

# MÓDULO 01: INTRODUÇÃO À PROGRAMAÇÃO

---

## LÓGICA DE PROGRAMAÇÃO E ALGORITMOS

---

No aprendizado de qualquer linguagem de programação é essencial desenvolver os conceitos relacionados 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

### Lógica de Programação

---

#### Lógica

---

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

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

#### Seqüência Lógica

---

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

**Seqüência Lógica são passos executados até atingir um objetivo ou solução de um problema.**

#### 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 seqüencial 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.

**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.**

## Algoritmo

---

Um algoritmo é formalmente uma seqüência finita de passos que levam à execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma seqüência 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 DVD, que explicam passo-a-passo como, por exemplo, gravar um evento.

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

- ☑ “Chupar uma bala”
  1. Pegar a bala;
  2. Retirar o papel;
  3. Chupar a bala;
  4. Jogar o papel no lixo.
  
- ☑ “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.

## Desenvolvendo algoritmos

---

### 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 for utilizada a linguagem de programação propriamente dita como, por exemplo, ADVPL.

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.

### Regras para construção do Algoritmo

---

Para escrever um algoritmo precisamos descrever a seqüência 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;
5. Procurar usar palavras que não tenham sentido dúbio.

### 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.

## Estudando algoritmos







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



**Mascar um chiclete**



**Utilizar um telefone público – cartão**

-  **Fritar um ovo**
-  **Trocar lâmpadas**
-  **Descascar batatas**
-  **Jogar o jogo da forca**
-  **Calcular a média de notas**
-  **Jogar o jogo da velha – contra o algoritmo**

### **Mascar um chiclete**

---

1. Pegar o chiclete.
2. Retirar o papel.
3. Mastigar.
4. Jogar o papel no lixo.

### **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.

### **Fritar um ovo**

---

1. Pegar frigideira, ovo, óleo e sal.
2. Colocar óleo na frigideira.
3. Ascender o fogo.
4. Colocar a frigideira no fogo.
5. Esperar o óleo esquentar.
6. Quebrar o ovo na frigideira.
7. Jogar a casca no lixo.
8. Retirar o ovo da frigideira quando estiver no ponto.
9. Desligar o fogo da frigideira.
10. Colocar sal a gosto.

### **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.

### Descascar batatas

---

1. Pegar faca, bacia e batatas.
2. Colocar água na bacia.
3. Enquanto houver batatas, descascar as batatas.
  - 3.1. Colocar as batatas descascadas na bacia.

### Jogar o jogo da forca

---

1. Escolher a palavra.
2. Montar o diagrama do jogo.
3. Enquanto houver lacunas vazias e o corpo estiver incompleto:
  - 3.1. Se acertar a letra: escrever na lacuna correspondente.
  - 3.2. Se errar a letra: desenhar uma parte do corpo na forca.

### Calcular a média de notas

---

1. Enquanto houver 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.

### Jogar o jogo da velha – contra o algoritmo

---

1. Enquanto existir um quadrado livre e ninguém ganhou ou empatou o jogo:
  - 1.1. Espere a jogada do adversário, continue depois.
  - 1.2. Se centro estiver livre: jogue no centro.
  - 1.3. Senão, se o adversário possuir 2 quadrados em linha com um quadrado livre, jogue neste quadrado.
  - 1.4. Senão, se há algum canto livre, jogue neste canto.

### 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. Enquanto houver notas a serem recebidas:
  - a. Receber a nota.

- b. 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.

### Teste de mesa:

---

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:  $45/6 \rightarrow 7.5$
5. Exibir a média obtida: Mensagem (Média: 7.5)

### Exercícios

Aprimorar os seguintes algoritmos descritos na apostila:

- ☒ Usar telefone público – cartão.
- ☒ Fritar um ovo.
- ☒ Mascar um chiclete.
- ☒ Trocar lâmpadas.
- ☒ Descascar batatas.
- ☒ Jogar o “Jogo da Força”.

# ESTRUTURAS DE PROGRAMAÇÃO

---

## Diagrama de bloco

---




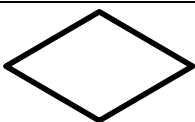
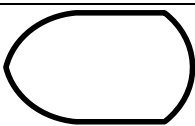

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

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.

### Simbologia

---

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

Símbolo	Função
 <b>Terminador</b>	Indica o início e o fim de um processamento.
 <b>Processamento</b>	Processamento em geral.
 <b>Entrada Manual</b>	Indica a entrada de dados através do teclado.
 <b>Decisão</b>	Indica um ponto no qual deverá ser efetuada uma escolha entre duas situações possíveis.
 <b>Exibição</b>	Mostra os resultados obtidos com um processamento.
 <b>Documento</b>	Indica um documento utilizado pelo processamento, seja para entrada de informações ou para exibição dos dados disponíveis após um processamento.

