Lógica de Programação

Protheus – Versão 12

Sumário

1. Introdução a Programação 3
1.1. Lógica de Programação e Algoritimos 3
1.2. Lógica de Programação 3
2. Desenvolvendo Algoritimos 5
2.1. Pseudocódigo 5
3. Estruturas de Programação 7
3.1. Diagrama de bloco 7
3.2. Representação de Algoritimos através de diagramas de bloco 8
4. Tipos de Dados 9
4.1. Dados Numéricos Inteiros 9
4.2. Dados Numéricos Reais 10
4.3. Dados Literais 10
4.4. Constantes Caractere 11
4.5. Cadeias de Caractere 11
4.6. Dados Lógicos 11
4.7. Vetores e Matrizes 12
5. Conceito e Utilidade de Variáveis 12
5.1. Definição de Variáveis em Algoritmos 13
6. Operadores da linguagem ADVPL 14
6.1. Operadores Matemáticos 14
6.2. Operadores de String 14
6.3. Operadores Relacionais 14
6.4. Operadores Lógicos 15
6.5. Operadores de Atribuição 15
6.6. Operadores de Incremento/Decremento 16
6.7. Operadores Especiais 17
7. Estruturas de decisão e repetição 17
7.1. Estruturas de decisão 18
7.2. Estruturas de repetição 20
8. Exercícios Propostos 23

Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

1. Introdução a Programação

1.1. Lógica de Programação e Algoritimos

No aprendizado de qualquer linguagem de programação é essencial desenvolver os conceitos selecionados a lógica e à técnica da escrita de um programa.

Com foco nesta necessidade, este tópico descreverá resumidamente os conceitos envolvidos no processo de desenvolvimento de um programa, através dos conceitos relacionados a:

- Lógica de programação
- Algoritmos
- Diagramas de blocos

1.2. Lógica de Programação

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

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

1.2.1. Sequência Lógica

Estes pensamentos podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

> Sequência Lógica são passos executados até atingir um objetivo ou solução de um problema.

1.2.2. Instruções

Na linguagem comum, entende-se por instruções “um conjunto de regras ou normas definidas para a realização ou emprego de algo”.
Em informática, porém, instrução é a informação 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.
Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc. 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.

1.2.3. Algoritmo

Um algoritmo é formalmente uma sequência finita de passos que levam à execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência 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 e decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um DVD, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples podem ser descritas por sequências lógicas, tais como:

- "Somar dois números quaisquer"
  1. Escreva o primeiro número no retângulo A;
  2. Escreva o segundo número no retângulo B;
  3. Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C

Um algoritmo não precisa ser linear, pode ter pontos de interrupção como While, If, For e Etc.

Como podemos perceber, um programa nada mais é que um tipo de algoritmo. Sua particularidade é que suas operações são específicas para o computador e restritas ao conjunto de instruções que o processador pode executar. Podemos considerar esse conjunto de instruções como a primeira linguagem de programação do computador, também chamada de linguagem de máquina. Classificamos as linguagens de programação segundo a sua proximidade com a linguagem de máquina. Quanto maior a semelhança com a linguagem de máquina, mais baixo é o nível da linguagem. As linguagens de programação mais semelhantes à linguagem de máquina são conhecidas como linguagens de baixo nível. Analogamente, linguagens de programação "distantes" da linguagem de máquina são conhecidas como linguagens de programação de alto nível. Linguagens de programação de alto nível são mais próximas da linguagem natural e guardam pouca similaridade com a linguagem da máquina em que serão executadas.
A linguagem de programação que um computador é capaz de compreender é composta apenas de números. Assim, fazer algoritmos na linguagem de programação do computador ou em sua linguagem de máquina é um processo extremamente complicado para nós, seres humanos, acostumados com a nossa própria linguagem. Por isso, para facilitar a programação de computadores, foi necessária a criação de um código que relacionasse a linguagem de máquina a uma linguagem mais fácil de ser compreendida exemplo: VB6, Java, Clipper, ADVPL, C# e etc.

Quando criamos um algoritmo em uma linguagem de programação é chamado de implementação, termo muito comum no desenvolvimento do programa.

2. Desenvolvendo Algoritimos

2.1. Pseudocódigo

É uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples (nativa a quem o escreve, de forma a ser entendida por qualquer pessoa) sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. Esta forma de representação de algoritmos é rica em detalhes, como a definição dos tipos das variáveis usadas no algoritmo. Na verdade, está representação é suficientemente para permitir a tradução de um algoritmo, sendo representado para uma linguagem de programação específica.

Por isso os algoritmos são independentes das linguagens de programação, sendo que 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.

2.1.1. Regras para construção do Algoritmo

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

1. Usar somente um verbo por frase;
2. Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática;
3. Usar frases curtas e simples;
4. Ser objetivo;

Procurar usar palavras que não tenham sentido dúbio.

