Aula 02 - Conteúdo

1) Recursividade
2) Diagramas de Execução
3) Exercícios

Recursividade

Uma função é dita ser recursiva quando é definida em termos de si mesmo, isto é, uma função recursiva faz a uma chamada a si mesma.

Fatorial de um número

```
0! = 1
1! = 1
2! = 2 . 1! = 2 . 1  = 2
3! = 3 . 2! = 3 . 2  = 6
4! = 4 . 3! = 4 . 6  = 24
5! = 5 . 4! = 5 . 24 = 120
n! = n . (n-1)!
```

Exemplo 2.1 – Fatorial (arquivos ProjEx201.dpr e Ex201.pas)

```
Function Fat(n:integer): integer;
begin
  if ( n = 0 ) then Fat := 1
  else Fat := n*Fat(n-1);
end;
```

Seqüência de Fibonacci

Essa seqüência tem o nome do matemático italiano que observou que o modelo de criação de coelhos obedece a seqüência abaixo:

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Esta sequência também pode ser observada nas seguintes situações:

- folhas nos ramos das plantas
- genealogia do zangão
- de pétalas de certas flores (margarida, primavera)

Exemplo 2.2 – Fibonacci (arquivos ProjEx201.dpr e Ex201.pas)

```
Function Fib(n:integer):integer;
begin
  if ( n <= 2 ) then Fib := 1
  else Fib := Fib(n-1) + Fib(n-2);
end;
```

A função acima não é eficiente pois ocorrem duas chamadas para o mesmo número. Por exemplo: Fibonacci (6) chama Fibonacci (4) duas vezes. Algumas vezes uma função recursiva pode ser substituído por uma não recursiva mais eficiente.

Durante o estágio do projeto, não se deve forçar nem impedir a recursividade. Na verdade devemos nos concentrar na resolução do problema de uma maneira clara e natural. A preocupação excessiva com eficiência no estágio do projeto pode prejudicar os esforços na resolução do problema.

Exemplo 2.3 – Fibonacci não recursivo (arquivos ProjEx201.dpr e Ex201.pas)

```
Function FibNR(n: integer):integer;
var i,ant,atual,seg: integer;
begin
  ant := 1;
  atual := 1;
  if ( n > 2 ) then
  begin
    ant := 1;
    atual := 1;
    for i:=3 to n do
    begin
      seg := atual + ant;
      ant := atual;
      atual := seg;
    end
  end;
  FibNR := atual;
end;
```

Torre de Hanoi

Segundo um mito indiano, o centro do mundo está sob a cúpula do templo de Benares. Nele, há uma placa de latão onde estão fixadas três agulhas de diamante. Ao criar o mundo, Brama colocou, em uma dessas agulhas, sessenta e quatro discos de ouro puro de tamanhos diferentes, estando o maior junto a base e o menor no topo, chamada de torre de Brama. Seguindo as imutáveis leis de Brama, os sacerdotes do templo mudam os discos de uma agulha para outra, dia e noite, sem cessar, e cada sacerdote move apenas um disco por vez, sem nunca colocar um disco maior sobre outro menor. Quando os sessenta e quatro discos tiverem sido transferidos de uma agulha para outra, a Torre, o templo e os sacerdotes serão transformados em pó, e o mundo desaparecerá com um trovão.

A Torre de Hanoi é uma simplificação da Torre de Brama feita pelo matemático francês Edouard Lucas, em fins do século XIX, onde o número de discos foi reduzido, mantendo-se, entretanto as regras de movimentação dos discos.

Problema: Mover os discos da haste A para a haste C usando a haste B como auxiliar.

Regras:

- mover um disco de cada vez
- um disco maior não pode sobrepor um disco menor

nº de discos	nº mínimo de movimentos
1	1
2	3
3	7
4	15
5	31
k	2^k -1

Obs: A Torre de Brama implica em 2^{64} -1 = 18.446.744.073.509.551.615 movimentos. Se os sacerdotes realizassem um movemento por segundo, 24 horas por dia, todos os dias do ano, sem cometer erros, levariam cerca de 6 bilhões de séculos para realizar a tarefa (ainda bem ...).