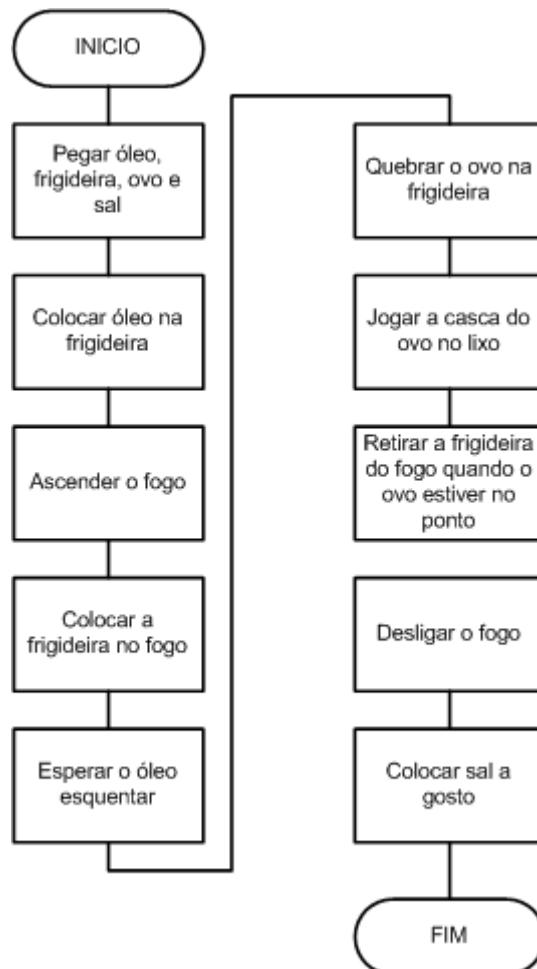


Cada símbolo irá conter uma descrição pertinente à forma com o qual o mesmo foi utilizado no fluxo, indicando o processamento ou a informação que o mesmo representa.

## Representação de algoritmos através de diagramas de bloco

### Algoritmo 01: Fritar um ovo

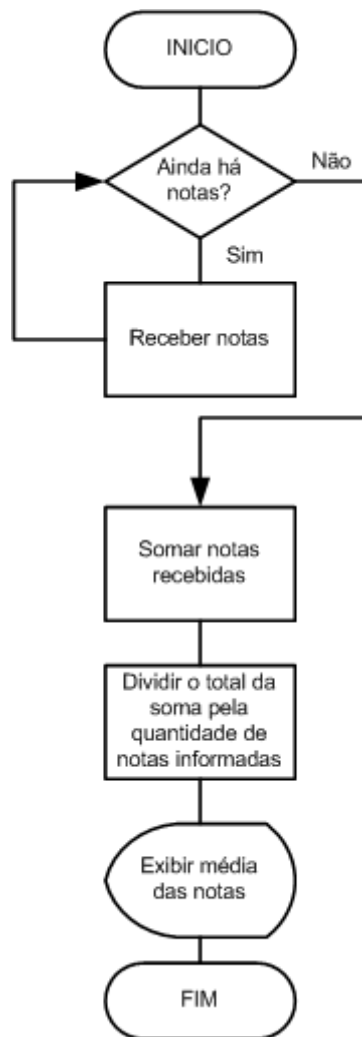
1. Pegar frigideira, ovo, óleo e sal.
2. Colocar óleo na frigideira.
3. Ascender o fogo.
4. Colocar a frigideira no fogo.
5. Esperar o óleo esquentar.
6. Quebrar o ovo na frigideira.
7. Jogar a casca no lixo.
8. Retirar o ovo da frigideira quando estiver no ponto.
9. Desligar o fogo da frigideira.
10. Colocar sal a gosto.





