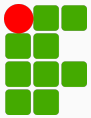
Pascal

01 - Introdução à Linguagem de Programação Pascal

Marcos Roberto Ribeiro



Instituto Federal Minas Gerais - Campus Bambuí

2018

Introdução

- Pascal é uma linguagem de programação estruturada criada em 1970 pelo suíço Niklaus Wirth;
- A linguagem Pascal é usada tanto para fins acadêmicos quanto para programação comercial;
- A utilização comercial da linguagem é realizada principalmente através de ambientes de desenvolvimento como Lazarus;
- No decorrer desta aula estudaremos cada parte da estrutura de programas escritos em pascal.

Estrutura de um Programa em Pascal

- A estrutura de um programa em pascal é mostrada a seguir, contudo programas mais simples podem não necessitar de todas as partes desta estrutura.

```
program NomePrograma;

uses Biblioteca_1, ..., Biblioteca_N;

const
  CONSTANTE_1 = valor_1;
  ...
  CONSTANTE_N = valor_N;

type
  TipoDefinido_1 = ...
  ...
  TipoDefinido_N = ...

var
  Variavel_1, Variavel_2: Tipo_1;
  Variavel_3, Variavel_4: Tipo_N;

begin
  instrucao_1();
  ...
  instrucao_N();
end.
```

Comentários

- A utilização de comentários é de extrema importância para um melhor entendimento do código;
- Em pascal podemos fazer comentários curtos (de apenas uma linha) iniciados com // e comentários longos (com várias linhas) delimitados por {};
- O programa a seguir exibe estes tipos de comentários.

```
program ExemploComentarios;

{ Este programa demonstra a utilização de comentários
  Este é um comentário longo com várias linhas }

begin
  instrucao();  // Este é um comentário curto
end.
```

Tipos de Variáveis

- A utilização de variáveis é fundamental na maioria dos algoritmos. A linguagem pascal possibilita a declaração de variáveis numéricas, lógicas e de caracteres.
- Valores lógicos

Tipo	Intervalo	Tamanho (bytes)
Boolean	*false* ou *true*	1

- Números Inteiros

Tipo	Intervalo	Tamanho (bytes)
Byte	0..255	1
Smallint ou Integer[1]	−32.768..32.767	2
Longint	−2.147.483.648..2.147.483.647	4

[1] Dependendo da diretiva *mode*, o tipo **Integer** pode ser associado a **Smallint**({$mode ObjFPC}) ou **Longint** ({$mode Delphi}). Por padrão vamos tratar o tipo **Integer** como **Smallint**.

Tipos de Variáveis

- Números Reais

Tipo	Intervalo	Tamanho (bytes)	Precisão[2]
Single ou Real[3]	$-3,4^{38}..3,4^{38}$	4	7 a 8
Double	$-1,7^{308}..1,7^{308}$	8	15 a 16
Extended	$-1,1^{4932}..1,1^{4932}$	10	19 a 20

[2] A precisão é o número de dígitos significantes da parte fracionária

[3] O tipo **Real** pode ser associado a **Single** ou **Double**, dependendo do processador. Por padrão vamos tratar o tipo **Real** como **Single**.

Tipos de Variáveis

- Caracteres

Tipo	Intervalo	Tamanho (bytes)
Char. Normalmente é usado para para representar um caractere ASCII[4]	#0..#255	1
String[5]	-	1 byte / caractere
ShortString	0..255 Caracteres	1 byte / caractere
AnsiString	0..*N* Caracteres[6]	1 byte / caractere
WideString[7]	0..*N* Caracteres	2 bytes / caractere

[4] Para maiores informações acesse `http://pt.wikipedia.org/wiki/ASCII`

[5] Dependendo da diretiva *H*, o tipo **String** pode ser associado a **ShortString**({$H-}) ou **AnsiString** ({$H+}). Por padrão vamos tratar o tipo **String** como **ShortString**.

