FACE - FUMEC - TECNOLOGIA SUPERIOR EM PROCESAMENTO DE DADOS
Disciplina : LTP1 - Linguagem Técnica de Programação 1 - 1996
Professor : Ricardo Tolentino.

TURBO PASCAL

PROJETISTA DA LINGUAGEM PASCAL:

Niklaus Wirth
Professor da Escola Politécnica de Zurique.

ANO DE APRESENTAÇÃO DA LINGUAGEM :

1971

ÁREA DE APLICAÇÃO :

Linguagem de programação estrutura
da para o desenvolvimento de aplicações de uso geral

OBJETO DESTE ESTUDO :

Linguagem Turbo Pascal 6.0, produzida pela Borland Intemational.

CONCEITOS BÁSICOS:

VARIÁVEIS NUMÉRICAS :

integer - Números inteiros entre -32768 e 32767.

real - Números reais de 11 algarismos significativos.

byte - Números inteiros entre 0 e 255.

VARIÁVEIS ALFANUMÉRICAS:

string - Caracteres alfanuméricos entre apóstrofos.
char - String de apenas um caractere entre apóstrofos.

VARIÁVEIS LÓGICAS:

boolean - Assume apenas valores booleanos, que são:
true (verdadeiro) ou false (falso).

OPERADORES ARITMÉTICOS :

* Multiplicação

/ Divisão

+ Soma

- Subtração

OPERADORES RELACIONAIS:

< Menor que

> Maior que

= Igual a

<> Diferente de

<= Menor ou igual a

>= Maior ou igual a

OPERADORES LÓGICOS:

not - negação

and - e

or - ou

xor - ou exclusivo

COMANDOS DE DECLARAÇÃO DE VARIÁVEIS :

No Turbo Pascal a declaração de variáveis é obrigatória no início do programa pois, caso isso não acorra, o compilador acusará erro de compilação.

Exemplos:

Variável inteira A, real B e booleana C.

```
Var

    A : integer;
    B : real;
    C : boolean;
```

Na declaração de uma variável string, necessita-se informar a sua máxima extensão. Esta declaração da extensão reserva uma área fixa na memória para o processamento.

Exemplo:

A variável string R com 13 caracteres, seria assim declarada:

```
Var

    R : string[13];
```

Para declarar mais de uma variável do mesmo tipo, pode-se proceder das duas seguintes formas:

```
Var

    x, y, z : integer;
```

ou

```
x : integer;
y : integer;
z : integer;
```

EXPRESSÃO DE ATRIBUIÇÃO DE VALORES :

A:=B+5;

O símbolo := equivale á uma flecha (<-), indicando que é uma atribuição.

PONTO E VÍRGULA:

O ponto e vírgula no Turbo Pascal indica o fim do comando e sua presença é obrigatória

INÍCIO E FIM DE PROGRAMA:

Todos os programas em Turbo Pascal devem começar e terminar desse modo:

```
program Exemplo;
.
.
begin
.
.
.
.
.
end.
```

O "program" indica o início do programa;

"Exemplo" é um nome qualquer dado ao programa;

"begin" indica o início e 'end" o fim do processamento.

COMENTÁRIOS:

Os comentários devem ser escritos entre chaves , {....} ou parênteses com asterisco - (*....*). Não há necessidade de terminar a linha do comentário com ponto e vírgula.

Exemplo:

{Exemplo de como escrever um comentário}

(* Pode-se escrever também desta foram *)

COMANDOS DE ENTRADA E SAÍDA:

SAÍDAS DE INFORMAÇÕES NO VÍDEO:

write - Apenas imprime

writeln - Imprime e envia o sinal de mudança de l1nlla (Line Feed)

Tudo o que se deseja imprimir deve vir entre parênteses.

Exemplo 1:
Imprimindo a variável A.

```
writeln(A);
```

Exemplo 2:
Imprimindo o texto sempre entre apóstrofos 1').

```
writelnl'confirma a Resposta (S/N) ? ');
```

Exemplo 3:

```
Program Prog01 ;

var

        LARGURA, COMPRIMENTO, ALTURA: integer;

begin

        LARGURA := 10;
        COMPRIMENTO := 3;
        ALTURA := 2;
        writeln('VOLUME = ', LARGURA*COMPRIMENTO*ALTURA,' Cm3'):

end.
```

O resultado do programa acima ficaria assim no vídeo:
VOLUME = 60 cm3

SAÍDA DE INFORMAÇÕES NA IMPRESSORA:

Os comandos que acionam a impressora são os mesmos do vídeo, ou seja, write e writeln, acrescidos do parâmetro (lst).

Exemplo:

```
writeln(lst,'VOLUME = ', LARGURA*COMPRIMENTO*ALTURA,' cm3');
```

OBS.: O comando de mudança de linha (line feed) é dado após a impressão no exemplo anterior. A saida do comando de impressão (VOLUME = 60 cm3) seria a mesma com os comandos abaixo:

```
write(lst,'VOLUME = ');
write(lst,LARGURA*COMPRIMENTO*ALTURA);
write(lst,'cm3');
```

ENTRADA DE DADOS:

read - Não inclui "tine feed" após a operação.

readln - inclui "line feed" após operação

Exemplo:

```
readln(QUANTIDADE);
```

OBS.: Quando for necessário mais de uma entrada read ou readln, os dados a serem digitados não devem ser separados por vírgulas e sim por espaço.

```
readln(COMPRIMENTO,LARGURA,ALTURA);
```

OBS.: É conveniente pedir apenas uma entrada de dado por cada comando read ou readln, para que não ocorra inconvenientes na entrada de textos.

Exemplo:

```
Program Prog02;

var
     C,L,A : integer;

begin

    write('DIGITE COMPRIMENTO');
    readln(C);
```

```
    write('DIGITE LARGURA');
    readln(L);

    write('DIGITE ALTURA');
    readln(A);

    writein('VOLUME = ',c*L*A,' cm3');

end.
```

OBS.: Quando usado o comando writeln sem variáveis a serem impressas, causará apenas o envio de um line feed (mudança de linha) para a tela ou impressora.

ESTRUTURA DE DECISÃO:

COMANDO IF:

Na estrutura de decisão em PASCAL, utiliza-se os comandos if / then / else / end, conforme apresentados nos exemplos abaixo.

Estrutura de decisão SIMPLES (if/ then / end):

Exemplo:

```
Program Prog03;
{Estrutura de Decisão Simples}

var

   VALOR : real;

begin

    write('DIGITE UM VALOR NEGATIVO');
    readln(VALOR);

    if VALOR > 0 then
      begin
         writeln('FOI DIGITADO UM VALOR POSITIVO !');
      end;

end.
```

OBS: Observe o END com ponto e vírgula após o if.

