**APOSTILA DE LÓGICA DE PROGRAMAÇÃO**

**1. RACIOCINIO LÓGICO**

**1.1 Significado de Raciocínio lógico**

Raciocínio lógico é um processo de estruturação do pensamento de acordo com as normas da [lógica](http://www.significados.com.br/logica/)que permite chegar a uma determinada conclusão ou resolver um problema.

Um raciocínio lógico requer consciência e capacidade de organização do pensamento.

É possível resolver problemas usando o raciocínio lógico. No entanto, ele não pode ser ensinado diretamente, mas pode ser desenvolvido através da resolução de exercícios lógicos que contribuem para a evolução de algumas habilidades mentais.

**1.2 Raciocínio lógico matemático ou quantitativo**

O raciocínio lógico matemático ou quantitativo é o raciocínio usado para a resolução de alguns problemas e exercícios matemáticos. Esses exercícios são frequentemente usados no âmbito escolar, através de problemas matriciais, geométricos e aritméticos, para que os alunos desenvolvam determinadas aptidões. Este tipo de raciocínio é bastante usado em áreas como a análise combinatória.

**2. INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO**

**2.1 Lógica**

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequencia lógica para o desenvolvimento.

Então o que é lógica?

**Lógica é a técnica de encadear pensamentos para atingir determinado objetivo.**

E o que é lógica de programação?

**Lógica de programação é o meio de fazer uso das técnicas de lógica matemática para estruturar as instruções que serão executadas pelo computador.**

**2.2 Instruções**

**Instruções são um conjunto de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar.**

Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica.

É evidente que essas instruções têm que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las.

Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

**2.3 Algoritmo**

**Um algoritmo é representação de uma sequencia finita de passos que levam a execução de uma tarefa.**

Podemos pensar em algoritmo como uma receita, uma sequencia de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por sequencias lógicas. Por exemplo:

“Somar dois números quaisquer”.

• Escreva o primeiro número no retângulo A   
• Escreva o segundo número no retângulo B   
• Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C

Retângulo A Retângulo B Retângulo C

**2.4 Programas**

Os programas de computadores nada mais são do que algoritmos escritos numa linguagem de computador (Pascal, C, COBOL, Fortran, Visual Basic entre outras) e que são interpretados e executados por uma máquina, no caso um computador. Notem que dada esta interpretação rigorosa, um programa é por natureza muito específico e rígido em relação aos algoritmos da vida real.

Atividade

1. Faça um algoritmo para descrever passo-a –passo o ato de escovar os dentes.
2. Faça um algoritmo para descrever todos os seus passos desde sair de casa até chegar na etec.
3. Faça um algoritmo para descrever passo-a –passo o ato de preparar um miojo.
4. Faça um algoritmo para informar o resultado do quadrado de um número.
5. Faça um algoritmo para descobrir a data de nascimento de uma pessoa a partir de sua idade

**2 DESENVOLVENDO ALGORITMOS**

**2.1 Pseudocódigo**

Os algoritmos são descritos em uma linguagem chamada pseudocódigo. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo, Visual Basic, estaremos gerando código em Visual Basic. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

**2.2 Regras para construção do Pseudocódigo**

Para escrever um algoritmo precisamos descrever a sequencia de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

• Usar somente um verbo por frase   
• Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática   
• Usar frases curtas e simples   
• Ser objetivo   
• Procurar usar palavras que não tenham sentido dúbio

**2.3 Fases**

No capítulo anterior vimos que ALGORITMO é uma sequencia lógica de instruções que podem ser executadas.

É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

CALCULAR O SALDO FINANCEIRO DE UM ESTOQUE

Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais.

