

UERGS - UNIVERSIDADE ESTADUAL DO RIO GRANDE DO SUL

CURSO SUPERIOR DE TECNOLOGIA EM AUTOMAÇÃO INDUSTRIAL

INFORMÁTICA II

ALGORITMOS

PROF. ANDRÉ LAWISCH

NOVO HAMBURGO, AGOSTO DE 2002

SUMÁRIO

SUMÁRIO	2
ALGORITMO	4
O QUE É?	4
Formas de representação	4
Descrição narrativa.....	4
Fluxograma convencional.....	5
Exemplo de um fluxograma	5
Pseudocódigo.....	6
Exemplo de um pseudocódigo	6
Exercício.....	7
LÓGICA	8
VARIÁVEIS.....	9
Variáveis de memória	9
Tipos de Variáveis.....	9
Variável Caracter.....	9
Variável Numérica	10
Variável Lógica	10
Variável Indexada.....	10
Vetor.....	10
Matriz.....	11
Classes de Variáveis.....	12
Variáveis Públicas	12
Variáveis Privadas.....	12
Exercícios	13
Declaração de Variáveis	14
Exemplo	14
Atribuição de Valores às Variáveis.....	15
Exemplo	15
Representação da atribuição.....	15
EXPRESSÕES E OPERADORES.....	17
Expressões	17
Expressão Aritmética	17
Operadores Aritméticos.....	18
Linearização de Expressões.....	18
Expressões Lógicas	19

Operadores Relacionais	19
Operadores Lógicos	19
Exemplo	20
Exercícios	20
Instruções de entrada e saída	21
Entrada de dados em variáveis.....	21
Representação da entrada de dados em variáveis:	21
Exemplos.....	21
Saída de informações em variáveis	21
Representação da saída de informações em variáveis	22
Exemplo de algoritmo com entrada e saída	22
Exercício.....	22
Estruturas de programação	23
Instrução seqüencial	24
Exemplo	24
Exercícios	24
Decisão simples	24
Exemplo	25
Sem opção para condição falsa	25
Com opção para condição falsa	26
Exemplos.....	26
Outro exemplo	27
Exercícios	28
Decisão Composta.....	28
Representação de decisão composta	29
Exemplo	29
Exercícios	30
Repetição condicional com pré-teste	30
Representação de Repetição com pré-teste	30
Exemplo	31
Repetição condicional com pós-teste.....	32
Representação de Repetição com pós-teste.....	33
Exemplo	33
Exercícios	34
Repetição finita	34
Representação de Repetição finita.....	35
Funcionamento da estrutura de repetição finita	36
Exercícios	37
Referências Bibliográficas	38

ALGORITMO

O QUE É?

- “É um **conjunto de regras** que permite a **resolução de um problema** ou a execução de um trabalho, através de um **número finito de operações.**”
- “É a **especificação da seqüência ordenada de passos** que deve ser seguida para a **realização de uma tarefa**, garantindo a sua **repetibilidade.**”
- “É um **conjunto finito de regras**, bem definidas, utilizadas para a **solução de um problema** num **tempo finito.**”

Formas de representação

Descrição narrativa

Os algoritmos são expressos em linguagem natural.



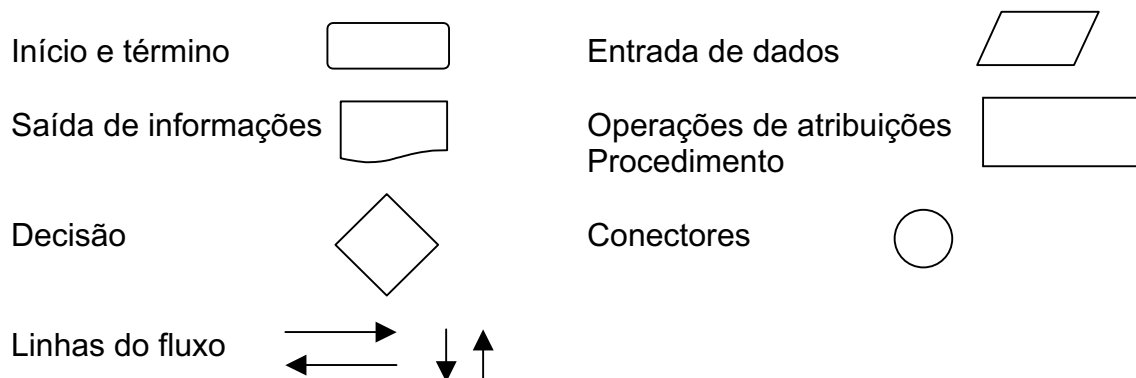
- Ex.:
1. Entrar no banheiro e tirar a roupa
 2. Abrir a torneira do chuveiro
 3. Entrar na água
 4. Ensaboar-se
 5. Tirar o sabão
 6. Sair da água

7. Fechar a torneira
8. Enxugar-se
9. Vestir-se
10. Sair do banheiro

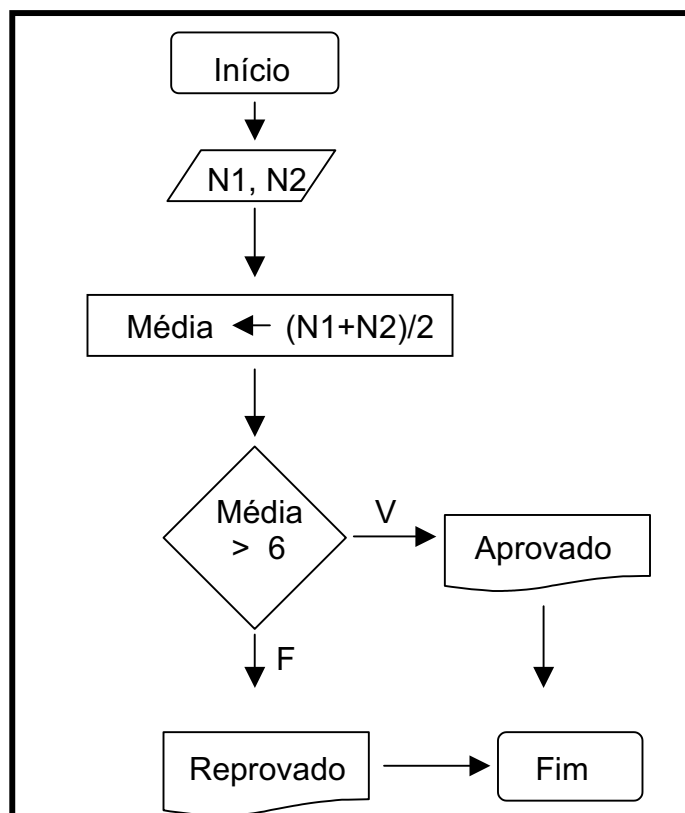
Esta representação é pouco utilizada, pois da margem a má interpretação.

Fluxograma convencional

É a representação gráfica do algoritmo. Utiliza-se formas geométricas diferentes para representar as ações.



Exemplo de um fluxograma



Nesta forma de representação, tem-se uma boa visão da lógica, seqüência dos passos, mas não se tem clareza das definições.