Estrutura de decisão COMPOSTA (if / then / else / end):

Na estrutura de decisão composta em PASCAL, é necessário tomar alguns cuidados. O comando end que precede o else do exemplo abaixo, não deve levar ponto e vírgula. O ponto e vírgula ali posicionado, indicaria que já chegou ao fim da atuação do comando if anterior. Portanto, somente o segundo e último end deve receber o ponto e vírgula. Veja os exemplos a seguir:

```
Program Progo4;
{Estrutura de Decisão Composta}

var
    VALOR : real;

begin

    writel'DIGITE UM VALOR: ');
    readln(VALOR)

    if VALOR > lo then
      begin
        writeln('O VALOR É MAIOR QUE 10');
      end
    else
      begin
        writeln('o VALOR É MENOR DO QUE lo');
      end;
end.
```

```
Program Prog05;

var
   VALOR : real;

begin

   write('DIGITE UM VALOR POSITIVO MENOR QUE 100:');
   readln(VALOR)

   if (VALOR > 0) and (VALOR < 100) then
     begin
       writeln('fol DIGITADO CORRETAMENTE')j
     end
   else
     begin
       writeln('fol DIGITADO ERRADO');
     end;

end.
```

COMANDO CASE:

O comando case é extremamente importante para estruturação de um programa que possua diversas opções de execução, tomando-o bem legível e estruturado, evitando o uso repetido do if.

Exemplo:

```
Program Prog06;

var

   VALOR : integer;

begin
   write('DIGITE UM NÚMERO ENTRE 0 e 2 INCLUSIVE');
   readln(VALOR);

   case VALOR of

     0 : begin
           writeln('NÚMERO DIGITADO = ZERO');
         end;

     1 : begin
           writeln('NÚMERO DIGITADO = UM');
         end;

     2 : begin
           writeln('NÚMERO DIGITADO = DOIS');
         end;

   end; {case)

end.
```

O comando case oferece uma opção de else (se não). Veja o exemplo abaixo:

```
Program Prog07;

var

    VALOR : integer;

begin
   write('DIGITE UM NÚMERO ENTRE 0 E 2 INCLUSIVE');
   readln(VALOR);

   case VALOR of
```

```
      0 : begin
            writelnl'Número Digitado = zero');
          end;

      1 : begin
            writeln('Número Digitado = um');
          end;

      2 : begin
            writeln('Número Digitado = dois');
          end;

      else

         begin
           writeln('VOCÊ DIGITOU FORA DA FAIXA DE 0 A 2 ');
         end;

      end; {case}

end.
```

OBS: A variável do case pode ser de qualquer tipo até agora citado. Vejamos um exemplo do comando case com uma variável de seleção do tipo char:

```
Program Prog08;

var
   RESP : char;

begin
   readln(RESP);

   case RESP of

     'S' : begin
             writeln('vocÊ DIGITOU S');
           end;

     'N' : begin
             writeln('vocÊ DIGITOU N');
           end;

   end; {case}

end.
```

OBS: A presença dos apóstrofos limitando as letras S e N indica que a comparação deve ser feita em relação a um string.

FUNÇÕES MATEMÁTICAS:

abs(X) - Retorna o valor absoluto de X.

arctan(X) - Retorna o arco tangente de X em radianos.

cos(X) - Retorna o cosseno de X (X deve estar em radianos).

exp(X) - Retoma o valor de e elevado à X.

frac(X) - Retorna a parte decimal (fracionária) de X.

int(X) - Retorna o valor inteiro de X.

ln(X) - Retorna o logaritimo natural ou neperiano de X.

sin(X) ~ Retorna o seno de X (X em radianos).

sqr(X) ~ Retoma o quadrado de X.

sqrt(X) - Retoma a raiz quadrada de X.

POTENCIAÇÃO:

A potenciação $A = B^C$ (A é igual a B elevado à C) obtem-se da seguinte forma:

A := exp(C*ln(B))

FUNÇÕES DE MANIPULAÇÃO DE STRING :

concat(X,Y,...) , Concatena as strings X, Y, ...

OBS : Pode-se também usar a soma (+) de strings.

Exemplo:

A :='AERO';

B := 'DINAMICA';

concat(A,B) retorna AERODINAMICA.

copy(X,Y,Z) - Extrai Z caracteres a partir do Y-ésimo caractere da string X.

Exemplo:

A := 'AERODINAMICA';

copy(A,5,4) retoma DINA.

lenght(X) - Retorna o comprimento da string X.

Exemplo:

A := 'AERODINAMICA';

lenght(A) retorna 12.

pos(X,Y) - Retorna a posição da string X dentro da string Y. Se não for encontrada retornará 0.

Exemplo:

A := 'FUMEC';

B :='M';

pos(A,B) retorna 3.

delete(A,B,C) - Remove C caracteres a partir da posição B da string A.

Exemplo:

A:='PANCADARIA';

delete(A,3,3); a variável A conterá Padaria.

insert(A,B,C) - Insere A na string B, a partir da posição C.

Exemplo:

A :='NCA';

B :='PADARIA';

C :=3;

insert(A,B,C); a variável B conterá PANCADARIA.

str(X,Y) - Converte o valor numérico de X num string Y.

Exemplo:

x := 15;

str(X,Y); a variável Y conterá a string 15.

val(X,Y,Z) - Converte a string X em número e atribui à variável Y. A variável Z deve ser inteira e indica o sucesso ou não da conversão. Em caso de sucesso, Z conterá zero, caso contrário, conterá a posição do caractere que ocasionou o problema.

Exemplo:

X :='1234';

val(X,Y,Z); a variável Y conterá o número 1234 e Z valerá zero.

FUNÇÕES DE TRANSFORMAÇÃO:

chr(x) - Retorna o caractere cujo código ASCII é X.

Exemplo:

chr(42); retornará o sinal *.

round(X) - Provoca o arredondamento na primeira casa decimal.

Exemplos:

x := 18.5;

round(X) retorna 19.

X := 17.4;

round(X) retoma 17.

trunc(X) - Retorna o maior inteiro não maior que X (menor ou igual a X)

FUNÇÕES ESPECIAIS:

keypressed - É uma função booleana que devolve o valor true se alguma tecla estiver sendo pressionada e false em caso contrário.

random - Retorna um número real randômico entre 0 e 1.

random(X) - Retoma um número real randômico entre 0 e X.

upcase(X) - Se o caractere X do tipo char for minúsculo, converte-o para maiúsculo. Caso contrário, não há alteração.

OBS: As funções e os procedimentos descritos até aqui são chamados predefinidos ou predeclarados. Mais adiante estudaremos como se utilizam as funções e os procedimentos declarados pelo próprio usuário.

