

***Programação***  
***ADVPL 1***

## Conteúdo

|   |           |
|---|-----------|
| <b>Capítulo 01 – ADVPL .....</b>                  | <b>4</b>  |
| Interface com o Usuário.....                      | 4         |
| Processos .....                                   | 4         |
| <b>Capítulo 02 – Programação .....</b>            | <b>5</b>  |
| Linhas de Programa .....                          | 5         |
| Tamanho de Linha .....                            | 6         |
| <b>Capítulo 03 – Estruturação.....</b>            | <b>7</b>  |
| <b>Capítulo 04 – Documentação.....</b>            | <b>9</b>  |
| <b>Capítulo 05 – Funcionalidades .....</b>        | <b>11</b> |
| Criação de Funções de Usuário .....               | 11        |
| Criação de Variáveis.....                         | 11        |
| <b>Capítulo 06 – Operadores Básicos.....</b>      | <b>14</b> |
| Matemáticos.....                                  | 14        |
| String.....                                       | 14        |
| Relacionais.....                                  | 14        |
| Lógicos .....                                     | 14        |
| Atribuições.....                                  | 15        |
| Especiais .....                                   | 15        |
| Procedências .....                                | 15        |
| Macro Substituição.....                           | 16        |
| <b>Capítulo 07 – Development Studio .....</b>     | <b>18</b> |
| Configuração de Ambiente .....                    | 19        |
| Criação de Projetos .....                         | 20        |
| Inserção de Programas no Projeto .....            | 21        |
| Compilando Programas.....                         | 22        |
| Executando o Programa.....                        | 24        |
| Depuração do Programa (DEBUG) .....               | 25        |
| <b>Capítulo 08 – Funções Pré-Existentes .....</b> | <b>28</b> |
| Manipulação de Variáveis Tipo Caracter .....      | 28        |
| Manipulação de Variáveis Tipo Númerica.....       | 30        |
| Manipulação de Variáveis Tipo Data .....          | 31        |
| Janelas de Aviso .....                            | 34        |
| <b>Capítulo 09 - Utilização de Vetores .....</b>  | <b>37</b> |
| Funções para manipulação de vetores .....         | 40        |
| Cuidados com Vetores .....                        | 43        |
| Matrizes como Estruturas.....                     | 43        |
| <b>Capítulo 10 – Controle de Fluxo .....</b>      | <b>44</b> |
| Comandos de Repetição.....                        | 44        |
| FOR...NEXT .....                                  | 44        |
| WHILE...ENDDO.....                                | 45        |
| Problemas Comuns .....                            | 46        |
| Comandos de Decisão .....                         | 46        |
| IF...ENDIF .....                                  | 46        |
| DO CASE...ENDCASE .....                           | 47        |

# Programação Advpl 1

---

|  |            |
|--|------------|
| <b>Capítulo 11 – Blocos de Códigos .....</b>           | <b>49</b>  |
| Definição .....  | 49         |
| Funções de Bloco .....                                 | 50         |
| <b>Capítulo 12 – Funções Diversas .....</b>            | <b>52</b>  |
| Funções de Ambientes.....                              | 52         |
| Funções de Servidor.....                               | 53         |
| Funções de Conexão com o Servidor .....                | 54         |
| Funções de Comunicação Server x Client .....           | 55         |
| <b>Capítulo 13 – Customização .....</b>                | <b>56</b>  |
| Parâmetros (SX6) .....                                 | 56         |
| Tabelas Genéricas (SX5).....                           | 60         |
| Função para acesso às Tabelas Genéricas .....          | 63         |
| Perguntas (SX1) .....                                  | 64         |
| Expressões ADVPL .....                                 | 70         |
| Pontos de Entrada.....                                 | 71         |
| Dicionário de Dados Ativos (SX2 e SX3) .....           | 72         |
| Criação de Indices (SIX) .....                         | 82         |
| Criação de Pastas (SXA) .....                          | 84         |
| Gatilhos (SX7).....                                    | 88         |
| Consulta Padrão (SXB) .....                            | 92         |
| <b>Capítulo 14 – Menus.....</b>                        | <b>96</b>  |
| Incluindo Item no Menu.....                            | 96         |
| <b>Capítulo 15 – Conceito de filiais .....</b>         | <b>101</b> |
| Arquivos Compartilhados .....                          | 101        |
| Arquivos Exclusivos.....                               | 101        |
| <b>Capítulo 16 – Tratamento de Base de Dados .....</b> | <b>103</b> |
| Criando Arquivos .....                                 | 103        |
| Criando Indices Temporários .....                      | 103        |
| Funções de Base de Dados.....                          | 104        |
| Posicionamento de Registros .....                      | 106        |
| <b>Capítulo 17 – Programando.....</b>                  | <b>108</b> |
| Criando um Programa Simples.....                       | 108        |
| <b>Capítulo 18 – Tela de Padrão Microsiga.....</b>     | <b>111</b> |
| AxCadastro().....                                      | 111        |
| Pergunte() .....                                       | 112        |
| PutSx1().....  | 113        |
| mBrowse().....   | 113        |
| Modelo 2.....  | 115        |
| Modelo 3.....  | 116        |
| <b>Capítulo 19 – MSEXECAUTO .....</b>                  | <b>118</b> |
| Vantagens .....  | 118        |
| Exemplo de Automação de Rotinas.....                   | 119        |

## Capítulo 01 – ADVPL

ADVPL (Advanced Protheus Language) surgiu com a tecnologia Protheus em 1994, derivada do Clipper e bibliotecas FiveWin, incorpora o padrão xBase para a manutenção de todo o código já existente do sistema de ERP Siga Advanced, possui comandos e funções, operadores, estruturas de controle de fluxo e palavras reservadas, contando também com funções e comandos criados pela própria Microsiga que a torna uma linguagem completa para a criação de aplicações ERP. Também é uma linguagem orientada a objetos e eventos, permitindo ao programador desenvolver aplicações visuais e criar suas próprias classes de objetos.

A Microsiga criou seu próprio ambiente de desenvolvimento (IDE) que edita programas, compila, interpreta e depura as funções de usuário que ficam armazenadas nas unidades de inteligência básicas, chamados *APO's* (*Advanced Protheus Objects*), que ficam guardados em um repositório carregado pelo Protheus no momento de sua execução. Toda função criada em AdvPl pode ser executada em qualquer ponto do ambiente Protheus.

Os programas em AdvPl são subdivididos nas seguintes categorias:

### Interface com o Usuário

São funções que realizem algum tipo de interface remota utilizando o protocolo de comunicação do Protheus. Podem ser criadas rotinas para a customização desde processos adicionais até mesmo relatórios. A grande vantagem é aproveitar todo o ambiente montado pelos módulos, utilizando telas e funções padronizadas. Com o ADVPL é possível criar uma aplicação do começo ao fim.

### Processos

São funções sem interface com o usuário, são rotinas como processos internos ou Jobs. Essas rotinas são executadas de forma transparente ao usuário.

Existem rotinas que são responsáveis por iniciar os processos executados através dos meios disponíveis de integração e conectividade no Protheus, subdivididas em:

- **Programação de RPC**

Executada através de uma biblioteca de funções disponíveis diretamente pelo Server do Protheus ou através de aplicações externas escritas em outras linguagens. Essa execução pode ser feita em outros servidores Protheus através de conexão TCP.

- **Programação Web**

São requisições http feitas em ADPVL que são executadas como um servidor Web, como processos individuais, enviando ou recebendo resultado das funções. Lembrando que as funções não devem ter comandos de interface com o usuário, nesse caso utiliza-se arquivos HTML contendo código ADVPL, conhecidos como ADVPL ASP, para a criação de páginas dinâmicas.

- **Programação TelNet**

O Protheus Server pode emular um terminal TelNet, através da execução de rotinas escritas em ADVPL, cuja interface final será um terminal TelNet ou um coletor de dados móvel.

## Capítulo 02 – Programação

Um programa de computador nada mais é do que um algoritmo escrito numa linguagem de computador com o objetivo de executar determinada tarefa. Esses comandos são gravados em um arquivo texto que é transformado em uma linguagem executável por um computador através da compilação. A compilação substitui os comandos que estão numa linguagem que nós entendemos por linguagem de máquina. No caso do AdvPl, esse código compilado será executado pelo Protheus Server a partir do Repositório.

Para programar em qualquer linguagem devem ser seguidas regras da linguagem, a qual chamamos de sintaxe da linguagem, se não obedecida essa sintaxe o programa causará erros que podem ser de compilação ou de execução.

**Compilação:** não permite nem que o código seja compilado, são comandos especificados incorretamente, operadores utilizados de forma errada, sintaxe incorreta, dentre outros.

**Execução:** são os que ocorrem no momento da execução. Podem ocorrer por muitas razões, dentre elas, funções não existentes, ou variáveis não criadas ou não inicializadas, etc.

### Linhas de Programa

As linhas existentes dentro de um arquivo texto de código de programa podem ser:

**Linhas de comando:** possuem os comandos ou instruções que serão executadas.

**Linhas de comentário:** possuem um texto qualquer, que não são executadas é muito utilizado para documentação e facilitar o entendimento do programa. Podemos comentar informações no programa utilizando os seguintes caracteres:

|    |  |
|----|--|
| && | && Inserir aqui seu comentário e/ou explicação |
| // | // Inserir aqui seu comentário e/ou explicação |
| *  | * Inserir aqui seu comentário e/ou explicação  |

Podemos documentar blocos de texto inicializando com /\* e finalizando com \*/

```
/*
Programa para recalcular custo médio
Autor:
Data:
*/
```

**Linhas Mistas:** possuem comandos ou instruções juntamente com comentários

Local nSoma := 0 // Digite aqui o comentário referente a linha de comando

## Tamanho de Linha

**Linha física:** delimitada pela quantidade de caracteres que pode ser digitado no editor de textos utilizado. Cada linha física é separada por um <Enter>.

**Linha lógica:** é aquela considerada para a compilação como uma única linha de comando. Quando queremos que mais de uma linha física componha a mesma linha de comando utilizamos o ";" (ponto e vírgula), exemplo:

```
If !Empty(cNome) .And. !Empty(cEnd) .And. ; <enter>
!Empty(cTel) .And. !Empty(cFax) .And. ; <enter>
!Empty(cEmail)
    GravaDados(cNome,cEnd,cTel,cFax,cEmail)
Endif
```

## Capítulo 03 – Estruturação

A linguagem ADVPL não tem padrões rígidos de estrutura do programa, mas normalmente utilizamos a seguinte estruturação:

**Identificação:** Área reservada para documentação do programa, como finalidade, data, autor, etc.

```
// Programa para identificar números pares e ímpares  
// Autor:  
// Data:
```

**Inicialização:** Área reservada a declarações de variáveis, abertura de arquivos. Apesar de em ADVPL ser permitido o uso de variáveis não declaradas anteriormente, é importante declará-las aqui para tornar o programa mais legível.

```
User Function fSeparaNum  
Local nBI  
Local cBIImpares := "" // Variável será utilizada para armazenar os números ímpares  
Local cBIPares := "" // Variável será utilizada para armazenar os números pares
```

**Corpo do programa:** Área reservada para a lógica do programa.

```
For nBI:=1 to 12  
If mod(nBI,2)== 0  
    cBIPares += Alltrim(str(nBI)) + "  
Else  
    cBIImpares += Alltrim(str(nBI)) + "  
Endif  
Next
```

**Encerramento:** Área reservada ao fechamento dos arquivos abertos, mostrar o resultado do processamento e encerramento do programa.

```
Msginfo("Pares "+cBIPares+" Ímpares "+cBIImpares) // Exibe em tela o resultado  
Return // Termina o programa
```

Veja como ficou nosso exemplo obedecendo as características da estrutura do código escrito em ADVPL:

# Programação Advpl 1

The screenshot shows a Development Studio window titled "Development Studio [Executando...]-[FSEPNUM.PRW]". The menu bar includes Arquivo, Editar, Inserir, Visualizar, Projetos, Executar, Ferramentas, Autorização, Janela, and Ajuda. The toolbar contains various icons for file operations like Open, Save, Print, and Run. The left pane shows a code editor with the following AdvPL code:

```
1 #INCLUDE "PROTHEUS.CH"
2 /*
3 Programa para identificar números pares e ímpares
4 Autor:
5 Data:
6 */
7
8 User Function fSepNum
9
10 Local nBI
11 Local cBIImpares := "" // Variável será utilizada para armazenar os números ímpares
12 Local cBIPares := "" // Variável será utilizada para armazenar os números pares
13
14 For nBI:=1 to 12
15   If mod(nBI,2)== 0
16     cBIPares += Alltrim(str(nBI)) + " "
17   Else
18     cBIImpares += Alltrim(str(nBI)) + " "
19   Endif
20 Next
21
22 Msginfo("Pares "+cBIPares+" - Ímpares "+cBIImpares) // Exibe em tela o resultado
23 Return // Termina o programa
```

The right pane displays a message box titled "TOTVS" with an information icon. The message reads "Pares 2 4 6 8 10 12 - Ímpares 1 3 5 7 9 11" and has an "Ok" button.

## Capítulo 04 – Documentação

Toda customização deve ser bem documentada, com informações relevantes e conceituais. O cabeçalho das funções deve conter o nome do autor, data de criação, uma descrição sumária de sua funcionalidade, a sintaxe e a descrição dos argumentos de entrada e saída. Pode também ser colocada a localização no menu, agendamento, etc.

### • Documentação em Funções:

/\*

|   |  |            |          |      |          |
|---|--|------------|----------|------|----------|
| Programa  | ADV0070A   | Autor      |          | Data | 23/07/04 |
| Descrição   | Verifica se campo do E2 pode ser alterado ou não |            |          |      |          |
| Modulo  | SIGAFIN  |            |          |      |          |
| Observação  | Especifico para Biale                            |            |          |      |          |
| Alterações e solicitações   |  |            |          |      |          |
| Descriptivo   |  | Solicitado | Atendido | Data |          |
| Incluida verificação de chamada por<br>MsExecAuto ISInCallStack('msexecauto') |  | Gerente    |          |      |          |

• \*/

User Function ADV0070A(nCampo)

Comandos...

Return

### • Documentação de solicitação de Customizações:

Também é necessária para uma melhor visualização da documentação e de todas as solicitações efetuadas pelos usuários ou gerentes com as devidas assinaturas.

Veja na página seguinte o exemplo de uma documentação de “solicitação de customização”.

## Dados da Requisição

|                            |   |
|----------------------------|---|
| Data Solicitação           | <data da solicitação>   |
| Área                       | <nome da área solicitante/centro de custo>                      |
| Responsável                | <nome do solicitante responsável>                               |
| Analista de Negócio        | < nome do analista de negócio >                                 |
| Volume de esforço Previsto | <quantidade de horas necessárias para o desenvolvimento/testes> |
| Data de entrega prevista   | <data de entrega prevista (uso da área de TI)>                  |

# Programação AdvPl 1

## Objetivo

Esta rotina tem por objetivo ... (descrever aqui o objetivo macro que a rotina deverá atender. Neste espaço não devem ser descritas as regras de negócio, pois há uma seção mais adiante pra atender a esta necessidade. Descrever o objetivo em uma linguagem que o usuário entenda).

## Programas envolvidos (análise inicial)

---

---

---

---

---

---

## Comentários/Observações do Responsável pela Customização

---

---

---

---

---

---

## Aprovações

| Nome | Assinatura | Data |
|------|------------|------|
|      |            |      |
|      |            |      |
|      |            |      |
|      |            |      |

## Capítulo 05 – Funcionalidades

### Criação de Funções de Usuário

Em ADVPL utilizamos funções da seguinte forma:

|                 |  |
|-----------------|--|
| Function        | Limitada a programas padrões, não permitindo ser compilada por clientes. |
| User Function   | Funções de usuário, podendo ser utilizadas para fazer customizações.     |
| Static Function | Funções chamadas pelas User Functions e pelas Functions.                 |

Especificamente Functions e User Functions podem ser chamadas de qualquer ponto do Sistema, se estiverem compiladas no repositório.

#### Características de funções definidas como “User Function”

- Podem ser executadas por todas as demais funções compiladas no repositório;
- Importante que o nome da função não ultrapasse 8 caracteres;
- Não pode existir mais de 1 função com o mesmo nome compilada no repositório;
- Para sua execução “sempre” deverá ser informado a sequencia de caracteres “U\_” antes de seu nome, exemplo U\_Calcula(), essa foi criada com o comando “User Function Calcula()”.

#### Características de funções definidas como “Static Function”

- Podem ser executadas somente dentro do arquivo fonte (PRW ou PRX) onde a mesma foi criada;
- Importante que o nome da função não ultrapasse 10 caracteres;
- Pode existir mais de 1 função com o mesmo nome compilada no repositório (mas não dentro do mesmo fonte (PRW ou PRX));
- Para sua execução deverá ser informado somente o nome da função, sem o uso da sequencia de caracteres “U\_” antes de seu nome, exemplo Calcula(), essa foi criada com o comando “Static Function Calcula()”.

### Criação de Variáveis

As variáveis devem ser declaradas no início da função que for utilizá-la.

Características importantes com relação a nomenclatura de variáveis:

1. Não devem iniciar com números ou caracteres especiais, para o último caso é permitido uso do “\_” (underline);
2. Não podem conter espaços;
3. Importante não ultrapassar 10 caracteres, pois somente esse primeiros serão reconhecidos.

#### • Identificadores de escopo

As variáveis declaradas em um programa ou função, são visíveis de acordo com o escopo onde são definidas. Como também do escopo depende o tempo de existência das variáveis.

# Programação AdvPl 1

---

A definição do escopo de uma variável é efetuada no momento de sua declaração.

**Public:** Visível a partir do momento que é criada até o término da aplicação, não deve ser utilizada para qualquer coisa pois ocupa espaço de memória.

**Private:** Visível na função que a criou e nas demais chamadas por esta, até que a função que a criou seja finalizada, e são criadas automaticamente quando não declaradas.

**Local:** Visíveis somente na função que a criou, variáveis locais ocupam pouco espaço de memória. E são criadas automaticamente quando provierem de parametrização de função.

**Static:** Visíveis em todas as funções (user function ou static function) criadas no fonte (PRW ou PRX) onde a mesma foi criada.

## • Identificadores de tipos

O tipo de variável identifica sua utilização na função. Toda variável deve estar tipada durante sua criação.

É convencional utilizarmos letras iniciais em variáveis para que possamos saber se o seu tipo é numérico, caractere, data ou lógico.

As variáveis têm os seguintes tipos:

| Tipo     | Exemplo | Descriptivo                |
|----------|---------|----------------------------|
| Caracter | cVar    | Aceita números e letras    |
| Numérica | nVar    | Aceita apenas números      |
| Data     | dVar    | Aceita apenas Datas        |
| Lógica   | LVar    | Aceita Verdadeira ou Falsa |
| Array    | aVar    | Aceita Vetores             |
| Objeto   | oVar    | Usada em objetos           |

## Variáveis e nomes de campos

Muitas vezes uma variável pode ter o mesmo nome que um campo de um arquivo ou tabela aberta no momento. Neste caso, o AdvPl privilegiará o campo. Assim uma referência a um nome que identifique tanto uma variável como um campo, resultará no conteúdo do campo.

Para especificar qual deve ser o elemento referenciado, deve-se utilizar o operador de identificação de apelido (->) e um dos dois identificadores de referência, MEMVAR ou FIELD.

```
cNomeCli := MEMVAR->A1_NOME ( ou M-> A1_NOME )
cNomeCli := FIELD->A1_NOME
```

O identificador FIELD pode ser substituído pelo apelido de um arquivo ou tabela aberto, para evitar a necessidade de selecionar a área antes de acessar o conteúdo de terminado campo.

```
cNomeCli := SA1->A1_NOME // SA1 é o nome da tabela de clientes
```

# Programação AdvPl 1

## Inicialização de Variáveis

Quando não atribuímos nenhum valor a uma variável no momento de sua declaração, corremos o risco de utilizá-la com valor "NIL" e isso pode causar erros fatais. Por isso, a inicialização de uma variável é de extrema importância.

## Utilização da Função CRIAVAR( )

Esta função cria uma variável, retornando o valor do campo, de acordo com o dicionário de dados. Avalia o inicializador padrão e retorna o conteúdo de acordo com o tipo de dado definido no dicionário.

**Sintaxe:** uRet := CriaVar(cCampo,lIniPad)

|         |   |
|---------|---|
| URet    | tipo de retorno de acordo com o dicionário de dados, considerando inicializador padrão. |
| CCampo  | Nome do campo   |
| LiniPad | Indica se considera (.T.) ou não (.F.) o inicializador padrao (X3_RELACAO)              |

No exemplo a seguir será criada a variável cNomeCli com o mesmo tipo e tamanho do campo A1\_NOME.

```
cNomeCli := CRIAVAR("A1_NOME")
```

A seguir será criada a variável cCodCli com o mesmo tipo e tamanho do campo A1\_COD, nesse exemplo foi informado o parâmetro .T., assim, caso o campo contenha alguma instrução em seu INICIALIZADOR PADRÃO (X3\_RELACAO), esse será executado e a variável será criada com conteúdo de acordo com a instrução contida nesse.

```
cCodCli := CRIAVAR("A1_NOME",.T.)
```

## Capítulo 06 – Operadores Básicos

### Matemáticos

Os operadores utilizados para cálculos matemáticos são:

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

### String

Os operadores utilizados 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)  |

### Relacionais

Os operadores utilizados para operações e avaliações relacionais são:

|               |                                    |
|---------------|------------------------------------|
| <             | Menor que                          |
| >             | Maior que                          |
| =             | Igual                              |
| ==            | Exatamente Igual (para caracteres) |
| <=            | Menor ou Igual                     |
| >=            | Maior ou Igual                     |
| <> ou # ou != | Diferente                          |

### Lógicos

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

|            |     |
|------------|-----|
| .And.      | E   |
| .Or.       | OU  |
| .Not. ou ! | NÃO |

# Programação AdvPI 1

## Atribuições

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

|           |   |
|-----------|---|
| =         | Atribuição Simples                              |
| :=        | Atribuição em Linha                             |
| +=        | 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             |
| %=        | Módulo (resto da divisão) e Atribuição em Linha |
| ++        | Incremento Pós ou Pré-fixado                    |
| --        | Decremento Pós ou Pré-fixado                    |

## Especiais

Além dos operadores comuns, o AdvPI possui alguns operadores especiais. Estas são suas finalidades:

|     |   |
|-----|---|
| ( ) | Utilizados para agrupar elementos em uma expressão priorizando essas condições, ao qual damos o nome de " <b>ordem de precedência</b> " da avaliação da expressão (da mesma forma que as regras matemáticas). Também servem para envolver os argumentos de uma função   |
| [ ] | Elemento de Matriz. Utilizados para especificar um elemento específico de uma matriz. Por exemplo, aArray[1,5], refere-se ao elemento da matriz aArray na linha 1, coluna 5.  |
| {}  | Definição de Matriz, Constante ou Bloco de Código. Por exemplo:<br>aArray := {1,2,3,4,5} cria uma matriz chamada aArray com cinco elementos.  |
| ->  | Identificador de Apelido (Alias). identifica um campo de um arquivo diferenciando-o de uma variável. Exemplo:<br>SA1->A1_NOME, o campo A1_NOME da tabela SA1, mesmo que haja uma variável com o nome A1_NOME, a partir desse Alias o campo da tabela que será acessado. |
| &   | Identifica uma avaliação de expressão através macrossubstituição.   |
| @   | Passagem de parâmetro por referência. 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.   |
|     | Passagem de parâmetro por valor. 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.   |

## Procedências

Caso as expressões complexas sejam de mesmo nível a avaliação será feita da esquerda para direita, exceto as matemáticas que obedecem o seguinte:

1. Exponenciação
2. Multiplicação e Divisão
3. Adição e Subtração

## Exemplo

Local nVar := 15+12/3+7\*3-2^3

Calcula-se a exponenciação:  $2^3 = 8$

Calcula-se divisão:  $12/3 = 4$

Calcula-se a multiplicação:  $7*3 = 21$

Efetua a soma e subtração:  $15 + 4 + 21 - 8$

O resultado desta expressão é 32

Para alterarmos a precedência utilizamos os parênteses, o grupo mais a esquerda será avaliado primeiro e assim por diante, como também o mais interno para o mais externo, caso haja parênteses dentro de parênteses.

Local nVar := := (15+12)/(3+7)\*3-2^3

Calcula-se a exponenciação:  $2^3 = 8$

Calcula-se o primeiro grupo de parênteses:  $15+12 = 27$

Calcula-se o segundo grupo:  $3+7=10$

Calcula-se a divisão:  $27/10 = 2.7$

Calcula-se a multiplicação:  $2.7 * 3 = 8.1$

Efetua a soma e subtração:  $8.1-8$

O resultado desta expressão é 0.1

## Macro Substituição

O operador de macro substituição, simbolizado pelo & ( "E" comercial ), é utilizado para a avaliação de expressões em tempo de execução, ou seja, o resultado será sobre o conteúdo do resultado da variável, exemplo:

```
cExpressao := "5 * 5" // Observe que cExpressao é uma variável do tipo caractere  
cVar := cExpressao // O conteúdo da variável cVar será o mesmo da cExpressao "5 * 5"  
nVar := &cExpressao // Já o conteúdo da variável nVar será o resultado do calculo 5 * 5
```

No exemplo acima foi criado a variável cExpressao com conteúdo caractere "5 \* 5".

Na sequencia criamos a variável cVar, essa conterá o mesmo conteúdo da primeira, isto é, "5 \* 5" do tipo caractere.

Em seguida foi criada a variável nVar, repare que seu nome inicia com "n", pois será do tipo numérica, isso porque, para sua criação utilizamos o simbolo "&" antes da variável cExpressao, assim o conteúdo da nova varável não sera a sequencia de caracteres "5 \* 5", mas sim o resultado do calculo 5 \* 5, resultando 25.

# Programação Advpl 1

## Exercícios

De acordo com a instrução fazendo um teste de mesa, informe ao final qual o tipo e conteúdo de cada variável criada.

```
01 User Function fProcVar()
02 Local cVal, nTotGer, IVarl, cTem, ICont, ITudo
03 Local cNom  := ""
04 Local nVal  := 5
05 Private nVal1:= 7 + 6, IVal, IQuas, nValor
06 Public IVar := nVal == 5
07 nVal1 += 15
08 nValor := (((nVal1- nVal)/2) * 3^2 + 1 )* -1
09 nTempo := "nValor * 3"
10 nTotGer:= &nTempo
11 nTotGer++
12 cTem  := "São Paulo"
13 cTem  += ", 15 de Dezembro"
14 ICont := "Dez" $ cTem
15 ITudo := cTem = "São"
16 IQuas := cTem == "São Paulo"
17 IVal  := nVal > 5
18 IValc := (nVal1 - 8) <= 13
19
20 fMudaFun(@nValor, ITudo, IQuas, IVal)
21 msginfo(nValor)
22 Return
23
24 Static Function fMudaFun(_val, _ITudo, _IQuas, _IVal)
25   msginfo(nVal1)
26 Return
27
28 User Function fContinua()
29   Msginfo(IVar)
30 Return
```

De acordo com o exercício anterior responda Verdadeiro ou Falso, nas afirmações a seguir e justifique:

a. (  ) \_ITudo na linha 25 será nula.

b. (  ) nTempo é do tipo caracter.

c. (  ) Icont tem o conteúdo Verdadeiro.

d. (  ) IVar não esta visível na linha 29

e. (  ) ITudo na linha 25 tem seu identificador Local

f. (  ) cNom na linha 18 é nula

G. (  ) nValor pode ser acessada na linha 25

## Capítulo 07 – Development Studio

É o IDE, ambiente de desenvolvimento integrado do Advanced Protheus. É através dele que é feito o acesso ao repositório, incluindo e excluindo funções. O IDE é utilizado tanto pelos clientes quanto pelos programadores da Microsiga, com ele podemos compilar e depurar os programas. O IDE possui todos os recursos disponíveis nas melhores ferramentas de desenvolvimento do mercado.

Podemos utilizar o IDE sem conexão ao Server, a conexão somente é feita durante atualização ou consulta de um repositório ou durante o processo de depuração.

### Dica

Na instalação padrão do Microsiga Protheus 11 encontramos o aplicativo DevStudio.exe (Development Studio) na pasta: C:\TOTVS 11\Microsiga\Protheus\bin\smartclient.

Os passos para o desenvolvimento de programas em AdvPl utilizando o IDE são:

### **Configuração de Ambiente**

Podemos configurar os ambientes que serão atualizados na compilação e depuração de programas

### **Criação do Projeto**

Criamos projetos como forma de organizar os programas criados de modo que qualquer outro analista possa entender o roteiro das customizações;

### **Criação do programa**

Através de edição podemos criar programas;

### **Compilação**

Compilamos os programas ou todo o projeto para enviarmos ao repositório o que foi mudado e para que o Protheus saiba o que queremos que faça;

### **Depuração**

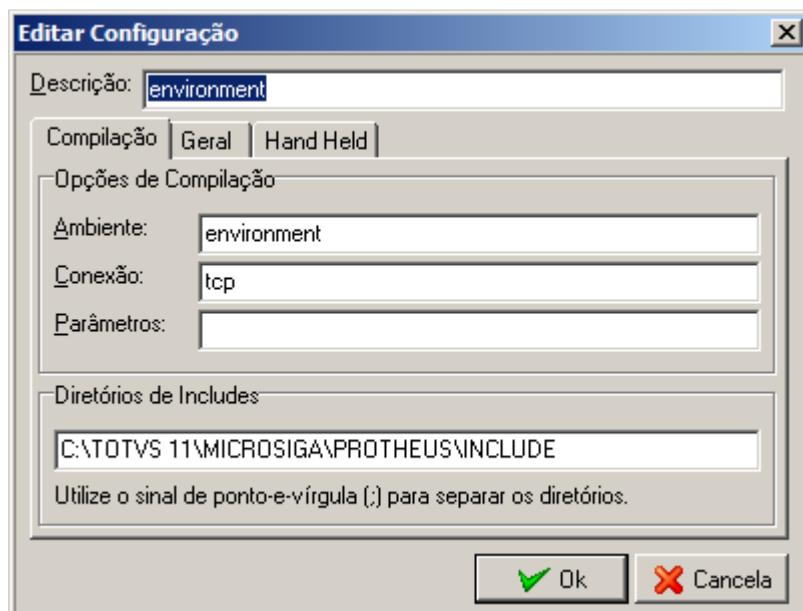
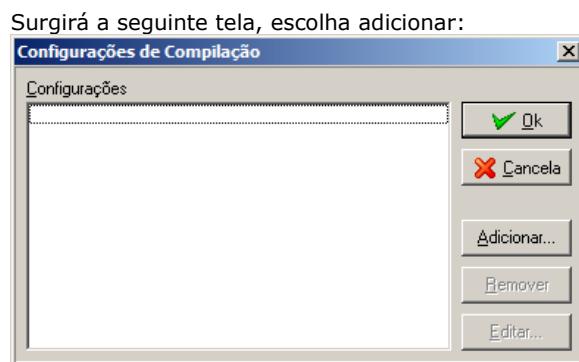
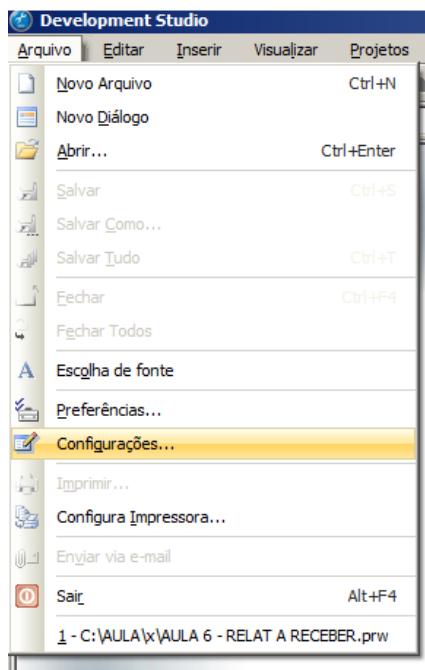
Execução passo a passo das rotinas, permitindo que o analista veja o andamento do programa e o conteúdo das variáveis, bem como as tabelas abertas;

# Programação AdvPl 1

## Configuração de Ambiente

Antes de começarmos, precisamos primeiro configurar o ambiente em que queremos trabalhar, para isso siga os procedimentos:

No aplicativo IDE escolha as opções Arquivo → Configurações.



**Descrição:** Nome sugestivo para o ambiente, geralmente colocamos o mesmo nome do ambiente no Server

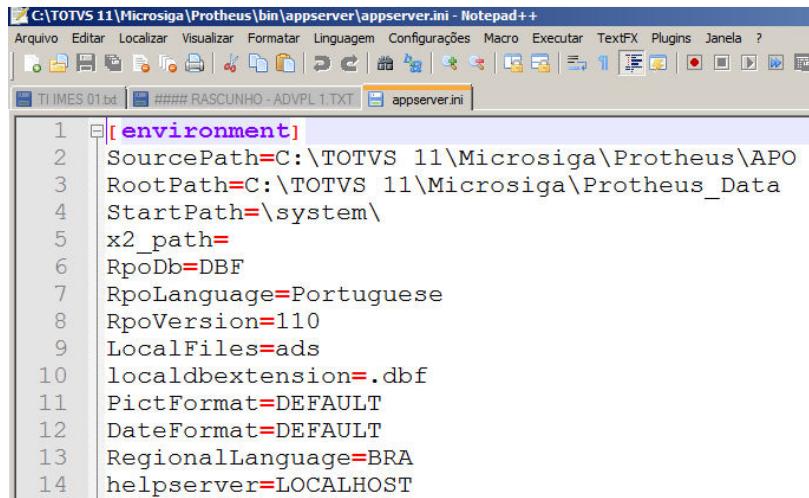
**Ambiente:** Nome do ambiente no server.

**Conexão:** nome da conexão utilizada, geralmente é TCP.

**Parâmetros:** Parâmetros utilizados (-m, -q, -a, -e) entre outros.

**Diretórios de Includes:** Caminho onde se encontram os includes, (\*.ch), separados por ponto e vírgula (;)

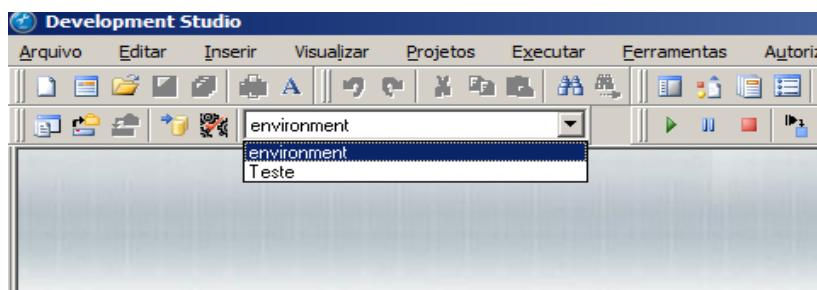
# Programação AdvPL 1



```
1 [environment]
2 SourcePath=C:\TOTVS 11\Microsiga\Protheus\bin\appserver\appserver.ini - Notepad++
3 RootPath=C:\TOTVS 11\Microsiga\Protheus_Data
4 StartPath=\system\
5 x2_path=
6 RpoDb=DBF
7 RpoLanguage=Portuguese
8 RpoVersion=110
9 LocalFiles=ads
10 localdbextension=.dbf
11 PictFormat=DEFAULT
12 DateFormat=DEFAULT
13 RegionalLanguage=BRA
14 helpserver=LOCALHOST
```

Repare que o ambiente informado no campo “Ambiente” na tela acima é o mesmo configurado no arquivo appserver.ini

Caso você tenha outros ambiente nesse arquivo, pode criar configurações específicas para cada um deles, exemplo caso tenha um ambiente de TESTES.

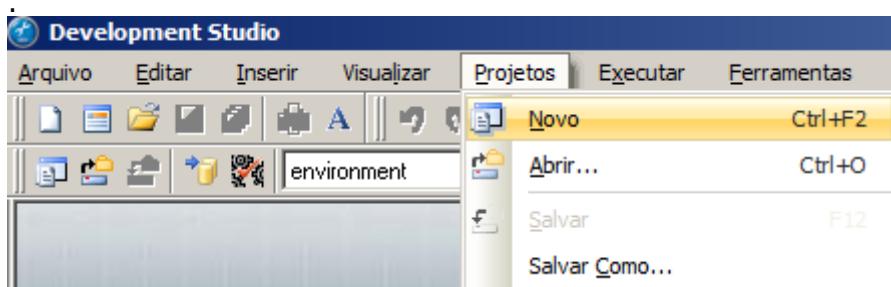


Os ambientes configurados estarão disponíveis na lista de “ambientes” conforme figura ao lado, apartir desse momento o desenvolvedor pode escolher o ambiente que deseja compilar seus programas.

