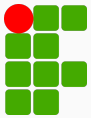
Pascal

03 - Modularização

Marcos Roberto Ribeiro



Instituto Federal Minas Gerais - Campus Bambuí

2018

Introdução I

- A modularização consiste em dividir um programa em partes menores, chamadas *sub-rotinas*, para facilitar o desenvolvimento e a manutenção do mesmo;
- As sub-rotinas representam pequenos problemas do programa principal e portanto possuem uma complexidade menor;
- As sub-rotinas podem ser trabalhadas de forma independente e a localização de um erro no programa é efetuada de forma mais precisa;
- As sub-rotinas são utilizadas para desempenhar sub-tarefas específicas como leitura ou saída de dados, cálculos matemáticos, dentre outras.
- Além de evitar a repetição de sequências de comandos idênticas, podemos destacar as seguintes vantagens da modularização:
  - Melhor legibilidade do código;
  - Um programa pode ser desenvolvido em equipe (cada programador cuida de uma parte);
  - As sub-rotinas podem ser testadas isoladamente na verificação de erros;
  - A manutenção do programa é facilitada, apenas algumas partes podem precisar de manutenção.

Introdução II

- Na linguagem *Pascal* podemos considerar que existem dois níveis de modularidade[1]:
  - Modularidade a nível de unidade (*Units*);
  - Modularidade a nível de procedimento (procedimentos e funções);
- Um programa deve ser desenvolvido de forma modular sempre que possível. Isto significa que se um mesmo código é executado em pontos diferentes do programa, então devemos criar uma função ou procedimento com este código e chamá-lo quando necessário;
- Porém só a criação de procedimentos e funções não é suficiente. Se existe uma *Unit* com muitos procedimentos e funções que realizam vários tipos de tarefas diferentes. Então devemos dividir esta *Unit* em várias *Units* com funções e procedimentos que desempenham atividades semelhantes;
- A diferença entre uma função e um procedimento é que a função retorna um valor como resultado e o procedimento não;

[1] Na verdade ainda existe uma modularização por parte da orientação a objeto, mas este tópico não será tratado nesta disciplina

Procedimentos

- Procedimentos são trechos de código que executam determinada tarefa ao serem chamados e depois retornam o controle para o ponto em que foram chamados;
- Apesar de ainda não termos criados procedimentos, já escrevemos diversos programas que utilizam procedimentos prontos do sistema, por exemplo, os procedimentos **ReadLn()**, **Write()** e **WriteLn()**;
- Tais procedimentos estão disponíveis para qualquer programa que façamos, pois fazem parte da **unit system** que é adicionada automaticamente para todo programa em Pascal;
- Agora vamos ver como utilizar procedimentos de outras *units* do sistema. Para isto devemos adicionar a *unit* a ser utilizada na declaração **uses**;
- Quando chamamos um procedimento ou função devemos informar os parâmetros em número, ordem e tipos corretos. Isto significa que se vamos usar um procedimento X que recebe um parâmetro inteiro e outro parâmetro String, nesta ordem, então devemos usar o código **X(a,b)**, onde **a** é **Integer** e **b** é **String**;

Exemplo

Como exemplo vamos utilizar os procedimentos **TextColor()**, **TextBackground()** e **ClrScr()** da **unit crt**;

```
program ExemploChamadaProcedimento;

{Unit que contém os procedimentos}
uses crt;

begin
   {Procedimento que define a cor do texto}
   TextColor(BLUE);
   {Procedimento que define a cor de fundo}
   TextBackground(BLACK);
   {Procedimento para limpar a tela}
   ClrScr();
   WriteLn('Olá!');
   ReadLn();
end.
```

Criando Procedimentos

- Os procedimentos devem ser criados antes do bloco principal **begin ... end.**;
- A estrutura básica de um procedimento é a seguinte:

```
{Cabeçalho do procedimento}
procedure NomeProcedimento;
begin {Início do corpo do procedimento}
    ...
end;  {Fim do corpo do procedimento}
```

- Neste ponto já podemos criar nossos próprios procedimentos, como exemplo vamos modificar o programa anterior e criar um procedimento para inicializar a tela.

