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

PROGRAMAÇÃO I – AULA 08

Disciplina: Linguagem de Programação PASCAL
Professor responsável: ***Fernando Santos Osório***
Semestre: 99/1
Horário: 61
E-mail: ***osorio@exatas.unisinos.tche.br***
Web:
http://www.inf.unisinos.tche.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. Sub-Rotinas

As *sub-rotinas* são um conjunto de comandos que são agrupados e recebem assim um nome específico que irá identificá-las. Desta forma, podemos usar as sub-rotinas para realizar tarefas que se repetem várias vezes na execução de um mesmo programa. O uso de sub-rotinas permite que possamos criar PROGRAMAS MODULARES, deixando estes mais bem estruturados. O uso de sub-rotinas nos leva ao desenvolvimento de programas com uma abordagem "*Top-Down*" ou "*Botton-Up*", onde vamos **dividir o problema para conquistá-lo** mais facilmente.

Em programação PASCAL nós muitas vezes usamos sub-rotinas sem sequer perceber que estamos fazendo uso deste tipo de recurso da linguagem. Exemplos de funções e procedimentos pré-definidos da linguagem Pascal (Turbo Pascal):

Function: Sqr, Sqrt, Length, UpCase, ReadKey, Trunc, Round, Int, Frac, ...

```
Quadrado := SQR (numero);
Raiz_Quadrada := SQRT (numero);
Numero_de_Letras := LENGTH (Texto);
Maiuscula := UPCASE (Minuscula);
Parte_Inteira := TRUNC (Numero_Real);
```

Procedure: ClrScr, GotoXY, WriteLn, ReadLn, Delay, Inc, Dec

```
CLRSCR;
GOTOXY (10,10);
WRITELN ('Hello World');
DELAY (1000);
INC (Contador);
```

Como pode ser visto acima, o uso de ***funções e procedimentos pré-definidos*** da linguagem Pascal é bastante comum... também notamos que existe uma diferença bem clara entre uma ***Function*** e uma ***Procedure***. Vamos agora estudar como criar as nossas próprias Functions e Procedures.

1.2. Estrutura Geral de um Programa em Pascal

Um programa em Pascal começa pela declaração de estruturas de dados e elementos globais do programa (*Uses*, *Const*, *Type* e *Var* globais), seguido da declaração/definição das funções e procedimentos, e terminando pela declaração das variáveis do programa (*Var* do programa principal) que termina pelo programa principal – *Begin/End* que contêm os comandos por onde se inicia a execução do programa.

Um programa bem estruturado **NÃO deve possuir variáveis globais declaradas** (aquelas que aparecem logo depois do "*Program*" ou "*Const*" ou "*Type*" iniciais), ou se isto não for possível de evitar, devemos usar o menos possível este tipo de variáveis. As **variáveis** que serão utilizadas pelo programa principal **devem ser declaradas** imediatamente antes do bloco do programa principal (**logo antes do último bloco *Begin/End***).

```
PROGRAM <nome_do_programa> ;
USES
   <nome_das_units> ;
CONST
   <nome_da_constante> = <valor> ;
TYPE
    <nome_do_novo_tipo_de_dados> = <definição_do_tipo> ;
VAR
   <nome_da_variável> :  <tipo_da_variável> ;
```

```
FUNCTION <nome_da_função> [ <lista_de_parâmetros> ] : <tipo_retornado> ;
VAR
   <declaração_das_variáveis_da_função> ;
BEGIN
  <comandos_da_função> ;
  <nome_da_função> := <valor_retornado> ;
END ;

PROCEDURE <nome_da_procedure> [ <lista_de_parâmetros> ] ;
VAR
   <declaração_das_variáveis_da_procedure> ;
BEGIN
  <comandos_da_procedure> ;
END;
```

```
{ Programa Principal }
VAR
    <nome_da_variável> : <tipo_da_variável> ;
BEGIN
    <comandos> ;
END .
```

1.3. Funções – FUNCTION

Uma função é um tipo especial de sub-rotina que retorna um resultado de volta ao "ponto" onde foi chamada. As *functions* tem a seguinte sintaxe:

```
FUNCTION <nome> ( <var_parâmetro>, <var_parâmetro>, ... ) : <tipo_retornado> ;
VAR
   <variável> : <tipo_da_variável> ;
   ...
BEGIN
   <comando> ;  ...
   <nome> := <valor_ou_expressão_retornada> ;
END ;
```