## Criação de Projetos

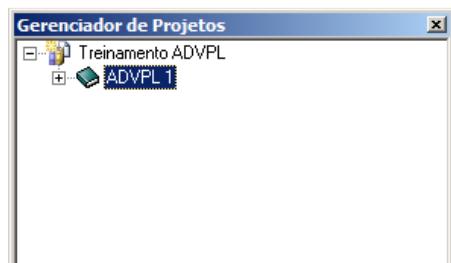
Para desenvolvimento das customizações é necessário a criação de um projeto para organização dos programas, para criação do mesmo siga os procedimentos:

No aplicativo IDE escolha as opções “Projeto → Novo”, na tela que será apresentada digite o nome do projeto.



Em seguida clique nas opções: “Projeto → Salvar como”.

Informe o nome do arquivo que conterá as configurações do projeto recém criado e a pasta onde será salvo. O arquivo terá como extensão “.PRJ”

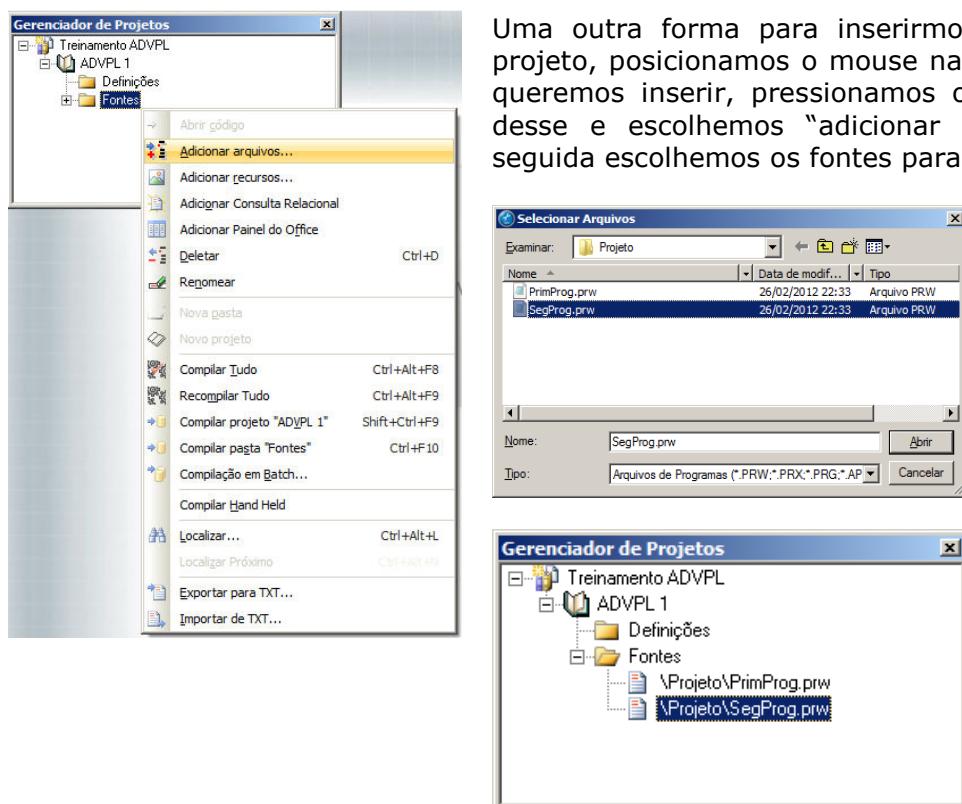
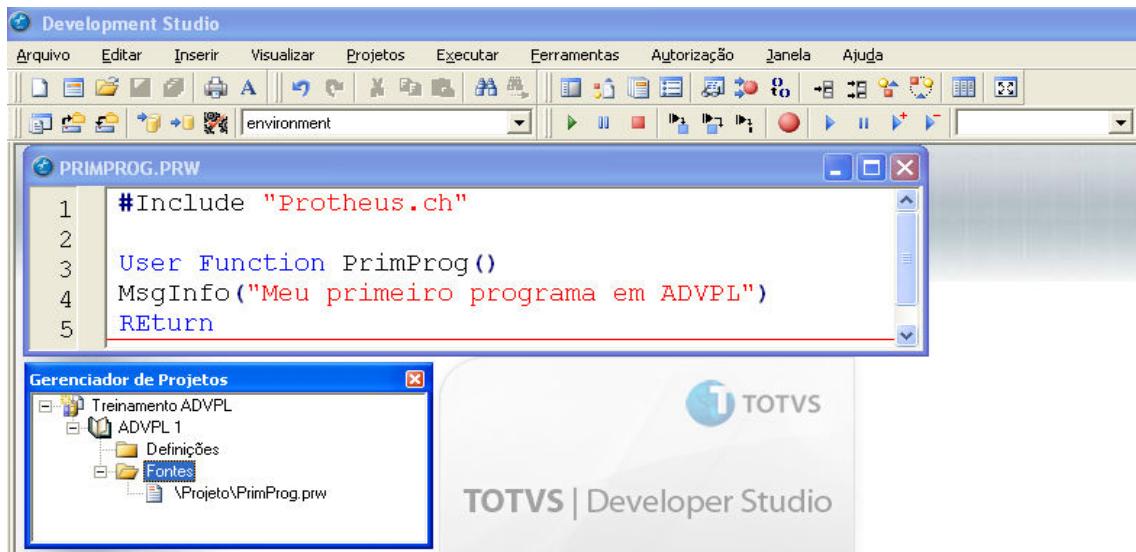


# Programação AdvPL 1

## Inserção de Programas no Projeto

Para inserção de programas no projeto, siga os procedimentos:

Após digitar e salvar o novo programa, clique com botão direito do mouse sobre o mesmo e arraste-o sobre a janela de apresentação do projeto, solte-o na pasta escolhida, no exemplo abaixo foi digitado o programa PrimProg.prw e inserido na pasta Fontes no projeto.



### Dica

Importante que qualquer alteração feira no projeto, isto é, a inclusão ou exclusão de fontes do projeto, você o salve, para isso clique na opção "Projeto → Salvar" ou utilize e tecla de atalho F12.

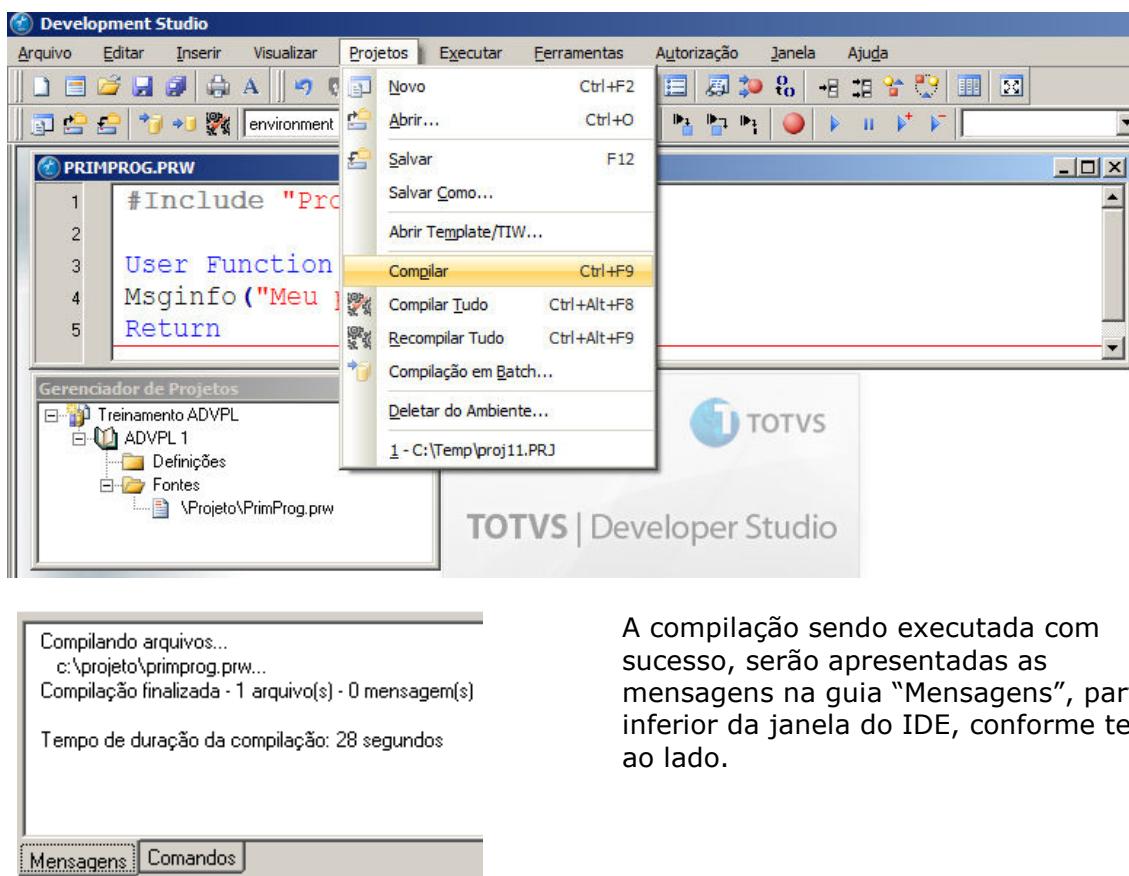
# Programação Advpl 1

## Compilando Programas

Agora que já temos o ambiente, projeto e o programa precisaram enviá-lo para o Protheus, para isso precisamos compilar o programa, e assim esse será inserido no repositório e pode ser interpretado (executado) pelo Protheus.

Para realizar a compilação do seu programa, fazendo assim com que o mesmo fique disponível no repositório de objetos, siga os procedimentos:

Com o programa que deseja compilar aberto, clique nas opções “Projetos → Compilar”, conforme tela abaixo. (Observe que você pode usar a tecla de atalho Ctrl+F9)



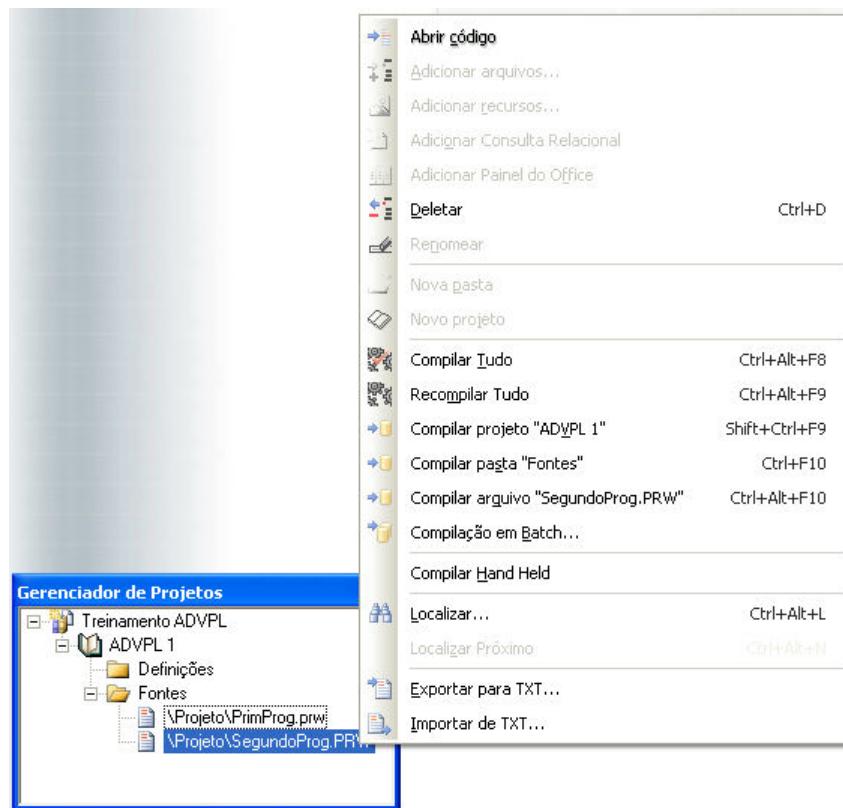
A compilação sendo executada com sucesso, serão apresentadas as mensagens na guia “Mensagens”, parte inferior da janela do IDE, conforme tela ao lado.

Nessa mesma tela serão apresentadas mensagens de erro, caso o compilador identifique em seu programa.

# Programação Advpl 1

Outra forma de você compilar seus programas é clicando com botão direito do mouse sobre o gerenciador de projetos, assim terá disponível as opções:

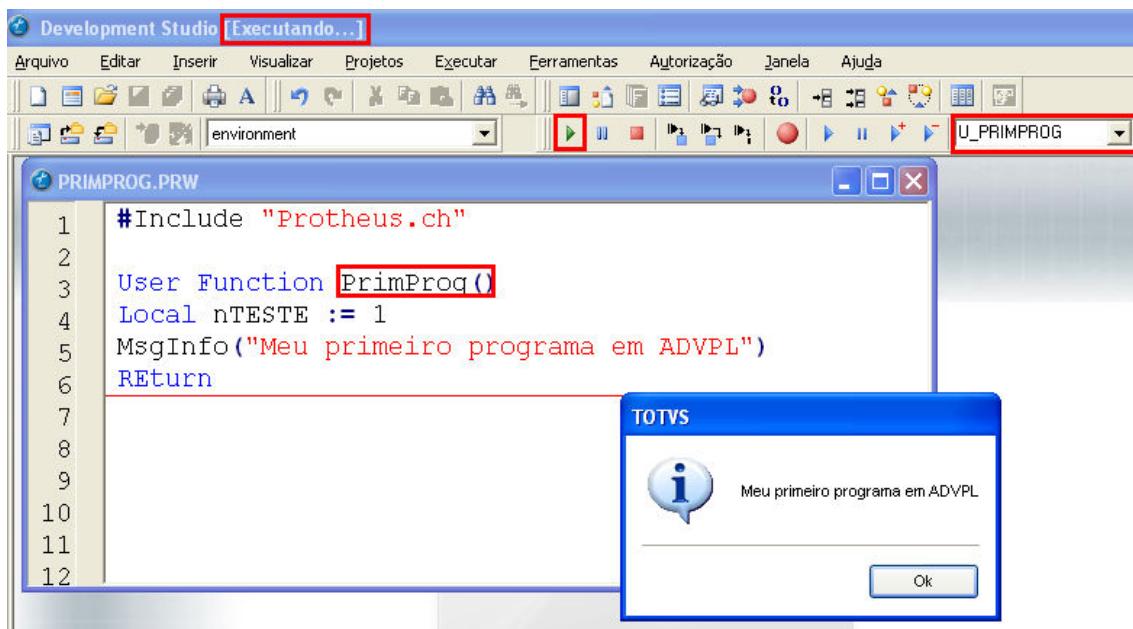
- Compilar tudo - todo o projeto, somente os que houve mudança desde a ultima compilação;
- Recompilar tudo – todo o projeto, sem exceção;
- Somente um determinado projeto;
- Somente os programas dentro de uma pasta;
- Apenas um programa.



## Executando o Programa

A partir desse momento sua rotina estará disponível no repositório de objetos (RPO), e poderá ser executada, para isso siga os procedimentos:

No campo  tipo combo localizado na barra de ferramentas digite o nome da função a ser executada, essa prescedida de "U\_", em nosso caso U\_PRIMPROG, em seguida clique no botão 'executar'  ou utilize a tecla de atalho F5.



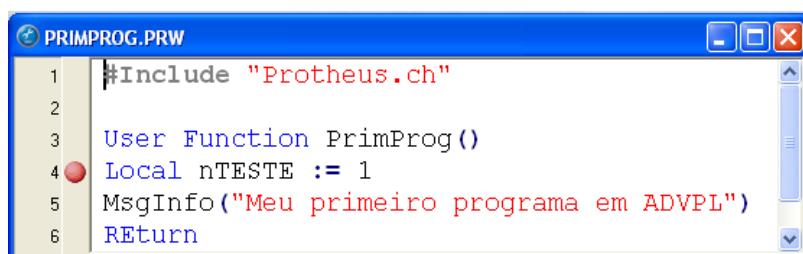
## Depuração do Programa (DEBUG)

Depuração de programas (em inglês: *debugging, debug*) é o processo de encontrar defeitos em aplicativos de software. Com essa ferramenta podemos executar nossa função “linha a linha”, analisando conteúdos de variáveis, resultados de cálculos, inclusive interagindo de certa forma com a execução de nosso programa pois podemos alterar o conteúdo das variáveis em tempo de execução.

### Marcando pontos de parada

Para podermos depurar um programa em ADVPL, devemos primeiramente marcar os pontos de parada (em inglês: break point), onde queremos parar sua execução.

Temos duas formas de incluir esses pontos de parada, primeiro você pode clicar na coluna que antecede a linha escolhida, incluindo assim um círculo vermelho  conforme tela:



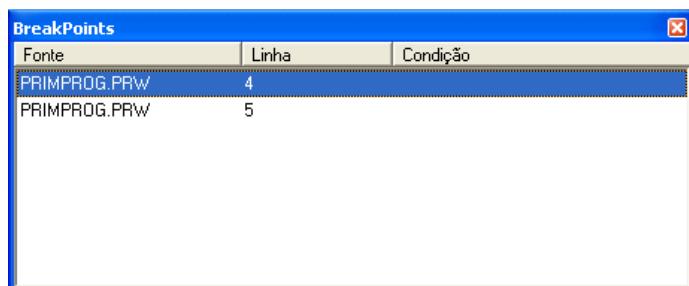
A screenshot of a Windows-style application window titled "PRIMPROG.PRW". The code editor displays the following pseudocode:

```
1 #Include "Protheus.ch"
2
3 User Function PrimProg()
4 • Local nTESTE := 1
5 MsgInfo("Meu primeiro programa em ADVPL")
6 Return
```

The fourth line, "Local nTESTE := 1", has a red circular breakpoint marker to its left. To the right of the code editor, there is explanatory text:

Você pode inserir quantos pontos de paradas foram necessários, bastando para isso escolher as linhas e marca-las.

Outra forma de incluir pontos de parada é posicionar o cursor de digitação na linha escolhida e teclar F9.



Utilizando o botão  ou o atalhos "Shift+Ctrl+B" você visualiza todos os pontos de paradas configurados.

### Executando o programa em modo de depuração

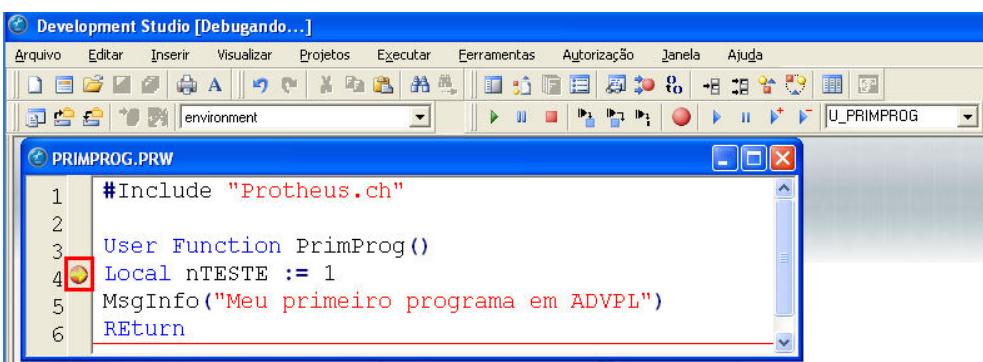
Assim que você inserir todos os pontos de parada necessários para depuração do programa, execute o programa para assim, podermos explorar os recursos disponíveis da ferramenta debug, para isso digite o nome da função no campo  e clique no botão executar, note que irá aparecer no círculo vermelho  uma seta amarela , essa informa a próxima linha que será executada, nesse momento temos a opção de visualizar o conteúdo de todas as variáveis criadas em nosso programa, logo abaixo serão demonstradas as formas de visualizar tais variáveis.

A execução do programa ficará parada até que você pressione a tecla F8, essa faz com que a linha corrente seja executada (no exemplo abaixo criará a variável nTeste), e a seta amarela aponte para a próxima linha, e a cada pressionar da tecla F8 a próxima linha ficará em foco para execução.

# Programação Advpl 1

Você também pode pressionar a tecla F5, isso fará com que o programa seja executado normalmente, isto é, sem execução linha a linha.

Caso encontre outro ponto de parada, novamente a execução direta sera interrompida ficando no aguardo da interação do desenvolvedor, esse teclar F8 para continuar.

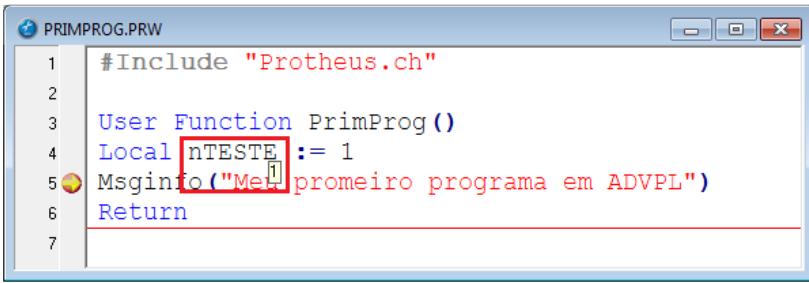


```
#Include "Protheus.ch"
User Function PrimProg()
Local nTESTE := 1
MsgInfo("Meu primeiro programa em ADVPL")
Return
```

Temos também as opções de execução descritas abaixo:

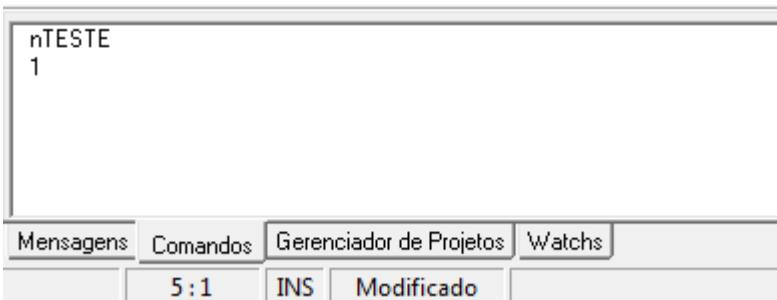
- ▶ F10 Pula execução da linha corrente
- ▶ F7 Executa até a posição do cursor de edição, esse é útil para executar rapidamente até a posição escolhida pelo desenvolvedor

## Visualizando conteúdo de variáveis



```
#Include "Protheus.ch"
User Function PrimProg()
Local nTESTE := 1
MsgInfo("Meu primeiro programa em ADVPL")
Return
```

A forma mais simples de visualizar o conteúdo de uma variável é posicionar o cursor do mouse sobre a mesma, aparecerá uma pequena janela com o atual conteúdo da variável escolhida.



|           |          |                         |        |
|-----------|----------|-------------------------|--------|
| Mensagens | Comandos | Gerenciador de Projetos | Watchs |
| 5 : 1     | INS      | Modificado              |        |

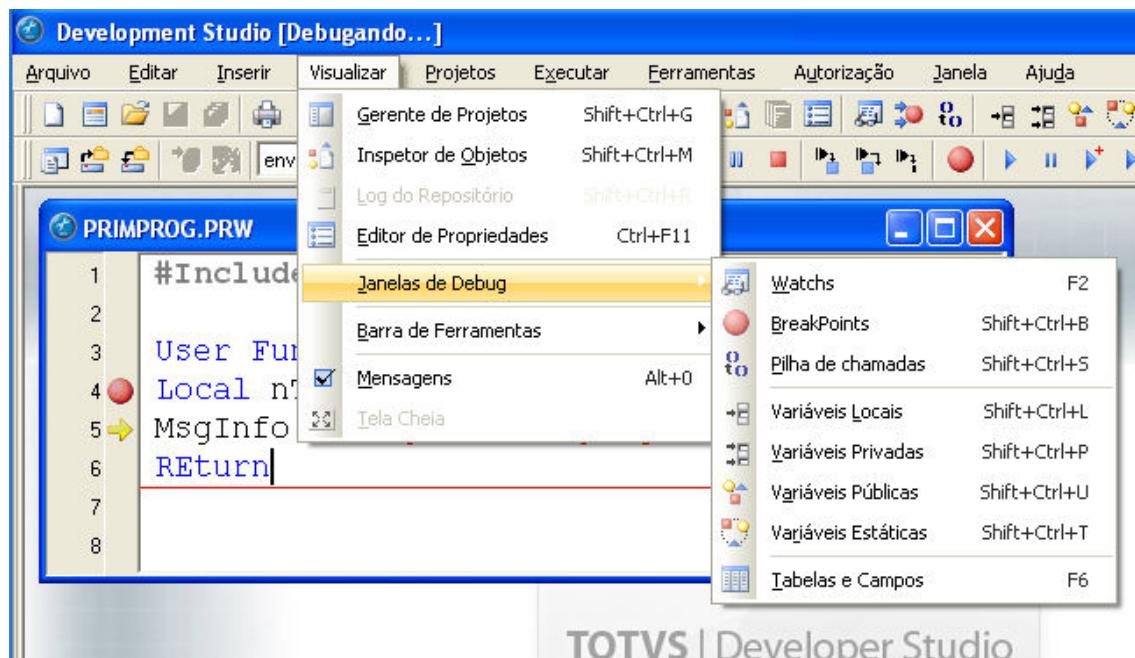
Também podemos visualizar o conteúdo de variáveis digitando seu nome da janela de comandos. Caso a mesma não esteja visível acesso as opções do menu 'Visualizar → mensagens'.

# Programação AdvPl 1

| Expressão | Tipo | Resultado |
|-----------|------|-----------|
| nTESTE    | N    | 1000      |

Através da janela Watches podemos acompanhar online as alterações das variáveis, caso essa não esteja visível, tecla F2.

Além das janelas citadas acima, temos a opção de visualizar as variáveis de acordo com seu ESCOPO, para isso acesse as opções 'Visualizar → Janelas de Debug' e escolha a opção desejada.



## Capítulo 08 – Funções Pré-Existentes

Por ser uma linguagem derivada do Clipper, podemos utilizar as diversas funções nativas da linguagem:

### Manipulação de Variáveis Tipo Caracter

|              |   |
|--------------|---|
| <b>VAL()</b> | Converte um dado tipo caractere em numérico |
|--------------|---|

**Sintaxe:** nVar := VAL( cValor )

| Argumento | Obrigat. | Tipo | Descrição                                |
|-----------|----------|------|--|
| cValor    | Sim      | C    | Conteúdo caractere que se quer converter |

**Exemplo:**

```
cCodigo := "000015"
nNum := Val(cCodigo)+1
msginfo(nNum) // nNum terá o valor de 16
```

|                 |                               |
|-----------------|-------------------------------|
| <b>SUBSTR()</b> | Retorna uma parte de um texto |
|-----------------|-------------------------------|

**Sintaxe:** cVar := SUBSTR( cConteudo, nValIni, nCount )

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cConteudo | Sim      | C    | Conteúdo que se quer extrair uma parte                    |
| nValIni   | Sim      | N    | Posição Inicial para extração                             |
| nCount    | Sim      | N    | Quant de caracteres a extrair a partir da Posição Inicial |

**Exemplo:**

```
cCodigo := "Paralelepipedo"
cPedaco1 := Substr(cCodigo,5,4) // cPedaco1 terá o conteúdo "lele"
cPedaco2 := Substr(cCodigo,1,6) // cPedaco2 terá o conteúdo "parale"
cPedaco3 := Substr(cCodigo,9,5) // cPedaco3 terá o conteúdo "piped"
```

|               |   |
|---------------|---|
| <b>LEFT()</b> | Retorna o conteúdo de uma qtde de caracteres a esquerda |
|---------------|---|

**Sintaxe:** cVar := LEFT( cConteudo, nCount )

| Argumento | Obrigat. | Tipo | Descrição                              |
|-----------|----------|------|--|
| cConteudo | Sim      | C    | Conteúdo que se quer extrair uma parte |
| nCount    | Sim      | N    | Qtde de caracteres a extrair           |

**Exemplo:**

```
cTexto := "Paralelepipedo"
cPedaco1 := Left(cTexto,5) // cPedaco1 terá o conteúdo "paral"
```

# Programação Advpl 1

**RIGHT()** Retorna o conteúdo de uma qtd de caracteres a direita

**Sintaxe:** cVar := RIGHT( cConteudo, nCount )

| Argumento | Obrigat. | Tipo | Descrição                              |
|-----------|----------|------|--|
| cConteudo | Sim      | C    | Conteúdo que se quer extrair uma parte |
| nCount    | Sim      | N    | Qtde de caracteres a extrair           |

**Exemplo:**

```
cCodigo := "Paralelepipedo"  
cPedaco1 := Right(cCodigo,5) // cPedaco1 terá o conteúdo "ipedo"
```

**PADC()** Centraliza um texto conforme a qtd de caracteres especificados

**Sintaxe:** cVar := PADC( cConteudo, nCount, cConteudo )

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cConteudo | Sim      | C    | Conteúdo que se quer centralizar                        |
| nCount    | Sim      | N    | Qtde de caracteres a para centralizar                   |
| cConteudo | Não      | C    | Caractere que ocupará os espaços em branco nos extremos |

**Exemplo:**

```
cTexto := "Parede"  
cVar1 := Padc(cTexto,14) // Retorna o conteúdo " Parede "  
cVar2 := Padc(cTexto,14,"#") // Retorna o conteúdo "####Parede#####"
```

**PADR()** Preenche com espaços em branco a direita

**Sintaxe:** PADR( cConteudo, nCount, cConteudo )

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| cConteudo | Sim      | C    | Conteúdo que se quer alinhar                           |
| nCount    | Sim      | N    | Qtde de caracteres a alinhar                           |
| cConteudo | Não      | C    | Caractere que ocupará os espaços em branco na esquerda |

**Exemplo:**

```
cTexto := "Parede"  
cVar := Padr(cTexto,14) // Retorna o conteúdo " Parede "
```

**PADL()** Preenche com espaços em branco a esquerda

**Sintaxe:** cVar := PADL( cConteudo, nCount, cConteudo )

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cConteudo | Sim      | C    | Conteúdo que se quer alinhar                          |
| nCount    | Sim      | N    | Qtde de caracteres a alinhar                          |
| cConteudo | Não      | C    | Caractere que ocupará os espaços em branco na direita |

**Exemplo:**

```
cTexto := "Parede"  
cVar := Padl(cTexto,14) // Retorna o conteúdo "Parede "
```

# Programação AdvPl 1

## ALLTRIM() Limpa espaços em branco iniciais e finais

**Sintaxe:** cVar := Alltrim( cConteudo )

| Argumento | Obrigat. | Tipo | Descrição                   |
|-----------|----------|------|-----------------------------|
| cConteudo | Sim      | C    | Conteúdo que se quer limpar |

**Exemplo:**

```
cTexto := " Parede "
cVar := Alltrim(cTexto) // cVar terá o conteúdo "Parede"
```

## LTRIM() Limpa espaços em branco a esquerda

**Sintaxe:** cVar := LTRIM( cConteudo )

| Argumento | Obrigat. | Tipo | Descrição                    |
|-----------|----------|------|------------------------------|
| cConteudo | Sim      | C    | Conteúdo que se quer alinhar |

**Exemplo:**

```
cTexto := " Parede "
cVar := LTRIM(cTexto) // cVar terá o conteúdo "Parede "
```

## RTRIM() Limpa espaços em branco a direita

**Sintaxe:** RTRIM( cConteudo )

| Argumento | Obrigat. | Tipo | Descrição                    |
|-----------|----------|------|------------------------------|
| cConteudo | Sim      | C    | Conteúdo que se quer alinhar |

**Exemplo:**

```
cTexto := " Parede "
cVar := RTRIM(cTexto) // cVar terá o conteúdo " Parede"
```

## Manipulação de Variáveis Tipo Númerica

### STR() Converte conteúdo númerico em caractere

**Sintaxe:** cVar := Str( nConteudo, nTamanho, nDecimais )

| Argumento | Obrigat. | Tipo | Descrição                       |
|-----------|----------|------|---------------------------------|
| nConteudo | Sim      | N    | Valor a converte                |
| nTamanho  | Não      | N    | Tamanho resultante da conversão |
| nDecimais | Não      | N    | Quantidade de casas decimais    |

**Exemplo:**

```
nVal := 1
cVar1 := Str(nVal) // cVar1 terá o conteúdo " 1"
cVar2 := Str(nVal,7,2) // cVar2 terá o conteúdo " 1.00"
```

Obs: Caso não seja informado o tamanho resultante, o padrão é 10

# Programação Advpl 1

## STRZERO() Converte número em caractere e preenche zero a esquerda

**Sintaxe:** cVar := Strzero( nConteudo, nCount )

| Argumento | Obrigat. | Tipo | Descrição                |
|-----------|----------|------|--------------------------|
| nConteudo | Sim      | N    | Valor a converte         |
| nCount    | Sim      | N    | Quantidade de caracteres |

**Exemplo:**

```
nVar := 1  
cVar := Strzero(nVar, 6) // cVar terá o conteúdo "000001"
```

Obs: Caso não seja informado o tamanho resultante, o padrão é 10

## CVALTOCHAR() Converte número em caractere limpando espaços em branco

**Sintaxe:** cVar := CVALTOCHAR( nConteudo )

| Argumento | Obrigat. | Tipo | Descrição        |
|-----------|----------|------|------------------|
| nConteudo | Sim      | N    | Valor a converte |

**Exemplo:**

```
nVar := 123.75  
cVar := CVALTOCHAR(nVar, 6) // cVar terá o conteúdo "123.75"
```

## TRANSFORM() Converte número e/ou caractere de acordo com uma máscara

**Sintaxe:** cVar := Transform( nConteudo, cMask )

| Argumento | Obrigat. | Tipo | Descrição           |
|-----------|----------|------|---------------------|
| nConteudo | Sim      | N    | Valor a converte    |
| cMask     | Sim      | N    | Formato do caracter |

**Exemplo:**

```
nVar := 1500  
cVal := Transform(nVar, "@E 99,999.99") // Resultado " 1.500,00"  
  
cVar := "13817710805"  
cCPF := Transform(cVar, "@R 999.999.999-99")
```

No segundo exemplo não ocorre uma conversão de conteúdo numérico em caractere, mas sim uma transformação de conteúdo caractere respeitando a máscara definida, resultando assim em "138.177.108-05"

## Manipulação de Variáveis Tipo Data

### DATE() Retornada data do sistema operacional

**Sintaxe:** dVar := DATE()

**Exemplo:**

```
dDataAtual := date() // Retornará a data corrente
```

# Programação AdvPl 1

## DTOS() Converte conteúdo tipo data em string

**Sintaxe:** cVar := DTOS( dData )

| Argumento | Obrigat. | Tipo | Descrição                       |
|-----------|----------|------|---------------------------------|
| dData     | Sim      | D    | Data a converter para AAAAMMMDD |

**Exemplo:**

Supondo que hoje é 31/01/2012

cData := dtos(date()) // Resultado da conversão "20120131"

## STOD() Converte conteúdo tipo string para data

**Sintaxe:** cVar := STOD( cData )

| Argumento | Obrigat. | Tipo | Descrição        |
|-----------|----------|------|------------------|
| cData     | Sim      | C    | Data a converter |

**Exemplo:**

cData := "20120131"

dData := STOD(cData) // Resultado da conversão 31/01/2012 (formato data)

## DTOC() Converte conteúdo tipo data para caractere

**Sintaxe:** cData := DTOC( dData )

| Argumento | Obrigat. | Tipo | Descrição                   |
|-----------|----------|------|-----------------------------|
| dData     | Sim      | D    | Data a converter DD/MM/AAAA |

**Exemplo:**

Supondo que hoje é 31/01/2012

cData := DTOC(DATE()) // Resultado da conversão "31/01/2012"

## CTOD() Converte conteúdo tipo caractere para data

**Sintaxe:** dVar := CTOD( cData )

| Argumento | Obrigat. | Tipo | Descrição        |
|-----------|----------|------|------------------|
| cData     | Sim      | C    | Data a converter |

**Exemplo:**

cData := "31/01/2012"

dData := CTOD(cData) // Resultado da conversão 31/01/2012 (tipo data)

## MONTH () Retorna o mês de uma data informada

**Sintaxe:** nMes := Month( dData )

| Argumento | Obrigat. | Tipo | Descrição |
|-----------|----------|------|-----------|
| dData     | Sim      | D    | Data      |

**Exemplo:**

nMes := Month(date()) // Retorna o número 01 para a data 31/01/2012

# Programação AdvPl 1

**DAY ()** Retorna o dia de uma data informada

**Sintaxe:** nVar := Day( dData )

| Arguento | Obrigat. | Tipo | Descrição |
|----------|----------|------|-----------|
| dData    | Sim      | D    | Data      |

**Exemplo:**

nDia := Day(date()) // Retorna o número 31 para a data 31/01/2012

**YEAR ()** Retorna o ano de uma data informada

**Sintaxe:** nVar := Year( dData )

| Arguento | Obrigat. | Tipo | Descrição |
|----------|----------|------|-----------|
| dData    | Sim      | D    | Data      |

**Exemplo:**

nAno := Year(date()) // Retorna o número 12 para a data 31/01/2012

**MESEXTENSO()** Retorna o Mês por extenso em português

**Sintaxe:** cMes := MESEXTENSO( DATE() )

| Arguento | Obrigat. | Tipo | Descrição |
|----------|----------|------|-----------|
| dData    | Sim      | D    | Data      |

**Exemplo:**

MesExtenso( dData )

cMes := MesExtenso(date()) // Retorna "JANEIRO" para a data 31/01/2012

**GRAVADATA()** Retorna data com formato diferente conforme parametros

**Sintaxe:** cData := GRAVADATA(ExpD1,ExpL1,ExpN1)

| Arguento | Obrigat. | Tipo | Descrição  |
|----------|----------|------|--|
| ExpD1    | Sim      | D    | Data   |
| ExpL1    | Não      | L    | .T. com Barra .F. sem Barra  |
| ExpN1    | Sim      | N    | 1-ddmmaa<br>2-mmddaa<br>3-aaddmm<br>4-aammdd<br>5-ddmmaaaa<br>6-mmddaaaa<br>7-aaaaddmm<br>8-aaaammdd |

**Exemplo:**

cData := GRAVADATA(dData,.F.,8) // Retorna 20120131

# Programação AdvPl 1

## Janelas de Aviso

**MSGINFO()** Apresenta uma mensagem de informação

**Sintaxe:** MsgInfo ( < cMensagem>, < cTitulo> )

| Argumento | Obrigat. | Tipo | Descrição                                      |
|-----------|----------|------|--|
| cMensagem | Sim      | C    | Indica a mensagem que será apresentada         |
| cTitulo   | Não      | C    | Indica o título da janela que será apresentada |

**Exemplo:**

MsgInfo("Mensagem !!!","Titulo")



**ALERT()** Apresenta uma mensagem ao usuário

**Sintaxe:** MsgInfo ( cMensagem )

| Argumento | Obrigat. | Tipo | Descrição                              |
|-----------|----------|------|--|
| cMensagem | Sim      | C    | Indica a mensagem que será apresentada |

**Exemplo:**

MsgInfo("Mensagem !!")