Exemplo

```
program ExemploProcedimento;

uses crt;

{Declaração do procedimento que inicializa a tela}
procedure PreparaTela();
begin
   {Este procedimento chama outros procedimentos}
   TextColor(BLUE);
   TextBackground(BLACK);
   ClrScr();
end;

{Bloco principal do programa}
BEGIN
   PreparaTela(); {Chamada ao procedimento criado}
   WriteLn('Olá!');
END.
```

Procedimentos

- Em nosso exemplo criamos apenas um procedimento, mas um programa pode ter vários procedimentos e um procedimento pode chamar outros procedimentos.
- Para que um procedimento A possa ser chamado por um procedimento B, o procedimento A deve ser declarado antes de B;

Exemplo com Vários Procedimentos I

```
program ExemploProcedimento2;

uses crt;

{Procedimento que escreve o cabeçalho}
procedure Cabecalho();
begin
   WriteLn(' Programa com múltiplos procedimentos');
   WriteLn('**********************************');
end;
```

Exemplo com Vários Procedimentos II

```
{Procedimento de inicialização da tela}
procedure PreparaTela();
begin
   TextColor(BLUE);
   TextBackground(WHITE);
   ClrScr();
   {Chama procedimento de cabeçalho}
   Cabecalho();
end;

{Procedimento que volta a tela ao normal}
procedure VoltaTelaNormal();
begin
   TextColor(WHITE);
   TextBackground(BLACK);
   ClrScr();
end;
```

Exemplo com Vários Procedimentos III

```
begin
   {Chama o procedimento PreparaTela}
   PreparaTela();
   WriteLn('Olá!');
   ReadLn();
   {Chama o procedimento VoltaTelaNormal}
   VoltaTelaNormal();
end.
```

Utilização de Variáveis em Procedimentos

- Um procedimento pode declarar variáveis para serem utilizadas apenas dentro do mesmo;
- Estas variáveis são chamadas de variáveis *locais* enquanto as variáveis do programa principal são chamadas de variáveis *globais*[2].

[2] As variáveis globais podem ser acessadas em qualquer parte do programa, mas devem ser evitados acessos a tais variáveis dentro de procedimentos.

Exemplo de Variáveis em Procedimentos I

```
...
{Variável global}
var N: Integer;

{Procedimento que anima a tela colorindo-a}
procedure AnimaTela();
{Variável local, só o procedimento atual acessa}
var Cor: Integer;
begin
   TextColor(BLUE);
   for Cor := 15 downto 0 do begin
      TextBackground(cor);
      ClrScr();
      {Procedimento que paraliza por n milissegundos}
      Delay(300);
   end;
end;
```

Exemplo de Variáveis em Procedimentos II

```
begin
   for N:=1 to 5 do begin
     WriteLn('Animação ', N);
     AnimaTela();
   end;
   WriteLn('Acabou');
   ReadLn();
end.
```

Passagem de Parâmetros

- Os procedimentos criados até agora desempenham sempre a mesma ação, pois estes procedimentos não recebem *parâmetros externos*;
- Um procedimento pode receber parâmetros para executar ações de acordo com os valores recebidos;
- Os parâmetros são valores passados na chamada do procedimento, desta maneira podemos considerá-los como sendo variáveis inicializadas.

Exemplo

Exemplo de procedimento com parâmetros

```
{Este procedimento recebe um parâmetro "cor" do tipo Byte e
↪  limpa a tela colorindo-a com a cor recebida}
procedure LimpaTelaColorido(Cor: Byte);
begin
   TextBackground(Cor);   {Parâmetro usado como uma variável}
   ClrScr();
end;
```

- Como exercício refaça o programa anterior usando o procedimento **LimpaTelaColorido**.

Funções

- Uma função é um tipo de sub-rotina que retorna um valor como resultado;
- A declaração de uma função possui basicamente a seguinte estrutura:

```
function NomeDaFuncao(Parametros): TipoRetornado;
begin
  ...
  NomeDaFuncao:=Resultado;
end;
```

- Como uma função deve retornar um resultado é necessário atribuir este resultado ao nome da função, normalmente este é o último comando da função;
- Pelo fato das funções retornarem valores, podemos utilizá-las em expressões como se fossem variáveis.