2.1.2. Fases

Para implementar um algoritmo de simples interpretação e codificação é necessário inicialmente dividir o problema apresentado em três fases fundamentais, as quais são:

- **ENTRADA**: São os dados de entrada do algoritmo;
- **PROCESSAMENTO**: São os procedimentos utilizados para chegar ao resultado final;
- **SAÍDA**: São os dados já processados.

2.1.3. Estudando algoritmos

Neste tópico serão demonstrados alguns algoritmos do cotidiano, os quais foram implementados utilizando os princípios descritos nos tópicos anteriores.

- **Utilizar um telefone público – cartão**
- **Trocar lâmpadas**

Calcular a média de notas

Utilizar um telefone público - cartão

1. Retirar o telefone do gancho.
2. Esperar o sinal.
3. Colocar o cartão.
4. Discar o número.
5. Falar no telefone.
6. Colocar o telefone no gancho.
7. Retirar o cartão.

Trocar lâmpadas

1. Se a lâmpada estiver fora do alcance, pegar uma escada.
2. Pegar a lâmpada nova.
3. Se a lâmpada queimada estiver quente, pegar um pano.
4. Tirar lâmpada queimada.
5. Colocar lâmpada nova.

Calcular a média de notas

1. Notas a serem recebidas:
   1.1. Receber a nota.
   1.2. Contar as notas recebidas.
2. Some todas as notas recebidas.
3. Divida o total obtido pela quantidade de notas recebidas.
4. Exiba a média das notas.

2.1.4. 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.

Para avaliar a aplicação do teste de mesa, utilizaremos o algoritmo de calcular a média de notas:

Algoritmo: Calcular a média de notas

1. Para cada nota informada, receber e registrar na tabela abaixo:
   ID Nota

2. Ao término das notas, a tabela deverá conter todas as notas informadas, como abaixo:

ID	Nota
1	8.0

2	7.0
3	8.0
4	8.0
5	7.0
6	7.0

3. Somar todas as notas: 45

4. Dividir a soma das notas, pelo total de notas informado: 45/6 -> 7.5

5. Exibir a média obtida: Mensagem (Média: 7.5)

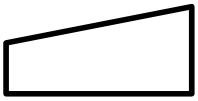
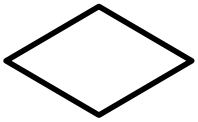
3. Estruturas de Programação

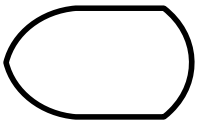
3.1. Diagrama de bloco

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 sequência de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

Existem diversos símbolos em um diagrama de bloco. No quadro abaixo estão representados alguns dos símbolos mais utilizados:

Símbolo	Função
 **Terminador**	Indica o início e o fim de um processamento.
 **Processamento**	Processamento em geral.
 **Entrada Manual**	Indica a entrada de dados através do teclado.
 **Decisão**	Indica um ponto no qual deverá ser efetuada uma escolha entre duas situações possíveis.

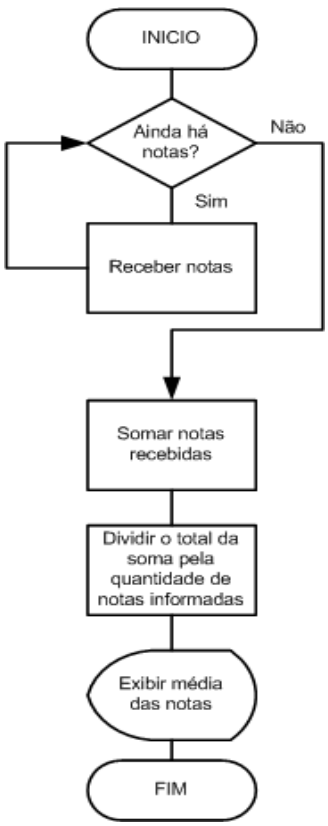
 **Exibição**	Mostra os resultados obtidos com um processamento.
 **Documento**	Indica um documento utilizado pelo processamento, seja para entrada de informações ou para exibição dos dados disponíveis após um processamento.

Cada símbolo irá conter uma descrição pertinente à forma com o qual o mesmo foi utilizado no fluxo, indicando o processamento ou a informação que o mesmo representa.

3.2. Representação de Algoritimos através de diagramas de bloco

Calcular a média de notas

1. Enquanto houver notas a serem recebidas:
    a. Receber a nota.
    b. Contas as notas recebidas
2. Some todas as notas recebidas.
3. Divida o total obtido pela quantidade de notas recebidas.4. Exiba a média das notas.



Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

4. Tipos de Dados

Todo o trabalho realizado por um computador é baseado na manipulação das informações contidas em sua memória. Os algoritmos irão manipular dados, que normalmente são fornecidos pelos usuários, e entregar resultados para estes usuários. Uma pergunta importante neste momento é: que tipo de dados poderemos manipular? As linguagens de programação normalmente estabelecem regras precisas para definir que tipos de dados elas irão manipular.
Existem três tipos básicos de dados que iremos manipular nos algoritmos que iremos criar:

- Dados numéricos
- Dados literais ou alfanuméricos
- Dados lógicos

4.1. Dados Numéricos Inteiros

O conjunto dos dados inteiros pode ser definido como

Z={...,-3,-2,0,1,2,...}.

