LINGUAGEM PASCAL

ÍNDICE

I – INTRODUÇÃO

Este curso destina-se àqueles que desejam se iniciar na linguagem Pascal, através do seu mais famoso compilador para a linha IBM/PC, o Turbo Pascal. O Turbo Pascal é muito mais que um compilador, pois ele é uma associação entre um compilador, um editor de textos e um linkeditor. Desta forma, o Turbo Pascal facilita sobremaneira o ato de programar. Além de tudo, o Turbo permite muitas facilidades e atividades que, com certeza, não estavam planejadas por Niklaus Wirth, o criador da linguagem Pascal. Levando-se em conta todas essas considerações, podemos até mesmo dizer que o Turbo Pascal seria uma nova linguagem, mais poderosa que a Pascal.

Gostaria de salientar que a melhor forma de aprender uma linguagem é programando, assim como a melhor forma de aprender a dirigir é entrar num automóvel e sair com ele por aí. O simples fato de ler este pequeno e simples curso de linguagem Pascal não basta para aprender a programar em Pascal.

Esta apostila parece ter sido escrita originalmente pelo professor Thelmo J. M. Mesquita, a julgar pelo nome do autor de alguns programas. A presente versão está um pouco mudada: dois capítulos finais da versão original foram cortados (para abranger apenas a matéria de um semestre), foram acrescentados outros exemplos e exercícios e pequenos trechos da redação foram modificados.

I.1 - A linguagem Pascal

Considero que a programação deve ser entendida como uma arte ou técnica de se construir algoritmos, sendo que estes são métodos ou "receitas" para se resolver problemas. Existem diversas linguagens para se programar, umas mais adequadas a certos tipos de algoritmos, outras a outros tipos. No entanto, uma linguagem de programação não deve ser um fim em si mesma, mas um meio, uma ferramenta para se traduzir os algoritmos em programas a serem executados por computadores. Desta forma, é importante que os cursos de programação não tenham como objetivo primordial a perfeição do conhecimento de uma linguagem específica. A linguagem deve, tão somente, refletir de maneira clara e facilmente compreensível os aspectos principais dos algoritmos.

Por tudo isso, devemos ter a preocupação de ensinar aos estudantes a formulação sistemática e metódica de algoritmos, através de técnicas que são características da programação.

Existem diversas linguagens de programação, podemos aprender e utilizar quantas desejarmos. Dizer qual a melhor é muito relativo. Há os que defendem o Basic, o Cobol, a C, o Pascal e tantas outras. Bom, mas a pergunta crucial que faço aqui é: Qual a primeira linguagem a ser aprendida? Neste ponto, defendo a linguagem Pascal.

De acordo com observações feitas por diversos professores, a maior parte das pessoas ficam ligadas para sempre à primeira linguagem que aprenderam, e, quando aprendem uma nova linguagem, têm uma certa tendência em desenvolver os algoritmos segundo o vocabulário e regras sintáticas da primeira linguagem, só que escritas na nova.

Por este motivo, acho que a escolha da primeira linguagem a ser ensinada deve ser feita de forma judiciosa. A primeira linguagem deve, desta forma, ser tal que forneça ao aprendiz a possibilidade de desenvolver algoritmos lógicos, sistemáticos, facilmente compreensíveis segundo os métodos modernos de programação e deve até possibilitá-lo a "dar asas à sua imaginação".

I.2 - Por que Turbo Pascal?

Um computador não pode entender nem executar instruções em linguagens de alto nível. Ele só entende linguagem de máquina. Desta forma, os programas em linguagens de alto nível devem ser traduzidos antes de serem executados pelo computador. Quem faz essa tradução são os programas tradutores.

Existem basicamente 2 tipos de programa tradutor: o interpretador e o compilador. Os dois aceitam como entrada um programa em linguagem de alto nível (fonte) e produzem como saída um programa em linguagem de máquina (objeto). A diferença entre eles está na forma de executar a tarefa de tradução. O interpretador traduz para a linguagem de máquina e roda uma linha por vez, até que todo programa seja executado. Já o compilador traduz para a linguagem de máquina todo o programa fonte e só então ele é executado.

Existem linguagens de programação interpretadas e compiladas. O Cobol é compilado, o Basic pode ser tanto compilado como interpretado e assim por diante. A linguagem Pascal é tradicionalmente compilada.

O processo de compilação deve seguir as seguintes etapas:

- Edição: utilizar um editor de textos para escrever e gravar nosso programa fonte.
- Compilação: utilizar um compilador para traduzir o programa fonte para um programa em linguagem de máquina.
- LinkEdição: finalmente, devemos juntar ao programa compilado as diversas rotinas necessárias que, normalmente, ficam armazenadas numa biblioteca.

Após todo esse processo, suponha que você chegue à conclusão de que o programa tenha que sofrer modificações. Você terá que repetir os três passos descritos, e assim sucessivamente até que o programa fique ao seu gosto.

O compilador Turbo Pascal facilita todo esse processo, pois integra um editor de textos, um compilador e um linkeditor. A compilação pode ser feita tanto em disco como em memória, o que faz com que seja bem rápida. Além disso, o Turbo Pascal atende aos padrões da linguagem Pascal definidos por Niklaus Wirth, "o pai da linguagem". Na realidade, o Turbo Pascal vai além, pois possui inúmeras procedures e funções a mais do que as existentes no padrão da linguagem Pascal.

I.3 - Equipamento necessário

Os exemplos e programas contidos neste curso foram escritos num PC 486DX 50 com dois drives de disquete, um winchester de 340 megabytes, um monitor monocromático e 640 Kbytes de memória RAM. No entanto, a configuração mínima poderia ser um IBM/PC-XT com um winchester de 40M.

II - UM PROGRAMA EM PASCAL

II.1 - O primeiro programa

Aqueles que nunca tiveram a oportunidade de fazer um programa em Pascal devem estar curiosos para saber como deve ser o seu aspecto. Por isso, antes de prosseguir com os meandros da linguagem Pascal, mostrarei um pequeno programa devidamente comentado.

PROGRAMA EXEMPLO.PAS -> *Pequeno exemplo de um programa em Pascal. Tem a finalidade única e exclusiva de mostrar os diversos componentes de um programa em Pascal.*
{Tudo que estiver entre chaves são comentários e não são levados em conta pelo compilador.}

```
Program Exemplo1; { este e o cabeçalho do programa }

USES Crt;   { Aqui estou utilizando uma UNIT chamada CRT; existem várias, e
              inclusive você pode criar as suas. Nestas units temos procedures e
              functions previamente compiladas. }

            { área de definição dos dados que serão usados no programa }
Const
      Meu_Nome = 'Thelmo'; { nesta área podemos definir todas as constantes que
                             quisermos utilizar no programa }
Var   idade              :integer;
      altura             :real;
      nome               :string[30];
      sexo               :char;
      { todas as variáveis que forem utilizadas no corpo do programa deverão ser
      declaradas na subárea Var }

                         { área das instruções do programa }
Begin
      ClrScr; { apaga a tela }
      For i:=1 to 80 do Write('-'); { escreve uma linha tracejada na tela }
      Writeln('Meu nome e -------> ',Meu_Nome);
      For i:=1 to 80 do Write('-');
      Write('Qual o seu nome ----> ');
      Readln(Nome);
      For i:=1 to 80 do Write('-');
      Write('Qual a sua idade ---> ');
      Readln(idade);
      For i:=1 to 80 do Write('-');
      Writeln('nossas idades somam --> ', 34+idade);
      For i:=1 to 80 do Write('-');
      Write('Prazer em conhece-lo');
End.
```

II.2 - Estrutura de um programa simples em Pascal

Todo programa em Pascal é subdividido em 3 áreas:

- cabeçalho do programa,
- área de declarações,
- corpo do programa.

Na definição padrão da linguagem Pascal, o cabeçalho do programa é obrigatório, no entanto, no Turbo Pascal ele é opcional. A área de declarações é subdividida em seis sub-áreas, a saber:

- Label
- Const
- Type
- Var
- Procedures
- Functions

As subáreas Label, Type, Procedures e Functions serão estudadas mais à frente. Darei agora, uma breve explicação das outras subáreas, pois mais para frente estudaremos cada uma delas com profundidade.

Todas as constantes que formos utilizar no nosso programa, podem, se assim desejarmos, ser definidas na subárea Const.

O Turbo Pascal tem basicamente 6 tipos de variáveis pré-definidas a saber: Integer, Real, Byte, Boolean, Char e String. No entanto, podemos definir novos tipos de variáveis na subárea Type (veremos isto mais adiante). Todas as variáveis utilizadas no programa devem ser declaradas na subárea Var, pois a alocação de espaço de memória para as variáveis é feita durante a compilação.

Estas sub-áreas não são obrigatórias. Por exemplo: se não vamos utilizar variáveis no nosso programa (coisa rara) então não precisamos utilizar a subárea Var. De acordo com a definição padrão da Linguagem Pascal, estas sub-áreas devem aparecer na seqüência que foi dada anteriormente, ou seja, Label - Const - Type - Var - Procedures - Functions. Mas no Turbo Pascal isto é livre.

Por fim, como dito no programa exemplo, existe a possibilidade de se usar a declaração USES, que nos permite utilizar UNITS que nada mais são do que bibliotecas de funções e procedures previamente declaradas.

III - NOÇÕES BÁSICAS PRELIMINARES

III.1 - Elementos básicos do Turbo Pascal

III.1.1 - Caracteres utilizados

Os caracteres que podem ser utilizados no Turbo Pascal são divididos em:

Letras: 'A' até 'Z', 'a' até 'z'
Números: 0,1,2,3,4,5,6,7,8 e 9
Especiais: + - * / = ^ < > ( ) [ ] { } . , : ; ' # $

Dois pontos em seguida (..) indicam um delimitador de faixa, por exemplo:

1..30 --> todos inteiros entre 1 e 30 inclusive.

O Turbo Pascal não faz distinção entre letras maiúsculas e minúsculas. Assim, nos programas utilizaremos os dois tipos, da forma que acharmos mais conveniente.

III.1.2 - Palavras reservadas

As palavras reservadas do Turbo Pascal são palavras que fazem parte da sua estrutura e têm significados pré-determinados. Elas não podem ser redefinidas, nem utilizadas como identificadores de variáveis, labels, procedures, functions etc. Algumas destas palavras reservadas são:

absolute(*)	And	array	Begin
case	Const	div	Do
downto	Else	end	external(*)
file	For	forward	Function
goto	If	in	inline(*)
label	Mod	nil	not
of	Or	packed	procedure
program	Record	repeat	set
shl(*)	shr(*)	string(*)	then
to	Type	until	var
while	With	xor(*)	

(*) --> não definidos no Pascal Standard

III.1.3 - Identificadores pré-definidos

O Turbo Pascal possui inúmeros identificadores pré-definidos, que não fazem parte da definição padrão da linguagem Pascal. Esses identificadores consistem em Procedures e Functions, que podem ser utilizados normalmente na construção de programas. Exemplos:

ClrScr : limpa a tela de vídeo
DelLine : deleta a linha em que está o cursor e assim por diante.

Constantemente, novas procedures e functions são criadas pela Borland (criadora do Turbo Pascal), aumentando desta forma o número de identificadores. São UNITS que tornam o Turbo Pascal mais poderoso do que ele já é.

III.1.4 - Regras para formação de identificadores

O usuário também pode definir seus próprios identificadores, na verdade nós somos obrigados a isso. Nomes de variáveis, de labels, de procedures, functions, constantes etc. são identificadores que devem ser criados pelo programador. Mas para isso existem determinadas regras que devem ser seguidas:

1-) O primeiro caractere do identificador deverá ser obrigatoriamente uma letra ou um underscore (_).
2-) Os demais caracteres podem ser letras, dígitos ou underscores.
3-) Um identificador pode ter no máximo 127 caracteres.
4-) Como já dissemos anteriormente, não pode serusada uma palavra reservada do Pascal.

