três atributos: nome, tipo de dado e a informação por ela guardada.

Exemplos:

VAR NOME **:literal[50]**

IDADE **:inteiro**

SALARIO **:real**

TEM_FILHOS **:lógico**

Regras para Nomeação de Variáveis

Regras Básicas

- Devem ser iniciadas sempre por uma letra
- Não devem conter caracteres especiais
- Não devem conter espaços em branco
- Não devem conter hífen entre os nomes (utilize underline)

Nome - e começar com uma letra e não deve conter nenhum carácter especial, exceto o underline (_).

Tipos e Informações

Tipo de dados - Pode ser do tipo numérico, literal ou lógico.

Informação - De acordo com o tipo de dado definido.

Atribuição de Valores

É utilizada para atribuir um valor a uma variável, ou seja, para armazenar um determinado conteúdo em uma variável.

A operação de atribuição, geralmente, é representada, nos algoritmos, por uma seta apontando para a esquerda.

Exemplos:

variável ← **constante** Ex.: idade ← 12

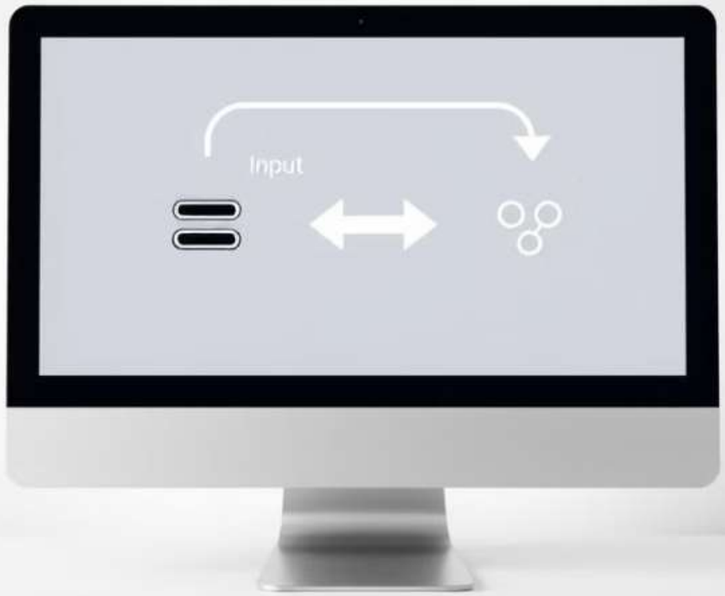
//Variável recebe valor constante

variável ← **variável** Ex.: preço ← valor

//Variável recebe valor de outra variável

variável ← **expressão** Ex.: A ← B + C

//Variável recebe valor de uma expressão



Entrada e Saída de Dados

Instruções Principais

Existem basicamente duas instruções principais em algoritmos que são:

Leia e Escreva

As instruções fundamentais para entrada e saída de dados em algoritmos são: Leia e Escreva.

Leia e Escreva

1

Leia: Entrada de Dados

A instrução Leia é utilizada quando se deseja obter informações do usuário por meio do teclado, ou seja, é um Comando de Entrada de Dados. Usa-se a instrução Leia, quando é necessário que o usuário do algoritmo digite algum dado; A instrução de entrada de dados (Leia) será responsável pela leitura e armazenamento desses dados na variável indicada.