Pseudocódigo

Também chamada de português estruturado, é a forma de representação mais utilizada, pois é rica em detalhes, como a definição dos tipos de variáveis usadas no algoritmo e, por ser muito semelhante a forma que os programas são escritos.

```
ALGORITMO <nome_do_algoritmo>  
<declaração_de_variáveis>  
<subalgoritmos>  
INÍCIO  
    <corpo_do_programa>  
FIM
```

Onde:

ALGORITMO – é uma palavra que indica o início da definição
<nome_do_algoritmo> é o nome dado ao algoritmo como forma de distingui-lo dos demais
<declaração_de_variáveis> local opcional onde são declaradas as variáveis globais
<subalgoritmos> local opcional para declarar os subalgoritmos (rotinas específicas que serão chamadas no corpo_do_programa)
INÍCIO e FIM – delimitadores do corpo_do_programa
<corpo_do_programa> local onde será escrito o programa

Exemplo de um pseudocódigo

```
ALGORITMO Média  
Var N1, N2, Média : real  
INÍCIO  
    Leia N1, N2  
    Média ← (N1 + N2) / 2  
    Se Média > 6  
        Então  
            Escreva “Aprovado”  
        Senão  
            Escreva “Reprovado”  
    Fim_se  
FIM
```

Exercício

Fazer a descrição narrativa, o fluxograma e o pseudocódigo da seguinte situação:

Faça um Spaghetti a Bolonhesa,
Ravioli a quatro queijos,
Sopa de capeletti.



LÓGICA

É definida basicamente como sendo o estudo das **leis do raciocínio** e do modo de aplicá-las corretamente na demonstração da verdade.

Chamamos de **algoritmo lógico** aquele algoritmo cujas instruções estão dispostas **ordenadamente** e de maneira **compreensível** por qualquer pessoa que possua conhecimentos básicos sobre o assunto.

Compreensível quer dizer que o algoritmo deve ser facilmente entendido, sem que seja necessário perder muito tempo para a tradução da idéia do mesmo.

Cada pessoa analisa um problema de forma diferente e há uma tendência de complicar a sua solução. Desta forma deve-se pensar muito na solução desejada, analisando todas as possibilidades, ao invés de utilizar a primeira solução que lhe vier na cabeça. Caso contrário, pode-se perder muito tempo tentando entendê-lo do que criando-o novamente com as alterações desejadas.

VARIÁVEIS

Variáveis de memória

São endereços de memória **RAM** (Randomic Access Memory) do computador, onde são armazenados temporariamente os dados utilizados por um programa durante o seu processamento e podem ter seus conteúdos alterados durante o processamento do programa.

Para se ter acesso ao conteúdo armazenado nestes endereços, devemos dar **nomes** às variáveis. Para formar o **nome da variável** devemos seguir algumas regras básicas:

1. Todo nome de variável deve começar com uma letra;
2. Poderão ser utilizadas todas as letras do alfabeto, exceto o **ç** e as **acentuadas**;
3. Os nomes não podem misturar letras maiúsculas e minúsculas;
4. Podem ser utilizados os algarismos de **0 a 9**;
5. Dos caracteres especiais, só pode ser utilizado o sublinhado (**_**).

Ex.: Nome_de_variavel , Nome_de_Variavel
Resp001, resp001, resp01, resp_1, RESP_1

Tipos de Variáveis

As variáveis de memória dividem-se basicamente em quatro tipos:

1. Caracter
2. Numérica
3. Lógica
4. Indexada

Variável Caracter

A variável caracter também é conhecida como variável alfanumérica, pois o seu conteúdo pode ser composto de letras, números e caracteres especiais.

Quando atribuímos à estas variáveis, valores constantes, estes valores devem estar delimitados por aspas (“ ”).

Ex.: nome ← “André”
Ende ← “Rua São Joaquim”

Variável Numérica

A variável numérica pode ter como conteúdo, todos os algarismos (0, 1, 2,, 9), os sinais positivo (+) e negativo (-) e o ponto decimal.

A variável numérica é dividida basicamente em dois tipos:

1. Inteira: é formada pelo conjunto dos números inteiros.

Ex.: {..., -3, -2, -1, 0, 1, 2, 3,}

2. Real: é formada pelo conjunto dos números reais (inteiros e fracionários positivos e negativos).

Ex.: {..., -3, -2, -1, -3/4, 0, 1, 2, 5/2, 3,}

Variável Lógica

É uma variável que pode armazenar em seu conteúdo apenas valores lógicos (verdadeiro ou falso). A representação do conteúdo é feita pelas letras T, Y, F e N, colocadas entre dois pontos.

.T. = true ou verdadeiro

.Y. = yes ou verdadeiro

.F. = false ou falso

.N. = not ou falso

Variável Indexada

É uma variável de memória segmentada em **colunas** ou em **linhas e colunas**, que pode possuir conteúdos diferentes, desde que do mesmo tipo e tamanho, em cada segmento.

É um conjunto dividido em parte e, cada parte é referenciada por um índice que representa a posição relativa dentro do conjunto.

As variáveis indexadas divididas em unidimensional e bidimensional, ou vetor e matriz, respectivamente.

Vetor

É formado por uma única linha e diversas colunas, como mostra a figura abaixo:

Vetor NOTA

5	6	7	← Valores
----------	----------	----------	-----------

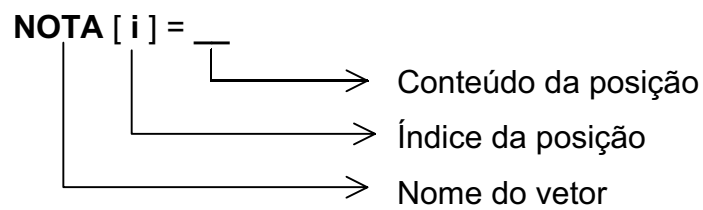
0 1 2 ← Índices

NOTA [0] = 5

NOTA [1] = 6

NOTA [2] = 7

Onde:



Matriz

É formada por um conjunto de linhas e colunas, conforme figura abaixo:

Matriz NOTA

0	5	6	7	← Valores
1	6	7	8	← Valores
	0	1	2	← Coluna

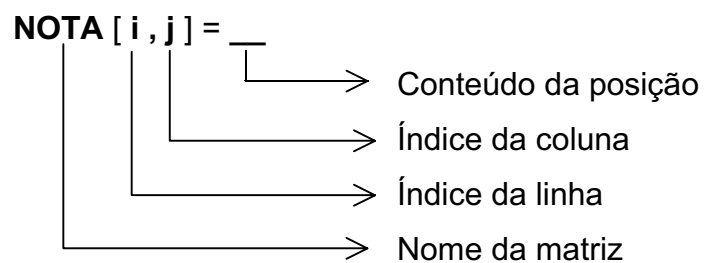
Linha →

NOTA [0,0] = 5

NOTA [1,0] = 6

NOTA [0,2] = 7

Onde:



Classes de Variáveis

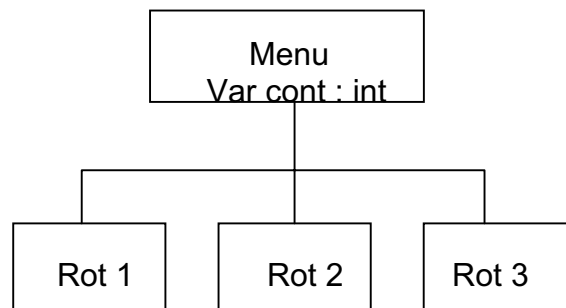
As variáveis numéricas, caracter, lógica e indexada podem ser classificadas em **públicas** ou **privadas**, dependendo da sua localização e de quem pode acessá-las.

Variáveis Públicas

“Pertencente ou destinado à coletividades” também conhecidas como variáveis globais. Elas são declaradas no topo do algoritmo (declaração_de_variáveis) e podem ser acessadas ou estão disponíveis a todos os módulos chamados pelo algoritmo.