ESTRUTURA DE REPETIÇÃO:

COMANDO WHILE:

os comandos a serem repetidos , ou seja, aqueles que serão submetidos ao loop, devem estar limitados por begin e end.

A condição de repetição é testado antes, e portanto, os comandos envolvidos no while podem não ser executados nenhuma vez.

Exemplo: Programa para calcular e imprimir os números inteiros de 0 a 100.

```
Program Prog09;

var

   I : real;

begin

    I:=o;

    while 1 <= 100 do
        begin
           writeln(I);
            I := I+1;
       end;

end.
```

As condições que acompanham o while podem ser compostas de not, and, or e xor, tal como o comando if.

COMANDO GOTO:

O comando goto desvia o processamento do ponto em que for encontrado, para outra linha do programa que está marcada com um label (rótulo). Este label necessita ser declarado no início do programa.

Este recurso (goto / label) nos permite abandonar de forma "radical" um loop, antes que seja executado até o final por vias normais. O goto é um comando "forte" que pode abandonar não só o loop controlado pelo while como também os loops gerados pelos outros dois comando de repetição que estudaremos mais adiante (for e repeat).

Exemplo:
O programa abaixo permite a entrada de no máximo dez números para serem somados dentro da variável SOMA, ou até ser digitado zero:

```
Program Prog10;
var
    I, SOMA : integer;
label
    FIM;
begin
    1 := 0;
    SOMA := 0;

    while I <= lo do
        begin
           write('DIGITE VALOR: ');
           readln(NUm);

           if NUM o 0 then
              begin
                 SOMA := SOMA + NUM;
              end
            else
              begin
                  goto FIM;
              end;

          I := I + 1 ;
        end;

        FIM:
           writeln ('SOMA DOS NÚMEROS= ', SOMA);

end.
```

COMANDO FOR:

As linhas do programa a serem controladas pelo comando for devem estar entre begin e end;

A variável de controle pode ser somente do tipo integer.

O passo (step) é sempre +1 ou -1.

Exemplo:

Programa para calcular e imprimir os valores numéricos de 1 a 10

```
Program Prog11;
```

```
var
    I : integer;

begin

    for I :=1 to lo do
      begin
         writeln(I);
      end;

end.
```

Desejando decrementar a variável de controle, teremos a seguinte configuração:

```
Program Prog12;

var
    I : integer;

begin

    for I := 10 downto 1 do
      begin
         writeln(I);
      end;

end.
```

Quando houver necessidade da existência de um for dentro da estrutura de outro for, a escrita do programa ficará assim:

```
Program Prog131

var
   I,J ; integer;

begin

    for I := 1 to 10 do
      begin
          .
          for J := 1 to 10 do
            begin
                .
                        .
                .
            end;
```

```
        .
    end;

end.
```

OBS: O comando repetitivo for deve ser utilizado nos locais em que a quantidade de vezes a ser repetida já está determinada antes da sua execução. Isto significa que não devemos fazer atribuições à variável de controle do for dentro da própria estrutura. Esta variável pode ser utilizada para ser impressa ou para cálculos, mas jamais deve receber algum valor.

COMANDO REPEAT / UNTIL:

O comando repeat controla repetições de trecho de programa tal como o for, mas há duas diferenças fundamentais:

- O número de vezes que se repetirá as linhas de programa situadas dentro do repeat / until não precisa estar predefinido. Os próprios comandos situados entre o repeat e o until podem determinar a saída ou não do loop de repetição.

- A verificação da suficiência ou não da condição de repetição é feita no fila do loop.

Exemplo:

Suponha um programa que só termina quando for digitado um número negativo:

```
Program Progl 4;

var
   VALOR : real;
begin
   VALOR := 0;

   repeat

        writel'DIGITE UM NÚMERO NEGATIVO') ;
       readln(VALOR);

   until VALOR < 0;

end.
```

OBS:

- Os comandos a serem repetidos não necessitam ser limitados por begin e end.

- As condições que acompanham o repeat / until podem ser compostas de not, and, or e xor, tal como no comando if.

- Os comandos internos ao repeat / until são executados pelo menos uma única vez.

Outro exemplo:

```
Program Prog15 ;

var
    I : real;

begin
    I:=0;

    repeat

      writeln(I);
      I := I + 1;

    until I > j 00
end.
```

OBS:

No repeat e while, o incremento ou decremento da variável de controle deve ser providenciado internamente na lógica por quem desenvolveu o programa.

RESUMINDO:

Os três tipos de comandos de controle de repetição devem ser utilizados visando às seguintes características:

- Comando for:
Utiliza-se no caso de saber o número de repetições antecipadamente e o passo de incremento ou decremento for sempre 1. Não se deve alterar o valor da variável de controle.

- Comando repeat:

Controla a condição de repetição após sua execução. Este comando é sempre executado pelo menos uma vez.

- Comando while:

Controla a condição de repetição antes da sua execução. Pode ocorrer situações em que o comando while não seja executado.

MATRIZES:

A declaração de uma matriz no Turbo Pascal é obrigatória. Por exemplo, para se declarar uma matriz MAT de uma única dimensão (vetor) composta de 50 números inteiros, seria feito da seguinte forma:

```
var
   MAT : array[1..50] of integer;
```

No caso de uma matriz bidimensional de 50 linhas e 100 colunas, composta de string de 10 caracteres, seria assim:

```
var
   MAT : array[1..50 , 1..100] of string[10];
```

OBS:
Os elementos que formam as matrizes são conhecidos com variáveis indexadas. Estas variáveis além de serem indexadas, podem ser utilizadas de todas as formas que são utilizadas as variáveis comuns (cálculos, impressões, etc...).

Exemplo:

programa para receber via teclado 5 elementos para serem armazenados no Vetor VET e somá,los na variável TOTAL:

```
Program Prog16;

var
    VET : array[1..5] ofinteger;
    TOTAL,I : integer;

begin
    TOTAL := 0;
     for I :=1to 5 do
       begin
          write('DIGITE ELEMENTO ',I,': ');
          readln(VET[I]);
          TOTAL := TOTAL + VET[I];
       end;

     writeln(TOTAL);

end.
```

Outro exemplo:

Programa para receber via teclado os elementos da Matriz MAT, que possui 10 Linhas e 5 colunas:

```
program Prog 17;
var
    LINHA, COLUNA : integer;
    MAT : array[1.. lo , 1..5] ofinteger;
begin
    for LINHA := 1 to lo do
      begin
         for COLUNA := 1 to 5 do
            begin
               writeI'DIGITE ELEMENTO ',LINHA,'X',COLUNA,':
               readln(MAT[LINHA,COLUNA]);
            end;
      end;
end.
```

Para transferir todos os elementos de uma matriz MAT para outra matriz chamada COPIA, pode-se utilizar a técnica de acesso a todos os elementos da matriz MAT, usando geralmente o comando for. Supondo que a matriz MAT de uma única dimensão (vetor) estivesse anteriormente carregada, poderíamos fazer da seguinte forma:

```
.

.
for I := 1 to lo do
    begin
       COPIA[I] := MAT[I];
    end;
.
.
```

ou simplesmente:

```
COPIA := MAT;
```

A atribuição acima pode ser feita com qualquer matriz, independente de sua dimensão.