Exemplos de identificadores válidos:

Meu_Nome
MEU_NOME igual ao anterior
__Linha
Exemplo23

Exemplos de identificadores não válidos:

2teste : começa com número
Exemplo 23 : tem um espaço

III.1.5 - Comentários

Comentários são textos que introduzimos no meio do programa fonte com a intenção de torná-lo mais claro. É uma boa prática em programação inserir comentários no meio dos nossos programas. No Turbo Pascal, tudo que estiver entre os símbolos (* e *) ou { e } será considerado como comentário.

III.1.6 - Números

No Turbo Pascal, podemos trabalhar com números inteiros e reais. Os números inteiros podem ser representados na forma hexadecimal, basta precedê-los do símbolo $. Os números reais também podem ser representados na forma exponencial. Isto varia de versão para versão do turbo Pascal. A seguir estão as faixas de valores válidas para a versão 7.0:

TIPO	FAIXA	FORMATO
Shorting	-128..127	Signed 8-bit
Integer	-32768..32767	Signed 16-bit
Longint	-2147483648.. 2147483647	Signed 32-bit
Byte	0..255	Unsigned 8-bit
Word	0..65535	Unsigned 16-bit

TIPO	FAIXA	DÍGITOS	BYTES
real	2.9e-39..1.7e38	11-12	6
single	1.5e-45..3.4e38	7- 8	4
double	5.0e-324..1.7e308	15-16	8
extended	3.4e-4932..1.1e4932	19-20	10
comp	-9.2e18..9.2e18	19-20	8

III.1.7 - Strings

Strings são conjuntos de caracteres entre aspas simples. Exemplos:

'isto é uma string'
'123456' etc.

III.2 - Definição de variáveis

As variáveis que forem utilizadas no corpo do programa, devem ser declaradas numa subárea específica chamada *Var*. Para estudar essa subárea devemos primeiro ver os tipos de variáveis pré-definidos em Turbo Pascal.

III.2.1 - Tipos de dados pré-definidos

Os tipos de dados pré-definidos em Turbo Pascal são divididos em duas categorias:

- *Escalares Simples:* Char, Boolean, todos os tipos de inteiros citados acima e todos os tipos de reais citados acima,
- *Escalares estruturados:* String, Array, Record, File, Set, e Text.

Inicialmente, iremos estudar os escalares simples e o tipo *String* pela sua utilização prática inicial. Os demais tipos estruturados serão vistos mais para a frente.

char: O tipo char corresponde a todos os caracteres que podem ser gerados pelo teclado: dígitos, letras e símbolos tais como **&**, #,***** etc. Os caracteres devem vir entre aspas simples: 'A', 'Z', '1', '%'.

boolean: O tipo boolean só pode assumir os valores **FALSE** e **TRUE**.

string: Este tipo é chamado de estruturado ou composto pois é constituído a partir de um tipo simples que é o *char*. O tipo *string* é composto por um conjunto de caracteres entre aspas simples: 'Jose', 'meu primeiro programa!'.

shortint - ***integer*** - ***longint*** - ***byte*** - ***word***: Ver tabela acima (III.1.6).

real - ***single*** - ***double*** - ***extended*** - ***comp***: Ver tabela acima (III.1.6).

III.2.2 - A declaração Var

Esta é a subárea onde devemos declarar todas as variáveis que iremos utilizar em nosso programa. Exemplo:

```
Program Exemplo;        (* cabeçalho do programa *)
  Var
     idade,número_de_filhos : byte;
     altura                 : real;
     sexo                   : char;
     nome                   : string[30];
     sim_ou_não             : boolean;
     quantidade             : integer;
   Begin      (* aqui começa o programa *)
        idade:=34;
        número_de_filhos:=2;
        sexo:='M';
        nome:='José';
        sim_ou_nao:=TRUE;
        quantidade:=3245;
   End.
```

Observações importantes:

1-) A palavra reservada ***Var*** aparece uma única vez num programa

2-) A sintaxe geral para declaração de variáveis é:
variável_1,variável_2,...,variável_n : tipo;

3-) Os espaços e comentários separam os elementos da linguagem. Você pode colocar quantos espaços quiser. Observe:
Varidade: integer; o compilador não reconhece a palavra Var
Var idade:integer; agora sim, ou se preferir
Var idade
: *integer*; dá na mesma.

4-) As instruções são separadas entre si por ponto e vírgula '**;**'. Se você quiser, pode colocar mais de uma instrução numa única linha. Lembre-se que o limite de caracteres numa linha é de 127.

5-) O tipo string deve ser procedido da quantidade máxima de caracteres que a variável pode assumir. A alocação de espaço de memória para as variáveis é feita durante a compilação, portanto o compilador precisa saber desse dado. Por outro lado, o fato de termos, por exemplo, atribuído o valor máximo de 30 não significa que tenhamos que utilizar os 30 caracteres e sim no máximo 30.

6-) Como última observação, acho muito mais claro e elegante declarar variáveis e ao mesmo tempo informar com comentários os devidos motivos. Exemplo:

```
Var
   idade,     (* idade de determinada pessoa *)
   i,j        (* utilizadas em loops        *)
        : integer;
   nome1,     (* nome genérico de pessoas    *)
   nome2      (* nome genérico de pessoas    *)
        : string[50];
```

III.2.3 - A declaração type

Além dos tipos de dados pré-definidos no Turbo Pascal, podemos também definir novos tipos através da declaraçäo *Type,* cuja sintaxe é detalhada no item VIII.1.

III.3 - Constantes

III.3.1 - A declaração const

Nesta subárea, podemos definir tantas constantes quantas quisermos. Sintaxe:

```
Const
     Meu_nome = 'Thelmo';
     cor_preferida = 'verde';
     número_máximo = 24345;              (* e assim por diante *)
```

Toda vez que nos referirmos às constantes acima, o Turbo Pascal substituí-las-á pelos seus respectivos valores.

III.3.2 - Constantes pré-definidas

Existem algumas constantes pré-definidas que podemos utilizar sem ter que declará-las, por exemplo:

```
PI = 3.1415926536E + 00
FALSE
TRUE
NIL                Pointer nulo, veremos mais adiante.
MAXINT = 32767
```

III.3.3 - Constantes tipadas ou inicialização de variáveis

A declaração de variáveis na subárea *Var* apenas reserva espaço de memória para elas, mas não as inicializa, ou seja, até que se atribuam valores a elas, seus valores serão desconhecidos.

Quando necessitarmos declarar a variável e dar seu valor inicial, podemos usar uma constante tipada, cuja sintaxe é:

```
Const variável : tipo = valor;
Exemplos:
    const Contador : integer = 100;
          c : char = 'A';
```

*Estamos definindo duas variáveis: contador, que é inteira e vale inicialmente **100**, e **c,** que é do tipo char e cujo valor inicial é 'A'.*

III.4 Operadores

III.4.1 - Operadores aritméticos

+	adição	/	divisão entre números reais
-	subtração	**DIV**	divisão entre números inteiros
*	multiplicação	**MOD**	resto da divisão

III.4.2 – Concatenação de strings

Esta operação é representada pelo sinal de adição, ou seja, +. Os operandos devem ser do tipo string ou char. Exemplos:

```
'Isto é uma ' + 'string' = 'Isto é uma string'
NomeCompleto := Nome + ‘   ‘ + Sobrenome;
```

PROGRAMA EXEMPLO : Mostra como utilizar operadores aritméticos

```
Program Operadores_aritimeticos;
Uses CRT;
Var  x,y,z : integer;
     r1,r2 : real;
Begin
  ClrScr;      (* limpa a tela *)
  x:=10;
  y:=20;
  z:=x+y;
  writeln(z);   (* escreve o valor de z na tela de vídeo *)
  x:= 20 DIV 3;
  y:= 20 MOD 3;
  writeln(x);   (* escreve 6 na tela *)
  writeln(y);   (* escreve 2 na tela *)
  r1:=3.24;
  r2:=r1/2.3;
  writeln(r2);
end.
```

IV - ENTRADA E SAÍDA DE DADOS

IV.1 - Write e Writeln

Estas são as principais procedures destinadas a exibir dados no vídeo. A diferença entre *write* e *writeln* reside no fato de que a procedure write escreve o parâmetro e mantém o cursor do lado daquilo que foi escrito, enquanto que writeln passa o cursor para a próxima linha. Estas procedures possuem 3 formas de sintaxe, a saber:

Primeira forma: Write(parâmetro_1,Parâmetro_2, ...);

```
Program Escrever1;
Uses CRT;
Var     i : integer;
        r : real;
        c : char;
        s : string[20];
Begin
  ClrScr;       (* apaga a tela e coloca o cursor em 1,1 *)
  Writeln('Exemplos de aplicação de writeln e write');
  writeln;   (* apenas pula uma linha *)
  i:=100;
  r:=3.14;
  c:='A';
  s:='interessante';
  writeln('Valor de i e igual a ',i);
  write('valor de r = ');
  writeln(r);
  writeln(c,'  ',s);
end.
```

Este programa resulta na seguinte tela:

```
Exemplos de aplicação de writeln e write
  Valor de i e igual a 100
  valor de r =   3.1400000000E+00
  A interessante
```

Segunda forma: Write(parâmetro : n);

onde *n* é um número inteiro que determina quantas colunas o cursor deve ser deslocado à direita, antes de o parâmetro ser escrito. Além disso, o parâmetro é escrito da direita para a esquerda, exemplo:

```
Program Escrever2;
Uses CRT;
Begin
  Writeln('A');
  Writeln('A':5);
end.
```

Este programa resulta na seguinte tela:

```
A
....A
```

Os pontos representam espaços em branco.

Terceira forma: Write(parâmetro : n : d);

onde ***n*** tem a mesma função que no caso anterior e *d* representa o número de casas decimais. Obviamente, *parâmetro* terá que ser do tipo Real.

```
Program Escrever3;
Uses CRT;
Var r : real;
Begin
   ClrScr;
   r:=3.14156;
   Writeln(r);
   Writeln(r:10:2);
End.
```

Este programa resulta na seguinte tela:

```
3.1415600000E+00
      3.14
```

IV.2 - Read e Readln

São utilizadas para ler dados digitados. A procedure *Read* lê um dado do teclado até se pressionar ENTER; cada tecla digitada é ecoada para o vídeo; após pressionar ENTER, o cursor permanece no mesmo lugar. *Readln* faz a mesma coisa, só que o cursor passa para a próxima linha. A sintaxe geral para estas procedures é:

```
Read (Var_1,Var_2,Var_3,...);
```

```
Program Ler1;
Uses CRT;
Var a,b,c:integer;
     Begin
       clrscr;
       readln(a,b,c);
       writeln (a,' ',b,' ',c);
     end.
```

```
Program Ler2;
Uses CRT;
Var i : integer;
    r : real;
    c : char;
    s : string[10];
Begin
  ClrScr;
  Write('Digite um numero inteiro ------> ');  Readln(i);
  Write('Digite um numero real ---------> ');   Readln(r);
  Write('Digite um caractere -----------> ');     Readln(c);
  Write('Digite uma string -------------> ');       Readln(s);
  Writeln;Writeln; (* pula duas linhas *)
  Writeln(i);
  Writeln(r);
  Writeln(c);
  Writeln(s);
  ReadKey;
End.
```

```
Program AreaTriang;  { calcula área de triângulos }
Uses CRT;
Var Base, altura: Real; (* base e altura do triângulo *)
Begin
  ClrScr;
  Writeln('CALCULO DA ÁREA DE TRIANGULOS':55);
  Writeln;
  Write('Valor da base ------> ');
  Readln(base);
  Writeln;
  Write('Valor da altura ----> ');
  Readln(altura);
  Writeln;  Writeln;
  Writeln('Área do triângulo = ',base*altura/2 : 10 : 2);
  ReadKey;
End.
```

IV.3 **<u>ReadKey</u>**: Lê uma tecla digitada sem que seja necessário pressionar a tecla ENTER

```
Program Ler3;
Uses CRT;
Var tecla: char;
Begin
   Write('digite uma tecla ->');
   Tecla:=readkey;
   Writeln;
    writeln('você digitou ',tecla);
end.
```

IV.4 - Impressora

Podemos enviar dados para a impressora através das procedures Write e Writeln. Para tanto, devemos colocar, antes dos parâmetros a serem enviados à impressora, o nome lógico LST. Exemplo:

```
Writeln('isto vai para o vídeo');
Writeln(lst,'isto vai para a impressora',' e isto também');
```

IV.5 - Funções e procedures para controle de vídeo

IV.5.1 – ClrScr: Limpa a tela e coloca o cursor na primeira coluna da primeira linha. A tela de vídeo é dividida em 80 colunas e 25 linhas.