Exemplos:

```
Program Calcula_Media;

Function Media (Nota1, Nota2 : Real) : Real ;   { Definição da Função }
Begin
    Media := (Nota1 + Nota2) / 2;
End;

Var
    N1, N2 : Real;
    Media_Final : Real;
Begin                                              { Programa Principal }
    Writeln ('Entre com 2 notas: ');
    Readln(N1, N2);
    Media_Final := Media (N1, N2);                 { Chamada da Função }
    Writeln ('A média final é ', Media_Final:2:2);
    Readln;
End.
```

```
Function Porcentagem (Valor : Real ; Indice : Real ) : Real ;
Var
   Resultado : Real;
Begin
   Resultado := Valor * ( Indice / 100 );
   Porcentagem := Resultado;
End.
...
Begin
  ...
   Salario_Bruto := 1500.00;
   Desconto _IRF :=  15.0;      { Desconto imposto retido na fonte = 15% do salário bruto }
   Salario_Liquido := Salario_Bruto – Porcentagem (Salario_Bruto, Desconto_IRF);
   ...
   Adicional_CPMF :=  Porcentagem (Valor_do_Cheque, 0.25 );  { CPMF = 0.25% do valor }
  ...
End.
```

Observações:

- Sobre os parâmetros de uma função:

  - Note que os nomes das variáveis na chamada da função não precisam ter o mesmo nome dos parâmetros declarados no cabeçalho da *function*. Os parâmetros são "casados" um-à-um de acordo com a ordem em que estes aparecem! (Ver no 1º exemplo que N1 vai "casar" com Nota1).
  - O número de parâmetros declarados no cabeçalho da função tem que ser o mesmo número de parâmetros passados na chamada da função!
  - Podemos ter diferentes tipos de parâmetros declarados no cabeçalho de uma função, basta que cada parâmetro declarado seja seguido de um ';' da mesma forma como fazemos na declaração de variáveis em Pascal.
    Exemplo: funcao_xyz (a,b,c : integer ; x,y : real ; txt : string) : boolean;

- Logo após a declaração do cabeçalho da função (nome e parâmetros), vamos começar a definir os dados e elementos locais a função, como se estivéssemos escrevendo um "mini-programa". Podemos ter constantes (*Const*), novos tipos de dados (*Type*) e variáveis (*Var*) que serão conhecidas e usadas APENAS localmente à função.

- Ao terminar a execução de uma função temos que retornar o resultado de alguma maneira para quem chamou esta função. O modo usado para retornar valores em uma função é através da atribuição do valor retornado ao nome da função (usado como se fosse o nome de uma variável). Desta forma, na parte do programa que chama a função vamos usar o nome desta função para obter de volta o resultado final da execução dela. **Atenção**: uma função só pode retornar UM ÚNICO valor! Mais tarde veremos um outro mecanismo para podermos retornar mais de um valor, mas para isso usaremos um método diferente do retorno das funções...

1.4. Procedimentos – PROCEDURE

Um procedimento é uma sub-rotina (módulo = mini-programa) que é usado a fim de executar uma certa tarefa. A *procedure* recebe um certo número de parâmetros de entrada, do mesmo modo que as funções, mas no entanto não retorna nenhum valor como saída. As *procedures* tem a seguinte sintaxe:

```
PROCEDURE <nome> ( <parâmetro>, ... ) ;          { Cabeçalho: Nome + Parâmetros }

CONST                                            { Constantes Locais da Procedure }
     <constante> = <valor> ;
VAR
      <variável> : <tipo_da_variável> ;          {Variáveis Locais da Procedure }
      ...

BEGIN
  <comando> ;
END;
```

Exemplos:

```
Program  Inutil;
Uses
   Crt;

Procedure Mensagem_de_Espera ( Msg: String ; Tempo_em_Segundos : Word) ;
Begin
    Clrscr;
    GotoXY (30,12);
    Write (Msg);
    Delay (Tempo_em_Segundos * 1000);
End;

Begin
    Mensagem_de_Espera ('Este programa...', 10);
    Mensagem_de_Espera ('Serve somente...', 10);
    Mensagem_de_Espera ('Para...', 10);
    Mensagem_de_Espera ('Você perder seu tempo !', 20);
    readln;
End.
```

```
Procedure Aluno;
Var
   Nota1, Nota2, Media : Real;
Begin
    Writeln ('Entre com 2 notas: ');
    Readln(Nota1, Nota2);
    Media := (Nota1 + Nota2) / 2;
    Writeln ('A média final é ', Media:2:2);
    Write ('Pressione uma tecla para continuar... ');
    ReadKey;
End;
...
Begin
    ...
    For Contador := 1 to 20
    Do  Aluno;
    ...
End.
```

Observações:

- Sobre os parâmetros de um procedimento: valem as mesmas observações feitas para as funções, ou seja, a ordem dos parâmetros na declaração e na chamada é muito importante ("casamento um-à-um"), o número e tipo dos parâmetros declarados e usados na chamada devem ser os mesmos, e por fim a sua declaração também segue o mesmo modelo da declaração de uma variável qualquer em Pascal. Os parâmetros de uma *procedure* definem uma interface com o "mundo" externo (no que se refere aos exemplos vistos, definem a entrada de dados).