As linguagens usadas para programar computadores são muito exigentes com a maneira com que os dados são representados. Por esta razão vamos passar a definir como deveremos representar os dados nos algoritmos. Os dados inteiros têm a seguinte forma:

NúmeroInteiro = [+,-]algarismo{algarismo}

Neste texto iremos usar uma notação especial para definir como iremos representar os elementos da linguagem. A medida que formos apresentando os elementos a notação será também explicada. No caso da definição dos dados inteiros temos os seguintes elementos. O elemento básico é o algarismo que é um dos seguintes caracteres:

Algarismo = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Os elementos entre colchetes são opcionais. Portanto, o sinal de + e - entre colchetes significa que um número inteiro pode ou não ter sínal. Em seguida temos um algarismo que é obrigatório. Isto é dados inteiros tem de ter pelo menos um algarismo. A seguir temos a palavra algarismo entre chaves, o que significa que um número inteiro deve ter pelo menos um algarismo e pode ser seguido por uma sequência de algarismos. Deste modo elementos que aparecem entre chaves são elementos que podem ser repetidos.

São portanto exemplos de números inteiros:

a) +3
b) 3
c) -324

Como exemplos de formas erradas de representação de números inteiros, temos:

a) +3.0 Não é possível usar ponto decimal
b) + 123 Espaços em branco não são permitidos

4.2. Dados Numéricos Reais

Os dados reais têm a seguinte forma:

[+,-]algarismo{algarismo}.algarismo{algarismo}.

Ou seja um número real pode ou não ter sinal, em seguida um conjunto de pelo menos um algarismo, um ponto decimal e depois um conjunto de pelo menos um algarismo. É importante notar que o separador entre a parte inteira e a fracionário é o ponto e não a vírgula.

São exemplos de números reais:

a) 0.5
b) +0.5
c) -3.1415

Abaixo mostramos exemplos de formas erradas de representação de números reais, temos:

a)	+3,0	Vírgula não pode ser separador entre as partes real e inteira
b)	0.333...	Não é possível usar representação de dízimas

4.3. Dados Literais

Dados literais servem para tratamento de textos. Por exemplo, um algoritmo podem necessitar de imprimir um aviso para os usuários, ou um comentário junto com um resultado numérico. Outra possibilidade é a necessidade de ler dados tais como nomes, letras, etc.

Este tipo de dados pode ser composto por um único caracter ou por um conjunto de pelo menos um destes elementos. Conjuntos são conhecidos como cadeias de caracteres, tradução da expressão em inglês, "*character string*".

Um ponto importante que deve ser abordado agora é o que se entende por caractere. Caracteres são basicamente as letras minúsculas, maiúsculas, algarismos, sinais de pontuação, etc. Em computação caracteres são representados por códigos binários e o mais disseminado de todos é o código ASCII. Este padrão foi definido nos Estados Unidos e é empregado pela quase totalidade dos fabricantes de computadores e programas. O apêndice mostra a tabela ASCII com estes códigos.

Os caracteres que normalmente são empregados nos algoritmos são os seguintes:

Letras maiúsculas:

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Letras minúsculas:

a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

Algarismos:

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Caracteres de pontuação:

; | : | ! | ? | * | (|) | \ | / | + | - | = | < | >

4.4. Constantes Caractere

Caracteres podem aparecer sozinhos, e neste caso são chamados de constante caracter e são representados entre o caracter '. Abaixo mostramos exemplos de constantes caracter:

- 'a'
- 'A'
- ';'
- '+'

4.5. Cadeias de Caractere

Cadeias de caracteres são conjuntos de um ou mais caracteres e são cercados pelo caracter ". Por exemplo:

- "Linguagem de programação"
- "Qual é o seu nome?"
- "12345"

4.6. Dados Lógicos

Este tipo de dados é intensamente aplicado durante o processo de tomada de decisões que o computador frequentemente é obrigado a fazer. Em muitos textos este tipo de dados também é chamado de dados booleanos, devido a George Boole, matemático que deu ao nome à álgebra (álgebra booleana) que manipula este tipo de dados. Os dados deste tipo somente podem assumir os valores: valores sim/não, 1/0, true/false.
Computadores tomam decisões, durante o processamento de um algoritmo, baseados nestes dois valores. Por exemplo, considere a sentença a seguir que é um caso típico de decisão que o computador é capaz de tomar sem intervenção humana.

- **Se está chovendo então procurar guarda-chuva.**

Nesta sentença temos a questão lógica "Se está chovendo". Esta expressão somente pode ter como resultado um de dois valores: verdade ou falso. Nos nossos algoritmos estes valores serão representados por verdade e falso. Mais adiante ficará claro como este tipo de dados será empregado nos algoritmos.