IV.5.2 - Gotoxy(x,y): Move o cursor para a coluna x e linha y. O canto superior esquerdo tem coordenadas (1,1) e o inferior direito (80,25).

```
Program Video1;
Uses CRT;
Var x,y : Byte;
Begin
   ClrScr;
   Gotoxy(10,2);
   Write('Coluna 10 da linha 2');
   x:=40;
   y:=10;
   Gotoxy(x,y);
   Write('Coluna 40 da linha 10');
  ReadKey;
End.
```

IV.5.3 – ClrEol: Limpa desde a posição atual do cursor até o final da linha.

IV.5.4 – Delline: Procedure que elimina a linha em que está o cursor. As linhas posteriores sobem, ocupando a que foi eliminada.

IV.5.5 – InsLine: Esta procedure faz exatamente o contrário de Delline, ou seja, insere uma linha na posição atual do cursor.

```
Program Video2;
Uses CRT;
Begin
   ClrScr;
   Writeln('linha 1');
   Writeln('linha 2');
   Writeln('linha 3');
   Writeln('linha 4');
   Gotoxy(1,2);       (* posiciona o cursor no início da linha 2 *)
   Delline;
End.
```

O programa anterior produzirá a seguinte tela, eliminando a string “linha 2”:

```
linha 1
linha 3
linha 4
```

```
Program Video3;
Begin
   ClrScr;
   Writeln('linha 1');
   Writeln('linha 2');
   Writeln('linha 3');
   Writeln('linha 4');
   Gotoxy(1,3);       (* cursor na 1a. coluna da 3a. linha *)
   InsLine;
   Write('teste');
   Gotoxy(1,20);
End.
```

Este Programa produzirá a seguinte tela:

```
linha 1
linha 2
teste
linha 3
linha 4
```

V - COMANDOS PARA CONTROLE DO FLUXO DO PROGRAMA

Os comandos de decisão (If, Case, While, Repeat...) a seguir permitem colocar uma condição e fazer o programa desviar, seguindo ora por um, ora por outro caminho. As condições podem ser formadas com operadores relacionais e lógicos.

V.1 - Operadores relacionais

O Turbo Pascal possui ao todo 7 operadores relacionais para a tomada de decisões:

=	igual
<>	diferente
>	maior que
<	menor que
>=	maior ou igual que
<=	menor ou igual que
IN	testa se um elemento existe em um conjunto

1-) Se A=30 e B=50 então
(A = B) FALSE
(A < B) TRUE

2-) Se A=TRUE e B=FALSE
(A <> B) TRUE
(A = B) FALSE

3-) Se A=50 , B=35, C='A' , D='B'
((A < B) OR (C < D)) TRUE
Neste exemplo, a avaliação será verdadeira se uma ou outra expressão for verdadeira; como C < D, então a resposta é TRUE.

V.2 - Operadores lógicos

As expressões formadas com os operadores relacionais, utilizadas em comandos *If* e outros comandos de decisão, podem ser combinadas com os operadores lógicos *and*, *or* e *xor,* formando expressões complexas. Expressões que se tornarem muito complexas devem ser organizadas usando-se parênteses.

AND *e* OR *ou* XOR *ou exclusivo*

A operação AND resulta em TRUE se e somente se todos os operandos forem TRUE; se um deles ou mais de um for FALSE então o resultado será FALSE. A operação OR resulta TRUE quando pelo menos um dos operandos for TRUE. A operação XOR resulta TRUE quando os operandos forem diferentes entre si, isto é, quando um for TRUE o outro deverá ser FALSE.

```
Program operadLogic;
Uses CRT;
Var x,y : boolean;
Begin
     x:=TRUE;
     y:=FALSE;
     Writeln( x OR y );        (* escreve TRUE *)
     Writeln( x AND y );       (* escreve FALSE *)
     Writeln( x XOR y );       (* escreve TRUE *)
     ReadKey;
End.
```

V.3 - If Then Else: O comando If permite ao programa tomar decisões. Ele pode ter duas sintaxes:

Primeira sintaxe: If <expressão_lógica> Then Comando;

Expressão_lógica pode ser simples ou até relações complexas. Se a expressão_lógica resultar verdadeira (TRUE), então o comando será executado, caso contrário, não. Para os casos em que tivermos mais de um comando para ser executado, eles deverão vir delimitados pelas palavras Begin e End.

```
If <expressão_lógica> Then
Begin
   Comando_1;
   Comando_2;
   Comando_3;
  . . .
End;
```

No caso anterior, se expressão_lógica for *TRUE,* todos os comandos entre as palavras *Begin* e *End* serão executados, caso contrário, nenhum deles. Observe que os comandos 1, 2 e 3 estão deslocados para a direita: isto facilita visualizar que eles dependem da condição *then* do *if* para serem executados (caso contrário, não serão). Esta endentação é importante para a clareza do programa.

Segunda sintaxe: If <expressão_lógica> Then Comando_1
Else Comando_2;

Neste caso, se *expressão_lógica* for TRUE então comando_1 será executado e comando_2 não; caso contrário, comando_2 será executado e comando_1 não. Repare que não temos ***;*** *(ponto e vírgula)* no final de comando_1: antes do *else* não colocamos o ponto e vírgula.

Podemos também escrever, usando begin .. end:

```
if <expressão> then
        begin
            comando_1;
            comando_2;
            . . .
        end         (* não tem ; *)
        else begin
            comando_3;
            comando_4;
            . . .
        end;
```

```
Program UsoIf1; { testa número digitado }
  uses crt;
  var i : integer;
  begin
    clrscr;
    write('digite um inteiro maior que 100 --> ');
    readln(i);
    writeln;  writeln;
    if (i>100) { observe a endentação }
       then writeln('você conseguiu')
       else writeln(i,' não e maior que 100');
    ReadKey;
  end.
```

Comando_1, 2... podem ser outras instruções If. Neste caso dizemos que temos *Ifs aninhados*. A endentação se torna mais importante quanto mais complexo for o ninho de ifs que for escrito. Neste caso é interessante raciocinar desenhando o aninhamento dos ifs como se fosse uma árvore de diretórios ou um organograma.

```
Program UsoIf2;  {Determina o maior numero entre dois lidos do teclado }
Uses CRT;
var numero_1,numero_2 : integer;
begin
  clrscr;
  write('primeiro numero ----> ');
  readln(numero_1);
  write('segundo numero -----> ');
  readln(numero_2);
  writeln;
  writeln;
{ observe a endentação dos comandos para aumentar a clareza do programa }
  if (numero_1 > numero_2)
    then write(numero_1,' e o maior')
    else if (numero_2 > numero_1)
          then writeln(numero_2,' e o maior')
          else writeln('são iguais');
  ReadKey;
end.
```

```
Program UsoIf3; {Coloca em ordem decrescente 3 números lidos do teclado}
Uses CRT;
var x,y,z : integer;
begin
  clrscr;
  write('primeiro numero --> ');   readln(x);
  write('segundo numero ---> '); readln(y);
   write('terceiro numero --> ');   readln(z);
  writeln;  writeln;
{ observe a endentação: importante para a clareza do programa  }
  if (x>=y)
    then if (x>=z)
          then if (y>=z)
                then writeln(x,' ',y,' ',z)
                else writeln(x,' ',z,' ',y)
          else writeln(z,' ',x,' ',y)
    else if (y>=z)
          then if (x>=z)
                then writeln(y,' ',x,' ',z)
                else writeln(y,' ',z,' ',x)
          else writeln(z,' ',y,' ',x);
  ReadKey;
end.
```

O bloco dos *Ifs* aninhados também pode ser escrito usando *begin .. end*. Observe a endentação: importante para a clareza do programa. Assim:

```
if (x>=y)
then begin
   if (x>=z)
   then begin
      if (y>=z)
         then writeln(x,' ',y,' ',z)
         else writeln(x,' ',z,' ',y);
    end
    else writeln(z,' ',x,' ',y);
end   { antes do else: não tem ; }
else begin
   if (y>=z)
   then begin
      if (x>=z)
         then writeln(y,' ',x,' ',z)
         else writeln(y,' ',z,' ',x);
   end
   else writeln(z,' ',y,' ',x);
end;
```

```
Program UsoIf4;
{ Determina se os numeros x y z digitados são lados de um triângulo:
    1-) Se x<y+z e y<x+z e z<x+y então x,y,z são lados de um triângulo e se:
    2-) x=y=z então é um triângulo Equilátero
    3-) x=y ou x=z ou y=z então é um triângulo Isósceles
    4-) x<>y<>z então é escaleno  }
Uses CRT;
   Var x,y,z : Real;
   Tecla : Char;
Begin
  ClrScr;
  Write('X = ');  Readln(x);
  Write('Y = ');  Readln(y);
  Write('Z = ');  Readln(z);
  Writeln;Writeln;
  If (x<y+z) and (y<x+z) and (z<x+y)
    Then If (x=y) and (x=z)
          Then Writeln('TRIÂNGULO EQUILÁTERO')
          Else If (x=y) Or (x=z) Or (y=z)
                Then Writeln('TRIÂNGULO ISÓSCELES')
                Else Writeln('TRIÂNGULO ESCALENO')
    Else Writeln('X,Y,Z NÃO SÃO LADOS DE UM TRIÂNGULO');
  ReadKey;
End.
```

V.4 – Case: Esta instrução nos permite selecionar uma opção baseada no valor de uma variável ou expressão. Existem duas sintaxes, a saber:

Sintaxe número 1:

```
Case <expressão ou variável> of
   <valor 1> : Comando_1;
   <valor 2> : Comando_2;
   . . .
   <valor n> : Comando_n;
End;

Ou

Case <expressão ou variável> of
   <valor 1> : Begin
                 comando_1;
                 comando_2;
                 . . .
               End;
   <valor 2> : Begin
                 comando_1;
                 comando_2;
                 . . .
               End;
   . . .
   . . .
   <valor n> : Begin
                 comando_1;
                 comando_2;
                 . . .
               End;
End;
```

A expressão ou variável no comando *Case* deve ser do tipo simples, normalmente Char ou Integer. Após a avaliação da expressão, seu valor ou o valor da variável é comparado com os diversos valores discriminados. Se houver algum que satisfaça, o(s) comando(s) subsequente(s) será(ão) executado(s).