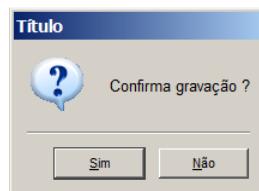
**MSGYESNO()** Apresenta uma mensagem ao usuário com opção SIM/NAO

**Sintaxe:** MsgInfo ( cMensagem, [cTitulo] )

| Argumento | Obrigat. | Tipo | Descrição                              |
|-----------|----------|------|--|
| cMensagem | Sim      | C    | Indica a mensagem que será apresentada |
| cTitulo   | Não      | C    | Título da janela                       |

**Exemplo:**

MsgYesNo ( "Confirma gravação ?" , "Titulo" )



**AVISO()** Apresenta janela de aviso exibindo o texto desejado e botões definidos pelo desenvolvedor

# Programação AdvPl 1

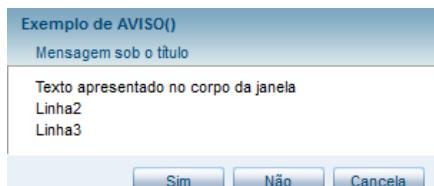
**Sintaxe:** MsgInfo ( cTitulo, cMensagem, aBotoes, nTamanho, cTexto )

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| cTitulo   | Sim      | C    | Título que será apresentado ao usuário                 |
| cMensagem | Sim      | C    | Indica a mensagem que será apresentada                 |
| aBotoes   | Sim      | A    | Array com botões que serão disponibilizados ao usuário |
| nFormato  | Não      | N    | Define o formato da janela apresentada                 |
| cTexto    | Não      | C    | Texto apresentado no corpo da janela                   |

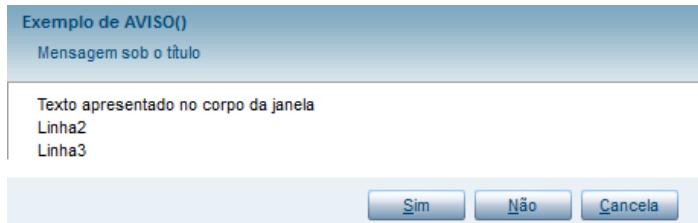
**Exemplo:**

```
cTitulo      := "Exemplo de AVISO()"
cMensagem   := "Texto apresentado no corpo da janela"+CRLF
cMensagem += "Linha2"+CRLF
cMensagem += "Linha3"
aBotoes      := {"Sim","Não","Cancela"}
nTamanho    := 1
cTexto       := "Mensagem sob o título"
```

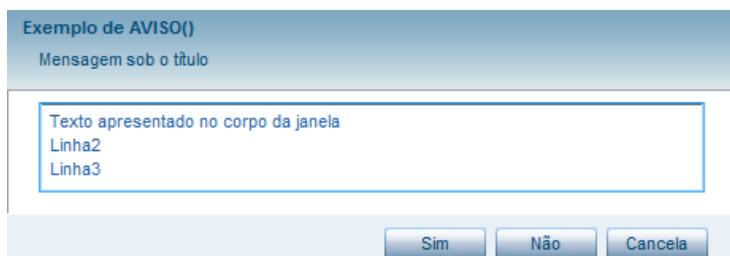
```
nRet := AVISO(cTitulo, cMensagem, aBotoes, nTamanho, cTexto )
```



Formato 1



Formato 2



Formato 3

# Programação AdvPl 1

---

## Exercícios

1. Utilizando as variáveis listadas abaixo, crie uma “função de usuário” que apresente na tela a frase: “Total de vendas do dia 20/01/2012 foi de 1.786,11”, para essa apresentação use a função MsgInfo().

nTotVenda → 1786.11

dDtVenda → 20/01/2012

2. Utilizando as variáveis listadas abaixo, crie uma função de usuário que calcule o valor do imposto e apresente na tela a frase: “Base de calculo: 5.000,00 Aliquota 18% Valor do imposto: 900,00”.

nBaseCalc → 5000

nAliq → 18

nImposto → 900

## Capítulo 09 - Utilização de Vetores

Um vetor é uma variável especial onde temos a possibilidade de armazenar dados de diferentes tipos, cada dado (elemento) do vetor é referenciado por um número com um número único, exemplo de um vetor com 7 elementos:

|                      |     |     |     |    |    |     |     |
|----------------------|-----|-----|-----|----|----|-----|-----|
| Nom do vetor: aArray | 1   | 2   | 3   | 4  | 5  | 6   | 7   |
| Conteúdo             | "A" | "B" | "C" | 55 | 77 | .F. | .T. |

Nesse exemplo temos o vetor chamado aArray com 7 elementos, sendo 3 do tipo "caractere", 2 do tipo "numérico" e 2 do tipo "lógico".

Observação: o vetor aArray é classificado com UNIDIMENSIONAL, a seguir trataremos outra classificação, a BI-DIMENSIONAL.

Vetores podem ser chamados de Array ou matrizes.

### Criando um vetor utilizando ADVPL

Para criar-mos um vetor temos a sintaxe abaixo, sempre que for criar um vetor deve utilizar os caracates especiais "{}", assim o ADVPL reconhece a variável como tipo "array".

Nesse exemplo será criado o vetor aNomes com conteúdo vazio:

```
aNomes := {}
```

Exemplos de criação de vetores com conteúdo:

```
aArray := {"A","B","C",55,77,.F.,.T.}  
aNomes := {"Roberto","Joao","Ana"} // Criando vetor com 3 elementos  
aIdade := {32,21,19} // Criando vetor com conteúdo numérico
```

### Acessando conteúdo de um vetor

Para acessarmos o conteúdo dos vetores, seja para alterar ou para apenas utilizar seu conteúdo, devemos "SEMPRE" infomar o número do elemento que queremos acessar, esse entre os operadores especiais "[]".

Exemplos de acesso a conteúdo de vetores:

1. Para mostrar o nome "Ana" na tela se utilizando da função MsgInfo():

```
MsgInfo(aNomes[3])
```

Nesse caso informamos o número 3 porque o elemento do vetor que queremos mostrar está nessa posição, da mesma forma podemos fazer para os demais elementos, apenas informando seu número.

2. Acesso ao conteúdo do para criação da variável cNomeCli, nessa foi gravado o segundo elemento do vetor "João":

```
cNomeCli := aNomes[2]
```

# Programação Advpl 1

- 
3. Agora vamos mostrar na tela o conteúdo dos 2 vetores criados acima, para isso devemos acessar cada um deles separadamente, veja:

```
cIdade := aNomes[1] + " +CVALTOCHAR(aIdade[1])
```

Repara que para ter-mos o conteúdo “Roberto 32” foi necessário acessar os 2 vetores, note também que foi preciso usar uma função de conversão “CVALTOCHAR”, pois o elemento número 1 do vetor aIdade é do tipo “numérico” e o elemento 1 do vetor aNomes é do tipo “caractere”.

## Alterando o conteúdo de um vetor

Ainda usando os vetores criados acima “aNomes e aIdade”, iremos demonstrar como podemos realizar a alteração de conteúdo.

```
aNomes[1] := "Jose"
```

Com a linha acima foi alterado o conteúdo do elemento 1 do vetor, antes era “Roberto”, agora será “Jose”.

```
aIdade[2] += 1
```

Com a linha acima foi alterado o conteúdo do elemento 2 do vetor, antes era 21 passou a ser 22.

## Vetores bi-dimensionais

Podemos ter os dados dos vetores “aNomes” e “aIdade” contidos em um único vetor.

Graficamente representados:

| 1       | 2  | 3    |    |     |    |
|---------|----|------|----|-----|----|
| Roberto | 32 | Joao | 21 | Ana | 19 |
| 1       | 2  | 1    | 2  | 1   | 2  |

Utilizando sintaxe ADVPL criamos o vetor acima da seguinte forma:

```
aPessoas := { {"Roberto",32} , {"Joao",21} , {"Ana",19} }
```

Veja que aPessoas é um vetor que tem 3 elementos do tipo unidimensional, sendo:

```
1 = {"Roberto",32}  
2 = {"Joao",21}  
3 = {"Ana",19}
```

Para acessar-mos seus conteúdo já não basta informar o número do elemento, mas também o número do elemento na segunda dimensão, por exemplo:

```
MsgInfo( aPessoas[2,1] )
```

Nesse caso será mostrado na tela o conteúdo “Joao”

Para acessar-mos o nome “Ana” temos que informar aPessoas[3,1].

# Programação AdvPL 1

Também podemos utilizar a sintaxe abaixo para acessar o conteúdo de um array bi-dimencional:

```
MsgInfo( aPessoas[2][1] )
```

Uma outra forma de demonstrar a utilização um vetor bi-dimencional é uma lista com linhas e colunas. Comparando com uma tabela, as linhas são os registros e as colunas são os campos, podemos então dizer que é uma tabela virtual.

Cada item do vetor é referenciado pela sua posição na lista, iniciando pelo número 1, veja exemplo:

| AArray | Col1 | Col2 | Col3 |
|--------|------|------|------|
| Lin1   | A    | B    | C    |
| Lin2   | D    | E    | F    |
| Lin3   | G    | H    | I    |

A montagem deste array em AdvPL se dá da seguinte forma:

```
aArray := { {"A", "B", "C"} , {"D", "E", "F"} , {"G", "H", "I"} }
```

```
MsgInfo(aArray[2][3]) //Exibe o Conteúdo da 2ª. Linha e 3a. Coluna, (F)  
MsgInfo(aArray[1][1]) //Exibe o Conteúdo da 1ª. Linha e 1a. Coluna, (A)  
MsgInfo(aArray[3][2]) //Exibe o Conteúdo da 3ª. Linha e 2a. Coluna, (H)
```

Para atribuirmos outro conteúdo utilizamos a seguinte sintaxe:

```
aArray[nLin][nCol] := cVar
```

Onde:

nLin -> Linha onde se encontra o conteúdo que se quer alterar

nCol -> Coluna onde se encontra o conteúdo que se quer alterar

cVar -> Novo Conteúdo

```
aArray[2][3] := "X" // Altera o Conteúdo da 2ª. Linha e 3a. Coluna de F para X  
aArray[1][1] := "Y" // Altera o Conteúdo da 1ª. Linha e 1a. Coluna de A para Y  
aArray[3][2] := "Z" // Altera o Conteúdo da 3ª. Linha e 2a. Coluna de H para Z
```

Após essa alteração a nossa lista virtual ficará assim:

| AArray | Col1 | Col2 | Col3 |
|--------|------|------|------|
| Lin1   | Y    | B    | C    |
| Lin2   | D    | E    | X    |
| Lin3   | G    | Z    | I    |

# Programação Advpl 1

## Funções para manipulação de vetores

**AADD()** Adiciona um novo elemento ao final do array

**Sintaxe:** AADD( aAlvo, expValor )

| Argumento | Obrigat. | Tipo  | Descrição   |
|-----------|----------|-------|---|
| aAlvo     | Sim      | A     | É o array ao qual o novo elemento será adicionado.        |
| expValor  | Sim      | Todos | É uma expressão válida que será o valor do novo elemento. |

**Exemplos:**

```
aArray := {}           // Resultado: aArray vazio  
AADD(aArray, 5)        // Resultado: { 5 }  
AADD(aArray, 10)       // Resultado: { 5, 10 }  
AADD(aArray, { 12, 10 }) // Resultado: { 5, 10, { 12, 10 } }
```

**ARRAY()** Cria um array com dados não inicializados

**Sintaxe:**

aArray := ARRAY( nQtdElem1 , [ nQtdElemn ]...)

| Argumento   | Obrigat. | Tipo | Descrição  |
|-------------|----------|------|--|
| nQtdElem1   | Sim      | N    | Quantidade de Elementos da 1ª dimensão do array.       |
| [nQtdElemN] | Não      | N    | Quantidade de Elementos das demais dimensões do array. |

**Exemplos:**

aArray := ARRAY(5)

Conteúdo criado pela linha de comando acima:

{ NIL, NIL, NIL, NIL, NIL }

aArray := ARRAY(3, 2)

Conteúdo criado pela linha de comando acima:

{ {NIL, NIL}, {NIL, NIL}, {NIL, NIL} }

**ASCAN()** Realiza pesquisa em array (unidimensional e bi-dimencional)

**Sintaxe:** ASCAN( aOrigem, expSearch, [ nStart ], [ nCount ] ) --> nStoppedAt

| Argumento | Obrigat. | Tipo  | Descrição  |
|-----------|----------|-------|--|
| aOrigem   | Sim      | A     | É o array onde será executada a pesquisa.  |
| expSearch | Sim      | Todos | Conteúdo que será pesquisado   |
| nStart    | Não      | N     | Elemento inicial para pesquisa, se não informado inicia-se a pesquisa a partir do elemento 1 |
| nCount    | Não      | N     | Quantidade de elementos "apartir do inicial" onde o ascan efetuará sua pesquisa              |

# Programação AdvPl 1

## Exemplo 1 (pesquisa em array unidimensional)

```
aArray := { "Tony", "Maria", "Sueli" }  
nPos := ASCAN(aArray, "Maria") // Result: 2  
nPos1 := ASCAN(aArray, "maria") // Result: 0
```

nPos terá seu conteúdo igual a 2, esse é o número do elemento onde a palavra "Maria" foi encontrada

nPos1 terá seu conteúdo 0 pois não foi encontrada nenhuma palavra igual "maria", note que a função faz distinção entre maiúsculo e minúsculo.

## Exemplo 2 (pesquisa em array bi-dimencional)

```
aArray5 := {}  
AADD(aArray5, {"um", "dois"})  
AADD(aArray5, {"tres", "quatro"})  
AADD(aArray5, {"cinco", "seis"})  
nPos3 := ASCAN(aArray5, {|x| x[2] == "quatro"}) // Result: 2
```

| AArray5 | Col1  | Col2   |
|---------|-------|--------|
| Lin1    | Um    | Dois   |
| Lin2    | Três  | Quatro |
| Lin3    | Cinco | Seis   |

Note o detalhe **x[2]**, isso faz com que a função aScan faça a pesquisa em todos os elementos, mas observando "sempre" a coluna 2.

Retornou o valor 2 na variável nPos3 pois foi encontrada a palavra "quatro" no elemento de número 2.

## ASORT() Ordena um array

**Sintaxe:** ASORT( aOrigem, [ nInicio ], [ nQtde ], [ bOrdem ] )

| Argumento | Obrigat. | Tipo            | Descrição   |
|-----------|----------|-----------------|---|
| aOrigem   | Sim      | A               | É o array que será classificado.  |
| nInicio   | Não      | N               | Onde será o inicio da ordenação. Caso seja omitido, será considerado o 1º elemento do array.  |
| nQtde     | Não      | N               | Quantidade de elementos que serão ordenados a partir do nInicio. Caso seja omitido, serão considerados todos elementos até o final do Array.  |
| bOrder    | Não      | Bloco de código | É um bloco de código ( code block ) opcional que indicará a ordem correta dos elementos. Caso ele não seja informado, o array será classificado em ordem crescente desde que não seja multidimensional. |

## Exemplo de ordenação de array unidimensional

Temos o array abaixo, seu conteúdo está desordenado, a partir da execução da função ASORT seu conteúdo é ordenado.

```
aArray := { 3, 5, 1, 2, 4 }  
ASORT(aArray) → Resultado após execução é { 1, 2, 3, 4, 5 }
```

# Programação AdvPl 1

## **Exemplo de ordenação em array bi-dimencional**

aArray1 := { {"Maria", 14}, {"Joao", 23}, {"Arlete", 16} }

| AArray1 | Col1   | Col2 |
|---------|--------|------|
| Lin1    | Maria  | 14   |
| Lin2    | Joao   | 23   |
| Lin3    | Arlete | 16   |

### **Ordenando pela coluna 1 (ordem crescente)**

Vamos ordenar nosso aArray1 pela coluna 1, isto é “ordem alfabética dos nomes”, o resultado desse ordenação será gravado em um novo array, chamo aArray2, veja:

aArray2 := ASORT(aArray1, , ,{|X,Y| X[1] < Y[1]} )

Resultado da ordenação: { {"Arlete", 16}, {"Joao", 23}, {"Maria", 14} }

| AArray1 | Col1   | Col2 |
|---------|--------|------|
| Lin1    | Arlete | 16   |
| Lin2    | Joao   | 23   |
| Lin3    | Maria  | 14   |

### **Ordenando pela coluna 1 (ordem decrescente)**

Agora vamos ordenar nosso aArray1 pela coluna 1, mas na ordem “decrescente”, o resultado desse ordenação será gravado em um novo array, chamo aArray2, veja:

aArray2 := ASORT(aArray1, , ,{|X,Y| X[1] > Y[1]} )

Resultado da ordenação: { {"Arlete", 16}, {"Joao", 23}, {"Maria", 14} }

| AArray1 | Col1   | Col2 |
|---------|--------|------|
| Lin1    | Maria  | 14   |
| Lin2    | Joao   | 23   |
| Lin3    | Arlete | 16   |

#### **Dica:**

Para realizar a ordenação por outra coluna basta informar o número dessa na sintaxe da função ASORT, exemplo:

ASORT(aArray1, , ,{|X,Y| X[2] > Y[2]} )

## Cuidados com Vetores

Vetores são listas de elementos virtuais, utilizamos então a memória, necessária para armazenar estas informações. Como as matrizes podem ser multidimensionais, a memória necessária será a multiplicação do número de itens em cada dimensão da matriz, portanto precisamos usá-las com cautela.

Quando seu conteúdo por algum motivo começar a ficar muito complexo, é recomendável utilizarmos outros recursos, com criação de tabelas temporárias.

Não há limitação para o número de dimensões que uma matriz pode ter, mas o número de elementos máximo (independentes das dimensões onde se encontram) é de 100000 (cem mil).

## Matrizes como Estruturas

Uma característica interessante do AdvPI é que uma matriz pode conter qualquer coisa: números, datas, lógicos, caracteres, objetos, etc. E ao mesmo tempo. Em outras palavras, os elementos de uma matriz não precisam ser necessariamente do mesmo tipo de dado.

## Exercícios

1. Criar um vetor “uni-dimencional” com os nomes de seus colegas de classe.
2. Escolha um de seus colegas e altere o seu nome, complementando com seu sobrenome.
3. Criar um vetor de 5 linhas com 3 Colunas e dar o nome de aArray1.
4. Criar um vetor de 10 linhas com 1 Coluna e dar o nome de aArray2.
5. Atribuir ao Vetor a Array1 linha 1 coluna 1 o conteúdo “X”
6. Atribuir ao Vetor a Array1 linha 3 coluna 2 o conteúdo 9
7. Atribuir ao Vetor a Array1 linha 5 coluna 3 o conteúdo .T.
8. Atribuir ao Vetor a Array1 linha 1 coluna 3 o conteúdo do aArray2 Linha 5 coluna

## Capítulo 10 – Controle de Fluxo

Podemos mudar a seqüência de fluxo de execução de um programa baseado em condições lógicas e a repetição da execução de pedaços de código quantas vezes forem necessárias para tal utilizamos estruturas de controle com um identificador de início e um de fim, e o que deve ser executado deve estar entre estes identificadores. Essas estruturas em AdvPl estão divididas em :

**Estruturas de Repetição:** Execução de uma seção de código mais de uma vez. Ex. For / While

**Estruturas de Decisão:** Execução de uma seção de código de acordo com uma condição lógica. Ex. If / Case.

### Comandos de Repetição

Em AdvPl existem dois comandos para a repetição de seções de código. O comando FOR...NEXT e o comando WHILE...ENDDO.

#### FOR...NEXT

A estrutura de controle FOR...NEXT repete uma seção de código em um número determinado de vezes.

##### Sintaxe:

```
FOR Variavel := nValorInicial TO nValorFinal [STEP nIncremento]
    Comandos...
    [EXIT]
    [LOOP]
NEXT
```

##### Parâmetros

|                                 |  |
|---------------------------------|--|
| Variável                        | Variável utilizada como contador. Se a variável não existir, será criada como uma variável privada (private).  |
| nValorInicial TO<br>nValorFinal | nValorInicial é o valor inicial para o contador; nValorFinal é o valor final para o contador. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.  |
| STEP nIncremento                | nIncremento é a quantidade que será incrementada ou decrementada no contador após cada execução da seção de comandos. Se o valor de nIncremento for negativo, o contador será decrementado. Se a cláusula STEP for omitida, o contador será incrementado em 1. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico. |
| Comandos                        | Especifica uma ou mais instruções de comando AdvPl que serão executadas.   |
| EXIT                            | Finaliza a repetição da seção de comandos imediatamente.   |
| LOOP                            | Retorna ao Início do FOR sem executar o restante dos comandos, incrementando ou decrementando normalmente.   |

Este exemplo separa os números pares e ímpares em variáveis para imprimir ou mostrar em tela.

```
For X := 1 to 12
    If mod(X,2)== 0
        cBIPares += Alltrim(str(X)) + " "
    Else
        cBIImpares += Alltrim(str(X)) + " "
    Endif
Next
```

# Programação AdvPl 1

Este outro exemplo irá mostrar na tela todos os nomes do nosso vetor aNomes

```
aNomes := {"Roberto","Joao","Ana"}
```

```
For X := 1 to 3  
    MsgInfo( aNomes[ X ] )  
Next
```

Veja que não mais acessamos o conteúdo do vetor aNomes passando o número do elemento, mas sim a variável X, isso porque a cada execução do comando FOR essa variável terá seu valor acrescido em 1, partindo do valor inicial 1.



## WHILE...ENDDO

A estrutura de controle WHILE...ENDDO, repete uma seção de código enquanto uma determinada condição resultar em verdadeiro (.T.)

### Sintaxe:

```
WHILE /Expressao  
    Comandos...  
    [EXIT]  
    [LOOP]  
ENDDO
```

### Parâmetros

|            |  |
|------------|--|
| LExpressao | Especifica uma expressão lógica cujo valor determina quando os comandos entre o WHILE e o ENDDO são executados. Enquanto o resultado de LExpressao for avaliado como verdadeiro (.T.), o conjunto de comandos são executados.                      |
| Comandos   | Especifica um ou mais instruções de comando AdvPl que serão executadas enquanto LExpressao for avaliado como verdadeiro (.T.).   |
| EXIT       | Transfere o controle de dentro do comando WHILE...ENDDO para o comando imediatamente seguinte ao ENDDO, ou seja, finaliza a repetição da seção de comandos imediatamente. Pode-se colocar o comando EXIT em qualquer lugar entre o WHILE e o ENDO. |
| LOOP       | Retorna o controle diretamente para a cláusula WHILE sem executar o restante dos comandos entre o LOOP e o ENDDO. A expressão em LExpressao é reavaliada para a decisão se os comandos continuarão sendo executados.                               |

Este exemplo separa os números pares e ímpares em variáveis para imprimir ou mostrar em tela.

```
Local nSomaPar := 0, nNumber := 350  
nNumber := Int(nNumber / 2)  
While nNumber > 0  
    nSomaPar++  
    nNumber:= nNumber - 2  
Enddo  
Alert("A quantidade de pares é: " + str(nSomaPar))  
Return
```



# Programação AdvPI 1

## Problemas Comuns

Looping, é uma repetição infinita, ou seja, por um erro na lógica a execução do while não finaliza, com isso o processo pode utilizar todo o recurso do servidor e diminuir a performance ou até mesmo travá-lo, tornando necessário o reinicio do sistema.

|   |  |
|---|--|
| <b>Exemplo 1:</b><br>Do While SE1->(!Eof())<br>@ lin, col SAY "teste"<br>Enddo        | Nesse exemplo está faltando o comando para passar para o próximo registro dbkip(), ou seja, ele nunca chegará no final de arquivo conforme a condição para sair do While   |
| <b>Exemplo 2:</b><br>aCampos := {}<br>Do while .T.<br>Aadd(aCampos, "Teste")<br>Enddo | Nesse exemplo o sistema ficará rodando até alcançar o limite da variável tipo Array, até que isso aconteça o programa estará utilizando a memória do servidor e com isso diminuindo o desempenho de toda a empresa.                    |
| <b>Exemplo 3:</b><br>Do While SE1->(!Eof())<br>Reclock("SE5", .T.)<br>Enddo           | Nesse exemplo o sistema ficará rodando até acabar o espaço em disco no servidor, diminuindo o desempenho de toda a empresa e impedindo o uso por todos os usuários, até que o arquivo seja ajustado, apagando os registros incorretos. |

## Comandos de Decisão

Em AdvPI existem dois comandos de decisão. O comando IF...ENDIF e o comando CASE...ENDCASE.

### IF...ENDIF

Executa um conjunto de comandos baseado no valor de uma expressão lógica.

#### Sintaxe:

```
IF <IExpressao lógica>
    comandos
    [ELSE
        Comandos...]
ENDIF
```

#### Parâmetros

|            |   |
|------------|---|
| IExpressao | Especifica uma expressão lógica que é avaliada. Se IExpressao resultar em verdadeiro (.T.), qualquer comando seguinte ao IF e antecedente ao ELSE ou ENDIF (o que ocorrer primeiro) será executado.<br>Se IExpressao resultar em falso (.F.) e a cláusula ELSE for definida, qualquer comando após essa cláusula e anterior ao ENDIF será executada. Se a cláusula ELSE não for definida, todos os comandos entre o IF e o ENDIF são ignorados. Neste caso, a execução do programa continua com o primeiro comando seguinte ao ENDIF. |
| Comandos   | Conjunto de comandos AdvPI que serão executados dependendo da avaliação da expressão lógica em IExpressao.  |

#### Exemplo

```
Local dVencto := CTOD("31/12/2010")
If Date() > dVencto
    Alert("Tituto vencido!")
Else
    Alert("Tituto a vencer!")
Endif
Return
```

## DO CASE...ENDCASE

Executa o primeiro conjunto de comandos cuja expressão condicional resulta em verdadeiro (.T.).

### Sintaxe:

```
DO CASE
CASE /Expressao1
Commandos
[CASE /Expressao2
Commandos
...
CASE /ExpressaoN
Commandos]
[OTHERWISE
Commandos]
ENDCASE
```

### Parâmetros

|                                   |   |
|-----------------------------------|---|
| CASE<br>Expressao1<br>Comandos... | Quando a primeira expressão CASE resultante em verdadeiro (.T.) for encontrada, o conjunto de comandos seguinte é executado. A execução do conjunto de comandos continua até que a próxima cláusula CASE, OTHERWISE ou ENDCASE seja encontrada. Ao terminar de executar esse conjunto de comandos, a execução continua com o primeiro comando seguinte ao ENDCASE.<br>Se uma expressão CASE resultar em falso (.F.), o conjunto de comandos seguinte a esta até a próxima cláusula é ignorado.<br>Após a execução, qualquer outra expressão CASE posterior é ignorada (mesmo que sua avaliação resultasse em verdadeiro). |
| OTHERWISE<br>Comandos             | Se todas as expressões CASE forem avaliadas como falso (.F.), a cláusula OTHERWISE determina se um conjunto adicional de comandos deve ser executado. Se essa cláusula for incluída, os comandos seguintes serão executados e então o programa continuará com o primeiro comando seguinte ao ENDCASE. Se a cláusula OTHERWISE for omitida, a execução continuará normalmente após a cláusula ENDCASE.   |

### Exemplo

```
Local nMes      := Month(Date())
Local cPeriodo := ""
DO CASE
CASE nMes <= 3
    cPeriodo := "Primeiro Trimestre"
CASE nMes >= 4 .And. nMes <= 6
    cPeriodo := "Segundo Trimestre"
CASE nMes >= 7 .And. nMes <= 9
    cPeriodo := "Terceiro Trimestre"
OTHERWISE
    cPeriodo := "Quarto Trimestre"
ENDCASE
Return
```

## Exercícios

1. Fazer Algoritmo que leia um vetor de 3 linhas e 5 colunas e imprima o seu conteúdo na tela, utilizando o que foi visto em vetores e estruturas de repetição.

2. Com base no Array informado mostrar na tela a media de cada aluno:

```
Local aArray := { { "Maria", 10, 7, 15, 31} ,;
                  { "Jose ", 15, 16, 21, 33} ,;
                  { "Petruncio", 8, 8, 8, 6} ,;
                  { "Firmino", 15, 16, 21, 33} ,;
                  { "Felizberto", 10, 17, 31, 25} }
```

3. Com base nesse mesmo array imprimir o nome informando mostrar na tela a media de cada aluno, conforme regra a seguir:

Nome do aluno seguida da palavra

14 **Aprovado** se a media for maior que 25

15 **Exame** se a media for entre 10 e 25

16 **Reprovado** se a media for menor que 10

4. Matheus é um homem de negócios e sempre viaja ao exterior e precisa controlar tudo que traz de lá. Sempre que ele traz mercadorias que ultrapassam R\$ 10.000,00, deve ser pago o imposto de 15%. Faça um algoritmo que leia o valor da mercadoria e grave na variável M o valor da mercadoria e se ultrapassar o valor, calcular o valor do imposto na variável I, caso não ultrapasse grave 0.

5. Calcule o Salário de um funcionário recebendo em uma variável o valor hora e em outra a quantidade de horas. Caso a qtde de horas ultrapasse a meta (180) acrescentar 2,00 por hora trabalhada e em seguida mostre na tela o salário a receber.

6. Desenvolva um fluxograma que:

- Leia 4 (quatro) números;
- Calcule o quadrado de cada um;
- Se o valor resultante do quadrado do terceiro for  $\geq 1000$ , imprima-o e finalize;
- Caso contrário, imprima os valores lidos e seus respectivos quadrados.

7. Elabore um algoritmo que dada a idade de um nadador classifique-o em uma das seguintes categorias:

8. Infantil A = 5 a 7 anos

Infantil B = 8 a 11 anos

Juvenil A = 12 a 13 anos

Juvenil B = 14 a 17 anos

Adultos = Maiores de 18 anos

## Capítulo 11 – Blocos de Códigos

### Definição

O tipo de dado chamado “bloco de código” permite que o código compilado seja tratado como um dado. Pode-se considerar um bloco de código como uma “**função sem nome**” atribuída a uma variável. E como o AdvPL trata os blocos de código como valores, eles podem ser passados como parâmetros e armazenados em variáveis.

As operações que podem atuar em um bloco de código são as de atribuição e avaliação. A atribuição de um bloco de código é feita geralmente a uma variável, mas eles também podem ser um elemento de um vetor ou passados como parâmetros. A avaliação de um bloco de código pode ser realizada sobre um valor, para cada elemento de um vetor, ou um bloco de código para avaliar cada registro de uma base de dados.

**Sintaxe:** uRet := { |<argumentos>| <expressões> }

| Argumento  | Obrigat. | Tipo     | Descrição   |
|------------|----------|----------|---|
| Argumentos | Não      | Diversos | Lista de argumentos que o bloco receberá, cada argumento deve ser separado por vírgula.   |
| Expressões | Sim      | Diversos | Lista de expressões que o bloco executará, cada instrução deverá ser separada por vírgula |
| URet       |          |          | Retorno será o resultado da última expressão da lista de expressões.                      |

**Exemplo:**

bBloco:= { | Param1, Param2| fTeste(Param, Param2) }

Como um bloco de código é como uma função sem nome, veja o bloco e sua representação equivalente abaixo:

| Bloco                       | Equivalente                                  |
|-----------------------------|--|
| bTest := {  nVar  nVar * 5} | Function SemNome( nVar )<br>Return(nVar * 5) |

# Programação AdvPl 1

## Funções de Bloco

**EVAL()** Executa um bloco de código qualquer, com passagem de parâmetros.

**Sintaxe:** uRet := Eval(bVar,Par1,Par2...Parx)

| Argumento       | Obrigat. | Tipo     | Descrição  |
|-----------------|----------|----------|--|
| BVar            | Sim      | Bloco    | Bloco a ser executado  |
| Par1,Par2..Parx | Não      | Diversos | Lista de argumentos que o bloco informado necessita receber.         |
| URet            |          |          | Retorno será o resultado da ultima expressão da lista de expressões. |

### Exemplos:

Criado um bloco de código com a instrução de multiplicar o valor do parâmetro passado através da variável nVar por 9.

```
bBloco := { |nVar| nVar * 10}  
nRet := Eval(bBloco,9) // Ao executar a função o valor da variável nRet será 90
```

Criado um bloco de código que irá multiplicar o valor da variável N1 por N2 e gravar o resultado na N3. Note que as 3 variáveis são criadas na execução do bloco de código, não são passadas por parâmetro.

```
bVar := {|| N1 := 6, N2 := 20, N3 := N1 * N2}  
nVar := Eval(bVar) // Após a execução da função o valor da variável nVar será 120
```

A execução do último bloco de código equivalente a função abaixo:

```
User Function fCalcula()  
N1:= 6  
N2:=20  
N3:= N1 * N2  
Return N3
```

Isto é, cada linha da função fCalcula é inserida no bloco de código, essas são separadas por "," (virgula), e "SEMPRE" o resultado da última expressão será retornada para a variável, no exemplo acima "N3 := N1 \* N2", o resultado dessa expressão será gravada da variável "nVar".

# Programação Advpl 1

**DBeVAL()** Executa um bloco de código enquanto obedecer a condição.

**Sintaxe:** dbEval(bVar,bFor,bWhile)

| Argumento | Obrigat. | Tipo  | Descrição  |
|-----------|----------|-------|--|
| BVar      | Sim      | Bloco | Bloco a ser executado  |
| BFor      | Não      | Bloco | Lista de argumentos que o bloco informado necessita receber.         |
| BWhile    | Não      | Bloco | Retorno será o resultado da ultima expressão da lista de expressões. |

**Exemplo:**

```
bAcao      := {|| aAdd(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})}
bCondicao := {|| X5_TABELA==cTabela}
dbEval(bAcao,,bCondicao)
```

**Equivalente a**

```
While !eof() .And. X5_TABELA==cTabela
    aAdd(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})
    dbSkip()
End
```

**AEval()** Executa um bloco de código, passando um array como parâmetro, a função percorre o array de acordo com os argumentos de inicio e quantidade de vezes do array.

