5. PROCEDIMENTOS E FUNÇÕES

Até .....

5.1 Procedimentos Simples

Procedimentos são rotinas (trechos ou módulos) de programas, capazes de executar uma tarefa definida pelo programador. Os programas desenvolvidos com procedimentos são ditos 'modulares'. Os programas desenvolvidos com procedimentos são mais legíveis e melhor estruturados.

Todo procedimento deverá ter um nome e um corpo (conjunto de instruções) e deverá ser escrito no campo de declaração de variáveis, imediatamente abaixo destas. Sua sintaxe pode ser vista abaixo:

```
Procedure <nome_do_procedimento> ;
<declaração_de_variáveis_locais> ;
Begin
  <comandos> ;
End;
```

Todo procedimento possui um espaço para declaração de variáveis, chamadas de variáveis locais, embora não seja obrigatório o seu uso. As variáveis declaradas no programa principal são chamadas de variáveis globais.

Dentro do procedimento podem ser utilizadas tanto variáveis locais quanto variáveis globais. Todas as variáveis locais aos procedimentos são alocadas somente quando o procedimento entra em execução, mas são liberadas quando o procedimento termina, perdendo assim, seus conteúdos. Caso seja necessário o aproveitamento dos dados manipulados, o procedimento deverá utilizar as variáveis globais.

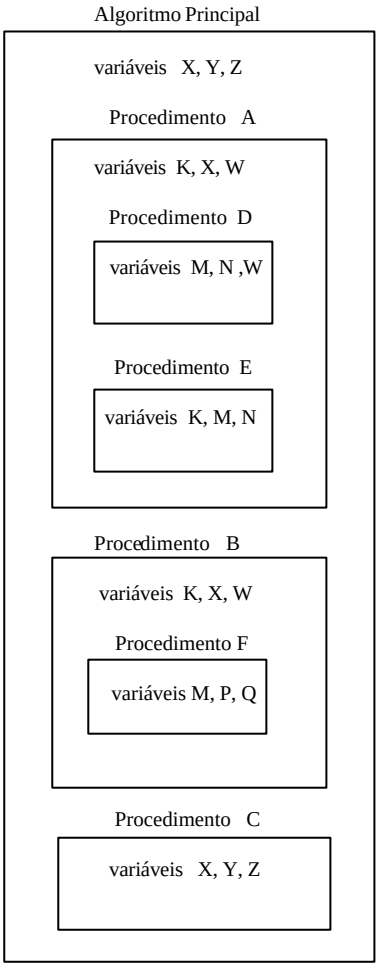
Para executar o procedimento (comandos), o mesmo deverá ser acionado pelo nome. Para exemplificarmos, vamos imaginar um procedimento chamado **Cálculo**, que serve para calcular uma operação matemática qualquer sobre as variáveis **X** e **Y**. Então, poderíamos ter o seguinte trecho do programa:

```
  :
Read( X );
Read( Y );
Cálculo ;
Write(Resultado);
  :
```

No exemplo acima, após a leitura da variável **Y**, deverá ser executado o corpo do procedimento **Cálculo**. Após a execução deste procedimento será executado o comando escreve o restante do programa.

Escopo da Variável

No campo de declaração de variáveis locais ao procedimento, poderão também ser definidos outros procedimentos internos. Assim, poderemos ter um programa com vários procedimentos, e dentro destes outros procedimentos, e assim por diante, conforme mostra o exemplo da figura abaixo:



Em função do exemplo acima, podemos definir basicamente 3 tipos de relação entre procedimentos: global, interno e adjacente, da seguinte forma:

1) Um procedimento P1 é dito global a um procedimento P2, se P2 está dentro de P1. Neste caso, P2 é interno a P1. No exemplo acima temos:

- A é global a D e E ; - B é global a F

2) Um procedimento P1 é dito interno a um procedimento P2, se P1 está dentro de P2. Neste caso P2 é global a P1.No exemplo acima temos:

- D e E são internos a A ; - F é interno a B

3) Dois procedimento P1 e P2 são adjacentes, se P1 não é interno e nem global a P2, ou vice-versa.

- A, B e C são adjacentes.
- D, E, F e C são adjacentes

4) O programa principal é global a todos os procedimentos, e consequentemente, todos os procedimentos são internos ao programa principal.

IMPORTANTE!!! Cada procedimento poderá utilizar suas próprias variáveis ou as variáveis mais globais a ele, mas nunca uma variável de um outro procedimento adjacente. O nível mais global em que uma variável possa ser utilizada é chamado **'escopo da Variável'**.