- Logo após a declaração do cabeçalho da *procedure* (nome e parâmetros), vamos começar a definir os dados e elementos locais ao procedimento, como se estivéssemos escrevendo um "mini-programa". Podemos ter constantes (*Const*), novos tipos de dados (*Type*) e variáveis (*Var*) que serão conhecidas e usadas APENAS localmente à *procedure*.

- Para executar um procedimento basta colocar seu nome (seguido dos respectivos parâmetros), e ao terminar a execução deste procedimento vamos continuar a execução a partir da linha seguinte a linha de comando de chamada da *procedure*. **Atenção**: uma *procedure* usualmente não retorna nenhum valor! Mais tarde veremos um mecanismo especial que permite o retorno de um ou mais valores, mas para isso usaremos um método diferente de passagem de parâmetros para as procedures...

1.5. Variáveis Globais *versus* Variáveis Locais

Programas que usam procedures ou functions terão dois tipos de variáveis: as *variáveis globais* à todo o programa e as *variáveis locais* as sub-rotinas e ao programa principal.

As **variáveis globais** possuem um problema no que se refere a boa estruturação de programas modulares, elas são visíveis para todos os módulos do programa, ou seja, qualquer um pode ler ou alterar o seu conteúdo. Imaginemos a seguinte situação: com a modularidade dos programas podemos trabalhar em equipe, onde cada pessoa ou sub-equipe é responsável pela codificação de um módulo do programa. Suponhamos que o programador Errolino resolva alterar o valor da variável 'Salario' que foi definida como uma global (acessível à todos) na procedure que ele implementou, mas ao mesmo tempo o programador Errovaldo havia colocado um valor muito importante nesta variável, logo antes de chamar a procedure feita pelo seu colega... pobre Errovaldo, nem ficará sabendo que foi demitido por culpa do Errolino que "sabotou" o módulo dele!

As **variáveis locais** surgem para evitar este problema. O ideal seria que pudéssemos isolar cada procedure/function e trocar apenas dados de entrada e saída. Cada procedure/function teria as suas próprias variáveis (variáveis locais) e assim não precisariam usar outras variáveis externas ao módulo. Mas nem sempre isso é possível, pois temos também que trocar informações e dados entre os diferentes módulos de um sistema... e é por isso que podemos passar parâmetros entre uma proceudure/function e outra. Portanto, podemos escrever programas mais bem estruturados utilizando **APENAS *variáveis locais*** e ***passagem de parâmetros*** entre as procedures/functions. Note que com o uso de variáveis locais a uma procedure/function as demais procedures/functions não tem direito e não conseguem ler ou alterar o valor de uma variável que não pertença a elas.

Para terminar, as variáveis globais são declaradas logo no início do programa, antes de começar a declarar os procedimentos e funções, e as variáveis locais são declaradas dentro dos procedimentos e funções. Veja o exemplo abaixo:

```
Program Exemplo;
Var
    Global: Integer;

Procedure  Sub_Rotina_1 (Parametro: Integer) ;
Var
    Local: Integer;
```

```
Begin          { Comandos da Procedure }
...
End;

Var
   Local_Prog_Principal: Integer;

Begin          { Comandos do Programa Principal }
  ...
End.
```