Exemplo 2.4 – Torre de Hanoi (arquivos ProjEx201.dpr e Ex201.pas)

```
Type
NumDiscos = 1 .. 100;
Haste = 'A' .. 'C';

Procedure Hanoi(n:NumDiscos; Orig,Dest,Aux:Haste);
begin
  if ( n = 1 ) then
    writeln('Mover disco 1 da haste ',Orig,' para a haste ',Dest)
  else
  begin
    Hanoi(n-1,Orig,Aux,Dest);
    writeln('Mover disco ',n,' da haste ',Orig,' para a haste
',Dest);
    Hanoi(n-1,Aux,Dest,Orig);
  end;
end;
```

Recursão Indireta

Se a função A chama a função B e B chama A, então, ambos, A e B são recursivos. Obseve que A deve ser definido antes de ser chamado em B e B deve ser definido antes de ser chamado em A.

Exemplo 2.5 – Recursão Indireta (arquivos ProjEx201.dpr e Ex201.pas)

```
#include <iostream.h>
Procedure funcB(b: integer); forward;

procedure funcA(a: integer);
begin
  if ( a < 20 ) then
  begin
    a := a + 1;
    funcB(a);
  end;
  ShowMessage('a = '+IntToStr(a));
end;

Procedure funcB(b: integer);
begin
  if ( b < 20 ) then
  begin
    b := b + 1;
    funcA(b);
  end;
  ShowMessage('b = '+IntToStr(b));
end;
```

Programa completo contendo a implementação em Pascal/Delphi dos algoritmos recursivos acima mostrados (arquivos ProjEx201.dpr e Ex201.pas)

```
unit Ex201;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;
type
  TFormEx201 = class(TForm)
    LabelFatorial: TLabel;
    EditFatorial: TEdit;
    ButtonFatorial: TButton;
```

```
    LabelFib: TLabel;
    EditFib: TEdit;
    ButtonFib: TButton;
    LabelFibNR: TLabel;
    EditFibNR: TEdit;
    ButtonFibNR: TButton;
    LabelObsFibNR: TLabel;
    LabelHanoi: TLabel;
    EditHanoi: TEdit;
    ButtonHanoi: TButton;
    LabelRecIndireta: TLabel;
    ButtonRecIndireta: TButton;
    procedure ButtonFatorialClick(Sender: TObject);
    procedure ButtonFibClick(Sender: TObject);
    procedure ButtonFibNRClick(Sender: TObject);
    procedure ButtonHanoiClick(Sender: TObject);
    procedure ButtonRecIndiretaClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  NumDiscos = 1 .. 100;
  Haste = 'A' .. 'C';

var
  FormEx201: TFormEx201;

implementation
{$R *.DFM}

Function Fat(n:integer): integer;
begin
  if ( n = 0 ) then Fat := 1
  else Fat := n*Fat(n-1);
end;

Function Fib(n:integer):integer;
begin
  if ( n <= 2 ) then Fib := 1
  else Fib := Fib(n-1) + Fib(n-2);
end;

Function FibNR(n: integer):integer;
var i,ant,atual,seg: integer;
begin
  ant := 1;
  atual := 1;
  if ( n > 2 ) then
  begin
    ant := 1;
    atual := 1;
    for i:=3 to n do
    begin
      seg := atual + ant;
      ant := atual;
      atual := seg;
    end;
  end;
  FibNR := atual;
```

```
end;

Procedure Hanoi(n:NumDiscos; Orig,Dest,Aux:Haste);
begin
  if ( n = 1 ) then
    ShowMessage('Mover disco 1 da haste '+Orig+' para a haste '+Dest)
  else
  begin
    Hanoi(n-1,Orig,Aux,Dest);
    ShowMessage('Mover disco '+IntToStr(n)+' da haste '+Orig+' para a
haste '+Dest);
    Hanoi(n-1,Aux,Dest,Orig);
  end;
end;

Procedure funcB(b: integer); forward;

procedure funcA(a: integer);
begin
  if ( a < 20 ) then
  begin
    a := a + 1;
    funcB(a);
  end;
  ShowMessage('a = '+IntToStr(a));
end;

Procedure funcB(b: integer);
begin
  if ( b < 20 ) then
  begin
    b := b + 1;
    funcA(b);
  end;
  ShowMessage('b = '+IntToStr(b));
end;

procedure TFormEx201.ButtonFatorialClick(Sender: TObject);
var res, x:integer;
begin
  x := StrToInt(EditFatorial.text);
  res := Fat(x);
  ShowMessage('Fatorial de '+IntToStr(x)+' = '+IntToStr(res));
end;

procedure TFormEx201.ButtonFibClick(Sender: TObject);
var res, x:integer;
begin
  x := StrToInt(EditFib.text);
  res := Fib(x);
  ShowMessage('Fibonacci de '+IntToStr(x)+' = '+IntToStr(res));
end;

procedure TFormEx201.ButtonFibNRClick(Sender: TObject);
var res, x:integer;
begin
  x := StrToInt(EditFibNR.text);
  res := FibNR(x);
  ShowMessage('Fibonacci de '+IntToStr(x)+' = '+IntToStr(res));
end;
```

```
procedure TFormEx201.ButtonHanoiClick(Sender: TObject);
var x:integer;
begin
  x := StrToInt(EditHanoi.text);
  Hanoi(x,'A','B','C');
end;

procedure TFormEx201.ButtonRecIndiretaClick(Sender: TObject);
begin
  funcA(10);
end;
end.
```