Exemplo: Faça um programa que calcule as raízes de uma equação do 2º grau, usando procedimentos.

```
Program Calcula-Raizes ;
Var     a, b, c, delta, x1, x2 : Real ;

            Procedure Leia_Coeficientes ;
            Begin
              Repeat
                 Read(a,b,c);
              Until (a <> 0) ;
            End;

            Procedure Calcula_Delta ;
            Begin
              delta := (b*b - 4*a*c) ;
            End;

            Procedure Calcula_Raízes ;
            Begin
              x1 := (-b+SQRT(delta))/(2*a) ;
              x2 := (-b-SQRT(delta))/(2*a) ;
            End;

Begin
  Leia_Coeficientes ;
  Calcula_Delta ;
  If delta < 0
     Then Write('Não existem raizes reais')
     Else Begin
             Calcula_Raizes ;
             Write(x1, x2);
          End;
End.
```

Funções Simples

Funções são rotinas similares aos procedimentos, só que retornam um valor após cada chamada. Uma função não deverá simplesmente ser chamada, como no caso dos procedimentos, mas deverá ser atribuída à alguma Var. Uma função deve ter um nome e um tipo, e sua sintaxe é mostrada abaixo.

```
Function <nome_da_função> : <tipo> ;
<declaração_de_variáveis_locais> ;
Begin
  <lista-de-comandos> ;
End;
```

O nome da função é quem deve assumir o valor da função, como se fosse uma variável. Assim como nos procedimentos, uma função deve ser definida dentro do espaço de declaração de variáveis do programa. Para que a função possa retornar um valor, este deverá ser explicitamente atribuído ao nome da função, dentro da rotina da função.

Exemplo 1: Faça um programa que leia continuamente vários vetores, de tamanhos variados, e calcule para cada um a soma de seus elementos, utilizando função. A leitura dos vetores deverá ser finalizada quando a soma do último for zero (0).

```
Program Soma_Vetores;
Var   n: Integer;
      v: Array[1..10] of Integer ;
      s : Real;

        Procedure Leitura_do_Vetor;
        Var     i : Integer ;
        Begin
          For i := 1  To n Do
              Read( v[i]);
        End;


        Function Soma : Real;
        Var  I: Integer ;  S: Real;
        Begin
            S :=  0;
            For i :=  1 To N Do
               S :=     S + v[i];
            Soma :=  S;    (* aqui é atribuído o valor de retorno *)
        End;

Begin
  Repeat
    Read(n);
    Leitura_do_Vetor;
```

```
    s := Soma ;
   Write('Soma = ', s);
  Until s = 0;
End.
```