4.7. Vetores e Matrizes

4.7.1. Vetor

Um Vetor (Array) ou Arranjo é o nome de uma matriz unidimensional considerada a mais simples das estruturas de dados. Geralmente é constituída por dados do mesmo tipo (homogêneos) e tamanho que são agrupados continuamente na memória e acessados por sua posição (índice - geralmente um número inteiro) dentro do vetor. Na sua inicialização determina-se o seu tamanho que geralmente não se modifica mesmo que utilizemos menos elementos.

Abaixo temos o exemplo de um vetor. Os valores internos seriam os dados alocados no vetor, enquanto seu tamanho é dado pelo número de casas disponíveis (no caso 8) e o índice representa a posição do dado no vetor (por exemplo podemos definir que 0 tem o índice 1, 2 tem índice 2, 8 tem índice 3 e assim sucessivamente).

0	2	8	9	10	11	15	18

Podemos trabalhar com vetores numéricos, executando operações sobre eles da mesma forma como executaríamos com variáveis numéricas comuns. Devemos assumir que ao declararmos um determinado vetor[índice], com um índice específico, estamos fazendo referência a um número.

4.7.2. Matriz

Matrizes são arranjos ordenados que ao contrário dos vetores podem ter n dimensões, sendo que estas dimensões lhes dão o nome n-dimensional . Uma matriz de duas dimensões será chamada bi-dimensional, uma de três dimensões tri-dimensional e assim consecutivamente. Funciona praticamente da mesma forma que um vetor exceto que utilizaremos o número n de índices para acessar um dado que queremos. Para efeitos de estudo por enquanto nos limitaremos somente às matrizes bidimensionais (duas dimensões linha X colunas). Assim se possuimos uma matriz bidimensional de duas linhas e duas colunas:

3	4
5	6

Consideramos que para acessarmos o valor 3, localizamos o índice por sua linha (1) e coluna (1), deste modo seu índice é (1,1). O valor quatro por exemplo será (1, 2).

5. Conceito e Utilidade de Variáveis

Como visto anteriormente, informações correspondentes a diversos tipos de dados são armazenadas na memória dos computadores. Para acessar individualmente cada uma destas informações, a princípio, seria necessário saber o tipo de dado desta informação (ou seja, o número de bytes de memória por ela ocupados) e a posição inicial deste conjunto de bytes na memória.

Percebe-se que está sistemática de acesso a informações na memória é bastante ilegível e difícil de se trabalhar. Para contornar esta situação criou-se o conceito de **variável**, que é uma entidade destinada a guardar uma informação.

Basicamente, uma variável possui três atributos: um nome, um tipo de dado associado à mesma e a informação por ela guardada.

- Nome
- Tipo de
- Dado
- Informação

Toda variável possui um nome que tem a função de diferenciá-la das demais. Cada linguagem e programação estabelece suas próprias regras de formação de nomes de variáveis. Adotaremos nesta apostila as seguintes regras:

- Nome de variável deve necessariamente começar com uma letra;
- Nome de variável não deve conter nenhum símbolo especial exceto a sublinha (_).

Exemplos:

1ANO	Errado (não começou com uma letra)
ANO1	Correto
A CASA	Errado (contém o caractere espaço em branco)
SAL/HORA	Errado (contém o caractere "/")
SAL_HORA	Correto
@DESCONTO	Errado (não começou com uma letra)

5.1. Definição de Variáveis em Algoritmos

Todas as variáveis utilizadas em algoritmos devem ser definidas antes de serem utilizadas. Isto se faz necessário para permitir que o compilador reserve um espaço na memória para as mesmas. Nos algoritmos apresentados nesta apostila será adotada a seguinte convenção, todas as variáveis utilizadas em algoritmos serão definidas no início do mesmo, por meio de um comando de uma das formas seguintes:

- **VAR** <nome_da_variável> : <tipo_da_variável>
- **VAR** <lista_de_variáveis> : <tipo_das_variáveis>

Palavra-chave **VAR** deverá estar presente sempre e será utilizada uma única vez na definição de um conjunto de uma ou mais variáveis, numa mesma linha poderão ser definidas uma ou mais variáveis do mesmo tipo. Para tal, deve-se separar os nomes das mesmas por vírgulas;

- Variáveis de tipos diferentes devem ser declaradas em linhas diferentes.

A forma de utilização deste comando ficará mais clara quando da utilização da representação de algoritmos em linguagem estruturada (pseudocódigo).

Esta convenção é válida para a representação de algoritmos na forma de pseudocódigo.
Em termos de fluxograma, não é usual adotar-se qualquer forma de definição de variáveis.

Exemplo de definição de variáveis:

```
VAR NOME : literal[10]
IDADE : inteiro
SALARIO : real
TEM_FILHOS : lógico
```

No exemplo acima foram declaradas quatro variáveis:

- A variável **NOME**, capaz de armazenar dados literais de comprimento 10 (dez caracteres);
- A variável **IDADE**, capaz de armazenar um número inteiro;
- A variável **SALARIO**, capaz de armazenar um número real;
- A variável **TEM_FILHOS**, capaz de armazenar uma informação lógica.

