

**CURSO TÉCNICO EM PROCESSAMENTO
DE DADOS**

APOSTILA DE LÓGICA DE PROGRAMAÇÃO

CAP

Criação de Algoritmos e Programas



PROFESSOR RENATO DA COSTA

"Não estamos aqui para sobreviver e sim para explorar a oportunidade de vencer adquirindo o saber!"

R E N A T O D A C O S T A

SUMÁRIO

PREFÁCIO	5
ALGORITMO	6
ALGORITMO NÃO COMPUTACIONAL	6
PROGRAMA	7
LINGUAGENS DE PROGRAMAÇÃO	7
ALGORITMOS EM “PORTUGOL”	8
TÉCNICAS DE PROGRAMAÇÃO	9
MATEMÁTICA NA INFORMÁTICA	11
OPERADORES ARITMÉTICOS	11
OPERADORES RELACIONAIS	11
LINEARIZAÇÃO DE EXPRESSÕES	12
MODULARIZAÇÃO DE EXPRESSÕES MATEMÁTICAS	12
OPERADORES ARITMÉTICOS ESPECIAIS (MOD E DIV)	13
EXPRESSÕES LÓGICAS	15
OPERADORES LÓGICOS	15
TABELA VERDADE	15
FUNÇÕES	18
BIBLIOTECAS DE FUNÇÕES	18
FUNÇÕES PRÉ-DEFINIDAS	18
TABELA GERAL DE PRIORIDADES	20
VARIÁVEIS	22
VARIÁVEIS DE ENTRADA E SAÍDA	22
IDENTIFICADORES	23
SINAL DE ATRIBUIÇÃO	24
CONSTANTES	24
SINAL DE IGUALDADE	24
TIPOS DE DADOS	25
TIPOS PRIMITIVOS DE DADOS	25

COMANDOS BÁSICOS DE ENTRADA E SAÍDA(INPUT/OUTPUT).....	26
FLUXO DE UM ALGORITMO	30
CORPO GERAL DE UM ALGORITMO	30
ESTRUTURAS SEQUÊNCIAIS.....	31
; PONTO E VÍRGULA ;	31
PRIMEIRO ALGORITMO	32
OUTROS ALGORITMOS DE EXEMPLO	33
{LINHAS DE COMENTÁRIO}.....	35
‘ASPAS SIMPLES’	36
ESTRUTURAS CONDICIONAIS SIMPLES	39
NINHOS DE SE.....	43
ALGORITMO CINCO.....	ERRO! INDICADOR NÃO DEFINIDO.
ESTRUTURAS DE CONDIÇÃO.....	47
ALGORITMO SEIS	ERRO! INDICADOR NÃO DEFINIDO.
ESTRUTURA DE REPETIÇÃO DETERMINADA	52
ALGORITMO SETE.....	ERRO! INDICADOR NÃO DEFINIDO.
ALGORITMO OITO	54
ESTRUTURA DE REPETIÇÃO INDETERMINADA COM VALIDAÇÃO INICIAL	57
ALGORITMO NOVE	57
ESTRUTURA DE REPETIÇÃO INDETERMINADA COM VALIDAÇÃO FINAL	62
ALGORITMO DEZ.....	62
ALGORITMO ONZE.....	63
PROGRAMAS EQUIVALENTES	65
EXERCÍCIOS.....	ERRO! INDICADOR NÃO DEFINIDO.

PREFÁCIO

O trabalho a que me propus é resultado de minha experiência em ministrar a disciplina CAP (criação de Algoritmos e Programas) desde 1996, motivado pela falta de texto relacionado às condições e necessidades do curso.

O objetivo principal da Lógica de Programação é demonstrar técnicas para resolução de problemas e, conseqüentemente, automatização de tarefas.

O aprendizado da Lógica é essencial para formação de um bom programador, servindo como base para o aprendizado de todas as linguagens de programação, estruturadas ou não.

De um modo geral esses conhecimentos serão de supra importância pois ajudarão no cotidiano, desenvolvendo um raciocínio rápido.

Partindo do princípio que "a única coisa constante no mundo é a mudança", forneço abaixo meu endereço eletrônico para que você possa me ajudar, enviando críticas, elogios ou sugestões que servirão para o eterno aprimoramento desse trabalho.

informatica@renatodacosta.net

www.renatodacosta.net

ALGORITMO

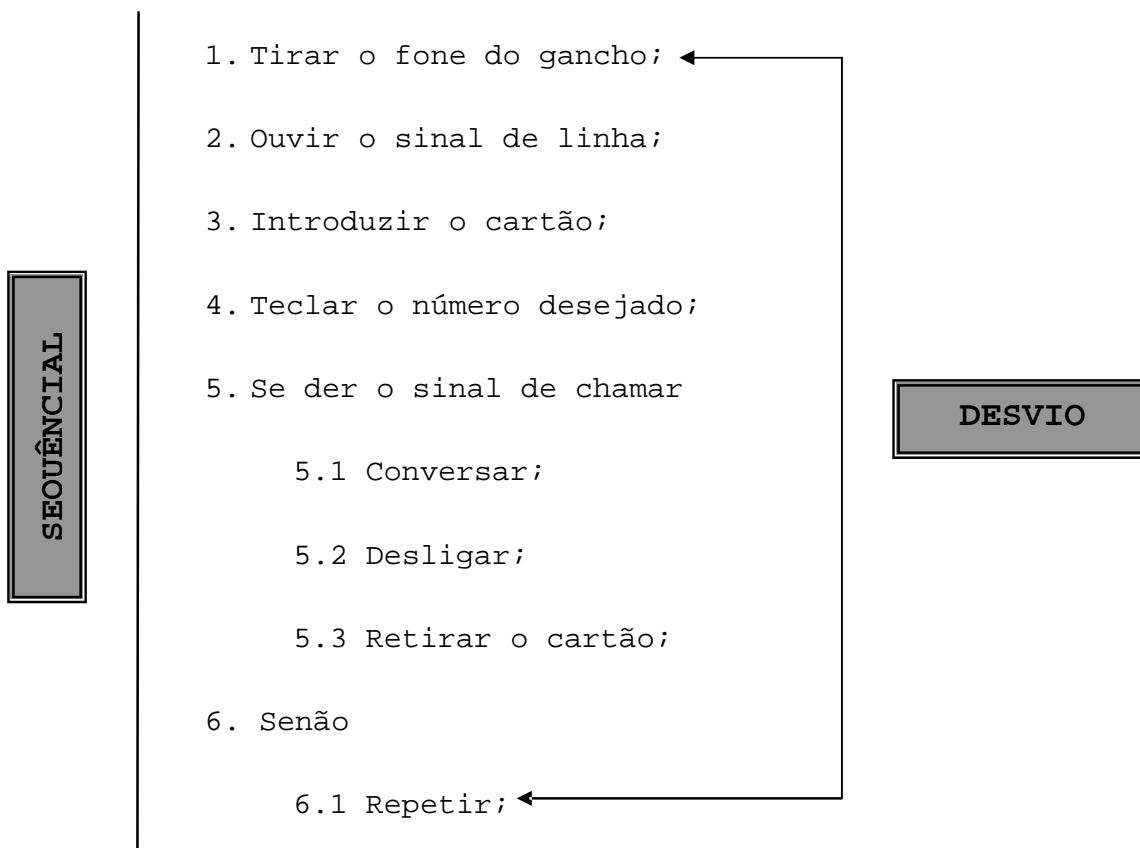
Um Algoritmo é uma seqüência de instruções ordenadas de forma lógica para a resolução de uma determinada tarefa ou problema.

Para criarmos um algoritmo iremos criar um texto estático, ou seja, escrito em uma folha de papel sem sofrer alterações, mas que possui um aspecto dinâmico abrangendo qualquer situação relacionada ao problema em questão.

ALGORITMO NÃO COMPUTACIONAL

Podemos criar algoritmos para tarefas do cotidiano, no exemplo abaixo é apresentado um Algoritmo não computacional cujo objetivo é usar um telefone público.

Início



Fim.

Para criamos um algoritmo devemos observar ou entender primeiro o padrão de comportamento de um processo e depois passa-lo para o papel.

PROGRAMA

Um programa nada mais é do que um algoritmo escrito em uma linguagem computacional.

Aprender uma linguagem de programação sem dominar a criação de algoritmos não faz sentido, seria o mesmo que ter um vocabulário vasto, mas não saber usar as palavras certas na hora certa. Sabendo algoritmo é relativamente fácil criar um programa em qualquer linguagem, basta pesquisar qual comando realiza a ação desejada.