PROCEDIMENTOS:

Um procedimento é uma sub-rotina que fica desmembrada da lógica principal do programa e que geralmente é necessário sua execução em vários pontos do programa. Para isto, basta determinar o processamento do procedimento através da referência ao seu nome, que deve ser devidamente declarado e escrito no início do texto do programa. Nas linhas em que for encontrado o nome do procedimento, será determinado a sua execução, e após ser executado, ocorrerá o retomo do processamento para a próxima linha após o nome que ordenou a execução.

Os procedimentos devem vir no início do programa (declarados pelo comando procedure), pois a chamada efetuada pelo programa principal utiliza o próprio nome do procedimento. Isto é necessário por que a medida que o compilador vai avançando no seu trabalho, ele confirma as palavras e as sentenças, para verificar se não há erro de sintaxe. Caso o compilador encontre uma palavra desconhecida, toma-se necessário analisar se foi erro do programador ou uma palavra nova criada por ele. Se a declaração do procedimento não viesse antes da sua utilização, o compilador não teria como distinguir entre um erro e a chamada de um procedimento.

```
program Prog18 ;

procedure MENSAGEM;
begin
   writeln ('LTPI - LINGUAGEM TURBO PASCAL');
end;

{--* PROGRAMA PRINCIPAL *--}
begin

    {-- MANDA EXECUTAR O PROCEDIMENTO --}
    MENSAGEM;
    {-- ESTE É O PONTO DE RETORNO DO PROCEDIMENTO --}

end.
```

OBSERVAÇÕES:

O Pascal em si possui poucos comandos e algumas liações e procedimentos pré-declarados dentro de si mesmo. O usuário pode criar a sua biblioteca de procedimentos e enriquecer o vocabulário do TURBO PASCAL. A força desta linguagem passa a depender da pesquisa e da experiência do seu usuário.

Não se pode desviar de um procedimento para outro através do comando goto, ou seja, de um subprograma para outro.

VARIAVEIS LOCAIS E GLOBAIS:

VARIAVEIS LOCAIS:

Para facilitar o entendimento do conceito de variáveis locais, imagine um grande programa (com vários procedimentos) escrito simultaneamente por diversos programadores. Cada programador não precisa preocupar-se com os nomes das variáveis que outro esteja utilizando para desenvolver o seu procedimento. Cada variável só vale dentro do seu respectivo procedimento e deve ser declarada cada vez que se fizer presente. Os valores contidos em variáveis locais perdem-se após a execução do procedimento.

VARIAVEIS GLOBAIS:

Estas variáveis valem para qualquer parte do programa, exceto no caso de ser novamente declarada como variável local dentro de um procedimento. Toda declaração de variáveis é feita no início do programa.

Exemplo:

```
program Prog19;

{-- DECLARAÇÃO DE VARIÁVEIS GLOBAIS--}

var

I: integer;

procedure TESTE;

var

    I : integer;
begin
   I := 5;
end;

{--* PROGRAMA PRINCIPAL *--}
begin

   I := 1;
   TESTE;

   writeln(I);

end.
```

OBSERVAÇÃO: Ao dar RUN no programa anterior, o valor impresso na tela deverá ser 1 e não 5, apesar da variável I ter sido também declarada dentro do

procedimento (variável local que perde o seu valor após a execução do procedimento). Esta declaração tornou-a uma variável local e portanto, distinta da I global. Se retirássemos a declaração da variável I que está dentro do procedimento esta seria tratada como variável global e o resultado a ser impresso seria 5.

PASSAGEM DE PARÀMETROS:

Um procedimento pode trabalhar com valores, mesmo que não sejam usadas variáveis globais. Para isto temos um recurso chamado passagens de parâmetro. O grande poder dos parâmetros está nesta correta passagem de valores entre o programa principal/procedimento e o procedimento chamado.

Observe o exemplo abaixo:

```
Program Prog20;

procedure MULT(A,B : real);
begin
    writeln(A * B);
end;

{--* PROGRAMA PRINCIPAL *--}
begin

     MULT (3,5);

end.
```

OBS.: O programa anterior consiste em um procedimento que imprime o valor do produto de dois números recebidos pela passagem de parâmetros e um programa principal que manda executar o procedimento MULT enviando os valores numéricos 3 e 5 como parametro.

Observações:

- Não foi utilizada nenhuma variável global.

- As variáveis A e B do procedimento não precisam ser declaradas, pois são variáveis de passagem e foram declaradas no início do procedimento na linha do comando procedure.

- O programa principal não utiliza nenhuma variável.

- As variaveis de passagem recebem os valores na mesma ordem em que são declaradas como parâmetros. Isto quer dizer que, para efeito de cálculo dentro do procedimento MULT, o A assumiu o valor 3 e B assumiu o valor 5. . .

Outro exemplo:

```
Program Prog21;

procedure MULT(A,B: REAL);
var
    C : real; (-- VARIÁVEL LOCAL -- j.

begin
    C ;= A * ~~
    writeln(C);
end.

{-* PROGRAMA PRINCIPAL *--}
begin

    MULT(3,5);

end.
```

OBS.: Neste programa foi incluída a variável C, que é local, para armazenamento temporário do resultado da multiplicação. É interessante separar cada um dos módulos do programa de forma bem visível para facilitar o acompanhamento, principalmente se os programas forem ficando extensos.

Utilizando somente variáveis globais, não haveria necessidade de passar parâmetros para o procedimento e o programa ficaria assim:

```
program Prog22;

var
    A,B,C : real (-- VARIÁVEIS GLOBAIS --)

procedure MULT;
begin
    C := A * B;
     writeln (C);
end;

{--* PROGRAMA PRINCIPAL *--}
begin

    A := 3 ;
    B := 5;
    MULT;

end.
```

OBS.: Neste caso, o writeln poderia estar no programa principal que iria imprimir do mesmo modo o mesmo resultado, desde que estivesse posicionado após a chamada do procedimento.

Há mais um modo de trabalhar com PROCEDIMENTO. Até agora, os valores eram passados a partir de uma chamada para o PROCEDIMENTO. O processo inverso, ou seja, passar valores de um procedimento para o programa principal ou procedimento que o chamou, pode ser efetuado mediante uma pequena alteração na declaração local, acrescentando o comando var.