Sintaxe número 2:

```
Case <expressão ou variável> of
   <valor 1> : Comando_1;
   <valor 2> : Comando_2;
   . . .
   <valor n> : Comando_n;
Else Comando;
End;
```

Neste caso, se o resultado da expressão ou o valor da variável não satisfizer nenhum dos valores discriminados, então o comando (ou os comandos) que estiver(em) na frente da cláusula *Else* será(ão) executado(s).

```
 Program Case1; {Exemplo de Case. Calcula a soma, ou a subtração, ou a
    multiplicação, ou a divisão entre dois números lidos do teclado}
Uses CRT;
Var  oper : Char;
     x,y  : Real;
Begin
  ClrScr;
  Write('Valor de X = ');
  Readln(x);
  Write('Valor de Y = ');
  Readln(y);
  Writeln;
  Write('Operação --> ');
  oper:=ReadKey;
  Writeln(oper);Writeln;
  Case Oper of   { observe também a endentação do Case }
      '+' : Write('X + Y = ':10,x+y:6:2);
      '-' : Write('X - Y = ':10,x-y:6:2);
      '*' : Write('X * Y = ':10,x*y:6:2);
      '/' : Write('X / Y = ':10,x/y:6:2);
      Else  Writeln(oper,'  não e operação');
  End; (* case *)
  ReadKey;
End. (* programa *)
```

V.5 – <u>For:</u> Este comando permite que um grupo de operações ou comandos sejam repetidos um certo número de vezes. Sintaxe geral:

For <variável> := <valor inicial> to/downto <valor final> do <comando>;

A variável deverá ser obrigatoriamente do tipo integer (qualquer um), char ou Boolean. A variação de *variável* entre *valor inicial* e *valor final* será crescente e de um em um, quando utilizamos a palavra *to*, e decrescente de um em um, quando utilizamos *downto*. Após o do, se tivermos n comandos a executar, devemos colocá-los dentro de um bloco begin .. end.

```
Program UsoFor1;
Uses CRT;
Var i : Integer;
Begin
  ClrScr;
  For i:=10 to 15 do Writeln(i); (* para i igual a 10 até 15 faça escreva i *)
  ReadKey;
  For i:=10 downto 1 do Writeln(i);
  ReadKey;
End.
```

```
Program UsoFor2;
Uses CRT;   { Escreve na tela os quadrados dos números de 1 até 20 }
Var i : Integer;
Begin
  ClrScr;
  For i:=1 to 20 do { observe o bloco begin .. end }
     Begin
       Write('Valor de i --> ');
       Write(i:3);
       Write('..... quadrado de i = ');
       Writeln(i*i:5);
     End;
  ReadKey;
End.

Program UsoFor3;
Uses CRT;
     {Este programa calcula a soma dos números compreendidos entre dois
     números lidos do teclado }
Var i,Numero_1,Numero_2,soma : Integer;
   Tecla : Char;
Begin
  ClrScr;
  Write('Primeiro Numero --> ');
  Readln(Numero_1);
  Write('Segundo Numero ---> ');
  Readln(Numero_2);
  Writeln;  Writeln;
  Soma:=0;
  For i:=Numero_1 to Numero_2 do Soma:=Soma+i;
  Writeln('Soma entre ',Numero_1,' e ',Numero_2,' = ',soma);
  Writeln;  Writeln;
  ReadKey;
End.
```

V.6 - <u>Repeat Until :</u> Repete um bloco de instruções até que uma certa condição seja satisfeita. Sua sintaxe é:

```
Repeat
   Comando_1;
   Comando_2;
   Comando_3;
   . . .
Until (expressão_lógica);
```

Neste caso não é necessário *begin .. end*. Todos os comandos entre as palavras reservadas *Repeat* e *Until* serão executadas, até que a *expressão lógica* seja verdadeira (TRUE). Obviamente, devemos ter o cuidado para que ela venha a ser TRUE em determinado momento, pois caso contrário, teremos um loop infinito (o programa fica preso dentro da estrutura *Repeat - Until*).

```
Program Repeat1; { Mostra o funcionamento da estrutura Repeat Until}
Uses CRT;
Var i : Integer;
Begin
  ClrScr;
  i:=1;
  Repeat
    Writeln(i);
    i:=i+1;
  Until i=10;
End.
```

```
Program Repeat2; { Programa que soma os números pares compreendidos
      entre dois números lidos do teclado}
Uses CRT;
Var par,numero_1,numero_2,soma:Integer;
Begin
  Clrscr;
  Soma:=0;
  Write('Primeiro Numero ---> ');
  Readln(numero_1);
  Write('Segundo Numero ----> ');
  Readln(numero_2);
  par:=numero_1;
    If par MOD 2 <> 0 then par:=par+1; (* Verifica se o primeiro número é
         par, se não for adiciona-se um *)
  Repeat
    Soma:=soma+par;
    par:=par+2;
  Until par>numero_2;
  Writeln;writeln;
  Write('Soma dos números pares entre ');
  Writeln(numero_1,' e ',numero_2,' = ',soma);
  ReadKey;
end.
```

```
Program Repeat3; { Programa para cálculo de fatorial.}
Uses CRT;
Var n,i,fatorial:integer;
Begin
  ClrScr;
  Write('Digite N maior que 0 --> ');
  Readln(n);
  Writeln;
  i:=1;
```

```
  if n>1
  then begin
     fatorial := 1;
     Repeat { comando repeat aninhado dentro de um If }
          i:=i+1;
          fatorial:=fatorial*i;
    Until i=n;
    Writeln('fatorial de ':30,n,' = ',fatorial);
  End
  Else Writeln('N deve ser > 0');
  readKey;
End.
```

V.7 – <u>While Do:</u> A estrutura *While..Do* permite controlar o número de vezes que uma instrução ou bloco de instruções será executado. Ela difere da instrução Repeat..Until porque esta só avalia a expressão lógica no final do primeiro *Loop*, enquanto que a instrução *While..Do* avalia a expressão lógica antes da primeira iteração. Isto significa que pode não ocorrer sequer a primeira iteração. A sintaxe de *While..Do* é:

```
While <expressão_lógica> Do <comando>;

ou

While <expressão_lógica> Do
Begin
  comando_1;
  comando_2;
  . . .
End;
```

```
Program While1;  { Programa exemplo que escreve na tela de 0 até 10 }
Uses CRT;
Var i : Integer;
Begin
  ClrScr;
  i:=0;
  While (i<11) Do
    Begin
      Writeln(i);
      i:=i+1;
    End;
End.
```

```
Program While2; { Uso do while p/controlar digitação: lê números digitados,
      calcula a média dos números lidos, a quantidade lida e a soma deles }
Uses CRT;
Const Quant_de_num : Integer = 0;
    Soma          : Real    = 0;
    Media         : Real    = 0;
```

```
Var   Numero      : Real;
      Tecla       : Char;
Begin
  ClrScr;
  Write('Digite valor numérico (menor que 0=fim) --> ');
  Readln(Numero);
  While (Numero>=0) Do
    Begin
      Soma := Soma + Numero;
      Quant_de_num := Quant_de_num + 1;
      Write('Digite valor numérico (menor que 0=fim) --> ');
      Readln(Numero);
    End;
  If Quant_de_num > 0
    Then Begin
           Media := Soma/Quant_de_num;
           Writeln;
           Writeln('Quantidade de números = ',Quant_de_num);
           Writeln('Soma ..... = ',Soma:10:2);
           Writeln('Media ..... = ',Media:10:2);
         End
     Else Writeln('Não se realizou cálculos');
   readKey;
End.
```

```
Program AreaFig;
```

{ Programa para cálculo de área de figuras. O programa usa:

- *um Repeat para controlar a digitação: observe que todo o programa está contido em um bloco repeat .. until,*
- *uma estrutura Case para selecionar a figura e*
- *uma estrutura while .. do para repetir a digitação dos dados de cada figura.*

Observe como é importante a endentação para ver a abrangência (ou escopo) de cada um destes comandos. Este programa já tem um tamanho tal que seria aconselhavel quebrá-lo em partes: procedures e functions }

```
Uses CRT;
Var escolha,continua : Char;
     x,y : real;
Begin
  Repeat
    ClrScr;
    Write('Calculo de área de figuras':53);
    Gotoxy(25, 5);Write('1 - Sair do programa');
    Gotoxy(25, 7);Write('2 - Triângulo');
    Gotoxy(25, 9);Write('3 - Quadrado');
    Gotoxy(25,11);Write('4 - Retângulo');
    Gotoxy(25,13);Write('5 - Circulo');
    TextBackGround(7);
```

```
TextColor(0+16);
Gotoxy(10,17);Write('Sua escolha ---> ');
escolha:=ReadKey;
Textbackground(0);
Textcolor(14);
Case escolha of
  '2' : Begin
          ClrScr;
          Writeln('Calculo da área de triangulos':55);
          continua:='s';
          While Continua='s' Do
          Begin
            Writeln;
            Write('Base   = ');
            Readln(x);
            Write('Altura = ');
            Readln(y);
            Writeln;
            Writeln('Área = ',x*y/2:8:2);
            Writeln;
            Writeln;
            Write('Mais cálculos (s/n) --> ');
            continua:=ReadKey;
            Writeln;Writeln;
          End;   { end while }
        End;    { end '2' }

  '3' : Begin
          ClrScr;
          Writeln('Calculo da área de quadrados':55);
          continua:='s';
          While Continua='s' Do
          Begin
            Writeln;
            Write('lado = ');
            Readln(x);
            Writeln; Writeln('Área = ',x*x:8:2);
            Writeln; Writeln;
            Write('Mais cálculos (s/n) --> ');
            continua:=Readkey;
            Writeln;Writeln;
          End;   { end while }
        End;    { end '3' }

  '4' : Begin
          ClrScr;
          Writeln('Calculo da área de retangulos':55);
          continua:='s';
          While Continua='s' Do
           Begin
```

```
                    Writeln;
                    Write('comprimento = ');
                    Readln(x);
                    Write('largura     = ');
                    Readln(y);
                    Writeln;
                    Writeln('Área = ',x*y:8:2);
                    Writeln;  Writeln;
                    Write('Mais cálculos (s/n) --> ');
                    continua:=readkey;
                    Writeln; Writeln;
                  End;   { end while }
                End;     { end '4' }

          '5' : Begin
                  ClrScr;
                  Writeln('Calculo da área de circulos':55);
                  continua:='s';
                  While Continua='s' Do
                  Begin
                    Writeln;
                    Write('raio = ');
                    Readln(x);
                    Writeln;
                    Writeln('Área = ',PI*x*x:8:2);
                    Writeln; Writeln;
                    Write('Mais cálculos (s/n) --> ');
                    continua:=readkey;
                    Writeln; Writeln;
                  End;   { end while }
                End;     { end '5' }
      End;
    Until escolha='1';   { fim do repeat }
  End.
```

V.8 <u>Uso do while e repeat para controle de digitação de tela</u>:

Quando usamos o while para controle da digitação – veja o programa While2 (pág. 25) – a variável usada no controle do loop (Numero) precisa ser digitada *antes* de entrar no while para a primeira iteração e novamente *ao final* do loop para a(s) iteração(ões) seguintes. Observe que as linhas de código:

```
Write('Digite valor numérico (menor que 0=fim) --> ');
Readln(Numero);
```

são colocadas *antes do while* (para a primeira iteração) e *antes do end* (que termina o bloco While para as outras iterações).

Quando usamos o repeat para controle da digitação – veja o programa AreaFig (pág. 26) – a variável usada no controle do loop *(escolha)* é digitada *dentro* do bloco repeat..until. Repetimos a seguir o bloco de controle da digitação do programa While2 usando o *while* e usando o *repeat* para efeito de comparação.

```
Program While2; { usando o while }
Begin
  ClrScr;
  Write('Digite valor numérico (menor que 0=fim) --> ');
  Readln(Numero);
  While (Numero>=0) Do
  Begin
      Soma := Soma + Numero;
      Quant_de_num := Quant_de_num + 1;
      Write('Digite valor numérico (menor que 0=fim) --> ');
      Readln(Numero);
  End; { veja o restante do programa na página 25 }

Program While2; { usando o Repeat }
Begin
   ClrScr;
   Repeat
      Write('Digite valor numérico (menor que 0=fim) --> ');
      Readln(Numero);
      If Numero > 0
      Then begin
         Soma := Soma + Numero;
         Quant_de_num := Quant_de_num + 1;
       End;
   Until (numero < 0);  { veja o restante do programa na página 25 }
```

Observe que ao usar o *Repeat* tivemos que colocar o If para não permitir que o último número digitado (menor ou igual a zero) afete o cálculo da média.