2

Escreva: Saída de Dados

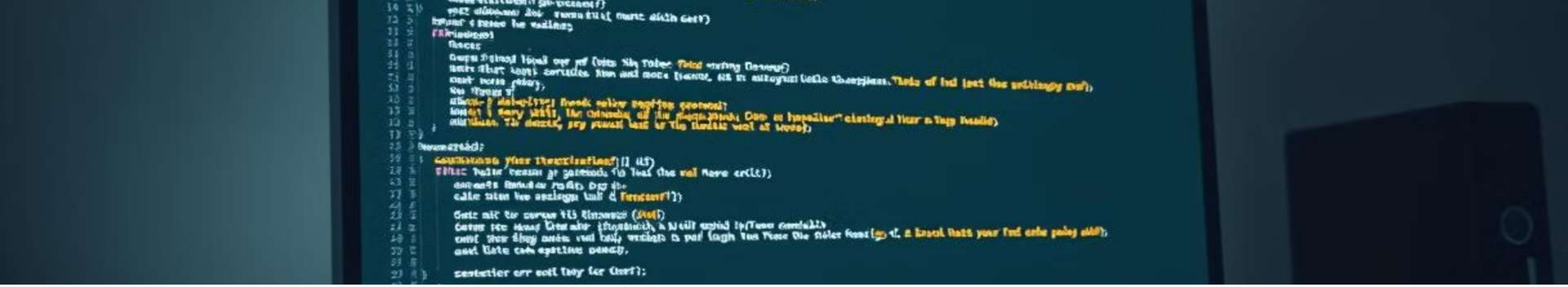
A instrução Escreva é utilizada para mostrar informações na tela do computador, ou seja, é um Comando de Saída de Dados. Usa-se a instrução Escreva quando é necessário mostrar algum dado do algoritmo para o usuário; A instrução de saída de dados (Escreva) será responsável pela exibição dos dados da variável, constante ou expressão na tela do computador.

3

Lendo e Escrevendo Instruções

Lendo instruções: A instrução de entrada de dados (Leia) será responsável pela leitura e armazenamento desses dados na variável indicada.

Escrevendo instruções: A instrução de saída de dados (Escreva) será responsável pela exibição dos dados da variável, constante ou expressão na tela do computador.



Comentários em Algoritmos

Importância dos Comentários

A inserção de comentários no decorrer do algoritmo facilita a leitura deste por outros programadores. Os comentários também servem para auxiliar o programador a relembrar o próprio código depois de um tempo sem utilizá-lo.

Sugestões para Escrita de Algoritmos

- Incluir comentários nas linhas mais importantes do programa
- Utilizar nomes significativos (que ajudem a identificar o conteúdo) para as variáveis e constantes
- Efetuar a indentação (alinhamento) das linhas para facilitar a leitura

Sintaxe:

leia (variável);

Sintaxe:

escreva (variável);

Sintaxe:

//comentário

Algoritmo de exemplo:

```
Algoritmo entrada_saida_dados
Início
   var nome :literal; //Cria a variável nome do tipo literal
   escreva ("Digite seu Nome"); //Solicita que seja digitado o nome
   leia (nome); //Lê e armazena na variável nome o valor digitado
   escreva ("Bom dia", nome); //Escreve a mensagem + nome
Fim
```

- **Sugestões**

  - **Na escrita do algoritmo (pseudocódigo):**
    - Incluir **comentários** nas linhas mais importantes do programa;
    - Utilizar **nomes significativos** (que ajudem a identificar o conteúdo) para as variáveis e constantes;
    - Efetuar a **indentação** (alinhamento) das linhas para facilitar a leitura.

Operadores

O que são Operadores?

Operadores são símbolos que representam atribuições, cálculos e ordem dos dados;
As operações possuem uma ordem de prioridades (alguns cálculos são processados antes de outros);
Os operadores são utilizados nas expressões matemáticas, lógicas, relacionais e de atribuição.

Tipos de Operadores

Quanto ao número de operandos sobre os quais atuam

Unários: quando atuam sobre um único operando.

Binários: quando atuam sobre dois operandos, que podem ser: duas variáveis, duas constantes, ou uma variável e uma constante.

Quanto ao tipo de dado dos operandos e do valor resultante de sua avaliação

- Operadores Aritméticos;
- Operadores de Atribuição;
- Operadores Lógicos;
- Operadores Relacionais.

Exemplos:

Unário:

-x (o valor armazenado no operando x passa a ser negativo)

x++ (incrementa +1 na variável x).