Exemplo com Função

```
program ExemploFuncao;

var A, R: Integer;

{Funão que calcula um número elevado ao cubo}
function Cubo(Numero: Integer): Integer;
begin
  Cubo := Numero * Numero * Numero;
end;

begin
  Write('Digite um número: ');
  ReadLn(A);
  {Podemos usar a função diretamente como parâmetro}
  WriteLn('O cubo de ', A, ' é: ', Cubo(A));
  {Podemos usar a função diretamente em uma expressão}
  R := A + 27 DIV Cubo(3);
  Write('O resultado da expressão é: ', R);
  ReadLn();
end;
```

Chamada a Procedimentos e Funções

- Quando usamos um procedimento A em um procedimento B na mesma *Unit* o procedimento A deve ser escrito antes do procedimento B:

```
procedure A();
begin
  ...
end;

procedure B();
begin
  A();
end;
```

Chamada a Procedimentos e Funções

- Outra possibilidade é declarar o cabeçalho do procedimento A antes de B usando a palavra chave **forward**:

```
procedure A(); forward;

procedure B();
begin
  ...
  A()
end;

procedure A();
begin
  ...
end;
```

Passagem de Parâmetros por Referências I

- Todos os exemplos de procedimento e funções que estudamos até o momento utilizam passagem de parâmetros por valor, isto significa que acontece uma cópia do valor passado e qualquer alteração de valor dentro do procedimento não é efetivada sobre o parâmetro;
- Em certas situações pode ser necessário que um procedimento ou função modifique o valor de um parâmetro e esta modificação seja refletida para o local onde ocorreu a chamada. Neste caso existe uma passagem de parâmetro por referência;
- Um procedimento com passagem de parâmetro por referência que já utilizamos é o **ReadLn()**. Tal procedimento recebe parâmetros, obtém os valores destes parâmetros com o usuário e os valores ficam gravados nas variáveis utilizadas como parâmetros;

Passagem de Parâmetros por Referências

- De forma sucinta podemos dizer que:
  - Na passagem de parâmetros por valor, acontece uma cópia do valor passado. Desta maneira, qualquer alteração de valor dentro do procedimento não é efetivada sobre o parâmetro que foi passado;
  - Na passagem de parâmetros por referência esta cópia não acontece. Neste caso, as modificação sobre o parâmetro são repassadas para o parâmetro externo que foi usado;
- Para ilustrar melhor como funciona a passagem de parâmetros por referência, vamos criar um procedimento que troca os valores de duas variáveis inteiras.

Exemplo com passagem de parâmetros por referência

```
...
var
  N1, N2: Integer;

{Troca valores entre "A" e "B"}
procedure TrocaInteiros(var A, B: Integer);
var
  Aux: Integer;
begin
  Aux := B;
  B := A;
  A := Aux;
end;

begin
  WriteLn('Informe dois números.');
  ReadLn(N1, N2);
  WriteLn('N1 = ', N1, 'N2 = ', N2);
  TrocaInteiros(N1,N2);
  WriteLn('Números trocados.');
  WriteLn('N1 = ', N1, 'N2 = ', N2);
  ReadLn();
end.
```

Unidades de código I

- As unidades de código *Units* são arquivos de código separados com constantes, definições de tipos, variáveis, procedimentos e funções que realizam atividades similares. A estrutura básica de uma *Unit* é a seguinte:

```
unit nome_da_unit;
interface
...
implementation
...
end.
```

- Podemos notar que uma *Unit* possuir dois blocos:
  interface Bloco de código que é compartilhado pela *Unit*;
  implementation Implementação da *Unit* que não é compartilhado;

Unidades de código II

- É importante observar também que o nome da *Unit* deve ser o mesmo nome do arquivo e deve-se utilizar preferencialmente letras minúsculas. Por exemplo, se uma *Unit* chama-se **abc**, então o arquivo deve ser salvo como **abc.pas**;
- A declaração de constantes, tipos e variáveis é realizada normalmente tanto na seção **interface** quanto na seção **implementation**;
- No caso de procedimentos e funções, a seção **interface** deve possuir apenas o cabeçalho enquanto a seção **implementation** deve possuir o cabeçalho e corpo;
- Como exemplo vamos desenvolver um programa em Pascal para verificar se as notas de alunos estão acima ou abaixo da média da turma.