LINGUAGENS DE PROGRAMAÇÃO

São Softwares básicos que permitem o desenvolvimento de programas em uma linguagem de alto nível semelhante a que pensamos ou escrevemos e que posteriormente são convertidas para uma linguagem de máquina interpretada pelo computador.

Possuem um poder de criação ilimitado, desde jogos, editores de texto, planilhas eletrônicas até sistemas operacionais. Existem várias linguagens de programação, cada uma com suas características de linguagem próprias.

Exemplos: Pascal, Clipper, C, Visual Basic, Delphi, Java entre outras.

Observação:

Existem linguagens que criam arquivos executáveis e outras que criam programas interpretados pelo próprio ambiente de desenvolvimento, implicando sempre no uso do mesmo para execução do programa.

Ex: dBase

ALGORITMOS EM "PORTUGOL"

Durante nosso aprendizado, iremos aprender a desenvolver nossos algoritmos em uma pseudolinguagem conhecida como "Portugol" ou Português Estruturado.

"Portugol" é derivado da aglutinação de Português + Algol. Algol é o nome de uma linguagem de programação usada no final da década de 50.

Vale ressaltar que não existe um padrão para os comandos usados nos algoritmos, cada professor, cada autor explica como acha mais fácil. Convenhamos que o mais importante é a lógica, ou seja, o pensamento elaborado e não a linguagem utilizada. Eu particularmente gosto e adotei uma linguagem algorítmica muito próxima de um Pascal traduzido. Por que? A linguagem Pascal é didática, dispõe de todos os recursos para o desenvolvimento de bons hábitos para criação de programas, além de ainda ser muito utilizada nas universidades atuais.

Existem ainda outras formas de se criar algoritmos, como através de fluxogramas onde cada instrução é representada por um desenho. Apesar dos fluxogramas darem uma boa visualização

do processo torna-se muito confuso e de difícil implementação quando usados para detalhar tarefas muito complexas.

Curiosidade:

A linguagem Pascal foi criada no início da década de 70 por Niklaus Wirth e possui esse nome em homenagem ao filósofo Blaise Pascal inventor da máquina de calcular mecânica em 1642.

Técnicas de Programação

Programação seqüencial: o programa é descrito através de várias linhas, executadas uma após a outra.

Programação estruturada: dispõe da possibilidade de dividir o programa em subprogramas (procedimentos ou funções) diminuindo o tempo de programação para tarefas repetitivas, facilitando a manutenção do mesmo e minimizando os erros.

Programação orientada a eventos: o programa na verdade é dividido em várias partes agregadas a objetos utilizados pelo ambiente da linguagem e esses trechos do programa são despertados por ações determinadas como um clique em um botão ou uma fatia de tempo.

Exercícios:

- 1) Por quê é importante estudar algoritmo?
- 2) Crie um algoritmo não computacional que descreva como trocar um pneu de um carro.
- 3) O que é um programa?
- 4) Cite 3 linguagens de programação atuais:
- 5) É necessário ter um computador para criar Algoritmos?
- 6) Por que podemos dizer que um algoritmo é estático e possui aspecto dinâmico?

MATEMÁTICA NA INFORMÁTICA

Como a maioria dos programas possui algum tipo de expressão matemática, iremos começar a estudar seus operadores.

OPERADORES ARITMÉTICOS

+ → Adição

- → Subtração

* → Multiplicação

/ → Divisão

^ ou ** → Exponenciação ex. $2^3 = 2 \wedge 3$ ou $2 ** 3$

Qual o resultado da expressão abaixo?

$2+2/2$

Lembre-se que a prioridade dentre os operadores descritos anteriormente é a mesma da matemática, primeira a exponenciação seguido da multiplicação e divisão e por último a soma e subtração. Logo o resultado é 3.

OPERADORES RELACIONAIS

> → Maior que

< → Menor que

>= → Maior ou Igual

<= → Menor ou Igual

= → Igual

<> → Diferente

LINEARIZAÇÃO DE EXPRESSÕES

Para a construção de Algoritmos todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linhas.

É importante também ressaltar o uso dos operadores correspondentes da aritmética tradicional para a computacional.

Exemplo:

$\left[\frac{2}{3} + (5-3) \right] + 1 =$		$(2/3 + (5-3)) + 1 =$
Tradicional		Computacional

MODULARIZAÇÃO DE EXPRESSÕES MATEMÁTICAS

A modularização é a divisão da expressão em partes, proporcionando maior compreensão e definindo prioridades para resolução da mesma.

Como pode ser observado no exemplo anterior, em expressões computacionais usamos somente parênteses "()" para modularização.

Na informática podemos ter parênteses dentro de parênteses.

Exemplos de prioridades:

$$(2+2)/2=2$$

Primeiro resolve-se o que está em parênteses.

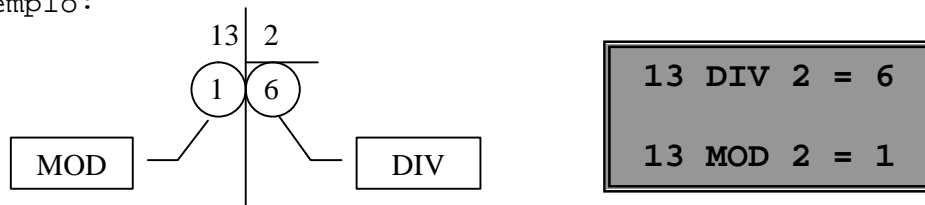
$$2+2/2=3$$

OPERADORES ARITMÉTICOS ESPECIAIS (MOD e DIV)

MOD → Retorna o resto da divisão entre 2 números **inteiros**.

DIV → Retorna o valor inteiro que resulta da divisão entre 2 números **inteiros**.

Exemplo:



Observação:

Como foi bem destacado acima nunca poderemos ter uma expressão tipo: $2,8 \text{ mod } 2$, pois $2,8$ não é um número inteiro.

Observe:

$$- 8 * 3 + 7 \text{ mod } 2 + 6 * 9$$

$$- \text{Calculando: } 24 + 1 + 54 = 79$$

Observação: A prioridade dos operadores especiais é igual a da multiplicação ou divisão.

Exercícios:

1) Calcule as expressões aritméticas abaixo:

a) $75 / 5 * 3 + 6 \text{ mod } 2 * 1,87$

b) $7 \text{ div } 2 + 6 \text{ mod } 2 * 5 + 6 / 6$

c) $5 * 2 ** 3 + 7 ** 2 * 3$

2) Sabendo que $A=3$, $B=7$, $C=8$, $D=42$, $G=5$, $H=-6$, calcule:

a) $- A * B * C \text{ div } D + G + H$

EXPRESSÕES LÓGICAS

As expressões compostas de relações baseadas em uma proposição sempre resultam em um valor lógico do tipo Verdadeiro ou Falso.

Exemplos:

$2+5>4 \rightarrow$ Verdadeiro

$3<>3 \rightarrow$ Falso

OPERADORES LÓGICOS

Atuam sobre expressões lógicas retornando resultados do tipo Falso ou Verdadeiro.

E	RETORNA VERDADEIRO, SE AMBAS AS PARTES DA EXPRESSÃO FOREM VERDADEIRAS.
OU	BASTA QUE UMA PARTE DA EXPRESSÃO SEJA VERDADEIRA PARA RETORNAR VERDADEIRO.
NÃO	INVERTE O ESTADO, DE VERDADEIRO PASSA PARA FALSO E VICE-VERSA.

Prioridades dos operadores Lógicos:

- a) NÃO \rightarrow Negação
- b) E \rightarrow Conjunção
- c) OU \rightarrow Disjunção

TABELA VERDADE

Supondo A e B como expressões lógicas vamos verificar os estados de cada linha da tabela abaixo:

A	B	A E B	A OU B	NÃO (A)
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

NÃO(3<>3) → Verdadeiro



Exercícios:

1) Responda o resultado lógico (V ou F) das expressões abaixo, sabendo que $A = V$, $B = V$, $C = F$, $D = V$, $G = V$.

- a) $A \text{ e } B \text{ e } C \text{ ou } D \text{ e não } G$
- b) $(A \text{ ou } B) \text{ e } C \text{ e } D \text{ e não } G \text{ ou } H$

2) Monte as tabelas verdade das expressões abaixo:

- a) $A \text{ ou } B \text{ e não } C$
- b) $A \text{ ou não } B \text{ e } C$
- c) $A \text{ e } B \text{ ou não } A$
- d) $A \text{ e } B \text{ ou } C \text{ ou não } B$

OBS: Quando temos uma coluna em uma tabela verdade com todos os valores verdadeiros chamamos de Tautologia e quando todos os valores são falsos chamamos de contradição.