**Sintaxe:** aEval(aArray,bBloco,nInicio,nCount)

| Argumento | Obrigat. | Tipo     | Descrição  |
|-----------|----------|----------|--|
| AArray    | Sim      | Array    | Array eu deverá ser percorrido   |
| BBloco    | sim      | Bloco    | Bloco com instruções a executar  |
| NInicio   | Não      | Numerico | Elemento que o bloco começará a percorrer, se não informado o default será 1                           |
| NCount    | Não      | Numerico | Numero de elementos que o bloco deve percorrer, se não informado, o default é ir até o final do array. |

**Exemplo:**

```
aNomes := {"Roberto","Joao","Ana"}
bAcao := {|X,Y| Msginfo(X+" "+CVALTOCHAR(Y)) }
aEval(aNomes, bAcao)
```

**Equivalente a**

```
aNomes := {"Roberto","Joao","Ana"}
For X:=1 to len(aNomes)
    Msginfo(aNomes [X])
Next
```

A variável X foi criada para guardar o elemento da matriz que está sendo lido no momento, tornando possível o dinamismo na apresentação da mensagem.

## Capítulo 12 – Funções Diversas

### Funções de Ambientes

#### **GetRemoteIniName()**

Retorna o nome do arquivo de configuração remota.

#### **GetEnvServer()**

Retorna o ambiente atual

#### **GetTheme()**

Retorna o tema escolhido.

#### **GetDBExtension()**

Retorna a extensão da Base (DBF, DTC, etc)

#### **Conout(cMens)**

Imprime cMens no console do Server

#### **OrdBagExt()**

Retorna a extensão do Índice(CDX, IDX, NTX, etc)

#### **Funname(n)**

Retorna o nome do fonte que executa a função na pilha, onde n é a posição na pilha.

Exemplo:

```
#include "protheus.ch"
User Function FuncServ()

cTemProc := "Menu: " + CARQMNU + CRLF //Nome do menu
cTemProc += "Modulo: " + cModulo+ CRLF // Modulo atual
cTemProc += "Ambiente: " + Getenvserver() + CRLF //Ambiente atual
cTemProc += "Usuario: " + cUserName + CRLF //Nome Usuario
cTemProc += "Programa: " + funname() + CRLF //Função chamada Menu
cTemProc += "Tema: " + GetTheme() + CRLF //Tema do usuário
cTemProc += "Extensão Base Local: " + GetDBExtension() + CRLF //Extensão LocalFiles
cTemProc += "Tipo Índice: " + OrdBagExt() + CRLF //Extensão de índices

conout(cTemProc)

Return Nil
```

# Programação Advpl 1

## Funções de Servidor

**GetsrvProfString()** Retorna o conteúdo do parâmetro da seção

**Sintaxe:** GetSrvprofString( cParam, cDefault)

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cParam    | Sim      | C    | Parâmetro da seção no arquivo INI   |
| cDefault  | Sim      | C    | Conteúdo Padrão, caso não seja encontrado o parâmetro "cParam" na seção do INI. |

**GetFuncArr()** Retorna um array com as funções do repositório do ambiente corrente

**Sintaxe:** aVar := GetFuncArr( cNomFun)

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| CNomFun   | Sim      | C    | Nome da função do repositório, se informado "A", retorna todas as funções do repositório. |

**GetFuncPrm()** Retorna parâmetros da função especificada

**Sintaxe:** aVar := GetFuncPrm( cFun)

| Argumento | Obrigat. | Tipo | Descrição                      |
|-----------|----------|------|--------------------------------|
| CFun      | Sim      | C    | Nome da função do repositório. |

**GetApoInfo()** Retorna um array com dados do programa informado dentro do repositório

**Sintaxe:** aVar := GetApoInfo( cNomProg)

| Argumento | Obrigat. | Tipo | Descrição        |
|-----------|----------|------|------------------|
| cNomProg  | Sim      | C    | Nome do programa |

Exemplo:

aDados := GetApoInfo( "MATA410.PRX" )

O array aDados terá as informações da rotina MATA410

- 1 Nome do arquivo
- 2 Codificada em ADVPL
- 3 Quem Compilou, TOTVS ou Cliente
- 4 Data da última Modificação.

## Funções de Conexão com o Servidor

### RpcClearEnv()

Limpa o ambiente

### RpcSetType()

Seleciona o Tipo o RPC

### RpcSetEnv()

repara o Ambiente para a empresa definida

**Sintaxe:** RpcSetEnv(cADEmp,cADFil,cUser,cSenha,cADMod,cFunName,aTables)

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cADEmp    | Sim      | C    | Código da Empresa   |
| cADFil    | Sim      | C    | Código da Filial  |
| cUser     | Não      | C    | Usuário do Protheus   |
| cSenha    | Não      | C    | Senha do Protheus   |
| cADMod    | Não      | C    | Sigla do módulo   |
| cFunName  | Não      | C    | Nome da rotina que será setada pelo retorno da função funname() |
| aTables   | Não      | A    | Array com os alias necessários                                  |

Exemplo:

User Function PrepEnv()

```
Local cEmp    := "99"
Local cFilial := "01"
Local cUser   := "Administrador"
Local cSenha  := "123456"
Local aTabelas := {"SA1", "SA2", "SA3"}
```

```
RpcClearEnv()
RpcsetType(3)
RpcSetEnv( cEmp , cFilial , cUser , cSenha , "FAT" , , aTabelas )
```

Return Nil

**Pode ser feito de outra forma, também chamado “forma equivalente”**

```
#INCLUDE "PROTHEUS.CH"
#INCLUDE "TBICONN.CH"
```

User Function PrepEnv()

```
Local cEmp    := "99"
Local cFilial := "01"
Local cUser   := "Administrador"
Local cSenha  := "123456"
```

```
RpcClearEnv()
RpcsetType(3)
```

```
PREPARE ENVIRONMENT EMPRESA cEmp FILIAL cFilial USER cUser PASSWORD cSenha ;
```

```
MODULO 'FAT' TABLES "SA1", "SA2", "SA3"
```

Return

# Programação Advpl 1

## Funções de Comunicação Server x Client

### CpyS2T() Copia arquivos do servidor para a máquina do cliente

**Sintaxe:** CPYS2T(cArqOri, cArqDes, lComp)

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| cArqOri   | Sim      | C    | Arquivo de Origem, visível a partir do rootpath do ambiente corrente |
| cArqDest  | Sim      | C    | Arquivo de Destino, visível na máquina do cliente                    |
| lComp     | Não      | L    | Define se compacta antes de enviar o arquivo.<br>Default .T.         |

Exemplo:

```
#include "protheus.ch"
User Function CopiaArq()
Local cArqOri := "\SYSTEM\SIGAFIN.XNU"
Local cArqDes := "C:\TEMP"

If CPYS2T(cArqOri, cArqDes, .T.)
    MSGINFO("Arquivo " + cArqOri + " copiado com sucesso para " + cArqDes, "COPIAARQ")
Else
    MSGINFO("Arquivo " + cArqOri + " não foi copiado! ", "COPIAARQ")
EndIf

Return Nil
```

### CpyT2S() Copia arquivos da máquina do cliente para o servidor

**Sintaxe:** CPYT2S(cArqOri, cArqDes, lComp)

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cArqOri   | Sim      | C    | Arquivo de Origem, visível na máquina do cliente                      |
| cArqDest  | Sim      | C    | Arquivo de Destino, visível a partir do rootpath do ambiente corrente |
| lComp     | Não      | L    | Define se compacta antes de enviar o arquivo.<br>Default .T.          |

Exemplo

```
#include "protheus.ch"
User Function CopiaArq()
Local cArqOri := "C:\TEMP\SIGAFIN.XNU"
Local cArqDes := "\SYSTEM"
If CPYT2S(cArqOri, cArqDes, .T.)
    MSGINFO("Arquivo " + cArqOri + " copiado com sucesso para " + cArqDes, "COPIAARQ")
Else
    MSGINFO("Arquivo " + cArqOri + " não foi copiado! ", "COPIAARQ")
EndIf
Return
```

## Capítulo 13 – Customização

Chamamos de customização tudo aquilo que fazemos no Protheus que foge do padrão, quando o padrão do Sistema não atende. Podemos customizar diversas situações:

- Parâmetros;
- Tabelas;
- Perguntas;
- Fórmulas - Expressões em AdvPL / User Function;
- Lançamentos Padrões via Expressões em AdvPL / User Function;
- Validações;
- Gatilhos;
- Campos de Arquivos;
- User Function via menu;
- Pontos de Entrada;
- Dicionário de Dados Ativo;
- SigaRPM;
- Crystal;
- Integração Office (Word , Excel).

### Parâmetros (SX6)

Podemos criar parâmetros específicos para utilizar durante o processamento no sistema. Os parâmetros podem ser criados pelo configurador.

Assim como uma variável o Parâmetro armazena um conteúdo chave na execução de processamentos padrão ou customizados, sendo possível criá-los ou alterá-los conforme as necessidades das rotinas.

Por padrão um parâmetro recebe uma nomenclatura especial para tais variáveis, é iniciado pelo prefixo "MV\_", é possível alterar estas informações sempre que necessário contratando o administrador do sistema.

É importante estabelecermos um padrão na criação de novos parâmetros, para não serem confundidos com parâmetros padrões, apesar do sistema gravar o campo X6\_PROPRI com "U", pode ocorrer uma coincidência na criação de parâmetros com nome igual a um parâmetro criado numa versão seguinte. Uma sugestão é colocar as iniciais da empresa seguida de números.

Existem funções que nos permite manipular os parâmetros.

# Programação AdvPl 1

## Procedimentos

No módulo Configurador acesse o menu “base de dados → dicionário → base de dados”, selecione a empresa desejada e clique na opção ‘parâmetros’, será apresentada a tela:

The screenshot shows the Protheus 11 Configuration interface with the 'Parametros' tab selected in the 'Gerenciador de Base de Dados'. The main window displays a grid of parameters with columns for Filial, Nome, Descrição, and various notes. The notes column contains detailed descriptions and conditions for each parameter, such as sequence generation rules and specific business logic. The interface includes standard Windows-style buttons and toolbars at the top.

Aqui nos podemos realizar a manutenção no cadastro de parâmetros.

Para visualizar melhor segue um exemplo, temos o parâmetro:

| Nome do Parâmetro | Descrição   | Conteúdo   |
|-------------------|---|------------|
| MV_ALIQUSS        | Define o percentual da alíquota para cálculo de ISS | Valor em % |

Clique no botão ‘pesquisar’ , informe o nome do parâmetro ‘MV\_ALIQUSS’ e clique em ‘Procurar’, na seqüência clique em ‘editar’ , sera apresentado a tela:

The dialog box titled 'Editar Parametro - MV\_ALIQUSS' has two tabs: 'Informações' and 'Descrição'. The 'Informações' tab is active, showing fields for Filial (selected as 1), Nome da Var. (set to MV\_ALIQUSS), Tipo (set to Numérico), and three Cont. fields (Por, Ing, Esp, all set to 5). At the bottom are buttons for 'Confirmar', 'Fechar', and 'Ações relacionadas'.

# Programação Advpl 1

Você pode alterar os campos conteúdo (nos três idiomas), a cada vez que a alíquota for alterada, não serão necessárias customizações no sistema, e sim a alteração deste.

Os parâmetros podem ser diferentes a cada filial da empresa, para isso é necessário entrar com a filial no campo FILIAL.

Para inclusão de um Parâmetro clique no botão 'incluir'  , será apresentado a tela abaixo, informe os dados conforme instrução:



[Filial]

Determina a filial a qual será armazenado o parâmetro, deixando em branco, essa será compartilhada entre todas as filiais

[Nome da Var]

Nome do parâmetro seguindo o padrão de nomenclatura.

[Tipo]

Este campo define o tipo de conteúdo registrado nos campos abaixo, podendo ser definidos por caractere , numérico , lógico , data e memo.

[Cont. Por]

Representa o valor da variável quando a linguagem do sistema for Portugues

[Cont. Ing / Cont. Esp]

Conteúdo do campo em Inglês / Espanhol respectivamente

## Alteração de um Parâmetro:

Clique na opção  para que o parâmetro possa ser alterado.

## Exclusão de um Parâmetro:

Clique na opção  para que o parâmetro possa ser excluído.

# Programação AdvPL 1

## Funções para tratamento de parâmetros em ADVPL

|   |
|---|
| <b>GETMV()</b> Retorna o conteúdo de um parâmetro cadastrado no SX6 |
|---|

**Sintaxe:** GETMV( *cNomPar*, [ *IRetPar* ], [ *uRet* ] )

| Argumento      | Obrig | Tipo | Descrição  |
|----------------|-------|------|--|
| <i>cNomPar</i> | Sim   | C    | Nome do parâmetro a ser pesquisado   |
| <i>IRetPar</i> | Não   | L    | * Se <i>IRetPar</i> for .F. (default se não informado) a função GETMV() irá retornar o conteúdo do parâmetro informado.<br>* Se <i>IRetPar</i> for .T., a função GETMV() irá retornar um valor lógico, informando se o parâmetro <i>cNomPar</i> existe .T. (verdadeiro) ou .F. (FALSO) se não existe |
| <i>uRet</i>    | Não   | U    | Valor que deve ser retornado se o parâmetro solicitado não existir. O valor desse parâmetro pode ser do mesmo tipo do <i>cNomPar</i> , isto é, caracter, numérico, lógico ou data.   |

**Exemplo:**

Grava na variável *cEmail* o conteúdo do parametro informado

*cEmail* := GETMV("MV\_XEMAIL")

Grava na variável *lExiste* .T. se o parâmetro MV\_XEMAIL existe na SX6 ou .F. caso esse não exista

*lExiste* := GETMV("MV\_XEMAIL",.T.)

Grava na variável *cEmail* o conteúdo "email@empresa.com.br" caso não existir na SX6 o parâmetro MV\_XEMAIL, isso assegura que a variável *cEmail* sempre terá informação

*cEmail* := GETMV("MV\_XEMAIL", , "email@empresa.com.br")

|   |
|---|
| <b>SUPERGETMV()</b> Retorna o conteúdo de um parâmetro cadastrado no SX6 de acordo com a filial informada |
|---|

**Sintaxe:** SUPERGETMV( *cNomPar*, [ *lHelp* ], [ *uRet* ] , [ *cFil* ] )

| Argumento      | Obrig | Tipo | Descrição  |
|----------------|-------|------|--|
| <i>cNomPar</i> | Sim   | C    | Nome do parâmetro a ser pesquisado   |
| <i>IRetPar</i> | Não   | L    | * Se <i>IRetPar</i> for .F. (default se não informado) a função GETMV() irá retornar o conteúdo do parâmetro informado.<br>* Se <i>IRetPar</i> for .T., a função GETMV() irá retornar um valor lógico, informando se o parâmetro <i>cNomPar</i> existe .T. (verdadeiro) ou .F. (FALSO) se não existe |
| <i>uRet</i>    | Não   | U    | Valor que deve ser retornado se o parâmetro solicitado não existir. O valor desse parâmetro pode ser do mesmo tipo do <i>cNomPar</i> , isto é, caracter, numérico, lógico ou data.   |
| <i>cFil</i>    | Não   | C    | Retorna parâmetro de acordo com a Filial, Default Filial atual. Se houver parâmetro para a filial, ela será priorizada.  |

**Exemplo:**

*CEmail\_SP* := GETMV("MV\_XEMAIL" , , "01" )

*CEmail\_RJ* := GETMV("MV\_XEMAIL" , , "02" )

# Programação Advpl 1

## PUTMV() Grava informação em parâmetros cadastrados na tabela SX6

**Sintaxe:** PUTMV( *cNomPar*, *cConteudo* )

| Arguento         | Obrig | Tipo | Descrição   |
|------------------|-------|------|---|
| <i>cNomPar</i>   | Sim   | C    | Nome do parâmetro a ser pesquisado  |
| <i>cConteudo</i> | Sim   | U    | Conteúdo a ser gravado no parâmetro, deve ter o tipo esperado no parâmetro. |

**Exemplo:**

PUTMV( "MV\_XEMAIL" , "filial\_saopaulo@empresa.com.br" )

O conteúdo do paramtro MV\_XEMAIL foi alterado de "email@empresa.com.br"  
para "filial\_saopaulo@emprsa.com.br"

## PUTMVFIL() Grava informação no parâmetro no SX6 de acordo com a filial

**Sintaxe:**

PUTMVFIL( *cNomPar*, *cConteudo*, *cFil* )

| Arguento         | Obrig | Tipo | Descrição   |
|------------------|-------|------|---|
| <i>cNomPar</i>   | Sim   | C    | Nome do parâmetro a ser pesquisado  |
| <i>cConteudo</i> | Sim   | U    | Conteúdo a ser gravado no parâmetro, deve ter o tipo esperado no parâmetro. |
| <i>cFil</i>      | Não   | C    | Filial do parâmetro.  |

**Exemplo:**

PUTMVFIL( "MV\_XEMAIL" , "filial\_saopaulo@empresa.com.br", "01" )

PUTMVFIL( "MV\_XEMAIL" , "filial\_rio@empresa.com.br", "02" )

## Tabelas Genéricas (SX5)

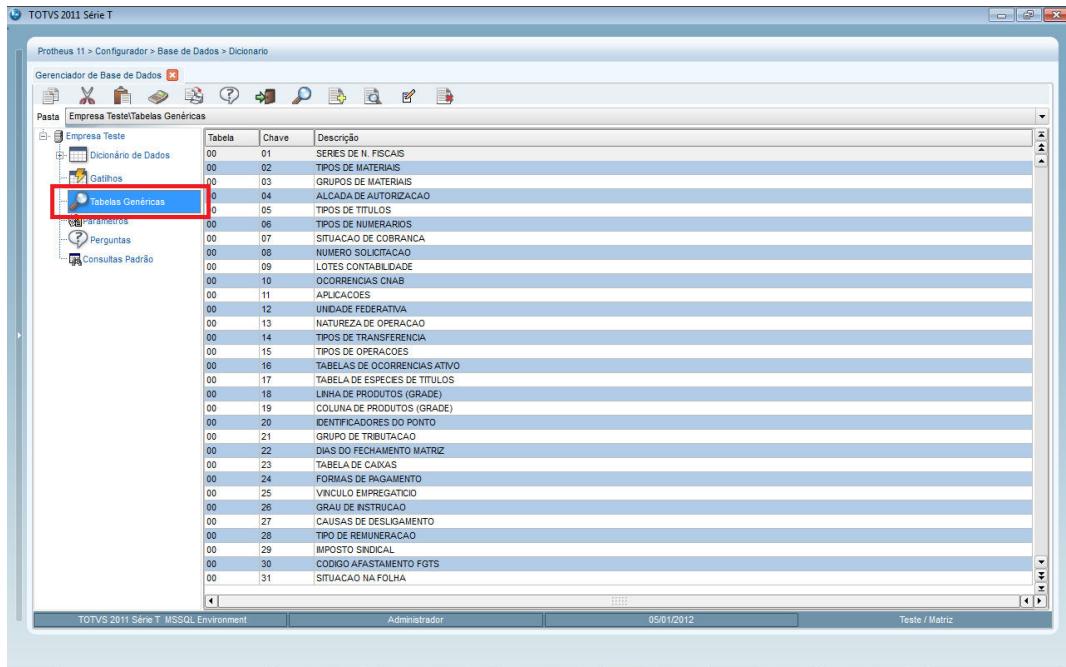
Podemos criar tabelas específicas para utilizar durante o processamento, fazer pesquisas e amarrações no sistema. As tabelas podem ser criadas pelo configurador.

A criação de tabelas deve ser feita quando temos alguma informação padronizada que contenha código e descrição, a Totvs reserva as tabelas iniciadas em Z para uso de clientes.

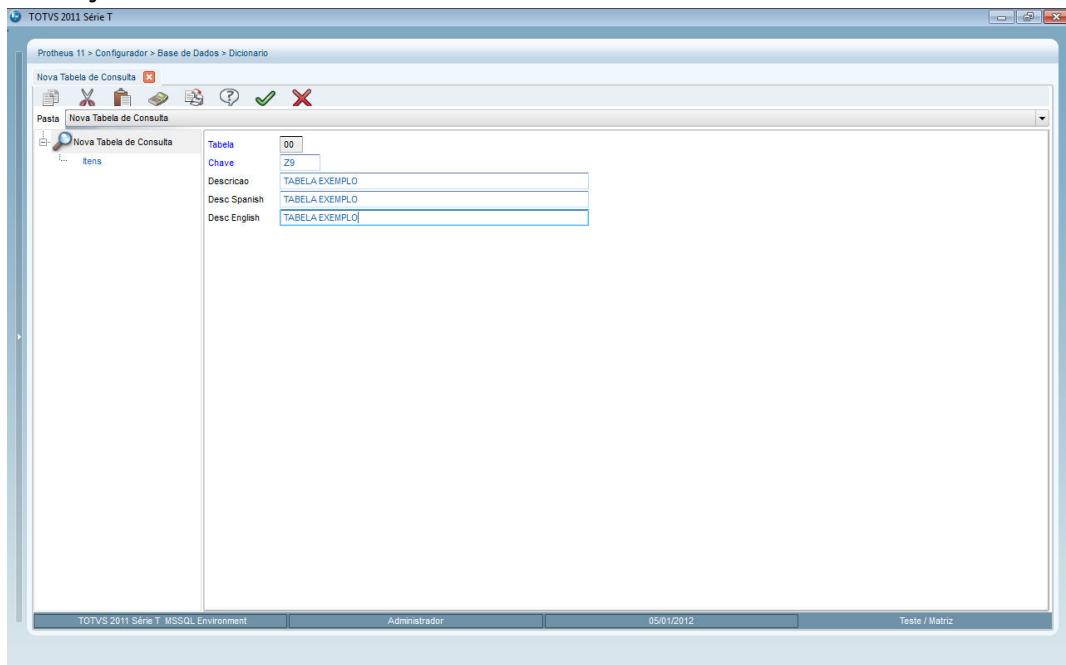
### Criando Tabelas Genéricas

No módulo Configurador acesse o menu "base de dados → dicionário → base de dados", selecione a empresa desejada e clique na opção 'tabelas genéricas', será apresentada a tela:

# Programação AdvPl 1



Clique no botão 'incluir' e cadastre a identificação da tabela genérica conforme instruções:



## [Tabela]

Tabela Genérica '00' que guarda a identificação de todas as tabelas genéricas do sistema.

## [Chave]

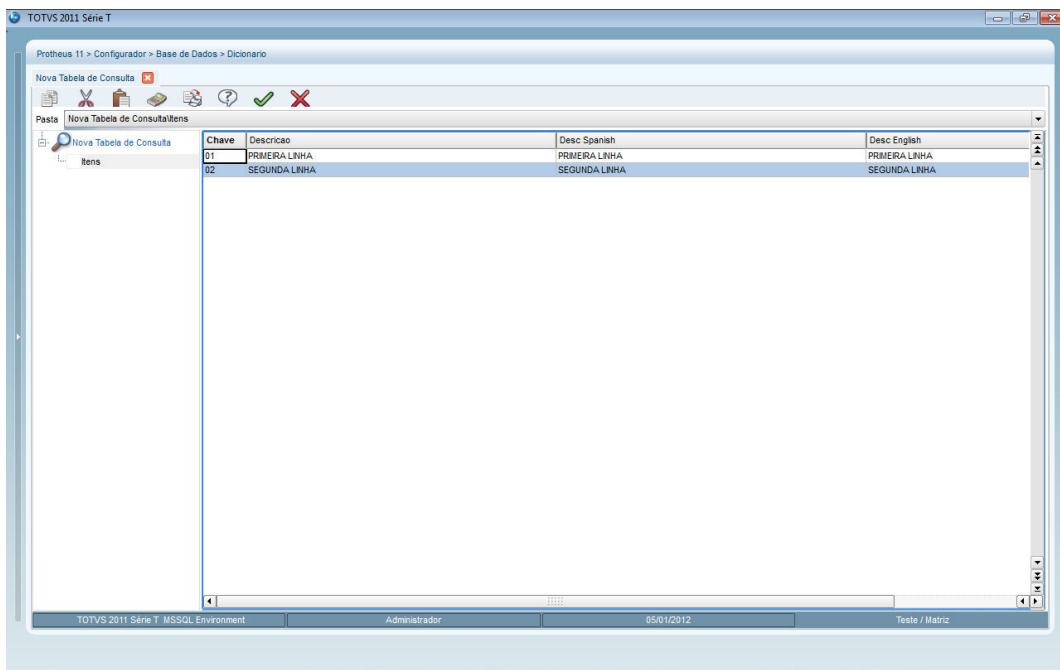
Numero da nova tabela a ser criada.

## [Descrição/Desc. Spanish/ Desc. English]

Descrição da tabela genérica nos três idiomas.

# Programação Advpl 1

Em seguida, clique em 'Itens' e adicione o conteúdo 'linhas' da nova tabela genérica conforme tela:



[Chave]

Identificação da Linha/Registro que esta sendo inserida.

[Descrição/Desc. Spanish/ Desc. English]

Descrição da Linha/Registro nos três idiomas.

# Programação AdvPl 1

## Função para acesso às Tabelas Genéricas

**TABELA()** Retorna o conteúdo de uma tabela cadastrada no SX5

**Sintaxe:** TABELA( *cCodTab*, *cChave*, [*lHelp*] )

| Arguento       | Obrig | Tipo | Descrição  |
|----------------|-------|------|--|
| <i>cCodTab</i> | Sim   | C    | Código da Tabela a pesquisar   |
| <i>cChave</i>  | Sim   | C    | Chave para pesquisa da tabela  |
| <i>lHelp</i>   | Não   | L    | Se .T. irá mostrar mensagem informando se o conteúdo pesquisado não foi encontrado, se .F. não mostra mensagem caso o conteúdo não seja encontrado |

### Exemplo:

`cVar := TABELA("02","PA",.F.)`

Nesse exemplo, caso a função encontre a chave "PA" na tabela "02" (Tipos de materiais), o conteúdo localizado será gravado na variável *cVar*. Caso não encontre, grava conteúdo em branco.

## Exercício

Crie uma tabela genérica conforme a orientação do Instrutor, e tenha as seguintes características:

### Tabela : Z6

01 - ABRIL  
02 - DIGERATI  
03 - FTD  
04 - FUTURA  
05 - GLOBAL

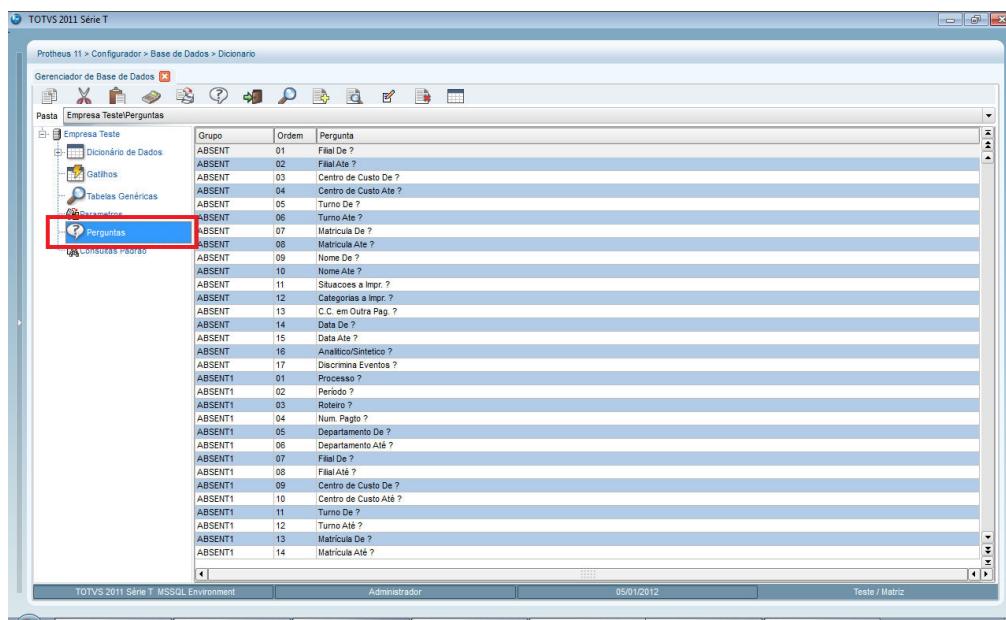
## Perguntas (SX1)

No Protheus a Tabela SX1 agrupa os dados das perguntas que são apresentadas nas Telas de Parâmetros dos relatórios e de processos do sistema.

Quando houver a necessidade de customizar algum relatório onde houver parametrizações, será necessário inserir/configurar as perguntas nesta tabela através de rotina própria do Cadastro de Perguntas do Microsiga Protheus.

## Procedimentos

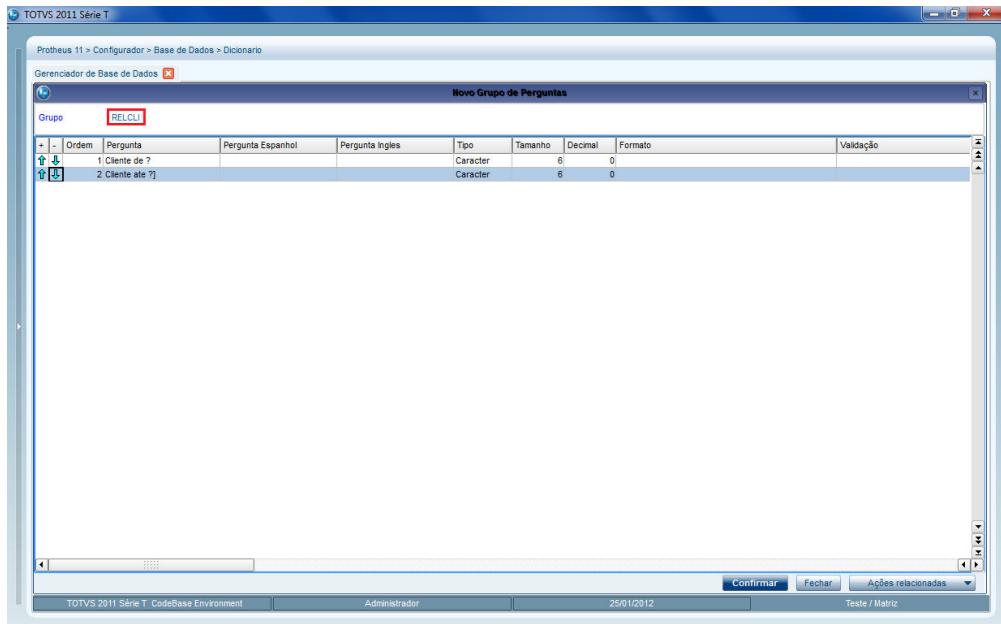
No módulo Configurador acesse o menu “**base de dados → dicionário → base de dados**”, escolha a empresa desejada e clique na opção ‘perguntas’, será apresentada a tela abaixo, contendo as perguntas criadas pela Microsiga:



Para incluir um novo grupo de perguntas, clique no botão ‘incluir’  , e siga os procedimentos:

As perguntas existentes poderão sofrer alterações, porém é aconselhável evitar alterações de grande impacto nas perguntas padrão do sistema.

# Programação AdvPl 1



## [Grupo]

Informe um nome para o conjunto de perguntas em cadastro.

## [Ordem]

Este campo é definido pelo sistema, e registra a ordem de apresentação das perguntas na tela.

## [Pergunta]

Informe a descrição da pergunta desejada que será apresentada ao usuário.

## [Pergunta Espanhol/Pergunta Inglês]

Informe a pergunta desejada traduzida para o espanhol e para o inglês , respectivamente.

## [Tipo]

Selecione o tipo do conteúdo de resposta da pergunta, se 'Caracter', 'Numérico' ou 'Data'.

## [Tamanho]

Informe o tamanho do campo, **exatamente como está no SX3**, para a resposta da pergunta. EX: SA1\_COD (Tamanho = 6)

## [Decimal]

Informe o número de casas decimais da resposta, se houver.

# Programação Advpl 1

---

## [Formato]

Informe o formato, podendo usar os pictures, estudadas no SX3 nessa apostila.

Ex: @! – todas as informações digitadas ficarão **MAIUSCULAS**

## [Validação]

Informe uma função ADVPL, pertinente a esta pergunta (abaixo exemplo).

## [Help]

Texto de explicativo da pergunta em questão

## [Objeto]

Selecione como a pergunta será apresentada na tela, se:

**Edit:** permite editar o conteúdo expresso no campo;

**Text:** formato de texto, que não permite alteração;

**Combo:** permite definir um alista de opções para o campo;

**Range:** permite o uso de dados fora de sequencia ou seqüenciais;

**File:** permite a seleção de um arquivo no servidor ou no ambiente local;

**Expression:** possibilita a utilização de um filtro;

**Check:** possibilita a seleção de até 5 Itens de campo check.

## [Consulta Padrão]

Caso o ‘Objeto’ tenha sido definido como ‘Edit’ pode ser inserido a referencia da “consulta padrão”.

## [Conteúdo]

Caso o ‘Objeto’ tenha sido definido como ‘Text’, informe o conteúdo do campo, que não poderá ser alterado.

## [Pré-seleção ]

Caso o ‘Objeto’ tenha sido definido como ‘Combo’, podemos inserir uma lista de itens para o usuário escolher, o campo de pré-seleção permite que seja sugerido ao usuário a seleção de um desses itens.

## [Item 1 a 5]

Informe nestes campos os itens que serão apresentados na tela como opção de preenchimento da pergunta, quando o campo "Objeto" for preenchido com ‘Combo’.

Caso não haja quebra de seqüência. pode haver até 5 itens para seleção.