Obs.:

++ significa adicionar +1 ao valor da variável

-- significa diminuir -1 do valor da variável

Binário:

z= x+y (somatória entre as variáveis x e y)

z=x+7 (somatória entre uma variável e uma constante)

Operadores Aritméticos

Operações Básicas

Conjunto de símbolos que representa as operações básicas da matemática como: **somar, subtrair, multiplicar, dividir e etc**.
Esses operadores somente poderão ser utilizados entre variáveis com os tipos de dados numéricos inteiros e/ou numéricos reais.

|×| Lista de Operadores

- Adição +
- Divisão /
- Negativo unário -
- Subtração -
- Restou ou módulo %
- Incremento ++
- Multiplicação *
- Positivo unário +
- Decremento --

Regras de Prioridade

Obedecem às regras matemáticas comuns: As expressões de dentro de parênteses são sempre resolvidas antes das expressões fora dos parênteses; Quando existe um parêntese dentro de outro, a solução sempre inicia do parêntese mais interno até o mais externo (de dentro para fora); Quando duas ou mais expressões tiverem a mesma prioridade, a solução é sempre iniciada da expressão mais à esquerda até a mais à direita. o Obedecem às regras matemáticas comuns:

Operadores Aritméticos		
Adição +	Divisão /	Negativo unário -
Subtração -	Restou ou módulo %	Incremento ++
Multiplicação *	Positivo unário +	Decremento --

- **Obedecem às regras matemáticas comuns:**
  - As expressões de dentro de parênteses são sempre resolvidas antes das expressões fora dos parênteses;
  - Quando existe um parêntese dentro de outro, a solução sempre inicia do parêntese mais interno até o mais externo (de dentro para fora);
  - Quando duas ou mais expressões tiverem a mesma prioridade, a solução é sempre iniciada da expressão mais à esquerda até a mais à direita.

- **Obedecem às regras matemáticas comuns:**

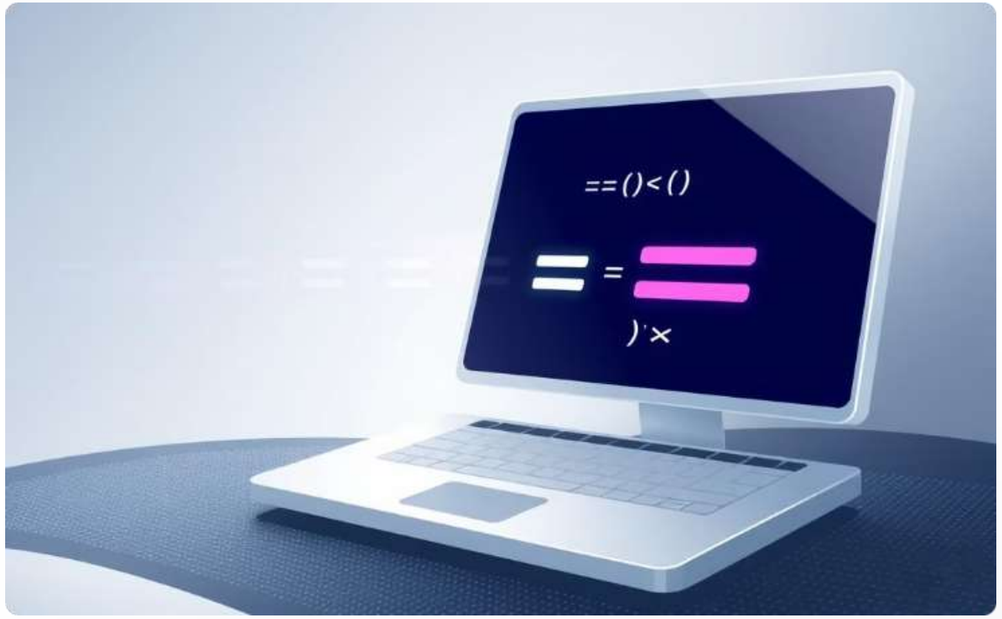
Operador	Operação	Prioridade
^, **	Exponenciação	1
/	Divisão	2
*	Multiplicação	2
+	Adição	3
-	Subtração	3