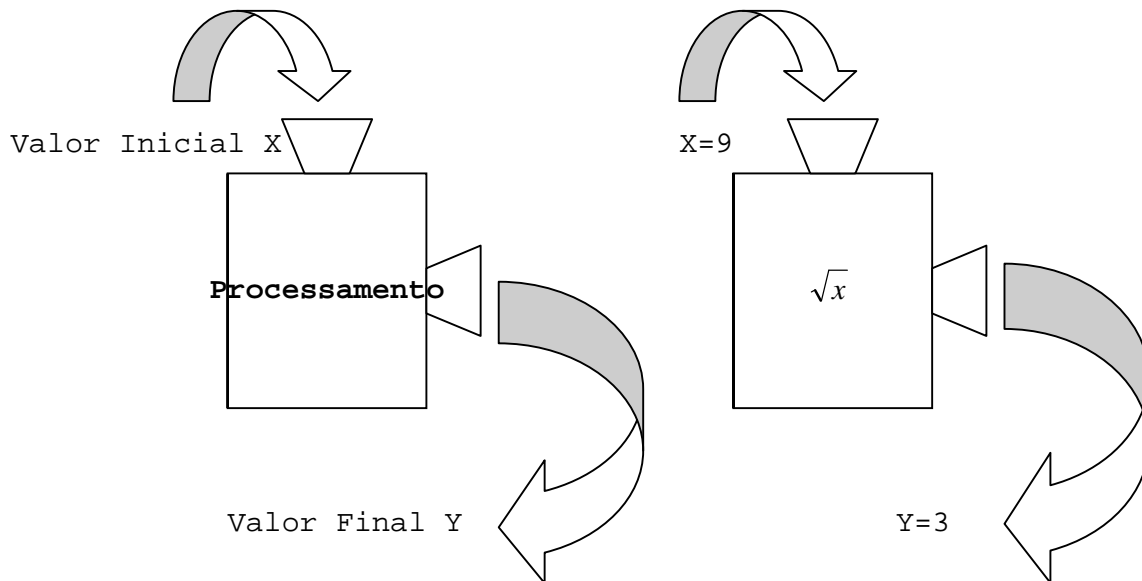
FUNÇÕES

Uma função é um instrumento (Sub-algoritmo) que tem como objetivo retornar um valor ou uma informação.

A chamada de uma função é feita através da citação do seu nome seguido opcionalmente de seu argumento inicial entre parênteses.

As funções podem ser predefinidas pela linguagem ou criadas pelo programador de acordo com o seu interesse.

Exemplos:



BIBLIOTECAS DE FUNÇÕES

Armazenam um conjunto de funções que podem ser usadas pelos programas.

FUNÇÕES PRÉ-DEFINIDAS

ABS()	VALOR ABSOLUTO	x
--------	----------------	---

SQRT()	RAIZ QUADRADA \sqrt{x} (Square Root)
SQR()	ELEVA AO QUADRADO x^2
TRUNC()	VALOR TRUNCADO Ex. trunc(7,9)=7 ou trunc(7,1) =7
ROUND()	VALOR ARREDONDADO Ex. Round() de 7,0 até 7,4 = 7 e de 7,5 até 7,9 = 8
LOG()	LOGARITMO
SIN()	SENO
COS()	COSENO
TAN()	TANGENTE

As funções acima são as mais comuns e importantes para nosso desenvolvimento lógico, entretanto, cada linguagem possui suas funções próprias. As funções podem ser aritméticas, temporais, de texto etc.

TABELA GERAL DE PRIORIDADES

PRIMEIRO	PARÊNTESES E FUNÇÕES
SEGUNDO	SINAIS DE MENOS E MAIS PARA OPERANDOS UNÁRIOS.
TERCEIRO	EXPONENCIAL
QUARTO	MOD, DIV, MULTIPLICAÇÃO E DIVISÃO
QUINTO	SOMA E SUBTRAÇÃO
SEXTO	OPERADORES RELACIONAIS
SÉTIMO	NÃO
OITAVO	E
NONO	OU

Exercícios:

1) Associe as colunas abaixo:

- | | |
|------------------------------------|---------|
| (a) Operador Aritmético | () não |
| (b) Operador Aritmético Especial | () >= |
| (c) Operador Relacional | () = |
| (d) Operador Lógico | () mod |
| | () ^ |
| | () div |
| | () <> |
| | () ** |

2) Assinale com X a expressão abaixo que está incorreta:

- () A ou não B
- () A não B ou C
- () A ou não B e C

3) Escreva os Resultados:

- a) `trunc(9.8888)*2`
- b) `round(2.5)**3`
- c) `(sqrt(81)*5)/5`
- d) `(sqrt(1000)*2,5+72**4)/0`
- e) `(10+5+9+7)/4`
- f) `abs(-98)*2+6`

VARIÁVEIS

Variáveis são endereços de memória destinados a armazenar informações temporariamente (durante a execução do algoritmo). Embora uma variável possa assumir diferentes valores, ela só pode armazenar um único valor a cada instante.

* Todo Algoritmo ou programa deve possuir variável!

Por exemplo, imagine que eu quero saber o dobro da sua idade. A fórmula seria: $\text{resposta} = \text{idade} * 2$.

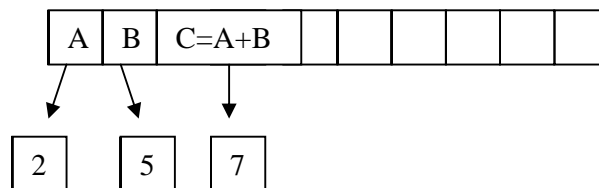
Neste exemplo temos 2 variáveis: a idade, que vai variar para cada leitor e a resposta que vai variar de acordo com a idade.

VARIÁVEIS DE ENTRADA E SAÍDA

Variáveis de Entrada armazenam informações fornecidas por um meio externo, normalmente usuários ou discos.

Variáveis de Saída armazenam dados processados normalmente dados intermediários ou resultados.

Exemplo:



De acordo com a figura acima A e B são Variáveis de Entrada e C é uma Variável de Saída.

IDENTIFICADORES

São os nomes significativos dados a variáveis, constantes e programas.

Regras Para construção de Identificadores:

- Não podem ter nomes de palavras reservadas (comandos da linguagem);
- Devem possuir como 1º caractere uma letra ou Underscore (_);
- Ter como demais caracteres letras, números ou Underscore;
- Ter no máximo 127 caracteres;
- Não possuir espaços em branco;
- A escolha de letras maiúsculas ou minúsculas é indiferente.

Exemplos:

NOME	TELEFONE	IDADE_FILHO
NOTA1	SALARIO	PI
UMNOMEMUITOCOMPRIDOEDIFICILDELER		
UM_NOME_MUITO_COMPRIDO_E_FACIL_DE_LER		

SINAL DE ATRIBUIÇÃO

Uma Variável nunca é eternamente igual a um valor, seu conteúdo pode ser alterado a qualquer momento. Portanto para atribuir valores a variáveis devemos usar o sinal de ~~":="~~ ou " \leftarrow ".

Exemplos:

A ~~:=~~ \leftarrow 2;

B \leftarrow 3; (lê-se da seguinte forma: B recebe 3)

C ~~:=~~ \leftarrow A + B;

Obs.: Dependendo da linguagem de programa;ao em que o algoritmo for implementada, esta pode utilizar como sinal de atribuição ~~":="~~ ou simplesmente o sinal de "=".

CONSTANTES

Assim como as variáveis, as constantes são endereços de memória destinados a armazenar informações, entretanto elas são fixas, inalteráveis durante a execução do programa.

Exemplo:

PI = 3.1416

SINAL DE IGUALDADE

As constantes são eternamente iguais a determinados valores, portanto, **quando construímos um algoritmo**, usamos o sinal de "=" **para identifica-las**.

Exemplos:

PI = 3.1416;

Empresa = 'Colégio de Informática L.T.D.A.'

V = Verdadeiro

TIPOS DE DADOS

Todas as Variáveis devem assumir um determinado tipo de informação.

O tipo de dado pode ser:

- Primitivo → Pré-definido pela linguagem;
- Sub-Faixa → É uma parte de um tipo já existente;
- Escalar → Definidos pelo programador.

Exemplos:

A : INTEIRO

PRIMITIVO

TIPO NOTA=[1..10] DE INTEIRO

SUB - FAIXA

TIPO SEMANA = (Segunda-feira, Terça-feira, Quarta-feira, Quinta-feira, Sexta-feira, Sábado, Domingo)

ESCALAR

TIPOS PRIMITIVOS DE DADOS

INTEIRO	ADMITE SOMENTE NÚMEROS INTEIROS. GERALMENTE É UTILIZADO PARA REPRESENTAR UMA CONTAGEM (QUANTIDADE).
REAL	ADMITE NÚMEROS REAIS (COM OU SEM CASAS DECIMAIS). GERALMENTE É UTILIZADO PARA REPRESENTAR UMA MEDIÇÃO.