# Programação Advpl 1

## Função para acessar os grupos de “perguntas” usando o ADVPL

**PERGUNTE()** Carrega e Monta a tela de perguntas cadastradas no SX1

### Sintaxe:

PERGUNTE( *cGrupo*, *lMostra* )

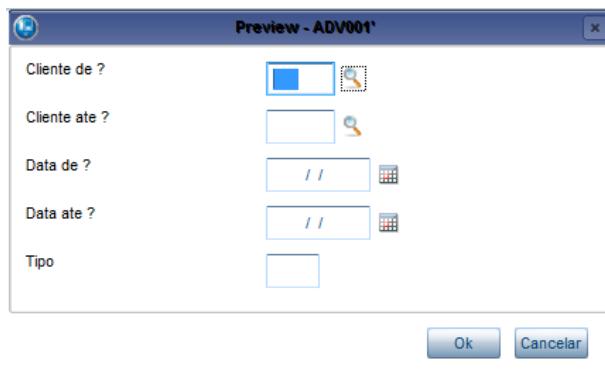
| Arguento       | Obrig | Tipo | Descrição   |
|----------------|-------|------|---|
| <i>cGrupo</i>  | Sim   | C    | Grupo de perguntas  |
| <i>lMostra</i> | Sim   | C    | .T. – Carrega e mostra tela com perguntas, .F. apenas carrega respostas das perguntas |

### Exemplo

*cPerg* := "ADV001" // Nome do grupo de perguntas (cadastrado na SX1)

PERGUNTE( *cPerg* )

Será apresentado ao usuário a tela de perguntas informada:



Para cada pergunta será criado uma variável com sua resposta, essa iniciando em **MV\_PAR**, no exemplo acima teremos as variáveis:

MV\_PAR01  
MV\_PAR02  
MV\_PAR03  
MV\_PAR04  
MV\_PAR05

# Programação AdvPl 1

## Criando função ADVPL para validar respostas do usuário no grupo de perguntas acima

| Grupo ADV00 |       |               |                   |                 |          |         |         |         |              |
|-------------|-------|---------------|-------------------|-----------------|----------|---------|---------|---------|--------------|
|             | Ordem | Pergunta      | Pergunta Espanhol | Pergunta Ingles | Tipo     | Tamanho | Decimal | Formato | Validação    |
| 1           | 1     | Cliente de ?  |                   |                 | Caracter | 6       | 0       |         |              |
| 2           | 2     | Cliente ate ? |                   |                 | Caracter | 6       | 0       |         |              |
| 3           | 3     | Data de ?     |                   |                 | Data     | 8       | 0       |         |              |
| 4           | 4     | Data ate ?    |                   |                 | Data     | 8       | 0       |         |              |
| 5           | 5     | Tipo          |                   |                 | Caracter | 1       | 0       |         | u_VdADV001() |

Observe que foi inserido uma função de usuário no campo “validação” da pergunta “Tipo”, a seguir veja o conteúdo da função VdADV001 e o resultado de sua execução:

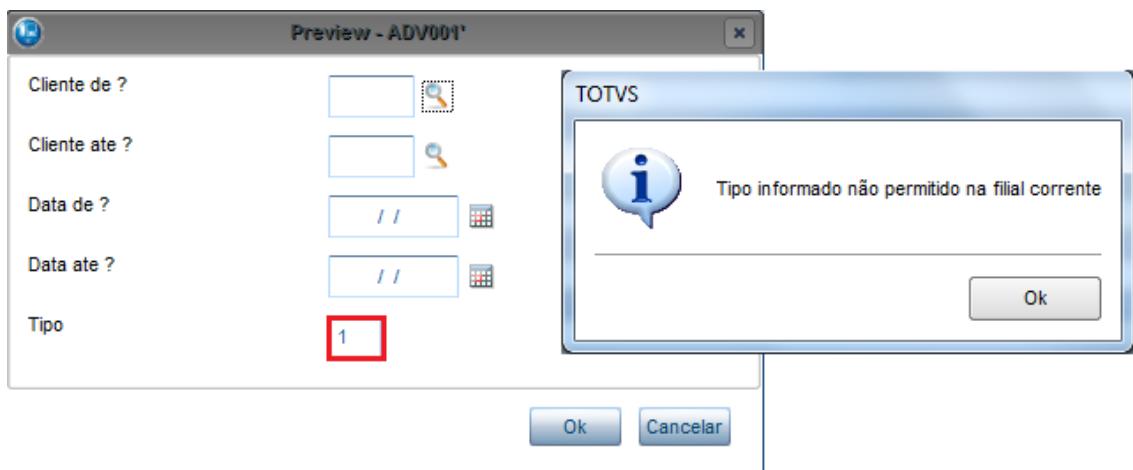
The screenshot shows the Development Studio interface with the title bar "Development Studio - [VALIDPERGUNTA.PRW]". The menu bar includes Arquivo, Editar, Inserir, Visualizar, Projetos, Executar, Ferramentas, Autorização, Janela, and Ajuda. The toolbar has various icons for file operations. The code editor window contains the following PFC (Protheus Function Call) code:

```
#INCLUDE "PROTHEUS.CH"

// EXEMPLO VALIDACAO DA PERGUNTA "TIPO" DO GRUPO
// DE PERGUNTAS "ADV001"
//
// SIMULA UMA SITUACAO EM QUE A RESPOSTA DA PERGUNTA "TIPO"
// NAO PODERA SER "1" CASO A FILIAL CORRENTE FOR "01"
//

USER FUNCTION VdAdv001()
IF CFILANT=="01"
  IF MV_PAR05=="1"
    MsgInfo("Tipo informado nmo permitido na filial corrente")
    RETURN .F.
  ENDIF
ENDIF
RETURN .T.
```

The project tree on the left shows a project named "Treinamento ADVPL" with a sub-project "ADVPL 1" containing "Fontes" (Sources) with files "\Projeto\PrimProg.prw" and "\Projeto\ValidPergunta.PRW", and "Definições" (Definitions).



# Programação Advpl 1

---

## Exercício

Crie um “grupo de perguntas” contendo as perguntas:

| Nome do grupo | ADV001 |         |                 |
|---------------|--------|---------|-----------------|
| Pergunta      | Tipo   | Tamanho | Consulta Padrão |
| Produto de    | C      | 15      | SB1             |
| Produto ate   | C      | 15      | SB1             |
| Tipo          | D      | 8       | 02              |

Na pergunta “TIPO” insira uma validação onde somente os tipo “PA” e “PI” poderão ser utilizados, lembrando que a resposta dessa pergunta estará na variável MV\_PAR03.

Na sequencia crie uma função em ADVPL fazendo uso do grupo de perguntas recém cadastrado, utilize a função PERGUNTE() e mostre na tela as respostas do usuário com a função MSGINFO().

## Expressões ADVPL

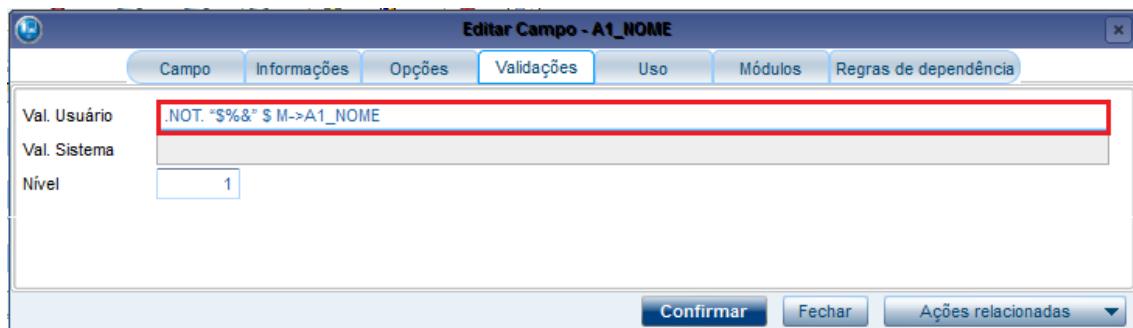
Podemos colocar qualquer “expressão em ADVPL” ou uma “User Function” em vários lugares no Protheus, dentre eles:

- Lançamentos padrões (CT5)
- Fórmulas (SM4)
- Validações na (SX3) campo X3\_VLDUSER
- Inicializador padrão na (SX3) campo X3\_RELACAO
- Validação nas perguntas (SX1) campo X1\_VALID

Para executar essas instruções é utilizado o recurso de macrosubstituição.

### Exemplo:

Utilizando expressão ADVPL no campo X3\_VALID para impedir que usuário digite os caracteres especiais "\$%&" no campo A1\_NOME:



Nesse exemplo, caso usuário digite algum desses caracteres no campo em questão “Nome do cliente”, o mesmo não conseguirá sair do campo, isto é, somente conseguirá ir para o próximo campo caso “**NÃO**” tenha digitado os caracteres especiais da validação.

Ainda utilizando esse exemplo, podemos usar uma função de usuário em lugar de uma expressão, podemos digitar no campo VAL.USUARIO acima a função de usuário U\_VDNOMCLI(), segue seu conteúdo para entendimento:

```
USER FUNCTION VDNomCli()
IF "$%&" $ M->A1_NOMCLI
    MsgInfo("Dados digitados $%& são invalidos !!!")
    RETURN .F.
ENDIF
RETURN .T.
```

Caso os caracteres especiais citados acima estejam na variável M->A1\_NOMECLI a função de validação informa ao usuário o problema e retorna .F., assim o usuário não poderá ir para o próximo campo.

## Pontos de Entrada

São aberturas nas rotinas padrões do sistema que deixa o sistema flexível permitindo o desenvolvimento de processos específicos a partir de uma rotina padrão do sistema, de acordo com a necessidade do cliente e dos analistas de campo, permitindo maior abrangência nos diversos segmentos.

As funções a seguir são usadas nos fontes da Microsiga para verificar se os pontos de entrada estão compilados no repositório de objetos.

**EXISTBLOCK()** - verifica a existência de uma função de usuário compilada no repositório de objetos da aplicação Protheus. Esta função é normalmente utilizada nas rotinas padrões da aplicação Protheus para determinar a existência de um ponto de entrada e permitir sua posterior execução.

**Sintaxe:** EXISTBLOCK(cFun)

| Arguento | Obrigat. | Tipo | Descrição                                |
|----------|----------|------|--|
| cFun     | Sim      | B    | cFunção Nome da função que será avaliada |
| uparam   | Não      | U    | Paarametros a enviar para a função       |

Retorno Lógico: .T. Existe a Função no repositório e .F. Não existe a função no repositório

**EXECBLOCK()** - Executa uma função de usuário que esteja compilada no repositório. Esta função é normalmente utilizada pelas rotinas padrões da aplicação Protheus para executar pontos de entrada durante seu processamento.

A função de usuário executada através da EXECBLOCK() não recebe parâmetros diretamente, sendo que estes estarão disponíveis em uma variável private denominada PARAMIXB.

A variável PARAMIXB é o reflexo do parâmetro xParam, definido na execução da função EXECBLOCK(). Caso seja necessária a passagem de várias informações, as mesmas deverão ser definidas na forma de um array, tornando PARAMIXB um array também, a ser tratado na função de usuário que será executada

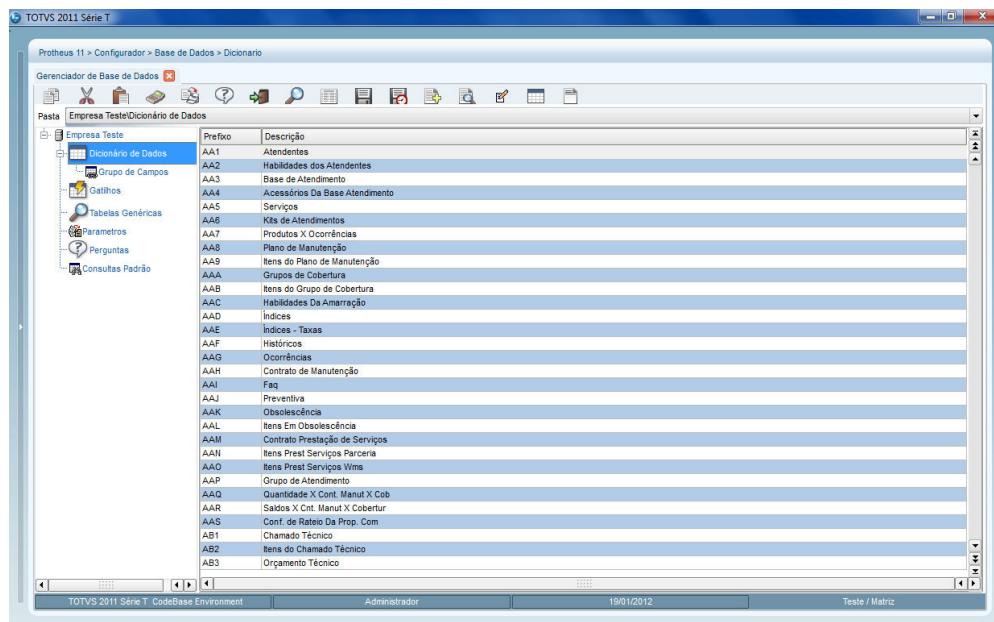
**Sintaxe:** ExecBlock(cblock, , , uparam)

| Arguento | Obrigat. | Tipo | Descrição              |
|----------|----------|------|------------------------|
| cblock   | Sim      | B    | Nome da Função Padrão  |
| uparam   | Não      |      | Parametros para o P.E. |

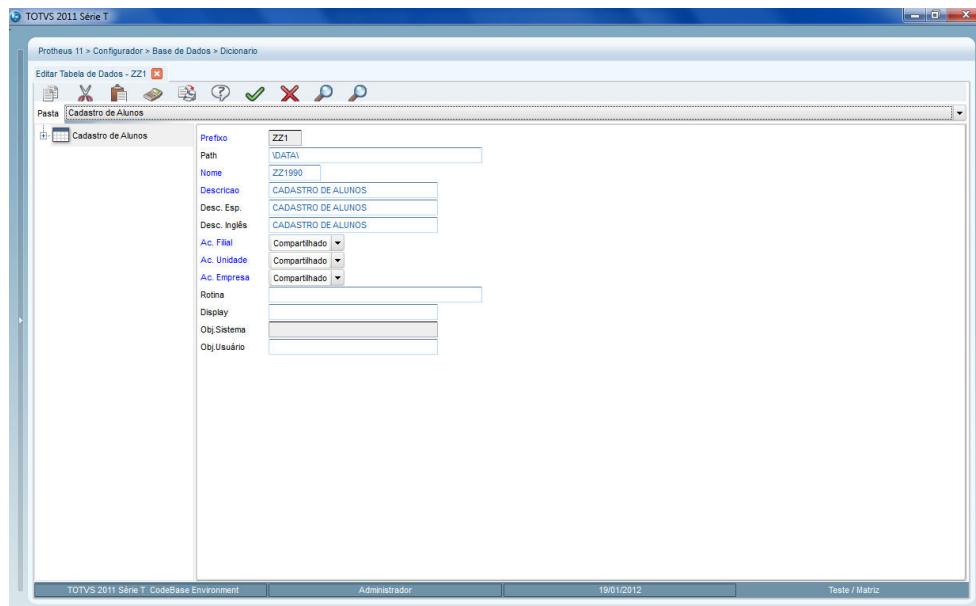
# Programação AdvPl 1

## Dicionário de Dados Ativos (SX2 e SX3)

No módulo Configurador (SIGACFG) acesse o menu “base de dados → dicionário → arquivo” ou “base de dados → dicionário → base de dados”, escolha a empresa na qual você quer criar a tabela e clique em “dicionário de dados”, será apresentada tela seguir:



Clique no botão ‘incluir’ , preencha os dados pertinentes e clique em “Ok”, você estará salvando os dados na tabela SX2.



# Programação Advpl 1

A seguir, descrição dos principais campos:

## [Prefixo]

Alias (apelido) da tabela que será criada, importante utilizar nomenclaturas estipuladas pela Microsiga para criação de tabelas customizadas, assim não correrá o risco de sua tabela ser substituída nas próximas atualizações.

Essa nomenclatura tem a seguinte composição:

SZ? – Onde “?” representa seqüência de números entre 0 à 9 e letras entre A à Z.

Z?? – Onde “??” representa seqüência similar a anterior, mas nesse caso para ambas as posições.

## [Path]

Localização física da tabela, quando se trabalha com padrão de banco de dados DBF ou CTREE, por padrão esses serão gravados na pasta \DATA\ abaixo do ROOTPATH.

## [Nome]

Nome real da tabela, no exemplo o nome será ZZ1 (prefixo definido acima) + 99 (número da empresa corrente) + 0 (padrão Siga).

## [Descrição]

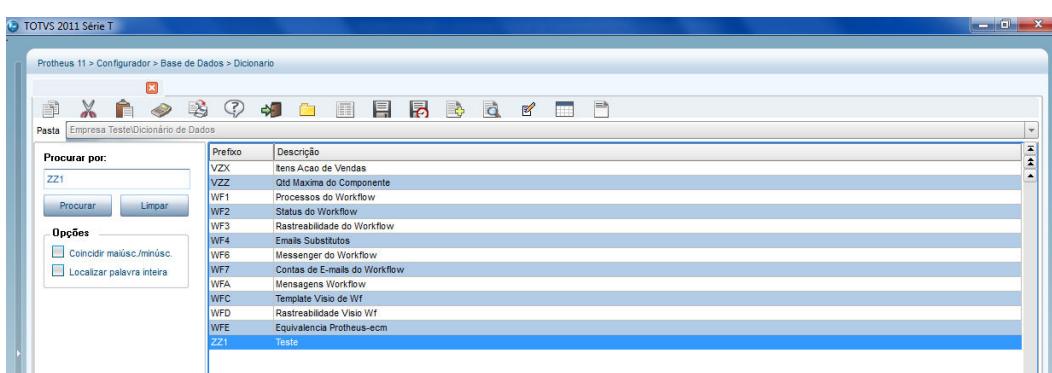
Descrição da tabelas nos três idiomas.

## [Ac Filial □ Ac Unidade □ Ac Empresa]

Campos que controlam o compartilhamento ou exclusividade dos dados, entre filial, unidade de negócio e empresa respectivamente.

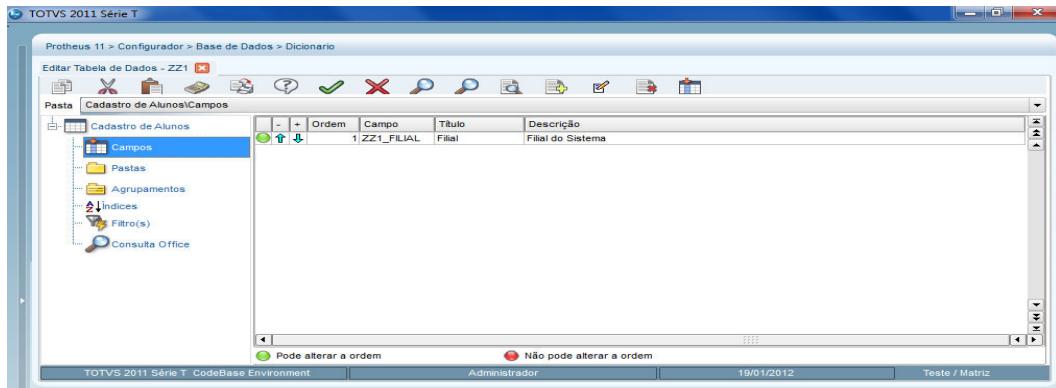
Clique em 'ok'  para salvar as informações referente a nova tabela, em seguida vamos incluir seus campos.

Para incluir os campos da nossa tabela, clique em ‘pesquisa’  e informe o nome da tabela no campo “procurar por” e clique no botão “procurar”, conforme tela abaixo:



Localizada a tabela, clique em ‘editar’ , em seguida expanda a estrutura conforme a tela:

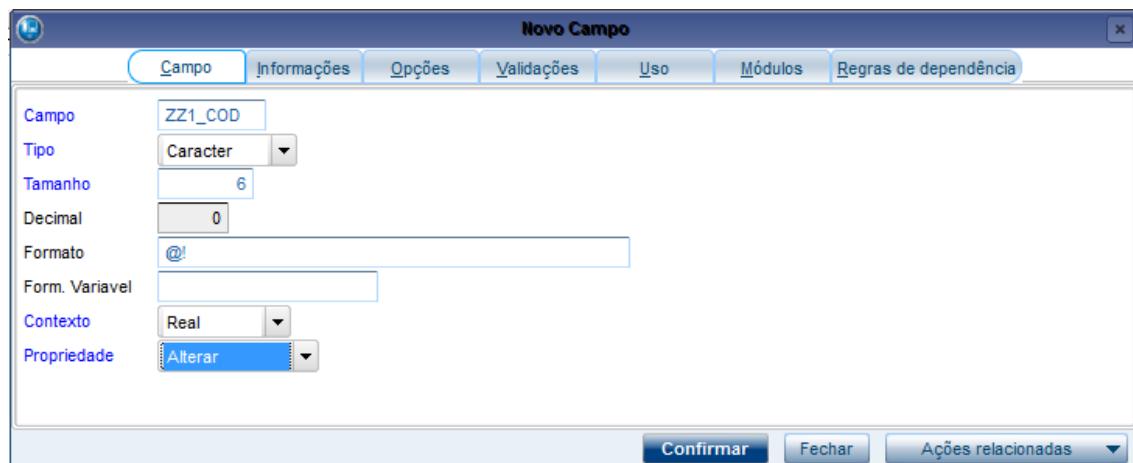
# Programação AdvPl 1



Note que o campo filial é criado automaticamente, não altere sua estrutura.

Para inclusão de novos campos, clique em 'incluir' , será apresentado uma tela com diversas "abas", a seguir trataremos as principais características de cada uma.

Como exemplo, criaremos o campo "Código do Aluno", preencha os campos conforme tela:



## [Campo]

Nome 'técnico' do campo, sempre deve ser precedido pelos dois ou três primeiros caracteres do nome da tabela, dependendo da regra:

- Se nome da tabela inicia com o caractere "S", por exemplo SZ1, o nome do campo deve ser os dois últimos caracteres, ficando Z1\_CAMPO.
- Se nome da tabela não inicia com o caractere "S", por exemplo ZZ1, o nome do campo deve ser os três caracteres, ficando ZZ1\_CAMPO

## [Tipo]

Define o Tipo de dados que serão aceitos no Protheus.

- Caractere: Permite a entrada de dados do tipo caracter, isto é, letras, números e símbolos especiais.
- Numérico: Permite a entrada de dados do tipo numérico, isto é, números e sinais de positivo (+), negativo (-) e separadores decimais.
- Lógico: Permite a entrada de dados do tipo lógico, isto é, formato verdadeiro/falso ou sim/não.
- Data: Permite a entrada de dados do tipo data, ou seja, DD/MM/AA.
- Memo.: Permite a entrada de dados extensos do tipo caracter.

# Programação AdvPl 1

## [Tamanho]

Tamanho do campo, conforme documentação do Microsiga Protheus estes são os tamanhos permitidos para os tipos de dados suportados.

| <b>Tipo do Campo</b> | <b>Tamanho Máximo</b>  |
|----------------------|--|
| Caracter             | Até 512 dígitos  |
| Numérico             | Até 18 dígitos   |
| Lógico               | 1 dígito   |
| Data                 | 8 dígitos  |
| Memo                 | O tamanho informado será o espaço reservado para apresentação, porém não há limite para o cadastramento. |

## [Decimal]

Caso o campo seja Numérico, pode-se definir o numero da casas decimais.

## [Formato]

Pode-se determinar o formato de campo 'mascara', desta forma se o usuário tentar utilizar dados fora da mascara o Protheus não aceitará.

## [Formato variável]

Pode-se definir um formato variável para a edição de um campo, como exemplo temos o campo A1\_CGC da tabela de clientes SA1, sendo um cliente pessoa física podemos definir o formato de CPF, sendo pessoa jurídica atribuímos o formato CNPJ, essa formatação se torna flexível de acordo com uma condição.

## **Mascara de formatação**

Máscara que age apenas sobre um caracter, ou seja, se o dado possuir cinco caracteres a serem formatados, serão necessários cinco caracteres na máscara para executar a formatação no dado por completo.

| <b>Máscara para um caracter</b> |  |
|---------------------------------|--|
| A                               | Aceita apenas caracteres alfabéticos.                              |
| L                               | Aceita apenas caracteres lógicos: T (True), F (False), "N" ou "S". |
| N                               | Aceita apenas letras ou números.                                   |
| X                               | Aceita qualquer caracter.  |
| Y                               | Aceita apenas Y (Yes) ou N (No).                                   |
| 9                               | Aceita somente números e sinais - Ex: 99999-999 ou @R99999-999     |
| !                               | Converte em letras maiúsculas.                                     |
| \$                              | Exibe o símbolo "\$" à frente de um número.                        |
| *                               | Exibe o asterisco à frente de um número.                           |
| ,                               | Exibe uma vírgula.   |

# Programação AdvPl 1

Máscara que age sobre todo o dado dentro de um campo, identificada pelo sinal arroba (@).

| Máscara para todo campo |  |
|-------------------------|--|
| @A                      | Aceita apenas caracteres alfabéticos.  |
| @B                      | O número é ajustado à esquerda.  |
| @C                      | Exibe o sinal "CR" depois de números positivos.  |
| @D                      | Exibe as datas no formato padrão.  |
| @E                      | Aceita a entrada de números no formato brasileiro, ou seja, uso da vírgula como separador de decimal e datas exibidas no formato DD/MM/AA. |
| @K                      | Permite exibir um valor provável do conteúdo do campo.   |
| @R                      | Permite exibir caracteres que não farão parte do conjunto de caracteres a serem gravados.  |
| @Sn                     | Permite o rolamento horizontal para preenchimento do campo no tamanho definido por "n".  |
| @X                      | Exibe a notação "DB" depois de números negativos.  |
| @)                      | Retorna um valor negativo entre parênteses.  |
| @(                      | Retorna um valor negativo entre parênteses com espaços em branco.  |
| @!                      | Converte todos os caracteres alfabéticos em letras MAIÚSCULAS.   |

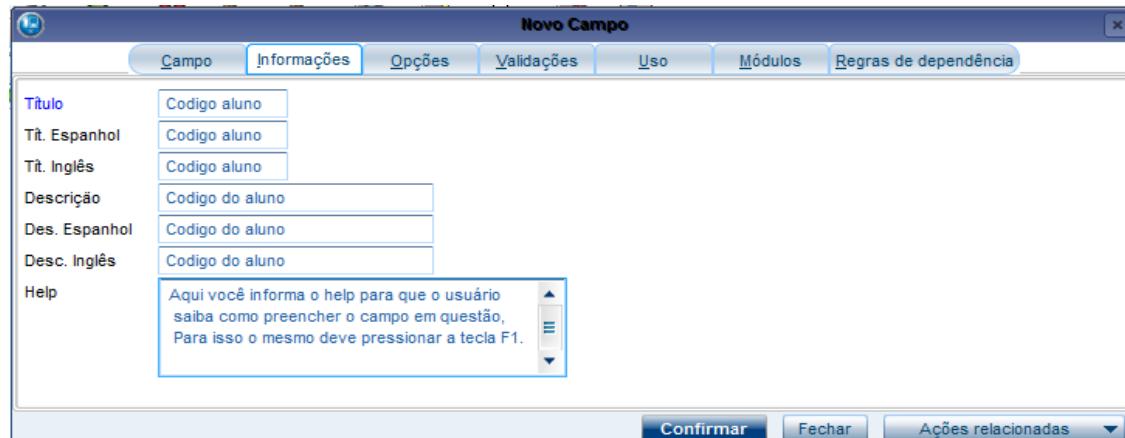
## [Contexto]

Define se o campo é real (existe fisicamente no bando de dados) ou se é um campo vindo de outra Tabela.

## [Propriedade]

Define se o campo pode ou não ser alterado.

Na 'aba' **Informações** vamos atribuir ao campo que esta sendo criado as informações que darão sentido a ele como o "título", "Descrição" e "informações de help".



# Programação Advpl 1

## [Titulo]

O 'Titulo' é o nome do campo que aparecerá para o usuário. Podem ser preenchido nos três idiomas.

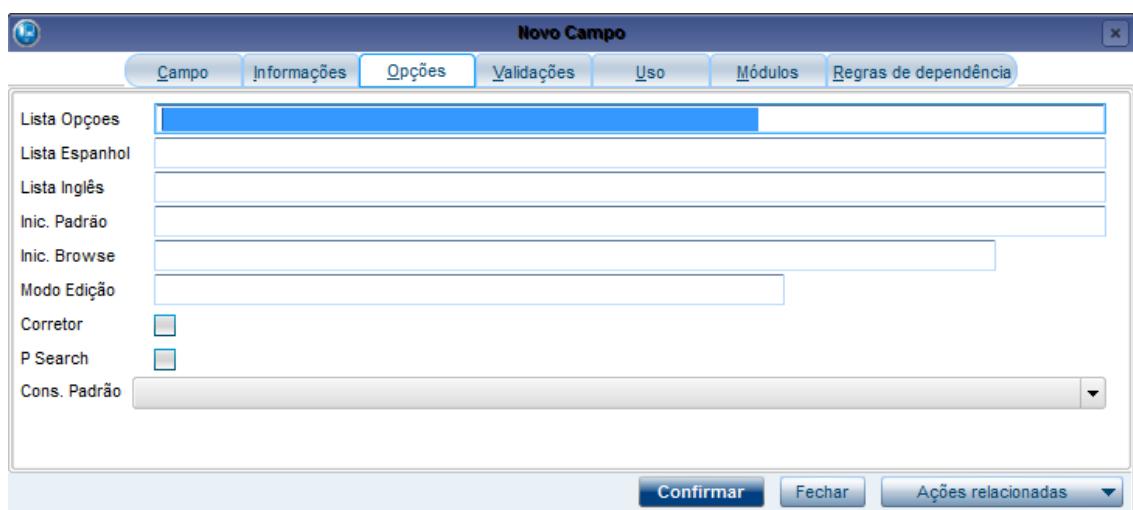
## [Descrição]

Explica sucintamente a função do campo.

## [Help]

Descrição das funcionalidades do campo, é exibido ao se clicar no campo e apertar a tecla F1.

Na 'aba' **Opções** vamos definir se o campo utiliza algum facilitador de preenchimento.



## [Lista de Opções]

O Facilitador 'Lista de Opções' permite que o usuário escolha uma informação dentre uma lista pré-definida. Exemplo de preenchimento da lista:

1=Primeiro;2=Segundo;3=Terceiro

P=Primeiro;S=Segundo;T=Terceiro

1=SIM;2=NÃO

S=SIM;N=NÃO

No campo será gravado somente o caractere referente a opção, exemplo "1" caso usuário escolha a opção "Primeiro".

## [Inic. padrão]

Conteúdo inicial padrão para o campo. Este dado será sugerido na inclusão de um novo registro.

Automaticamente **executado na inclusão**, alimentando o campo que foi configurado para sua utilização. Pode ter seu conteúdo fixo, ser uma função de usuário ou até mesmo uma função protheus.

Para exemplificar o uso de um valor fixo, podemos definir o preenchimento automático na inclusão de clientes, fixar ou melhor dizendo sugerir o conteúdo "SÃO PAULO" para o campo A1\_MUN

# Programação AdvPl 1

Para utilizar uma função de usuário, é necessário criar uma “User Function” e atribuir sua chamada no campo “Inicializador padrão”, exemplo U\_NomeMun(), o retorno dessa função irá preencher o campo.

## [Inic. browse]

Permite a apresentação de um campo virtual com qualquer conteúdo na tela chamada pela função mBrowse. Este campo deve possuir uma expressão ou função de usuário que deve devolver o conteúdo a ser exibido. **Importante:** poderá causar lentidão no sistema.

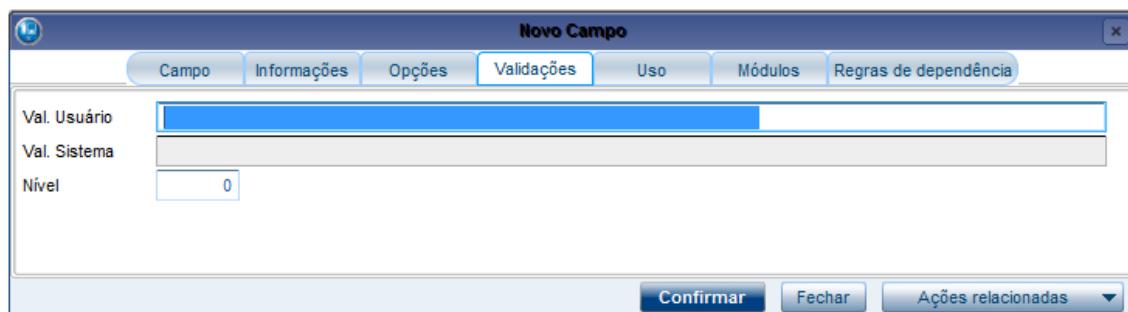
## [Modo de edição]

Permite a edição de um campo apenas quando determinada condição for verdadeira. A regra poderá ser uma “expressão em ADVPL”, “função de usuário”. Note que o retorno deverá ser, **obrigatoriamente**, do tipo lógico (True ou False), por exemplo, se você informar nesse campo a variável PUBLICA INCLUI, o campo somente será editado no momento da inclusão, mesma situação se você entrar com a variável PUBLICA ALTERA.

## [Consulta Padrão]

Nome da consulta padrão ou número da tabela a ser chamada sempre que a tecla F3 for acionada.

Na ‘aba’ **Validações** vamos estabelecer o critério de aceitação das informações digitadas pelo usuário.



## [Validação do usuário]

Utilizando-se de funções Protheus, funções de usuário ou expressões AdvPl, é possível validar se o conteúdo do campo foi preenchido corretamente, caso positivo, o usuário poderá desfocar do campo correto e editar o próximo campo do cadastro, caso negativo, o mesmo não conseguirá desfocar do campo até que entre com um conteúdo válido.

## [Validação do sistema]

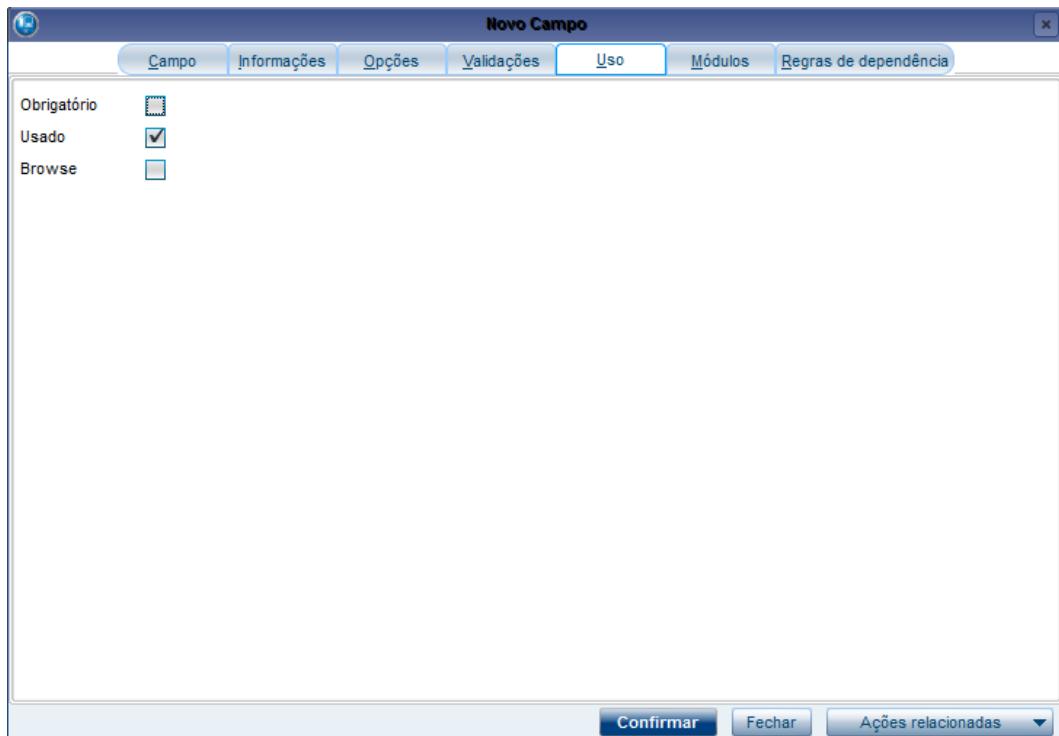
Similar a validação do usuário, porém essa campo é utilizado exclusivamente pela Microsiga.

## [Nível]

Através deste campo é possível restringir o acesso de um usuário a um determinado campo. O ‘nível do campo’ deve ser inferior ou igual ao ‘nível do usuário’ e o campo não deve ser Obrigatório.

# Programação AdvPl 1

Na ‘aba’ **Uso** definimos a “obrigatoriedade” da informação, se o campo é “usado” e se ele aparece no “browse”.



[Obrigatório]

Indica a obrigatoriedade do preenchimento do campo para o cadastro.

[Usado]

Indica se o campo está sendo usado pelo sistema.



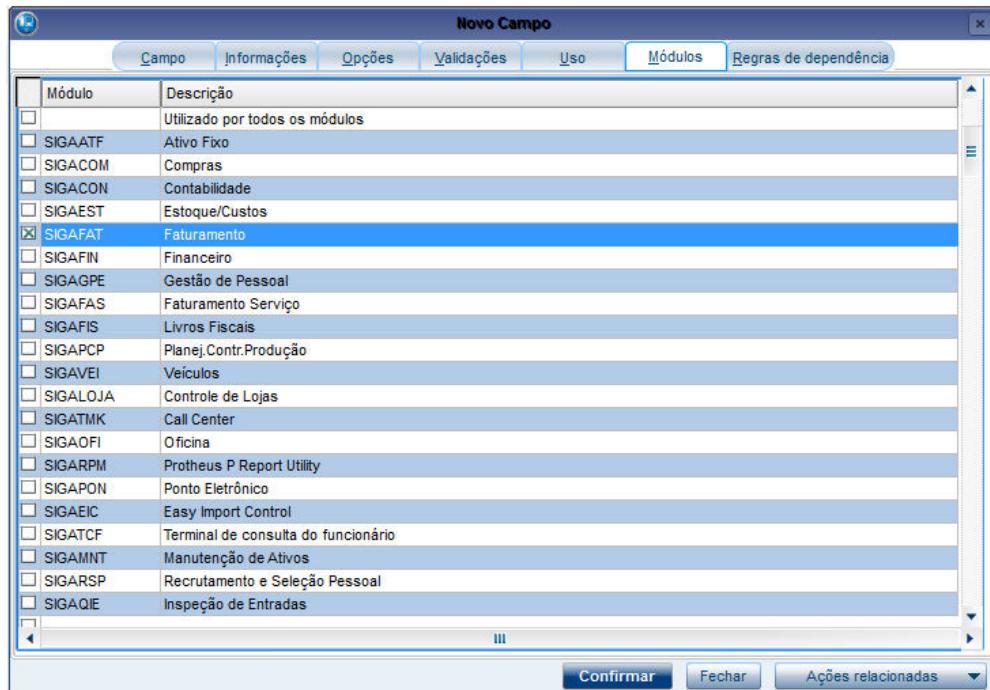
Só é permitida a deleção de campos não usados.