Saída

Entrada

Processamento

Onde temos:

ENTRADA: São os dados de entrada do algoritmo

PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado final

SAÍDA: São os dados já processados

**2.4 Exemplo de Algoritmo**

Imagine o seguinte problema: Calcular a média final dos alunos da 3ª Série. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde: Média Final = P1 + P2 + P3 + P4 ........................................................................................................................  
 4

Para montar o algoritmo proposto, faremos três perguntas:

a) Quais são os dados de entrada?   
R: Os dados de entrada são P1, P2, P3 e P4.

b) Qual será o processamento a ser utilizado?   
R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro)

c) Quais serão os dados de saída?  
R: O dado de saída será a média final

Algoritmo

- Receba a nota da prova1   
- Receba a nota de prova2   
- Receba a nota de prova3   
- Receba a nota da prova4   
- Some todas as notas e divida o resultado por 4   
- Mostre o resultado da divisão

**2.5 Teste de Mesa**

Após desenvolver um algoritmo ele deverá sempre ser testado. Este teste é chamado de TESTE DE MESA, que significa seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

**3 DIAGRAMA DE BLOCO**

**3.1 O que é um diagrama de bloco?**

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

Com o diagrama podemos definir uma sequencia de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

**3.2 Simbologia**

Existem diversos símbolos em um diagrama de bloco. No decorrer do curso apresentaremos os mais utilizados.

Veja no quadro abaixo alguns dos símbolos que iremos utilizar:

|  |  |
| --- | --- |
| **Símbolo:** | **Função:** |
| TERMINAL | Indica o INICÍO ou FIM de um procedimento/programa.  Ex: Início do Algoritmo. |
| ENTRADA | Operação de entrada de dados.  Ex: Leitura de uma informação. |
| PROCESSAMENTO | Processamento em Geral.  Ex: Cálculo de dois Números. |
| DECISÃO | Indica uma decisão a ser tomada.  Ex: Verificação de Sexo. |
| EXIBIÇÃO | Mostra informações ou Resultados.  Ex: Exibir na tela o resultado de um cálculo |
| IMPRESSÃO | Imprime o resultado ou gera um relatório.  Ex: Imprime o resultado de uma Ordenação |
| DESVIO | Permite o desvio para um ponto do programa.  Ex: Mudança de página do algoritmo |
| ENTRADA MANUAL | Indica a entrada através do teclado.  Ex: Digitação de um valor. |

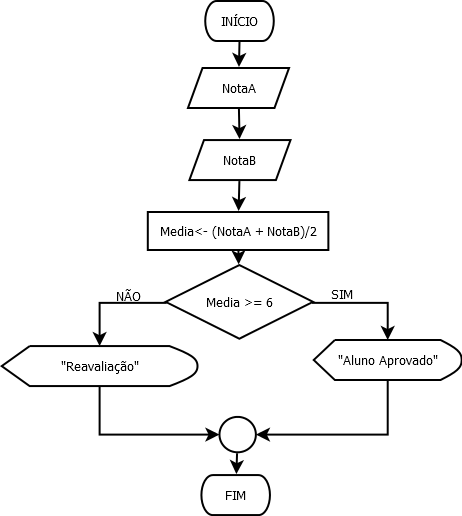
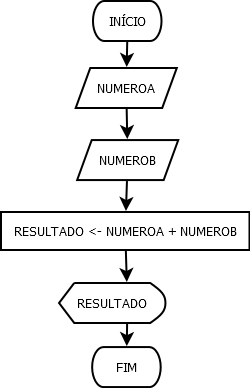
OBSERVAÇÕES:

- Dentro do símbolo sempre terá algo escrito, pois somente o símbolo não traz a informação completa;

- Os símbolos são conectados uns aos outros por meio de setas

- Todos os símbolos podem ter vários fluxos (setas) de entradas , mas so podem ter **UM** fluxo (seta) de saída;

- A exceção à regra é o símbolo de DECISÃO que sempre terá dois fluxos de saída, sendo um para resultado **VERDADEIRO** e outro para **FALSO**;

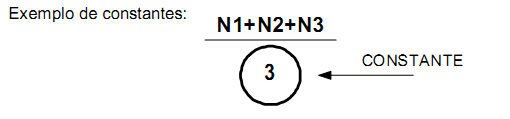
**** Exemplos de Algoritmos:

**4 CONSTANTES, VARIÁVEIS E TIPOS DE DADOS**

Variáveis e constantes são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado. Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Um programa deve conter declarações que especificam de que tipo são as variáveis que ele utilizará e às vezes um valor inicial. Tipos podem ser por exemplo: inteiros, reais, caracteres, etc. As expressões combinam variáveis e constantes para calcular novos valores.