CARACTERE	ADMITE CARACTERES ALFANUMÉRICOS. OS NÚMEROS QUANDO DECLARADOS COMO CARACTERES TORNAM SE REPRESENTATIVOS E PERDEM A ATRIBUIÇÃO DE VALOR.
LÓGICO	ADMITE SOMENTE VALORES LÓGICOS(VERDADEIRO/FALSO).

COMANDOS BÁSICOS DE ENTRADA E SAÍDA(INPUT/OUTPUT)

LER → Comando de entrada que permite a leitura de Variáveis de Entrada. Alguns autores tratam esse comando com Receber, o resultado é o mesmo.

ESCREVER → Comando de saída que exibe uma informação na tela do monitor. Alguns autores tratam esse comando com Exibir.

IMPRIMIR → Comando de saída que envia uma informação para a impressora.

Exemplos:

Imagine que queremos obter um número do usuário e guardar em uma variável chamada NUM.

```
Ler (num);
```

Agora queremos pegar esse valor e calcular o dobro dele e guardar esse valor na variável DOBRO.

```
Dobro ← num * 2;
```

Para exibir o resultado seria:

Escrever (dobro)

Até que não é tão difícil não é mesmo?

Exercícios:

1) Responda:

- a) Represente a entrada de um dado pelo teclado guardando-o na variável DADO.
- b) Represente a atribuição do valor 1000 para a variável X.
- c) Defina um novo valor para variável X, como sendo seu valor anterior acrescido de 1.
- d) Defina uma variável K como sendo o resto da divisão de A por B.
- e) Represente a saída da questão anterior.

2) Assinale com X os identificadores válidos:

- | | | |
|-------------------------------------|------------------------------------|--|
| <input type="checkbox"/> valor | <input type="checkbox"/> x2 | <input type="checkbox"/> salario-liquido |
| <input type="checkbox"/> nota aluno | <input type="checkbox"/> b248765 | <input type="checkbox"/> alv7c9 |
| <input type="checkbox"/> 3 x 9 | <input type="checkbox"/> Renato | <input type="checkbox"/> ah! |
| <input type="checkbox"/> "oi" | <input type="checkbox"/> MuitoFeio | <input type="checkbox"/> quase(x) |

3) Identifique os tipos das variáveis abaixo:

- a) nome
- b) cep
- c) telefone
- d) idade

e) quantidade

f) peso

g) altura

h) salário

i) cor

j) tem_carro

FLUXO DE UM ALGORITMO

Todo Algoritmo é composto de um fluxo básico:

Entrada → Processamento → Saída

CORPO GERAL DE UM ALGORITMO

Iremos aprender o corpo geral de um algoritmo passo a passo, entendendo cada área do mesmo.

Todo algoritmo precisa ter a primeira linha como uma identificação (nome) de acordo com o objetivo proposto:

ALGORITMO <<identificador>>;

Em seguida podemos declarar as constantes, que são sempre opcionais.

CONST

Declaramos o nome da constante e seu valor:

<<identificador>> = <<dado>>;

A declaração de variáveis é praticamente obrigatória, pois um programa sem variáveis só poderia existir para saída de informações, o que não é muito comum.

VAR

Devemos dar um nome a variável e definir o seu tipo (inicialmente iremos aprender usando apenas tipos primitivos de dados). Podemos colocar uma variável em cada linha ou declarar muitas em uma mesma linha separando as por vírgulas, desde que elas sejam de mesmo tipo.

<<identificador1>> : <<tipo>>;

<<identificador1>> : <<tipo>>;

Finalmente iremos colocar a palavra reservada que determina o início do algoritmo, ela irá agrupar vários comandos.

ÍNICIO

Aqui podemos escrever os comandos de entrada e saída de dados, as fórmulas e os demais procedimentos.

<<comando1>>;

<<comandoN>>

Após o bloco de comandos iremos fechar o algoritmo com a respectiva palavra reservada.

FIM.

ESTRUTURAS SEQÜENCIAIS

Como pode ser analisado no tópico anterior, todo programa possui uma estrutura seqüencial (seqüência de comandos) determinada por um ÍNICIO e FIM.

; PONTO E VÍRGULA ;

O sinal de ponto e vírgula ";" indica a existência de um próximo comando (passa para o próximo).

Na estrutura ÍNICIO e no comando que antecede a estrutura FIM não se usa ";".

PRIMEIRO ALGORITMO

Segue um algoritmo que vai receber dois números inteiros digitados pelo usuário e calcular a soma.

```
ALGORITMO SOMA;  
  
VAR  
  
    NUMERO1, NUMERO2, SOMA: INTEIRO;  
  
INICIO  
    |  
    | LER (NUMERO1);  
    |  
    | LER (NÚMERO2);  
    |  
    | SOMA ← NUMERO1+NUMERO2;  
    |  
    | ESCREVER (SOMA)  
    |  
FIM.
```

Observe que o algoritmo acima demonstra bem o fluxo definido anteriormente. Primeiro é feita a entrada de dados (leitura de variáveis), depois o processamento (cálculo da soma) e em seguida a saída de dados (exibição da soma obtida no processamento).

Agora se quiséssemos criar um programa baseado nesse algoritmo precisaríamos apenas estudar quais palavras reservadas da linguagem desejada exercem as funções desejadas pelo algoritmo.

Observe os exemplos:

Em Pascal:

```
PROGRAM EXEMPLO:  
  
VAR  
  
    NUMERO1, NUMERO2, SOMA: INTEGER;  
  
BEGIN  
    |  
    |   READ (NUMERO1);  
    |  
    |   READ (NÚMERO2);  
    |  
    |   SOMA ← NUMERO1+NUMERO2;  
    |  
    |   WRITE (SOMA)  
    |  
END.
```

OUTROS ALGORITMOS DE EXEMPLO

Segundo exemplo:

Segue um Algoritmo que lê o nome e as 4 notas bimestrais de um aluno. Em seguida o Algoritmo calcula e escreve a média obtida.

```
ALGORITMO MEDIA_FINAL;
```

```
VAR
```

```
    NOTA1, NOTA2, NOTA3, NOTA4, MEDIA: REAL;
```

```
    NOME : CARACTERE [35] {podemos ou não definir o tamanho de caracteres que uma variável desse tipo pode assumir}
```

```
INICIO
```

```
LER (NOME);  
  
LER (NOTA1, NOTA2, NOTA3, NOTA4);
```

```
MEDIA := (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4;
```

```
ESCREVER (NOME);  
  
ESCREVER (MEDIA);
```

FIM.

Observe, sempre mantendo o fluxo (envolvido pelos retângulos) entrada, processamento e saída.

Agora vamos criar um algoritmo que utilize uma constante.

Criaremos um Algoritmo que lê o raio de uma circunferência e calcula sua área. Sabendo que a área da circunferência é igual ao valor de π multiplicado pelo quadrado do raio, sendo $\pi = 3,1416$.

ALGORITMO AREA_CIRCUNFERENCIA;

CONST

```
PI = 3.1416;
```

VAR

```
RAIO, AREA : REAL;
```

INICIO

```
LER (RAIO);  
  
AREA ← PI * RAIO**2;
```

```
    |
    |     ESCREVER (AREA)
    |
FIM.
```

Para concluirmos nossos algoritmos seqüenciais vamos fazer um quem tenha uma variável auxiliar.

Iremos criar um algoritmo que leia 2 números inteiros A e B, troque seu conteúdo e os exiba.

ALGORITMO TROCATUDO;

VAR

A,B, AUXILIAR: INTEIRO;

INICIO

```
    |
    |     LER(A);
    |
    |     LER (B);
    |
    |     AUX←A;
    |
    |     A←B;
    |
    |     B←A;
    |
    |     ESCREVER (A,B)
    |
FIM.
```

{LINHAS DE COMENTÁRIO}

Podemos inserir em um Algoritmo comentários para aumentar a compreensão do mesmo, para isso bastam que os comentários fiquem entre Chaves "{}".

Exemplo:

```
LER (RAIO); {ENTRADA}
```

'ASPAS SIMPLES'

Quando queremos exibir uma mensagem para a tela ou impressora ela deve estar contida entre aspas simples, caso contrário, o computador irá exibir o conteúdo de uma variável ou identificar a mensagem como Variável Indefinida.