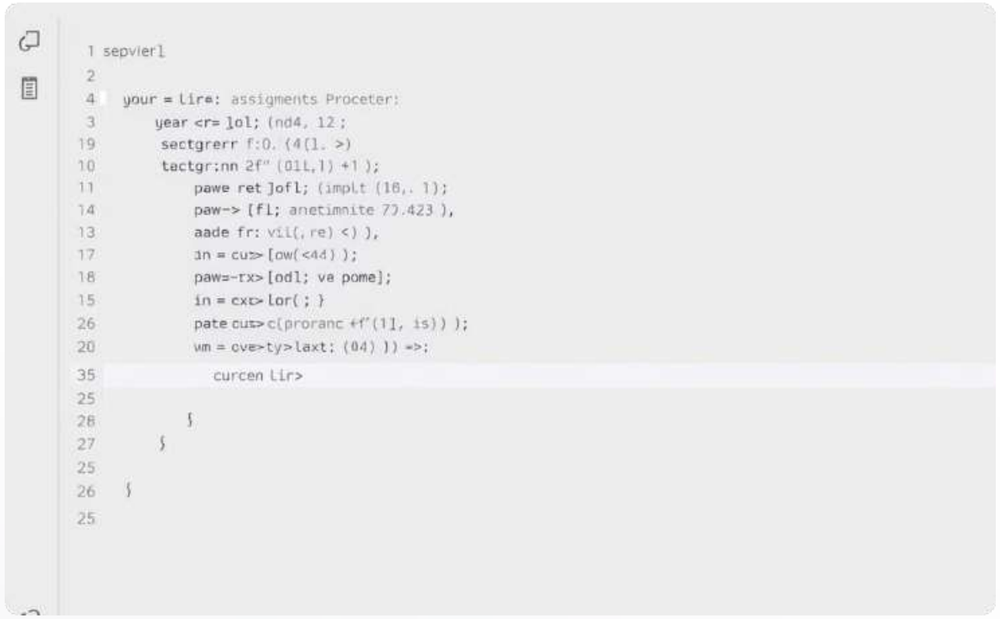
Exemplo:

```
Algoritmo Calculo_Area_Quadrado

var lado, area :real;

Início

   Leia lado;

   area ← (lado * lado);

   Escreva "A área do quadrado é" + area;

Fim
```

Operadores de Atribuição

Têm como função retornar um valor atribuído de acordo com a operação indicada; A operação é feita entre os dois operandos, sendo atribuído o resultado ao primeiro.





Atribuição Simples e Composta

Atribuição simples =

Atribuição com subtração -=

Atribuição com divisão /=

Outros Operadores de Atribuição

Atribuição com adição +=

Atribuição com multiplicação *=

Atribuição com módulo %

▪ Operadores de Atribuição

- Têm como função retornar um valor atribuído de acordo com a operação indicada;
- A operação é feita entre os dois operandos, sendo atribuído o resultado ao primeiro.

Operadores de Atribuição		
Atribuição simples **=**	**Atribuição com subtração** **-=**	**Atribuição com divisão** **/=**
Atribuição com adição **+=**	**Atribuição com multiplicação** ***=**	**Atribuição com módulo** **%=**

Exemplo:

```
Algoritmo Calculo_Area_Circulo

var raio, area :real;

real PI = 3.14;

Início

   Leia raio;

   area ← (pi) * (raio)**2;

   Escreva "A área do círculo é" + area;

Fim
```

Operadores Lógicos

Definição e Função

Operadores lógicos fazem comparações com o objetivo de avaliar expressões em que o resultado pode ser verdadeiro ou falso, ou seja, implementando a lógica booleana. O retorno desta comparação é sempre um valor do tipo booleano (lógico).

Tipos de Operadores Lógicos

- **Conjunção:** e/and/&& - As duas condições devem ser verdadeiras para que o resultado seja verdadeiro.
- **Disjunção:** ou/or/|| - Pelo menos uma condição deve ser verdadeira para que o resultado seja verdadeiro.
- **Negação:** não/not - Inverte o valor do resultado da condição.

