UNISINOS - UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6) – Curso: Informática

PROGRAMAÇÃO I – AULA 09

Disciplina: Linguagem de Programação PASCAL
Professor responsável: ***Fernando Santos Osório***
Semestre: 2001/2
Horário: 63

E-mail: ***osorio@exatas.unisinos.br***
Web:
http://inf.unisinos.br/~osorio/prog1.html
Xerox : *Pasta 54* (Xerox do C6/6)

1. Sub-Rotinas : Funções e Procedimentos – *FUNCTION / PROCEDURE*

Programação Modular em Pascal = Uso de Sub-Rotinas: Functions e Procedures.

1.1. Variáveis e Parâmetros nas Sub-Rotinas

As *Procedures* e *Function* podem receber e retornar dados através dos parâmetros – variáveis que vem descritas dentro da lista de parâmetros entre parênteses, ou também no caso específico das *Functions*, podemos retornar valores atribuindo o resultado final (um único resultado) ao nome da função.

```
Function Nome_Funcao (Parametro1 : integer ; Param2 : real ; Param3, Param4 : char) : Real;
Var
   Variavel_Local:Integer;
   Resultado: Real;
 Begin
   ...  {Comandos da Function }
   Nome_Funcao := Resultado;
End;
```

Podemos dizer que os parâmetros são como variáveis que vem de fora da sub-rotina. Na verdade os parâmetros se comportam exatamente como as variáveis, ou seja, podem ser considerados como variáveis que "vieram" de uma outra parte do programa, podendo ser exibidos, usados em expressões e até mesmo ter seu conteúdo alterado. Por enquanto nós só estudamos os chamados parâmetros *passados por valor*, o que faz com que qualquer modificação no seu conteúdo não afete o valor da variável original que foi usada na chamada da sub-rotina e passada como parâmetro.

Observações Importantes sobre Parâmetros e Variáveis Locais:

- *Nomes e Acesso*: o nome de um parâmetro ou de uma variável local a uma sub-rotina ***só é conhecido dentro da sub-rotina***, ou seja, este nome de variável não será acessível fora da rotina onde ele foi declarado ou recebido como parâmetro. Por este motivo que chamamos as variáveis das *Procedures* e *Functions* de ***variáveis locais,*** pois estas variáveis são de uso apenas local.

- *Nomes duplicados*: é possível termos nomes de variáveis locais ou de parâmetros idênticos aos nomes de outras variáveis locais ou até mesmo globais. É importante salientar que no caso de variáveis locais terem o mesmo nome, em duas sub-rotinas distintas, estas variáveis serão diferentes (armazenadas em locais diferentes da memória) e uma não vai afetar a outra (podem ser mesmo declaradas de tipos diferentes). Caso tenhamos uma variável global declarada com o mesmo nome de uma variável local, vale a regra básica de que a variável que será acessada é sempre a mais "interna", ou seja, a ***local sempre sobrepõe a global*** (ou a mais externa).

```
Program Exemplo;
Var
   Dado : Integer;

Procedure Zera_Tudo (Parametro : Integer);
Var
   Dado : Integer;
Begin
    Parametro := 0;
    Dado:= 0;
End;

Begin
   Dado:=100;
   Zera_Tudo (Dado);
   WriteLn (Dado);                { Qual é o valor escrito na tela ? }
   ReadLn;
End.
```

- *Tempo de Vida*: as variáveis locais e parâmetros possuem uma ***vida curta***, ou seja, são criadas (alocadas na memória => ganham um 'espaço' para elas) quando a *procedure* ou *function* começar a ser executada e são destruídas ao terminar a execução da *procedure* ou *function*. Por isso costumamos chamar as variáveis globais de estáticas (criadas no início da execução do programa e permanecem assim até o fim), e as variáveis locais / parâmetros são chamadas de variáveis dinâmicas (pois vão sendo criadas e destruídas durante a execução do programa). Logo a vida de uma variável local dura entre o início da execução da sub-rotina (momento em que é chamada) e o término da execução desta sub-rotina (quando chegamos ao "end" da sub-rotina estas variáveis são destruídas). Isto explica em parte porque uma variável local não pode ser acessada fora da sub-rotina...

```
Function Media (Nota1, Nota2 : Real) : Real;
{   Nota1 e Nota2 são criadas. O valor do primeiro parâmetro contido na
    chamada desta function é copiado dentro de Nota1 e o valor do segundo
    parâmetro é copiado em Nota2.                                        }
Var
    Total : Real;
{ A variável local Total é criada – Atualmente contêm apenas "lixo" dentro dela. }
Begin
    Total := Nota1 + Nota 2;
    Media := Total / 2;
End;
{   Os parâmetros Nota1 e Nota2 são destruídos (desalocados da memória) assim
    como a variável Total. O único valor que é retornado é a Media       }
```

- *Inicialização*: variáveis globais são inicializadas com zero e variáveis locais não são inicializadas, logo contêm "lixo" (valores indeterminados). Apesar de sabermos como são inicializadas as variáveis, um programa bem estruturado ***sempre*** inicializa as variáveis com valores iniciais, antes de usar estas variáveis.