Exemplo:

```
AREA←180
```

```
ESCREVER ('AREA OBTIDA =', AREA) {COMANDO DE SAÍDA}
```

```
AREA OBTIDA = 180 {RESULTADO GERADO NA TELA}
```

EXERCÍCIOS:

- 1) Crie os algoritmos abaixo:
 - a) Leia o lado de um quadrado e calcule sua área.
 - b) Leia a base e a altura de um triângulo e calcule sua área.
 - c) Leia 3 números inteiros e calcule a soma dos mesmos.
 - d) Leia uma despesa gasta em uma mesa de restaurante e calcule os 10%.
 - e) Obtenha uma distância percorrida por um carro em km e a quantidade de tempo gasta em horas e calcule a velocidade média em km/h.
 - f) Leia uma temperatura dada em graus celsius e escreva a correspondente em graus farenheit.
 - g) Obtenha os valores dos catetos de um triângulo retângulo em cm e escreva o valor de sua hipotenusa, também em cm.
 - h) Leia um número, calcule e escreva seu cubo.
 - i) Ler o número de balas que vem em um saco, o valor unitário da bala e calcular o valor do saco.
 - j) Ler o lado de um quadrado e escrever seu perímetro.

k) Ler dois números inteiros e exibir a sua soma, o módulo da diferença entre eles, seu produto e o resto da divisão dos mesmos.

2) Dado o algoritmo abaixo, escreva o valor de X:

ALGORITMO DESAFIO1;

VAR

X, Y, Z: REAL;

INICIO

Y ← 10; Z ← 14; X ← 4;

Y ← Z MOD X + Y * Z + 2;

X ← SQRT(Y) ;

ESCREVER (X)

FIM.

ESTRUTURAS CONDICIONAIS SE

Executa uma seqüência de comandos de acordo com o resultado de um teste.

A estrutura de decisão SE pode ser Simples ou Composta, baseada em um resultado lógico, a partir daí uma alternativa será executada.

Simples:

SE <<CONDIÇÃO>> ENTÃO

<<COMANDO1>> ;

<<COMANDON>>

Só são executados se a condição for verdadeira!!!

FIM-SE

Composta:

SE <<CONDIÇÃO>> ENTÃO

<<COMANDO1>> ;

<<COMANDON>>

São executados se a condição for verdadeira!!!

SENÃO

<<COMANDO1>> ;

<<COMANDON>> ;

Só são executados se a condição for FALSA!!!

FIM-SE

Obs.: Em algumas linguagens de programação não usa-se ponto e virgula(;) antes do comando ELSE, representado no algoritmo pela instrução SENA0.

Exemplos:

Segue um Algoritmo que lê 2 números e escreve o maior.

ALGORITMO ACHA_MAIOR;

VAR A, B: INTEIRO;

INICIO

LER (A, B);

SE A>B ENTÃO

ESCREVER (A)

SENÃO

ESCREVER (B)

FIM-SE

FIM.

Segue um Algoritmo que lê o nome e as 4 notas bimestrais de um aluno. Em seguida o Algoritmo calcula e escreve a média obtida pelo aluno escrevendo também se o aluno foi aprovado ou reprovado.

Média para aprovação = 6

ALGORITMO MEDIA_FINAL;

VAR

NOTA1, NOTA2, NOTA3, NOTA4, MEDIA: REAL;

NOME : CARACTERE [35]

INICIO

```
LER (NOME) ;
```

```
LER (NOTA1, NOTA2, NOTA3, NOTA4) ;
```

```
MEDIA := (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4 ;
```

```
SE MEDIA >= 6 ENTÃO
```

```
    ESCREVER ( 'APROVADO' )
```

```
SENÃO
```

```
    ESCREVER ( 'REPROVADO' ) ;
```

```
    ESCREVER ( NOME, MEDIA )
```

```
FIM-SE
```

```
FIM.
```

Exercícios:

- 1) Crie os algoritmos abaixo:
 - a) leia um número inteiro e escreva se ele é par ou ímpar.
 - b) Leia a idade de uma pessoa e escreva se ela uma criança, adolescente ou um adulto. Utilizando 3 estruturas condicionais simples. Observação: criança até 12 anos, adolescente até 18 e acima disso iremos considerar todos como adultos.
 - c) Fazer a entrada de um número via teclado e em seguida verificar se é negativo. Se for trocar lhe o sinal. Calcular e exibir a raiz quadrada do número positivo.
 - d) Ler o tempo de serviço em anos e o salário de um funcionário de uma empresa, se ele tiver mais de 5 anos de tempo de serviço e um salário inferior a R\$500,00 calcular o novo salário com reajuste de 20%, caso contrário dar um reajuste de 5%. Exiba o novo salário do funcionário.

NINHOS DE SE

Usados para tomadas de decisões com mais de 2 opções.

Forma Geral:

```
SE <<CONDIÇÃO>> ENTÃO  
|  
|   <<COMANDO1>>;  
|  
|   <<COMANDON>>  
|  
SENÃO  
|  
|   SE <<CONDIÇÃO>> ENTÃO  
|  
|       <<COMANDO1>>  
|  
|   SENÃO  
|  
|       <<COMANDO1>>  
|  
FIM-SE  
FIM-SE
```

Exemplos:

Segue um Algoritmo que lê 3 números e escreve o maior.

```
ALGORITMO ACHA_MAIOR;
```

```
VAR A, B, C : INTEIRO;
```

```
INICIO
```

```
    LER (A, B, C);
```

```
    SE (A>B) E (A>C) ENTÃO
```


Exercícios:

- 1) Crie os algoritmos abaixo:
 - a) Leia a idade de uma pessoa e escreva se ela uma criança, adolescente ou um adulto. Utilizando SE. Observação: criança até 12 anos, adolescente até 18 e acima disso iremos considerar todos como adultos.
 - b) Leia 3 números inteiros (A, B e C) e escreva seus respectivos valores ordenados de modo crescente. Ex. ENTRADA: A=3, B=1, C=2; PROCESSAMENTO: Ordenar os valores; SAÍDA: A=1, B=2 e C=3.
 - c) Crie um algoritmo que leia o salário de 1 funcionário e de acordo com a tabela abaixo, calcule e escreva seu reajuste e novo salário:

Salário	Percentual de Reajuste
Até R\$500.00	20%
Acima de R\$500.00 até R\$1000.00	15%
Acima de R\$1000.00	10%

Ex. Salário=R\$100.00; reajuste=R\$20,00; novo salário=R\$120.00.

ESTRUTURA DE CONDIÇÃO CONFORME

A estrutura de condição CONFORME equivale a um ninho de SE, usada quando dispomos de mais de 2 opções para uma decisão. Seu modo é mais fácil de ser compreendido:

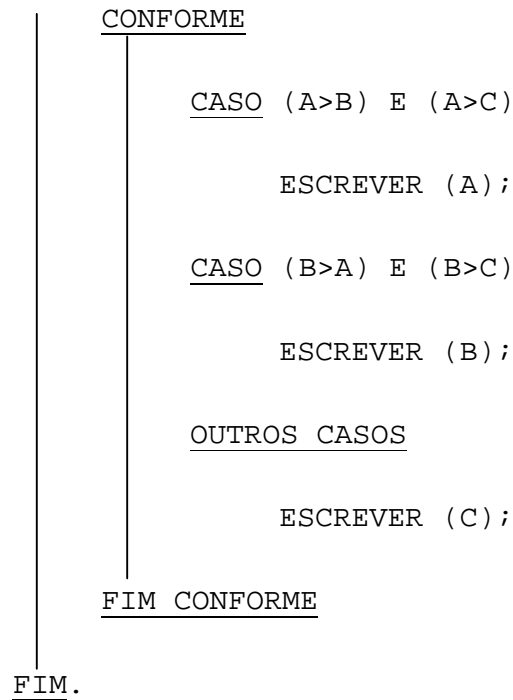
Forma Geral:

```
CONFORME  
|  
|  CASO <<CONDIÇÃO1>>  
|      <<COMANDO1>>;  
|  
|  CASO <<CONDIÇÃO2>>  
|      <<COMANDO1>>;  
|  
|  OUTROS CASOS  
|      <<COMANDO1>>;  
|  
FIM CONFORME
```

Iremos usar o mesmo exemplo do capítulo anterior para demonstrar a estrutura CONFORME. Observe como o algoritmo fica menor e de melhor compreensão.