Exemplo:

```
program Prog23;

var

    A,B,C : real;

procedure MULT(X,Y : real; var Z: real);
begin
   Z := X * Y;
end;

{--* PROGRAMA PRINCIPAL *--}
begin

    A := 3;
    B := 5;
    MULT(A,B,C);
    writeln(C);

end.
```

OBS.: Quando se usa este tipo de declaração do procedimento (usando var), permite-se que um valor seja retomado de um procedimento para utilização fora dele.

No exemplo, a variável local Z passou o valor 15 de volta para a variável global C, quando o processamento retornou do procedimento para o programa principal.

Vejamos os dois exemplos a seguir:

```
program Prog24;

var
    A,B : real;

procedure SOMA(X,Y : real);
begin
     X := X + Y;
     writeln (X);
end;
```

```
{--* PROGRAMA PIIINCIPAL *--}
begin

    A := 2;
    B := lo;
    SOMA (A,B);
    writeln (A);

end.
```

OBS.: Os resultados obtidos ao rodar o programa acima são, respectivamente 12 e 2. Observe que a variável global A permanece com o mesmo valor, pois houve apenas a passagem de parâmetro no sentido de ida.

```
program Prog25;

var
     A,B : real;

procedure SOMA(var X,Y : real);
begin
     X := X + Y;
     writeln (X);
end;

{--* PROGRAMA PRINCIPAL --}
begin

     A := 2;
     B := lo;
     SOMA(A,B);
     writeln (A);

end.
```

Desta vez, os resultados impressos são respectivamente 12 e 12. Devido a presença do var na declaração, o valor da variável X foi passado de volta para a variável A ao retornar para o programa principal.

FUNÇÕES:

A estrutura de uma função (function) é muito parecida com um procedimento. Pode-se imaginar que uma função é um procedimento com características especiais quanto ao retorno de valores. No Turbo Pascal, uma função pode ser tão bem elaborada quanto um programa qualquer.

Uma característica que distingue uma função de um procedimento é que a função pode ser impressa, atribuida ou participar de cálculos como se fosse uma variável qualquer.

```
program Prog26;

var
    A,B,C: real;

function PROD(X,Y : real) : real;
begin
    PROD := X * Y;
end;

{--* PROGRAMA PRINCIPAL *--}
begin

    A := 2;
    B := 110;
    C := PROD(A,B);
    write (C);

end.
```

OBS.: O resultado impresso será 20.

Na declaração da função, além do que está dentro dos parênteses, há mais uma definição, indicando que a função é do tipo real, ou seja, retorna ao local onde foi chamada, assumindo a condição de uma variável real. Neste caso, a função só assume valores coerentes com essa declaração.

Como uma função pode ser impressa, atribuida ou participar de cálculos como uma variável qualquer, isto nos permite, em vez de utilizar a variável C, imprimir diretamente a função da seguinte maneira:

```
WRITELN (PROD(A,B));
```

E teremos o mesmo resultado.

Outro exemplo:

Vejamos uma função que soma dois números inteiros digitados via teclado:

```
Program Prog27;

var

  x,y;integer;
```

```
function SOMA(A,B : integer) : integer;
begin
  SOMA := A + B;
end;

 {-* PROGRAMA PRINCIPAL *--}
begin

  readln (X);
  readln (Y);
  writeln (SOMA (X,Y));

end.
```

OBS.: Na declaração da função podemos acrescentar a cláusula VAR e obter os mesmos recursos adicionais do procedimento feito desta forma.

```
program Prog28;

var

  A,B,C: real;

function PROD(X : real; var Y : real) : real;
begin
  PROD := X * Y;
  X := 1
  Y := 1
end;

{--* PROGRAMA PRINCIPAL *--}
begin

  A := 2;
  B := lo;
  writeln (PROD (A,B));

end.
```

Após a chamada da função, as variáveis X e Y terminam valendo 1. Mas na declaração da função somente a variável Y recebe VAR. Isto implica que o valor final de Y é passado de volta ao local onde foi solicitada a função, fazendo com que B passe a tomar-se 1 em vez do valor original l0 Isto tudo não ocorreu com a variável A, que continua valendo 2 tal como antes.

No caso de uma função ser do tipo STRING, deve-se definir primeiro o tipo de dado e depois colocar o nome deste tipo na declaração da função.

```
Type
```

```
  STR1O : string[10];

function EXEMPLO(         ) : STR1O;
```

FUNÇÕES E PROCEDIMENTOS PRÉ-DECLARADOS PARA COTROLE DE TELA

O Turbo Pascal possui algumas funções e procedimentos pré-declarados para controle da tela:

clrscr - Comando para limpar a tela.

clreol - Este comando apaga todos os caracteres da linha que estão à direito do cursor.

delline - Apaga a linha onde está o cursor e causa scroll nas linhas seguintes de tal modo que preencha a linha deletada.

insline - Este comando funciona de modo oposto ao delline. Insere uma linha vazia na posição onde está o cursor. Provoca scroll nas linhas seguintes.

gotoxy(X,Y) - Movimenta o cursor para coluna X, linha Y da tela. A contagem das colunas e linhas começa no 1, ou seja, o canto superior esquerdo possui coordenadas (1 ,1).

lowvideo - Diminui a luminosidade dos caracteres na tela.

highvideo - Aumenta a luminosidade dos caracteres na tela.

norrnvideo - Retoma a luminosidade normal.

OUTROS RECURSOS:

delay(X) - Provoca uma pausa no processamento equivalente a X millissegundos. X deve ser definido como número inteiro.

exit - Abandona o bloco corrente. Estando numa sub-rotina, retoma ao bloco onde foi feita a sua chamada. Se usado no programa principal, termina a execução do mesmo.

Halt - Interrompe o processamento do programa e retoma ao nível de sistema operacional.

randomize - Inicializa o gerador de números aleatórios com um valor aleatório.

```
prograrn Prog29;

var
```

```
    I: integer;

begin

    clrscr;

    for I := 1 to 20 do

    begin
        gotoxy ( lo,();
         writeln('** LTP1 - FUMEC **');
    end;

end.


prograrn Prog30;

begin

    clrscr;
    gotoxy(30,08); writel'li Incluir ');
    gotoxy(30, lo); writel'2] Alterar ');
    gotoxy(30,12); ~vritel'3] Excluir ');
    gotoxy(30,14); writel'4] Pesquisar');
    gotoxy(30,16); writel's] Finalizar');
    gotoxy(30,20); ~vritel'opção ? ');

end. '
```

DECLARAÇÃO DE OUTROS TIPOS ESPECIAIS DE DADOS (TYPE):

- SCALAR (ESCALAR);

-SUBRANGE(SUBINTERVALO);

-SET(CONJUNTO);

- RECORD (REGISTRO).