Revisando:

```
ALGORITMO <nome_do_algoritmo>  
<declaração_de_variáveis> declaradas aqui  
<subalgoritmos>  
INÍCIO  
    <corpo_do_programa>  
FIM
```



Como a variável **cont** foi declarada no início do algoritmo (módulo chamador), ela será visível por todas as rotinas chamadas ou estas rotinas podem acessar o valor de **cont** e alterar seu conteúdo.

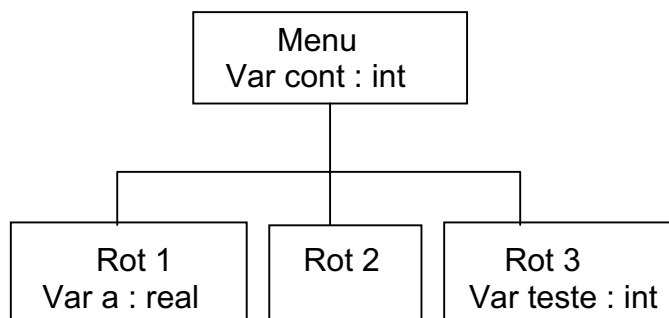
Variáveis Privadas

“que não é público, particular” são chamadas de variáveis locais e são declaradas dentro do corpo_do_algoritmo.

Revisando:

```

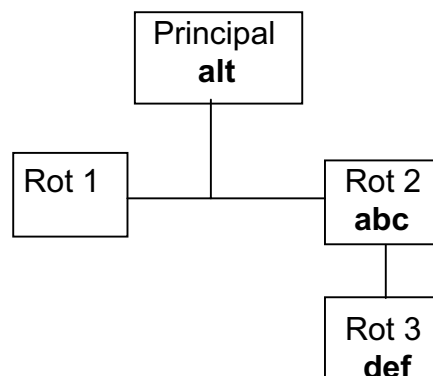
ALGORITMO <nome_do_algoritmo>
<declaração_de_variáveis>
<subalgoritmos>
INÍCIO
    <corpo_do_programa> declaradas aqui
FIM
  
```



As variáveis privadas ou locais só são vistas pela rotina que a criou. No caso acima, a variável **cont** continua acessível a todas as rotinas, mas as variáveis **a** e **teste** estão acessíveis somente às rotinas que as criaram.

Exercícios

- Classifique as alternativas a seguir, de acordo com a convenção:
 [**C**]aracter, [**L**]ógico, [**N**]umérico, [**I**]ndexado
 258,23 "Maria" .F. a[10]
 "a[09]" top[10,5] 1 .Y.
- Defina variável global e local.
- De acordo com o esquema abaixo, marque as alternativas corretas:



- alt** só pode ser alterada pelo módulo Principal

- () A variável **def** é local para Rot 3
- () Rot 1 não pode acessar nenhuma variável
- () A variável **abc** só pode ser modificada por Rot 2

Declaração de Variáveis

Como vimos anteriormente, a declaração das variáveis pode ser feita em dois níveis. No primeiro, a variável **pública/global** será declarada fora do <corpo_do_programa> e no segundo, a variável **privada/local** será declarada dentro do <corpo_do_programa>.

Revisando:

```

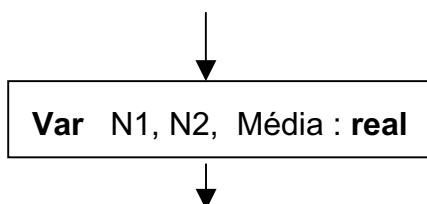
ALGORITMO <nome_do_algoritmo>
<declaração_de_variáveis> /* aqui são declaradas as variáveis globais*/
<subalgoritmos>
INÍCIO
    <corpo_do_programa> /*aqui são declaradas as variáveis locais*/
FIM
  
```

Exemplo

No pseudocódigo

Pública/Global	Privada/Local
ALGORITMO Média Var N1, N2, Média : real INÍCIO Leia N1, N2 Média (N1 + N2) /2 Se Média > 6 Então Escreva "Aprovado" Senão Escreva "Reprovado" Fim_se FIM	ALGORITMO Média INÍCIO Var N1, N2, Média : real Leia N1, N2 Média (N1 + N2) /2 Se Média > 6 Então Escreva "Aprovado" Senão Escreva "Reprovado" Fim_se FIM

No fluxograma:



Como no fluxograma o nível de detalhamento é pequeno, não existe a diferença entre uma variável pública e privada. As duas são declaradas da mesma forma, conforme o exemplo acima.

Atribuição de Valores às Variáveis

A atribuição de um valor a uma variável consiste na mudança do valor atual por outro valor do mesmo tipo ou, é a ação de substituição do conteúdo de uma variável onde, ela “perde” seu valor e passa a assumir um novo valor.

Se forem observados os exemplos, sempre foi utilizada a seta (\leftarrow) para representar uma atribuição e nunca o sinal de igualdade (=). O motivo é simples, a igualdade é um operador relacional que estabelece uma relação entre operandos. Veremos adiante que podemos utilizar o operador relacional para estabelecer uma condição a ser testada. Desta forma, não podemos confundir a atribuição e o operador relacional igual, mesmo que a representação utilizada na prática seja a mesma.

Exemplo

Atribuição	Operador relacional igual
Média $\leftarrow (N1+N2)/2$	Se Média = 6 então “Aprovado”

Representação da atribuição

<u>Pseudocódigo</u>	<u>Fluxograma</u>	
< variável> \leftarrow < expressão>;	<table border="1"> <tr> <td>< variável> \leftarrow < expressão></td> </tr> </table>	< variável> \leftarrow < expressão>
< variável> \leftarrow < expressão>		

Na prática, a maioria das linguagens de programação não faz distinção com relação a simbologia utilizada. Desta forma, devemos analisar o contexto em que esta simbologia está inserida para podermos definir; é uma atribuição ou uma igualdade.

Analise as situações apresentadas a seguir, e marque se são atribuições e/ou igualdades:

A = B + 2 () atribuição () igualdade

A = B () atribuição () igualdade

B = 6 () atribuição () igualdade

Se você marcou atribuição ou igualdade, pode ter acertado ou errado. É difícil fazer esta análise sem verificar o contexto onde estas expressões estão inseridas. Vamos tentar novamente?

A = B + 2 () atribuição () igualdade

SE A = B ENTÃO () atribuição () igualdade

B = 6 () atribuição () igualdade

Considerando as expressões apresentadas acima, fica mais fácil de se fazer a análise, pois o contexto foi definido. A instrução SE <exp> ENTÃO necessita da declaração de uma **exp** lógica e, desta forma, o símbolo = representa uma igualdade e não uma atribuição. Nos outros dois casos, não é apresentada explicitamente uma instrução que manipula exp lógica. Partindo deste princípio, consideramos que as expressões são aritméticas e o símbolo = é uma atribuição.

Agora que você está craque em atribuições, responda:

A = 6

A = A+1

A = ? Qual o valor de A?

EXPRESSÕES E OPERADORES

Expressões

No sentido matemático, são *representações simbólicas* de seqüência de operações a serem feitas sobre determinados valores, visando a obtenção de um resultado.

Toda a expressão é composta de uma seqüência de operações (doravante denominadas de **operadores**) que atuam sobre determinados valores (doravante denominados de **operandos**), assegurando um resultado final.

$$\textcircled{B} + \textcircled{C} - \textcircled{D}$$

$$\textcircled{A} > \textcircled{B}$$

Expressão Aritmética

Expressão Lógica

O conjunto de operandos e operadores que compõe uma expressão determina o tipo de expressão e define o tipo de resultado a ser obtido.