[Browse]

Indica se o campo será exibido na MBrowse, tela inicial onde aparecem geralmente os botões Inserir, Alterar e etc...

Na ‘aba’ **Módulos** temos a possibilidade de restringir o uso desta tabela ou campo a um único módulo ou ambiente é possível na aba Módulos, conforme tela abaixo:

# Programação Advpl 1



No exemplo na tela acima, o campo em questão será apresentado somente se a tabela for editada no módulo "faturamento (SIGAFAT)".

## Na 'aba' Regras de Dependência. (Conceito MCV)

Definir as regras de dependência entre campos que utilizam o modelo MVC de desenvolvimento.

**Observação:**

✓ O modelo MVC é tratado exclusivamente no treinamento ADVPL V, não será tratado nesse treinamento.

### Para o seu conhecimento:

Alguns campos do Microsiga Protheus tem características de dependência para preenchimento, esses tem algum tipo de ligação que exigem um tratamento especial, como exemplo temos os campos A1\_CODIGO e A1\_LOJA do cadastro de clientes SA1, no Protheus a referência do cliente é a junção desses campos, nesse caso então o ideal é que o campo LOJA seja inserido após a digitação do campo CÓDIGO, usando conceito de 'regras de dependência', podemos inserir a regra no dicionário de dados.

Campo que dispara a regra é chamado CAMPO DOMÍNIO, no nosso exemplo A1\_CLIENTE, nesse caso o A1\_LOJA é chamado CONTRA DOMÍNIO.

# Programação AdvPl 1

## Três formas de utilizar a regra de dependência:

Pré validação, o conteúdo do 'contra domínio' não poderá ser inserido antes do conteúdo do campo 'domínio';

Pós validação, no caso de atualização do 'domínio'      ' será feito uma reavaliação do conteúdo do 'contra domínio';

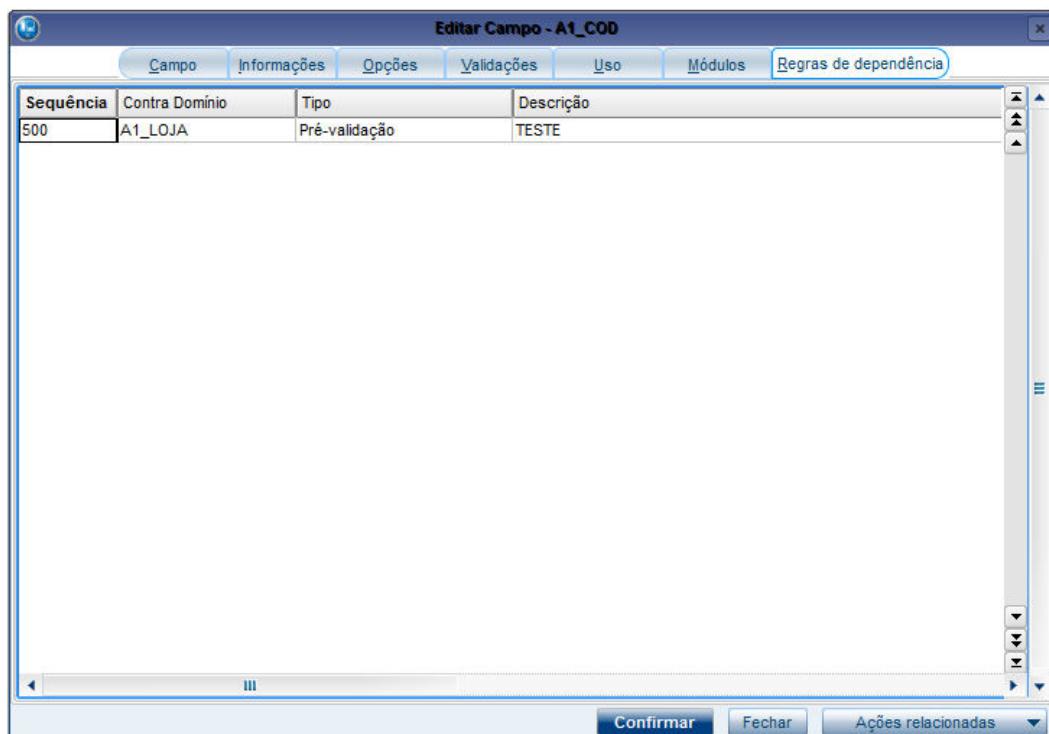
Pré e pós validação, combina as validações acima citadas.

## Seqüência de numeração de controle das 'regras de dependências':

001 à 499 são regras inseridas pela Microsiga;  
500 à 999 podem ser inseridas pelo usuário.

Usuário não poderá excluir as regras pré inseridas pela Microsiga.

Ao excluir um campo no configurador, as 'regras de dependências' serão excluídas, tanto inseridas pelo usuário quanto pela Microsiga.



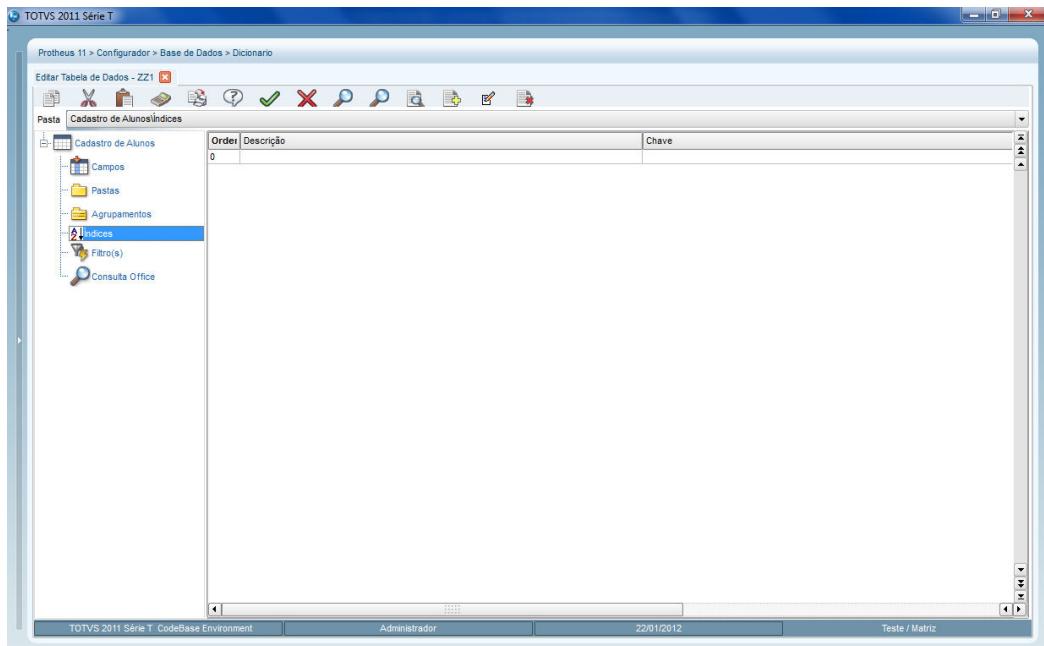
Pronto, terminamos o cadastro de nosso campo, clique no botão CONFIRMAR para gravar na SX3. Na seqüência vamos criar os índices da nossa tabela.

# Programação AdvPl 1

## Criação de Índices (SIX)

O Protheus permite a Inclusão de Índices personalizados na base, a inclusão destes itens visa melhorar o desempenho geral do sistema.

Para realizar a inclusão de índices, seleciona a opção “Índice”, será apresentado a tela:



Para inclusão de um índice, cliquei no botão ‘incluir’  , e siga a orientação a seguir:

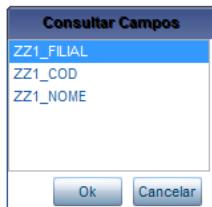


### [Chave]

Nesse campo você deverá informar os ‘campos’ que farão parte da composição do índice, denominada ‘chave de índice’.

Para selecionar os campos a serem incluído na chave, clique em “ações relacionadas → campos”, será mostrado a tela:

# Programação AdvPl 1



Seleciona o campo desejado e clique em OK, o mesmo será inserido no campo 'chave', para inserção de mais campos realize o processo novamente, iniciando em 'ações relacionadas'.

## Fique Atento:

- ✓ A chave do índice, isto é a relação dos campos que farão parte de sua composição deve sempre ser iniciada pelo campo XXX\_FILIAL, onde XXX é o prefixo da tabela onde estamos criando nosso índice.

[Nickname]

A única garantia que seu índice não será deletado / substituído é a utilização do Nickname, necessário utilizá-lo quando na criação de índices em tabelas padrões, assim nas próximas atualização do sistema, esses índices não sofrerão alteração.

[Descrição]

Descrição do índice em questão nos três idiomas.

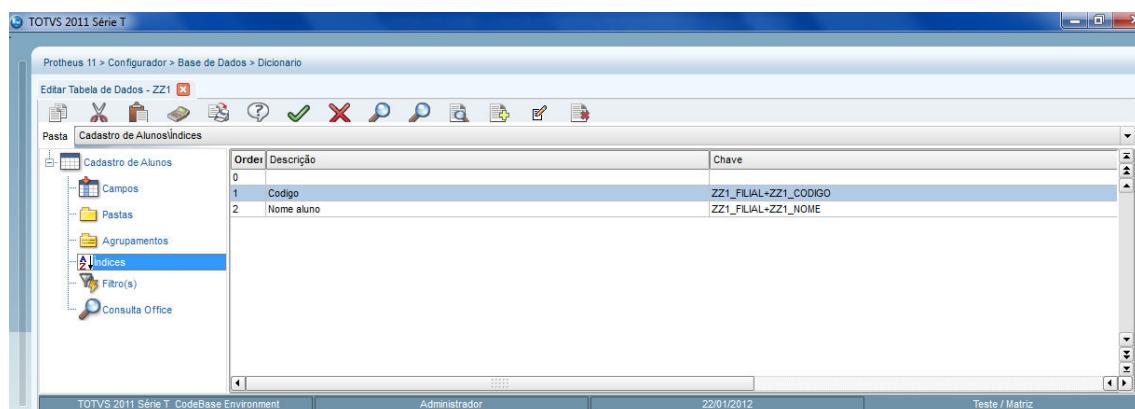
[Mostra pesq.]

Necessário informar esse campo quando você precisa disponibilizar pesquisa utilizando o índice.

Tela após o preenchimento dos campos:



Clique no botão CONFIRMAR será apresentada a tela:



# Programação AdvPl 1

Note que fizemos a inclusão de mais uma chave de índice, essa utilizando os campos ZZ1\_FILIAL + ZZ1\_NOME, para isso foi clicado no botão 'incluir'  e realizados dos os passos anteriormente.

Agora precisamos salvar as alterações que fizemos na estrutura da nossa tabela, para isso clique no botão 'OK' , em seguida clique no ícone  para efetivar as alterações no banco de dados.

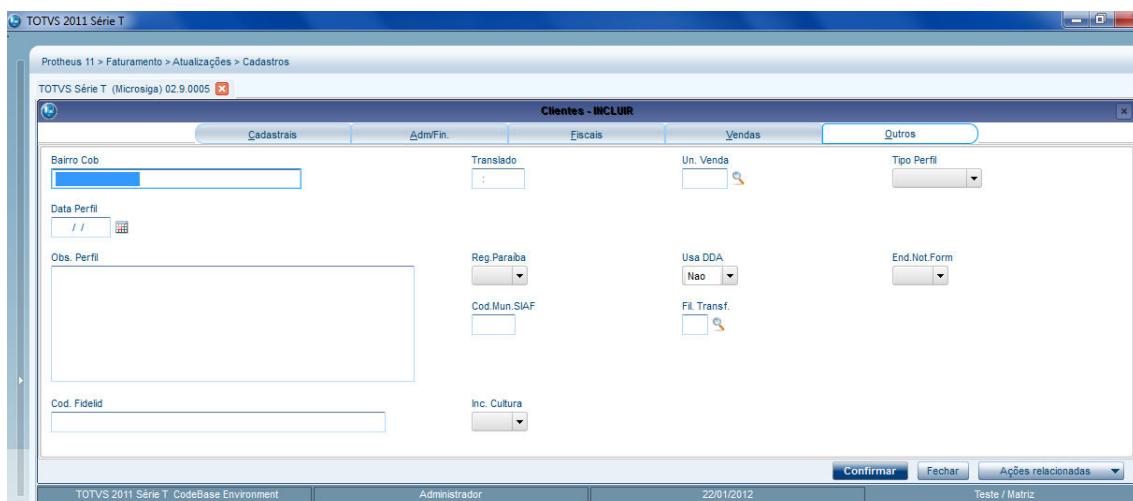


Esse ultimo processo é a efetivação das alterações no banco de dados, somente poderá ser realizada em modo exclusivo, para isso, nenhum usuário poderá estar utilizando o sistema.

## Criação de Pastas (SXA)

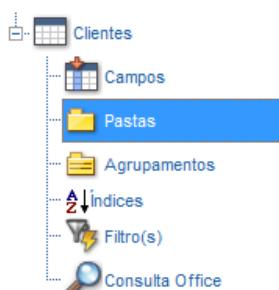
O Protheus permite a organização dos dados em Pastas para melhor entendimento de um cadastro.

Para exemplificar sua utilização, vamos dar manutenção nas pastas da tabela SA1 que é nosso cadastro de clientes, atualmente essa tabela esta configurada com as pastas 'Cadastrais', 'Adm/Fin', 'Fiscais', 'Vendas' e 'Outros' conforme tela:



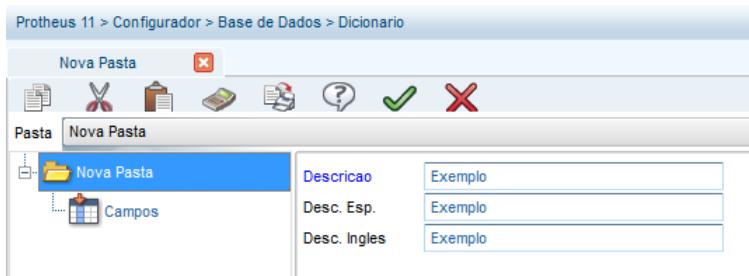
Vamos realizar a inclusão de uma nova pasta intitulado "Exemplo".

No módulo configurador, localize a tabela "SA1", e clique em 'editar' . Clique na opção 'pastas':

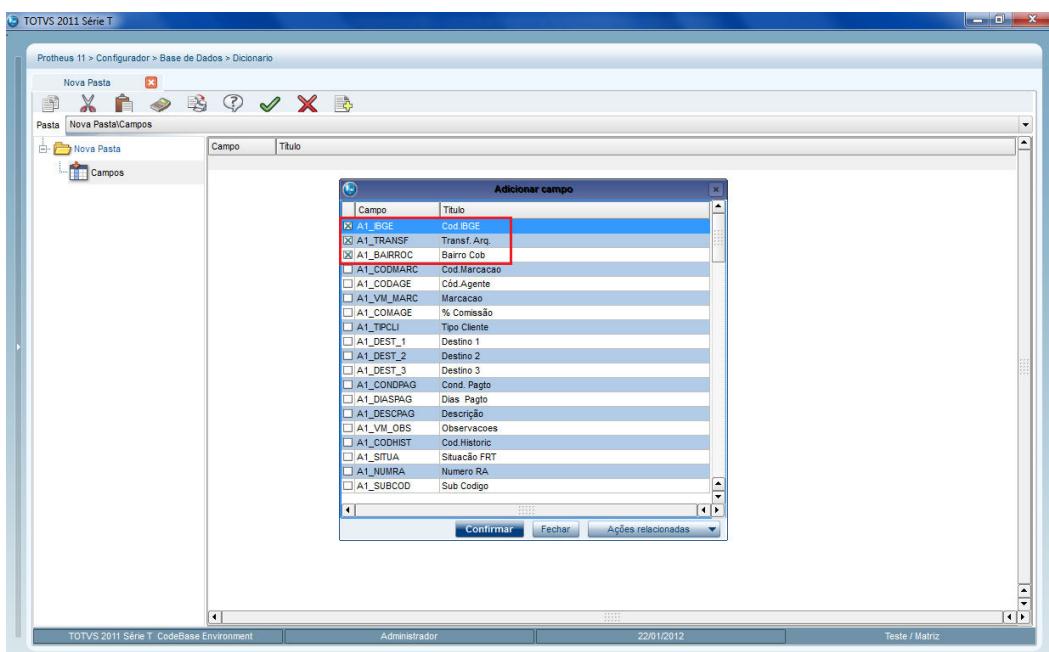


# Programação AdvPl 1

Clique no botão 'incluir'  , informe o nome da nova pasta nos três idiomas.



Clique em Campos, em seguida 'incluir'  para inserir os campos que deseja que sejam apresentados na nova guia, escolha os campos e clique em confirmar, conforme tela abaixo, em seguida clique em confirmar.



Na seqüência clique no botão 'ok'  e 'atualizar base de dados'  para salvar as alterações no dionario.

# Programação AdvPl 1

## Exercício

Crie as tabelas abaixo conforme orientação do instrutor

### SX2

|            |                         |
|------------|-------------------------|
| PREFIXO    | SZ1                     |
| PATCH      | \DATA\                  |
| DESCRIÇÃO  | CABEÇALHO PEDIDO VENDAS |
| AC FILIAL  | EXCLUSIVO               |
| AC UNIDADE | EXCLUSIVO               |
| AC EMPRESA | EXCLUSIVO               |

### SX3

| CAMPO      | TÍTULO        | TIPO | TAM | DIVERSOS   |
|------------|---------------|------|-----|--|
| Z1_NUM     | Numero pedido | C    | 6   | CONTEXTO: REAL<br>PROPRIEDADE: VISUALIZAR<br>INIC. PADRÃO: GETSXENUM("SZ1","Z1_NUM")<br>USADO / BROWSE |
| Z1_DATA    | Data venda    | D    | 8   | CONTEXTO: REAL<br>OBRIGATÓRIO / USADO/ BROWSE  |
| Z1_CLIENTE | Cliente       | C    | 6   | CONTEXTO: REAL<br>CONSULTA PADRÃO: SA1<br>OBRIGATÓRIO / USADO / BROWSE                                 |
| Z1_LOJA    | Loja          | C    | 2   | CONTEXTO: REAL<br>OBRIGATÓRIO / USADO / BROWSE   |
| Z1_NOMECLI | Nome cliente  | C    | 30  | CONTEXTO: VIRTUAL<br>USADO   |
| Z1_STATUS  | Status        | C    | 1   | CONTEXTO: REAL<br>USADO / BROWSE<br>OPÇÕES: 1=Atendido;2=Não Atendido                                  |

### SIX

| INDICE | CHAVE                     |
|--------|---------------------------|
| 1      | Z1_FILIAL + Z1_NUM        |
| 2      | Z1_FILIAL + Z1_CLIENTE    |
| 3      | Z1_FILIAL + DTOS(Z1_DATA) |

### SX2

|            |                        |
|------------|------------------------|
| PREFIXO    | SZ2                    |
| PATCH      | \DATA\                 |
| DESCRIÇÃO  | ITENS DO PEDIDO VENDAS |
| AC FILIAL  | EXCLUSIVO              |
| AC UNIDADE | EXCLUSIVO              |
| AC EMPRESA | EXCLUSIVO              |

# Programação Advpl 1

---

## SX3

| CAMPO      | TÍTULO            | TIPO | TAM  | DIVERSOS   |
|------------|-------------------|------|------|--|
| Z2_NUM     | Numero pedido     | C    | 6    | CONTEXTO: REAL<br>USADO  |
| Z2_PRODUTO | Código do produto | C    | 15   | CONTEXTO: REAL<br>CONSULTA PADRÃO: SB1<br>OBRIGATÓRIO / USADO / BROWSE |
| Z2_QUANT   | Quantidade        | N    | 8,2  | CONTEXTO: REAL<br>OBRIGATÓRIO / USADO /BROWSE                          |
| Z2_UNIT    | Preço unitário    | N    | 12,4 | CONTEXTO: REAL<br>OBRIGATÓRIO / USADO / BROWSE                         |
| Z2_TOTAL   | Total do item     | N    | 10,2 | CONTEXTO: REAL<br>USADO / BROWSE                                       |

## SIX

| INDICE | CHAVE              |
|--------|--------------------|
| 1      | Z2_FILIAL + Z2_NUM |

# Programação Advpl 1

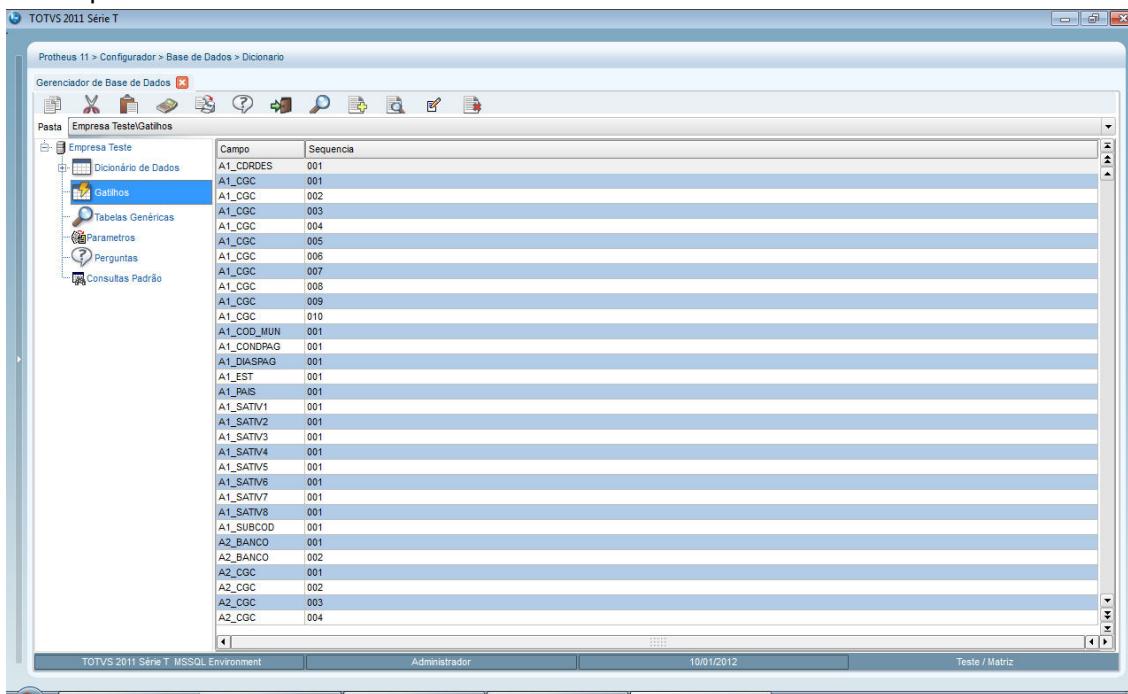
## Gatilhos (SX7)

Rotina que é executada assim que o campo configurado (DOMINIO) tive seu conteúdo alterado, nesse momento podemos realizar uma pesquisa ou um cálculo utilizando função de usuário ou expressão ADVPL, o resultado será gravado em um outro campo (CONTRA DOMINIO).

### Procedimento

Para incluir um novo gatilho ou dar manutenção em um existente, siga os procedimentos:

No módulo configurador selecione o menu “base de dados → dicionários→ gatilhos”, será apresentada a tela abaixo:



A partir dessa tela podemos incluir um novo gatilho ou alterar um existente.

Os gatilhos “sempre” estarão atrelados a um campo de tabela (SX3), então para poder alterá-los, devemos fazer a pesquisa clicando no botão e informar o nome de um campo, por exemplo vamos pesquisar o campo E2\_FORNECE, esse está atrelado ao disparo um gatilho.

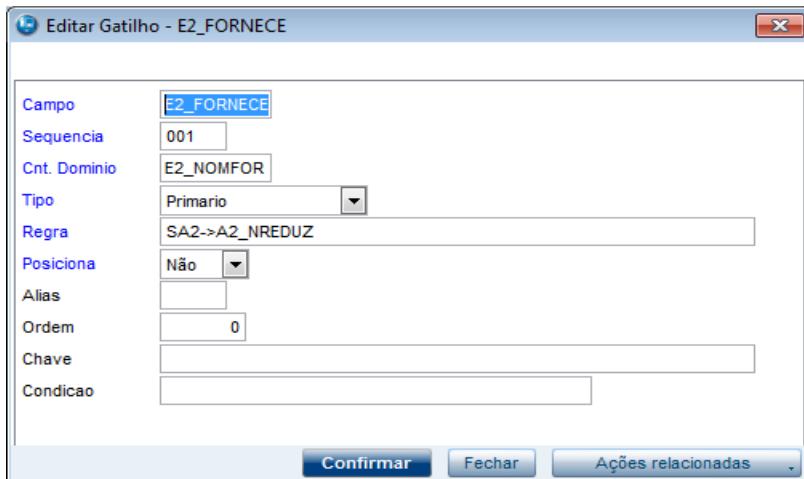
The screenshot shows the 'Gatilhos' search interface. In the 'Procurar por:' field, 'E2\_FORNECE' is typed. Below the search bar are two buttons: 'Procurar' (Search) and 'Ligar' (Link). To the right of these buttons is a 'Opções' (Options) section with checkboxes for 'Coincidir maiúsc./minúsc.' (Match uppercase/lowercase) and 'Localizar palavra inteira' (Match whole word). The main table lists trigger fields and their sequences. The first entry, 'E2\_FORNECE', has a sequence of 001 and is highlighted.

| Campo      | Seqüencia |
|------------|-----------|
| E2_FORNECE | 001       |
| E2_FORNECE | 002       |
| E2_LOJA    | 001       |
| E2_NATUREZ | 001       |
| E2_TXMOEDA | 001       |
| E5_AGENCIA | 001       |
| E5_BANCO   | 001       |
| E5_CONTA   | 001       |
| E5_EDTPMS  | 001       |
| E5_NATUREZ | 001       |
| E5_PROJMS  | 001       |
| E5_PROJPMS | 002       |
| E5_TASKPMS | 001       |

Observe que o campos pesquisado está atrelado a duas configurações de gatilhos, então no momento que o mesmo for alterado ambas serão executadas sequencialmente.

# Programação Advpl 1

Vamos estudar os campos e o funcionamento do gatilho em foco.



Ao ser alterado o conteúdo do campo E2\_FORNECE, o gatilho realiza a operação de gravação do conteúdo do campo A2\_NREDUZ no campo E2\_NOMFOR.

Veja tela abaixo o campo NOME FORNECE preenchido com o conteúdo "TESTE", esse foi alimentado automaticamente com a digitação do conteúdo "000001" no campo FORNECEDOR

## [Campo]

Nome do campo que ao ser alterado irá iniciar o processo de atualização dos outros campos no sistema.

## [Seqüência]

Seqüência de execução do gatilho no sistema, ele pode preceder ou anteceder um outro gatilho no mesmo campo.

## [Cnt Domínio]

Campo que receberá o preenchimento automático ao ser executado o Gatilho.

## [Tipo]

Seleciona o tipo do gatilho, sendo:

- **Tipo P:** Primário para atualizações visuais e externas do mesmo arquivo.
- **Tipo E:** Estrangeiro para atualizações de dados em outros arquivos.
- **Tipo X:** Posicionamento para situar o arquivo mencionado no alias sem efetuar nenhuma atualização. Utilizado para casos em que o usuário deseja estabelecer um relacionamento entre os arquivos.

# Programação Advpl 1

---

## [Regra]

Expressão em ADVPL cujo o resultado é transportado para o Contra Domínio.

## [Posiciona]

Caso seja selecionada a opção "Sim" o ponteiro se movimenta no outro arquivo com base na expressão definida no campo 'Chave' ou "Não", caso contrário.

## [Alias]

Se Posiciona for igual a "sim" , então deverá ser informado qual o alias a Posicionar

## [Ordem]

Determina a ordem para pesquisa do Alias informado.

## [Chave]

Chave a ser pesquisada no alias informado de acordo com o campo 'Posiciona'.

## [Condição]

Informe a condição usando "expressão ADVPL, "execblocks" ou "função de usuário", issa irá determinar se o gatilho será executado.

# Programação AdvPl 1

## Exercícios

Crie os 2 gatilhos abaixo, ambos irão realizar uma busca na tabela de cliente e irá alimentar o campo contra domínio Z1\_NOMECLI, note que somente serão executados se a expressão no campo CONDIÇÃO for verdadeira.

| GATILHO NA DIGITACAO CAMPO CLIENTE |  |
|------------------------------------|--|
| CAMPO                              | Z1_CLIENTE   |
| CNT DOMINIO                        | Z1_NOMECLI   |
| TIPO                               | PRIMARIO   |
| REGRA                              | SA1->A1_NOME   |
| POSICIONE                          | NÃO  |
| ALIAS                              | SA1  |
| ORDEM                              | 1  |
| CHAVE                              | XFILIAL("SA1") + M->Z1_CLIENTE + M->Z1_LOJA                |
| CONDIÇÃO                           | .NOT. ( EMPTY( M->Z1_CLIENTE ) .AND. EMPTY( M->Z1_LOJA ) ) |

| GATILHO NA DIGITACAO CAMPO LOJA |  |
|---------------------------------|--|
| CAMPO                           | Z1_LOJA  |
| CNT DOMINIO                     | Z1_NOMECLI   |
| TIPO                            | PRIMARIO   |
| REGRA                           | SA1->A1_NOME   |
| POSICIONE                       | NÃO  |
| ALIAS                           | SA1  |
| ORDEM                           | 1  |
| CHAVE                           | XFILIAL("SA1") + M->Z1_CLIENTE + M->Z1_LOJA                |
| CONDIÇÃO                        | .NOT. ( EMPTY( M->Z1_CLIENTE ) .AND. EMPTY( M->Z1_LOJA ) ) |

# Programação Advpl 1

## Consulta Padrão (SXB)

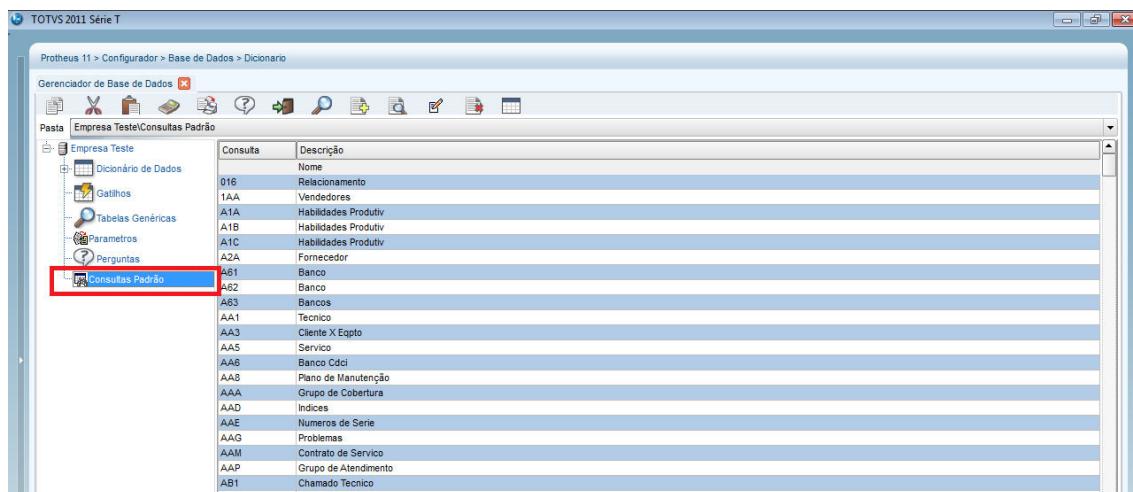
As consultas padrões são as opções de pesquisa dos arquivos do sistema. Estas consultas são apresentadas, por exemplo, na pesquisa de um fornecedor para preencher o campo "Fornecedor" de determinada rotina.

As opções para pesquisa deste fornecedor são apresentadas na tela de consulta, permitindo que se pesquise o fornecedor pelo código, pelo nome, pelo CNPJ etc. Estas definições são estruturadas nesta rotina.

### Procedimentos

Para dar manutenção no cadastro de "consultas padrões", no módulo Configurador acesse o menu "base de dados → dicionário → base de dados", selecione a empresa e clique na opção "consulta padrão".

O sistema apresenta na tela a relação das consultas padrão utilizadas no sistema:



Para incluir uma nova consulta, clique no botão 'incluir' e preencha os campos conforme descrição a seguir:

The screenshot shows the 'Nova Consulta' (New Query) dialog box. On the left, under 'Tipo de Consulta' (Type of Query), 'Consulta Padrão' is selected (indicated by a checked radio button). On the right, the 'Informações' (Information) section contains four input fields: 'Consulta:' with value 'SA1SZC', 'Descrição:' with value 'CLIENTES X AGENCIA', 'Descrição Espanhol:' with value 'CLIENTES X AGENCIA', and 'Descrição Ingles:' with value 'CLIENTES X AGENCIA'. At the bottom right of the dialog are two buttons: 'Avançar >>' (Next >>) and 'Cancelar' (Cancel).

# Programação AdvPl 1

## [Tipo de Consulta]

Selecione o tipo da consulta a ser cadastrada, neste caso 'Consulta Padrão', existem outros tipos de consulta conforme a seguir:

- **Consulta Padrão:** permite criar uma estrutura onde é possível realizar pesquisas em tabelas utilizando índices pré-definidos.
- **Consulta Específica:** a consulta específica é um programa customizado, criado via função de usuário (user function) para apresentar determinada informação para a consulta.
- **Consulta Usuários:** permite criar uma estrutura onde é possível realizar pesquisas na tabela de cadastro de usuários do sistema.
- **Consulta Grupo de Usuários:** permite criar uma estrutura onde é possível realizar pesquisas na tabela de cadastro de grupos de usuários do sistema.

Para cada tipo de consulta, os dados a serem informados no cadastro serão distintos.

Na área "Informações", preencha os campos conforme descrição a seguir:

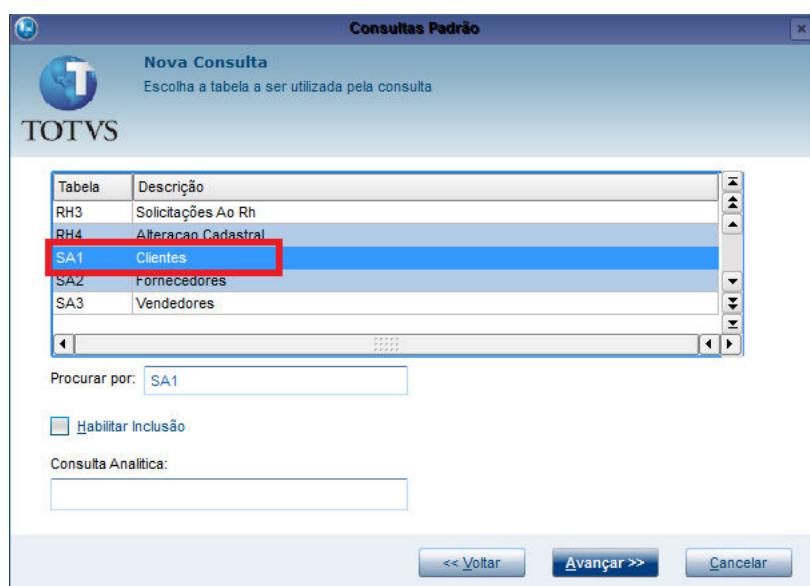
## [Consulta]

Aqui você deve informar o nome da consulta padrão, essa será utilizada como referência para a mesma.

## [Descrição]

Aqui você deve informar a descrição da consulta padrão, nos três idiomas.

Após preenchidos os dados da primeira tela, clique no botão 'avançar' **Avançar >>**, o sistema apresenta a tela para seleção da tabela a ser utilizada na consulta:



# Programação Advpl 1

Nesta configuração apresentamos os seguintes campos a saber:

[Grid de Dados]

Apresenta as tabelas disponíveis.

[Procura por]

Informe o nome da tabela para pesquisa na lista acima.

[Habilita Inclusão]

Permite a inclusão de registros não encontrados, permitindo o avanço de um cadastro sem que o usuário necessite sair de sua tela.