Trabalho Individual: Faça um programa em Pascal para o gerenciamento de uma biblioteca utilizando um vetor. O programa deverá permitir as seguintes operações: Inserção, Eliminação, Alteração, Consulta e Listagem de livros. A solução básica está a seguir.

```
Program Gerencia_Biblioteca ;
Const n = 3000 ;    (* no máximo 3000 exemplares *)
Type   tipo_ficha = Record
                       código: Integer  ;
                       título: String [20] ;
                       autor: String [30] ;
                       área: Integer ; (* 1: Humanas ; 2: Exatas ; 3: Biológicas *)
                    End ;

      tamanho, opção, código : Integer ;
      tipo_fichario: Array[1..n] of tipo_ficha ;

Var  fichario : tipo_fichario ;

      Procedure Inicializa_Variáveis;
      Var  i : Integer;
      Begin
        tamanho := 0 ;
        For i := 1 To n Do
          fichario [i].código := -1 ; (* indica que o Record está disponível *)
      End;




      Procedure Apresenta_Menu;
      Begin
        Write('1 - Inserir Novo Cadastro');
        Write('2 - Eliminar Cadastro Velho');
        Write('3 - Atualizar Cadastro');
        Write('4 - Consultar Livros Cadastrados');
        Write('5 - Listar Acervo');
        Write('6 - Sair');
       Repeat
          Read( opção );
        Until (opção > 0) and (opção < 7) ;
```

```
End ;

Procedure  Insere;
Var

        Function Cheia: Boolean ;
        Begin
          Cheia := (tamanho = n) ;
        End ;

        Function Posição_Livre: Integer;
        Var    i : Integer  ;
        Begin
          i :=    1 ;
          While (fichario[i].código <> -1)
                   i :=  i + 1 ;
          Posição_Livre :=     i ;
        End ;
Begin
  If Not Cheia
     Then  Begin
                Inc(tamanho);
                pos := Posição_Livre ;
                With fichario[pos] Do
                   Begin
                      Read(código, título , autor,area);
                   End;
                Write('Cadastro Inserido');
           End
     Else  Write('Fichario Lotado') ;
End;
```

```
Function Busca: Integer  ;
Var   achou, acabou: Boolean ;
        i : Integer  ;
Begin
  achou := False ;
  acabou := False ;
  i  := 0 ;
  While (Not achou) And (Not acabou) Do
     Begin
        i := i + 1 ;
```

```
            achou :=  (fichario[i].código = código) ;
            acabou :=  (i = n) ;
        End ;
    If achou
        Then Busca := i
        Else Busca :=  -1 ;
End ;

Procedure  Elimina;
Var  pos : Integer  ;
Begin
    Read(código);
    pos := busca ;
    If (pos = -1)
        Then Write('Cadastro Inexistente')
        Else Begin
                fichario[pos].código := -1 ;
                tamanho := tamanho -1 ;
                Write('Cadastro Eliminado');
             End;
End;

Procedure  Altera;
Var  pos : Integer  ;
Begin
    Read(código);
     pos := busca ;
    If (pos = -1)
        Then Write('Cadastro Inexistente')
        Else   Begin
                  With fichario[pos] Do
                       Read(código, título , autor, area);
                  Write('Cadastro Alterado');
               End;
End;



Procedure  Consulta;
Var  pos : Integer  ;
Begin
    Read(código);
     pos := busca ;
    If (pos = -1)
        Then Write('Cadastro Inexistente')
        Else   With fichario[pos] Do
                    Write(código, título , autor, área);
```

```
End;

Procedure Lista;
Var t, pos: Integer ;
Begin
  t :=  0; pos :=  1;
  While (pos <= tamanho) Do
    Begin
      If (fichario[pos].código <> -1)
        Then Begin
               With fichario[pos] Do
               Write(código, título, autor, área) ;
                t :=  t + 1;
             end ;
       pos  :=  pos + 1 ;
    End ;

Begin
  Inicializa_Variáveis ;
 Repeat
    Apresenta_Menu ;
    Case opção of
       1: Insere ;
       2: Elimina ;
       3: Altera ;
       4: Consulta ;
       5: Lista ;
    end;
  until opção = 6 ;
end.
```

Procedures e Funções com Passagem de Parâmetros

Até agora, os procedimentos e as funções eram executadas sem que nenhum dado fosse passado específicamente para eles, a não ser as variáveis globais. Muitas vezes precisamos executar um procedimento e/ou função sobre conjuntos de dados diferentes.

Imaginemos uma função que calcula a raiz quadrada de um número (**SQR**). Precisamos em cada momento que ela nos retorne a raiz quadrada para um valor diferente. Assim, passamos para ela um valor, chamado parâmetro, como por exemplo:

SQRT(9) ou **SQRT**(x)

Para chamarmos uma função ou procedimento deste tipo, devemos colocar o nome seguido de um conjunto de parâmetros entre parênteses. Os parâmetros devem ser em igual quantidade e de mesmos tipos, conforme foi definida a função ou o procedimento.

Existem 2 (dois) tipos de passagem de parâmetros: por referência ou por valor:

Passagem por Referência: Os parâmetros devem ser declarados dentro de parênteses, conforme exemplos da declaração abaixo:

Procedure Leitura (a,b,c **: Real) ;** ou

Function Soma (x,y **: Real**): **Real** ;

Neste caso, todas as modificações realizadas nos parâmetros dentro do corpo da função ou do procedimento serão repassados para as variáveis passadas durante a chamada. É como se o procedimento ou função estivessem usando as próprias variáveis que foram passadas como parâmetros.

Para chamarmos o procedimento e a função acima, devemos escrever algo do tipo:

Leitura (x1, x2, x3) **;** ou
s := Soma (f, g) ;

Devemos salientar que as variáveis declarados no procedimento ou na função são chamadas 'parâmetros declarados', e que as variáveis passadas durante a chamada são chamadas 'parâmetros de chamada'. Não existe a necessidade dos nomes dos parâmetros declarados serem iguais aos parâmetros de chamada.