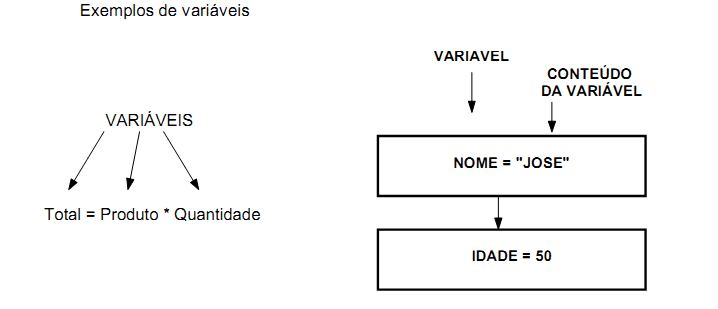
**4.1 Constantes**

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.



**4.2 Variáveis**

Variável é a representação simbólica dos elementos de certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.



**4.3 Tipos de Variáveis**

As variáveis e as constantes podem ser basicamente de quatro tipos: Numéricas, caracteres, Alfanuméricas ou lógicas.

**• inteiro:** define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais.

**• real:** define variáveis numéricas do tipo real, ou seja,com casas decimais.

**• caractere:** define variáveis do tipo string, ou seja, cadeia de caracteres.

**• logico:** define variáveis do tipo booleano, ou seja, com valor VERDADEIRO ou FALSO.

**4.4 Nomes de Variáveis e sua Declaração**

Os nomes das variáveis devem começar por uma letra e depois conter letras, números ou underline, até um limite de 30 caracteres. As variáveis podem ser simples ou estruturadas (na versão atual, os vetores podem ser de uma ou duas dimensões). Não pode haver duas variáveis com o mesmo nome, com a natural exceção dos elementos de um mesmo vetor.

A seção de declaração de variáveis começa com a palavra-chave var, e continua com as seguintes sintaxes:

<lista-de-variáveis>: <tipo-de-dado>

<lista-de-variáveis>: vetor "["<lista-de-intervalos>"]" de <tipo-de-dado>

Na <lista-de-variáveis>, os nomes das variáveis estão separados por vírgulas. Na <lista-de-intervalos>, os <intervalo>são separados por vírgulas, e têm a seguinte sintaxe:

<intervalo>: <valor-inicial>.. <valor-final>

Na versão atual do VisuAlg, tanto <valor-inicial>como <valor-final>devem ser inteiros. Além disso, exige-se evidentemente que <valor-final>seja maior do que <valor-inicial>.

Exemplos:

var a: inteiro

Valor1, Valor2: real

vet: vetor [1..10] de real

matriz: vetor [0..4,8..10] de inteiro

nome\_do\_aluno: caractere

sinalizador: logico

Note que não há a necessidade de ponto e vírgula após cada declaração: basta pular linha. A declaraçãode vetores é análoga à linguagem Pascal: a variável vetacima tem 10 elementos, com os índices de [1] a [10], enquanto matriz corresponde a 15 elementos com índices [0,8], [0,9], [0,10], [1,8], [1,9], [1,10], ... até [4,10]. O número total de variáveis suportado pelo VisuAlg é 500 (cada elemento de um vetor é contado individualmente).

**4.5 Constantes e Comandos de Atribuição.**

Um comando (ou instrução) pode ser definido como sendo uma ação a ser executada num dado momento pelo algoritmo.

O VisuAlg tem três tipos de constantes:

• Numéricos: são valores numéricos escritos na forma usual das linguagens de programação. Podem ser inteiros ou reais. Neste último caso, o separador de decimais é o ponto e não a vírgula, independente da configuração regional do computador onde o VisuAlg está sendo executado. O VisuAlg também não suporta separadores de milhares.

• Caracteres: qualquer cadeia de caracteres delimitada por aspas duplas (").

• Lógicos: admite os valores VERDADEIRO ou FALSO.