- *Tipos de parâmetros*: existem duas formas de passar parâmetros para as funções e procedimentos – ***passagem por valor*** (só envia valores) e ***passagem por referência*** (permite enviar e também receber valores). No item seguinte vamos discutir em mais detalhes este assunto.

1.2. Passagem de Parâmetros

As procedures e functions podem passar parâmetros de duas maneiras: a primeira, que já vínhamos usando, é através de uma cópia do valor do parâmetro passado na chamada da sub-rotina dentro da variável declarada como parâmetro da sub-rotina – ***passagem de parâmetros por valor***; e a segunda maneira é através da passagem do endereço onde se encontra a variável usada como parâmetro na chamada da sub-rotina – ***passagem de parâmetros por referência***.

1.2.1. Passagem de Parâmetros por Valor – BY VALUE

Na passagem de ***parâmetros por valor é passada realmente uma cópia do valor*** indicado na chamada da *Procedure* ou *Function*. As conseqüências imediatas deste tipo de procedimento são:

- Se alterarmos o conteúdo da variável recebida como parâmetro estaremos alterando a cópia do valor original e portanto o original não é modificado.

- Não podemos retornar nenhum valor de volta ao procedimento que chamou uma sub-rotina pois todo valor que for alterado em um parâmetro afeta somente a cópia que é destruída ao final da execução da sub-rotina (parâmetros e variáveis locais são destruídos).

Exemplo de passagem de parâmetros por valor (forma padrão que já vínhamos usando):

```
Procedure Exibe_Valor ( Valor : Real );
Begin
    WriteLn ('Valor atual: ',Valor);
    Valor := 0.0;
End;
...
Begin
   ...
   Exibe_Valor (Salario_do_Chefe);          { Ainda bem que é By Value !!! }
   Gera_Folha_de_Pagamento;
   ...
End.
```

Podemos dizer que a passagem de parâmetros BY VALUE é feita entregando o 'xerox' das informações, e jamais o original. Desta forma a sub-rotina pode 'riscar e rabiscar' os dados que foram passados, sem no entanto danificar o nosso precioso 'documento original'.

1.2.2. Passagem de Parâmetros por Referência – BY REFERENCE

Na passagem de ***parâmetros por referência é passado o endereço da variável*** indicada na chamada da *Procedure* ou *Function* - Usamos a palavra **VAR** na lista de parâmetros. As conseqüências imediatas deste tipo de procedimento são:

- Se alterarmos o conteúdo da variável recebida como parâmetro estaremos alterando o valor original, pois temos a nossa disposição o endereço exato onde esta se localiza na memória, e portanto *podemos alterar diretamente o valor original*.

- Quando usarmos a passagem de parâmetros por referência **NÃO** podemos passar valores numéricos ou expressões como parâmetro na chamada da função, tem que ser uma variável única onde possamos definir exatamente o seu endereço. Exemplo:

```
Procedure  Parametro_Por_Referencia (Var Valor : Real );
Begin
   ...
End;
...
Begin
   Parametro_Por_Referencia ( Total  );              { Funciona... OK }
   Parametro_Por_Referencia ( Total + 5  );          { NÃO Funciona... }
   Parametro_Por_Referencia ( Total  * Fator  );     { NÃO Funciona... }
   Parametro_Por_Referencia ( 320.50 );              { NÃO Funciona... }
End.
```

{ *As chamadas acima só funcionariam se fosse feita uma passagem de parâmetros por valor* }

- ***Podemos retornar um ou mais valores de volta*** ao procedimento que chamou uma sub-rotina ***através do uso de parâmetros por referência***, pois uma vez que a sub-rotina utilize este tipo de parâmetros é dado a ela o "direito" de alterar a variável original (através da referência ao seu endereço) afetando diretamente o valor original.

- Conforme havíamos indicado, todo o parâmetro (assim como as variáveis locais) são destruídos ao final da execução da sub-rotina. Isso também é válido para a passagem de parâmetros por referência, mas na realidade o que é destruído é a referência à variável original usada na chamada (cópia do endereço) e não a própria variável.