[Consulta Analítica]

Exibe a 'Consulta Padrão' através de um próprio (User Function).

As consultas analíticas são acessadas pela tecla [F5] nas rotinas de manutenção.

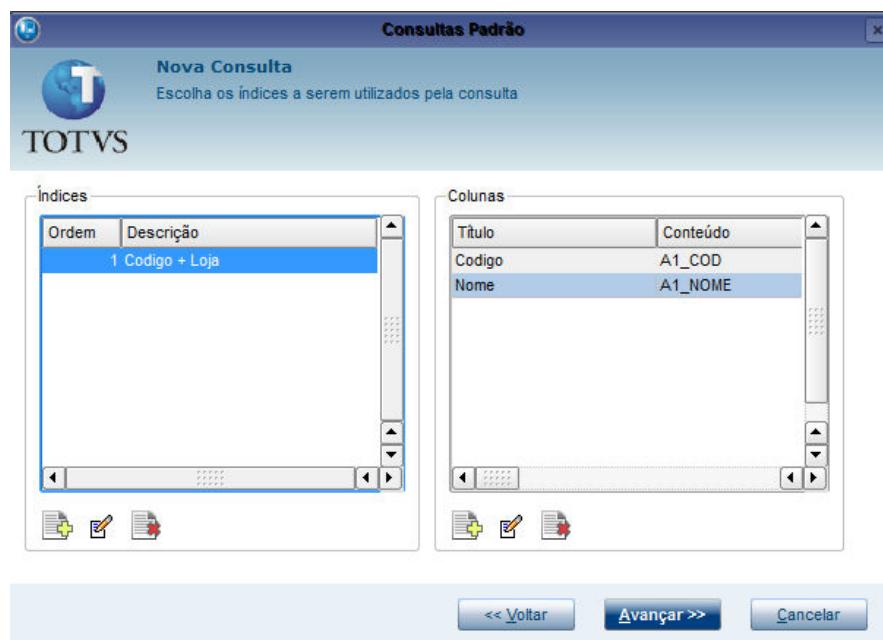
Exemplos de 'Consultas Padrões' com 'Consulta Analítica':

AIA – TABELA DE PREÇOS

SL2 – PRODUTO.

Clique no botão 'avançar' **Avançar >>**.

O sistema apresenta nova tela, onde devem ser informados os Índices utilizados pela consulta, e suas respectivas colunas:



[Grid Indices]

Permite informar em que ordem os campos serão apresentados na consulta 'linhas'.

[Grid Colunas]

Permite informar em que ordem os campos serão apresentados nas colunas.

Informados os dados referentes aos índices, clique no botão 'avançar' **Avançar >>**, o sistema apresenta a tela para montagem de filtros e retornos:

# Programação Advpl 1



## [Filtro]

Podemos utilizar esse campo para inserir uma regra de filtro, onde somente os dados que equivalem a regra preenchida serão apresentados na consulta padrão. Essa regra poderá ser uma expressão ADVPL ou uma função de usuário, exemplo:

'CC2\_EST == M->A1\_EST'.

## [Retorno]

Informe os campos que terão seus conteúdos retornados assim que usuário escolher um registro.

Preenchidos os campos, clique no botão "Finalizar" para concluir a inclusão da consulta padrão.

## Exercício

Com orientação do instrutor crie uma nova consulta padrão lendo a tabela de CLIENTE (SA1) e a utilize na tabela SZ1.

## Capítulo 14 – Menus

### Incluindo Item no Menu

Os menus do sistema podem ter suas opções reformuladas de maneira que cada usuário possa ter um menu de acesso próprio de acordo com o tipo de trabalho por ele desenvolvido. É possível criar um único menu a partir de opções de diversos ambientes do Protheus.

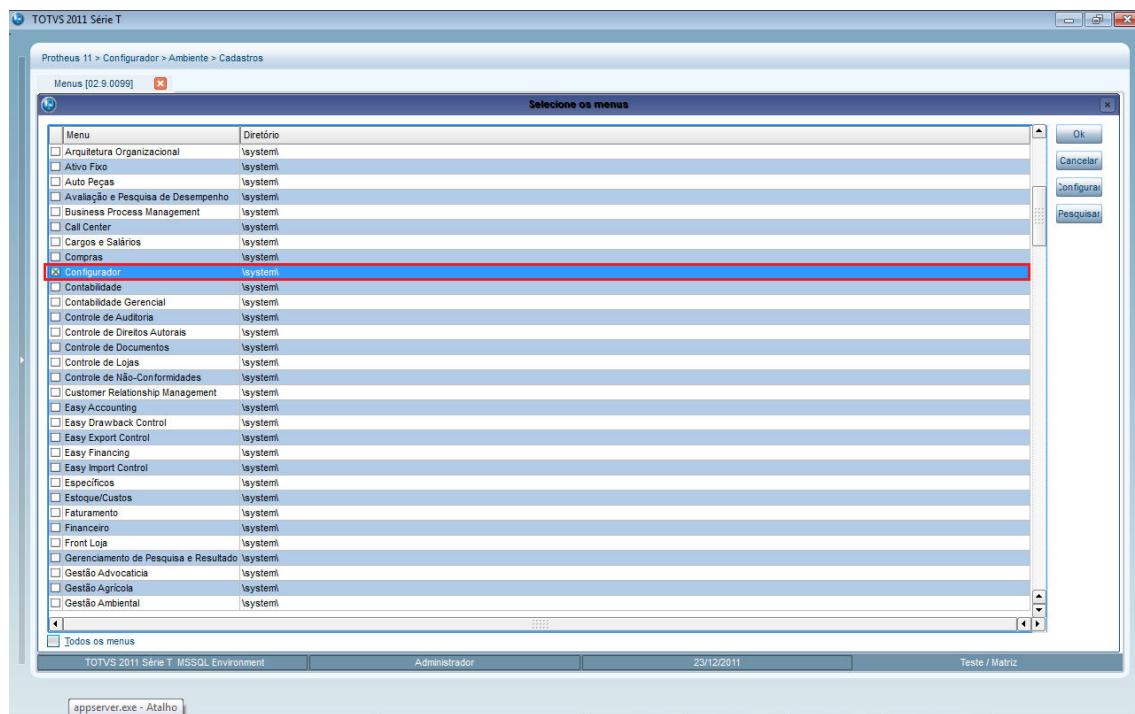
O programa de “Configuração de Menus” deve ser utilizado em conjunto ao “Configuração de Senhas”.

Neste programa, deve ser definido um menu para cada usuário e na configuração de senhas, associa-se este menu com a senha do usuário que determinará quais ambientes, rotinas, empresas e filiais ele poderá acessar.

A configuração de menus parte de um já existente (Menu de Entrada) que ao ser redefinido irá gerar um novo menu (Menu de Saída).

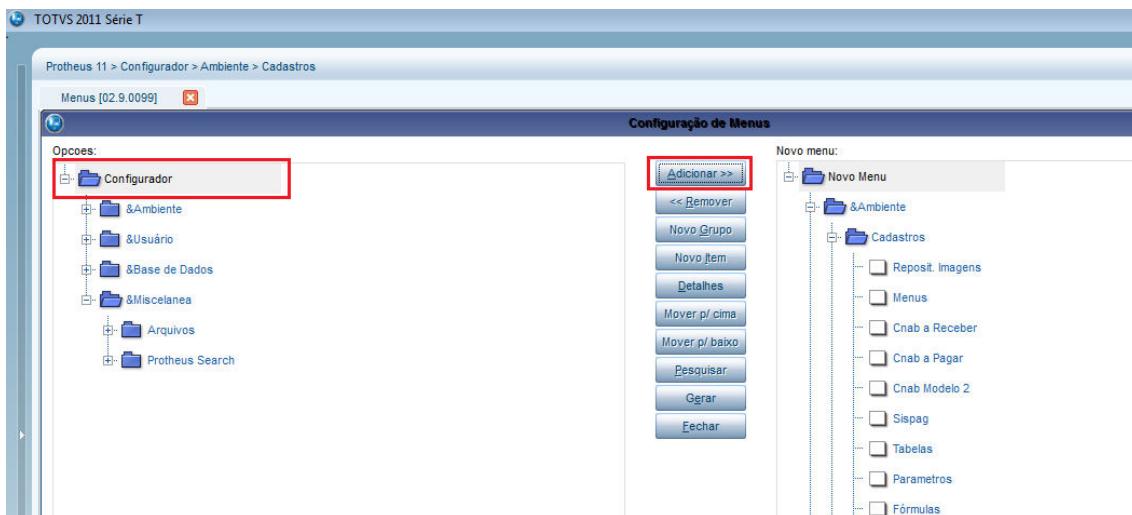
#### Procedimentos

No módulo configurador acesse o menu “ambiente → cadastros → menus”, será apresentada a tela a seguir:



Clique no campo  Todos os menus para desmarcar todos, em seguida escolha o módulo que pretende alterar o menu clicando no check de referencia, clique em ok.

# Programação Advpl 1



As configurações do menu escolhido serão apresentados nessa tela, para que seja possível dar manutenção em sua estrutura, selecione a primeira opção do menu, no exemplo a opção selecionada foi “Configurador”, e clique no botão “Adicionar”.

A partir de agora, todo conteúdo do menu corrente será apresentado na janela a direita entitulada “Novo menu”, agora podemos dar manutenção em sua estrutura.

Segue instrução de cada botão apresentado na tela anterior:

**Adicionar >>** Permite que seja adicionado na janela “Novo menu” todo conteúdo do menu corrente, desde que a primeira opção desse esteja selecionada, como foi no exemplo anterior. Mas também permite que possamos adicionar uma opção de outro menu, desde que esse tenha sido selecionado, isto é, podemos selecionar 2 ou mais menus, escolhe-se somente 1 que terá seu conteúdo copiado na íntegra para a janela manutenção, assim, podemos selecionar outra opção qualquer os outros menus e adicioná-las no que está sendo alterado.

**<< Remover** Permite que se faça a remoção de uma opção “item” ou um grupo de opções do menu que está sendo alterado.

**Novo Grupo** Permite que seja criado um novo grupo no menu que está sendo alterado.

**Novo Item** Selecionando um grupo no menu que está sendo editado, esse botão tem a função de criar um novo item dentro desse, abaixo faremos uma simulação dessa operação.

**Detalhes** Permite que seja editado o conteúdo de uma opção ou a descrição de um grupo.

**Gerar** Permite que seja gravado o conteúdo de um novo menu ou mesmo a gravação das alterações do menu corrente, abaixo faremos uma simulação dessa operação.

**Fechar** Encerra a operação de manutenção de menus, caso não tenha clicado no botão “Gerar”, as alterações ou inclusões realizadas no menu corrente não serão gravadas.

# Programação AdvPl 1

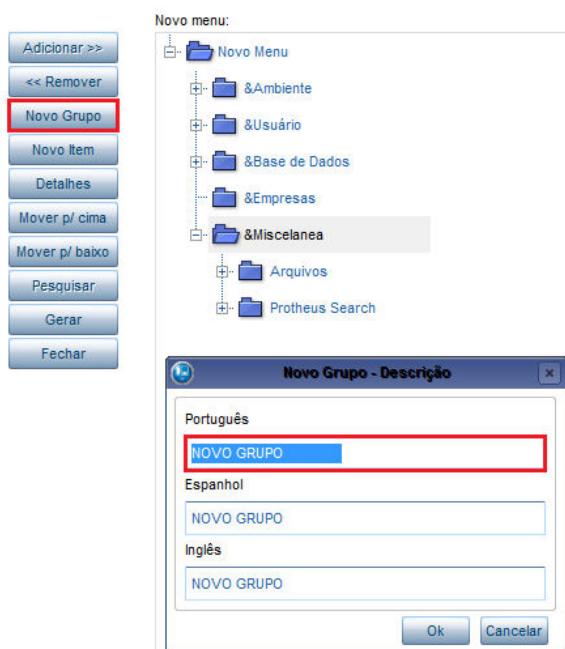
## Fique Atento:

- ✓ Para que seja possível dar manutenção no menu escolhido, é imprescindível que se clique no botão “Adicionar”, isso com a primeira opção do menu selecionada, antes de fazer qualquer alteração para não sobrepor indevidamente o menu de produção.

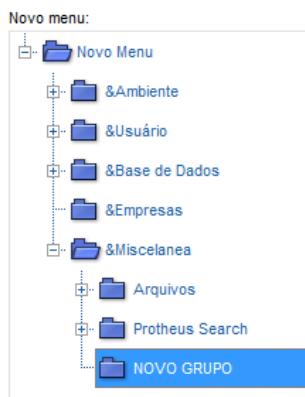
## Procedendo alterações no menu corrente

Criação de um novo grupo:

Após ter clicado no botão “Adicionar” e o menu a ser alterado estiver disponível na janela “Novo menu”, escolha onde o novo grupo dera ser criado e clique no botão “Novo grupo”, será apresentada tela abaixo:

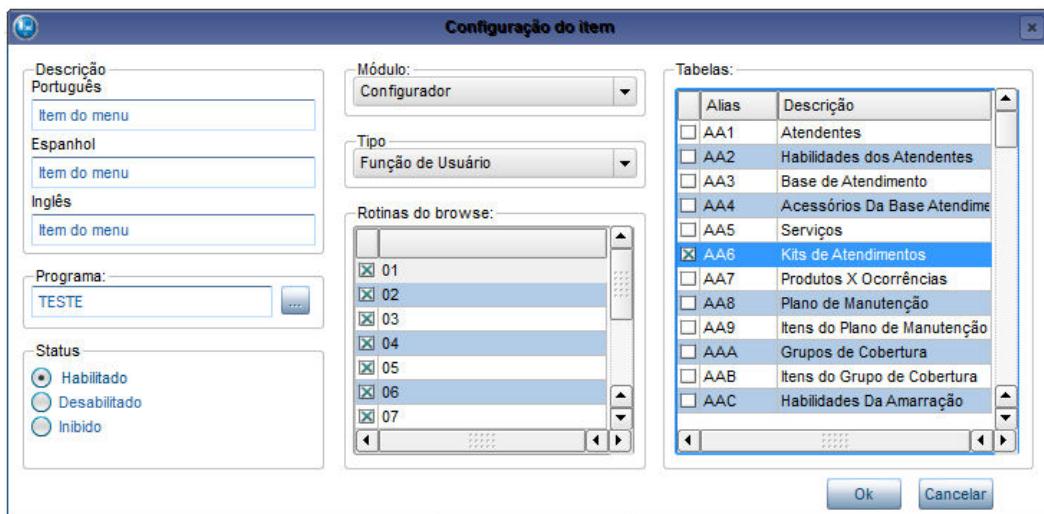


Pronto, criamos nosso novo grupo, agora vamos mostrar como se adiciona uma nova opção, segue procedimentos:



Com o grupo que ira receber o novo item selecionado, clique no botão “Novo item”, será apresentado a tela a seguir:

# Programação Advpl 1



Edite os dados conforme a orientação a seguir:

**[Descrição]**

Aqui você informa a descrição que irá ser apresentado ao usuário nos 3 idiomas.

**[Programa]**

Aqui você informa o nome da função que será executada no momento que o usuário clica na opção do menu

**[Status]**

Habilitado, opção do menu em operação e disponível ao usuário;

Desabilitado, opção não estará visível nem disponível ao usuário.

Inibido, opção será apresentada, mas não estará disponível;

**[Módulo]**

Informe o módulo no qual o menu em edição pertence

**[Tipo]**

Indica o tipo da nova opção que pode ser: função Protheus, SIGARPM, função de usuário, função template, relatório Crystal ou consulta genérica relacional.

**[Rotinas do browse]**

Define o acesso às opções dos menus. O Sistema apresenta 10 opções equivalentes às funções de cada programa na seqüência em que aparecem. Exemplo: em um programa de atualização, são possíveis as opções 1-Pesquisar, 2-Visualizar, 3-Incluir, 4-Alterar e 5-Excluir.

Clique sobre o número correspondente à opção que deve ter seu acesso desabilitado. Em seguida ela fica desmarcada.

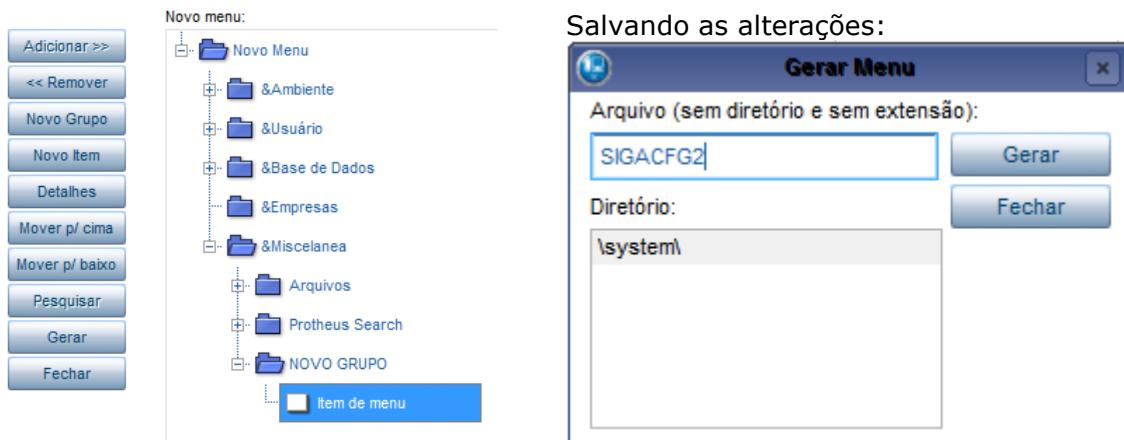
Após a configuração, quando o usuário acessar o programa correspondente e selecionar esta opção é exibida uma mensagem sobre a impossibilidade de acesso.

**[Tabelas]**

Seleciona as tabelas que serão abertas por essa opção.

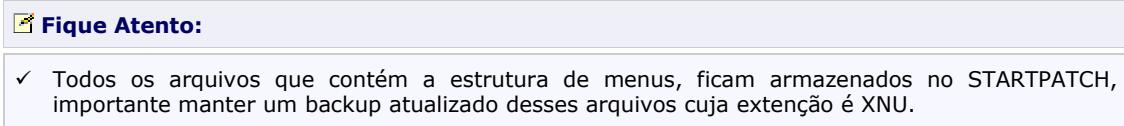
Após ter preenchido todos os dados clique em OK surgira a seguinte tela:

# Programação AdvPl 1



Clicando no botão “Gerar”, será apresentado a tela acima, no campo “Arquivo” você deve informar o nome do arquivo que será gravado em disco, pode-se utilizar um nome já existente, caso queira sobrepor o mesmo, ou criar um novo com suas opções específicas acima incluídas.

Para efetivar a gravação, clique no botão “Gerar”.



## Exercícios

Criar um novo menu com acompanhamento do professor.

Dar seu nome ao menu.

## Capítulo 15 – Conceito de filiais

O Protheus permite a criação de “empresas”, “unidades de negócio” e “filiais”, permitindo tratamento de compartilhamento de informações entre essas entidades.

### **Exemplo:**

Empresa 01 Filial 01..99

..

Empresa 99 Filial 01..99

Para cada empresa existe um jogo de Arquivos de dicionários e de tabelas também.  
Sxxnn0 -> xx – Prefixo do arquivo e nn – Código da empresa

### **Exemplo:**

Dicionário-> SX1010, SX2010, SX3010, etc

Tabelas -> SA1010, SA2010, SB1010, DA1010, etc

As filiais ficam dentro do arquivo, todo arquivo deve ter um campo filial.

xx\_FILIAL -> xx – Prefixo do arquivo, quando a tabela começa com “S”

xxx\_FILIAL -> xxx – Prefixo do Arquivo, quando a tabela não começa com “S”

Exemplo: A1\_FILIAL, A2\_FILIAL, B1\_FILIAL, DA1\_FILIAL.

## Arquivos Compartilhados

Quando o arquivo está definido como compartilhado, no campo filial será gravado espaço em branco, essa definição é feita no dicionário de tabelas SX2 no campo X2\_MODO = “C”. Assim todas as filiais enxergarão todos os registros.

### **Exemplo:**

| A1_FILIAL | A1_COD | A1_NOME   |
|-----------|--------|-----------|
|           | 000001 | CLIENTE A |
|           | 000002 | CLIENTE B |
|           | 000003 | CLIENTE C |
|           | 000004 | CLIENTE D |

## Arquivos Exclusivos

Quando o arquivo está definido como exclusivo, no campo filial será gravada a filial, essa definição é feita no dicionário de tabelas SX2 no campo X2\_MODO = “E”, Assim somente a filial enxergará o registro.

### **Exemplo:**

| A1_FILIAL | A1_COD | A1_NOME   |
|-----------|--------|-----------|
| 01        | 000001 | CLIENTE A |
| 01        | 000002 | CLIENTE B |
| 02        | 000003 | CLIENTE C |
| 01        | 000004 | CLIENTE D |

# Programação AdvPl 1

---

**XFILIAL** – Retorna a filial de um Alias específico.

**Sintaxe:** XFILIAL( *cAlias* )

| Argumento     | Obrig | TTipo | Descrição                                     |
|---------------|-------|-------|---|
| <i>cAlias</i> | Sim   | C     | Alias do arquivo que deseja retornar a filial |

**Exemplo:**

`xFilial("SA1")`

Jamais use um campo filial de uma tabela para executar um dbSeek() em outra tabela. Pois uma tabela poderá ser compartilhada (campo filial em branco), enquanto que a outra poderá ser compartilhada (campo filial preenchido). A variável *cFilAnt* contém a filial que o usuário está no momento , e a variável *cEmpant* contém a empresa e a filial.

## Capítulo 16 – Tratamento de Base de Dados

O Tratamento de base de dados em ADVPL é idêntico ao Clipper, utiliza-se as mesma funções nativas para isso. No Protheus temos algumas funções que podemos utilizar para criação de arquivos e tambem criamos a partir do módulo configurador.

### Criando Arquivos

**CRIATRAB()** Cria arquivo de trabalho, com nome temporário.

**Sintaxe:** CriaTrab( [aCampos], lExclus )

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| aCampos   | Não      | A    | É o array com os campos da tabela a criar.<br>Quando nulo apenas retorna um nome temporário e não cria a tabela. |
| lExclus   | Sim      | L    | .T. cria tabela exclusiva<br>.F. cria tabela compartilhada   |

**Exemplo:**

```
Local aCampos := {{'T_COD'      , "C", 6,0},;
                  {'T_NOME'     , "C", 30,0},;
                  {'T_ENDERECHO', "C", 30,0},;
                  {'T_DATA'     , "D", 8,0},;
                  {'T_NUMERO'   , "N", 17,2}}}

// Cria o arquivo fisicamente em DBF ou DTC no STARTPATCH
cArqTra := CriaTrab( aCampos, .T. )

// Realiza a abertura do arquivo recem criado
dbUseArea(.t.,,cArqTra,"TMP",.f.,.f.)

// Você pode selecionar a tabela como se fosse uma tabela padrão Protheus
DBSELECTAREA("TMP")
```

### Criando Índices Temporários

**INDREGUA()** Cria índice temporário.

**Sintaxe:** INDREGUA( cAlias, cArqNtx, cIndCond,[cTipo], [cFiltro], cMensagem, [lMostra] )

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cAlias    | Sim      | C    | Alias do arquivo que quer criar índice                          |
| cArqNtx   | Sim      | C    | Nome físico do índice   |
| cIndCond  | Sim      | C    | Chave do índice   |
| cTipo     | Não      | C    | Tipo de índice "C" Crescente (DEFAULT)<br>"D" Decrescente       |
| cFiltro   | Não      | C    | Filtro desejado   |
| cMensagem | Sim      | C    | Mensagem a aparecer na tela                                     |
| lMostra   | Não      | L    | .T. Mostra Régua progressão (default)<br>.F. Não mostra a régua |

# Programação AdvPl 1

## **Exemplo:**

```
// Somente gera um nome temporario pois o índice ainda não foi criado  
cArqTemp := CriaTrab( Nil, .F. )  
  
// Realizando a indexação e criando o arquivo temporário fisicamente  
IndRegua( "TMP", cArqTemp, "T_COD","C", "T_COD > '000500' ",;  
"Criando indice...", .T.)  
  
// Deleta arquivo temporário criado pelo IndRegua()  
fErase( cArqtemp + OrdBagExt() )
```

## Funções de Base de Dados

**DBAPPEND()** Inserir um registro em branco na tabela da área corrente.

**Sintaxe:** DBAPPEND()

**DBSELECTAREA()** Seleciona uma Área de Trabalho.

**Sintaxe:** DBSELECTAREA(cAlias)

| Argumento | Obrigat. | Tipo   | Descrição  |
|-----------|----------|--------|--|
| cAlias    | Sim      | C ou N | Se numérico, seleciona área pelo número, se Caracter, seleciona a área pelo Apelido. |

## **Exemplo:**

DbSelectArea("SA1")

Seleciona a área onde esta aberta a tabela SA1, realiza a abertura caso essa não esteja aberta.

**DBSETORDER()** Seleciona um índice pela ordem.

**Sintaxe:**

DBSETORDER(nOrd)

| Argumento | Obrigat. | Tipo | Descrição        |
|-----------|----------|------|------------------|
| nOrd      | Sim      | N    | Número do Índice |

## **Exemplo:**

```
DbSelectArea("SA1")  
DbSetOrder(1) // Seleciona o indice que controla a chave FILIAL + CODIGO  
DbSetOrder(2) // Seleciona o indice que controla a chave FILIAL + NOME  
DbSetOrder(0) // Desabilita a utilizacao de qualquer ordem
```

**DBGOTOP()** Posiciona o ponteiro no primeiro registro de acordo com a ordem corrente.

**Sintaxe:** DBGOTOP()

# Programação AdvPl 1

**DBGOBOTTOM()** Posiciona o ponteiro no ultimo registro de acordo com a ordem corrente.

**Sintaxe:** DBGOBOTTOM()

**DBSEEK()** Pesquisa registros em uma tabela utilizando uma determinad ordem do índice.

**Sintaxe:** DBSEEK(cChave, ISoftSeek)

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| cChave    | Sim      | C    | Chave a ser procurada  |
| ISoftSeek | Não      | L    | .T. Posiciona no registro da chave, se não encontrar posiciona no registro mais próximo a chave.<br>.F. – (Default) Posiciona no registro da chave, se não encontrar posiciona no fim do arquivo (EOF) |

**Exemplo:**

```
DbSelectArea("SA1")
DbSetOrder(1) // Seleciona o indice que controla a chave FILIAL + CODIGO
DbSeek( XFILIAL("SA1") + "000001" )
```

**DBSKIP()** Salta registro da tabela corrente.

**Sintaxe:** DBSKIP(nStep)

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| NStep     | Nao      | N    | Default 1. Qtde de registros que será saltado. Se positivo avança registros, Se negativo retrocede registros. |

**DBFILTER()** Retorna a expressão do filtro da tabela corrente.

**Sintaxe:** cFiltroAtu := DBFILTER()

**DBSETFILTER()** Ativa filtro lógico para tabela selecionada.

**Sintaxe:** DBSETFILTER(bFiltro, cFiltro)

| Argumento | Obrigat. | Tipo | Descrição                       |
|-----------|----------|------|---------------------------------|
| Bfiltro   | Sim      | B    | Bloco com a expressão de filtro |
| CFiltro   | Sim      | C    | Expressão do Filtro             |

**DBCLEARFIL()** Limpa o filtro da tabela corrente.

**Sintaxe:** DBCLEARFIL()

**DBGOTO()** Posiciona no registro especificado da tabela corrente.

**Sintaxe:** DBGOTO(nReg)

| Argumento | Obrigat. | Tipo | Descrição          |
|-----------|----------|------|--------------------|
| NReg      | Sim      | N    | Numero do Registro |

# Programação AdvPl 1

**DBUSEAREA()** Abre uma tabela e a deixa corrente (selecionada).

**Sintaxe:** DBUSEAREA( INew, cDriver, cNomArq, cAlias, IShared, IReadOnly)

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| LNew      | Não      | L    | .T. Abre em uma nova Area<br>.F. (Default) Abre na área corrente, fechando a área anteriormente utilizada. |
| CDriver   | Não      | C    | Driver da tabela a abrir. Default DBFCDX   |
| cNomArq   | Sim      | C    | Nome da tabela a abrir   |
| cAlias    | Sim      | C    | Apelido  |
| LShared   | Não      | L    | .T. Abre a tabela compartilhada<br>.F. (Default) Abre a tabela exclusiva                                   |
| IReadOnly | Não      | L    | .T. Abre a tabela somente leitura<br>.F. (Default) Permite gravação na tabela                              |

**Exemplo:**

```
Local aCampos := {{'T_COD'      , "C" ,6,0},;,  
                   {'T_NOME'     , "C" ,30,0},;  
                   {'T_ENDERECO' , "C" ,30,0}}  
cArqTra := CriaTrab( aCampos, .T. )  
dbUseArea(.t., "DBFCDX", cArqTra, "TMP", .f., .f.)
```

## Posicionamento de Registros

O posicionamento correto de registros é fundamental para a funcionalidade completa dos programas, é um grande causador de erros de lógica.

Algumas dicas para posicionamento de registros são :

**É sempre bom utilizar o Alias antes de comando de Base de Dados.**

**Exemplos:**

SB1->(DBGOTOP()) – Posiciona no primeiro registro do arquivo SB1 de acordo com a ordem que esta selecionada no momento.

SB1->(DBGOBOTTOM()) – Posiciona no ultimo registro do arquivo SB1 de acordo com a ordem que esta selecionada no momento.

SB1->(DBSEEK(XFILIAL() + "000100")) - Busca em SB1 o registro que obedeça a chave estipulada de acordo com a ordem selecionada no momento.

**Ao executar um DBSEEK(), verificar se localizou o registro, exemplo:**

```
If SB1->(dbSeek( xFilial("SB1") + "000100" )) = .F.  
  MsgInfo("Código não localizado")  
Endif
```

Sempre verifique a existência do registro, pois o programa deve prever que a base de dados pode não estar consistente, e um alerta ajuda a identificar estes casos. Em casos de relatórios, atentar-se para imprimir a mensagem de forma correta.

Se for executada a função RECLOCK(cAlias, .F.), para alteração do registro atual, em um arquivo no estado de EOF() (caso falhe um DBSEEK() ) será abortado o programa e gravado um arquivo texto de nome MSRLOCK.EOF que poderá ser usado para averiguações.

Quanto ao comando DO WHILE não esquecer de incluir a condição referente à filial (quando esta leitura for de registros de uma filial).

# Programação Advpl 1

---

## **Exemplo :**

```
dbSelectArea("SB1")
SB1->(dbSeek(xFilial("SB1")))
Do While SB1->(! Eof() .And. B1_FILIAL == xFilial("SB1"))
    // Processamento
    SB1->(dbSkip())
Enddo
```

Ao criar uma função que irá desposicionar registros, use a função GETAREA() e RESTAREA(), para voltar tudo à posição original. Exemplo:

```
Dbselectarea("SD1")
aAreaSD1 := Getarea() // Armazena o ambiente do arquivo SD1
SD1->(dbsetorder(3))
SD1->(dbseek(xfilial("SD1") + DTOS("01/03/01"), .T.))
Do While !Eof() .And. D1_FILIAL == xfilial("SD1") .And. ;
    DTOS(D1_EMISSAO) <= DTOS(mv_par02)
    // Processamento
    Dbskip()
Enddo
Restarea(aAreasd1) // Restaura o ambiente do arquivo SD1
```

## • Função Posicione

Podemos também buscar uma informação em determinado campo usando apenas uma função.

### **Sintaxe:**

Posicione(cAlias, nOrdem, cChave, cCampo)

Exemplo:

```
Posicione("SB1", 1, xFilial("SB1") + cCodigo, "B1_DESC")
```

Desta forma, será efetuada uma busca no SB1, na ordem 1, chave da busca xFilial("SB1") + cCodigo e será retornado o conteúdo do campo "B1\_DESC". Note que esta função, não restaura a posição original do arquivo alvo (no caso SB1).

É necessário colocar a FILIAL do arquivo na chave passada como parâmetro, caso ela exista na chave do índice.

## • Função Existcpo

Retorna se determinada chave existe ou não no arquivo.

Sintaxe :

ExistCpo(cAlias,cChave,nOrdem)

Exemplo :

```
ExistCpo("SB1", 1, cCodigo, "B1_DESC")
```

Desta forma, será efetuada uma busca no SB1, na ordem 1, chave cChave. E será retornado se a chave foi encontrada ou não (.T. ou ,F.). Neste caso não é necessário passar a filial. Ela será inserida automaticamente na chave de pesquisa pela função.

## Capítulo 17 – Programando

Agora podemos criar novos programas para exercitarmos o conhecimento adquirido.