SCALAR

"SCALAR TYPE" é um tipo de dado que se assemelha a um vetor com seus elementos ordenados, sendo que o próprio usuário pode definir os seus

tipos de dados. Por exemplo, se criar-mos dados do tipo Dias da Semana (DOMINGO, SEGUNDA, etc), significa que a variável associada a este tipo, somente poderá assumir os dados que são os Dias da Semana.

Exemplo:

```
program Prog3 1 ;

type
    Semana                                                    =
(DOMINGO,SEGUNDA,TERÇA,QUARTA,QUINTA,SEXTA,SABADO);

var
   DIA : Semana;

begin

    for DIA := DOMINGO to SABADO do
        begin
            .........
                .........
                .........
           .........
       end;

end.
```

Observações:

- O comando type foi usado para a declaração (definição) do tipo.

- O tipo de dado Semana é composto pelos valores definidos no programa, que são os Dias da Semana (de DOMINGO a SÁBADO).

- A variável DIA assume valores que são os Dias da Semana e é usada para controlar o comando for.

- A repetição é executada sete vezes e serão atribuídas à variável DIA todas as sete possibilidades definidas.

Um escalar definido pelo usuário não pode ser impresso através de um comando write ou writeln, devendo ser detectado de modo indireto como no exemplo a seguir., que se utiliza das 4 estações do ano:

```
prograrn Prog32;

type
   Estacoes = (VERÃO, OUTONO, INVEMIO, PRIMAVERA);

var
```

```
   EST : Estacoes;

begin

    for EST := PRIMAVERA downto VERAO do
        begin

            case EST of

                  VERÃO              : begin
                                          Writeln('MUITO CALOR');
                                       end;

                 OUTONO              : begin
                                         Writeln('AS FOLHAS CAEM');
                                       end;

                 INVERNO             : begin
                                          Writeln('MUITO FRIO');
                                       end;

                  PRIMAVERA          : begin
                                          Writeln('MUITAS FLORES');
                                       end;

           end;

        end;

end.
```

As saídas do programa anterior, serão as seguintes:
MUITAS FLORES
MUITO FRIO
AS FOLHAS CAEM
MUITO c ALOR

No exemplo acima no comando for foi utilizado a cláusula downto, que inverteu a ordem de impressão dos dados tipo Estações.

SUBRANGE (SUBINTERVALO):

Os tipos integer, real, byte, char e boolean também fazem parte dos ESCALARES, sendo que já estão PREDEFINIDOS.

Todos os dados do tipo escalar apresentam subintervalos (subrange) em que a sua existência é VÁLIDA. O tipo de definição de dados conhecido como subrange (subintervalo) possui como característica principal, a utilização de apenas trechos de algo maior e mais completo.

```
program Prog33.

type
   Nalunos = 1..50;

var
    ALUNO . Nalunos;

begin

     writeln('DIGITE O NÚMERO DO ALUNO:');
     readln(ALUNO);
     writeln(ALUNO);

end;
```

No programa anterior, suponha que o número de alunos que possam pertencer à turma noturna de LPT1 seja 50, surgindo assim, a limitação do seu valor entre 1 e 50. Portanto, não existe a possibilidade de a variável encarregada de controlar este número ultrapassar o limite inferior e o superior. Por esse motivo, declarou-se que o tipo de dados Nalunos está na faixa de 1..50 e que, a variável ALUNO é do tipo Nalunos. O tipo Nalunos é um subintervalo de um ESCALAR predefinido.

Outro Exemplo:

```
prograrn Prog34;

type
   Minusc = 'a'..'z';
   Meses = (JAN,FEV,MAR,ABR,MAI,JUN,JUL,AGO,SET,OUT,NOV,DEZ);
   Prisemest = JAN..JUN;
   Segsemest = JUL..DEZ;
```

Observações:

- No exemplo anterior, o tipo de dados Minusc é um subintervalo do escalar char (Letras minúsculas do Alfabeto).

- O tipo de dados Prisemest (meses do primeiro semestre) e Segsemest (meses do segundo semestre) são subintervalos de um tipo de dados definidos como Meses.

-O Turbo Pascal checa a validade dos subintervalos somente quando a diretiva de compilação R estiver ativada.

CONJUNTO SET:

Os dados do tipo conjunto são reuniões de diversos elementos do mesmo tipo (escalar), exceto os números reais. Vejamos o exemplo abaixo:

```
prograrn Prog35 ;

type ,,
   Numeros = set of byte;

var
   PRIMO, ÍMPAR, PAR, FINAL1, FINAL2 : Numeros;

begin

   PRIMO := [1,2,3,5,7]; ,
   ÍMPAR :# [1,3,5,7,9];  ;  '
   PAR := [2,4,6,8]; /
   FINAL1 := PRIMO + ÍMPAR;
   FINAL2 := PRIMO * PAR;

end.
```

O programa acima possui as seguintes características:

- Define-se Numeros como do tipo de dado set of formado por números do tipo byte.
- Definem-se as variáveis de conjunto PRIMO, ÍMPAR, PAR, FINAL1 e FINAL2 como do tipo Numeros.

- Atribuem-se os respectivos números (maior que 0 e menor que 10).

- Executa-se a operação de união (+) e intersecção (*) entre os conjuntos.

- As variáveis de conjunto FINAL1 e FINAL2, conterão os seguintes elementos:

FINAL1 : 1,2,3,5,7,9
FINAL2 : 2

- A operação dos conjuntos obedece às seguintes notações:

Decisões lógicas com conjuntos:

A = B ---> A igual a B
A <> B ---> A diferente de B
A >= B ---> A contém B
A <= B ---> A está contido em B
A in B ---> .A pertence aB

Exemplo utilizando o operador in:

```
program Prog36;

var
   LETRA : char;

begin

    write('DIGITE UMA LETRA MINÚSCULA ENTRE a e f :');
    readln(LETRA);

    if LETRA in ['a'..'f] then
      begin
          writeln('POSITIVO');
      end
else
       begin
           writeInl'NEGATIVO');
       end;

end. '
```

Observações:

- O programa anterior imprime 'POSITIVO' se a letra digitada estiver conforme o que foi solicitado na mensagem. Caso contrário, o comando if desvia o processamento para imprimir a mensagem 'NEGATIVO'.

- Um detalhe importante é que o set (conjunto) foi definido na própria sentença do if. Poderia também ser declarado na forma normal, utilizando-se type e var.