[6] O número de caracteres é limitado pelo tamanho da memória disponível

[7] O tipo **WideString** é utilizando para para representar caracteres *unicode*. Para maiores informações acesse `http://pt.wikipedia.org/wiki/Unicode`

Declaração de Variáveis e Atribuição de Valores

- Como podemos ver na estrutura de um programa em pascal. A declaração de variáveis é realizada após a palavra chave **var**;
- A atribuição de valores a variáveis é realizada com o operador :=;
- Logo adiante, temos o trecho de um programa utilizando diversas variáveis.

```
program ExemploVariaveis;

{Declaração de Variáveis}
var
  Contador: Integer; // Variável contadora inteira
  N1, N2: Real; // Variáveis fracionárias
  Letra: Char; // Variável caractere para uma letra
  Nome: String; // Variável caractere para várias letras
begin
  Contador := 1;
  N1 := 2.5;
  Letra := 'M';
  Nome := 'Marcos';
  Letra := #45; // Atribui a letra M usando seu código ASCII
end.
```

Entrada e Saída de Dados

- Basicamente a entrada e saída de dados é feita com as instruções **readln()** e **write()**, respectivamente;
- A instrução **readln()** permite que o usuário informe valores para variáveis (digitando o valor e pressionando ENTER para cada variável). Para utilizar esta instrução devemos passar como parâmetros as variáveis separadas por vírgulas cujos valores serão informados pelo usuário;
- O próximo programa demonstra a utilização da instrução **readln()**;

```
program ExemploRead;
var
  N1, N2: Integer;
  Nome: String;
begin
  readln(Nome, N1, N2);
end.
```

- Caso o usuário informe um valor incompatível para uma variável, como uma letra para número, o programa gerará uma mensagem de erro e será finalizado.

A Instrução write()

- A exibição de mensagens por um programa é extremamente recomendada, pois facilita muito a vida do usuário fornecendo informações acerca do que deve ser feito;
- A instrução **write()** serve para exibir mensagens para ao usuário, podendo receber como parâmetros variáveis, expressões e valores literais;
- O próximo programa demonstra a utilização da instrução **write()**;

```
program ExemploWrite;
var
  Idade: Integer;
  Nome: String;
begin
  write('Informe seu nome.');
  readln(Nome);
  write('Informe sua idade.');
  readln(Idade);
  write('Olá, ', Nome, '. Sua idade é ', Idade);
end.
```

- Nas duas primeiras instruções **write()** informamos apenas um **String** literal. Na última mensagem, mesclamos literais com variáveis para exibir uma mensagem personalizada.

Formatando Valores Numéricos com a Instrução write()

- A instrução **write()**[8] permite formatar parâmetros numéricos através dos modificadores **:N:D**. Estes modificadores são muito úteis na exibição de valores decimais (que por padrão são formatados em notação científica) e para alinhar valores numéricas a direita;
- O modificador **:N** exibe a parte inteira do número em **N** caracteres, se o número não possuir **N** dígitos inteiros, são acrescentados espaços a esquerda para completar os **N** caracteres;
- O modificador **:D** faz com que seja exibidas apenas **D** casas decimais do número;

```
program ExemploWriteFormatacao;
var
  NR1, NR2: Real;
begin
  NR1 := 3.3314;          NR2 := 2345.76923;
  writeln('Exibição padrão de números:');
  writeln(NR1, NR2);
  writeln('Exibição de números formatados:');
  writeln(NR1:10:2, NR2:10:2);
end.
```

[8] A Instrução **writeln()** é semelhante a instrução **write()**, porém é inserida uma nova linha após a mensagem.

Constantes

- Uma constante é uma posição de memória que armazena um valor fixo. Tal valor não poderá ser alterado durante sua execução do programa. Um exemplo de utilização é para delimitar tamanhos de vetores;
- O próximo programa exemplifica a declaração de constantes.

```
program ExemploConstantes;
{Declaração de Constantes}
const
  LARGURA = 50;
  ALTURA = 20;
  AREA = LARGURA * ALTURA;
begin
  writeln('Altura: ', ALTURA);
  writeln('Largura: ', LARGURA);
  writeln('Área: ', AREA);
end.
```

Expressões Aritméticas

- A linguagem pascal permite a criação de expressões aritméticas combinando variáveis, *números literais*[9], constantes, parênteses e os seguintes operadores aritméticos:

  + Adição[10];
  - Subtração;
  * Multiplicação;
  / Divisão de números reais (o resultado de expressões com este operador é um número real);
  div Divisão de números inteiros;
  mod Resto da divisão de números inteiros;

[9] Números literais são números digitados diretamente no código fonte

[10] O sinal de + também pode ser usado para concatenar **Strings**.