### Algoritmo 02: 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.



## 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

### Estruturas de decisão

Os comandos de decisão são utilizados em algoritmos cuja solução não é obtida através da utilização de ações meramente seqüenciais, 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:

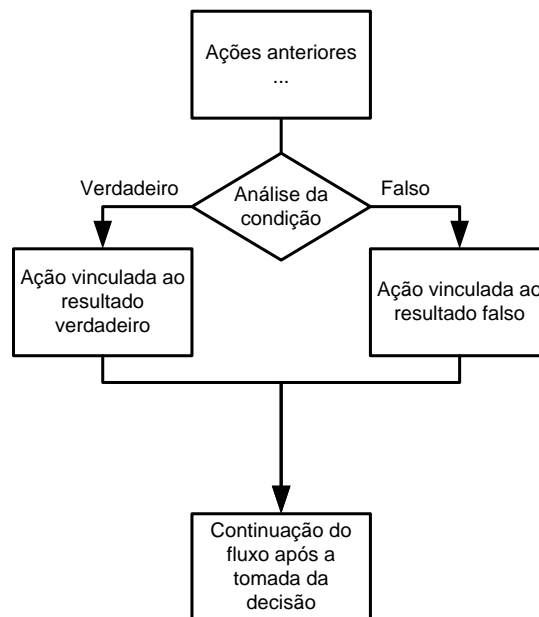
- ☑ **IF...ELSE**
- ☑ **DO CASE ... CASE**

#### IF...ELSE

---

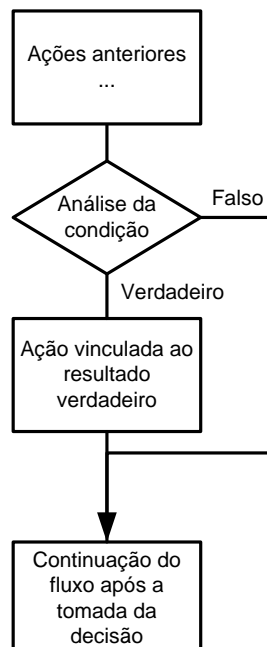
A estrutura IF...ELSE (Se/Senão) permite a análise de uma condição e a partir da qual ser executada uma de duas ações possíveis: Se a análise da condição resultar em um valor verdadeiro ou se a análise da condição resultar em um valor falso.

### 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





*Importante*

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



*Dica*

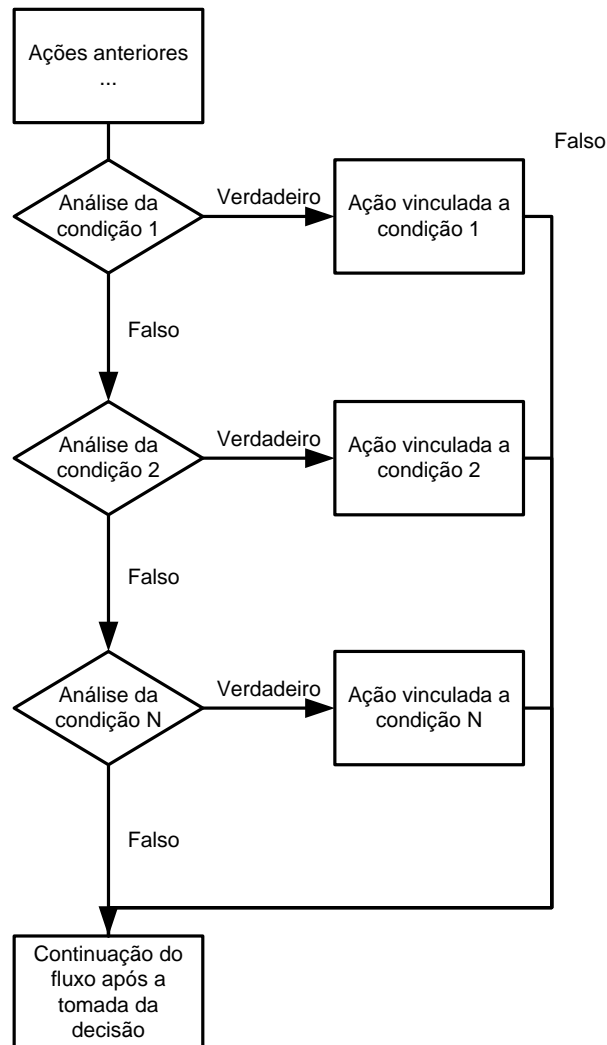
A linguagem ADVPL possui uma variação para a estrutura IF...ELSE, descrita como IF...ELSEIF...ELSE.

Com esta estrutura é possível realizar a análise de diversas condições em seqüência, para as quais será avaliada somente a ação da primeira expressão cujo análise resultar em um valor verdadeiro.

## 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.



**Importante**

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.