ARQUIVOS DE DADOS EM DISCO:

Como sabemos, ao desligar-mos o computador, as informações contidas na memória RAM serão perdidas, pois trata-se de uma memória volátil.

A utilização de uma unidade de leitura e gravação em disco magnético, possibilita o armazenamento magnético permanente daqueles dados da memória RAM, que seriam perdidos ao desligar-mos o computador. Além disso, os discos magnéticos rígidos (WINCHESTERS) possuem espaço de armazenamento várias vezes maior que o da memóriaRAm.

O Turbo Pascal possui três tipos básicos de arquivos em Disco:

- Arquivo de texto;

- Arquivo com tipo definido;

- Arquivo de dados sem tipo definido;

O objetivo do estudo nesta apostila, será o arquivo de dados sem tipo definido (acesso randômico ou aleatório).

DEFINIÇÕES:

REGISTRO:

É um conjunto de Campos de Dados.

CAMPOSDEDADOS:

Os campos de dados são diferentes dados ou informações que relacionados, constituem o Registro. Os Campos de Dados podem ser de vários tipos (real, string, etc...), e devem ser dimensionados de acordo com a necessidade anual e fatura.

ARQUIVODEDADOS:

Agrupamento organizado de informações armazenadas em bytes, que podem ser acessadas quando necessário.

É um conjunto organizado de Registros de informações. Os registros são independentes um do outro, mas possuem sua estrutura intema de campos igual.

BUFFER

Buffer é uma região de memória que tem a mesma extensão e divisão dos campos do registro associado a ele. Esta região serve de "molde" intermediário para se encaixar o registro.

Durante o processamento, ao ser lido um registro, os seus dados serão trazidos para o buffer, tomando-os disponíveis para o processamento.

Para gravar um registro, devemos prepará-lo no buffer e depois mandar gravá-lo no Disco. Isto faz com que os dados que estavam no buffer sejam transferidos para o arquivo. A leitura de um arquivo em disco bem como sua gravação são duas das operações mais lentas que um computador executa. O tempo gasto para uma unidade de disco localizar dados, corresponde a um período de anos para um microprocessador. Pequenas partes da memória denominadas "Buffers" são reservadas para que os dados sejam usados em operações de disco. Os "Buffers" agilizam o processamento, reduzindo a quantidade de leituras e de gravações em disco.

ESTRUTURA DE UM PROGRAMA EM TURBO PASCAL

CABEÇALHO DO PROGRAMA:

Nome do Programa

Diretivas do Compilador

SEÇÃO DE DADOS:

Declarações de Constante

Declarações de Tipo

Declarações de Variável

Declarações de Label

SEÇÃO DE ROTINAS:

Procedures

Funções

SECÃO DA LÓGICA PRINCIPAL.-

Bloco de Programa

DECLARAÇÃO DE UM ARQUIVO DE DADOS DE ACESSO RANDÔNICO

Um registro utilizado no arquivo randômico deve ter a sua extensão física fixa, ou seja, cada campo deve possuir um número fixo de bytes. Devido a esta razão, precisamos definir a máxima extensão dos campos que irão compor este registro.

Exemplo:

Considere um arquivo de um cadastro de ARMAS DE FOGO de uma loja especializada em Caça e Pesca. Este cadastro possui campos que indicam os seguintes

- Tipo de arma (RevolveR, Pistola, Carabina, Espingarda...);
- Fabricante ou marca (Taurus, Rossi, CBC, IMBEL, etc);
- N° de série da arma (ex : FJ70514);
- Calibre (.38, .32, .22, .380, 7.65, 12, 6.35, etc...);
- Acabamento (Inox, Acetinada, Oxidada, etc ...);

- Capacidade (6 cartuchos, 5, 13, etc...);
- N° de canos (1 ou 2).

Vamos agora definir a máxima extensão dos campos que irão compor os registros do arquivo de armas:

Tipo 10 bytes;
Marca 6 bytes;
Série 7 bytes;
Calibre 4 bytes;
Acaba 9 bYtes;
Capacid 2 bYtes;
Ncanos 1 byte.

A declaração da estrutura deste arquivo ficaria assim:

```
type
   ARMA = record
          TIPO          : string[10];
          MARCA         : string[6];
          SERIE         : string[7];
          CALIBRE       : real;
          ACABA         : string[9];
          CAPACID       : integer;
          NCANOS        : byte;
   end;

var
    Bufferarma : ARMA;
```

Observações:

- Os campos TIPO, MARCA, SERIE, CALIBRE, ACABA, CAPACID, e NCANOS, formam um registro do tipo ARMA.
- Foi necessário criar e identificar um buffer para leitura e gravação. Este buffer chama-se Bufferarma e sua formatação (molde) é igual a do registro ARMA.

Para trabalharmos inicialmente, com os campos de um registro sem, no entanto, gravar ou ler dados, devemos saber que uma Variável de Campo não pode ser diretamente referenciada. Para fazer-mos a referência precisamos colocar o nome do buffer seguido de um ponto (.) e o nome do campo.

Exemplo:

```
program Prog37;

type
   ARMA = record
          TIPO                :  string[10];
```

```
        MARCA          : string[6];
        SERIE          : string[7];
        CALIBRE        : real;
        ACABA          : string[9];
        CAPACID        : integer;
        NCANOS         : byte;
    end;


var
    Bufferarma : ARMA;

begin

    write('DIGITE O TIPO DA ARMA:');
    readln(Bufferarrna, TIPO);
    wriiteln(Bufferarma, TIPO);
end.
```

Observações:

- O campo Bufferarma.TIPO foi utilizado para armazenar, mesmo não existindo operações em disco, o dado digitado via teclado, e captado pelo readln.

- O Turbo Pascal admite mais de um campo com o mesmo nome desde que sejam de Buffers diferentes. Para que não haja problemas de diferenciação, toma-se necessário a colocação do buffer correspondente antes do nome do campo.

COMANDOS DE MANIPULAÇÃO DE ARQUIVOS:

assign() - Associa um Identificador de arquivo (nome do arquivo a ser utilizado no programa) com o nome do arquivo a ser utilizado no disco. A partir dessa atribuição o arquivo em disco deixará de ser referido pelo original do disco. Todas as operações de tratamento de arquivo farão referência ao identificador.

Ex: assign(ARQPROG, 'ARQDISCO.DAT');

rewrite() - Este comando se comporta de duas formas:
Quando um arquivo que já existe for aberto com rewrite, seu conteúdo será apagado e o "pointer" será posicionado no início do arquivo. Quando o arquivo não existir, o comando rewrite criará um novo arquivo com o nome especificado na instrução assign, deixando-o aberto.