Retorno de cada expressão		E	OU
Expressão A	Expressão B	A e B	A ou B
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

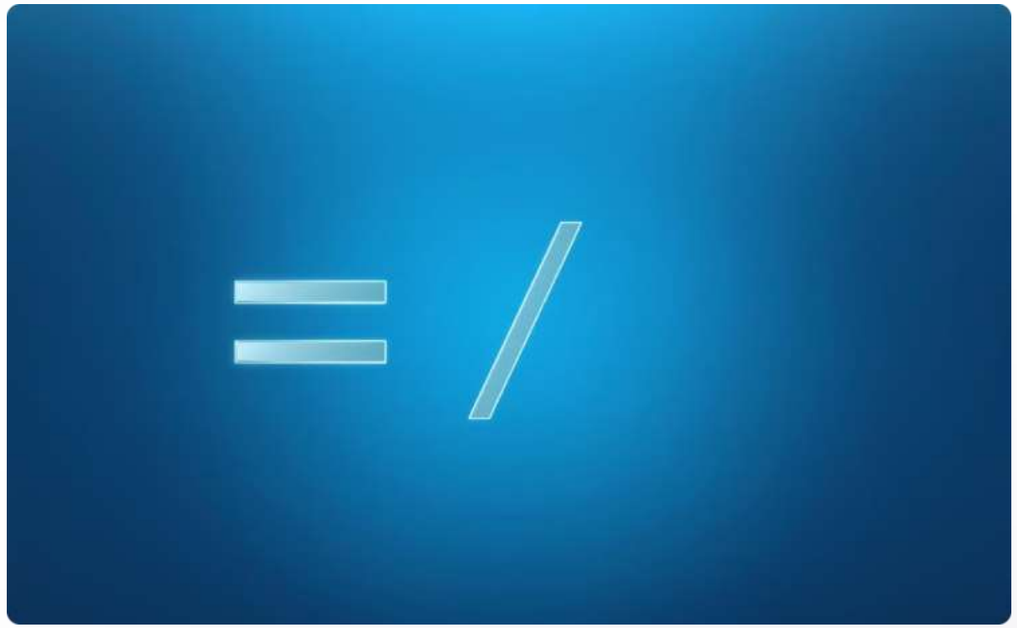
Exemplo:

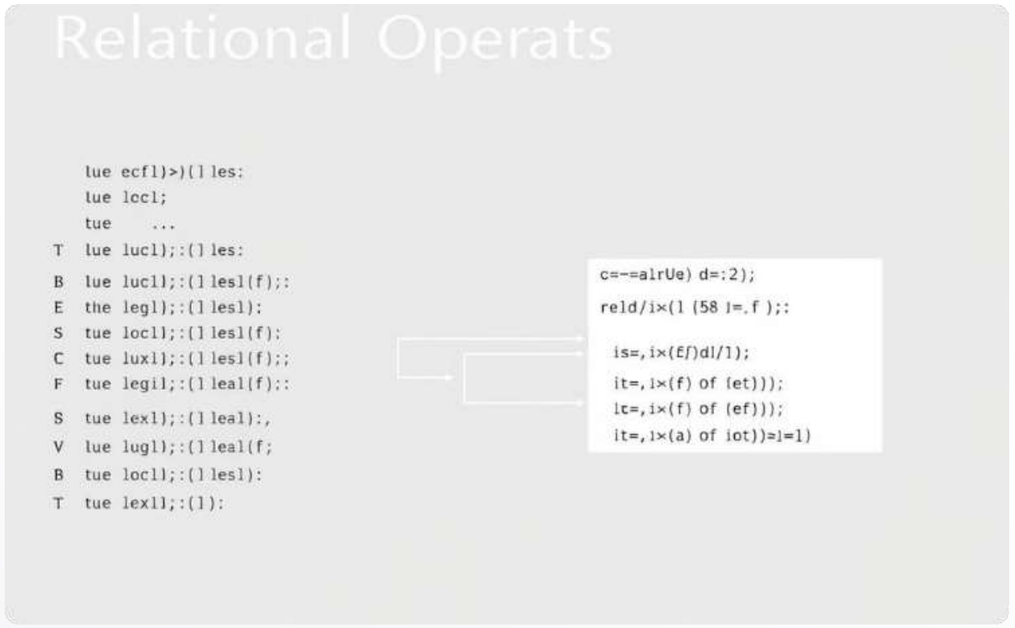
```
Algoritmo Verifica_Aluno_Aprovado
var nota, frequencia :real;
Início
  Leia nota, frequencia;
  if nota >=7 e frequencia >= 70%
    Escreva “Aprovado”;
  else
    Escreva “Reprovado”;
Fim
```

Operadores Relacionais

São utilizados para comparar valores entre variáveis e expressões do mesmo tipo;

O retorno desta comparação é sempre um valor do tipo booleano (verdadeiro/falso).