Diagrama de Execução

Representar (simular) a execução de um programa e mostrar o escopo de variação das variáveis (intervalo de validade de uma variável).

Exemplo 2.6 – Escopo de variáveis (arquivos ProjEx202.dpr e Ex202.pas)

```
Var x,y,z:integer;
Procedure P1;
var x: integer;
 begin
  x := 2;                    // x é de P1
  x := z; z := y; y := x;    // y e z são globais
end;
Procedure P2(x: integer);
var y: integer;
begin
  y := 0;                      //x e y são de P2
  while ( x <= z ) do y:=y+1;  //z é global
end;
Procedure Principal26;
begin
  readln(x); readln(y); readln(z); // x,y,z sao globais
  if (x >= y) then P1
  else P2(x);
end;
```

Exemplos de Diagramas de Execução

Exemplo 2.7 – Escopo de variáveis e passagem de parâmetros (arquivos ProjEx202.dpr e Ex202.pas)

```
program Ex27;
var a,b,c : integer;
Procedure P(var a,b,c:integer);
  Procedure Q(var a,b:integer; c:integer);
    Procedure R(var a:integer; b,c:integer);
    begin
      a:=a+8; b:=b+8; c:=c+8;
      writeln('Proc R: ',a,' ',b,' ',c);
    end;
  begin
    a:=a+5; b:=b+5; c:=c+5;
    R(a,b,c);
    writeln('Proc Q: ',a,' ',b,' ',c);
  end;
begin
  a:=a+3; b:=b+3; c:=c+3;
  Q(a,b,c);
  writeln('Proc P: ',a,' ',b,' ',c);
```

```
end;
Procedure Principal27;
begin
  a:=1; b:=1; c:=1;
  P(a,b,c);
  writeln('Prog G: ',a,' ',b,' ',c);
end;
```

Exemplo 2.8: Diagrama de Execução para a função Fatorial de n = 4

Exemplo 2.9: Diagrama de Execução para a Torre de Hanoy com 3 discos

Exemplo 2.10: Diagrama de execução da função recursiva abaixo (arquivos ProjEx202.dpr e Ex202.pas)

```
Program B;
var x,y,z: integer;
Function G(var a,b:integer):integer;
var z:integer;
begin
  z:=a-1;
  if ( b = 0 ) then G:= 5*a
  else G:=4*G(b,z);
  a:=4*b;
  b:=5*z;
end;
Procedure Principal210;
begin
  x:=2; y:=4;
  z := G(x,y);
  writeln(z,'  ',x,'  ',y);
end;
```