Exemplo com Expressões Aritméticas

```
program ExemploExpressoes;
var
  N1, N2: Integer; // Variáveis inteiras
  NR: Real; // Variável real
begin
  N1 := 10;
  N2 := 54;
  NR := 23.54;
  N1 := N1 * N2 div 2; // Expressão correta
  // N1 := N1 * N2 div NR; // Descomente e observe o erro
  // N1 := N1 * N2 / 2; // Descomente e observe o erro
  NR := NR / N2;  // Expressão correta
end.
```

Exemplo com Concatenação

```
program ExemploConcatena;
var
  Silaba1, Silaba2, Palavra: String;
begin
  writeln('Informe duas sílabas');
  readln(Silaba1);
  readln(Silaba2);
  Palavra := Silaba1 + Silaba2;
  writeln('Palavra formada: ', Palavra);
end.
```

Exercícios

- Crie um programa para calcular a área de um triângulo retângulo;
- Faça um programa que receba nome, nome do meio e sobrenome e forme o nome completo da pessoa.

Relações

- Relações são comparações entre valores numéricos utilizando operadores relacionais. Os operadores relacionais disponíveis na linguagem Pascal são:

  `>` Maior;
  `>=` Maior ou igual;
  `<` Menor;
  `<=` Menor ou igual;
  `=` Igual;
  `<>` Diferente

- Exemplo:

```
program ExemploOpRelacionais;
...
begin
  ...
  Logico := (5 > 3); // Lógico recebe true
  Logico := (5 < 3); // Lógico recebe false
  ...
end.
```

Expressões Lógicas

- A Pascal permite a criação de expressões lógicas através da combinação de variáveis lógicas, dos literais **true** (verdadeiro) e **false** (falso), de parênteses e dos seguintes operadores lógicos:

  and Operador binário equivalente ao operador $\wedge$ da lógica relacional. O resultado do operador é **true** de ambos operandos forem **true**;

  or Operador binário equivalente ao operador $\vee$ da lógica relacional. O resultado do operador é **true** de um dos operandos for **true**;

  not Operador unário equivalente ao operador $\neg$ da lógica relacional. O resultado do operador é o inverso do operando;

- Como o resultado de uma relação é um valor lógico, as expressões lógicos podem conter termos que são relações entre parênteses. Por exemplo, a expressão "$(5 > 3)$ **and** $(8 <= 10)$" é uma expressão lógica válida.

Expressões Lógicas

- O programa a seguir possui algumas expressões lógicas válidas.

```
program ExemploExpLogicas;
...
var
  N1, N2: Integer;
  Logico: Boolean;
...
begin
  ...
  Logico := (5 > 3) and (8 <= 10);
  Logico := (N1 <> 50) or (N2 >= 21);
  ...
end.
```

Estruturas de Seleção

- As estruturas de seleção permitem desviar o fluxo de execução do programa de acordo com determinadas situações;
- As estruturas de seleção disponíveis na linguagem Pascal são as instruções **if** e **case**;
- O **if** permite executar instruções de acordo com uma condição booleana;
- O **case** é útil para decidir a execução de instruções a partir do valor de uma expressão ou variável.

if ... then

- A forma mais simples de utilização do **if** é a seguinte[11]:

```
if (Expressão Lógica) then begin
  instrução X;
end;
```

- A **Instrução X** será executada apenas se o resultado da **Expressão Lógica** for **true**;
- Exemplo:

```
writeln('Informe sua idade');
readln(Idade);
if (Idade >= 18) then begin
  writeln('Você é maior de idade');
end;
```

[11] Caso a **Expressão Lógica** seja apenas um valor lógico os parênteses podem ser dispensados, mas sua utilização pode ser conveniente por uma questão de padronização. As instruções **begin ... end** podem ser suprimidas caso exista apenas uma instrução a ser executada, porém vamos utilizar sempre tais instruções para manter um padrão.

Blocos de Instruções

- O programa seguinte demonstra a utilização de da instrução **if...then**;

```
program ExemploIf;
var
  Idade: Integer;
begin
  writeln('Informe a sua idade.');
  readln(Idade);
  if (Idade >= 18) and (Idade <= 65) then begin
    writeln('Seu voto é obrigatório!');
    writeln('Vote consciente!');
  end;
end.
```

- Observe que o bloco **begin ... end;** termina com ponto e vírgula, diferente do bloco **begin ... end.** principal que termina com ponto final.

if ... then ... else

- Com o **if ... then** é possível testar uma condição e executar instruções se esta condição for verdadeira;
- Já o **if ... then ... else** executa algumas instruções se a condição for verdadeira ou outras instruções se a condição for falsa;
- O próximo trecho de código exemplifica a utilização do **if ... then ... else**[12];

```
if (Idade >= 18) and (Idade <= 65) then begin
  writeln('Seu voto é obrigatório!');
  writeln('Vote consciente!');
end else begin
  writeln('Seu voto não é obrigatório.');
end;
```

- Note que utilizamos um bloco **begin ... end** para o **else** mesmo havendo apenas uma instrução. Nos próximos códigos vamos adotar esta prática como padronização.