V.9 - <u>Labels e Goto:</u> A instrução *Goto* permite desviar a seqüência de execução do programa para um determinado *Label* pré-definido. Para utilizarmos algum *Label*, ele deve, obrigatoriamente, ser declarado na subárea *Label*.

```
Program LblGoTo1;  {Programa para colocar em ordem crescente 3 números lidos
   do teclado. Uso do GoTo para controle de digitação }
    Uses CRT;
    Label Inicio;
    Var x,y,z : Integer;
       tecla : Char;
    Begin
      Inicio:
      ClrScr;
      Write('Primeiro numero --> ');
      Readln(x);
      Write('Segundo numero ---> ');
      Readln(y);
      Write('Terceiro numero --> ');
      Readln(z);
```

```
    Writeln;
    Writeln;
    If (x>=y)
      Then If (x>=z)
             Then If (y>=z)
                    Then Writeln(x,' ',y,' ',z)
                    Else Writeln(x,' ',z,' ',y)
             Else Writeln(z,' ',x,' ',y)
      Else If (y>=z)
             Then If (x>=z)
                    Then Writeln(y,' ',x,' ',z)
                    Else Writeln(y,' ',z,' ',x)
             Else Writeln(z,' ',y,' ',x);
    Writeln;
    Write('Deseja Continuar --> ');
    Tecla:=Readkey;
    If ((Tecla = 'S') OR (Tecla = 's')) Then Goto Inicio;
  End.

  Program LblGoTo2; { Determina se os valores x y z digitados formam um
                      triângulo. Supondo que x,y,z, sejam os valores lidos, então:
       1-) Se x<y+z e y<x+z e z<x+y então x,y,z são lados de um triângulo e se:
       2-) x=y=z então é um triângulo Equilátero
       3-) x=y ou x=z ou y=z então é um triângulo Isósceles
       4-) x<>y<>z então é escaleno }
Label INICIO;
  Uses CRT;
  Var x,y,z : Real;
     Tecla : Char;
  Begin
  INICIO:
    ClrScr;
    Write('X = ');  Readln(x);
    Write('Y = ');  Readln(y);
    Write('Z = ');  Readln(z);
    Writeln;Writeln;
    If (x<y+z) and (y<x+z) and (z<x+y)
      Then If (x=y) and (x=z)
             Then Writeln('TRIÂNGULO EQUILÁTERO')
             Else If (x=y) Or (x=z) Or (y=z)
                    Then Writeln('TRIÂNGULO ISÓSCELES')
                    Else Writeln('TRIÂNGULO ESCALENO')
      Else Writeln('X,Y,Z NÃO SÃO LADOS DE UM TRIÂNGULO');
    Writeln;Writeln;
    Write('Deseja Continuar ? --> ');
    Tecla:=ReadKey;
    If (Tecla='s') Or (Tecla='S')
      Then Goto INICIO;
  End.
```

```
Program LblGoTo3; { Cálculo de fatorial de um número lido do teclado }
   Uses CRT;
   Label Inicio,fim;
   Var n,Fatorial,i : Integer;
   Begin
     Clrscr;
   Inicio:
     Write('N = ( menor que 0 = fim) --> ');
     Readln(n);
     If n<0 then goto Fim;
     Fatorial:=1;
     Writeln;
     If (n>0)
       Then For i:=1 to n do
                Fatorial:=Fatorial*i;
     Writeln('Fatorial de ':30,n,' = ',fatorial);
     Writeln;
     Goto Inicio;
   Fim:
   End.
```

VI - PROCEDURES

Uma das técnicas mais utilizadas e tida como vantajosa na criação de programas grandes é a modularização. Consiste em dividir o programa em diversos módulos ou subprogramas, de certa forma dependentes uns dos outros. Existe um módulo que é o principal, a partir do qual são chamados os outros. Este módulo recebe o nome de programa principal, enquanto que os outros são chamados de subprogramas. No Turbo Pascal, existem dois tipos de subprogramas:

- Procedures (procedimentos)
- Functions (funções)

Um subprograma é como se fosse um programa: tem a estrutura praticamente igual à de um programa, pode ser chamado pelo programa principal, por outro subprograma e até por ele mesmo (recursividade).

VI.1 – Programa com procedures e funções

A estrutura de um programa completo em Pascal, com procedures e funções, pode ser vista no exemplo a seguir.

```
Program ProcFunc;  { este e o cabeçalho do programa }
USES Crt;   { units que serão juntadas na compilação e linkedição }
Label  fim;        { a partir deste instante posso utilizar o label fim }
Const  Meu_Nome = 'Thelmo'; { nesta área podemos definir todas as constantes que
              quisermos utilizar no programa }
Type n = (BRASILEIRA, PORTUGUESA, INGLESA, FRANCESA, ALEMÃ,
```

```
        AMERICANA);  { definição de  tipo de variável criado pelo programador  }

Var     idade           :integer;
        altura          :real;
        nome            :string[30];
        sexo            :char;
        nacionalidade   : n;
        { todas as variáveis utilizadas no corpo do programa devem ser declaradas
        na subárea Var. Estas variáveis também podem usadas dentro das
        procedures e functions, pois têm escopo global. A variavel nacionalidade foi
        definida pela declaração Type  }

Procedure Linha; { a procedure equivale ao conceito de sub-rotina ou subprograma.
        Sua estrutura pode se tornar tão complexa como a de um programa. Esta
        procedure apenas traça uma linha na posição atual do cursor }
Var i: integer; { esta variável só vale dentro desta procedure }
Begin
        For i:=1 to 80 do Write('-');
end;

Function Soma(x,y: integer): integer; { uma function também é um subprograma
        como a procedure, mas retorna um valor no ponto onde é chamada; esta
        function aqui retorna um valor do tipo inteiro }
Begin
        Soma:=x+y;
end;

        { Podemos definir quantas procedures e functions quisermos }

                { Aqui começa o programa propriamente dito }
Begin
        ClrScr; { apaga a tela }
        Linha;  { Executa a procedure linha }
        Writeln('Meu nome e -------> ',Meu_Nome);
        Linha;
        Write('Qual o seu nome ----> ');
        Readln(Nome);
        Linha;
        Write('Qual a sua idade ---> ');
        Readln(idade);
        Linha;
        Writeln('nossas idades somam --> ',Soma(34,idade));
        Linha;
        goto fim; { as linhas a seguir serão puladas, até o label Fim }
        nacionalidade:=BRASILEIRA;
        Write('Minha nacionalidade e brasileira');
fim:
        Write('Prazer em conhece-lo');
End.
```

Como já vimos, todo programa em Pascal é subdividido em 3 áreas: cabeçalho do programa, área de declarações e corpo do programa. A área de declarações é subdividida em seis subáreas: Label - Const - Type – Var – Procedures – Functions – Corpo do programa.

Já vimos a declaração *USES* (item II.2). Na subárea *Label*, devemos declarar todos os labels que serão usados no corpo do programa. Os labels são utilizados em conjunto com a instrução *goto*. Também já vimos a subárea *Const* no mesmo item II.2.

Na subárea Type podemos criar novos tipos de variáveis. A seguir, todas as variáveis utilizadas no programa devem ser declaradas na subárea *Var*, pois a alocação de espaço de memória para as variáveis é feita durante a compilação.

Na subárea *Procedures*, podemos definir quantas sub-rotinas quisermos. Elas são chamadas durante o programa pelos seus respectivos nomes.

Na subárea *Functions* podemos definir novas funções que serão utilizadas no programa. O Turbo Pascal possui inúmeras funções pré-definidas.

Estas subáreas não são obrigatórias. De acordo com a definição padrão da Linguagem Pascal, estas subáreas devem aparecer na seqüência: *Uses - Label - Const - Type - Var - Procedures - Functions*. Mas no Turbo Pascal isto é livre.

VI.2 - Declaração de procedures

Uma procedure tem praticamente a mesma estrutura de um programa, ou seja, ela contém um cabeçalho, área de declarações e o corpo da procedure. Na área de declarações da procedure, podemos ter as seguintes sub-áreas:

Label - Const - Type - Var - Procedures - Functions.

Devemos salientar que tudo que for declarado dentro das sub-áreas só será reconhecido dentro da procedure. Mais para frente, voltaremos a falar sobre isso.

```
Program ExProc1;  (* cabeçalho do programa *)
    Uses CRT;

    Procedure linha;      (* cabeçalho da procedure linha *)
    Var i : integer;      (* subárea Var da procedure linha *)
    Begin                 (* corpo da procedure linha *)
      for i:=1 to 80 do write('-');
    End;

    Begin                 (* corpo do programa principal *)
      ClrScr;
      linha;              (* ativação da procedure linha *)
      writeln('teste');
      linha;              (* ativação da procedure linha, novamente *)
      readKey;
    End.
```

O programa anterior faz o seguinte:

1-) Apaga a tela e coloca o cursor em 1,1

2-) Executa a procedure *linha*

3-) Escreve a palavra *teste* e, em seguida, executa novamente a procedure *linha*.

A procedure *linha* traça uma linha a partir da posição atual do cursor. Uma observação importante a ser feita neste instante é que a variável inteira *i*, definida dentro da procedure linha, só existe dentro da procedure; isto significa que toda vez que ativamos a procedure *linha*, a variável 'i' é criada e toda vez que saímos da procedure *linha*, ela é destruída.

VI.3 - Passagem de parâmetros

No exemplo acima, ao ativarmos a procedure linha, não houve passagem de parâmetros, mas poderia haver, repare no exemplo abaixo:

```
Program ExProc2;
    Uses CRT;
    Var i,j:integer;

    Procedure soma(x,y:integer); { parâmetros declarados: tipo inteiro }
    Begin
      writeln(x+y);
    end;

    Begin    { começo do programa principal }
      ClrScr;
      soma(3,4); { parâmetros passados: duas constantes inteiras }
      i:=45;
      j:=34;
      soma(i,j); { parâmetros passados: duas variáveis inteiras }
      readKey;
    end.
```

A procedure *soma* depende de dois parâmetros inteiros, e, ao ativarmos esta procedure, devemos fornecer os dois parâmetros. Esses parâmetros podem ser duas constantes inteiras ou duas variáveis inteiras: os parâmetros passados devem ser do mesmo tipo que os parâmetros declarados.

```
Program ExProc3;
    Uses CRT;
    Var i,j:integer;
    Procedure soma(x,y:integer;h,g:real);
    Begin  { parâmetros declarados: dois inteiros e dois reais }
      writeln(x+y);
      writeln(h/g:10:2);
    end;

    Begin   { começo do programa principal }
      ClrScr;
      i:=34;
      j:=35;
      soma(i,j,3.4,4.5); { parâmetros passados: dois inteiros e dois reais }
    End.
```

No exemplo anterior houve passagem de parâmetros para as procedures, mas elas também podem passar dados de volta para o programa que as chama, usando a declaração *var*. Exemplo:

```
Program ExProc4;
Uses CRT;
Var i : Integer;
Procedure Soma(x,y:Integer;Var z:Integer);
Begin
  z:=x+y;
End;
Begin    { começo do programa principal }
  ClrScr;
  Soma(3,4,i); { i voltará com o valor da soma = 7 }
  Writeln(i);
End.
```

Da forma como foi declarada a procedure soma, quando a ativamos com a seqüência Soma(3,4,i), ocorrem as seguintes passagens:

- O número 3 é passado para x
- O número 4 é passado para y
- O valor do parâmetro z é devolvido para i.

Como podemos ver, houve passagem de dados do programa chamador para a procedure e da procedure para o programa chamador.