Exemplo com Unidades de Código (Unit alunos) I

```
unit alunos;

{Bloco compartilhado pela Unit}
interface

const
  N_ALUNOS = 10;

type
  TVetNotas = array [1..N_ALUNOS] of Real;

{Função que calcula a média das notas (compartilhada)}
function CalculaMediaNotas(VetNotas: TVetNotas):Real;

{Implementação das sub-rotinas declaradas no interface}
implementation
```

Exemplo com Unidades de Código (Unit alunos) II

```
{Função que calcula a soma das notas (não compartilhada)}
function CalculaSomaNotas(VetNotas: TVetNotas):Real;
var
  Contador: Integer;
  Soma: Real;
begin
  Soma:= 0;
  for Contador := 1 to N_ALUNOS do begin
    Soma := Soma + VetNotas[Contador];
  end;
  CalculaSomaNotas:= Soma;
end;

function CalculaMediaNotas(VetNotas: TVetNotas):Real;
begin
  CalculaMediaNotas:= CalculaSomaNotas(VetNotas) / N_ALUNOS;
end;
end.
```

Exemplo com Unidades de Código (program NotasAbaixoMedia)

```
program NotasAbaixoMedia;
uses alunos;              {Utiliza a Unit "alunos"}
var
  Contador: Integer;
  Notas: TVetNotas;  {"TVetNotas" declarado na Unit "alunos"}
  Media: Real;
begin
  for Contador := 1 to NUM_ALUNOS do begin
    WriteLn('Informe a nota do aluno ', Contador);
    ReadLn(Notas[Contador]);
  end;
  {Função "CalculaMediaNotas" implementada na Unit "alunos"}
  Media := CalculaMediaNotas(Notas);
  WriteLn('A média da turma é ', Media:0:2);
  WriteLn();
  WriteLn('Os alunos com notas abaixo da média são:');
  for Contador := 1 to NUM_ALUNOS do begin
    if Notas[Contador] < Media then begin
      WriteLn(Contador, ' : ', Notas[Contador]:0:2);
    end;
  end;
end.
```

Units que Utilizam Units

- Se a **Unit x** possui um procedimento **x1()** em sua **interface** então uma **Unit y** que *usa* a **Unit x** pode chamar o procedimento **x1()**;
- Porém se a **Unit x** possui um procedimento **x2()** somente no bloco **implementation** uma **Unit y** que usa a **Unit x** não pode chamar **x2()**;
- Para uma **Unit y** usar uma **Unit x** basta acrescentarmos **uses x** na **Unit y**;
- Se **Z** usa **Y** e **Y** usa **X**, então **Z** usa **X**?

  Sim Se **Y** usa **X** dentro de **interface**;

  Não Se **Y** usa **X** dentro de **implementation**;

Unidades de código

```
Unit a;

interface

procedure A1();

implementation

procedure A1();
begin
  ...
end;

procedure A2();
begin
  ...
end;
```

```
Unit b;

uses a;
...
implementation

procedure B1()
begin
  A1(); // Funciona
  A2(); // Erro
end;
```

Principais Rotinas da Unit *crt*

ClrScr() Limpa (preenche com espaços) a tela;

ClrEol() Limpa da posição do cursor até o fim da linha;

Delay(MS:Word) Para a execução do programa em *MS* milissegundos;

DelLine() Remove a linha na posição do cursor;

GotoXY(X,Y) Posiciona o cursor na posição horizontal *X* e vertical *Y*. *X* e *Y* são números de 1 a 255;

InsLine() Insere uma linha na posição do cursor;

ReadKey():char Obtém uma tecla pressionada;

TextBackground(Cor:Byte) Muda a cor de fundo do texto;

TextColor(Cor:Byte) Muda a cor do texto;

WhereX(), WhereY() Retornam as posições horizontal e vertical do cursor;

Window(XIni, YIni, XFim, YFim) Delimita uma janela de (XIni, YIni) a (XFim, YFim) onde o cursor fica preso.