A atribuição de valores a variáveis é feita com o operador <-. Do seu lado esquerdo fica a variável à qual está sendo atribuído o valor, e à sua direita pode-se colocar qualquer expressão (constantes, variáveis, expressões numéricas), desde que seu resultado tenha tipo igual ao da variável.

Alguns exemplos de atribuições, usando as variáveis declaradas acima:

a <- 3

Valor1 <- 1.5

Valor2 <- Valor1 + a

vet[1] <- vet[1] + (a \* 3)

matriz[3,9] <- a/4 - 5

nome\_do\_aluno <- "José da Silva"

sinalizador <- FALSO

**4.6 Comando de Entrada de Dados.**

Na prática de construção de programas, será muito comum o uso de comandos que proporcionam a entrada de dados para o computador. Assim, devemos ter uma representação correspondente em nível de algoritmo para a entrada de dados. Utilizaremos o comando leia para efetuar a entrada de dados para o algoritmo, conforme sintaxe abaixo.

leia (<lista-de-variáveis>)

Recebe valores digitados pelos usuário, atribuindo-os às variáveis cujos nomes estão em <lista-de-variáveis>(é respeitada a ordem especificada nesta lista).

Exemplo:

Algoritmo: “Entrada de dados”

X: inteiro

A: real

Nome: caractere

inicio

leia(X)

leia(A)

leia(nome)

fimalgoritmo

**4.7 Comando de Saída de Dados.**

Assim como para entrada de dados, na prática de construção de programas será muito comum o uso de comandos que proporcionam a saída de dados gerados pelo computador. Assim, devemos ter uma representação correspondente em nível de algoritmo para a saída de dados. Utilizaremos o comando escreva para efetuar a saída de dados do algoritmo, conforme sintaxe abaixo:

escreva(variável, constante, expressão);

onde o algoritmo mostrará os valores de variáveis, constantes e/ou expressões.

Uma variação é a instrução escreval , conforme abaixo:

escreval (<lista-de-expressões>).

Idem ao anterior, com a única diferença que pula uma linha em seguida. É equivalente ao writelndo Pascal.

Exemplo:

Algoritmo “saída de dados”

X,Y: inteiro

A,B: real

Nome: Caractere

Início

Escreval (“Entre com o valor de X:”);

Leia (X);

escreval (“Entre com o valor de A:”);

leia (A);

escreval (“Entre com o nome da pessoa:”);

leia (nome);

Y <- X \* 3;

B <- (A \* 2,4) / 3;

Escreval (“A partir dos valores de X e A, Y e B valem: “,Y, B);

Fimalgoritmo

**5 OPERADORES**

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos três tipos de operadores:

• Operadores Aritméticos   
• Operadores Relacionais   
• Operadores Lógicos

**5.1 Operadores Aritméticos**

Os operadores aritméticos são os utilizados para obter resultados numéricos. Além da adição, subtração, multiplicação e divisão, podem utilizar também o operador para exponenciação. Os símbolos para os operadores aritméticos são:

|  |  |
| --- | --- |
| Operação | Símbolo |
| Adição | + |
| Subtração | - |
| Multiplicação | + |
| Divisão | / |
| Resto de divisão inteira | MOD ou % |
| Potência | ^ |

Hierarquia das Operações Aritméticas

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

**5.2 Operadores Relacionais**

Os operadores relacionais são utilizados para comparar String de caracteres e números. Os valores a serem comparados podem ser caracteres ou variáveis.

Estes operadores sempre retornam valores lógicos (verdadeiro ou falso/ True ou False)

Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses.

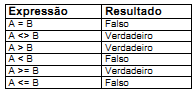
Os operadores relacionais são:

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

Exemplo:

Tendo duas variáveis A = 5 e B = 3

Os resultados das expressões seriam:



**5.3 Operadores Lógicos**

Os operadores lógicos servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso.

Os operadores lógicos são:

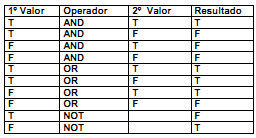
E AND  
OU OR  
NÃO NOT

**E / AND –** Uma expressão AND (E) é verdadeira se todas as condições forem verdadeiras

**OR/OU** **–** Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira

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

A tabela abaixo mostra todos os valores possíveis criados pelos três operadores lógicos (AND, OR e NOT).