### Criando um Programa Simples

Utilizando o IDE, vamos criar um relatório de produtos:

```
#INCLUDE "protheus.ch"

User Function Rela01

// Declaracao de Variaveis

Local cDesc1      := "Este programa tem como objetivo imprimir relatorio "
Local cDesc2      := "de acordo com os parametros informados pelo usuario."
Local cDesc3      := "Relatorio 1"
Local cPict        := ""
Local titulo       := "Relatorio 1"
Local nLin         := 80
Local Cabec1      := "Codigo      Descricao          Tipo"
Local Cabec2      := ""
Local imprime     := .T.
Local aOrd         := {}
Private lEnd       := .F.
Private lAbortPrint:= .F.
Private CbTxt      := ""
Private limite      := 80
Private tamanho     := "P"

// Coloque aqui o nome do programa para impressao no cabecalho
Private nomeprog := "Rela01"

Private nTipo      := 18
Private aReturn    := { "Zebrado", 1, "Administracao", 2, 2, 1, "", 1}
Private nLastKey   := 0
Private cbtxt      := space(10)
Private cbcont     := 00
Private CONTFL     := 01
Private m_pag       := 01

//Coloque aqui o nome do arquivo usado para impressao em disco
Private wnrel      := "Rela01"

Private cString := "SB1"
dbSelectArea("SB1")
dbSetOrder(1)

// Monta a interface padrao com o usuario...

Wnrel:=SetPrint(cString,
NomeProg,"RELA01",@"titulo,cDesc1,cDesc2,cDesc3,.F.,aOrd,.T.,Tamanho,,.F.)

If nLastKey == 27
    Return
```

# Programação Advpl 1

```
Endif
SetDefault(aReturn,cString)
If nLastKey == 27
    Return
Endif
nTipo := If(aReturn[4]==1,15,18)
RptStatus({|| RunReport(Cabec1,Cabec2,Titulo,nLin) },Titulo)
Return

// Funcao auxiliar chamada pela RPTSTATUS. A funcao / RPTSTATUS
// monta a janela com a regua de processamento.

Static Function RunReport(Cabec1,Cabec2,Titulo,nLin)

Local nOrdem

(cString)->(dbSetOrder(1))

// SETREGUA -> Indica quantos registros serao processados para a regua
SetRegua(RecCount())

SB1->(dbGoTop())
While SB1->(!EOF())

// Verifica o cancelamento pelo usuario...
If lAbortPrint
    @nLin,00 PSAY "*** CANCELADO PELO OPERADOR ***"
    Exit
Endif

// Impressao do cabecalho do relatorio. . .

If nLin > 55 // Salto de Página. Neste caso o formulario tem 55 linhas...
    Cabec(Titulo,Cabec1,Cabec2,NomeProg,Tamanho,nTipo)
    nLin := 8
Endif

@nLin,00 PSAY SB1->B1_COD
@nLin,17 PSAY SB1->B1_DESC
@nLin,49 PSAY SB1->B1_TIPO

nLin := nLin + 1 // Avanca a linha de impressao

dbSkip() // Avanca o ponteiro do registro no arquivo
EndDo

//Finaliza a execucao do relatorio...
SET DEVICE TO SCREEN

//Se impressao em disco, chama o gerenciador de impressao...
If aReturn[5]==1
    dbCommitAll()
    SET PRINTER TO
    OurSpool(wnrel)
Endif
MS_FLUSH()
Return
```

# Programação Advpl 1

Na criação de um relatório algumas variáveis e seus tipos são convencionados para a utilização da biblioteca de funções de relatório.

| Arguento | Ident   | Tipo | Descrição  |
|----------|---------|------|--|
| wnRel    | Local   | C    | Nome default do relatório em disco   |
| cbCont   | Local   | N    | Contador   |
| Cabec1   | Local   | C    | 1ª linha do cabeçalho do relatório   |
| Cabec2   | Local   | C    | 2ª linha do cabeçalho do relatório   |
| Cabec3   | Local   | C    | 3ª linha do cabeçalho do relatório   |
| Tamanho  | Local   | C    | Tamanho do Relatório (P = 80 colunas, M = 132 colunas, G = 220 colunas)            |
| cDesc1   | Local   | C    | 1ª linha da descrição do relatório   |
| cDesc2   | Local   | C    | 2ª linha da descrição do relatório   |
| cDesc3   | Local   | C    | 3ª linha da descrição do relatório   |
| Limite   | Local   | N    | Quantidade de colunas no relatório (80,132,220)                                    |
| Titulo   | Local   | C    | Título do Relatório  |
| aReturn  | Private | A    | Matriz com as informações para a tela de configuração de impressão                 |
| Nomeprog | Private | C    | Nome do programa do relatório  |
| cString  | Private | C    | Alias do arquivo principal do relatório para o uso de filtro                       |
| Li       | Private | N    | Controle das linhas de impressão. Seu valor inicial é a qtde de linhas por página. |
| m_pag    | Private | N    | Controle do número de páginas do relatório   |
| aOrd     | Private | A    | Matriz contendo as ordens para a impressão. Ex.:<br>aOrd:=>{"Código","Descrição"}  |
| nLastKey | Private | N    | Utilizado para controlar o cancelamento da impressão do relatório                  |
| cPerg    | Private | C    | Nome da pergunta a ser exibida para o usuário                                      |
| aLinha   | Private | A    | Matriz que contem informações para impressão de relatórios cadastrais              |

## SETPRINT – Função de impressão padrão do Protheus

### Sintaxe:

```
SetPrint( cAlias, cPrograma, [ cPerg ], [ cTitulo ], [ cDesc1 ], [ cDesc2 ], [ cDesc3 ]
], [ IDic ], [ aOrd ], [ ICompres ], [ cTam ], [ uPar1 ], IFiltro, [ ICrystal ], [
cNomeDrv ], [ uPar2 ], [ IServidor ], [ cPortaImpr ] ) -> caracter
```

| Arguento   | Ident | Tipo | Descrição   |
|------------|-------|------|---|
| cAlias     | Sim   | C    | Alias do arquivo a ser impresso.                                  |
| cPrograma  | Sim   | C    | Nome do arquivo a ser gerado em disco                             |
| cPerg      | Não   | C    | Grupo de perguntas cadastrado no dicionário SX1.                  |
| cTitulo    | Não   | C    | Título do relatório   |
| cDesc1     | Não   | C    | Descrição do relatório.   |
| cDesc2     | Não   | C    | Continuação da descrição do relatório.                            |
| cDesc3     | Não   | C    | Continuação da descrição do relatório.                            |
| IDic       | Não   | L    | Permite a escolha dos campos a serem impressos.                   |
| aOrd       | Não   | A    | Ordem(s) de impressão.  |
| ICompres   | Não   | L    | Se verdadeiro (.T.) habilita escolha o formato da impressão.      |
| cTam       | Não   | C    | Tamanho do relatório "P", "M" ou "G".                             |
| uPar1      | Não   | U    | Parâmetro reservado   |
| IFiltro    | Não   | L    | Se verdadeiro (.T.) permite a utilização do assistente de filtro. |
| ICrystal   | Não   | L    | Se verdadeiro (.T.) permite integração com Crystal Report.        |
| cNomeDrv   | Não   | C    | Nome de um driver de impressão.                                   |
| uPar2      | Não   | U    | Parâmetro reservado.  |
| IServidor  | Não   | L    | Se verdadeiro (.T.) força impressão no servidor.                  |
| cPortaImpr | Não   | C    | Define uma porta de impressão padrão.                             |

## Capítulo 18 – Tela de Padrão Microsiga

### AxCadastro()

Tela de cadastro simples, com poucas opções de customização, a qual é composta de:

- Browse padrão para visualização das informações da base de dados, de acordo com o Dicionário de Dados.(X3\_BROWSE = 'S')
  - Funções padrões para visualização de registros simples, sem a opção de cabeçalho e itens.
- ✓ axPesqui() – Tela de pesquisa de acordo com dicionario (SIX)  
✓ AxVisual() – Tela de visualização de acordo com o dicionario (SX3)  
✓ AxInclui() – Tela de inclusão de acordo com o dicionario (SX3)  
✓ AxAltera() – Tela de alteração de acordo com o dicionario (SX3)  
✓ AxDeleta() – Tela de exclusão de acordo com o dicionario (SX3)

**Sintaxe:** AxCadastro(cAlias, cTitulo, [cVldExc], [cVldAlt], [arotadic], [bpre], [bok], [aauto], [nopcauto],; [btts], [bnotts], [abuttons]

| Argumento | Obrigat. | Tipo | Descrição  |
|-----------|----------|------|--|
| cAlias    | Sim      | C    | Alias do arquivo                                   |
| cTitulo   | Sim      | C    | Titulo da Janela                                   |
| cVldExc   | Não      | C    | Nome de função para validação da exclusão          |
| cVldAlt   | Não      | C    | Nome de função para validação da tela              |
| Arrotadic | Não      | A    | Array de botões adicionais no roteiro              |
| Bpre      | Não      | B    | CodeBlock Executado antes da Interface             |
| Bok       | Não      | B    | CodeBlock Executado na validação da Interface      |
| Aauto     | Não      | A    | Array com dados para execução de rotina automática |
| Nopcauto  | Não      | N    | Opção para execução de rotina automática           |
| Btts      | Não      | B    | CodeBlock Executado dentro da transação            |
| Bnotts    | Não      | B    | CodeBlock Executado fora da transação              |
| Abuttons  | Não      | A    | CodeBlock com botões adicionais na enchoicebar     |

### Retorno:- Lógico

- .T. Clique no botão OK da tela  
.F. Clique o botão Cancela da tela

```
Exemplo:  
#include "protheus.ch"  
User Function CadSZ9()  
  
Local cAlias := "SZ9"  
Local cTitulo := "Cadastro de Ordem "  
Local cVldExc := ".T."  
Local cVldAlt := ".T."  
  
(cAlias)->(dbSetOrder(1))  
AxCadastro(cAlias,cTitulo,cVldExc,cVldAlt)  
  
Return Nil
```

# Programação AdvPl 1

## Pergunte()

Tela de parâmetros para interação com usuário. A função inicializa as variáveis de pergunta (mv\_par01,...) baseada na ordem da pergunta inserida na tabela SX1

**Sintaxe:** Pergunte( cPergunta , [IMostra] , [cTitle] )

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cPergunta | Sim      | C    | Alias do arquivo  |
| IMostra   | Não      | L    | .T. exibirá a tela para edição e carregará as perguntas<br>.F. apenas carregará as perguntas nas variáveis MV_PAR99 |
| cTitle    | Não      | C    | Título da janela de perguntas   |

Retorno:- Lógico

- .T. Clique no botão OK da tela
- .F. Clique o botão Cancela da tela

Exemplo:

```
#include "protheus.ch"
User Function ADV0001()

Local cPerg := PADR("AULAXX",10)
If Pergunte(cPerg, .T.)
    Chamafun()
Else
    MsgInfo("Usuario clicou em cancelar", "ADV0001 - Funcao modelo")
Endif

Return Nil
```

# Programação Advpl 1

## PutSx1()

Permite a inclusão de um único item de pergunta em um grupo de definido no Dicionário de Dados (SX1) via código de programação. Todos os vetores contendo os textos explicativos da pergunta devem conter até 40 caracteres por linha.

### Sintaxe:

PutSx1(cGrupo, cOrdem, cPergunt, cPerSpa, cPerEng, cVar, cTipo, nTamanho, nDecimal, nPresel,; cGSC, cValid, cF3, cGrpSxg ,cPyme,cVar01,cDef01, cDefSpa1 , cDefEng1, cCnt01, cDef02,; cDefSpa2,cDefEng2, cDef03, cDefSpa3, cDefEng3, cDef04, cDefSpa4, cDefEng4,cDef05,; cDefSpa5, cDefEng5, aHelpPor, aHelpEng, aHelpSpa, cHelp)

| Argumento | Obrigat. | Tipo | Descrição   |
|-----------|----------|------|---|
| cGrupo    | Sim      | C    | Grupo de perguntas do SX1 (X1_GRUPO)  |
| cOrdem    | Sim      | C    | Ordem do parâmetro no grupo (X1_ORDEM)  |
| cPergunt  | Sim      | C    | Descrição da pergunta em português  |
| cPerSpa   | Não      | C    | Descrição da pergunta em espanhol   |
| cPerEng   | Não      | C    | Descrição da pergunta em inglês   |
| cVar      | Sim      | C    | Nome da variável de controle auxiliar (X1_VARIAVL)                                      |
| cTipo     | Sim      | C    | Tipo do parâmetro   |
| nTamanho  | Sim      | N    | Tamanho do conteúdo do parâmetro  |
| nDecimal  | Não      | N    | Número de decimais para conteúdos numéricos   |
| nPresel   | Não      | N    | Define qual opção do combo é a padrão para o parâmetro.                                 |
| cGSC      | Sim      | C    | Define se a pergunta será do tipo G – Get ou C – Choice (combo)                         |
| cValid    | Não      | C    | Expressão de validação do parâmetro   |
| cF3       | Não      | C    | Código da consulta F3 vinculada ao parâmetro  |
| cGrpSxg   | Não      | C    | Código do grupo de campos SXG para atualização automática, quando o grupo for alterado. |
| cPyme     | Não      | C    | Se a pergunta estará disponível no ambiente Pyme  |
| cVar01    | Não      | C    | Nome da variável MV_PAR+“Ordem” do parâmetro.   |
| cDef01    | Não      | C    | Descrição da opção 1 do combo em português  |
| cDefSpa1  | Não      | C    | Descrição da opção 1 do combo em espanhol   |
| cDefEng1  | Não      | C    | Descrição da opção 1 do combo em inglês   |
| cCnt01    | Não      | C    | Conteúdo padrão ou último conteúdo definido como respostas para este item               |
| cDef0x    | Não      | C    | Descrição da opção X do combo em português  |
| cDefSpax  | Não      | C    | Descrição da opção X do combo em espanhol   |
| cDefEngx  | Não      | C    | Descrição da opção X do combo em inglês   |
| aHelpPor  | Não      | A    | Vetor simples contendo as linhas de help em português para o parâmetro.                 |
| aHelpEng  | Não      | A    | Vetor simples contendo as linhas de help em inglês para o parâmetro.                    |
| aHelpSpa  | Não      | A    | Vetor simples contendo as linhas de help em espanhol para o parâmetro.                  |
| cHelp     | Não      | C    | Conteúdo do campo X1_HELP   |

## mBrowse()

Monta um browse padrão do sistema, conforme os parâmetros, de acordo com dicionario.

### Sintaxe:

MBrowse(nt, nl, nb, nr, calias, afixe, ccpo, nposi, cfun, ndefault, acolors, ; ctopfun, cbotfun, nfreeze, bparbloco, lnotopfilter, lseeall, lchgall, cexprfiltop)

# Programação Advpl 1

|    | Argumento    | Obrigat. | Tipo | Descrição  |
|----|--------------|----------|------|--|
| 1  | NT           | Não      | N    | Linha Inicial  |
| 2  | NI           | Não      | N    | Coluna Inicial   |
| 3  | Nb           | Não      | N    | Linha Final  |
| 4  | Nr           | Não      | N    | Coluna Final   |
| 5  | Calias       | Sim      | C    | Alias do arquivo que será visualizado no browse.<br>Para utilização de arquivos de trabalho, o nome do alias deve ser obrigatoriamente 'TRB' e o parâmetro aFixe torna-se obrigatório.   |
| 6  | Afixe        | Não      | A    | Array bi-dimensional contendo os nomes dos campos fixos pré-definidos.<br>A estrutura do array é diferente para arquivos que fazem parte do dicionário de dados e para arquivos de trabalho.<br>Arquivos que fazem parte do dicionários de dados<br>[n][1]=>Descrição do campo<br>[n][2]=>Nome do campo<br>Arquivos de trabalho<br>[n][1]=>Descrição do campo<br>[n][2]=>Nome do campo<br>[n][3]=>Tipo<br>[n][4]=>Tamanho<br>[n][5]=>Decimal<br>[n][6]=>Picture  |
| 7  | Ccpo         | Não      | C    | Campo a ser validado se está vazio ou não para exibição do bitmap de status.<br>Quando esse parâmetro é utilizado, a primeira coluna do browse será um bitmap indicando o status do registro, conforme as condições configuradas nos parâmetros cCpo, cFun e aColors.  |
| 8  | Nposi        | Não      | N    |  |
| 9  | Cfun         | Não      | C    | Função que retornará um valor lógico para exibição do bitmap de status.<br>Quando esse parâmetro é utilizado, o parâmetro cCpo é automaticamente desconsiderado.   |
| 10 | Ndefault     | Não      | N    | Número da opção do aRotina que será executada quando for efetuado um duplo clique em um registro do browse. O default é executar a rotina de visualização.   |
| 11 | Acolors      | Não      | A    | Array bi-dimensional para possibilitar o uso de diferentes bitmaps de status.<br>[n][1]=>Função que retornará um valor lógico para a exibição do bitmap<br>[n][2]=>Nome do bitmap que será exibido quando a função retornar .T.<br>(True).<br>O nome do bitmap deve ser um resource do repositório e quando esse parâmetro é utilizado os parâmetros cCpo e cFun são automaticamente desconsiderados.  |
| 12 | Ctopfun      | Não      | C    | Função que retorna o limite superior do filtro baseado na chave de índice selecionada.<br>Esse parâmetro deve ser utilizado em conjunto com o parâmetro cBotFun.   |
| 13 | Cbotfun      | Não      | C    | Função que retorna o limite inferior do filtro baseado na chave de índice selecionada.<br>Esse parâmetro deve ser utilizado em conjunto com o parâmetro cTopFun.   |
| 14 | Nfreeze      | Não      | N    | Qtde de colunas que ficarão congeladas   |
| 15 | Bparbloco    | Não      | B    | Bloco de código que será executado no ON INIT da janela do browse. O bloco de código receberá como parâmetro o objeto da janela do browse.   |
| 16 | Lnotopfilter | Não      | L    | Valor lógico que define se a opção de filtro será exibida no menu MBrowse.<br>.T. => Não exibe a opção no menu<br>.F. => (default) Exibe a opção no menu.<br>A opção de filtro na MBrowse está disponível apenas para TopConnect.  |
| 17 | Lseeall      | Não      | L    | Identifica se o Browse deverá mostrar todas as filiais.<br>O valor default é .F. ( False ), não mostra todas as filiais. Caso os parâmetros cTopFun ou cBotFun sejam informados esse parâmetro será configurado automaticamente para .F. ( False )<br>Parâmetro válido à partir da versão 8.11.<br>A função SetBrwSeeAll muda o valor default desse parâmetro.   |
| 18 | Lchgall      | Não      | L    | Identifica se o registro de outra filial está autorizado para alterações.<br>O valor default é .F. ( False ), não permite alterar registros de outras filiais.<br>Quando esse parâmetro está configurado para .T. ( True ), o parâmetro lSeeAll é configurado automaticamente para .T. ( True ).<br>Caso os parâmetros cTopFun ou cBotFun sejam informados esse parâmetro será configurado automaticamente para .F. ( False ).<br>Parâmetro válido à partir da versão 8.11.<br>A função SetBrwChgAll muda o valor default desse parâmetro. |
| 19 | Cexprfiltop  | Não      | C    | Expressão para efetuar o filtro no browse com sintaxe em TopConnect.   |

# Programação Advpl 1

**cCadastro**- Título que será exibido do browse.

**aRotina** - Array contendo as funções que serão executadas pela Mbrowse, nele será definido o tipo de operação a ser executada (inclusão, alteração, exclusão, visualização, pesquisa, etc.), e sua estrutura é composta de 5 (cinco) dimensões:

- [n][1] - Título;
- [n][2] - Rotina;
- [n][3] - Reservado;
- [n][4] - Operação (1-pesquisa; 2-visualização; 3-inclusão; 4-alteração; 5-exclusão);

Exemplo:

```
#include "protheus.ch"
User Function brw00001()

Private cCadastro := "Cadastro de Exemplo"
Private aRotina := {}
AADD(aRotina,{"Pesquisar" , "AxPesqui",0,1})
AADD(aRotina,{"Visualizar" , "AxVisual",0,2})
AADD(aRotina,{"Incluir" , "AxInclui",0,3})
AADD(aRotina,{"Alterar" , "AxAltera",0,4})
AADD(aRotina,{"Excluir" , "AxDeleta",0,5})

Private cString := "SA1"

SA1->(dbSetOrder(1))
mBrowse( ,,,cString)

Return
```

## Modelo 2

Tela de Manutenção Utilizando uma Tabela (Cabecalho/Itens).

### Sintaxe:

Modelo2(cTitulo,aCab,aRod,aGetdados,[nOpc],[cLinOk],[cTudOk],aCols,[bF4],[cInitPos],[nMax],[aCoord],[lDelGet],[lMaximized],[aButtons])

|    | Argumento  | Obrigat. | Tipo | Descrição   |
|----|------------|----------|------|---|
| 1  | cTitulo    | Sim      | C    | Título da Janela  |
| 2  | aCab       | Sim      | A    | Array com dados do cabeçalho<br>Ex:{cVariavel,{nLin,nCol},cTitulo_Campo,cPicture,Funcao_valid,F3,lEditavel} |
| 3  | aRod       | Sim      | A    | Array com dados do rodapé<br>Ex:{cVariavel,{nLin,nCol},cTitulo_Campo,cPicture,Funcao_valid,F3,lEditavel}    |
| 4  | aGetdados  | Sim      | A    | Array com posições da getdados<br>Ex:{nLinIni,nColIni,nLinFim,nColFim}                                      |
| 5  | nOpc       | Nao      | N    | Modo de operação (3/4 Inclui ou altera, 6 altera, outros visualiza)<br>Ex: nOpc := 3                        |
| 6  | cLinOk     | Não      | C    | Função para validar linha - Ex: "allwaystrue()"   |
| 7  | cTudOk     | Não      | C    | Função que valida tudo - Ex: "allwaystrue()"  |
| 8  | aCols      | Sim      | A    | Array com os campos que podem ser alterados<br>Ex:{“A1_COD”,“A1_NREDUZ”}                                    |
| 9  | bF4        | Não      | B    | Bloco de Código Chamado pelo F4 - Ex: {  MsgAlert("Teste")}   |
| 10 | cInitPos   | Não      | C    | String com nome do campo que deve ser inicializado ao teclar seta para baixo.                               |
| 11 | nMax       | Não      | N    | Número máximo de linhas   |
| 12 | aCoord     | Não      | A    | Array com as coordenadas das janelas  |
| 13 | lDelGet    | Não      | L    | Determina se a linha da getdados pode ser apagada ou não  |
| 14 | lMaximized | Não      | L    | Maximiza a tela   |
| 15 | aButtons   | Não      | A    | Array com botões a serem acrescentados no toolbar   |

Retorno:- Lógico

- .T. Clique no botão OK da tela
- .F. Clique o botão Cancela da tela

# Programação Advpl 1

## Modelo 3

Tela de Manutenção Utilizando duas Tabelas (Cabecalho/Itens).

|    | Argumento    | Obrigat. | Tipo | Descrição  |
|----|--------------|----------|------|--|
| 1  | cTitulo      | Sim      | C    | Título da Janela                                 |
| 2  | cAlias1      | Sim      | C    | Alias da tabela do Cabeçalho                     |
| 3  | cAlias2      | Sim      | C    | Alias da Tabela dos Itens                        |
| 4  | aEnchoice    | Sim      | A    | Array com os campos da enchoice(Cabecalho)       |
| 5  | cLinOk       | Não      | C    | Função de Validação de Itens                     |
| 6  | cTudOK       | Não      | C    | Função de Validação tudo                         |
| 7  | nOpc1        | Não      | N    | Opção de edição da Enchoice                      |
| 8  | nOpc2        | Não      | N    | Opção de edição da Getdados                      |
| 9  | cFieldOk     | Não      | C    | Função de validação dos campos da Getdados       |
| 10 | lVirtual     | Não      | L    | Permite visualizar campos virtuais na Enchoice   |
| 11 | nLinhas      | Não      | N    | Numero Maximo de Linhas da Getdados              |
| 12 | aAltEnchoice | Não      | A    | Array com os campos alteráveis da enchoice       |
| 13 | nFreeze      | Não      | N    | Qtde de colunas que ficarão congeladas           |
| 14 | aButtons     | Não      | A    | Array com botões a serem acrescentados a toolbar |
| 15 | aCoord       | Não      | A    | Array com as coordenadas da tela                 |
| 16 | nSize        | Não      | N    | Numero de linhas da enchoice                     |

Retorno:- Lógico

- .T. Clique no botão OK da tela
- .F. Clique o botão Cancela da tela

Exemplo:

```
#include "protheus.ch"
User Function CadM3()
Private cAlias1      := "SC5"
Private cAlias2      := "SC6"
Private aHeader       := {}
Private aCols         := {}
Private cCadastro    := "Modelo 3"
Private aRotina       := {}
AAdd(aRotina, {"Pesquisar" , "axPesqui" , 0, 1})
AAdd(aRotina, {"Visualizar" , "U_Incmod3" , 0, 2})
AAdd(aRotina, {"Incluir"   , "u_Incmod3" , 0, 3})
AAdd(aRotina, {"Altera"    , "u_Incmod3" , 0, 4})
AAdd(aRotina, {"Excluir"   , "u_Incmod3" , 0, 5})
mBrowse(,,,cAlias1)
Return
=====
User Function Incmod3(cAlias, nReg, nOpc)
Local nAdi      := 0
Local i       := 0
Private j:=1
//Criar as variaveis de memoria
RegToMemory(cAlias1, (nOpc==3))
RegToMemory(cAlias2, (nOpc==3))
aHeader       := {}
aCols         := {}
//Monta aheader
dbSelectArea("SX3")
dbSetorder(1)
dbSeek(cAlias2)

While SX3->(!EOF()) .AND. SX3->X3_ARQUIVO == cAlias2
  If X3USO(SX3->X3_USADO) .AND. cNIVEL >= SX3->X3_NIVEL // .AND. ALLTRIM(SX3->X3_CAMPO) $ "C6_ITEM/C6_PRODUTO/C6_UM/C6_QTDVEN/C6_PRCVEN/C6_VALOR"
    AAdd (aHeader, {Trim(SX3->X3_TITULO),;01 - titulo}
```

# Programação Advpl 1

```
SX3->X3_CAMPO    ; //02- nome do campo
SX3->X3_PICTURE   ; //03 - mascara do campo
SX3->X3_TAMANHO   ; //04 - tamanho
SX3->X3_DECIMAL   ; //05 - decimais
SX3->X3_VALID     ; //06 - validacao
SX3->X3_USADO     ; //07 - USADO
SX3->X3_TIPO      ; //08 - TIPO DO CAMPO
SX3->X3_ARQUIVO   ; //09 - ALIAS
SX3->X3_CONTEXT}  //10 - Virtual ou Real
ENDIF
SX3->(DBSKIP())
ENDDO
//Monta acols
If nOpc == 3 //Inclusão
    AAdd(aCols, Array(Len(aHeader) + 1))
    For nAdi:= 1 to Len(aHeader)
        aCols[1][nADI] := Criavar(aHeader[nADI][2])
    Next
    ACols[1][len(aHeader)+1] := .F.

Else //Visual/Altera/Exclui
    (cAlias2)->(dbSetOrder(1))
    (cAlias2)->(dbSeek(xFilial(cAlias2) + (cAlias1)->C5_NUM))
    While (cAlias2)->(!EOF()).And.(cAlias2)->C6_FILIAL == xFilial(cAlias2) .And. ;
        (cAlias2)->C6_NUM == (cAlias1)->C5_NUM
        AAdd(aCols, Array(len(aHeader)+1))
        For i := 1 To Len(aHeader)
            If aHeader[i][10] <> "V"
                aCols[len(aCols)][i] := (cAlias2)->&(aHeader[i][2])
            Else
                aCols[len(aCols)][i] := CriaVar(aHeader[i][2])
            EndIf
        Next
        ACols[j][len(aHeader)+1] := .F.
        (cAlias2)->(dbSkip())
        j++
    Enddo
Endif
IRet := Modelo3(cCadastro, cAlias1, cAlias2, , "Allwaystrue", "Allwaystrue" , nOpc, nOpc)

Return
```

## Exercício

Fazer um programa que trate a Amarracao Cliente x Produto  
Tabelas Envolvidas:

- Cadastro de Clientes (SA1) , sempre visual (não permite alteracao)
- Amarracao Produtos x Clientes (SA7) Obedece a escolha do usuario no browse
- Browse (Cadastro de Clientes)
- Itens do aRotina
- Incluir, Alterar ,visualizar , excluir os dados

### Validações:

Pesquisa Padrão (AXPESQUI)

Não permitir incluir o mesmo produto para o mesmo cliente.

Não permitir alteração caso não exista nenhum item amarrado para o cliente selecionado.

Na exclusão, excluir apenas os itens ( SA7), Não deixar excluir caso não exista nenhum item amarrado para o cliente selecionado.

Trazer o acols já preenchido conforme registro posicionado pelo usuário.

## Capítulo 19 – MSEXECAUTO

Fazer manutenção automática (inclusão, alteração e exclusão) das rotinas de manipulação de dados do sistema, automatizando o processo de entrada de dados sem a necessidade de desenvolver rotinas específicas.

### Vantagens

- 1) **Interface** : Os dados de entrada são enviados a rotina em forma de campos e conteúdos (array) e desta forma não é necessário a apresentação de nenhuma interface ao usuário.
- 2) **Segurança** : A utilização de rotinas automáticas aumenta consideravelmente a segurança do sistema, uma vez que utiliza as validações padrões e diminui os problemas causados por atualização de versão ou inclusão de customizações nas rotinas padrões do sistema.
- 3) **Agilidade no processo** : Aumenta consideravelmente o tempo de desenvolvimento das customizações que necessitam de entrada de dados. Exemplo: Importação de pedido de venda.

### Procedimentos

**Sintaxe:** MSEXecAuto({|x,y|fpgmpad(x,y)},aCampos,nOpc)

| Argumento | Obrigat. | Tipo | Descrição                           |
|-----------|----------|------|-------------------------------------|
| fPgmPad   | Sim      | B    | Nome da Função Padrão               |
| aCampos   | Sim      | A    | Array com os campos a gravar        |
| nOpc      | Não      | N    | Opção do Browse que será utilizada. |

Existem duas maneiras de utilizar a rotina automática, sendo elas:

- Sem Interface

Para a utilização da rotina automática sem interface deve-se, configurar o ambiente utilizando-se o comando PREPARE ENVIRONMENT e chamar diretamente o nome da função.

```

Exemplo:
#include "protheus.ch"
User Function IncProd()
Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private lMsHelpAuto := .t. // se .t. direciona as mensagens de help para o arq. de log
Private lMsErroAuto := .f. //necessario a criacao, pois sera atualizado quando houver
                           //alguma inconsistencia nos parametros
PREPARE ENVIRONMENT EMPRESA '99' FILIAL '01' MODULO 'FAT'
aRotAuto:= {{'B1_COD' ,GETSXENUM('SB1', 'B1_COD') ,Nil},,
{'B1_DESC' , 'Produto teste',Nil},,
{'B1_TIPO' , 'PA' ,Nil},,
{'B1_UM' , 'UN' ,Nil},,
{'B1_LOCPAD' , '01' ,Nil},,
{'B1_PICM' , 0 ,Nil},,
{'B1_IPI' , 0 ,Nil},,
{'B1_PRV1' , 100 ,Nil},,
{'B1_LOCALIZ','N' ,Nil},,
{'B1_CODBAR' , '789888800001' ,Nil}}
MSEXecAuto({|x,y| mata010(x,y)},aProduto,nOpc)
If lMsErroAuto
   //mostrar na tela o log informando qual a inconsistencia ocorreu durante a rotina
   Mostraerro()
   Return .F.
EndIf
Return .T.

```

- Com Interface

# Programação Advpl 1

Para a utilização da rotina automática com interface deve-se apenas colocar no menu.

```
Exemplo:  
#include "protheus.ch"  
User Function IncProd()  
  
Local aRotAuto := {}  
Local nOpc := 3 // inclusao  
Private lMsHelpAuto := .t. // se .t. direciona as mensagens de help para o arq. de log  
Private lMsErroAuto := .f. //necessario a criacao, pois sera atualizado quando houver  
//alguma incosistencia nos parametros  
  
aRotAuto:= {{'B1_COD' ,GETSXENUM('SB1', 'B1_COD') ,Nil},;  
{'B1_DESC' , 'Produto teste',Nil},;  
{'B1_TIPO' , 'PA' ,Nil},;  
{'B1_UM' , 'UN' ,Nil},;  
{'B1_LOC PAD' , '01' ,Nil},;  
{'B1_PICM' , 0 ,Nil},;  
{'B1_IPI' , 0 ,Nil},;  
{'B1_PRV1' , 100 ,Nil},;  
{'B1_LOCALIZ' , 'N' ,Nil},;  
{'B1_CODBAR' , '78988880001' ,Nil}}  
MSExecAuto({{|x,y| mata010(x,y)}},aProduto,nOpc)  
  
If lMsErroAuto  
//mostrar na tela o log informando qual a inconsistência ocorreu durante a rotina  
Mostraerro()  
Return .f.  
EndIf  
Return .t.
```

**Onde usar:** Existem varias situações em que podemos usar:

- Customizações de rotinas que automatizam processos entre empresas e facilitam o trabalho do usuário.
- Importação de dados para o sistema.
  - ✓ Arquivo texto,
  - ✓ XML,
  - ✓ Sites da Web,
  - ✓ ADVPL ASP,
  - ✓ Web Services.
  - ✓ Workflow
- Replicação de Dados, como cópia de pedidos, produtos, clientes, fornecedores, liberação de pedidos, etc

## Exemplo de Automação de Rotinas

# Programação Advpl 1

```
Tipo de Entrada e Saída:  
#include "protheus.ch"  
User Function IncTes()  
Local aRotAuto := {}  
Local nOpc := 3 // inclusão  
Private lMsHelpAuto := .t.  
Private lMsErroAuto := .f.  
//Entrada  
aTes:=>{"F4_CODIGO" , "011" , Nil},;  
 {"F4_TIPO" , "E" , Nil},;  
 {"F4_ICM" , "S" , Nil},;  
 {"F4_IPI" , "S" , Nil},;  
 {"F4_CREDICM" , "S" , Nil},;  
 {"F4_CREDIPI" , "S" , Nil},;  
 {"F4_DUPPLIC" , "S" , Nil},;  
 {"F4_ESTOQUE" , "S" , Nil},;  
 {"F4_CF" , "1102" , Nil},;  
 {"F4_TEXTO" , "COMPRA", Nil},;  
 {"F4_PODER3" , "N" , Nil},;  
 {"F4_LFICM" , "T" , Nil},;  
 {"F4_LFIPI" , "T" , Nil},;  
 {"F4_DESTACA" , "N" , Nil},;  
 {"F4_INCIDE" , "N" , Nil},;  
 {"F4_COMPL" , "N" , Nil}}  
MSExecAuto({|x,y| mata080(x,y)},aTes,nOpc)  
//Saída  
aTes:=>{"F4_CODIGO" , "511",Nil},;  
 {"F4_TIPO" , "S" , Nil},;  
 {"F4_ICM" , "S" , Nil},;  
 {"F4_IPI" , "S" , Nil},;  
 {"F4_CREDICM" , "S" , Nil},;  
 {"F4_CREDIPI" , "S" , Nil},;  
 {"F4_DUPPLIC" , "S" , Nil},;  
 {"F4_ESTOQUE" , "S" , Nil},;  
 {"F4_CF" , "5102" , Nil},;  
 {"F4_TEXTO" , "COMPRA", Nil},;  
 {"F4_PODER3" , "N" , Nil},;  
 {"F4_LFICM" , "T" , Nil},;  
 {"F4_LFIPI" , "T" , Nil},;  
 {"F4_DESTACA" , "N" , Nil},;  
 {"F4_INCIDE" , "N" , Nil},;  
 {"F4_COMPL" , "N" , Nil}}  
MSExecAuto({|x,y| mata080(x,y)},aTes,nOpc)  
If lMsErroAuto  
Mostraerro()  
Return .f.  
EndIf  
Return .t.
```

# Programação AdvPl 1

```
Documento de Entrada
#include "protheus.ch"
User Function Incnfe()

Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private lMsHelpAuto := .t.
Private lMsErroAuto := .f.
aCab := {{"F1_TIPO", "N" ,NIL},;
 {"F1_FORMUL", "N" ,NIL},;
 {"F1_DOC", "000001" ,NIL},;
 {"F1_SERIE", "UNI" ,NIL},;
 {"F1_EMISSAO", dDataBase ,NIL},;
 {"F1_FORNECE", "000001" ,NIL},;
 {"F1_LOJA", "01" ,NIL},;
 {"F1_COND", "001" ,NIL},;
 {"F1_ESPECIE", "NF" ,NIL}};

For nx:=1 to 3
    aLinha:={}
    AADD(aLinha, {"D1_COD" , "EX"+Str(nx,1,0) ,NIL})
    AADD(aLinha, {"D1_UM" , "PC" ,NIL})
    AADD(aLinha, {"D1_QUANT", 10000 ,NIL})
    AADD(aLinha, {"D1_VUNIT", 1 ,NIL})
    AADD(aLinha, {"D1_TOTAL", 10000 ,NIL})
    AADD(aLinha, {"D1_TES" , "001" ,NIL})
    AADD(aLinha, {"D1_LOCAL", "01" ,NIL})
    AADD(aItens,aLinha)
Next nx
MSExecAuto({|x,y,z| MATA103(x,y,z)},aCab,aItens,nOpc)
If lMsErroAuto
Mostraerro()
Return .f.
EndIf
Return .t.
```

## Fornecedores

```
#include "protheus.ch"

User Function IncFor()
Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private lMsHelpAuto := .t.
Private lMsErroAuto := .f.

aFornecedor:={{"A2_COD" , GETSXENUM('SA2' , 'A2_COD') ,Nil},;
 {"A2_LOJA" , "01" ,Nil},;
 {"A2_NOME" , "FORNECEDOR VENDE BARATO S.A." ,Nil},;
 {"A2_NREDUZ", "VENDE TUDO" ,Nil},;
 {"A2_TIPO" , "F" ,Nil},;
 {"A2_END" , "AV LEONARDO DA VINCE, 608" ,Nil},;
 {"A2_MUN" , "SAO PAULO" ,Nil},;
 {"A2_EST" , "SP" ,Nil},;
 {"A2_COND" , "001" ,Nil}};

MSExecAuto({|x,y| mata020(x,y)},aFornecedor,nOpc)

If lMsErroAuto
Mostraerro()
Return .f.
EndIf

Return .t.
```

```
Clientes

#include "protheus.ch"

User Function IncCli()

Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private lMsHelpAuto := .t.
Private lMsErroAuto := .f.

aCliente:={{"A1_COD"      ,GETSXENUM('SA1', 'A1_COD'),Nil},;
           {"A1_LOJA"     , "01"                  ,Nil},;
           {"A1_NOME"     , "CLIENTE COMPRA TUDO S.A." ,Nil},;
           {"A1_NREDUZ"   , "COMPRA TUDO"          ,Nil},;
           {"A1_TIPO"     , "F"                   ,Nil},;
           {"A1_END"      , "RUA DO ROCIO 123"    ,Nil},;
           {"A1_MUN"      , "SAO PAULO"           ,Nil},;
           {"A1_EST"      , "SP"                 ,Nil},;
           {"A1_COND"     , "001"                ,Nil}}}

MSExecAuto({|x,y| mata030(x,y)},aCliente,nOpc)

If lMsErroAuto
  Mostraerro()
  Return .f.
EndIf

Return .t.
```

## Exercícios

1. Criar uma rotina que inclua automaticamente pelo MSEXECAUTO um pedido de vendas com 2 itens quaisquer.

Funcao de inclusao: MATA410()

Tabelas: SC5 e SC6

2. Criar uma rotina automática para o cadastro de Clientes, para isso crie um grupo de perguntas com as seguintes perguntas:

Nome, Endereco, CNPJ, telefone, tipo

- Use o MSEexecauto com esses campos.
- Coloque no aRotina do Browse do exercício anterior.
- Lembre-se de gravar o código usando a função GETSXENUM()
- Mostre na tela o código gerado