Unit setas.pas I

```
{Programa que usa as setas para movimentar o cursor}
unit setas;

interface

const
  LIMITEY = 20; {Limite para a posição vertical}
  LIMITEX = 60; {Limite para a posição horizontal}
type {Tipos de comandos tratadospelo programa}
  TComando = (DIR, ESQ, CIMA, BAIXO, SAIR, NADA);

{Função que retorna comando de acordo com tecla pressionada}
function PegaComando(): TComando;
{Movimenta o cursor na tela de acordo com um comando}
procedure Movimenta(Comando: TComando);

implementation

uses crt; {Usa a biblioteca de funções crt}
```

Unit setas.pas II

```
function PegaComando(): TComando;
var
  Tecla: char;  {Armazena a tecla pressionada}
begin
  Tecla := ReadKey(); {Pega a tecla pressionada}
  if (Tecla = #0) then begin {Tratamento para teclas especiais}
    Tecla := ReadKey();
  end;
  case Tecla of {Testa qual tecla foi pressionada}
    'e':PegaComando := SAIR;
    #75:PegaComando := ESQ;
    #77:PegaComando := DIR;
    #72:PegaComando := CIMA;
    #80:PegaComando := BAIXO;
    else PegaComando := NADA;
  end;
end;
```

Unit setas.pas III

```
procedure Movimenta(Comando: TComando);
var
  X, Y:Byte; {Variáveis para armazenar a posição do cursor}
begin
  X:=WhereX(); {Obtém a posição horizontal}
  Y:=WhereY(); {Obtém a posição vertical}
  ClrScr();  {Limpa a tela}
  case Comando of {Muda a posição de acordo com o comando}
    ESQ: begin
      if (X > 1) then begin
        X := X - 1;
      end;
    end;
    DIR: begin
      if (X < LIMITEX) then begin
        X := X + 1;
      end;
    end;
```

Unit setas.pas IV

```
    CIMA: begin
      if (Y > 1) then begin
        Y := Y - 1;
      end;
    end;
    BAIXO: begin
      if (Y < LIMITEY) then begin
        Y := Y + 1;
      end;
    end;
  end;
  GotoXY(X, Y); {Muda o cursor para a nova posição}
end;
END.
```

Programa movcursor.pas

```
{Programa que usa as setas para movimentar o cursor}
program movcursor;

uses setas; {Usa a biblioteca de funções setas}

var {Variável para armazenar os comandos}
  Comando: TComando;

{Bloco principal do programa}
BEGIN
  repeat
    Comando := PegaComando(); {Pega um comando}
    Movimenta(Comando); {Movimenta o cursor}
  until (Comando = SAIR); {Repete até que o comando seja SAIR}
END.
```

Referências I

Canneyt, M. V.

Free pascal reference guide.

`http://www.freepascal.org/docs-html/ref/ref.html`.

Evaristo, J. (1999).

***Programando com Pascal*.**

Book Express, Rio de Janeiro, 2 edition.

Farrer, H., Becker, C. G., Faria, E. C., de Matos, H. F., dos Santos, M. A., and Maia, M. L. (2008).

***Programação estruturada de computadores*.**

LTC, Rio de Janeiro, 3 edition.

Referências II


Forbellone, A. V. and Eberspacher, H. F. (2005).
Lógica de programação.
Prentice Hall, São Paulo, 3 edition.


Manzano, J. A. N. G. and de Oliveira, J. F. (2011).
Algoritmos.
Érica, São Paulo, 25 edition.


Manzano, J. A. N. G. and Yamatumi, W. Y. (2001).
Programando em turbo pascal 7.0 e free pascal compiler.
Érica, São Paulo, 9 edition.