1. *Expressão aritmética*: produz como resultado um número.

2. *Expressão lógica*: produz como resultado um valor lógico.

Para cada uma das expressões acima, pode ser definido um conjunto de operações possíveis. Estas operações podem ser:

- Monádicas: um operador atua sobre um operando.
- Diádicas: um operador atua sobre dois operandos.

Independente da forma de representação, qualquer expressão pode ser decomposta em seqüências encadeadas de operações monádicas ou diádicas.

Ex.: $X - (-Y)$

Expressão Aritmética

É uma constante ou uma variável ou uma combinação de constantes e/ou variáveis por meio de operadores aritméticos. Estes operadores indicam que operações e em que ordem elas devem ser executadas, visando o resultado final.

Operadores Aritméticos

Na resolução de uma expressão aritmética, devemos seguir a ordem pré-definida de execução dos operadores que aparecem nesta expressão.

A ordem de prioridade dos operadores aritméticos é a seguinte:

1. Potenciação (******, **^**) e monádicas (-)
2. Multiplicação (*****) e divisão (**/**)
3. Adição (**+**) e subtração (**-**)
4. Parênteses podem alterar esta ordem
5. Segue-se da esquerda para a direita nos casos de indeterminação (mais de um operador com a mesma prioridade).

Exemplo:

Qual a ordem de resolução no caso da expressão $\frac{\sqrt{B^2 - 4AC}}{2A}$

- | | |
|--------------|----------------------------------|
| 1. B^2 | 5. $\sqrt{B^2 - 4AC}$ |
| 2. $4A$ | 6. $2A$ |
| 3. $4AC$ | |
| 4. B^2-4AC | 7. $\frac{\sqrt{B^2 - 4AC}}{2A}$ |

Obs.: Os passos 5 e 6 podem aparecer invertidos, pois esta nova ordem não irá alterar o resultado final.

A decomposição de uma expressão deverá atender sempre o princípio básico: a solução das subexpressões deve conduzir à solução de toda a expressão.

Linearização de Expressões

Consiste em “pegar” uma expressão matemática, normalmente escrita em várias linhas, e reescrevê-la em uma única linha. Só assim o computador irá entendê-la. Para isto, devemos utilizar a simbologia e a ordem de prioridade dos operadores aritméticos.

Exemplo:

$$X = \frac{A^2 - 5B + \frac{C}{2}}{(B + 3)7}$$

$$X = (A**2-5*B+C/2) / ((B+3)/7)$$

Expressões Lógicas

Todo cálculo lógico é baseado em proposições. As proposições são combinadas através de relações e inter-relações, e é possível demonstrar a veracidade ou a falsidade destas combinações.

Operadores Relacionais

As relações fazem parte das expressões lógicas, como operandos, uma vez que seu resultado é lógico. As relações são definidas entre expressões aritméticas, através dos operadores relacionais.

Os operadores relacionais são:

- Maior (>)
- Menor (<)
- Maior ou igual (>=, =>)
- Menor ou igual (<=, =<)
- Igual (=)
- Diferente (<>, #, !=)

Operadores Lógicos

As inter-relações são feitas através de operadores lógicos. Um operador lógico é aquele cujos operandos são expressões lógicas e o resultado é um valor lógico (V ou F).

Devido ao restrito domínio dos valores lógicos, é possível definir os resultados de forma compacta através de “Tabela Verdade”. Esta tabela mostra o resultado das operações para todas as possíveis combinações dos operandos.

Os operadores lógicos utilizados são:

- NOT (~)
- AND (^)
- OR (v)

Tabela verdade dos operadores lógicos NOT, AND e OR.

A	B	~A	A^B	AvB
V	V	F	V	V
F	V	V	F	V
V	F	F	F	V
F	F	V	F	F

Através das combinações de operadores aritméticos, relacionais e lógicos, conseguimos elaborar expressões lógicas simples ou complexas, que resultam nas condições utilizadas nos algoritmos com seleção.

Exemplo

$$A = B, A > B \text{ AND } C = D$$

$$\begin{array}{ccc} A > B & \text{AND} & B = C & A = 5, B = 3, C = 3 \\ \text{V} & & \text{V} & \\ & & \text{V} & \end{array}$$

Exercícios

1. Resolva a expressão, considerando $a=5$, $b=1$ e $c=2$, demonstrando a ordem em que as operações são resolvidas e o valor final de X.

$$X = \frac{(a+b)^2 \cdot (a-b) + 4a - 5b + c}{2}$$

2. Quais as operações necessárias para intercambiar os valores de duas variáveis A e B, de modo que A fique com o valor de B e B com o de A.

3. Se $X=15$ e forem executadas as seguintes expressões:

$$X = X + 3$$

$$X = X - 6$$

$$X = X/2$$

$$X = 3X$$

Qual o valor de X?

4. Linearizar as seguintes expressões:

$$Z = \sqrt{(2\Pi IFL)^2 + \frac{1}{(2\Pi IFL)^2}} + R^2$$

$$A = \frac{\frac{1}{C}}{\frac{X+Y}{Z-T}} - \left(4N + \frac{X-Y^2}{\frac{1}{C}} \right)$$

5. Resolva as expressões:

$$A + 5 > C - 2 \quad \text{OR} \quad B + A - C < A + C - B$$

$$A=7, \quad B=7, \quad C=10$$

$$(A \geq B \text{ AND } D = C) \text{ OR } (D \leq E \text{ AND } B \geq C)$$

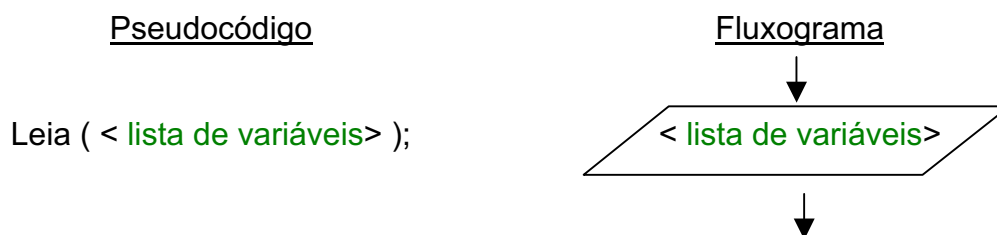
$$D=B, \quad E>B, \quad A=E, \quad C>E$$

INSTRUÇÕES DE ENTRADA E SAÍDA

Entrada de dados em variáveis

Consiste na atribuição de um valor através da digitação, e não através do processamento do programa utilizado.

Representação da entrada de dados em variáveis:



Onde:

< lista de variáveis > → nome de uma ou mais variáveis que receberão os valores digitados. No caso de haver mais de uma variável, elas devem estar separadas por vírgula.

💣 Obs.: a variável ou variáveis utilizadas na entrada de dados devem ter sido declaradas anteriormente.

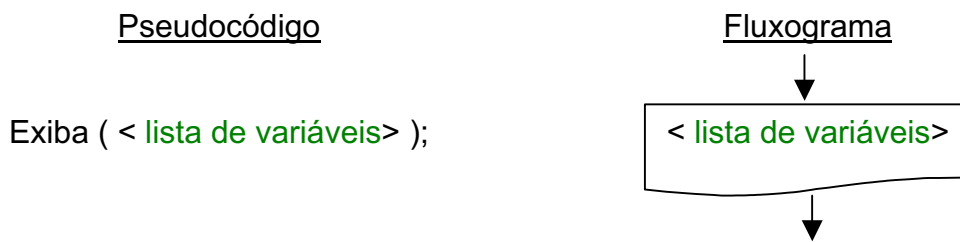
Exemplos

```
Leia (a,b,c);  
Leia (nome_do_aluno);
```

Saída de informações em variáveis

É a forma que utilizamos para exibir o valor de uma variável, constante ou uma mensagem. É apresentar o resultado do processamento em qualquer dispositivo de saída de informações.