Segue um Algoritmo que lê 3 números e escreve o maior.

```
ALGORITMO ACHA_MAIOR;  
  
VAR A, B, C : INTEIRO;  
  
INICIO  
|  
|  LER (A, B, C);
```



Podemos ter inúmeras opções (CASOS) em uma estrutura conforme. A Condição OUTROS CASOS seria equivalente a um SE-NÃO, ou seja, se nenhuma das opções for verdadeira ela é executada. Logo podemos concluir que a opção OUTROS CASOS pode ou não existir nessa estrutura (não há obrigatoriedade).

Caso existam várias condições que possam ser verdadeiras dentro da estrutura, somente a **primeira** será executada, desprezando as demais.

Exercícios:

- 1) Crie um algoritmo que leia um número inteiro, verifique se ele pode representar um mês existente em nosso calendário (1..12), caso positivo escreva o mês correspondente.
- 2) Reescreva os exercícios do capítulo anterior utilizando a estrutura CASO.
- 3) Uma instituição filantrópica estabeleceu um nível de donativos para seus membros, baseando-se na categoria de renda. Há 4 categorias: (1) não muito alta (2) suficiente (3) confortável (4) alta.
- 4) Um membro se dirige ao computador da instituição que lhe pede um número correspondente ao nível apropriado de renda. O computador usa o número para determinar o nível de donativo. A doação recomendada para cada nível é a seguinte:

Categoria 1 = R\$ 10.00

Categoria 2 = R\$ 20.00

Categoria 3 = R\$ 50.00

Categoria 4 = R\$ 100.00

O computador mostra na tela o total esperado.

“A sua contribuição é de: R\$” xx.xx

Escreva o algoritmo que permitirá a criação do programa acima.

5) Observe e responda:

Variáveis com valores armazenados:

A = VERDADEIRO

B = FALSO

C = FALSO

D = VERDADEIRO

Trecho do algoritmo:

SE (A e B) ENTÃO

ESCREVER (‘1’)

SENÃO

SE D ENTÃO

CONFORME

CASO A

ESCREVER (‘2’);

CASO B

ESCREVER (‘3’);

CASO C

ESCREVER (‘4’);

```

                                     CASO D
                                     ESCREVER ( '5' );
                                     OUTROS CASOS
                                     ESCREVER ( '2' );
                                     FIM CONFORME
                                     FIM SE
FIM SE
ESCREVER ( '7' )
```

Quais os valores serão exibidos de acordo com o trecho do algoritmo acima?

- 6) Crie um algoritmo que leia os coeficientes de uma equação de segundo grau (A, B e C), calcule o valor de Delta e caso possível escreva suas raízes. Lembrando que:

$$\Delta = B^2 - 4AC$$

Se $\Delta = 0$ raiz única, se Δ for maior que 0 então raízes distintas (X1 e X2), se Δ for menor que 0 raízes imaginárias.

$$X = \frac{-B \pm \sqrt{\Delta}}{2A}$$

ESTRUTURA DE REPETIÇÃO "PARA"

Quando uma seqüência de comandos deve ser executada repetidas vezes, tem-se uma estrutura de repetição.

A estrutura de repetição, assim como a de decisão, envolve sempre a avaliação de uma condição.

Na repetição PARA, o algoritmo apresenta previamente a quantidade de repetições desejadas.

Forma Geral:

```
PARA <<VARIÁVEL INTEIRA>> := <<VALOR INICIAL>> ATÉ <<VALOR FINAL>> FAÇA  
|  
| <<COMANDO1>>;  
|  
FIM PARA
```

A repetição por padrão determina o passo do valor inicial até o valor final como sendo 1. Determinadas linguagens possuem passo -1 ou permitem que o programador defina o passo.

ALGORITMO SETE

Segue abaixo um algoritmo que vai exibir 10 vezes a frase "Editora Brasport" de maneira arcaica:

```
ALGORITMO REPETE1;
```

INICIO

```
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );  
  
    ESCREVER ( 'EDITORA BRASPORT' );
```

FIM.

Que algoritmo grande para algo tão simples, não? Ainda bem que no Word existe Copiar e Colar caso contrário não colocaria esse exemplo... Imagine ainda se fosse para exibir 100 vezes! Ufa... Vamos ver do modo mais simples agora:

ALGORITMO REPETE2;

VAR I:INTEIRO;

VARIÁVEL IMPLEMENTADA DE I EM I

INICIO

PARA I=1 ATÉ 10 FAÇA

```
    ESCREVER ( 'EDITORA BRASPORT' )
```


ALGORITMO FATORIAL;

VAR

FAT, N, I :INTEIRO;

INICIO

FAT := 1;

LER (N);

PARA I DE 1 ATÉ N FAÇA

FATORIAL = FATORIAL * I

FIM PARA

FIM

Exercícios:

1) Crie um algoritmo que escreva os 100 primeiros números ímpares:

2) Crie um algoritmo que escreva os 15 primeiros números múltiplos de 5.

3) Crie um algoritmo que leia um número inteiro N e escreva os N primeiros números múltiplos de 3.

4) Crie um algoritmo que escreva os 17 primeiros termos da seqüência abaixo:

1, 1, 2, 3, 5, 8, 13, 21 ...

5) Crie um algoritmo que escreva os 20 primeiros termos da seqüência abaixo:

1, 3, 7, 15, 31, 63,127...

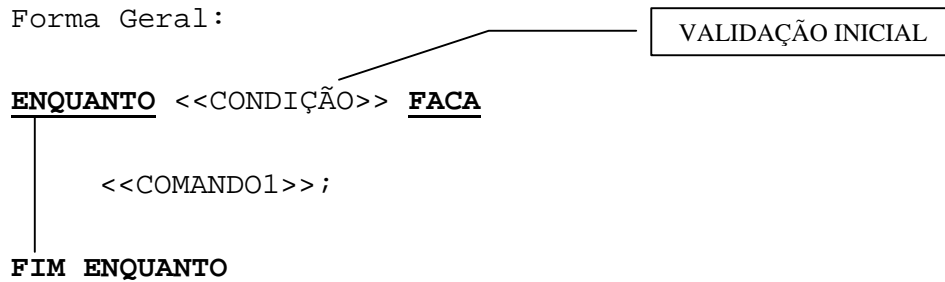
6) Ler as notas de 30 alunos de uma turma. Calcular e exibir a média da turma.

ESTRUTURA DE REPETIÇÃO INDETERMINADA COM VALIDAÇÃO

INICIAL

É usada para repetir N vezes uma ou mais instruções. Tendo como vantagem o fato de não ser necessário o conhecimento prévio do número de repetições.

Forma Geral:



ALGORITMO NOVE

Segue um algoritmo que calcule a soma dos salários dos funcionários de uma empresa. O programa termina quando o usuário digitar um salário menor que 0.

```
PROGRAMA SOMA_SALARIOS;  
  
VAR SOMA, SALARIO : REAL;  
  
INICIO  
    SOMA:=0;  
  
    SALARIO:=0;  
  
    ENQUANTO SALARIO>=0 FAÇA  
        LER (SALARIO);  
  
        SOMA:=SOMA+SALARIO
```

```
    |   FIM ENQUANTO  
    |  
    |   ESCREVER (SOMA)  
    |  
FIM.
```

Se o primeiro valor testado for falso, a repetição terminará sem que suas instruções sejam executadas, pois o teste precede os comandos. Logo o número de repetições varia de 0 a N vezes.

A estrutura de repetição ENQUANTO pode substituir a estrutura PARA, mas a recíproca nem sempre é verdadeira, lembrando que cada situação deve ditar a estrutura ideal.

Abaixo segue um exemplo de cálculo de fatorial utilizando a estrutura ENQUANTO.

```
ALGORITMO FATORIAL;  
  
VAR  
  
    FAT, N, I :INTEIRO;  
  
INICIO  
    |  
    |   FAT := 1;  
    |  
    |   LER (N);  
    |  
    |   I=0;  
    |  
    |   ENQUANTO I<=N FAÇA  
    |       |  
    |       |   FATORIAL = FATORIAL * I  
    |       |  
    |       |   I:=I+1   {CONTADOR}
```

FIM ENQUANTO

FIM

**TODAS AS VARIÁVEIS QUE ACUMULAM VALORES DEVEM
RECEBER UM VALOR INICIAL.**

REPITA ENQUANTO FOR VERDADEIRO!

Exercícios:

1) Crie um algoritmo que leia a idade dos alunos de uma classe e calcule sua média. A leitura termina (Flag) quando a idade for igual a 999.

OBS: Flag é um termo utilizado para sinalizar algo, no nosso caso para sinalizar o fim da repetição.

2) Crie um algoritmo que escreva os 15 primeiros números múltiplos de 5 utilizando a estrutura ENQUANTO.

