Protheus 10

Educação Corporativa

Lógica e Inicialização à Programação ADVPL

microsiga

Matriz - Av. Braz Leme, 1.717 - 02511-000 - São Paulo - SP - Brasil.
Tel.: 55 (11) 3981 - 7001 www.microsiga.com.br

Sumário

Processamento de dados

Os problemas que serão resolvidos através da programação são baseados no conceito básico de processamento de dados. As informações são recebidas pelo programa e são processadas, gerando uma resposta no final. A primeira análise que devemos fazer para buscar a solução é identificar quais informações são realmente necessárias (dados de entrada) e quais são as respostas que a solução pede (dados de saída).

Entrada, processamento (algoritmo) e saída

Observação: Todo o programa, independente da sua finalidade, trabalha dessa forma. Ou seja: entrada, processamento e saída.

O primeiro trabalho do programador é, a partir do problema proposto, dividir as informações que serão manipuladas pelo programa em informações de entrada e saída e determinar, em linhas gerais, quais serão os procedimentos para chegar à solução do problema. A definição destes procedimentos é o que chamamos de Algoritmo, e o processamento é a execução destes procedimentos, que transformam e manipulam os dados de entrada, resultando os dados de saída.

Dentre as formas de representação de algoritmos, podemos citar o fluxograma convencional.

Linguagem de programação

Para um programa ser desenvolvido, é necessário que o programador use "um programa", que programa? Uma linguagem de programação. Linguagem de programação nada mais é do que um programa que permite ao programador criar programas.

Existem duas categorias de linguagem de programação:

Linguagem de baixo nível. Nesta categoria, o programador trabalha com códigos "complexos", ditos "linguagem de máquina".

Linguagem de alto nível. Nesta categoria, o programador trabalha com códigos "mais simples", pois os códigos são em inglês, ou seja, fica muito mais fácil a compreensão.
Exemplos de linguagem de programação: C, C++, Visual basic, Advpl e etc.

Programa fonte e executável

Quando o programador digita seus programas, nós podemos chamar o mesmo de programa fonte.
Obs.: Apenas o programador, analista tem acesso ao programa fonte. Salvo exceções.

O usuário final terá acesso ao programa executável. Ou seja, as funcionalidades do sistema em si.

Variáveis

Variáveis são "espaços reservados na memória do computador para armazenar dados durante o processamento do programa".

Quando se declara, "cria-se" a variável, é necessário definir, quais serão os tipos de dados que a mesma manipulará.

Por exemplo: numérico, caractere (string, alfanumérico), lógico ou data.

Também e necessário informar, se a variável será: local, privada, pública, statica ou global.

O Contexto de Variáveis dentro de um Programa

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

Local nNumero := 10

Esta linha de código declara uma variável chamada nNumero indicando que seu escopo é local.

Os identificadores de escopo são:

- LOCAL
- STATIC
- PRIVATE
- PUBLIC

Variáveis Locais

Variáveis locais são pertencentes apenas ao escopo da função onde foram declaradas. Devem ser explicitamente declaradas com o identificador LOCAL, como no exemplo:

```
Function Pai()
Local nVar := 10, aMatriz := {0,1,2,3}
<comandos>
Filha()
<mais comandos>
Return(.T.)
```

Neste exemplo, a variável nVar foi declarada como local e atribuída com o valor 10. Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada. Quando a execução da função Pai terminar, a variável nVar é destruída. Qualquer variável com o mesmo nome no programa que chamou a função Pai não é afetada.

Variáveis locais são criadas automaticamente cada vez que a função onde forem declaradas for ativada. Elas continuam a existir e mantêm seu valor até o fim da ativação da função (ou seja, até que a função retorne o controle para o código que a executou). Se uma função é chamada recursivamente (por exemplo, chama a si mesma), cada chamada em recursão cria um novo conjunto de variáveis locais.

A visibilidade de variáveis locais é idêntica ao escopo de sua declaração. Ou seja, a variável é visível em qualquer lugar do código fonte em que foi declarada. Se uma função é chamada recursivamente, apenas as variáveis locais criadas na mais recente ativação são visíveis.

Variáveis Estáticas

Variáveis estáticas funcionam basicamente como as variáveis locais, mas mantêm seu valor através da execução. Variáveis estáticas devem ser declaradas explicitamente no código com o identificador STATIC.