**6 Estrutura básica do Portugues Estruturado no VisualG**

O formato básico do nosso pseudocódigo é o seguinte:

***algoritmo"semnome"***

***// Função :***

***// Autor :***

***// Data :***

***// Seção de Declarações***

***inicio***

***// Seção de Comandos***

***Fimalgoritmo***

Observe que as linhas iniciadas com // são apenas comentários ou seja não serão executadas pelo programa.

**7. OPERAÇÕES LÓGICAS**

Operações Lógicas são utilizadas quando se torna necessário tomar decisões em um diagrama de bloco.

Num diagrama de bloco, toda decisão terá sempre como resposta o resultado VERDADEIRO ou FALSO.

**7.1 ESTRUTURA DE DECISÃO**

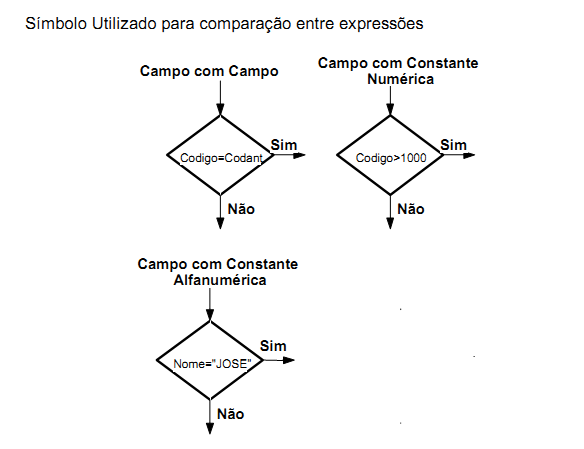
Como vimos no capítulo anterior em “Operações Lógicas”, verificamos que na maioria das vezes necessitamos tomar decisões no andamento do algoritmo. Essas decisões interferem diretamente no andamento do programa. Trabalharemos com dois tipos de estrutura. A estrutura de Decisão e a estrutura de Repetição

**7.2 Comandos de Decisão**

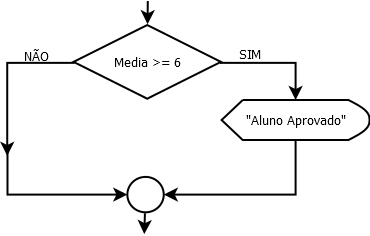
Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estruturas de programas que não são totalmente sequenciais. Com as instruções de SALTO ou DESVIO pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores. As principais estruturas de decisão são: “Se Então”, “Se então Senão” e “Caso Selecione”.

**7.2.1 SE ENTÃO / IF ... THEN**

A estrutura de decisão “SE/IF” normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando SE/IF então execute determinado comando.



Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5.0, veja no exemplo de algoritmo em diagrama de blocos ficaria assim:



Em VisuaGl Basic

Comando de Desvio Condicional

se <expressão-lógica>entao

<seqüência-de-comandos>

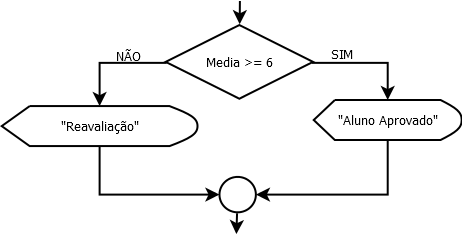
fimse

Ao encontrar este comando, o VisuAlg analisa a <expressão-lógica>. Se o seu resultado for VERDADEIRO, todos os comandos da <seqüência-de-comandos> (entre esta linha e a linha com fimse) são executados. Se o resultado for FALSO, estes comandos são desprezados e a execução do algoritmo continua a partir da primeira linha depois do fimse.