6. Operadores da linguagem ADVPL

6.1. Operadores Matemáticos

Os operadores utilizados em ADVPL para cálculos matemáticos são:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
** ou ^	Exponenciação
%	Módulo (Resto da Divisão)

6.2. Operadores de String

Os operadores utilizados em ADVPL para tratamento de caracteres são:

+	Concatenação de strings (união).
-	Concatenação de strings com eliminação dos brancos finais das strings intermediárias.
$	Comparação de Substrings (contido em).

6.3. Operadores Relacionais

Os operadores utilizados em ADVPL para operações e avaliações relacionais são:

<	Comparação Menor
>	Comparação Maior
=	Comparação Igual
==	Comparação Exatamente Igual (para caracteres)

Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

<=	Comparação Menor ou Igual
>=	Comparação Maior ou Igual
<> ou # ou !=	Comparação Diferente

6.4. Operadores Lógicos

Os operadores utilizados em ADVPL para operações e avaliações lógicas são:

.And.	E lógico
.Or.	OU lógico
.Not. ou !	NÃO lógico

6.5. Operadores de Atribuição

Os operadores utilizados em ADVPL para atribuição de valores a variáveis de memória são:

:=	Atribuição Simples
+=	Adição e Atribuição em Linha
-=	Subtração e Atribuição em Linha
*=	Multiplicação e Atribuição em Linha
/=	Divisão e Atribuição em Linha
**= ou ^=	Exponenciação e Atribuição em Linha

6.5.1.Atribuição Simples

O sinal de igualdade é utilizado para atribuir valor a uma variável de memória.

```
nVariavel := 10
```

6.5.2.Atribuição em Linha

O operador de atribuição em linha é caracterizado por dois pontos e o sinal de igualdade. Tem a mesma função do sinal de igualdade sozinho, porém aplica a atribuição às variáveis. Com ele pode-se atribuir mais de uma variável ao mesmo tempo.

```
nVar1 := nVar2 := nVar3 := 0
```

Quando diversas variáveis são inicializadas em uma mesma linha, a atribuição começa da direita para a esquerda, ou seja, nVar3 recebe o valor zero inicialmente, nVar2 recebe o conteúdo de nVar3 e nVar1 recebe o conteúdo de nVar2 por final. Com o operador de atribuição em linha, pode-se substituir as inicializações individuais de cada variável por uma inicialização apenas:

```
nVar1 := 0, nVar2 := 0, nVar3 := 0
por
nVar1 := nVar2 := nVar3 := 0
```

O operador de atribuição em linha também pode ser utilizado para substituir valores de campos em um banco de dados.

6.5.3.Atribuição Composta

Os operadores de atribuição composta são uma facilidade da linguagem ADVPL para xpressões de cálculo e atribuição. Com eles pode-se economizar digitação:

Operador	Exemplo	Equivalente
+=	X += Y	X = X + Y
-=	X -= Y	X = X - Y
*=	X *= Y	X = X * Y
/=	X /= Y	X = X / Y
**= ou ^=	X **= Y	X = X ** Y
%=	X %= Y	X = X % Y

6.6. Operadores de Incremento/Decremento

A linguagem ADVPL possui operadores para realizar incremento ou decremento de variáveis. Entende-se por incremento aumentar o valor de uma variável numérica em 1 e entende-se por decremento diminuir o valor da variável em 1. Os operadores são:

++	Incremento Pós ou Pré-fixado
--	Decremento Pós ou Pré-fixado

Os operadores de decremento/incremento podem ser colocados tanto antes (pré-fixado) como depois (pós-fixado) do nome da variável. Dentro de uma expressão, a ordem do operador é muito importante, podendo alterar o resultado da expressão. Os operadores incrementais são executados da esquerda para a direita dentro de uma expressão.

```
Local nA := 10
Local nB := nA++ + nA
```

O valor da variável nB resulta em 21, pois a primeira referência a nA (antes do ++) continha o valor 10 que foi considerado e imediatamente aumentado em 1. Na segunda referência a nA, este já possuía o valor 11. O que foi efetuado foi a soma de 10 mais 11, igual a 21. O resultado final após a execução destas duas linhas é a variável nB contendo 21 e a variável nA contendo 11.
No entanto:

```
nA := 10
nB := ++nA + nA
```

Resulta em 22, pois o operador incremental aumentou o valor da primeira nA antes que seu valor fosse considerado.

Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

6.7. Operadores Especiais

Além dos operadores comuns, o ADVPL possui alguns outros operadores ou identificadores. Estas são suas finalidades:

()	Agrupamento ou Função
[]	Elemento de Matriz
{}	Definição de Matriz, Constante ou Bloco de Código
->	Identificador de Apelido
&	Macro substituição
@	Passagem de parâmetro por referência
||	Passagem de parâmetro por valor

- Os parênteses são utilizados para agrupar elementos em uma expressão, mudando a ordem de precedência da avaliação da expressão (segundo as regras matemáticas por exemplo). Também servem para envolver os argumentos de uma função.

- Os colchetes são utilizados para especificar um elemento específico de uma matriz.
  Por exemplo, A[3,2] refere-se ao elemento da matriz A na linha 3, coluna 2.