Símbolos de Comparação

Operadores Relacionais: Igual == Maior > Maior ou Igual >=

Uso em Código

Diferente != ou <> Menor < Menor ou Igual <=

Operadores Relacionais		
Igual **==**	**Maior** **>**	**Maior ou Igual** **>=**
Diferente **!= ou <>**	**Menor** **<**	**Menor ou Igual** **<=**

Exemplo:

```
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
```

Estruturas de Seleção

Você sabe o que são Estruturas de Seleção, também conhecidas como Estruturas Condicionais?

São comandos que auxiliam no direcionamento da sequência de execução de um programa por meio da avaliação de condições lógicas.

Têm como função validar condições e comparar o resultado destas.

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

Para que servem as Estruturas de Seleção?

As Estruturas de Seleção permitem alterar o Fluxo de Execução do algoritmo, de forma a selecionar qual parte deve ser executada. Essa "decisão" de execução é tomada a partir de uma condição, que pode resultar apenas dois valores: verdadeiro ou falso. Uma condição é representada por expressões relacionais ou lógicas.

Tipos de Estruturas de Seleção

Funcionamento

Após executar as funções de validação e comparação, as estruturas de seleção irão executar os blocos de comando, definidos de acordo com o resultado da comparação (verdadeiro ou falso).

Tipos

- If/Else (Se/Então)
- Switch/Case (Escolha/Caso)

ESTRUTURA DE SELEÇÃO IF/ELSE

Classificação

Tipos de estruturas IF/ELSE:

- ✓Simples;
- ✓Compostas;
- ✓Aninhadas.

ESTRUTURA DE SELEÇÃO IF/ELSE

Estruturas de Seleção Simples

Como funciona?

- A condição é verificada a cada passagem pela estrutura IF/SE;
- Se a condição for satisfeita (verdadeira), são executadas as instruções entre chaves (então);
- Se a condição NÃO for satisfeita (falso), as instruções entre chaves não são executadas, sendo executado o código logo após as chaves;
- O IF/SE sempre executará o bloco de comando ou instrução única se a condição entre parênteses retornar um resultado booleano verdadeiro. Caso contrário, o bloco de comando ou a instrução única não serão executadas.

ESTRUTURA DE SELEÇÃO IF/ELSE

Sintaxe no Algoritmo:

```
Se <comandos>
  Então <instruções>;
FimSe
```

Exemplo:

```
Algoritmo verifica_numero
Início
  var x, y :inteiro
  x ← 10
  y ← 20
  Se (x < y) Então
    Escreva "X é menor que Y.";
  FimSe
Fim
```

ESTRUTURA DE SELEÇÃO IF/ELSE

Estruturas de Seleção Composta

Como funciona?

- ✓ A condição é verificada a cada passagem pela estrutura IF/SE;
- ✓ Se a condição for satisfeita (verdadeira), são executadas as instruções entre chaves do IF/SE;
- ✓ Se a condição NÃO for satisfeita (falso), são executadas as instruções dentro das chaves do ELSE/SENÃO;
- ✓ As instruções do ELSE/SENÃO serão executadas somente quando o valor da condição do IF/SE for falso.

ESTRUTURA DE SELEÇÃO IF/ELSE

Sintaxe no Algoritmo:

```
Se <condição> Então
{
   <instruções>;
}
Senão
{
   <instruções>
}
FimSe
```

Exemplo:

```
Algoritmo verifica_numero
Início
   var x, y :inteiro
   x ← 30
   y ← 20
   Se (x < y) Então
      Escreva "X é menor que Y.";
   Senão
      Escreva "X é maior que Y.";
   FimSe
Fim
```

ESTRUTURA DE SELEÇÃO IF/ELSE

Estruturas de Seleção Aninhada

- É utilizada, em geral, quando é necessário realizar várias comparações com a mesma variável;
- É chamada de aninhada porque na sua representação fica uma seleção dentro de outra seleção;
- Também é conhecida como seleção “encadeada”;
- Permite fazer a escolha de apenas um entre vários comandos possíveis.