O escopo das variáveis estáticas depende de onde são declaradas. Se forem declaradas dentro do corpo de uma função ou procedimento, seu escopo será limitado àquela rotina. Se forem declaradas fora do corpo de qualquer rotina, seu escopo é todo o arquivo de programa.

Neste exemplo, a variável nVar é declarada como estática e inicializada com o valor 10:

```
Function Pai()
Static nVar := 10
<comandos>
Filha()
<mais comandos>
.Return(.T.)
```

Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada. Diferente de variáveis declaradas como LOCAL ou PRIVATE, nVar continua a existir e mantém seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subseqüentes da função Pai.

Variáveis Privadas

A declaração é opcional para variáveis privadas. Mas podem ser declaradas explicitamente com o identificador PRIVATE.

Uma vez criada, uma variável privada continua a existir e manchem seu valor até que o programa ou função onde foi criada termine (ou seja, até que a função onde foi criada retorne para o código que a executou). Neste momento, é automaticamente destruída.

Em termos mais simples, uma variável privada é visível dentro da função de criação e todas as funções chamadas por esta, a menos que uma função chamada crie sua própria variável privada com o mesmo nome.

Por exemplo:

```
Function Pai()
Private nVar := 10
<comandos>
Filha()
<mais comandos>
Return(.T.)
```

Neste exemplo, a variável nVar é criada como privada e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e, diferente de uma variável local, pode ser acessada pela função Filha.

Quando a função Pai terminar, nVar será destruída e qualquer declaração de nVar anterior se tornará acessível novamente.

Variáveis Públicas

Podem-se criar variáveis públicas dinamicamente no código com o identificador PUBLIC. As variáveis públicas continuam a existir e mantêm seu valor até o fim da execução.

É possível criar uma variável privada com o mesmo nome de uma variável pública existente. Entretanto, não é permitido criar uma variável pública com o mesmo nome de uma variável privada existente.

Uma vez criada, uma variável pública é visível em todo o programa onde foi declarada até que seja escondida por uma variável privada criada com o mesmo nome. A nova variável privada criada esconde a variável pública existente, e esta se tornará inacessível até que a nova variável privada seja destruída. Por exemplo:

```
Function Pai()
Public nVar := 10
<comandos>
Filha()
<mais comandos>
Return(.T.)
```

Neste exemplo, nVar é criada como pública e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e pode ser acessada. Diferente de variáveis locais ou privadas, nVar ainda existe após o término da à execução da função Pai.

• Diferentemente dos outros identificadores de escopo, quando uma variável é declarada como pública sem ser inicializada, o valor assumido é falso (.F.) e não nulo (nil).

Variáveis e nomes de campos

Muitas vezes uma variável pode ter o mesmo nome que um campo de um arquivo ou tabela aberto 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.

```
cRes := MEMVAR->NOME
```

Esta linha de comando identifica que o valor atribuído à variável cRes deve ser o valor da variável de memória chamada NOME.

```
cRes := FIELD->NOME
```

Neste caso, o valor atribuído à variável cRes será o valor do campo NOME existente no arquivo ou tabela aberto na área atual.

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.

cRes := CLIENTES->NOME

Operadores Comuns

Na documentação sobre variáveis há uma breve demonstração de como atribuir valores a uma variável da forma mais simples. O AdvPl amplia significativamente a utilização de variáveis através do uso de expressões e funções. Uma expressão é um conjunto de operadores e operando cujo resultado pode ser atribuído a uma variável ou então analisado para a tomada de decisões. Por exemplo:

```
Local nSalario := 1000, nDesconto := 0.10
Local nAumento, nSalLiquido
nAumento := nSalario * 1.20
nSalLiquido := nAumento * (1-nDesconto)
```

Neste exemplo são utilizadas algumas expressões para calcular o salário líquido após um aumento. Os operando de uma expressão podem ser uma variável, uma constante, um campo de arquivo ou uma função.

Operadores Matemáticos

Os operadores utilizados em AdvPl para cálculos matemáticos são:

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

Operadores de String

Os operadores utilizados em AdvPl para tratamento de caracteres são:

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

Operadores Relacionais

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

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

Operadores Lógicos

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

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

Operadores de Atribuição

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

=	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

Atribuição Simples

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

nVariavel = 10

Atribuição em Linha

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

nVar1 := nVar2 := nVar3 := 0

Quando diversas variáveis são inicializada em uma mesma linha, a atribuição começa da direita para a esquerda, ou seja, nVar3 recebe o valor zero inicialmente, nVar2 recebe o conteúdo de nVar3 e nVar1 recebe o conteúdo de nVar2 por final.