## 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:

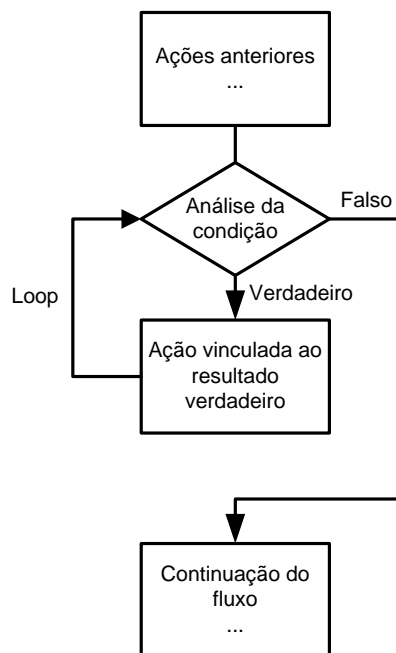
- ☑ **WHILE...END**
- ☑ **FOR...TO...NEXT**

### WHILE...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 esta sintaxe tem o mesmo efeito do WHILE...END.

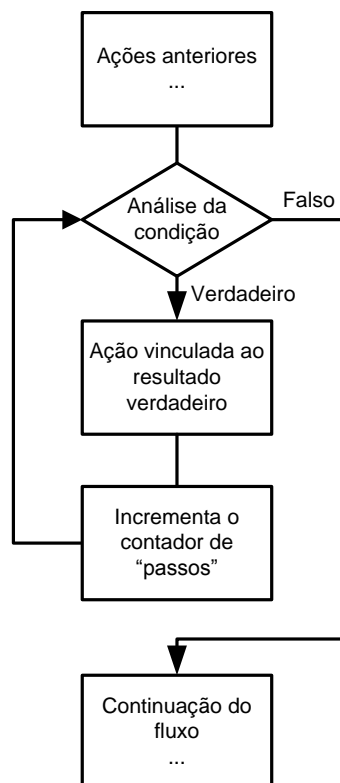
## FOR...TO...NEXT

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





**Importante**

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.

### Exercícios

Montar os diagramas de blocos para os algoritmos desenvolvidos no exercício anterior:

- ☒ Usar telefone público – cartão.
- ☒ Fritar um ovo.
- ☒ Mascar um chiclete.
- ☒ Trocar lâmpadas.
- ☒ Descascar batatas.
- ☒ Jogar o “Jogo da Forca”.



## Operadores da linguagem ADVPL

### Operadores Matemáticos

Os operadores utilizados em ADVPL para cálculos matemáticos são:

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

### 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).

### Operadores Relacionais

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

<	Comparação Menor
>	Comparação Maior
=	Comparação Igual
==	Comparação Exatamente Igual (para caracteres)
<=	Comparação Menor ou Igual
>=	Comparação Maior ou Igual
<> ou # ou !=	Comparação Diferente

### Operadores Lógicos

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

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

## Operadores de Atribuição

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

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

### ☒ Atribuição Simples

O sinal de igualdade é utilizado para atribuir valor a uma variável de memória.  
`nVariavel := 10`

### ☒ 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.

### ☒ **Atribuição Composta**

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

Operador	Exemplo	Equivalente a
<code>+=</code>	<code>X += Y</code>	<code>X = X + Y</code>
<code>-=</code>	<code>X -= Y</code>	<code>X = X - Y</code>
<code>*=</code>	<code>X *= Y</code>	<code>X = X * Y</code>
<code>/=</code>	<code>X /= Y</code>	<code>X = X / Y</code>
<code>**=</code> ou <code>^=</code>	<code>X **= Y</code>	<code>X = X ** Y</code>
<code>%=</code>	<code>X %= Y</code>	<code>X = X % Y</code>

### **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:

<code>++</code>	Incremento Pós ou Pré-fixado
<code>--</code>	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.

## 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.

## MÓDULO 02: A LINGUAGEM ADVPL

A Linguagem ADVPL teve seu início em 1994, sendo na verdade uma evolução na utilização de linguagens no padrão xBase pela Microsiga Software S.A. (Clipper, Visual Objects e depois FiveWin). Com a criação da tecnologia Protheus, era necessário criar uma linguagem que suportasse o padrão xBase para a manutenção de todo o código existente do sistema de ERP Siga Advanced. Foi então criada a linguagem chamada *Advanced Protheus Language*.

O ADVPL é uma extensão do padrão xBase de comandos e funções, operadores, estruturas de controle de fluxo e palavras reservadas, contando também com funções e comandos disponibilizados pela Microsiga que a torna uma linguagem completa para a criação de aplicações ERP prontas para a Internet. Também é uma linguagem orientada a objetos e eventos, permitindo ao programador desenvolver aplicações visuais e criar suas próprias classes de objetos.