ESTRUTURA DE SELEÇÃO IF/ELSE

Sintaxe no Algoritmo:

```
Se <condição> Então
  Se <condição> Então
    <instruções>;
  FimSe
Senão
  <instruções>;
FimSe
```

Exemplo:

```
Algoritmo novo_salario
Início
  var salario, novo_salario :real
  Se (salario < 500) Então
    novo_salario < -- salario 1.20;
  Senão
    Se (salario <= 1000) Então
      novo_salario ← salario 1.10;
    Senão
      novo_salario ← salario 1.05;
    FimSe
  FimSe
Fim
```

ESTRUTURA DE SELEÇÃO SWITCH/CASE

Para que serve?

A estrutura **Switch/Case-Escolha/Caso** é utilizada quando é necessário testar a mesma variável com uma série de valores (várias vezes).

ESTRUTURA DE SELEÇÃO SWITCH/CASE

Estrutura padrão

Como funciona?

- ✓ A variável a ser testada deve ser sempre do tipo inteiro ou literal;
- ✓ É utilizado para oferecer várias opções ao usuário, deixando que escolha um valor dentre vários;
- ✓ A principal vantagem desse comando é que ele evita uma série de testes com o comando IF/SE;
- ✓ Funciona de maneira semelhante ao IF/SE encadeado;

ESTRUTURA DE SELEÇÃO SWITCH/CASE

A condição após o SWITCH/ESCOLHA informa o valor que será comparado em cada CASE/CASO;

No primeiro CASE/CASO é verificado se o valor recebido como parâmetro é igual ao seu valor;

Se o valor do parâmetro informado for o mesmo (igual) do CASE/CASO, será executado o trecho de código dentro do respectivo CASE/CASO;

o Se o valor do parâmetro informado for diferente do CASE/CASO, será testada a condição do próximo CASE/CASO;

ESTRUTURA DE SELEÇÃO SWITCH/CASE

O comando BREAK/PARE é utilizado para forçar a saída do SWITCH/ESCOLHA ao se entrar em um CASE/CASO;

Sem o BREAK/PARE todos os CASE/CASO serão testados, mesmo que algum CASE/CASO já tenha atendido a condição;

O comando DEFAULT/SENÃO é opcional e define um fluxo alternativo para as situações não atendidas por nenhum CASE/CASO;

O trecho de código dentro do DEFAULT/SENÃO será executado apenas quando o valor de nenhum CASE/CASO for igual ao valor do parâmetro informado.

ESTRUTURA DE SELEÇÃO SWITCH/CASE

Sintaxe no Algoritmo:

```
Escolha <condição>
  Caso1: <expressão>
    <instruções>;
    Pare;
  Caso2: <expressão>
    <instruções>;
    Pare;
  Senão:
    <instruções>;
    Pare;
FimEscolha
```

Exemplo:

```
Algoritmo informa_sexo
Início
  var sexo :literal
  Escolha (sexo)
    Caso ("F"):
    Escreva "Sexo feminino";
    Pare;
  Caso ("M"):
    Escreva "Sexo masculino";
    Pare;
  FimEscolha
Fim
```

ESTRUTURAS DE REPETIÇÃO

O que são?

- São comandos que permitem que uma sequência de instruções seja executada várias vezes até que uma condição seja satisfeita;
- Se uma instrução ou uma sequência de instruções precisa ser executada várias vezes, deve-se utilizar uma estrutura de repetição.
- Para que servem?
- Servem para repetir um conjunto de instruções sem que seja necessário escrevê-las várias vezes;

ESTRUTURAS DE REPETIÇÃO

O que são?

- Permitem que um trecho do algoritmo seja repetido, em um número determinado ou indeterminado de vezes, sem que o código a ser repetido tenha que ser escrito novamente;
- As estruturas de repetição também são chamadas de Laços ou Loops.