Com o operador de atribuição em linha, podem-se substituir as inicializações individuais de cada variável por uma inicialização apenas:

Local nVar1 := 0, nVar2 := 0, nVar3 := 0

por

Local nVar1 := nVar2 := nVar3 := 0

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

Atribuição Composta

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

Operador	Exemplo	Equivalente a
+=	X += Y	X = X + Y
-=	X -= Y	X = X – Y
*=	X *= Y	X = X * Y
/=	X /= Y	X = X / Y
**= o u ^=	X **= Y	X = X ** Y
%=	X %= Y	X = X % Y

Operadores de Incremento/Decremento

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

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

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

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

O valor da variável nB resulta em 21, pois a primeira referência a nA (antes do ++) continha o valor 10 que foi considerado e imediatamente aumentado em 1. Na segunda referência a nA, este já possuía o valor 11. O que foi efetuado foi à soma de 10 mais 11, igual a 21. O resultado final após a execução destas duas linhas é a variável nB contendo 21 e a variável nA contendo 11.

No entanto:

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

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

Variáveis de Memória

Estruturas de Controle

Operadores Especiais

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

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

Os parênteses são utilizados para agrupar elementos em uma expressão mudando a ordem de precedência da avaliação da expressão (segundo as regras matemáticas por exemplo). Também servem para envolver os argumentos de uma função. Os colchetes são utilizados para especificar um elemento específico de uma matriz. Por exemplo, A[3,2], refere-se ao elemento da matriz A na linha 3, coluna 2.

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

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

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

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

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

Ordem de Precedência dos Operadores

Dependendo do tipo de operador, existe uma ordem de precedência para a avaliação dos operando. Em princípio, todas as operações com os operadores, são realizadas da esquerda para a direita se eles tiverem o mesmo nível de prioridade.

A ordem de precedência, ou nível de prioridade de execução, dos operadores em AdvPl é:

Operadores de Incremento/Decremento pré-fixado
Operadores de String
Operadores Matemáticos
Operadores Relacionais
Operadores Lógicos
Operadores de Atribuição
Operadores de Incremento/Decremento pós-fixado

Em expressões complexas com diferentes tipos de operadores, a avaliação seguirá essa seqüência. Caso exista mais de um operador do mesmo tipo (ou seja, de mesmo nível), a avaliação se dá da esquerda para direita. Para os operadores matemáticos entretanto, há uma precedência a seguir:

Exponenciação
Multiplicação e Divisão
Adição e Subtração

Considere o exemplo:

Local nResultado := 2+10/2+5*3+2^3

O resultado desta expressão é 30, pois primeiramente é calculada a exponenciação 2^3(=8), então são calculadas as multiplicações e divisões 10/2(=5) e 5*3(=15), e finalmente as adições resultando em 2+5+15+8(=30).

Alteração da Precedência

A utilização de parênteses dentro de uma expressão altera a ordem de precedência dos operadores. Operando entre parênteses são analisados antes dos que se encontram fora dos parênteses. Se existirem mais de um conjunto de parênteses não-aninhados, o grupo mais a esquerda será avaliado primeiro e assim sucessivamente.

Local nResultado := (2+10)/(2+5)*3+2^3

No exemplo acima primeiro será calculada a exponenciação 2^3(=8). Em seguida 2+10(=12) será calculado, 2+5(=7) calculado, e finalmente a divisão e a multiplicação serão efetuadas, o que resulta em 12/7*3+8(=13.14).

Se existirem vários parênteses aninhados, ou seja, colocados um dentro do outro, a avaliação ocorrerá dos parênteses mais intero em direção ao mais externo.

Expressões

O termo expressão significa um dado ou uma combinação de dois ou mais dados de qualquer tipo. Vamos abordar quatro tipos de dados:

Tipo caractere: Formado por qualquer conjunto de caracteres que constam na tabela Ascii, sempre representado entre aspas.

Tipo numérico: Formado por valores negativos, positivos ou iguais a zero, especificamente usado para operações matemáticas.

Tipo lógico: Existem apenas dois dados possíveis: verdadeiro (.T.) ou falso (.F.), sempre intercalados por ponto.

Tipo data: Aceitam apenas datas coerentes, ou seja, a data é checada quanto a duração do mês (28, 29, 30 ou 31) e a existência do mês (1 a 12).