[12] O **end** antes do **else** não possui ; (ponto-e-vírgula)

ifs Aninhados

- Muitas vezes é preciso utilizar **ifs** aninhados para tratar certas situações. O programa a seguir exibe uma destas situações.

```
program ExemploIfsAninhados;
var
  Idade: Integer;
begin
  writeln('Informe a sua idade.');
  readln(Idade);
  if (Idade >= 16) then begin
    if (Idade >= 18) and (Idade <= 65) then begin
      writeln('Seu voto é obrigatório.');
    end else begin
      writeln('Seu voto é facultativo.');
    end;
  end else begin
    writeln('Você ainda não pode votar.');
  end;
end.
```

case

- Para entender o **case** vamos analisar o seguinte programa:

```
program ExemploCase;
var
  N1, N2: Real;
  Operacao: Integer;
begin
  writeln('Informe dois números reais');
  readln(N1,N2);
  writeln('Informe 1 (Adição) ou 2 (Subtração)');
  readln(Operacao);
  case Operacao of
    1:
    begin
      writeln(N1:0:4, ' + ', N2:0:4, ' = ', (N1 + N2):0:4);
    end;
    2:
    begin
      writeln(N1:0:4, ' - ', N2:0:4, ' = ', (N1 - N2):0:4);
    end else begin
      writeln('Operação inválida!');
      writeln('Tente novamente.');
    end;
  end;
end.
```

- Podemos observar que de acordo com a operação escolhida o **case** executa a instrução correspondente. O **else** é executado se nenhuma das opções do **case** foi tratada.

Exemplo com case

- A seguir temos um programa que exibe o conceito de uma dada pontuação obtida poderia utilizando a instrução case.

```
program ExemploCaseConceito;

var
  Pontos: Integer;
  Conceito: char;

begin
  writeln('Digite a pontuação obtida');
  readln(Pontos);
  case Pontos of
    0 .. 49: begin
      Conceito := 'D';
    end;
    50 .. 69 : begin
      Conceito := 'C';
    end;
    70 .. 89 : begin
      Conceito := 'B';
    end;
    90 .. 100: begin
      Conceito := 'A';
    end;
  end;
  writeln(Pontos, ' pontos correspondem a o conceito ', Conceito);
end.
```

Estruturas de Repetição

- As estruturas de repetição, também chamadas de laços, são muito úteis para executar repetidamente várias instruções;
- A linguagem Pascal possui as seguinte estruturas de repetição:

  for ... to ... do Usado quando o programa tem conhecimento do número de repetições a serem executadas;

  while ... do Usado para repetir instruções uma ou mais vezes enquanto uma condição for verdadeira;

  repeat ... until Usado para repetir instruções zero ou mais vezes até que um condição se torne verdadeira.
- Nas próximas páginas veremos três programas, cada um implementado com uma estrutura de repetição diferente, mas que realizam a mesma tarefa.

for ... to ... do

- O programa a seguir utiliza uma estrutura de repetição **for ... to ... do** para imprimir na tela a sequência de números de 1 a 100.

```
program ExemploFor

var
  Contador: Integer;

begin
  for Contador := 1 to 100 do begin
    writeln(Contador);
  end;
end.
```

- O **for** faz o incremento da variável **Contador** automaticamente a cada iteração.
- A estrutura de repetição **for ... downto ... do** funciona de forma análoga, porém o contador e decrementado.

while ... do

- O programa a seguir possui uma estrutura de repetição **while ... do** para imprimir na tela a sequência de números de 1 a 100.

```
program ExemploWhile;

var
  Contador: Integer;

begin
  Contador:=1;
  while (Contador < 100) do begin
    writeln(Contador);
    Contador := Contador + 1;
  end;
end.
```

- Quando utilizamos um **while** devemos inicializar a variável contadora antes e incrementá-la manualmente dentro do laço.

repeat ... until

- O programa a seguir possui uma estrutura de repetição **repeat ... until** para imprimir na tela a sequência de números de 1 a 100.

```
program ExemploRepeat;

var
  Contador: Integer;

begin
  Contador:=0;
  repeat
    Contador := Contador + 1;
    writeln(Contador);
  until (Contador = 100);
end.
```

- No laço **repeat** também devemos inicializar a variável contadora e fazer seu incremento manualmente.