Exemplo de passagem de parâmetros por referência:

```
Procedure Inicializa_Valor ( VAR Valor : Real );
Begin
    WriteLn ('Valor antigo: ',Valor);
    Valor := 120.0;
    WriteLn ('Valor após inicialização: ',Valor);
End;
...
Begin  ...
   Salario := 4000.00;
   Inicializa_Valor (Salario);        { Após a chamada da rotina, Salario será alterado }
... End.
```

EXERCÍCIOS – AULA 09 – Passagem de parâmetros por referência

1. Faça um programa que leia 2 valores e chame uma sub-rotina (procedure) que receba estas 2 variáveis e troque o seu conteúdo, ou seja, esta rotina é chamada passando duas variáveis A e B por exemplo, e após a execução da rotina A conterá o valor de B e B terá o valor de A.

2. Altere o programa anterior de forma a ler 4 valores em 4 variáveis: A, B, C, D. Use a rotina "troca_valores" implementada no programa anterior para trocar os valores de A com B, depois de C com D, mostre como ficaram os valores das variáveis, e para concluir destroque seus conteúdos mostrando novamente na tela como ficaram estas variáveis.

Valor1: ***10***
Valor2: ***20***
Valor3: ***30***
Valor4: ***40***
Trocando Valor1 com Valor2 e Valor3 com Valor4. Resultado:
Valor1: **20** - Valor2: **10** - Valor3: **40** - Valor4: **30**
Trocando Valor1 com Valor2 e Valor3 com Valor4. Resultado:
Valor1: **10** - Valor2: **20** - Valor3: **30** - Valor4: **40**

3. Faça um programa que leia dois valores e chame uma sub-rotina que receba estes 2 valores de entrada e retorne o maior valor na primeira variável e o segundo maior valor na segunda variável. Escreva o conteúdo das 2 variáveis na tela.

4. Faça um programa que leia três valores e chame uma sub-rotina que receba estes 3 valores de entrada e retorne o maior valor entre estes três valores na primeira variável, o segundo maior valor na segunda variável e o terceiro maior valor na terceira variável. Exibir os valores ordenados na tela (sugestão: use a sub-rotina troca_valores).

5. Faça um programa que leia um número e gere todos os números primos entre 1 e este número fornecido, exibindo-os na tela. O programa deve ter uma sub-rotina que determine se um determinado número é ou não primo.

6. Usando a sub-rotina que exibe uma moldura na tela em uma janela definida pelo usuário, faça uma outra sub-rotina que escreva uma mensagem na tela pedindo para ler 2 números (um de cada vez), leia estes números e retorne para o programa principal. A mensagem e a leitura dos números deve ser feita dentro de uma janela com moldura. Em seguida exiba a média aritmética simples destes dois números também usando uma janela com moldura.

Entre com o 1o. número: ***5.2***

Entre com o 2o. número: ***7.4***

Média: **6.3**

7. Faça um programa que leia 5 números. Este programa deve ter um menu que permita ao usuário escolher qual opção de cálculo ele deseja realizar: média aritmética simples, média ponderada (ler os pesos associados a cada nota que serão informados pelo usuário), desvio padrão, maior valor e menor valor. A leitura dos 5 valores também deve ser uma das opções do menu.

Exemplo:

>> Estatística <<

1 – Entrar com os dados (5 valores)
2 – Calcular a média aritmética simples dos dados
3 – Calcular a média ponderada dos dados (fornecer 5 pesos)
4 – Calcular o desvio padrão dos dados
5 – Achar o maior valor
6 – Achar o menor valor
7 – Sair do programa (Fim)

Entre com a sua opção: ***1***

Valor1: 5.3 Valor2: 8.2 Valor3: 7.3 Valor4: 3.7 Valor5: 7.1

Entre com a sua opção: 2

Média Aritmética Simples: **6.32**

Entre com a sua opção: ***7***

FIM!

Obs: Desvio Padrão = Raiz quadrada da divisão do somatório do quadrado das diferenças entre valor e a média de todos os valores, pelo número total de dados.

8. Faça um jogo no qual o computador desafia dois usuários para ver quem tem mais memória. O jogo deve gerar um número de 0 à 9 e apresentar ao jogador 1 durante 5 segundos. Depois deve repetir este procedimento para o jogador 2. Em seguida, ele deve pedir ao jogador 1 para informar o número escolhido, e após pedir ao jogador 2 para fazer a mesma coisa. Na segunda rodada, o computador deve gerar mais um número entre 0 e 9, que será "concatenado" à direita do primeiro número. Então repetiremos o procedimento de exibição e teste de memorização para os dois usuário. O programa deve continuar adicionando números (casas adicionadas ao final do número) até que um dos dois jogadores não consiga mais memorizar perfeitamente o número. Para concluir, o programa deve informar qual dos dois jogadores ganhou, quantas casas foram memorizadas ao total, e em que casa foi que o jogador perdedor errou (primeira casa errada começando da esquerda para a direita). Dicas: procure desenvolver o programa em módulos; armazene a seqüência de números dentro de uma string.

   => Seu programa ficou bem modular? Seria fácil de adaptá-lo para 3 ou mais jogadores?