Programa completo contendo a implementação em Pascal/Delphi dos exemplos 2.6, 2.7 e 2.10 (arquivos ProjEx202.dpr e Ex202.pas)

```
unit Ex202;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TFormEx202 = class(TForm)
    ButtonEx26: TButton;
    ButtonEX27: TButton;
    Button210: TButton;
    procedure ButtonEx26Click(Sender: TObject);
    procedure ButtonEX27Click(Sender: TObject);
    procedure Button210Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormEx202: TFormEx202;
  x,y,z:integer;
  a,b,c:integer;
```

```
implementation
{$R *.DFM}

//Exemplo 2.6
Procedure P1;
var x: integer;
 begin
  x := 2;                     // x é de P1
  x := z; z := y; y := x;    // y e z são globais
  ShowMessage('Proc P1: '+IntToStr(x)+' '+IntToStr(y)+'
'+IntTOStr(z));
end;
Procedure P2(x: integer);
var y: integer;
begin
  y := 0;                      //x e y são de P2
  while ( x <= z ) do y:=y+1; //z é global
  ShowMessage('Proc P2: '+IntToStr(x)+' '+IntToStr(y));
end;
Procedure Principal26;
begin
  x:=10; y:=5; z:=3; // x,y,z sao globais
  if (x >= y) then P1
  else P2(x);
end;

//Exemplo 2.7
Procedure P(var a,b,c:integer);
  Procedure Q(var a,b:integer; c:integer);
    Procedure R(var a:integer; b,c:integer);
    begin
      a:=a+8; b:=b+8; c:=c+8;
      ShowMessage('Proc R: '+IntToStr(a)+' '+IntToStr(b)+'
'+IntTOStr(c));
    end;
  begin
    a:=a+5; b:=b+5; c:=c+5;
    R(a,b,c);
    ShowMessage('Proc Q: '+IntToStr(a)+' '+IntToStr(b)+'
'+IntTOStr(c));
  end;
begin
  a:=a+3; b:=b+3; c:=c+3;
  Q(a,b,c);
  ShowMessage('Proc P: '+IntToStr(a)+' '+IntToStr(b)+'
'+IntTOStr(c));
end;
Procedure Principal27;
begin
  a:=1; b:=1; c:=1;
  P(a,b,c);
  ShowMessage('Principal 27: '+IntToStr(a)+' '+IntToStr(b)+'
'+IntTOStr(c));
end;

//Exemplo 2.10
Function G(var a,b:integer):integer;
var z:integer;
begin
  z:=a-1;
  if ( b = 0 ) then G:= 5*a
```

```
  else G:=4*G(b,z);
  a:=4*b;
  b:=5*z;
end;
Procedure Principal210;
begin
  x:=2; y:=4;
  z := G(x,y);
  ShowMessage('G(x,y)= '+IntToStr(z)+' x='+IntToStr(x)+'
y='+IntTOStr(y));
end;

procedure TFormEx202.ButtonEx26Click(Sender: TObject);
begin
  Principal26;
end;
procedure TFormEx202.ButtonEX27Click(Sender: TObject);
begin
  Principal27;
end;
procedure TFormEx202.Button210Click(Sender: TObject);
begin
  Principal210;
end;
end.
```

Exercícios

2.01) Elabore uma função **recursiva** em linguagem Pascal para realizar a operação de exponenciação a^b, onde a e b são inteiros

Function Eleva(x,y:integer): integer;

Exemplos:

Eleva(2,5) retorna 32 ($2^5 = 32$)

Eleva(3,4) retorna 81 ($3^4 = 81$)

2.02) Desenvolva uma função recursiva para efetuar a somatória de 1 até n (1 + 2 + 3 + ... + n - 1 + n)

Function Somat(n:integer):integer;

Exemplos:

Somat(8) retorna 36 (1+2+3+4+5+6+7+8 = 36)

Somat(10) retornna 55 (1+2+3+4+5+6+7+8+9+10 = 55)

2.03) Elabore uma função recursiva para encontrar os números do Triângulo de Pascal

Triângulo de Pascal

	col 1	col 2	col 3	col 4	col 5	col 6	col 7
lin1	1						
lin 2	1	1					
lin 3	1	2	1				
lin 4	1	3	3	1			
lin 5	1	4	6	4	1		
Lin 6	1	5	10	10	5	1	
:	:	:	:	:	:	:	...

Function Pascal(lin,col:integer):integer ;
Exemplos:
Pascal(4,2) retorna 3 (linha 4 e coluna 2)
Pascal(6,4) retorna 10 (linha 6 e coluna 4)
Pascal(9,1) retorna 1 (linha 9 e coluna 1)

2.04) Desenvolva um algoritmo recursivo em linguagem Pascal para a Função de Ackerman:

definição da função de Ackerman:
a(m.n) = 1 se m = 0
a(m,n) = a(m-1,1) se m <> 0 e n = 0
a(m,n) = a(m-1,a(m,n-1)) se m <> 0 e n <> 0
Function Ackerman(m,n:integer): integer;

2.05) Suponha a existência de um vetor de inteiros
const MAX 10;
Vet: array [1..MAX] of integer;

a) escreva uma função recursiva para somar o n primeiros elementos de Vet
b) escreva uma função recursiva para encontrar o maior elemento de Vet
c) escreva um procedimento recursivo para inverter Vet. Repare que as variáveis inic e fim são as posições inicial e final do vetor.