3) Ler o preço unitário de uma mercadoria e a quantidade comprada. Calcular e imprimir o total da compra.

4) Ler o preço de uma caixa de biscoitos, sabendo que a caixa contém 20 pacotes, calcular e exibir o preço de cada pacote.

5) Ler uma série de números do teclado. Exibir o maior e o menor número lido. O Flag é um número negativo qualquer (que não deve ser considerado)

6) Ler o nome o sexo (codificado por M ou F) e a idade dos funcionários de uma empresa. Exibir a quantidade de homens, de mulheres

e a idade média dos homens e das mulheres dessa empresa. A entrada de dados termina quando o nome do funcionário for "FIM".

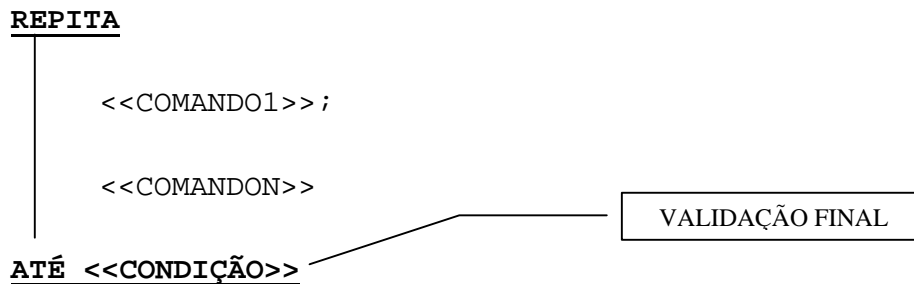
ESTRUTURA DE REPETIÇÃO INDETERMINADA COM VALIDAÇÃO

FINAL

Assim como a estrutura ENQUANTO É usada para repetir N vezes uma ou mais instruções.

Sua principal característica é ter a validação final, fazendo com que a repetição seja executada ao menos uma vez.

Forma Geral;



ALGORITMO DEZ

Segue um algoritmo que calcule a soma dos salários dos funcionários de uma empresa. O programa termina quando o usuário digitar um salário menor que 0.

ALGORITMO SOMA_SALARIOS;

VAR

SOMA, SALARIO : REAL;

INICIO

SOMA := 0;

REPITA

|

```
                LER (SALARIO);  
                SOMA:=SOMA+SALARIO  
                ATE SALARIO<0;  
                ESCREVER (SOMA)  
FIM.
```

ALGORITMO ONZE

Segue um algoritmo que escreve os 100 primeiros números pares.

```
ALGORITMO PARES_2;  
VAR I, PAR, CONTADOR : INTEIRO;  
INICIO  
    CONTADOR := 0;  
    PAR := 0;  
    REPITA  
        ESCREVER (PAR);  
        PAR := PAR+2;  
        CONTADOR := CONTADOR+1;  
    ATE CONTADOR=100  
FIM.
```

REPITA ATÉ SER FALSO!

Exercícios:

- 1) Determine os valores de A, B e C que serão exibidos ao final da execução do programa:

ALGORITMO LOUCO;

VAR

A, B, C, CONT:INTEIRO;

INICIO

A:=8;

B:=10;

C:=7;

PARA CONT DE 1 ATÉ 9 FAÇA

A:=A+B

B:=B+1

FIM-PARA

SE B>13 ENTÃO

C:=C^2

SENÃO

C:=C*2

FIM-SE

EXIBIR (A,B,C)

FIM.

Programas Equivalentes

O algoritmo onze poderia ter sido criado com qualquer estrutura de repetição. Portanto como já foi citado anteriormente, estou ressaltando que podemos ter algoritmos que são escritos de maneiras diferentes, mas, funcionam realizando o mesmo objetivo. A esses damos o nome de Equivalentes.

Loop

Devemos ter atenção ao trabalharmos com estruturas de repetição indeterminada, para não cairmos no erro de Loop, ou seja, criarmos um programa com uma rotina que se torne eterna, pois o processamento ira causar erro e travamento da máquina.

Trecho de algoritmo com exemplo de Loop Eterno:

```
ENQUANTO 10>2 FAÇA  
    ESCREVER ( 'BRASIL' )  
FIM ENQUANTO
```

Como 10 será sempre maior do que 2, a condição será sempre verdadeira e o programa irá repetir a palavra VASCO eternamente.

ESTRUTURA DE DADOS

A Estrutura de Dados é uma técnica de programação que possibilita armazenar dados na memória ou em disco objetivando atingir o menor consumo de espaço possível no menor tempo.

TIPOS DE DADOS

Tipos Primitivos de Dados → Como estudado anteriormente, são pré-definidos pela linguagem e permitem a definição de novos tipos.

Ex: Inteiro, Real, Caractere, Lógico etc.

Tipos de Dados Estáticos → São gerados a partir de tipos já existentes e não sofrem alterações em suas "características" (escopo) durante a execução do programa.

Ex: Vetor, Matriz e Registro.

Tipos de Dados Dinâmicos → São aqueles que sofrem alterações durante a execução do programa.

Ex: Ponteiros (Pilha, Fila, Lista, Árvore etc).

Os tipos de dados estáticos e dinâmicos serão estudados a seguir.

VETOR

O vetor é uma estrutura de dados homogênea (de mesmo tipo) e unidimensional. Também conhecida como Variável Indexada, Tabela ou Array.

Imagine que queremos criar um algoritmo que leia 15 idades, calcule e escreva a soma, a média e depois exiba uma idade qualquer solicitada pelo usuário.

Inicialmente poderíamos pensar em usar apenas uma variável, e acumular seus valores para calcular a soma e a média entretanto com isso a cada idade lida, a anterior seria apagada da memória não permitindo uma posterior consulta do usuário.

Podemos então criar um programa com 15 variáveis de idade, solução fácil, mas, nada prática. Suponha que houvesse uma alteração no programa, e não fossem mais 15 idades e sim 150... que trabalhadeira seria só para declarar as variáveis heim???

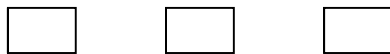
A necessidade de utilizar um Vetor surge nessa situação, ele nada mais é que um conjunto de variáveis de mesma característica.

Abaixo segue a representação de três variáveis:

Idade1

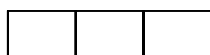
Idade2

Idade3



Abaixo segue a representação de 1 vetor para substituir as variáveis:

Idade



Idade[1]

Idade[2]

Idade[3]

O vetor possui um índice que identifica a posição do dado armazenado, esse índice é do tipo Inteiro e fica representado entre colchetes.

Podemos imaginar o vetor desta forma:

Cidadão N°	Idade
1	18
2	23
3	30
4	15
5	20
6	23
7	18
8	19
9	20
10	65
11	23
12	56
13	22
14	24
15	9

Onde a coluna cidadão é o Índice e a coluna Idade é o Vetor.

Forma geral para declaração do vetor:

```
<<NOME>> : VETOR[1..N] DE <<TIPO>>;
```

Segue então a resolução do algoritmo exemplo:

```
ALGORITMO VETORIDADE;
```

```
VAR
```

```
    IDADE:VETOR[1..15] DE INTEIRO;
```

```
    SOMA, CIDADAO, I: INTEIRO;
```

```
        MEDIA:REAL;  
INICIO  
        SOMA:=0;  
        PARA I:=1 ATÉ 15 FAÇA  
            LER(IDADE[I])  
            SOMA:=SOMA+IDADE[I]  
        FIM-PARA  
        ESCREVER (SOMA);  
        MEDIA := SOMA/15;  
        ESCREVER (MEDIA);  
        ESCREVER (´DIGITE O NUMERO DO CIDADAO O QUAL DESEJA  
OBTER A IDADE´);  
        LER (CIDADA0);  
        ESCREVER (IDADE[CIDADA0])  
FIM.
```

Observação:

A primeira estrutura de repetição carrega o vetor com dados do usuário, de modo muito mais prático que seria a leitura de 15 variáveis, calculando a soma...