**7.2.2 SE ENTÃO SENÃO / IF ... THEN ... ELSE**

A estrutura de decisão “SE/ENTÃO/SENÃO”, funciona exatamente como a estrutura “SE”, com apenas uma diferença, em “SE” somente podemos executar comandos caso a condição seja verdadeira, diferente de “SE/SENÃO”, pois sempre um comando será executado independente da condição, ou seja, caso a condição seja “verdadeira” o comando da condição será executado, caso contrário o comando da condição “falsa” será executado.

Em diagrama



Em VisualG

se <expressão-lógica> entao

<seqüência-de-comandos-1>

senao

<seqüência-de-comandos-2>

Fimse

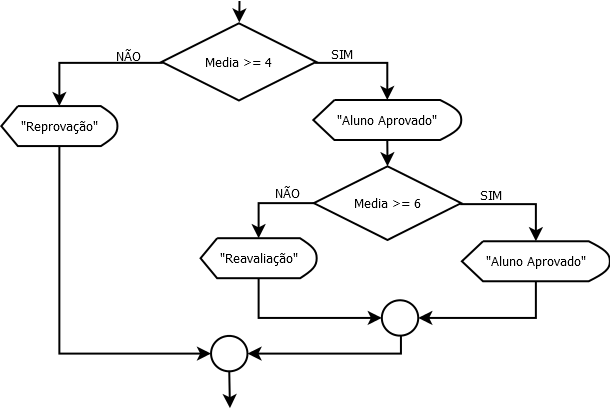
Nesta outra forma do comando, se o resultado da avaliação de <expressão-lógica>for VERDADEIRO, todos os comandos da <seqüência-de-comandos-1>(entre esta linha e a linha com senao) são executados, e a execução continua depois a partir da primeira linha depois do fimse. Se o resultado for FALSO, estes comandos são desprezados e o algoritmo continua a ser executado a partir da primeira linha depois do senao, executando todos os comandos da <seqüência-de-comandos-2>(até a linha com fimse).

Estes comandos equivalem ao if...then e if...then...else do Pascal. Note que não há necessidade de delimitadores de bloco (como begine end), pois as seqüências de comandos já estão delimitadas pelas palavras-chave senaoe fimse.

O VisuAlg permite o aninhamento desses comandos de desvio condicional.

**7.2.3 – Estrutura de decisão encadeada, ou decisão aninhada.**

No exemplo aanterior está sendo executada uma condição que, se for verdadeira, executa o comando “APROVADO”, caso contrário executa o segundo comando “REPROVADO”. Podemos também dentro de uma mesma condição testar outras condições. Como no exemplo abaixo:



Exemplo em VisualG

se MEDIA >= 5 entao

se MEDIA >= 7.0 entao

Escreval (“Aluno APROVADO”)

senao

Escreval (“Aluno Necessita fazer outra Avaliação”)

fimse

senao

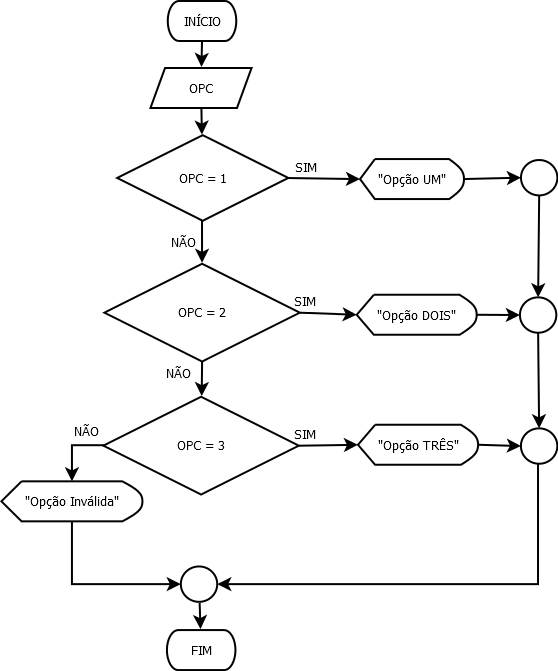
Escreval (“Aluno REPROVADO”)

fimse

**7.2.4 CASO SELECIONE / SELECT ... CASE**

A estrutura de decisão CASO/SELECIONE é utilizada para testar, na condição, uma única expressão, que produz um resultado, ou, então, o valor de uma variável, em que está armazenado um determinado conteúdo. Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula “Caso”.