2.06) Faça o Diagrama de Execução para o programa abaixo

```
var a,b,c:integer;
Procedure R(var a:integer; b,c:integer);
begin
  a := a + 8; b := b + 8; c := c + 8;
  writeln('Funcao R: a=',a,' b=',b,'c=',c);
end;
Procedure Q(var a,b: integer; c:integer);
begin
  a := a+5; b :=b+5; c := c+5;
  R(a,b,c);
  writeln('Funcao Q: a=',a,' b=',b,'c=',c);
end;
Procedure P(var a,b,c : integer);
begin
  a := a + 3; b := b+3; c :=c+3;
  Q(a,b,c);
  writeln('Funcao P: a=',a,' b=',b,'c=',c);
end;
Procedure main;
begin
  a := 1; b := 1; c := 1;
  writeln('Funcao main, antes de P(a,b,c): a=',a,' b=',b,' c=',c);
  P(a,b,c);
  writeln('Funcao main, depois de P(a,b,c): a=',a,' b=',b,' c=',c);

end;
```

2.07) Faça o Diagrama de Execução para o programa abaixo:

```
var x,y:integer;
Function G(var a,b:integer):integer;
var z:integer;
```

```
begin
  z:=a-1;
  if ( b = 0 ) then G:= 3*a
  else G:=2*G(b,z);
  a:=5*b;
  b:=3*z;
end;
Procedure main;
begin
  x:=2; y:=4;
  z := G(x,y);
  writeln(z,x,y);
end;
```

2.08) Faça o diagrama de execução do algoritmo torre de hanoi dado em aula para 3 discos.

2.09) Faça o diagrama de execucação do algoritmo da sequência de Fibinacci para n = 4.

Respostas

2.01) função exponencial a^b

```
Function Eleva(x,y:integer): integer;
begin
  if ( y = 0 ) then Eleva := 1
  else Eleva := x * Eleva(x,y-1);
end;
```

2.02) Somatória de 1 a n (1 + 2 + 3 + ... + n - 1 + n)

```
Function Somat(n:integer):integer;
begin
  if ( n = 1 ) then Somat := 1
  else Somat := n + Somat(n-1);
end;
```

2.03) Triângulo de Pascal

```
Function Pascal(lin,col:integer):integer;
begin
  if ( col = 1 ) or (lin = col ) then Pascal := 1
  else Pascal := Pascal(lin-1,col-1)+Pascal(lin-1,col);
end;
```

2.04) Função de Ackerman:

```
   a(m.n) = 1                    se m = 0
   a(m,n) = a(m-1,1)             se m <> 0 e n = 0
   a(m,n) = a(m-1,a(m,n-1))      se m <> 0 e n <> 0   *)
Function Ackerman(m,n:integer): integer;
begin
  gotoxy(10,15); write(c); c:=c+1;
  if ( m = 0 ) then Ackerman := n+1
  else
  begin
    if ( n = 0 ) then Ackerman := Ackerman(m-1,1)
    else Ackerman := Ackerman(m-1, Ackerman(m,n-1) );
  end
end;
```

2.05) Funções recursivas com vetor

a) Soma dos n primeiros elementos de um vetor

```
Function Soma_Vetor(A:vetor; n:integer): integer;
begin
  if ( n = 1 ) then Soma_Vetor := A[1]
  else Soma_Vetor := A[n] + Soma_Vetor(A,n-1);
end;
```

b) Maior e menor elemento de um vetor – VetMaior e VetMenor

```
Function Vet_Maior(A:vetor;n:integer):integer;
var aux:integer;
begin
  if ( n=1 ) then Vet_Maior:=A[1]
  else
  begin
    aux := Vet_Maior(A,n-1);
    if ( aux > A[n] ) then vet_Maior := aux
    else Vet_Maior := A[n];
  end;
end;

Function Vet_Menor(A:vetor;n:integer):integer;
var aux:integer;
begin
  if ( n=1 ) then Vet_Menor:=A[1]
  else
  begin
    aux := vet_Menor(A,n-1);
    if ( aux < A[n] ) then Vet_Menor := aux
    else Vet_Menor := A[n];
  end;
end;
```

c) Inverter os elementos de um vetor - VetInv

```
Procedure VetInv(var A:vetor;inic,fim:integer);
var aux : integer;
begin
  if ( inic < fim ) then
  begin
    aux := A[inic];
    A[inic] := A[fim];
    A[fim] := aux;
    VetInv(A,inic+1,fim-1);
  end;
end;
```

2.06, 2.07, 2.08 e 2.09

Caro aluno, estes exercícios são por sua conta. Divirta-se.