- As chaves são utilizadas para a especificação de matrizes literais ou blocos de código. Por exemplo, A:={10,20,30} cria uma matriz chamada A com três elementos.

- O símbolo **->** identifica um campo de um arquivo, diferenciando-o de uma variável.
  Por exemplo, FUNC->nome refere-se ao campo nome do arquivo FUNC. Mesmo que exista uma variável chamada nome, é o campo nome que será acessado.

- O símbolo **&** identifica uma avaliação de expressão através de macro e é visto em detalhes na documentação sobre **macro substituição**.

- O símbolo **@** é utilizado para indicar que durante a passagem de uma variável para uma função ou procedimento ela seja tomada como uma referência e não como valor.

- O símbolo **||** é utilizado para indicar que durante a passagem de uma variável para uma função ou procedimento, ela seja tomada como um e valor não como referência.

7. Estruturas de decisão e repetição

A utilização de estruturas de decisão e repetição em um algoritmo permite a realização de ações relacionadas a situações que influenciam na execução e solução do problema.

Como foco na utilização da linguagem ADVPL, serão ilustradas as seguintes estruturas:

- **Estruturas de decisão**
  - IF...ELSE
  - DO CASE ... CASE

- **Estruturas de repetição**
  - WHILE...END
  - FOR...NEXT

7.1. Estruturas de decisão

Os comandos de decisão são utilizados em algoritmos cuja solução não é obtida através da utilização de ações meramente sequenciais, permitindo que estes comandos de decisão avaliem as condições necessárias para optar por uma ou outra maneira de continuar seu fluxo.

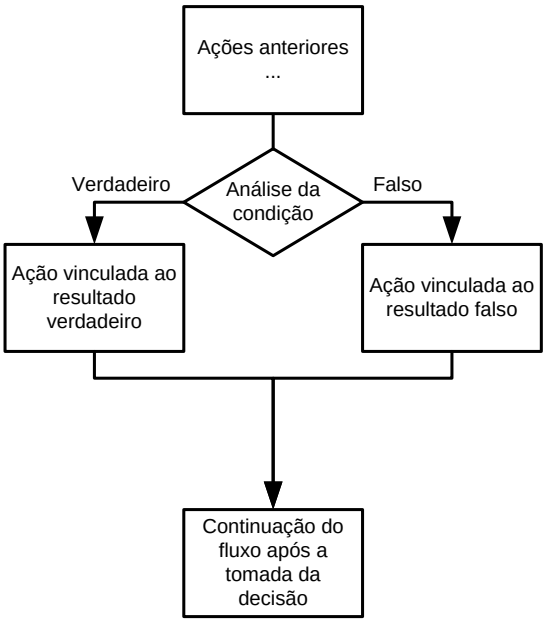
As estruturas de decisão que serão analisadas são:

- **IF...ELSE**
- **DO CASE ... CASE**

7.1.1.IF...ELSE

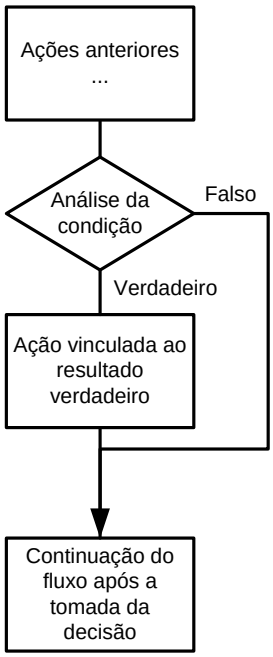
A estrutura IF...ELSE (Se/Senão) permite a análise de uma condição e a partir da qual ser executada uma de duas ações possíveis: Se a análise da condição resultar em um valor verdadeiro ou se a análise da condição resultar em um valor falso.

Representação 01: IF...ELSE com ações para ambas as situações



Esta estrutura permite ainda que seja executada apenas uma ação, na situação em que a análise da condição resultar em um valor verdadeiro.