VI.4 - A declaração forward

Às vezes precisamos usar uma *function* ou *procedure* antes de a ter declarado. Suponha o programa abaixo:

```
Program ExProc5;
Uses CRT;
Procedure Soma(x,y:Integer);
Begin
  linha;  { procedure linha: ainda não foi declarada }
  Writeln(x+y);
End;

Procedure Linha;
Var i:integer;
Begin
  For i:=1 to 80 do Write('-');
End;

Begin
  ClrScr;
  Soma(3,4);
End.
```

Repare que a procedure *Soma* chama uma procedure chamada *linha* que está declarada mais à frente. Portanto, ao compilar o programa, o compilador irá reclamar que não conhece o identificador *Linha,* porque a compilação é feita de cima para baixo e da esquerda para a direita. Para tanto, podemos usar a declaração *Forward*, cuja finalidade é a de indicar ao compilador que determinada procedure está definida mais para frente.

```
Program ExProc5;
      Uses CRT;
      Procedure Linha; Forward;
      Procedure Soma(x,y:Integer);
{ o restante do programa permanece idêntico }
```

VI.6 - O escopo de objetos num programa

Reparem o exemplo abaixo:

```
Program ExProc6;
      Uses CRT;
      Const a=100;              (* constante global *)
      Label fim;                (* Label global *)
      Var i,x,y : Integer;      (* variáveis globais *)

      Procedure Linha;
      Var i : Integer;          (* i é local à procedure linha *)
      Begin
        For i:=1 to 80 do Write('-');
      End;

      Procedure Teste;
        Procedure SubTeste; { procedure SubTeste é local à procedure Teste }
        Begin
          Write('Estive em sub_teste');
        End;
      Begin   { início do corpo da procedure Teste }
        Sub_teste;
        Writeln;
      End;

      Begin   { início do corpo do programa principal }
        ClrScr;
        i:=100;
        Linha;
        x:=20;
        y:=30;
        Teste;
        Linha;
        Writeln('i=',i,' y=',y,' x=',x);
      End.
```

Todos os elementos (constantes, variáveis, labels etc.) que forem definidos antes de começar o corpo do programa são considerados *globais* e podem ser utilizados por todas as procedures, functions e pelo próprio programa. O espaço para tais elementos é criado durante a compilação. Os elementos declarados dentro de uma procedure só existem dentro da procedure: ao declararmos uma variável dentro de uma procedure, toda vez que ativarmos a procedure, tal variável será criada e ao sairmos da procedure ela será destruída. Portanto, dizemos que esta variável é *local* à procedure.

No entanto, vemos no nosso exemplo que existe uma variável *i* inteira declarada antes do início do programa, portanto *global*, e outra dentro da procedure *linha*, portanto *local* a esta procedure. Não há problema, pois o Turbo Pascal irá considerá-las diferentes: quando estivermos dentro do programa, teremos acesso à variável global e quando estivermos dentro da procedure, teremos acesso à variável local.

VII - FUNCTIONS

As funções são quase idênticas às procedures. A principal diferença é que o indicador da função assume o valor de retorno da função. Uma função sempre retorna um valor e, em Turbo Pascal, este valor é retornado no nome da função.

VII.1 - Declaração de funções

A declaração de uma função é muito parecida com de uma procedure que por sua vez é parecida com a de um programa, senão vejamos:

```
Function NomeDaFuncao(parâmetros) : Tipo do dado retornado;
    < área de declarações >
    Begin
        corpo da função
    End;
```

O nome da função deve seguir as mesmas regras para nomes de identificadores em Turbo Pascal. Dentro dos parênteses devemos declarar os parâmetros e seus respectivos tipos dos quais a função depende. O tipo de valor retornado pela função também deve ser declarado.

Na área de declarações, podemos declarar labels, constantes, variáveis e até mesmo outras Procedures e Functions. Devemos lembrar que tais elementos só podem ser utilizados dentro do corpo da função, pois são locais a ela.

O Turbo Pascal possui várias funções e procedures pré-definidas; algumas já usamos, outras iremos ver no decorrer do curso.

```
Program ExFunc1;
Uses CRT;
Var x,y : Real;                 (* variáveis globais *)
 Function Soma(a,b:real):real;     (* Soma é uma função que depende de
          dois parâmetros reais e devolve um valor real *)
Begin
  Soma:=a+b;  (*o valor da funçåo é atribuído ao seu nome *)
End;
```

```
Begin  { corpo do programa principal }
  ClrScr;
  x:=Soma(4,5);
  y:=Soma(3,6)-Soma(45.5,5.6);
  Writeln(x:10:2,y:10:2);
  Writeln;
  Write('Valor de x --> ');
  Readln(x);
  Write('Valor de y --> ');
  Readln(y);
  Writeln;
  Writeln(Soma(x,y):10:2);
End.
```

```
Program FatFunc; { Calcula o fatorial usando o conceito de Function }
Uses CRT;
Var n : Integer;
      tecla : char;

Function Fatorial(numero:integer) : Real;
Var i   : Integer;
     Fat : Real;
Begin (* da função Fatorial *)
  Fat:=1;
  If numero>1
    Then Begin
           i:=1;
           Repeat
             i:=i+1;
             Fat:=Fat*i;
           Until i=numero;
    End;
  Fatorial:=Fat;
End; (* da função fatorial *)

Begin  (* do programa *)
  ClrScr;
  Repeat
    Write('Digite n (menor que 0 = fim) --> ');
    Readln(n);
    Writeln;
    If n >= 0
       Then Writeln('Fatorial de n = ',fatorial(n):10:0);
    Writeln;
  Until n < 0;
End. (* do programa *)
```

```
Program FiboFunc; {Programa para determinar um determinado elemento da
        seqüência de Fibonacci. A seqüência de Fibonacci é definida como
                Fib(0)  =  0
                Fib(1)  =  1
                Fib(n)  =  Fib(n-1) + Fib(n-2) }
      { Assim, o elemento atual é calculado pela soma dos dois anteriores }
      Uses CRT;
      Var numero:integer;
          tecla : char;

      Function Fib(n:integer):integer;
      Var  a1,a2,i,pe : Integer;
      Begin
        if n=0
          Then Fib:=0
          Else If n=1
                Then Fib:=1
                Else Begin
                       a1:=0;
                       a2:=1;
                       i:=1;
                       Repeat
                         pe:=a1+a2;
                         i:=i+1;
                         a1:=a2;
                         a2:=pe;
                       Until i=n;
                       Fib:=a2;
                End;
      End;  { fim da function }

      Begin
        ClrScr;
        Repeat
          Write('Fib(');
          Read(numero);
          Writeln(') = ',fib(numero));
          Writeln;
          Write('Deseja continuar? S-sim --> ');
          tecla :=  ReadKey;
        Until ((tecla <> 'S') and (tecla <> 's'));
      End.
```

VII.3 - Recursividade

A linguagem Pascal permite a utilização de funções recursivas. Uma função é dita recursiva quando ela chama a si mesma. Devemos tomar cuidado ao lidar com esse tipo de função, pois podemos criar loops infinitos. Existem pessoas que têm facilidade para pensar

recursivamente e outras não. A recursividade permite criar funções elegantes e torna os programas mais fáceis de serem entendidos. Abaixo, temos os mesmos programas anteriores, utilizando o conceito de recursividade.

```
Program FatorRec;
Uses CRT;
Var n : Integer;
      tecla : char;

Function Fat(n:integer):real;
Begin
  if n=0
     Then Fat:=1
     Else Fat:=n*Fat(n-1);   (*a função Fat chama a si própria *)
End;

Begin  (* do programa *)
  ClrScr;
  Repeat
     Write('Digite n (menor que 0 = fim) --> ');
     Readln(n);
     Writeln;
     If n >= 0
        Then Writeln('Fatorial de n = ',fat(n):10:0);
     Writeln;
  Until n < 0;
End. (* do programa *)


Program FiboRec;
Uses CRT;
Var numero:integer;
      tecla : char;

Function Fib(n:integer):integer;
Begin
  If n=0
  Then Fib:=0
  Else If n=1
         Then Fib:=1
         Else Fib:=Fib(n-1)+fib(n-2);
End;

Begin  { início do corpo do programa principal }
  ClrScr;
  Repeat
      Write('Fib(');
      Read(numero);
      Write(') = ',fib(numero));
      Writeln;
```

```
        Write('Deseja continuar? S-sim --> ');
        tecla :=  ReadKey;
    Until ((tecla <> 'S') and (tecla <> 's'));
End.
```

VIII - TIPOS DE DADOS ESTRUTURADOS

Até agora definimos dados do tipo simples ou não estruturados, como *Byte, Integer, Real, Char e Boolean*. Entretanto, existem outros tipos de dados chamados complexos ou estruturados no Turbo Pascal, a saber:

- Array, Record, Set, File, String.

Já falamos sobre o tipo estruturado *String*, por ser muito utilizado. O tipo *file* refere-se a arquivos de discos e será mais tarde.

VIII.1 - A declaração type

Além dos tipos de dados pré-definidos no Turbo Pascal, podemos também definir novos tipos através da declaração *Type*. A sua sintaxe geral é:

Type identificador = (valor1,valor2,valor3,valor4, ... ,valorN);

O identificador deve seguir as regras dadas anteriormente e entre os parentêses estão os valores que podem ser assumidos. Exemplos:

```
Type
   cor         = (azul,vermelho,branco,verde,amarelo);
   dia_útil    = (segunda,terça,quarta,quinta,sexta);
   linha       = string[80];
   idade       = 1..99;
```

(a partir deste instante, além dos tipos de dados pré-definidos, podemos também utilizar os novos tipos definidos cor,dia_útil, linha e idade *)*

```
Var
    i          : integer;
   d          : idade;
   nome   : linha;
   dia        : dia_útil;
   cores   : cor;
        (* etc. *)
```

Quando informamos os valores que os dados podem assumir através da declaração type, o Turbo Pascal presume, automaticamente, que o valor da direita vale mais que o da esquerda. Por exemplo: no caso da definição de cor, *amarelo* vale mais que *verde*, que por sua vez vale mais que *branco* e assim por diante. Dizemos que os dados estão ordenados.

VIII.2 - Array

Imagine declarar 100 variáveis do tipo integer, isso seria feito da seguinte forma:

```
Var i1,i2,i3,i4,...,i100 : Integer;
```

Isso pode parecer uma brincadeira de mau gosto, mas é possível. Mas podemos também dizer que é um grande incômodo. E se além dessas 100 variáveis, precisarmos também 1000 do tipo Char ou 2000 ou ... . Como podemos ver, as coisas podem se complicar.

VIII.2.1 - Arrays unidimensionais

Turbo Pascal nos fornece um tipo de dado estruturado chamado Array, que nos permite criar um grande número de variáveis de determinado tipo,sem os inconvenientes anteriores.

Exemplo 1:

```
Type Arranjo = Array[1..100] of Integer;
Var  i : Arranjo;
```

ou, sem usar o *Type*:

```
Var i : Array[1..100] of Integer;
```

Após qualquer uma das declarações acima, temos 100 variáveis do tipo *Integer*, cujos nomes são: i[1] i[2] i[3] . . . - i[100]

Exemplo 2:

```
Type  faixa   = 1..2000;
      Arranjo = Array[faixa] Of Char;
Var  Arranjo_simples : Arranjo;
```

Com estas declarações, criamos 2000 variáveis do tipo char com o nome Arranjo_simples.

```
Program ExArray1; {Lê 10 números inteiros do teclado e os escreve na tela
                   ao contrário do que foram lidos}
Uses Crt;
Type faixa   = 1..10;
      arranjo = Array[faixa] Of Integer;
Var  a : arranjo;
     i  : Integer;
Begin
  ClrScr;
  For i:=1 to 10 do
  Begin
      Write('a[',i:2,'] = ');
      Readln(a[i]);
  End;
```

```
  ClrScr;
  For i:=10 downto 1 do writeln(a[i]);
End.
```

```
Program ExArray2; {Digita até 100 números reais e os coloca em ordem }
Uses CRT;
Const Num_max = 100;
Type  faixa   = 1..Num_max;
       arranjo = Array[faixa] of Real;
Var   i,j,n : Integer;
      a     : arranjo;
      z     : Real;
Begin
  ClrScr;
  Writeln('Ordenação de números lidos do teclado':40+19);
                    {escreve no  meio da linha}
  Writeln; Writeln; { pula duas linhas }
  n:=0;
  Writeln('Digite os numeros. Digite um num < 0 para terminar':40+19);
  Writeln;  Writeln;
  Repeat
    n:=n+1;
    Write('a[',n:3,'] = ');
    Readln(a[n]);
  Until (n=Num_max) Or (a[n]<0);
  n:=n-1; { elimina o ultimo no. lido pois e' negativo }
  ClrScr;
  For i:=1 to n-1 Do
    For j:=i+1 to n Do
      If a[i] >= a[j]
        Then Begin
               z:=a[i];
               a[i]:=a[j];
               a[j]:=z;
         End;
  For i:=1 to n Do Writeln(a[i]:10:2);
end.
```

```
Program ExArray3; {Programa semelhante ao anterior só que coloca em
       ordem até 100 nomes lidos do teclado}
Uses CRT;
Const Num_max = 100;
Type  faixa   = 1..Num_max;
       nomes   = String[30];
     arranjo = Array[faixa] of nomes;
Var   i,j,n : Integer;
      a     : arranjo;
      z     : nomes;
```

```
Begin
  ClrScr;
  Writeln('Ordenação de nomes lidos do teclado':40+19);
                          {escreve no meio da linha}
  Writeln;Writeln; { pula duas linhas }
  n:=0;
  Writeln('Digite os nomes. Digite zero para terminar':40+19);
  Writeln;Writeln;
  Repeat
    n:=n+1;
    Write('a[',n:3,'] = ');
    Readln(a[n]);
  Until (n=Num_max) Or (a[n]='0');
  n:=n-1; { elimina o ultimo nome lido pois e' zero }
  ClrScr;
  For i:=1 to n-1 Do
    For j:=i+1 to n Do
      If a[i] >= a[j]
        Then Begin
               z:=a[i];
               a[i]:=a[j];
               a[j]:=z;
             End;
  For i:=1 to n Do Writeln(a[i]:30);
end.
```

```
Program ExArray4; { Lê as notas de alunos de uma classe e depois lista os
          alunos e as respectivas notas menores que 5.0}
  Uses CRT;
  Const  No_de_alunos = 30;
  Type   Classe = Array[1..No_de_alunos] Of Real;
  Var    n : Integer;
         a : Classe;
  Begin
    ClrScr;
    For n:=1 to No_de_alunos Do
    Begin
        Write('Aluno no. ',n:2,' ---> ');
        Readln(a[n]);
    End;
    ClrScr;
    Writeln('Alunos com media menor que 5':40+15);
    Writeln('numero  nota');
    For n:=1 to No_de_alunos Do
      If a[n]<5
        Then Writeln(n:2,a[n]:10:1);
  End.
```