Matriz

A matriz é uma estrutura de dados homogênea multidimensional.

```
<<NOME>>: MATRIZ [1 : N, 1:M] DE <<TIPO>>
```

```
ALGORITMO LER_NOMES;
```

```
VARIÁVEIS
```

```
    Nomes      : VETOR [1:4,1:4] DE CARACTERE;
```

```
    I,J       : INTEIRO;
```

```
BEGIN
```

```
    PARA I ← 1 ATE 4, PASSO 1, FACA
```

```
        PARA J ← 1 ATE 4, PASSO 1, FACA
```

```
            LEIA (NOMES [ I , J ] )
```

```
        FIM PARA
```

```
    FIM PARA
```

```
    PARA I ← 1 ATE 4, PASSO 1, FACA
```

```
        PARA J ← 1 ATE 4, PASSO 1, FACA
```

```
            ESCREVA ( '===NOMES===' );
```

```
            ESCREVA ( '[' , I , ':' , J , ']' , NOMES[ I , J ] )
```

```
        FIM PARA
```

```
    FIM PARA
```

```
FIM
```

Exercícios:

1)Seja VET um vetor com o seguinte conteúdo:

9	2	5	3	1
1	2	3	4	5

Considerando os valores das posições de VET como dados de entrada, verifique o que será impresso pelo trecho do algoritmo abaixo:

```
VAR
    VET : VETOR [ 1 .. 5 ] DE INTEIRO
    I : inteiro
INICIO
    PARA I DE 1 ATE 5 FAÇA
        INICIO
            LEIA ( VET [ I ] )
        FIM
    PARA I DE 2 ATE 4 FAÇA
        INICIO
            ESCREVA ( VET [ I ] )
        FIM
FIM
```

2) Faça um programa que leia quatro números inteiros, coloque-os em um vetor e mostre-os na ordem inversa da leitura.

3) Faça um programa que leia uma lista de 20 números, colocando-os em um vetor e, após o término da leitura, mostre os elementos com índice maior ou igual a 10.

4) Crie um algoritmo que deve aceitar o nome e o salário de 10 pessoas, reajustar cada salário em 10% e exibir o nome de cada pessoa e o seu novo salário.

5) Fazer um algoritmo que leia dez números inteiros para um vetor de 10 posições e calcule então a soma de todos os elementos do vetor. Ao final, imprima a soma dos elementos.

6) Fazer um algoritmo que leia dez elementos de um vetor A, e construa um vetor B, através da seguinte lei de formação: cada elemento de B deve ser o elemento correspondente em A multiplicado por 2. Ao final imprima todos os elementos de B.

- 7) Faça um algoritmo para ler e imprimir uma matriz 2x4 de números inteiros.
- 8) Dado uma matriz de ordem 3x3 faça um algoritmo que:
- a) Calcule a soma dos elementos da primeira coluna;
 - b) Calcule o produto dos elementos da primeira linha;
 - c) Calcule a soma de todos os elementos da matriz;
 - d) Calcule a soma do diagonal principal;
 - e) Soma da diagonal secundária;
- 8) Dado uma matriz de ordem NxN faça um algoritmo que verifique se a matriz é simétrica ($a_{ij}=a_{ji}$).
- 10) Dado uma matriz NxM de valores reais faça um algoritmo que faça a leitura destes valores e ao final da leitura de todos, imprimir o seguintes relatório:
- a) Qual a Soma dos valores de cada coluna da matriz;
 - b) Listar os valores que são menores que a média dos valores;
 - c) Qual a soma dos elementos da diagonal secundária;

MODULARIZAÇÃO

É uma técnica de programação que consiste basicamente na divisão de um algoritmo complexo em diversos módulos simples com funções bem definidas, gerenciados por um módulo principal.

A modularização aumenta a produtividade do algoritmo, uma vez que os módulos de uso geral podem ser agrupados em bibliotecas para serem reaproveitados no futuro, sem a necessidade de um novo desenvolvimento e de uma nova bateria de testes para a eliminação de erros, pois os mesmos já foram eliminados quando da criação inicial dos módulos.

Os módulos classificam-se em dois tipos:

Procedimentos e
Funções.

PROCEDIMENTO

É um módulo que é ativado através da colocação de seu nome em alguma parte do programa. Desta forma, assim que o nome de um procedimento é encontrado, ocorre um desvio no programa, para que os comandos do módulo sejam executados. Ao término do módulo, a execução retornará ao ponto subsequente a chamada do procedimento.

Obs.: O Procedimento deve ser declarado ANTES do programa principal, uma vez que, ao ser ativado pelo programa principal, o computador já deve saber de sua existência.

Alem de economizar memória de programa, mas também para estruturar a programação.

```
ALGORITMO PROCEDIMENTO;  
VAR  
  A, B, M: REAL;  
  
PROCEDIMENTO MEDIA  
  INICIO  
    M ← (A + B)/2  
  FIM  
  
INICIO  
  ESCREVA ('Informe dois números:');  
  LEIA (A, B);  
  MEDIA  
FIM
```

Neste ponto caber uma explicação sobre variáveis.

Todas as variáveis declaradas no início do programa podem também ser usadas pelos procedimentos. Diz-se que o escopo daquelas variáveis compreende todos os blocos do programa que foram definidos depois.

Poderíamos definir variáveis cujo escopo se limitasse ao bloco de um (ou mais) procedimentos, bastando que uma declaração VAR seja codificada após a declaração procedimento.

Damos o nome de **variáveis globais** para aquelas variáveis que são definidas logo após o comando VAR do programa principal, sendo desta forma visíveis em qualquer parte do programa.

Damos o nome de **variáveis locais** às variáveis que são declaradas dentro de uma sub-rotina (módulo), sendo que as mesmas só podem ser manipuladas dentro da sub-rotina que as declarou, não sendo visíveis em nenhuma outra parte do programa.

Obs.: É possível definir variáveis globais e locais com o mesmo nome, sendo que qualquer mudança no conteúdo da variável local não afetará o conteúdo da variável global.

```
ALGORITMO ORDENACAO;
VAR
  A, B: INTEIRO
PROCEDIMENTO ORDENAR
VAR
  AUX : INTEIRO;
INICIO
  SE ( A > B ) ENTAO
    INICIO
      AUX ← A;
      A ← B;
      B ← AUX;
    FIM
  FIM
INICIO
  ESCREVA ( 'Informe dois números: ' );
  LEIA ( A, B );
  ORDENAR;
  ESCREVA ( 'Os números ordenados são: ', A, B );
FIM
FUNÇÃO
```

Também é um bloco de programa contendo INICIO e FIM e sendo identificada por um nome através do qual também será referenciada em qualquer parte do programa principal.

Uma função, embora seja bastante semelhante a um procedimento, tem a característica especial de retornar ao programa que a chamou um único valor associado ao nome da função.

Exercícios:

1) Deverá ser criado um programa que faça uso de uma sub-rotina de função que retorne o valor da soma de dois números fornecidos como parâmetros.

2) Criar um programa "Calculadora" que apresente um menu de seleções no programa principal. Este menu deverá dar ao usuário a possibilidade de escolher uma entre quatro operações aritméticas. Escolhida a opção desejada, deverá ser solicitada a entrada de dois números e, processada a operação, deverá ser exibido o resultado.

Obs: Este programa deverá ser um conjunto de 7 módulos, um principal e 6 secundários. O módulo principal efetuará o controle sobre os 6 módulos secundários, sendo eles: ENTRADA, SAIDA, SOMA, SUBTRACAO, MULTIPLICACAO e DIVISAO.

Utilize PROCEDIMENTOS.

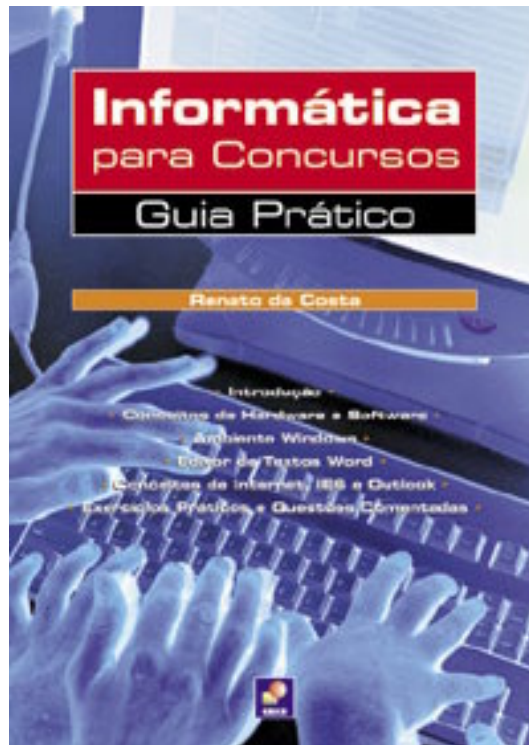
3) Refaça o algoritmo anterior utilizando FUNÇÕES onde for possível.

PROFESSOR RENATO DA COSTA

Educação:

- Pós Graduação em Docência do Ensino Superior – UCAM
- Licenciatura Plena em Informática – UCAM
- Curso Superior de Tecnologia em Processamento de Dados – UniverCidade
- Licenciando em Matemática – UniSUAM
- Curso Técnico em Processamento de Dados – GAP Tamandaré

Autor do livro:



This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.