IMPORTANTE!!! Tudo o que for feito com os parâmetros declarados será feito com os parâmetros de chamada, quando se usar passagem por referência. As variáveis declaradas se comportam como se fossem as próprias variáveis de chamada.

Ex: Seja os seguintes procedimento e função:

```
Procedure Leitura (a,b,c : Real) ;
Begin
  Read( a,b,c);
end;

Function Soma ( x, y : Real): Real ;
Begin
  Soma :=    x + y ;
end;
```

poderíamos ter o seguinte trecho de programa:

```
  :
Leitura (m,n,p) ;
s1 :=    Soma (m,n) ;
s2 :=    Soma (n,p) ;
s3 :=    Soma (m,p) ;
Write(s1, s2, s3 );
```

:

Passagem por Valor: Os parâmetros devem ser declarados fora do parênteses, e dentro do espaço de declaração de variáveis locais. A chamada dos procedimentos e das funções é feita da mesma forma que da passagem por referência.

Neste caso, as alterações sofridas pelas variáveis declaradas não serão repassadas para as variáveis de chamada. Este tipo de passagem de parâmetros deve ser utilizado quando só interessa o valor do dado para o procedimento ou para a função.

Utilizando o mesmo exemplo anterior, teríamos a seguinte declaração:

Procedure Leitura (a,b,c) **;**
Var a,b,c : **Real** ; ou

Function Soma (x,y): **Real** ;
Var x,y **: Real ;**

Analisando este exemplos, poderemos facilmente concluir que a função 'Soma' realmente não necessitava de parâmetros por referência, pois eles não eram de fatos alterados. Mas podemos também perceber que o procedimento 'Leitura' deve necessariamente utilizar parâmetros por referência, pois senão os dados lidos serão perdidos após o término do procedimento. Assim, concluímos que as formas corretas para declarar estes procedimentos seriam:

Procedure Leitura (a,b,c **: Real) ;** e

Function Soma (x,y): **Real** ;
Var x,y **: Real ;**

IMPORTANTE!!! Podemos utilizar nos procedimentos e nas funções, uma mistura de parâmetros por rererência e por valor, de acordo com as necessidades do programa.

Exemplo: Faça um programa que leia 3 vetores numéricos A, B e C, ordenados e de tamanhos N1, N2 e N3 respectivamente, e junte-os em um único vetor resultante R ordenado de tamanho N1+N2+N3.

Program Concatena_Vetores**;**
Const n = 100 ; m = 300 ;
Type tipo_vetor: **Array**[1..n] **of Integer ;**
Var A, B, C: tipo_vetor **;**
tam_A, tam_B, tam_C, tam_R **: Integer ;**
R: **Array**[1..m] **of Integer**,

Procedure Leia_Vetor(V: tipo_vetor ; tam : **Integer** **);**

```
Var  i : Integer ;
Begin
  Read( tam );
  For i :=  1  To tam Do
    Read( V[i] );
End;

Procedure  Atribui (V: tipo_vetor; i,j: Integer );
Begin
   R[j] := V[i] ;
   i := i + 1 ;  j := j + 1 ;
End;

Procedure  Descarrega_Segundo(A,B:tipo_vetor ; i,j,k: Integer );
Begin
  If (A[i] < B[j])
      Then Atribui(A,i,k)
      Else Atribui(B,j,k);
End;

Procedure Descarrega_Final(A: tipo_vetor ; i,j : Integer );
Begin
  Repeat
     Atribui(A,i,j) ;
   until (j > tam_R) ;
End;
```

```
Procedure Concatena(A, B, C:tipo_vetor;
                              tam_A, tam_B, tam_C: Integer );
Var      iA, iB, iC, iR : Integer ;
Begin
   iA := 1 ; iB := 1 ; iC := 1 ; iR := 1 ;
   tam_R := tam_A + tam_B + tam_C ;
  Repeat
    If (A[iA] < B[iB]) and (A[iA] < C[iC])
        Then Atribui(A,iA,iR)
        Else If (B[iB] < A[iA]) and (B[iB] < C[iC])
                  Then Atribui(B,iB,iR)
```

```
                    Else Atribui(C,iC,iR);
            Until (iA > tam_A) or (iB > tam_B) or (iC > tam_C) ;
            If (iA > tam_A)
                Then Descarrega_Segundo(B,C,iB,iC,iR)
                Else If (iB > tam_B)
                        Then  Descarrega_Segundo(A,C,iA,iC,iR)
                        Else  Descarrega_Segundo(A,B,iA,iB,iR);
            If (iA ≤ tam_A)
               Then Descarrega_Final(A,iA,iR)
               Else If (iB ≤ tam_B)
                        Then Descarrega_Final(B,iB,iR)
                        Else Descarrega_Final(C,iC,iR);
        End ;
Begin
  Leia_Vetor (A, tam_A) ;
  Leia_Vetor (B, tam_B) ;
  Leia_Vetor (C, tam_C) ;
  Concatena(A, B, C, tam_A, tam_B, tam_C) ;
end.
```