Os exemplos de procedure a seguir são padrões de código que você pode usar em diversas ocasiões. As duas primeiras são para digitação de arrays: na primeira conhecemos o número

de termos t a ser digitado; na segunda, não conhecemos t: o usuário digitará –1 para indicar o fim da digitação do array. A terceira procedure lista o array na tela. O tipo do array Tvetor foi definido numa declaração Type na área global. O programa a seguir usa estas procedures.

```
Procedure DigitaNConhec(var v: TVetor; t: integer);  { t é conhecido }
var i: integer;
begin
  writeln('Digite os elementos');
  for i := 1 to t
     do readln(v[i]);
end;

Procedure DigitaNDesc(var v: TVetor; var t: integer);
{  t retorna o número de elementos digitados  }
begin
  writeln('Digite os elementos – digite -1 para encerrar');  { uso do flag –1 }
  t := 1;
  readln(v[1]);
  while (v[1] <> -1)
  do begin
     t := t + 1;
     readln(v[t]);
  end;
  t := t – 1;
end;

Procedure ExibeVetor(var v: TVetor; t: integer);  { t já é conhecido }
var i: integer;
begin
  writeln('Elementos do vetor');
  for i := 1 to t
     do writeln(v[i]);   { ou write(v[i], ' '); }
end;


Program MedNotas;
Uses Crt;
type TMaterias = array[1..5] of string;
       TNotas = array[1..5] of real;
var vMaterias: TMaterias;
     vNotas: TNotas;
const iNotas = 5;

procedure DigitaMaterias(var v: TMaterias; t: integer);
var i: integer;   { idêntica à DigitaNConhec }
begin
  writeln('Digite as materias');
  for i := 1 to t
     do readln(v[i]);
end;
```

```
procedure DigitaNotas(var v: TNotas; t: integer);
var i: integer;   { idêntica à DigitaNConhec }
begin
  writeln('Digite as Notas');
  for i := 1 to t
    do readln(v[i]);
end;

procedure Lista(var vm: TMaterias; var vn: TNotas; t: integer);
var i: integer;   { similar à ExibeVetor  }
begin
  writeln('Materias   Notas');
  for i := 1 to t
  do begin
    writeln;
    write(vm[i]:8, vn[i]:8:1);
  end;
end;

function Media(var v: TNotas; t: integer): real;
var i: integer;
   soma: real;
begin
  soma := 0;
  for i := 1 to t
    do soma := soma + v[i];
  Media := soma /t;
end;

{ programa principal }
begin
  clrScr;  { limpa a tela }
  DigitaMaterias(vMaterias, iNotas);
  writeln;
  DigitaNotas(vNotas, iNotas);
  writeln;
  Lista(vMaterias, vNotas, iNotas);
  writeln;
  write('Media: ', Media(vNotas, iNotas):9:1);
  readkey;
end.
```

VIII.2.2 – <u>Conversão de números binários</u>

Um interessante problema de uso de arrays é a conversão de números binários para a base decimal e vice-versa. Nós operamos na base 10, porque trabalhamos com 10 algarismos, 0..9. Na base 2 há somente 2 algarismos, 0 e 1. Assim, temos que representar todos os números da base 10 utilizando somente 0s e 1s. Parece complicado ? Nem tanto, veja a correspondência:

BASE 10	BASE 2	BASE 10	BASE 2
0	0	1	1
2	10	3	11
4	100	5	101
6	110	7	111
8	1000	9	1010
10	1010	11	1011
12	1100	...	...

Para converter um número da base 10 para a base 2, dividimos o número que queremos converter por dois, sucessivamente até que o resto seja 0, depois pegamos os restos de baixo para cima.

Exemplo: (23) => (????)
10 2

23 / 2 dá 11 e sobra 1
11 / 2 dá 5 e sobra 1
5 / 2 dá 2 e sobra 1
2 / 2 dá 1 e sobra 0
1 / 2 dá 0 e sobra 1
Portanto (23) => (10111)
10 2

Para converter da base 2 para a base 10, devemos somar as potências de 2:
(10111) => (???)
2 10

4 3 2 1 0 ← potências de 2
(1 0 1 1 1)

$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$
$16 + 0 + 4 + 2 + 1 = 23$

VIII.2.3 - Arrays Multidimensionais

Nos itens anteriores, trabalhamos com Arrays de uma dimensão. No entanto, é possível trabalhar com arrays de mais de uma dimensão e, nesses casos, eles são chamados de arrays multidimensionais ou matrizes.

Exemplo: Var a : array[1..10,2..5] Of Integer;

Na declaração acima, definimos um Array de 40 elementos chamado 'a'. Ele é constituído de 10 linhas numeradas de 1 a 10 por 4 colunas numeradas de 2 a 5. O acesso a cada elemento é feito da seguinte forma:

```
a[1,2]   a[1,3] ...   a[1,5]
a[2,2]   a[2,3] ...   a[2,5]
...      ...   ...   ...
a[10,2]  a[10,3] ...  a[10,5]
```

Poderíamos definir o mesmo array da seguinte forma:

```
Var a : array[1..10] of array[2..5] Of Integer;
```

ou da seguinte forma:

```
Type b = array[2..5] Of Integer;
Var a : array[1..10] Of b;
```

Podemos também definir arrays de maior número de dimensões pelo mesmo processo. O exemplo a seguir tem três dimensões:

```
Var a : array[1..5,1..6,1..7] Of Integer;
```

```
Program Matriz1; {Programa Matriz => lê uma matriz do teclado e em
      seguida multiplica uma coluna ou linha por uma constante. }
Uses CRT;

(* definição das constantes do programa *)
Const NUM_MAX_COL = 20; (* número máximo de colunas *)
      NUM_MAX_LIN = 10;  (* número máximo de linhas  *)
Var a : array[1..NUM_MAX_LIN,1..NUM_MAX_COL] of integer;
    i,j,k,p, nl,nc      : integer;
    lc                  : char;
Begin
  ClrScr;
    (* lê o número de linhas da matriz *)
  Repeat
    Write('Numero de linhas da matriz ----------> ');
    Readln(nl);
  Until nl<=NUM_MAX_LIN;
    (* lê o número de colunas da matriz *)
  Repeat
    Write('Numero de colunas da matriz ---------> ');
    Readln(nc);
  Until nc<=NUM_MAX_COL;
    (* lê a constante de multiplicaåo *)
  Write('Constante para multiplicação --------> ');
  Readln(k);
    (* pergunta se é uma coluna ou linha para ser multiplicada *)
  Repeat
    Write('Coluna ou linha para mult. (c/l) ----> ');
    Readln(lc);
  Until (lc='c') Or (lc='l');
```

```
      (* pergunta pelo número da coluna ou da linha a ser  multiplicada *)
  If lc='c'
  Then Repeat
         Write('Numero da coluna para a multip. -----> ');
         Readln(p);
       Until p<=nc
  Else Repeat
         Write('Numero da linha para a multip. ------> ');
         Readln(p);
       Until p<=nl;
  Writeln;
  TextBackGround(7);
  TextColor(15+16);
  Gotoxy(24,7);
  Write('Entre com os elementos da matriz');
  textcolor(8);
  For i:=1 to nl do
    for j:=1 to nc do
      Begin
        gotoxy(8*j,i+8);
        Write('+');
      End;
  TextBackGround(0);
  Textcolor(13);
       (* lê os elementos da matriz *)
  For i:=1 to nl do
    for j:=1 to nc do
      Begin
        gotoxy(8*j,i+8);
        Read(a[i,j]);
      End;
      (* faz a multiplicaçåo da coluna ou da linha *)
  if lc='c'
    Then for i:=1 to nl do a[i,p]:=a[i,p]*k
    Else for j:=1 to nc do a[p,j]:=a[p,j]*k;
  TextBackGround(0);
  TextColor(15+16);
  Gotoxy(24,7);
     (* apresenta o resultado final na tela *)
  Write('.....Resultado final.....');
  textcolor(13);

  For i:=1 to nl do
    for j:=1 to nc do
      Begin
        gotoxy(8*j,i+8);
        Write(a[i,j]);
      End;
ReadKey;
End.
```

As procedures usadas no programa a seguir são de uso geral. As duas primeiras permitem digitar uma matriz de tamanho conhecido (m, n) numa única coluna (DigMat1Lin) ou em uma tabela (DigMatNCol). A terceira permite listar a matriz em uma tabela (ExibeMatriz). A última permite digitar uma matriz de tamanho indeterminado (DigMatAberta).

```
Program Matriz1;
Uses Crt;
type TMatriz = array[1..3, 1..5] of real;
var mMat: TMatriz; { matriz com 3 linhas e 5 colunas }
     lin, col: integer;

procedure DigMat1Lin(var mat: TMatriz; m, n: integer);
var i, j: integer;
begin
  writeln('Digite os elementos da matriz, 1 por linha');
  for i := 1 to m
  do begin
    writeln;
    for j := 1 to n
      do readln(mat[i,j]);
  end;
end;

procedure DigMatNCol(var mat: TMatriz; m, n: integer);
var i, j: integer;
begin
  writeln('Digite os elementos da matriz, por coluna');
  for i := 1 to m
  do begin
    for j := 1 to n
    do begin
      gotoXy(j*4, i+1); { posiciona o cursor antes de digitar }
      read(mat[i,j]);
    end;
  end;
end;

procedure ExibeMatriz(var mat: TMatriz; m, n: integer);
var i, j: integer;
begin
  writeln('Exibe os elementos da matriz, por coluna');
  for i := 1 to m
  do begin
    for j := 1 to n
      do write(mat[i,j]:5:2);
    writeln;
  end;
end;
```

```
procedure DigMatAberta(var mat: TMatriz; var m, n: integer);
var i, j, iCol: integer;
begin
  writeln('Digite elementos da matriz: -1:fim da linha -2:fim da matriz');
  m := 1;
  n := 1;
  iCol := 0;
  gotoXy(n*4, m+1);  { digita a primeira coluna da primeira linha }
  readln(mat[m, n]);
  while (mat[m, n] <> -2)
  do begin
    while (mat[m, n] <> -1) and (mat[m, n] <> -2)
    do begin
      n := n + 1;
      gotoXy(n*4, m+1);
      read(mat[m, n]);
    end;
    if (mat[m, n] <> -2)
    then begin
      if n > iCol  { guarda o número maior de colunas digitado }
        then iCol := n;
      mat[m, n] := 0;  { zera o último elemento digitado }
      m := m + 1; { desce uma linha }
      n := 1;  { reinicia a digitação na primeira coluna }
      gotoXy(n*4, m+1);
      read(mat[m, n]);
    end;
  end;
  m := m - 1;
  n := iCol - 1;
end;

{ programa principal }
begin
  clrScr;
  DigMat1Lin(mMat, 3, 5);
  ExibeMatriz(mMat, 3, 5);
  readkey;
  clrScr;
  DigMatNCol(mMat, 3, 5);
  ExibeMatriz(mMat, 3, 5);
  readkey;
  clrScr;
  DigMatAberta(mMat, lin, col);
  ExibeMatriz(mMat, lin, col);
  readkey;
end.
```

IX – OUTROS TIPOS ESTRUTURADOS

IX.1 - Tipo Record

Até o presente momento, trabalhamos com estruturas que envolvem dados do mesmo tipo. O tipo *Record* nos permite criar um tipo de dado que é composto de itens de vários tipos. Estes itens dos quais o tipo Record é formado recebem o nome de campos. Suponha que queiramos armazenar os seguintes dados a respeito de uma pessoa: *Nome – Idade – Sexo – Altura*. Até o momento, não temos nenhum tipo de variável capaz de fazer isso, pois os quatros itens são de tipos diferentes, a saber:

Nome ---> String
Idade --> Integer
Sexo ---> Char
Altura -> Real

Como veremos a seguir, o tipo *Record* pode resolver o problema.