Tipo memo: Aceita textos de até 64 KB.

Instruções

Instrução é tudo aquilo que é ordenado ao sistema, esperando-se que o mesmo realize imediatamente uma operação.

Diferenças entre Comando e Função

As instruções dividem-se em duas modalidades: comando e função.

Os conceitos abaixo são muito importantes para que se possam compreender as diferenças entre os dois.

Comando: é uma instrução que executa uma tarefa;

Função: é uma instrução que pode receber um ou mais argumentos. Executa um processamento e devolve uma expressão.

Banco de dados

Nós podemos dizer que banco de dados é um conjunto de informações armazenadas de maneira organizada. Por exemplo. Imagine que uma empresa de materiais de construções deseja ter, a carteira de fornecedores e produtos que os mesmos fornecem, de uma maneira organizada e de fácil acesso para: inclusões, consultas, alterações; e exclusões.

Para resolver este "problema", nós poderíamos criar um banco de dados com duas tabelas. Uma para armazenar apenas as informações dos fornecedores, tais como: Nome, endereço, contato, código do fornecedor e etc...

A outra tabela, seria para armazenar as informações dos produtos, tais como:

Data da validade do produto, nome do produto (descrição), unidade, preço unitário e etc.
A seguir, seria criado um relacionamento entre as tabelas e desta maneira, seu eu precisar saber quais produtos um fornecedor me fornece, com o banco de dados seria muito mais simples e rápido.

Observação: Quando o programador, cria um programa, o mesmo precisa "vincular" o mesmo, a um banco de dados, de modo que os dados gerados ou informados pelo programa, fiquem armazenados.

Estruturas de controle

As estruturas de controle são fundamentais em qualquer linguagem de programação. É através das mesmas que se controlam: quantas vezes uma determinada rotina deverá ser executada, que caminho o programa deverá seguir, seguindo condições estabelecidas pelo programa e etc....

As principais são: **If, While, For e etc**...

If (se)
Else (Se não)
Endif (Fim se)

Toda vez que for usada o If, deverá ser usado o EndIf. É possível ter vários If dentro do outro.
Porém, não se esqueça, se você abrir três If, deverá usar o EndIf três vezes.

Observação: Crie ou use uma configuração (ambiente) no Ide e também não se esqueça de abrir ou criar um projeto no mesmo.

Exemplo 1

```
#Include "Rwmake.ch"

User Function ExercIf()

nN1:=3
nN2:=8

If nN1>nN2
```

```
    MsgAlert("O numero " + AllTrim(Str(nN1) + " e maior que " + AllTrim(Str(nN2))))

Else
    MsgAlert("O numero " + AllTrim(Str(nN2) + " e maior que " + AllTrim(Str(nN1))))

EndIf
Return()
```

Exemplo 2

Usando o Elseif

```
User Function ExercIf2()
cEst="MA"
If cEst="RJ"
        Alert("Estado RJ, Icms de 3%")
Elseif (cEst="SP")
        Alert("Estado SP, Icms de 5%")
Elseif(cEst="MG")
        Alert("Estado MG, Icms de 6%")
Elseif(cEst="MA")
        Alert("Estado MA, Icms de 2%")
Else
        Alert("Estado invalido")
Endif
Return()
```

While / EndDo

While(Enquanto)
O código que estiver abaixo do While, será executado enquanto a condição dentro o mesmo for verdadeira.

Exemplo

```
User Function TesteWhile()

Local nCnt:=1
Local aX

While nCnt<=10
```

For
For (Para) Next
O Código que estiver dentro do For será executado por um número determinado de vezes.

Exemplo 1

```
User Function TFor()
nI:=0

For nI:=1 to 5

        MsgAlert(nI)

Next

Return()
```

Observação: Na estrutura For, quando o fluxo é executado pela segunda vez, a variável inicial, que no nosso exemplo é nI, fica valendo dois, na terceira, três e assim por diante, não precisa usar nI++.
Porém, se for necessário que o incremento não seja de um em um pode ser usado o Step. O Step pode ser usado tanto para incrementar o contador como o contrário.