No exemplo do diagrama de blocos abaixo, é recebido uma variável “Op” e testado seu conteúdo, caso uma das condições seja satisfeita, é atribuído para a variável Titulo a String “Opção X”, caso contrário é atribuído à string “Opção Errada”.



O VisuAlg implementa (com certas variações) o comando casedo Pascal. A sintaxe é a seguinte:

escolha <expressão-de-seleção>

caso <exp11>, <exp12>, ..., <exp1n>

<seqüência-de-comandos-1>

caso <exp21>, <exp22>, ..., <exp2n>

<seqüência-de-comandos-2>

...

outrocaso

<seqüência-de-comandos-extra>

fimescolha

Veja o exemplo a seguir, que ilustra bem o que faz este comando:

algoritmo "Times"

var

time: caractere

inicio

escreva ("Entre com o nome de um time de futebol: ")

leia (time)

escolha time

caso "Flamengo", "Fluminense", "Vasco", "Botafogo"

escreval ("É um time carioca.")

caso "São Paulo", "Palmeiras", "Santos", "Portuguesa"

escreval ("É um time paulista.")

outrocaso

escreval ("Não é um time importante.")

fimescolha

fimalgoritmo

**7.3 Comandos de Repetição**

Utilizamos os comandos de repetição quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado.

Trabalharemos com modelos de comandos de repetição:

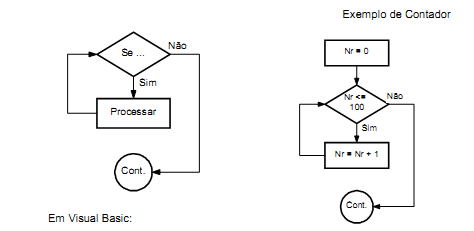
• Enquanto x, processar (Do While ...Loop);   
• Repita ..., Até que x (Do... Loop Until)   
• Para ... faça (For ... To ... Next)

O VisuAlg implementa as três estruturas de repetição usuais nas linguagens de programação: o laço contado para...ate...faca(similar ao for...to...dodo Pascal), e os laços condicionados enquanto...faca(similar ao while...do) e repita...ate(similar ao repeat...until). A sintaxe destes comandos é explicada a seguir

**7.3.1 Enquanto ... faça (Do While ... Loop)**

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação. Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

Em diagrama de bloco a estrutura é a seguinte:



Em VisualG

Enquanto ... faça

Esta estrutura repete uma seqüência de comandos enquanto uma determinada condição (especificada através de uma expressão lógica) for satisfeita.

enquanto <expressão-lógica>faca

<seqüência-de-comandos>

fimenquanto

<expressão-lógica> Esta expressão que é avaliada antes de cada repetição do laço. Quando seu resultado for VERDADEIRO, <seqüência-de-comandos>é executada.

fimenquanto Indica o fim da <seqüência-de-comandos>que será repetida. Cada vez que a execução atinge este ponto, volta-se ao início do laço para que <expressãológica> seja avaliada novamente. Se o resultado desta avaliação for VERDADEIRO, a <seqüência-de-comandos>será executada mais uma vez; caso contrário, a execução prosseguirá a partir do primeiro comando após fimenquanto.

No exemplo a seguir, os números de 1 a 10 são exibidos em ordem crescente.

algoritmo "Números de 1 a 10 (com enquanto...faca)"

var

j: inteiro

inicio

j <- 1

enquanto j <= 10 faca

escreva (j)

j <- j + 1

fimenquanto

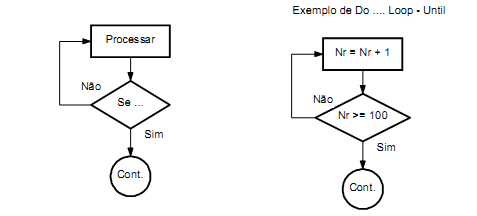
fimalgoritmo

Importante:Como o laço enquanto...facatesta sua condição de parada antesde executar sua seqüência de comandos, esta seqüência poderá ser executada zero ou mais vezes.