Estruturas de Repetição Aninhadas

- Em muitos casos, é preciso utilizar estruturas de repetição *ainhadas* ou laços *aninhados*, ou seja, um laço é inserido dentro de outro laço;
- Na maioria dos caso de laços aninhados o número de repetições total será a multiplicação das repetições do laço interno pelas repetições do laço externo;
- Para um exemplo vamos considerar um conjunto de números naturais $A = \{1, 2, 3, ..., n\}$ e um programa que pretenda exibir o produto cartesiano $A \times A$;
- O referido programa deverá combinar cada elemento de A com todos os elementos de A, seu código é exibido a seguir.

```
program CartesianoN;

var
  N, C1, C2 : Integer;

begin
  write('Digite o numero de elementos do conjunto: ');
  readln(N);
  writeln('{');
  for C1 := 1 to N do begin
    for C2 := 1 to N do begin
      writeln('(', C1, ', ', C2, ')');
    end;
  end;
  writeln('}');
end.
```

Estruturas de Repetição Aninhadas

- Em outras situações o laço interno pode depender do laço externo;
- Por exemplo, considerando novamente um conjunto $A = \{1, 2, 3, ..., n\}$, e que fosse preciso obter os subconjuntos de dois elementos contidos em A. Nesta situação não podemos combinar um elemento com ele mesmo, pois os conjuntos não possuem elementos repetidos;
- A seguir temos o código para o programa mencionado.

```
program Subconjuntos2;
var
  N, C1, C2 : Integer;
begin
  write('Digite o numero de elementos do conjunto: ');
  readln(N);
  for C1 := 1 to N do begin
    for C2 := C1 + 1 to N do begin
      writeln('{', C1, ', ', C2, '}');
    end;
  end;
end.
```

Funções Pré-definidas I

Algumas da linguagem Pascal (incluídas na unit **System**):

Abs(n: real ou inteiro) Retorna o valor absoluto de **n**;

Cos(n: real) Retorna o cosseno de **n**;

Exp(n: real) Retorna e^n;

Frac(n: real) Retorna retorna a parte fracionária de **n**;

Int(n: real) Retorna a parte inteira de **n**;

Ln(n: real) Retorna o logaritmo neperiano de **n**;

Pi() Retorna o valor de π.

Random(n: inteiro positivo) Retorna um número aleatório maior ou igual a zero e menor que **n**;

Randomize() Inicializa o gerador de números aleatórios com o relógio do sistema. Devemos executar a função **Randomize()** pelo menos uma vez antes de executar qualquer função **Random()**;

Round(n: real) Arredonda **n** para o valor inteiro mais próximo;

Funções Pré-definidas II

Sin(n: real) Retorna o seno de **n**;
Sqrt(n: real ou inteiro) Retorna a raiz quadrada de **n**;
Sqr(n: real) Retorna n^2;
Trunc(n: real) Obtém a parte inteira de **n**.

Definição de Tipos

- Além dos tipos de dados já existentes, a linguagem Pascal permite a definição de novos tipos com a declaração **type**;
- Após a definição dos tipos podemos declarar variáveis com os novos tipos;
- Podemos observar a definição de novos tipos no próximo programa[13];

```
program NovosTipos;
type
  TDiaSemana = (domingo, segunda, terca, quarta, quinta, sexta, sabado);
var
  Dia: TDiaSemana;
  Nome: String;
begin
  writeln('Informe seu nome e o dia da semana')
  readln(nome);
  readln(Dia);
  writeln('Olá, ', Nome);
  writeln('Tenha um(a) ótima(a) ', dia);
end.
```

- Ainda veremos outras maneiras de definir novos tipos de dados.

[13] O uso das rotinas **readln()** e **write()** diretamente com tipos definidos é suportado apenas em versões mais recentes do Free Pascal.

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.


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.


Wikipédia.
Pascal (linguagem de programação).
https://pt.wikipedia.org/wiki/Pascal.