Exemplo 2

```
User Function TFor2()
nI:=0
For nI:=1 to 5 Step 2

        MsgAlert(nI)
Next
Return()
```

Exemplo 3

```
User Function TFor2()
nI:=0

For nI:=1 to 5 Step 2

        MsgAlert(nI)

Next

Return()
```

Exemplo 4

```
User Function TFor3()
nI:=0

For nI:=5 to 1 Step -2

        MsgAlert(nI)

Next
Return()
```

Indentação

Todo programa, deve estar muito bem documentado. Desta forma, facilita o entendimento para quem for analisar o código e até mesmo para fazer ajustes no mesmo.

Também é muito importante que o programa seja indentado. Desta forma, fica muito mais fácil de analisar o mesmo.

Veja este exemplo sem indentação e a seguir, com.

```
If nN1>nN2
MsgAlert("O numero " + AllTrim(Str(nN1) + " e maior que " + AllTrim(Str(nN2))))
Else
MsgAlert("O numero " + AllTrim(Str(nN2) + " e maior que " + AllTrim(Str(nN1))))
EndIf
If nN1>nN2

    MsgAlert("O numero " + AllTrim(Str(nN1) + " e maior que " + AllTrim(Str(nN2))))

Else

    MsgAlert("O numero " + AllTrim(Str(nN2) + " e maior que " + AllTrim(Str(nN1))))

EndIf
```

Funções

Algumas funções

SRT – Converte número em caractere.
SQRT – Calcula a raiz quadrada do número ou variável informada como parâmetro.
ALLTRIM – Tira todos os espaços em branco.

Desenhando telas

Exemplo

Define MSDialog oDlg Title "Consulta de Produtos" From 0,0 To 200,500 Pixel

@ 020,192 SAY TIME()
@ 030,130 SAY "MICROSIGA"
@ 040,100 SAY "CONSULTA DE PRODUTOS"
@ 050,010 SAY "CODIGO DO PRODUTO" //"Etiqueta que aparecerá na tela
@ 050,080 GET cCODIGO PICTURE "@!" VALID ValCodProd()

@ oDlg:nHeight/2-30,oDlg:nClientWidth/2-70 Button oBtnOk Prompt "&Ok" Size 30,15 Pixel Action Confirma() Message "Clique aqui para Confirmar" Of oDlg
@ oDlg:nHeight/2-30,oDlg:nClientWidth/2-35 Button oBtnCancel Prompt "&Cancelar" Size 30,15 Pixel Action oDlg:End() Cancel Message "Clique aqui para Cancelar" Of oDlg

Activate MSDialog oDlg Centered

Digite este código no ide. Compile e execute o mesmo.

Observação: Em primeiro lugar, é necessário definir a tela, com o Define MsDialog.
Quantos todos os elementos da janela estiverem na mesma, é necessário usar o Activete MsDialog, que ativará a tela.

Iniciação de Programas em ADVPL (Algoritmo)

Observação: Um dos grandes erros dos profissionais como um todo, não só os de tecnologia é: achar que conhece muito e acaba menosprezando os procedimentos que a cautela manda seguir.

Por exemplo: é comum o profissional sentar na frente do computador e começar a digitar sem parar. É claro que eu não estou generalizando.

Porém, a maioria se esquece do principal, que é o trabalho de análise.

Nós não podemos nos esquecer que todo programa de computador, independente da: finalidade, preço e função trabalham com o famoso E P S.

Entrada, Processamento e Saída. Se os dados de entrada não forem preenchidos corretamente o processamento será deficiente e os relatórios apenas nos mostrarão isso.

Por isso, o trabalho de análise é fundamental.

Por estes motivos, eu recomendo que antes de nós começar-mos a digitar, vamos trabalhar com o algoritmo.

Use o que nós chamamos de portugol para resolver estes exercícios.

Só depois o Advpl.

Exercícios

Exercício 01 – Crie uma função para solucionar o seguinte problema.

Desenvolva uma função que calcule a média de três notas.

Exercício 02 – Desenvolva uma função que calcule a raiz quadrada do número 49.

Exercício 03 – Agora, use a função anterior como base, e calcule a raiz quadrada do número digitado pelo usuário.

Exercício 04 – Desenvolva uma função que calcule a tabuada do número 2.

Exercício 05 – Desenvolva uma função que calcule a tabuada do número desejado (informado).

Exercício 06 - Agora, desenvolva quatro programas: Inclusão, Consulta, Alteração e Exclusão.
O objetivo principal destes exercícios é: colocar em prática o que foi apresentado na teoria, e "alimentar" o SZ6. Não se esqueça de usar algoritmos.

Anotações

Número de registro: XBAP10270707