Representação da saída de informações em variáveis



Quando utilizarmos a instrução de saída de informações, podemos anexar **mensagens** para facilitar o entendimento da ação desejada.

Exemplo:

```
Exiba ("O nome do aluno é", nome_do_aluno);  
Exiba (a,b,c);  
Exiba ("Digite o nome do aluno");
```

Exemplo de algoritmo com entrada e saída

ALGORITMO Exemplo

INÍCIO

```
nome: caracter ; } Declaração das variáveis  
ender: caracter; }  
Exiba ("Digite seu nome: ");  
Leia (nome);  
Exiba ("Digite seu endereço: ");  
Leia (ender);  
Exiba ("É bom te ver ", nome);  
Exiba ("Hoje vamos fazer uma festa na ", ender);
```

FIM

Exercício

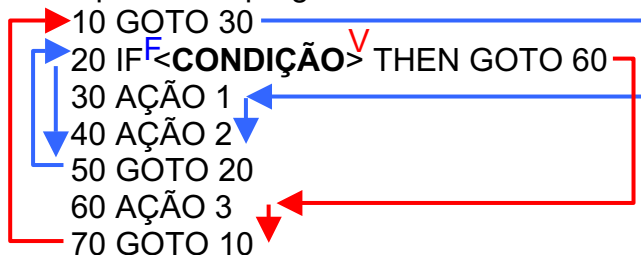
1. Fazer o fluxograma do exemplo acima.

ESTRUTURAS DE PROGRAMAÇÃO

Programação estruturada pode ser definida como sendo uma técnica de programação, na qual a construção é feita com base em estruturas de simples controle.

A rigidez imposta por essas estruturas é um fator importante para a compreensão do programa, pois, uma vez que não existe comandos de desvio, o programador é obrigado a utilizar uma ***lógica coerente*** que se adapte as estruturas.

Exemplo de um programa não estruturado em BASIC:



Tente entender a lógica deste programa. É complicada? Isto acontece porque o uso de comando de desvio gera uma desordenação na lógica, deixa o programa poluído e de difícil compreensão.

As estruturas de controle têm a finalidade de despoluir através da ordenação lógica, estabelecendo um início, meio e fim e as estruturas são dispostas numa ordem tal que, ao chegar no final do algoritmo (última linha), o processamento tenha sido executado por completo.

As estruturas de programação são:

1. Instrução seqüencial;
2. Decisão simples;
3. Decisão dupla;
4. Repetição condicional com pré-teste;
5. Repetição condicional com pós-teste;
6. Repetição finita.

Instrução seqüencial

É uma instrução ou conjunto de instruções declaradas de forma seqüencial, não possuindo nenhum tipo de repetição ou desvio (cada linha do algoritmo é executada uma única vez).

Exemplo

```
Algoritmo MÉDIA
Início
  N1, N2 :int;
  Exiba ("Digite duas notas");
  Leia (N1, N2);
  Media ← (N1 + N2)/2;
  Exiba ("Sua média é: ",Media);
Fim
```

Exercícios

1. Escreva um algoritmo que escreva os números pares existentes no intervalo de 20 a 30.
2. Construa um algoritmo que leia a idade de uma pessoa, expressa em anos, meses e dias e escreva-a expressa em dias.
3. Construa um algoritmo que leia a idade de uma pessoa, expressa em dias e escreva-a expressa em anos, meses e dias.
4. Faça um algoritmo para calcular o volume de uma esfera de raio R.
5. Escreva um algoritmo que lê um valor em R\$ e o decompõe no menor número de notas possíveis de 1, 2, 5, 10, 20, 50 e 100. Escrever o número lido e a quantidade de notas necessárias.

Decisão simples

É aquela estrutura onde se tem um ou mais de seus passos (instruções) subordinados a uma condição. Esta condição é uma expressão lógica que será criada ou desenvolvida de acordo com o caso que se quer verificar. Até o momento foi visto como podemos criar estruturas simples e seqüenciais. Há casos de estruturas seqüências que necessitam de verificação, ou seja, ao ler uma variável, o pseudocódigo deve verificar a validade deste valor.

Exemplo

Algoritmo EXEMPLO

Var n1, n2, media: Real;

INÍCIO

Leia(n1, n2);
 media = (n1+n2)/2;
 Exiba(media);

FIM

LEGENDA

n1 e n2 → notas bimestrais

media → media das notas n1 e n2

Análise a seguinte situação: as variáveis n1 e n2 foram declaradas como REAL e isto garante que elas possam assumir **qualquer valor** pertencente ao conjunto dos números reais. Mas isto não garante que os valores assumidos (digitados) são valores válidos.

Considerando que as variáveis n1 e n2 foram projetadas para assumir valores de notas e que, os valores aceitos para nota na escola pertencem ao intervalo de 0 a 10 inclusive, deve-se ter o cuidado de contemplar esta particularidade das variáveis. Como fazer? Observe as estruturas apresentadas abaixo e compare o exemplo acima com o de baixo.

A estrutura de decisão simples tem duas formas de ser representada:

Sem opção para condição falsa

Pseudocódigo

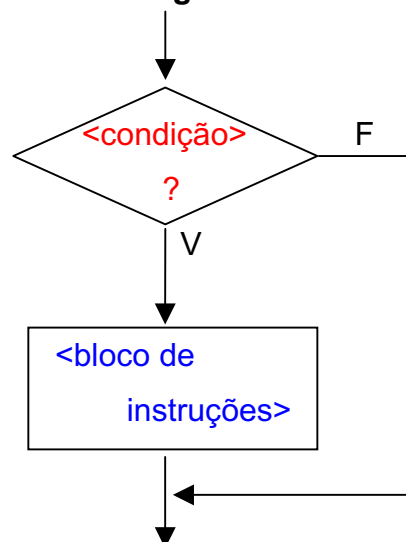
Se <condição> então

<instrução 1>;
 <instrução 2>;
 <instrução 3>;
 <instrução n>;

Fim_se

Neste caso, se a condição for verdadeira, serão executadas as instruções que foram declaradas entre o **então** e o **Fim_se**, caso contrário, nenhuma instrução é executada.

Fluxograma

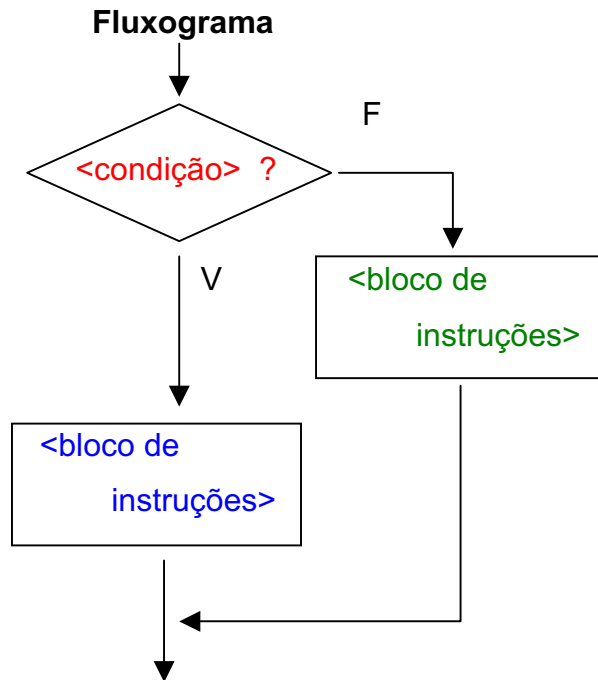


Com opção para condição falsa

Pseudocódigo
Se <condição> **então**
 <instrução 1>;
 <instrução 2>;
 <instrução 3>;
 <instrução m>;

senão
 <instrução m+1>;
 <instrução m+2>;
 <instrução m+3>;
 <instrução n>;

Fim_se
 Neste caso, se a condição for verdadeira, serão executadas as **instruções** que foram declaradas entre o **então** e o **senão**, caso contrário, serão executadas as **instruções** que foram declaradas entre o **senão** e o **Fim_se**.