**7.3.2 Repita ..., Até (Do ... Loop Until)**

Neste caso, executa-se primeiro o bloco de operações e somente depois é realizado o teste de condição. Se a condição for verdadeira, o fluxo do programa continua normalmente. Caso contrário é processado novamente os comandos antes do teste da condição.

Em diagrama de Bloco



Em VisualG

Repita ... até

Esta estrutrura repete uma seqüência de comandos até que uma determinada condição (especificada através de uma expressão lógica) seja satisfeita.

repita

<seqüência-de-comandos>

ate <expressão-lógica>

repita Indica o início do laço.

ate <expressão-lógica> Indica o fim da <seqüência-de-comandos> a serem repetidos. Cada vez que o programa chega neste ponto, <expressão-lógica>é avaliada: se seu resultado for FALSO, os comandos presentes entre esta linha e a linha repitasão executados; caso contrário, a execução prosseguirá a partir do primeiro comando após esta linha.

Considerando ainda o mesmo exemplo:

algoritmo "Números de 1 a 10 (com repita)"

var j: inteiro

inicio

j <- 1

repita

escreva (j)

j <- j + 1

ate j > 10

fimalgoritmo

Importante:Como o laço repita...atetesta sua condição de parada depoisde executar sua seqüência de comandos, esta seqüência poderá ser executada uma ou mais vezes.

**7.3.3 Para ... faça**

Esta estrutura repete uma seqüência de comandos um determinado número de vezes.

para <variável>de <valor-inicial>ate <valor-limite>[passo <incremento>] faca

<seqüência-de-comandos>

fimpara

<variável> É a variável contadora que controla o número de repetições do laço. Na versão atual, deve ser necessariamente uma variável do tipo inteiro, como todas as expressões deste comando.

<valor-inicial> É uma expressão que especifica o valor de inicialização da variável contadora antes da primeira repetição do laço.

<valor-limite > É uma expressão que especifica o valor máximo que avariável contadora pode alcançar.

<incremento > É opcional. Quando presente, precedida pela palavra passo, é uma expressão que especifica o incremento que será acrescentado à variável contadora em cada repetição do laço. Quando esta opção não é utilizada, o valor padrão de <incremento> é 1. Vale a pena ter em conta que também é possível especificarvalores negativos para <incremento>. Por outro lado, se a avaliação da expressão <incremento> resultar em valor nulo, a execução do algoritmo será interrompida, com a impressão de uma mensagem de erro.

fimpara Indica o fim da seqüência de comandos a serem repetidos. Cada vez que o programa chega neste ponto, é acrescentado à variável contadora o valor de <incremento>, e comparado a <valor-limite >. Se for menor ou igual (ou maior ou igual, quando <incremento >for negativo), a seqüência de comandos será executada mais uma vez; caso contrário, a execução prosseguirá a partir do primeiro comando que esteja após o fimpara.

<valor-inicial >,<valor-limite > e<incremento >são avaliados uma única vezantes da execução da primeira repetição, e não se alteram durante a execução do laço, mesmo que variáveis eventualmente presentes nessas expressões tenham seus valores alterados.

No exemplo a seguir, os números de 1 a 10 são exibidos em ordem crescente.

algoritmo "Números de 1 a 10"

var j: inteiro

inicio

para j de 1 ate 10 faca

escreva (j)

fimpara

fimalgoritmo

**Importante:** Se, logo no início da primeira repetição, <valor-inicial > for maior que <valor-limite > (ou menor, quando <incremento>for negativo), o laço não será executado nenhuma vez. O exemplo a seguir não imprime nada.

algoritmo "Numeros de 10 a 1 (não funciona)"

var j: inteiro

inicio

para j de 10 ate 1 faca

escreva (j)

fimpara

fimalgoritmo

Este outro exemplo, no entanto, funcionará por causado passo -1:

algoritmo "Numeros de 10 a 1 (este funciona)"

var j: inteiro

inicio

para j de 10 ate 1 passo -1 faca

escreva (j)

fimpara

fimalgoritmo