Representação 02: IF...ELSE somente com ação para situação verdadeira

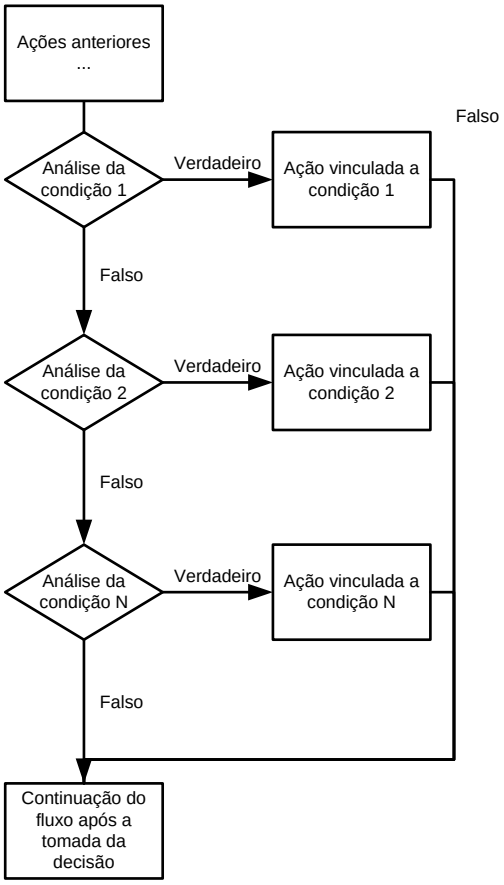


Apesar das linguagens de programação possuírem variações para a estrutura IF...ELSE, conceitualmente todas as representações podem ser descritas com base no modelo apresentado.
Com esta estrutura é possível realizar a análise de diversas condições em sequência, para as quais será avaliada somente a ação da primeira expressão cujo análise resultar em um valor verdadeiro.

7.1.2.DO CASE...CASE

A estrutura DO CASE...ENDCASE (Caso) permite a análise de diversas condições consecutivas, para as quais somente a condição para a primeira condição verdadeira será sua ação vinculada e executada.

O recurso de análise de múltiplas condições é necessário para solução de problemas mais complexos, nos quais as possibilidades de solução superam a mera análise de um único resultado verdadeiro ou falso.



Apesar das linguagens de programação possuírem variações para a estrutura DO CASE...CASE, conceitualmente todas as representações podem ser descritas com base no modelo apresentado.

7.2. Estruturas de repetição

Os comandos de repetição são utilizados em algoritmos nas situações em que é necessário realizar uma determinada ação ou um conjunto de ações para um número definido ou indefinido de vezes, ou ainda enquanto uma determinada condição for verdadeira.

As estruturas de decisão que serão analisadas são:

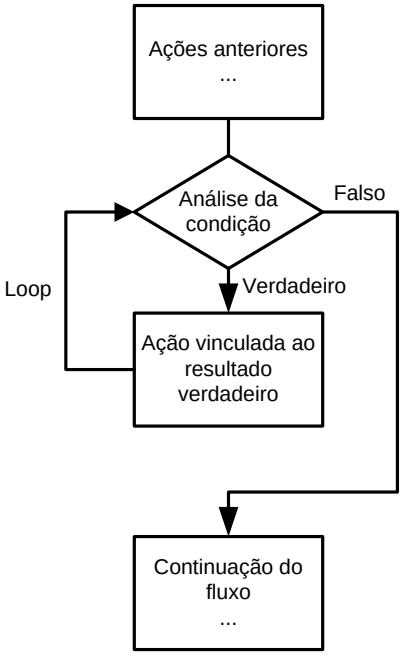
- **WHILE...END**
- **FOR...TO...NEXT**

Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

7.2.1.HILE...END

Nesta estrutura, o conjunto de ações será executado enquanto a análise de uma condição de referência resultar em um valor verdadeiro. É importante verificar que o bloco somente será executado, inclusive se na primeira análise a condição resultar em um valor verdadeiro.

Representação: WHILE...END



Existem diversas variações para a estrutura WHILE...END, na qual há a possibilidade da primeira execução ser realizada sem a análise da condição, a qual valerá apenas a partir da segunda execução.

A linguagem ADVPL aceita a sintaxe DO WHILE...ENDDO, que em outras linguagens representa a situação descrita anteriormente (análise da condição somente a partir da segunda execução), mas em ADVPL está sintaxe tem o mesmo efeito do WHILE...END.

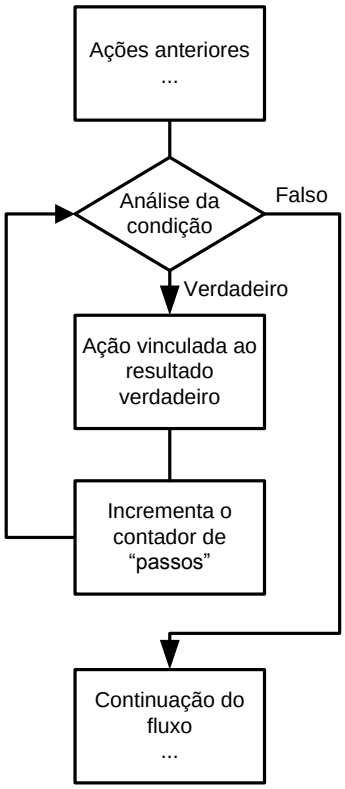
7.2.2.FOR...TO...NEXT

Nesta estrutura, o conjunto de ações será executado uma quantidade de vezes definida, normalmente referenciada como “passo”.

Para cada “passo” realizado pela estrutura FOR...TO...NEXT, será avaliada uma condição que verificará se foi atingido o número de execuções previamente definido. Desta forma a estrutura compreende um controle de número de “passos” executados, o qual é incrementado na análise da expressão NEXT.

Semelhante a estrutura WHILE...END, a primeira ação somente será realizada mediante um resultado verdadeiro na análise da condição.