Aplicando as estruturas de decisão, podemos corrigir o exemplo acima e ainda, apresentá-lo de várias formas. Veja:

Exemplos

Algoritmo EXEMPLO

Var n1, n2, media: Real;

INÍCIO

Leia(n1, n2);

Se n1>=0 and n1<=10 and n2>=0 and n2<=10 **então**

 media = (n1+n2)/2;

Exiba(media);

Fim_se

FIM

Algoritmo EXEMPLO

Var n1, n2, media: Real;

INÍCIO

Leia(n1, n2);

Se n1>=0 and n1<=10 and n2>=0 and n2<=10 **então**

 media = (n1+n2)/2;

Exiba(media);

Senão

```
    Exiba("Nota(s) inválida(s)");  
  Fim_se  
FIM
```

Algoritmo EXEMPLO**Var n1, n2, media: Real;****INÍCIO**

```
  Leia(n1, n2);  
  Se n1<0 or n1>10 or n2<0 or n2>10 então  
    Exiba("Nota(s) inválida(s)");  
  Senão  
    media = (n1+n2)/2;  
    Exiba(media);  
  Fim_se  
FIM
```

Algoritmo EXEMPLO**Var n1, n2, media: Real;****INÍCIO**

```
  Leia(n1, n2);  
  Se n1>=0 and n1<=10 então  
    Se n2>=0 and n2<=10 então  
      media = (n1+n2)/2;  
      Exiba(media);  
    Senão  
      Exiba("Nota(s) inválida(s)");  
    Fim_se  
  Senão  
    Exiba("Nota(s) inválida(s)");  
  Fim_se  
FIM
```

OSB: Não deixe de observar as estruturas utilizadas e as condições estabelecidas.

Outro exemplo

Algoritmo Báskara**Início**

```
  a, b, c:int;  
  x1, x2:real;  
  Leia (a);  
  Se a=0 então  
    Exiba ("Esta equação não é de 2° grau");  
  senão
```

```
Leia (b, c);
D ← b*b - 4*a*c;
Se d<0 então
    Exiba ("Raízes imaginárias");
senão
    x1 = ( -b + (d)^(1/2) ) / ( 2*a );
    x2 = ( -b - (d)^(1/2) ) / ( 2*a );
    Exiba ("As raízes da equação são: ", x1, x2);
Fim_se
Fim_se
Fim
```

Exercícios

1. Crie um algoritmo que lê um número qualquer e exibe uma mensagem informando se ele é positivo ou negativo.
2. Elaborar um algoritmo que faça a leitura de um número inteiro e positivo e apresente uma mensagem informando se o número é par ou ímpar.
3. Desenvolva um algoritmo que leia dois valores numéricos e apresente a diferença do maior pelo menor.
4. Crie um algoritmo que lê três valores e exibe o maior deles.
5. Crie um algoritmo que lê três valores e exibe os três em ordem crescente.
6. Elabore um algoritmo que lê três números (média do trimestre) e calcula a média anual da seguinte forma:
Os pesos dos trimestres são 2, 3.5 e 4.5 respectivamente;
A maior média será multiplicada pelo maior peso e assim sucessivamente;
No final, some os produtos e divida pela soma dos pesos.

Decisão Composta

Esta estrutura permite que seja escolhida **uma** entre **várias** condições possíveis. Associada a escolha, podemos ter uma instrução ou um bloco de instruções que serão executadas.

Representação de decisão composta

Pseudocódigo

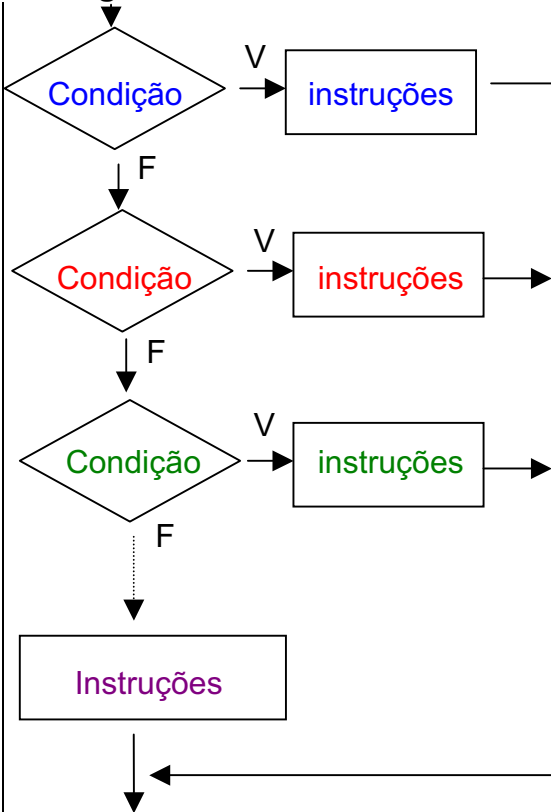
Escolha

Caso <condição 1>
 <bloco de instruções 1>
Caso <condição 2>
 <bloco de instruções 2>
Caso <condição 3>
 <bloco de instruções 3>
Caso <condição 4>
 <bloco de instruções 4>
Caso <condição n>
 <bloco de instruções n>
Caso contrário
 <bloco de instruções n+1>

Fim_escolha

Nesta estrutura, é executado o <bloco de instruções ?> correspondente a <condição ?> verdadeira, e o controle passa para a primeira instrução após o Fim_escolha. Caso nenhuma destas <condições ?> for verdadeira, será executado o caso contrário e o controle passa para a primeira instrução após o Fim_escolha.

Fluxograma



Exemplo

Algoritmo decisão_composta

Início

```

turma: int;
Exiba ("Digite a sua turma: ");
Leia (turma);
turma ← turma/1000;

```

Escolha

```

Caso turma = 1
    Exiba ("Seu curso é a Química-Diurno");
Caso turma = 2
    Exiba ("Seu curso é a Eletrotécnica-Diurno");
Caso turma = 3
    Exiba ("Seu curso é a Mecânica-Diurno");
Caso turma = 4
    Exiba ("Seu curso é a Eletrônica-Diurno");

```

```

Caso turma = 5
  Exiba ("Seu curso é a Química-Noturno");
Caso turma = 6
  Exiba ("Seu curso é a Eletrotécnica-Noturno");
Caso turma = 7
  Exiba ("Seu curso é a Mecânica-Noturno");
Caso turma = 8
  Exiba ("Seu curso é a Eletrônica-Noturno");
Caso turma = 9
  Exiba ("Seu curso é a Segurança do Trabalho-Noturno");
Caso contrário
  Exiba ("Esta turma não existe");
Fim_escolha
Fim

```

Exercícios

1. Fazer o fluxograma do exemplo acima.
2. Elaborar um algoritmo que solicite ao aluno a sua turma e informa qual o curso, a série e o turno que ele estuda.