1.6. Exemplo de programa com menu e procedures

```
program CALCULADORA;
uses
  Crt;
{*** Sub-rotinas de calculos ***}

  procedure ROT_ADICAO;
  var
    X, A, B : real;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Adicao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A + B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    readkey;
  end;

  procedure ROT_SUBTRACAO;
  var
    X, A, B : real;
  begin
    clrscr;
    gotoxy(30, 1); write('Rotina de Subtracao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A - B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    readkey;
  end;

  procedure ROT_MULTIPLICACAO;
  var
    X, A, B : real;
  begin
    clrscr;
    gotoxy(28, 1); write('Rotina de Multiplicacao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A * B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    readkey;
  end;
```

```
  procedure ROT_DIVISAO;
  var
    X, A, B : real;
  begin
    clrscr;
    gotoxy(32, 1); write('Rotina de Divisao');
    gotoxy( 5, 6); write('Entre um valor para A: '); readln(A);
    gotoxy( 5, 7); write('Entre um valor para B: '); readln(B);
    X := A / B;
    gotoxy( 5,10); write('O resultado equivale a: ', X:4:2);
    gotoxy(25,24); writeln('Tecle algo para voltar ao menu');
    readkey;
  end;
{*** Programa Principal ***}

var
  OPCAO : char;
begin
  OPCAO := '0';
  while (OPCAO <> '5') do
    begin
      clrscr;
      gotoxy(33, 1); write('Menu Principal');
      gotoxy(28, 6); write('1 ................. Soma');
      gotoxy(28, 8); write('2 ............ Subtracao');
      gotoxy(28,10); write('3 ........ Multiplicacao');
      gotoxy(28,12); write('4 .............. Divisao');
      gotoxy(28,14); write('5 ...... Fim de Programa');
      gotoxy(28,18); write('Escolha uma opcao ....: ');
      readln(OPCAO);
      if (OPCAO <> '5') then
        case OPCAO of
           '1' : Rot_Adicao;
           '2' : Rot_Subtracao;
           '3' : Rot_Multiplicacao;
           '4' : Rot_Divisao;
        else
           gotoxy(27,24); writeln('Opcao invalida - Tecle algo');
           OPCAO := readkey;
        end;
    end;
end.
```

- *Observação sobre o programa acima*: note como se repete o trecho de código referente a entrada de dados (leitura de A e B). Este procedimento poderia se transformar numa sub-rotina... mas como fazer para ler e *retornar* dois valores ao mesmo tempo?

EXERCÍCIOS – AULA 08

1. Faça um programa com uma sub-rotina que receba 3 valores de entrada e retorne o maior valor entre estes três valores.

2. Faça um programa que peça para ler 2 notas e depois mostre:
   - A média aritmética simples;
   - A média ponderada entre os dois valores (nota 1 com peso 1 e nota 2 com peso 2);
   - O valor necessário para recuperar a pior nota e passar com média igual ou superior a 6.0.
   Faça um programa modular, usando uma sub-rotina para o cálculo de cada tarefa descrita acima.

3. Faça uma sub-rotina para validar uma data fornecida pelo usuário (baseado no exercício da lista da aula 03).

4. Faça uma sub-rotina que calcule X elevado à Y. Leia 2 valores de X e Y e exiba o resultado na tela. Exemplo: 2 elevado à 3 é igual à 2*2*2 = 8.

5. Faça uma sub-rotina "Br_UpCase" que dado um caracter qualquer retorne o mesmo caracter sempre em maiúsculo, aceitando inclusive os caracteres acentuados da língua portuguesa.

6. Criar uma *procedure* que desenhe uma moldura na tela do micro. Faça no programa principal uma chamada a esta *procedure*, desenhando a moldura e após escrevendo "Hello World" no meio da tela (X=35, Y=12). Para desenhar a moldura use os caracteres especiais da tabela ASCII estendida do Turbo Pascal. Exemplo: pressione a tecla ALT e ao mesmo tempo um dos seguintes números no teclado numérico - ALT + 200, ALT + 201, ALT + 205, ALT + 186, ALT + 187, ALT + 188.

7. Altere o programa anterior de maneira que quando o usuário apertar uma tecla (*readkey*), a tela seja limpada, a moldura desenhada novamente e seja escrito "Bye-Bye World" no meio da tela. O programa deve terminar automaticamente após um *delay* de 5 segundos.

8. Faça um programa para o cálculo do fatorial, mas desta vez crie uma rotina separada que realize o cálculo do fatorial, e também utilize uma rotina que desenha uma moldura na tela (reaproveitamento de código).

9. Faça uma rotina genérica para criar molduras na tela com o tamanho especificado pelo programa através dos parâmetros que são passados para esta rotina. Os parâmetros vão indicar a linha inicial e final da moldura na tela, assim como a coluna inicial e final da moldura. No programa principal use esta rotina para emoldurar o seu nome na tela.

10. Faça um programa que calcule a raiz quadrada de um número sem usar a função Sqrt do Pascal. Você deve implementar a sua própria função para o cálculo da raiz quadrada. Obtenha uma raiz com uma precisão de no mínimo 2 casas corretas após a vírgula.

11. Faça um teste de mesa para o programa dado abaixo e indique qual é o valor das variáveis A e B que é escrito na tela – dentro da procedure e no programa principal.

```
Program Teste_de_Mesa;

Procedure  Altera (A, B : Integer );
Begin
    Writeln (A,' e ',B);
    A := 10;
    B :=-10;
    WriteLn (A,' e ',B);
End;

Var
    A, B: Integer;
Begin
    A := 1;
    B := 2;
    WriteLn (A,' e ',B);
    Altera (B, A);
    WriteLn (A,' e ',B);
End.
```