Representação: FOR...TO...NEXT



A estrutura FOR...TO...NEXT, dependendo da linguagem de programação, permite a realização de um incremento simples a cada execução da instrução NEXT, ou a adição de outro valor ao contador, o qual deverá especificado de acordo com a sintaxe da linguagem. Em ADVPL pode ser utilizada a instrução “STEPS” para alterar o valor a ser adicionado no contador de passos a cada execução da instrução NEXT, sendo que este valor poderá ser até negativo, viabilizando uma contagem decrescente.

Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©
Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

8. Exercícios Propostos

1. Dada a declaração de variáveis:

VAR A, B, C : **inteiro**
X, Y, Z : **real**
NOME, RUA : **literal[**20**]**
L1, L2 : **lógico**

Classifique as expressões seguintes de acordo com o tipo de dado do resultado de sua validação, em **I** (inteiro), **R** (real), **L** (literal), **B** (lógico) ou **N** (quando não for possível defini-lo):

( ) A + B + C
( ) A + B + Z
( ) NOME + RUA
( ) A B
( ) A Y
( ) NOME RUA
( ) L1 .OU. L2
( ) RUA <> NOME
( ) A + B / C
( ) A + X / Z
( ) A + Z / A
( ) A B = L1
( ) (A = B)
( ) X + Y / Z
( ) X = Z / A
( ) L1 ** L2
( ) A + B / L2
( ) X < L1 / RUA

2. Para as mesmas variáveis declaradas no exercício 1, às quais são dados os valores seguintes:

A = 1
B = 2
C = 3
X = 2.0
Y = 10.0
Z = -1.0
L1 = .V.
NOME = “PEDRO”
RUA = “PEDRINHO”
L2 = .F.

Determine o resultado da avaliação das expressões abaixo:

A + C / B →
A + B + C →
C / B / A →
-X * B →

- (X * B) →
(-X) * B →
NOME + RUA →
NOME = RUA →
L1 .OU. L2 →
(L1 .E.(.NÃO. L2)) →
(L2 .E. (.NÃO. L1)) →
X Y .E. C = B →
(C - 3 * A) (X + 2 * Z) →

3. Escreva um algoritmo que armazene o valor 10 em uma variável A e o valor 20 em uma variável B. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em A passe para B e vice-versa. Ao final, escrever os valores que ficaram armazenados nas variáveis.

4. Calcule a média de quatro números inteiros dados.

5. Ler um valor e escrever se é positivo ou negativo (considere o valor **zero como positivo**).

6. O custo ao consumidor de um carro novo é a soma do custo de fábrica com a porcentagem do distribuidor e dos impostos, ambos aplicados ao custo de fábrica. Supondo que a porcentagem do distribuidor seja de 12% e a dos impostos de 45%, prepare um algoritmo para ler o custo de fábrica do carro e imprimir o custo ao consumidor.

7. O cardápio de uma lanchonete é dado abaixo. Prepare um algoritmo que leia a quantidade de cada item que você consumiu e calcule a conta final.

Hambúrguer..... R$ 3,00
Cheeseburger..... R$ 2,50
Fritas..... R$ 2,50
Refrigerante..... R$ 1,00
Milkshake..... R$ 3,00

8. Uma companhia de carros paga a seus empregados um salário de R$ 500,00 por mês mais uma comissão de R$ 50,00 para cada carro vendido e mais 5% do valor da venda. Elabore um algoritmo para calcular e imprimir o salário do vendedor num dado mês recebendo como dados de entrada o nome do vendedor, o número de carros vendidos e o valor total das vendas.

9. Faça um algoritmo que informa se o número digitado é ímpar ou par.

10. Faça um algoritmo para ler: quantidade atual em estoque, quantidade máxima em estoque e quantidade mínima em estoque de um produto. Calcular e escrever a quantidade média

((quantidademédia = quantidade máxima + quantidade mínima)/2).

Se a quantidade em estoque for maior ou igual a quantidade média escrever a mensagem Não efetuar compra', senão escrever a mensagem 'Efetuar compra'.

11. Ler dois valores e imprimir uma das três mensagens a seguir:

'Números iguais', caso os números sejam iguais
'Primeiro é maior', caso o primeiro seja maior que o segundo;
'Segundo maior', caso o segundo seja maior que o primeiro.

12. Classifique os dados especificados abaixo de acordo com seu tipo, assinalando com **I** os dados do tipo inteiro, com **R** os reais, com **C** os literais, com **L** os lógicos (booleanos), e com **X** aqueles para os quais não é possível definir *a priori* um tipo de dado.

( ) 0.21
( ) 1
( ) V
( ) "0."
( ) 1%
( ) "José"
( ) 0,35
( ) .F.
( ) -0.001
( ) .T.
( ) +3257
( ) "a"
( ) "+3257"
( ) +3257.
( ) "-0.0"
( ) ".F."
( ) ± 3
( ) .V.
( ) .V
( ) "abc"
( ) F
( ) C
( ) Maria
( ) +36

Escreva um algoritmo que mostra a tabuada de um número qualquer, fornecido pelo usuário.
Por exemplo, para a tabuada do 2, o resultado impresso deve ser:
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20