Ex: rewrite(ARQPROG);

reset() - Abre um arquivo já existente e posiciona o "pointer" no início do arquivo. Se o arquivo não existir, ocorrerá um erro de I/O. O erro poderá ser solucionado se for desativada á diretiva I do compilador, com a instrução (SI-) e, em seguida, testar o erro com a função ioresult para definir a estratégia na programação.

Ex: reset(ARQPROG);

close() - Esvazia o buffer e fecha o arquivo. Faz com que todos os dados contidos no buffier temporário sejam gravados no disco (descarrega o buffer). Além disso, também fecha o arquivo.

Ex: close(ARQPROG);

flush()- Esvazia um buffer de saída antes dele ser preenchido (grava os dados em disco).

Ex: flush(ARQPROG);

seek() - Procura determinado registro num arquivo através do número do registro.

Ex: seek(ARQPROG,NumReg);

read - Lê o registro apontado pelo SEEK, trazendo-o para o buffer, e posiciona o "pointer" no próximo registro.

Ex: read(ARQPROG,iIUFFER);

write() - Grava o buffer no disco, no endereço apontado pelo seek e, em seguida, posiciona o "pointer" no próximo registro.

Ex: write(ARQPROG,BUFFER);

ioresult - Informa um código de erro quando são realizadas operações de Entrada/Saída. Se ioresult não for igual a zero, significa que houve algum erro (geralmente é usado para testar a existência ou inexistência de um arquivo na hora de sua abertura).

filesize() - Informa a quantidade de registros do arquivo.

Ex: seek(ARQPROG,filesize(ARQPROG)); --> Esta instrução posiciona o "pointer" no final do arquivo ARQPROG para executar inclusão de novos registros. Como o primeiro registro de um arquivo em Pascal possui número zero, a instrução acima posiciona o "pointer" no próximo registro livre para inclusão (após o último ocupado).

filepos()- Informa a posição atual do "pointer" no arquivo. Indica qual é o número do registro corrente.

Ex: filepos(ARQPROG)

Diretiva I de Ativação do Compilador:

As diretivas de ativação ativam ou desativam recursos especiais do Turbo Pascal, como por exemplo, a diretiva I, que verifica de erros de Entrada/Saída.

Existem outras diretivas de ativação além da I, que são conhecidas por este nome por que só possuem duas condições (ativada ou desativada).

O formato de escrita para ativar (+) ou desativar (-) uma diretiva, é um S seguido da letra da diretiva entre delimitadores de comentários.

Ex: {$I+} e {$I-} ou {*$I+*} e {*$I-*}.

Um arquivo deve ser declarado da seguinte forma:

```
var
    [Var. Arquivo] : file of [Var, de Registro];
```

ObS: No turbo Pascal pode-se abrir no máximo 15 arquivos ao mesmo tempo.

Vejamos um exemplo prático:

```
program Prog36;

type
   ARMA =record
         TIPO          :  string[10];
         MARCA         :  string[6];
         SERIE         :  string[7];
         CALIBRE       :  real;
         ACABA         :  string[9];
         CAPACID       :  integer;
         NCANOS        :  byte;
   end;

var
    Bufferarrna : ARMA;
    ARQARMA : file of ARMA;

begin

    assign(ARQARMA,'CADASTRO.DAT');
    rewrite(ARQARMA);
    seek(ARQARMA,0);
    Bufferarrna.TIPO := 'REVOLVER';
    Bufferarrna.MARCA := 'TAURUS';
    Bufferarrna.SERIE := 'FJ53941';
    Buffcrarrna.CALIBRE := .38;
```

```
    Bufferarrna.ACABA := 'INOX';
    Bufferarrna.CAPACID := 6;
    Bufferarrna.NCANOS := 1;

    write(ARQARMA,Bufferarrna);
    close(ARQARMA);

end.
```

Após a definição de Campos (Tipo, Marca, ...), Registro (... record), Buffer (Bufferarrna....) e Arquivo (---file of...), utiliza-se o comando assign para associar o nome do arquivo usado intemamente no programa (ARQARMA) ao nome do arquivo verdadeiro em disco (CADASTRO.DAT).

Observações:

- O comando rewrite(ARQARMA) criou e manteve aberto o arquivo ARQARMA.

- O commando seek colocou o registro 0 sob a mira do "pointer" de indicação de registro.

- Em seguida, houve as atribuições diretas aos campos do buffer (Bufferarrna), que trabalham como variáveis. Os valores também poderiam ter sido atribuídos via teclado, capturados pelos comandos read ou readln.

- O comando write gravou o registro 0 do arquivo ARQARMA com o conteúdo de Bufferarma.

Observação Importante:

O Turbo Pascal permite a gravação a partir do registro 0. Um registro NÃO pode ser acessado no caso da inexistência do anterior, ou seja, o registro 3 somente pode ser acessado se existir o registro 2. A única excessão é para o registro 0.

REFERÊNCIA AOS CAMPOS - COMANDO WITH.

As referências feitas aos campos de um registro tornam-se extensas, pois são compostas de [nome do buffer].[nome do campo]. Utilizando o comando with, podemos abreviar e escrevei" somente os nomes dos campos. O trecho do programa em que esta abreviação é permitida, ileve estar assim limitado:

```
with [nome do buffer] do

begin .
.
```

```
.
.
.
end;
```

Aproveitamos o exemplo seguinte para utilizar o comando with e também, para o processo de leitura através do apontamento do registro desejado com o comando seek e leitura com comando read.

```
Program Prog38;
type
    ARMA = record
            TIPO          :  string[10];
            MARCA         :  string[6];
            SERIE         :  string[7];
            CALIBRE       :  real;
            ACABA         :  string[9];
            CAPACID       :  integer;
            NCANOS        :  byte;
     end;

var
    Bufferarma : ARMA;
    ARQARMA : file of ARMA;

begin

    assign(ARQARMA,'CADASTRO.ARQ');
    reset(ARQARMA);
    seek(ARQARMA,0);
    read(ARQARMA,BUFFERARMA);

    with BUFFERARMA do
    begin
        writeln(TIPO);
        writeln(MARCA);
        writeln(SERIE);
        writeln(CALIBRE);
        writeln(ACABA);
        writeln(CAPACID);
        writeln(NCANOS);
    end;

    close(ARQARMA);

end.
```

Observações:

- Já que este programa lê os dados gravados pelo programa anterior (Prog37), utilizou-se o comando reset para abrir o arquivo, por que o arquivo já existia anteriormente.

- Apontou-se para o registro 0 (gravado anteriormente no Prog37) através do comando seek. '

- Foi lido para o buffer o registro 0 através do comrnando read.

- Devido a utilização do with Bufferarma do, foi possível a impressão com a abreviatura das Variáveis de Campo (usou-se apenas TIPO, em vez de Bufferarma.TIPO).