2° TRABALHO INDIVIDUAL

1) Faça um mapa de memória para a declaração abaixo, supondo que a memória disponível inicie na posição 1000. Indique neste mapa as posições inicial e final para cada Var.

```
Var   A: Real ;
      B: Array[1..10] of Lógico ;
      N : String [5] ;
      Ficha: Record
                placa: String [7] ;
                código: Integer  ;
             End ;
```

2) Faça um procedimento que tenha 5 parâmetros, onde os 3 primeiros são por valor e os 2 últimos são por referência. Este procedimento deverá calcular a soma e o produto dos 3 primeiros parâmetros, retornando os resultados no quarto e quinto parâmetros respectivamente.

3) Supondo que você tenha uma tabela (vetor de registros), definida da forma abaixo:

```
Const  n = 1000;
Type   tipo_reg = Record
                     nome: String [15] ;
                     salário: Real ;
                  End ;
```

Var tabela: **Array**[1..n] **of** tipo_reg**;**

Onde a tabela deve ser organizada da seguinte maneira:

a) A inserção de um novo registro deve ser sempre após o último elemento válido.
b) Para controlar qual registro é o último registro válido, deve ser inserido após o último registro um registro *flag*, contendo um salário negativo (‘marca de final de tabela’)
c) Para se eliminar um registro, deve-se colocar o último registro válido do lugar do elemento eliminado e atualizar o registro flag para o registro que foi transferido.

Implemente:

1) Uma função de inserção, que tenha como parâmetro o elemento a ser inserido. A função deverá retornar **True** ou **False** conforme o sucesso ou não da operação.
2) Uma função de remoção, que tenha como parâmetro o elemento a ser removido. A função deverá retornar **True** ou **False** conforme o sucesso ou não da operação.

Obs: Implemente as funções Cheia e Vazia e as utilize.

RECURSIVIDADE

A recursividade ocorre quando dentro de um procedimento (ou função) existe uma chamada (direta ou indireta) para o mesmo procedimento (ou função), tal como os exemplos a seguir:

Exemplo Genérico:

```
procedure Calcula(x: integer);
var y: real;
begin
   : :
   if x> 0
     then Calcula(y);
   : :
end;
```

Exemplo Prático: Escreve os números compreendidos entre 1 até N recursivamente. Neste exemplo, o procedimento “escreva” é recursivo e serve para escrever de “início” até “fim”.

```
program Escrever;
var N : integer;
    : :
    procedure Escreva(início, fim :integer);
    begin
       write(início);
       if início < fim
         then Escreva(início+1, fim);
```

```
    end;
   : :
begin
     : :
  read(N);
  Escreva(1,N);
    : :
end.
```

Exemplo Prático: Somar os elementos de um vetor A recursivamente. Neste exemplo, o procedimento "soma" é recursivo e serve para somar os elementos contidos entre as posições "início" e "fim" do vetor A.

```
program SomarElementosVetor;
var ...
     : :
    procedimento SomaVetor(início, fim :integer);
    begin
      if início = fim
         then SomaVetor := A[início]
         else SomaVetor:= A[início] + SomaVetor(início+1, fim);
   end;
     : :
begin
     : :
  read(N);
  write('Soma do Vetor = ', SomaVetor(1,N));
    : :
end.
```

A recursividade pode ser aplicada para resolver problemas onde a solução no estado atual depende da solução do estado anterior. As diversas chamadas recursivas ocorrem em instâncias diferentes, como se houvessem vários procedimentos (ou funções) iguais sendo chamados, mas cada instância conservam separadamente suas próprias variáveis locais, ou seja, as variáveis utilizadas em uma instância são diferentes daquelas utilizadas nas outras instâncias.