Quando compilados, todos os arquivos de código tornam-se unidades de inteligência básicas, chamados APO's (de *Advanced Protheus Objects*). Tais APO's são mantidos em um repositório e carregados dinamicamente pelo PROTHEUS Server para a execução. Como não existe a linkedição, ou união física do código compilado a um determinado módulo ou aplicação, funções criadas em ADVPL podem ser executadas em qualquer ponto do Ambiente Advanced Protheus.

O compilador e o interpretador da linguagem ADVPL é o próprio servidor PROTHEUS (PROTHEUS Server), e existe um Ambiente visual para desenvolvimento integrado (PROTHEUS IDE), em que o código pode ser criado, compilado e depurado.

Os programas em ADVPL podem conter comandos ou funções de interface com o usuário. De acordo com tal característica, tais programas são subdivididos nas seguintes categorias:

### **Programação Com Interface Própria com o Usuário**

---

Nesta categoria, entram os programas desenvolvidos para serem executados através do terminal remoto do Protheus, o Protheus Remote. O Protheus Remote é a aplicação encarregada da interface e da interação com o usuário, sendo que todo o processamento do código em ADVPL, o acesso ao banco de dados e o gerenciamento de conexões é efetuado no Protheus Server. O Protheus Remote é o principal meio de acesso a execução de rotinas escritas em ADVPL no Protheus Server, e por isso permite executar qualquer tipo de código, tenha ele interface com o usuário ou não. Porém, nesta categoria são considerados apenas os programas que realizem algum tipo de interface remota, utilizando o protocolo de comunicação do Protheus.

Podem-se criar rotinas para a customização do sistema ERP Microsiga Protheus, desde processos adicionais até mesmo relatórios. A grande vantagem é aproveitar todo o

Ambiente montado pelos módulos do ERP Microsiga Protheus. Porém, com o ADVPL é possível até mesmo criar toda uma aplicação, ou módulo, do começo. Todo o código do sistema ERP Microsiga Protheus é escrito em ADVPL.

### **Programação Sem Interface Própria com o Usuário**

---

As rotinas criadas sem interface são consideradas nesta categoria porque geralmente têm uma utilização mais específica do que um processo adicional ou um relatório novo. Tais rotinas não têm interface com o usuário através do Protheus Remote, e qualquer tentativa nesse sentido (como a criação de uma janela padrão) ocasionará uma exceção em tempo de execução. Estas rotinas são apenas processos, ou Jobs, executados no Protheus Server. Algumas vezes, a interface destas rotinas fica a cargo de aplicações externas, desenvolvidas em outras linguagens, que são responsáveis por iniciar os processos no servidor Protheus, por meio dos meios disponíveis de integração e conectividade no Protheus.

De acordo com a utilização e com o meio de conectividade utilizado, estas rotinas são subcategorizadas assim:

#### ☒ **Programação por Processos**

Rotinas escritas em ADVPL podem ser iniciadas como processos individuais (sem interface), no Protheus Server através de duas maneiras: Iniciadas por outra rotina ADVPL por meio da chamada de funções como StartJob() ou CallProc() ou iniciadas automaticamente, na inicialização do Protheus Server (quando propriamente configurado).

#### ☒ **Programação de RPC**

Através de uma biblioteca de funções disponível no Protheus (uma API de comunicação), podem-se executar rotinas escritas em ADVPL diretamente no Protheus Server, por aplicações externas escritas em outras linguagens. Isto é o que se chama de *RPC* (de *Remote Procedure Call*, ou *Chamada de Procedimentos Remota*).

O servidor Protheus também pode executar rotinas em ADVPL, em outros servidores Protheus, através de conexão TCP/IP direta, utilizando o conceito de *RPC*. Do mesmo modo, aplicações externas podem requisitar a execução de rotinas escritas em ADVPL, pela conexão TCP/IP direta.

#### **Programação Web**

O Protheus Server pode também ser executado como um servidor Web, respondendo a requisições HTTP. No momento destas requisições, pode executar rotinas escritas em ADVPL como processos individuais, enviando o resultado das funções como retorno das requisições para o cliente HTTP (como por exemplo, um Browser de Internet). Qualquer rotina escrita em ADVPL, que não contenha comandos de interface, pode ser executada através de requisições HTTP. O Protheus permite a compilação de arquivos

HTML, contendo código ADVPL embutido. São os chamados arquivos ADVPL ASP, para a criação de páginas dinâmicas.

☒ **Programação TelNet**

TelNet é parte da gama de protocolos TCP/IP que permite a conexão a um computador remoto, através de uma aplicação cliente deste protocolo. O PROTHEUS Server pode emular um terminal pela execução de rotinas escritas em ADVPL. Ou seja, pode-se escrever rotinas ADVPL cuja interface final será um terminal TelNet ou um coletor de dados móvel.