ESTRUTURAS DE REPETIÇÃO

Funcionamento

- As estruturas de repetição envolvem a avaliação de uma condição (teste);
- A avaliação 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;
- Se o resultado da condição for Verdadeiro, é iniciada a repetição ou, caso esteja em execução, é reiniciada a execução das instruções dentro da Estrutura de Repetição;

ESTRUTURAS DE REPETIÇÃO

Funcionamento

- A avaliação da condição é sempre novamente realizada após a execução da última instrução dentro da estrutura de repetição;
- A única Estrutura de Repetição que não realiza a avaliação da condição antes de iniciar é a Do/While (Faça/Enquanto).
- Desta forma, é assegurado que todas as instruções dentro da Estrutura de Repetição do Do/While serão executadas pelo menos uma vez.

ESTRUTURAS DE REPETIÇÃO

Tipos de Estruturas de Repetição

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

ESTRUTURA DE REPETIÇÃO FOR

Características

- ✓ Deve ser usada quando o número exato de repetições é conhecido;
- ✓ Utiliza uma variável de controle que deve ser do tipo Inteiro ou Literal.

ESTRUTURA DE REPETIÇÃO FOR

Estrutura padrão

Como funciona?

- For: comando que inicializa a estrutura de repetição. Sua condição é testada antes de executar qualquer instrução dentro do laço;
- Variável de inicialização: comando de atribuição que inicia uma variável de controle do laço. É executada apenas uma vez, no início do laço;

ESTRUTURA DE REPETIÇÃO FOR

Estrutura padrão

- Condição: determina o final do laço (repetição). Normalmente é uma
- expressão lógica. É verificada antes da execução do laço. Se for Verdadeira, as instruções dentro do laço são executadas. Se for Falsa o laço é finalizado;
- Incremento/decremento: é executado sempre no final do laço, mudando o valor da variável de controle a cada repetição do laço.

ESTRUTURA DE REPETIÇÃO FOR

Sintaxe no Algoritmo:

Para <valor> **Até** <condição> **Faça**

{

<instruções>

}

FimPara

Obs.: Ao invés de **incremento** pode ser feito um **decremento** do valor da **variável de inicialização**.

ESTRUTURA DE REPETIÇÃO FOR

Exemplo:

```
Algoritmo Imprimir_numeros_de_1_a_100

Início

  var contador :inteiro;

  Para contador ← 1 Até 100 Faça

    Escreva (contador);

  FimPara

Fim
```

ESTRUTURA DE REPETIÇÃO WHILE

Estrutura padrão

Características

- É a estrutura de repetição mais simples;
- É ideal para situações em que não se sabe o número exato de vezes em que o bloco de instruções deve ser repetido;
- Pode ser utilizado para substituir laços FOR.

ESTRUTURA DE REPETIÇÃO WHILE

Estrutura padrão

Como funciona?

- A condição é validada antes de cada repetição do laço;
- Enquanto a condição for Verdadeira, o bloco de instruções dentro do laço é executado;
- Quando a condição se torna Falsa, o laço é finalizado.

ESTRUTURA DE REPETIÇÃO WHILE

Sintaxe no Algoritmo:

Enquanto <condição> **Faça**

{

<instruções>

}

FimEnquanto

ESTRUTURA DE REPETIÇÃO WHILE

Exemplo:

```
Algoritmo Imprimir_numeros_de_1_a_10
Início
  var contador :inteiro;
  contador← 1
  Enquanto (contador < 10) Faça
    Escreva contador;
    contador ← contador +1
  FimEnquanto
Fim
```

ESTRUTURA DE REPETIÇÃO DO/WHILE

Características

- ✓ Testa a condição de validação do laço apenas no final do comando. Desta forma, é assegurado 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.

ESTRUTURA DE REPETIÇÃO DO/WHILE

Estrutura padrão

Como funciona?

- Na primeira vez que o laço for executado todas as instruções dentro deste serão executadas, independente da condição estabelecida;
- 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.

ESTRUTURA DE REPETIÇÃO DO/WHILE

Sintaxe no Algoritmo:

```
Faça
{
    <instruções>
} Enquanto <condição>;
```

Exemplo:

```
Algoritmo Imprimir_numeros_de_1_a_10
Início
  var contador :inteiro;
  contador ← 1
  Faz
    Escreva contador;
    contador ← contador +1
  Enquanto (contador < 10);
Fim
```