Usando a função anterior de somar vetores, com algumas adaptações, podemos implementar uma função para somar linhas de uma matriz. Neste caso, a função "SomaLinha" somará os elementos da linha "linha" compreendidos entre "início" e "N". A

função “SomaMatriz” utilizará a função “SomaLinha” para somar as linhas compreendidas entre “L1” e “M”. Supoe-se que a Matriz B tenha MxN elementos.

```
procedure  SomaLinha(linha, início :integer);
begin
    if  início = N   /* último elemento da linha */
       then SomaLinha := B[linha, início]
       else SomaLinha:= B[linha, início] + SomaLinha(linha, início+1);
end;

function SomaMatriz(L1: integer): real;
begin
  if L1 = M   /* última linha da matriz */
     then SomaMatriz := SomaLinha(L1,1)
     else SomaMatriz := SomaLinha(L1,1) + SomaMatriz(L1+1)
end;

function SomaMatriz(L1, L2: integer): real;
var soma: real;
begin
   : :
   soma := SomaLinha(L1,1);
   if L1 < L2
     then soma := soma + SomaMatriz(L1+1, L2);
   : :
   SomaMatriz := soma;
end;

function SomaLinha(linha, coluna: integer): real;
var soma: real;
begin
    : :
    soma := M[linha, coluna];
    if coluna < N
      then soma := soma + SomaLinha(linha, coluna+1)
    : :
    SomaLinha := soma;
end;
```

Fatorial

Como exemplo clássico temos o cálculo da fatorial. Sabemos que N! depende de (N-1)!, e este por sua vez depende de (N-2)!, e assim por diante até que N seja 1, quando então

teremos que fatorial de 1 é igual a 1 mesmo. Devemos observar nos exemplos que existe uma condição de parada para a recursividade, pois um procedimento ou função não pode ficar chamando a si próprio indefinidamente. A solução da fatorial pode ser obtida a seguir:

```
function fatorial (N : integer) : real;
begin
   If N <= 1
      then fatorial := 1
      else  fatorial := N * fatorial (N-1);
end;
```

Torre de Hanoi

Um outro exemplo clássico é o da Torre de Hanoi. A Torre de Hanoi compõe-se de uma pilha de elementos ordenados, onde somente é permitido somente empilhar um elemento menor sobre um elemento maior. Assim sendo, o maior elemento deverá estar sempre na base e o menor sempre no topo. Veja o exemplo a seguir. Ele contém uma torre de Hanoi na base A.

Bem, a idéia é mover esta pilha para uma outra base B usando uma terceira base de apoio C. Os elementos devem ser empilhados e desempilhados um a um de forma que os elementos menores sejam colocados sobre os maiores.

A idéia geral da solução é que se existe uma torre de N elementos, então para mudar o elemento da base antes será necessário mudar os (N-1) elementos que estão acima da base. Mas estes (N-1) elementos formam na verdade uma torre de Hanoi menor e para você mudá-la será necessário mudar os (N-2 elementos que estão acima da sua base e o raciocínio se repete recursivamente. Podemos implementar a solução usando 3 vetores O, D e A, representando a torre de origem (O), a torre auxiliar (A) e a torre destino (D). Podemos utilizar números inteiros para representar os elementos empilhados. A solução pode ser vista na forma simplificada a seguir.

```
procedure  Hanoi (var O, D, A: tipo_torre; N: integer);
var elem : integer;
begin
   if N  <> 0
     then begin
            Hanoi(O,A,D,N-1);
            RetiraElemento(O, Elem);
            ColocaElem(Elem, D);
```

```
            Hanoi(A,D,O,N-1);
          End;
end;
```

Onde:

```
const Max = 100;
type tipo_torre = record
                    torre: array[1..Max] of integer;
                    quant: integer;
                  end;

procedure RetiraElemento(var T: tipo_torre; var X: integer);
begin
   X := T.torre[T.quant];
   dec(T.quant);
end;

procedure InsereElemento(X: integer; var T: tipo_torre);
begin
  inc(T.quant);
  T.torre[T.quant] := X;
end;

procedure InicializaTorres(N:integer);
var i : integer;
begin
  for i:= 1 to N do
      O.torre[i] := N-i+1;
  O.quant := N;
  D.quant := 0;
  A.quant := 0;
end;
```

Programa Principal

```
Begin
 Read(N);
 InicializaTorres(N);
 Hanoi(O,D,A,N);
End.
```

Seria interessante acrescentar no programa anterior trechos para a apresentação e visualização da torre, que será transferida do vetor O para o vetor D usando o A como auxiliar.