Repetição condicional com pré-teste

A estrutura de repetição condicional com pré-teste permite que várias instruções sejam executadas **repetidamente** enquanto uma condição qualquer for verdadeira, sendo que primeiro testa a condição e depois executa as instruções.

Representação de Repetição com pré-teste

Pseudocódigo

Enquanto <condição>

```

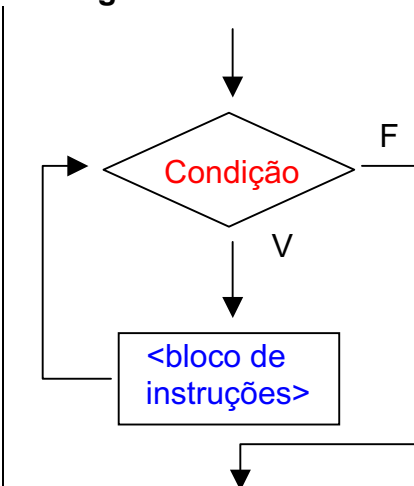
<instrução 1>;
<instrução 2>;
<instrução 3>;
<instrução n>;

```

Fim_enquanto

Nesta estrutura, caso a condição for verdadeira, são executadas as instruções que estão entre a **condição** e o **Fim enquanto**, caso contrário, será executada a primeira instrução após o **Fim enquanto**

Fluxograma



Este ciclo de instruções executadas repetidas vezes chama-se de loop. Assim, sempre que for utilizada esta estrutura, deve-se ter certeza de que existe uma opção para condição falsa, caso contrário, este loop se torna infinito.

Exemplo

Algoritmo repetição_1

Início

turma: int;

teste=1:int;

Enquanto teste=1

Exiba (“Digite a sua turma: “);

Leia (turma);

turma ← turma/1000;

Escolha

Caso turma = 1

Exiba (“Seu curso é a Química-Diurno”);

Caso turma = 2

Exiba (“Seu curso é a Eletrotécnica-Diurno”);

Caso turma = 3

Exiba (“Seu curso é a Mecânica-Diurno”);

Caso turma = 4

Exiba (“Seu curso é a Eletrônica-Diurno”);

Caso turma = 5

Exiba (“Seu curso é a Química-Noturno”);

Caso turma = 6

Exiba (“Seu curso é a Eletrotécnica-Noturno”);

Caso turma = 7

Exiba (“Seu curso é a Mecânica-Noturno”);

Caso turma = 8

Exiba (“Seu curso é a Eletrônica-Noturno”);

Caso turma = 9

Exiba (“Seu curso é a Segurança do Trabalho-Noturno”);

Caso contrário

Exiba (“Esta turma não existe”);

teste=0;

Fim_escolha

Fim_enquanto

Fim

Algoritmo repetição_2

Início

turma: int;

Exiba (“Digite a sua turma: “);

Leia (turma);

turma ← turma/1000;

Enquanto turma > 0 and turma < 10

Escolha

Caso turma = 1

```
    Exiba ("Seu curso é a Química-Diurno");
Caso turma    = 2
    Exiba ("Seu curso é a Eletrotécnica-Diurno");
Caso turma    = 3
    Exiba ("Seu curso é a Mecânica-Diurno");
Caso turma    = 4
    Exiba ("Seu curso é a Eletrônica-Diurno");
Caso turma    = 5
    Exiba ("Seu curso é a Química-Noturno");
Caso turma    = 6
    Exiba ("Seu curso é a Eletrotécnica-Noturno");
Caso turma    = 7
    Exiba ("Seu curso é a Mecânica-Noturno");
Caso turma    = 8
    Exiba ("Seu curso é a Eletrônica-Noturno");
Caso turma    = 9
    Exiba ("Seu curso é a Segurança do Trabalho-Noturno");
Caso contrário
    Exiba ("Esta turma não existe");
Fim_escolha
Exiba ("Digite a sua turma: ");
Leia (turma);
turma ← turma/1000;
Fim_enquanto
Fim
```

Repetição condicional com pós-teste

A estrutura de repetição condicional com pós-teste permite que várias instruções sejam executadas **repetidamente** enquanto uma condição qualquer **não** for verdadeira, sendo que primeiro são executadas as instruções e depois é testada a condição

Representação de Repetição com pós-teste

Pseudocódigo

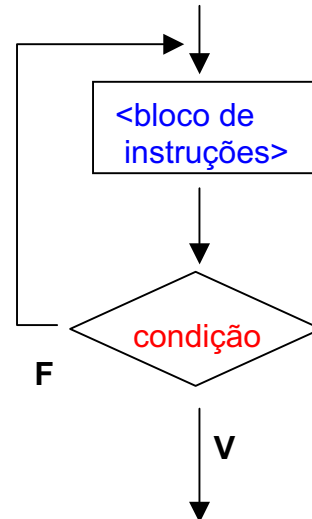
Repita

```
<instrução 1>;
<instrução 2>;
<instrução 3>;
<instrução n>;
```

Até_que <condição>

Nesta estrutura, são executadas as instruções que estão entre o **Repita** e o **Até_que**. No **Até_que** é testada uma condição qualquer e, enquanto esta condição for falsa, as instruções de 1 a n são executadas. Se a condição for verdadeira, o comando repita é encerrado e a primeira instrução após este comando é executada.

Fluxograma



Este ciclo de instruções executadas repetidas vezes chama-se de loop. Assim, sempre que for utilizada esta estrutura, deve-se ter certeza de que existe uma opção para condição verdadeira, caso contrário, este loop se torna infinito.

Exemplo

Algoritmo repetição_pós_teste

Início

```
turma: int;
```

Repita

```
Exiba ("Digite a sua turma: ");
```

```
Leia (turma);
```

```
turma ← turma/1000;
```

Escolha

```
Caso turma = 1
```

```
Exiba ("Seu curso é a Química-Diurno");
```

```
Caso turma = 2
```

```
Exiba ("Seu curso é a Eletrotécnica-Diurno");
```

```
Caso turma = 3
```

```
Exiba ("Seu curso é a Mecânica-Diurno");
```

```
Caso turma = 4
```

```
Exiba ("Seu curso é a Eletrônica-Diurno");
```

```
Caso turma = 5
```

```
Exiba ("Seu curso é a Química-Noturno");
```

```
Caso turma = 6
```

```
Exiba ("Seu curso é a Eletrotécnica-Noturno");
```

```
Caso turma = 7
    Exiba ("Seu curso é a Mecânica-Noturno");
Caso turma = 8
    Exiba ("Seu curso é a Eletrônica-Noturno");
Caso turma = 9
    Exiba ("Seu curso é a Segurança do Trabalho-Noturno");
Caso contrário
    Exiba ("Esta turma não existe");
Fim_escolha
Até_que turma < 1 or turma > 9
Fim
```

Exercícios

1. Escrever um algoritmo que leia um número não determinado de valores e calcule a média aritmética dos valores lidos, a quantidade de valores positivos e negativos e o percentual de valores positivos e negativos. Para encerrar a execução do algoritmo, o valor lido deve ser igual a zero. Exibir os resultados.
2. Fazer um algoritmo que leia uma quantidade não determinada de números positivos e calcule a quantidade de números pares e ímpares, a média dos valores pares e ímpares e a média geral dos números lidos. Para encerrar a execução do algoritmo, o valor lido deve ser igual a zero. Exibir os resultados.
3. Chico tem 1,5 metros e cresce 2 centímetros por ano, enquanto Zé tem 1,1 metros e cresce 3 centímetros por ano. Construa um algoritmo que calcule e exiba quantos anos serão necessários para que Zé seja maior que Chico.
4. Escrever um algoritmo que leia um número não determinado de pares de valores **m,n**, todos inteiros e positivos, um de cada vez, e calcule e escreva a soma de **n** inteiros consecutivos a partir de **m** inclusive, finalizando quando os valores de **m** e **n** forem iguais a zero.

Repetição finita

Anteriormente, foram vistas duas formas de se elaborar uma repetição. Uma, utiliza a estrutura **Enquanto** e a outra, **Repita**. Ambas permitem que a repetição possa ser realizada sem se ter o conhecimento de quantas vezes elas serão executadas, ou seja, o início é conhecido e o fim será determinado pela falsidade de uma condição (não se conhece de antemão o número de vezes que uma determinada seqüência de instruções deverá ser executada). Já no caso da repetição finita, tanto o início como o fim devem ser conhecidos no instante em que esta estrutura for executada. A estrutura de repetição finita tem o seu funcionamento controlado por uma variável de controle denominada de contador. Vejamos seu funcionamento:

Representação de Repetição finita

Pseudocódigo

Para **<variável>** de **<início>** até **<fim>** passo **<valor>**

<instrução 1>;
<instrução 2>;
<instrução 3>;
<instrução n>;

Fim_para

Onde:

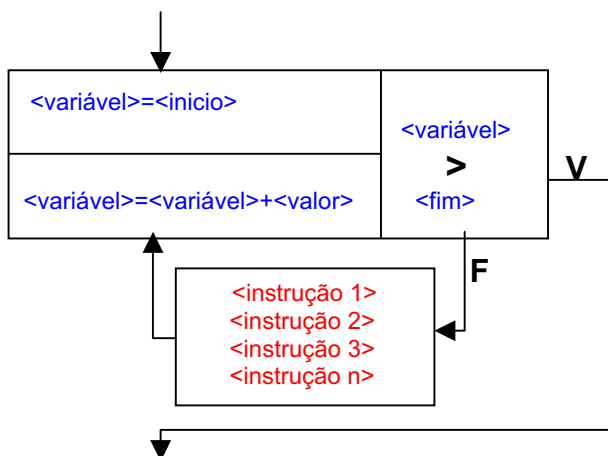
<variável> → nome da variável de controle (a que será utilizada como contador);

<início> → valor inicial de <variável>, podendo ser outra variável, uma constante ou mesmo uma expressão aritmética;

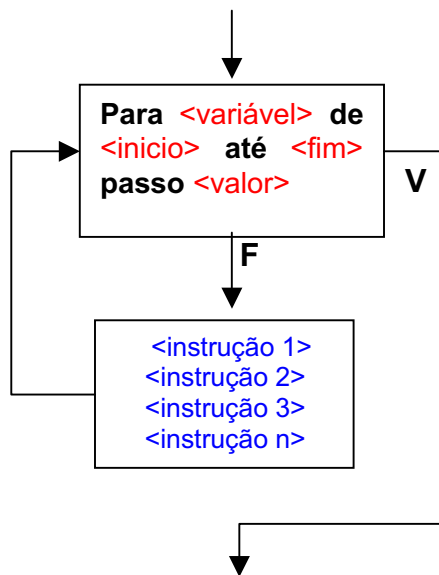
<fim> → valor final de <variável> que limita o número de repetições, podendo ser outra variável, uma constante ou mesmo uma expressão aritmética;

<valor> → valor que deverá ser somado ou subtraído de <variável>, podendo ser outra variável, uma constante ou mesmo uma expressão aritmética;

Fluxograma (versão André)



Fluxograma (versão extraída da bibliografia)



Funcionamento da estrutura de repetição finita

Para **X** de **2** até **100** passo **2**

Escreva(X);

Fim_para

1. No início da execução, **X** recebe o valor inicial (**2**) e executa o passo **Escreva(X);**
2. Ao atingir a cláusula, **Fim_para**, há o desvio para o início da estrutura e, **X** que possui valor **2** é incrementado em 2 unidades (**passo 2**);
3. Após o incremento (**X=4**) é realizada a comparação entre o novo valor de **X** e o valor final, que no exemplo é **100**. Se o valor de **X** for maior que o valor final, a repetição é encerrada e o controle passa para a primeira instrução após o **Fim_para**, caso contrário, a repetição continua sendo executada e sucessivos valores de **X** serão escrito pela instrução **Escreva(X)**, até que **X** seja maior que valor final.

No exemplo acima, a repetição finita é controlada pelo incremento da variável de controle (contador). Podemos ter casos em que a repetição seja controlada pelo decremento da variável de controle e, neste caso, deve-se declarar um **<valor>** negativo e **<início>** deve ser menor que **<fim>**. A condição que verifica o término da repetição também será invertida. Se o valor de **X** for menor que o valor final, a repetição é encerrada e o controle passa para a primeira instrução após o **Fim_para**, caso contrário, a repetição continua sendo executada e sucessivos valores de **X** serão escrito pela instrução **Escreva(X)**, até que **X** seja menor que valor final.

Para **X** de **100** até **2** passo **-2**
 Escreva(X);
Fim_para

Exercícios

1. Escreva um algoritmo que gere números de 1000 a 1999 e escreva aqueles que, divididos por 11, tem resto igual 5 ($11 \text{ mod } 5$).
2. Elabore um algoritmo que leia 50 valores inteiros e positivos e:
 - a) Encontre o maior valor;
 - b) Encontre o menor valor;
 - c) Calcule a média aritmética dos valores lidos;Exibir os resultados.
3. Elabore um algoritmo que escreva os números pares existente no intervalo de X a Y.
4. Escrever um algoritmo que gera e escreve os número primos existentes no intervalo de A a B.
5. Elabore um algoritmo que escreva os cinco primeiros números perfeitos. São números perfeitos, todos aqueles em que a soma dos seus divisores é igual ao número em questão. EX: $6 = 1 + 2 + 3$.

REFERÊNCIAS BIBLIOGRÁFICAS

FREEDMAN, Allan. Dicionário de Informática. São Paulo - Makron Books, 1996.

NASCIMENTO, Angela; HELLER, Jorge. Introdução à Informática. São Paulo – McGraw-Hill, 1990.

ORTH, Afonso Inácio. Algoritmos. Rio Grande do Sul – Gráfica Pallotti.

VENÂNCIO, Cláudio Ferreira. Desenvolvimento de Algoritmos: uma nova abordagem. São Paulo – Editora Érica, 1997.

SALIBA, Walter Luiz Caram. Técnicas de Programação: uma abordagem estruturada. São Paulo - McGraw-Hill, 1992.

BERG, Alexandre Cruz e FIGUEIRÓ, Joice Pavék. Lógica de Programação. RS – Editora da Ulbra, 1998.

MANZANO, José Augusto N. G. e OLIVEIRA, Jayr Figueiredo de. ALGORITMOS. Lógica para Desenvolvimento de Programação. São Paulo – Érica, 1996.

SALVETTI, Dirceu Douglas e BARBOSA, Lisbete Madsen. Algoritmos. São Paulo – Ed. Makron Books, 1998.

MANZANO, José Augusto N. G. e OLIVEIRA, Jayr Figueiredo de. Estudo Dirigido de Algoritmos. São Paulo – Ed. Érica, 1997.