IX.1.1 - Definição de Records

A definição de uma variável do tipo record começa com a palavra reservada Record. Seguem-se os campos (variáveis) e os seus tipos. A palavra reservada End seguida de ponto e vírgula termina a definição do Record. Exemplo:

```
Var Nome_Do_Registro    : Record
                Nome    : String[30];
                Idade   : Integer;
                Sexo    : Char;
                Altura  : Real;
            End;
```

OU:

```
Type Registro = Record
                Nome    : String[30];
                Idade   : Integer;
                Sexo    : Char;
                Altura  : Real;
             End;
Var Nome_Do_Registro : Registro;
```

IX.1.2 - Acesso aos elementos da estrutura

Para acessar os elementos da estrutura, ou seja, os campos, nós devemos incluir o nome da variável *record* seguida de um ponto e depois o nome do campo. Exemplos:

```
Nome_Do_Registro.Altura := 1.78;
Nome_Do_Registro.Sexo   := 'M';
Etc...
```

```
  Program ExRecord1; {Lê uma variável do tipo record do teclado e em
          seguida a mostra no monitor}
Uses CRT;
Type Pessoas = Record
               Nome    : String[30];
               Idade   : Integer;
               Sexo    : Char;
               Altura  : Real;
             End;
Var p : Pessoas;
Begin
  ClrScr;
  Write('Nome ------> ');
  Readln(p.Nome);
  Write('Idade -----> ');
  Readln(p.Idade);
  Write('Sexo ------> ');
  Readln(p.Sexo);
  Write('Altura ----> ');
  Readln(p.Altura);
  Writeln;
  Writeln('Voce digitou os seguintes dados :');
  Writeln;Writeln;
  Writeln(p.nome);
  Writeln(p.idade);
  Writeln(p.sexo);
  Writeln(p.altura:6:2);
End.
```

Podemos também definir arrays de records. Vejam o exemplo abaixo:

```
Program ExRecord2; { Programa para ler dados de no máximo 20 pessoas.
          Em seguida é feita uma listagem em ordem alfabética pelo nome}
Uses CRT;
Label fim;
Type Pessoas = Record
               Nome    : String[30];
               Idade   : Integer;
               Sexo    : Char;
               Altura  : Real;
             End;
Var p     : array[1..20] of Pessoas;
      i,x,y : Integer;
     s     : Pessoas;

Begin
  ClrScr;
  i:=0;
```

```
    Repeat
      i:=i+1;
      Write('Nome (0=fim) -> ');
      Readln(p[i].Nome);
      if p[i].Nome='0' then goto fim;
      Write('Idade --------> ');
      Readln(p[i].Idade);
      Write('Sexo ---------> ');
      Readln(p[i].Sexo);
      Write('Altura -------> ');
      Readln(p[i].Altura);
      Writeln;
  fim:
    Until ((p[i].Nome='0') or (i=20));
    If i<20 then i:=i-1;
    For x:=1 to i-1 do
      For y:=x+1 to i do
        If ((p[x].nome) >= (p[y].nome))
          then begin
                 s:=p[x];
                 p[x]:=p[y];
                 p[y]:=s;
               End;
    ClrScr;
    Writeln('NOME':30,'IDADE':6,'SEXO':5,'ALTURA':8);
    For x:=1 to i do
      Writeln(p[x].nome:30,p[x].idade:6,p[x].sexo:5,p[x].altura:8:2);
  End.
```

IX.1.3 - <u>Declaração With</u>

Se existe uma série de campos de uma variável do tipo record que será acessada repetidamente, pode ser cansativo ter que escrever o nome da variável na frente do campo diversas vezes. Para isto, podemos utilizar a declaração With. Sua forma é:

```
WITH Variável_do_tipo_record DO comando;
```

ou:

```
WITH Variável_do_tipo_record DO
  Begin
    comando_1;
    comando_2;
    . . .
  End;
```

```
Program Exemplo_1; { lê uma variável tipo record e em seguida a mostra }
Uses CRT;
Type Pessoas = Record
                 Nome    : String[30];
                 Idade   : Integer;
                 Sexo    : Char;
                 Altura  : Real;
               End;
Var p : Pessoas;
Begin
  ClrScr;
  With p do
    Begin
      Write('Nome ------> ');
      Readln(Nome);
      Write('Idade -----> ');
      Readln(Idade);
      Write('Sexo ------> ');
      Readln(Sexo);
      Write('Altura ----> ');
      Readln(Altura);
      Writeln;
      Writeln('Você digitou os seguintes dados :');
      Writeln;Writeln;
      Writeln(nome);
      Writeln(idade);
      Writeln(sexo);
      Writeln(altura:6:2);
    End;
End.
```

IX.2 - Tipo Set

Na matemática, usamos uma linguagem não só adequada às suas necessidades, mas também ao estudo de outras ciências. Uma boa parte dessa linguagem vem da teoria de conjuntos. Em matemática, definimos um conjunto como sendo uma coleção de objetos, nomes, números etc. Chamamos de elementos aos objetos, nomes, números etc. que pertencem a esse conjunto.

Na linguagem Pascal também podemos utilizar estes conceitos. Um conjunto é uma coleção de elementos semelhantes. O tamanho do conjunto pode ser variável, até no máximo 256 elementos. Um conjunto pode consistir em zero ou mais elementos do mesmo tipo base que, obrigatoriamente deverá ser um tipo simples, podendo ser qualquer escalar com exceção do *REAL*. Os conjuntos têm seus elementos inclusos em colchetes e separados por vírgulas. Podemos ter também a representação da sub-faixa. Exemplos:

```
[1,3,5,7,9,11,13]      - alguns inteiros
[3..7]                 - inteiros entre 3 e 7
[3,4,5,6,7]            - equivalente  ao anterior
['A'..'Z']             - caracteres alfabéticos maiúsculos
[gol,passat,fusca]     - marcas de carro
[]                     - conjunto vazio
```

A forma geral para declaração de conjuntos é:

```
Type
   <identificador> = SET OF <tipo base>;
```

Exemplos:

```
Type
   caracteres       = set of Char;
   letras_maiúsculas = set of 'A'..'Z';
   dígitos          = set of 0..9;
   carros           = set of (fusca,gol,escort,opala);
Var c       : caracteres;
   letras  : letras_maiúsculas;
   números : dígitos;
   marca   : carros;
```

IX.2.1 Operações em tipos Set

Atribuição: (**:=**): Para atribuir valores aos conjuntos. Exemplos:

```
c := ['a','e','i','o','u'];
letras := ['B'..'H'];
números := [0,3,5];    etc...
```

União: (+): O operador união é representado pelo sinal '+'. A união entre dois conjuntos resulta num terceiro conjunto, constituído dos elementos dos dois conjuntos; exemplos:

```
a := [1,2,3];
b := [2,3,4,5];
c := a+b;          resulta  c = [1,2,3,4,5]
```

Intersecção: (*) : Representada pelo sinal '*'. A intersecção entre dois conjuntos, resulta num terceiro conjunto, constituído pelos elementos que fazem parte tanto de um como do outro conjunto. Exemplo:

```
a := [1,2,3];
b := [2,3,4,5];
c := a*b;          resulta  c = [2,3]
```

Diferença: (**-**): Representada pelo sinal '-'. Retorna um conjunto, cujos elementos estão num conjunto mas não no outro. Exemplo:

```
a := [1,2,3,6];
b := [2,3,4,5];
c := a-b;          resulta  c = [1,6]
c := b-a;          resulta  c = [4,5]
```

IX.2.2 Operadores relacionais:

```
a =  b  todos elementos estão em ambos conjuntos
a <> b  alguns ou todos elementos não estão em ambos conjuntos
a >= b  todos elementos de b estão em a
a <= b  todos elementos de a estão em b
a IN b  a é um elemento do conjunto b
```

Neste último caso, a deve ser um elemento do mesmo tipo base do conjunto b.

```
Program ExSet1; {Lê uma tecla e a envia para o monitor até que se digite  'S' ou
          's' ou 'N' ou 'n'}
Uses CRT;
Var tecla : Char;
Begin
  ClrScr;
  Repeat
    Read(kbd,tecla);
    Write(tecla);
  Until tecla IN ['s','S','n','N'];
End.
```

```
  Program ExSet2; {lê uma tecla e diz se é número, letra maiúscula ou letra
          minúscula até que se leia um '?'}
Uses CRT;
Type símbolos = Set of Char;
Var Maiusc, Minusc, Números : símbolos;
    tecla : char;
Begin
  ClrScr;
  Maiusc  := ['A'..'Z'];
  Minusc  := ['a'..'z'];
  Numeros := ['0'..'9'];
  Repeat
    Read(kbd,tecla);
    If tecla IN Maiusc
      Then Writeln('MAIUSCULA')
      Else if tecla IN minusc
             Then Writeln('minuscula')
             else if tecla IN numeros
                    Then Writeln('numero')
                    else Writeln('nada');
  Until tecla = '?';
End.
```

```
   Program ExSet3; {Programa que conta o número de vogais, número de
          consoantes e de brancos numa frase lida do teclado}
Uses CRT;
Type símbolos = set of char;
Var Alfabeto, vogais, consoantes : símbolos;
   frase : string[50];
   v,c,b,x : integer;
Begin
  Vogais:=['a','e','i','o','u','A','E','I','O','U'];
  alfabeto:=['a'..'z']+['A'..'Z'];
  consoantes:=alfabeto-vogais;
  Clrscr;
  Write('Digite uma frase --> ');
  Readln(frase);
```

```
    b:=0;c:=0;v:=0;
          (* a função length() devolve o tamanho da string *)
    For x:=1 to length(frase) do
      if frase[x] in vogais
        then v:=v+1
        else if frase[x] in consoantes
              then c:=c+1
              else if frase[x] = ' ' then b:=b+1;
    Writeln;
    writeln(b,' brancos');
    Writeln(c,' consoantes');
    Writeln(v,' vogais');
  End.
```

X – EXERCÍCIOS E USO DO TURBO PASCAL

X.1 – <u>Exercícios: uso de comandos seqüenciais:</u> faça programas para as tarefas a seguir usando apenas comandos sequenciais e as funções da lista.

X.1.1 digitar três números reais a, b, c. Calcular a média m dos três. Imprimir os três números digitados e o resultado. Alterar depois o programa para imprimir formatado com duas decimais.

X.1.2 digitar um número inteiro. Imprimir o valor do algarismo das unidades.

X.1.3 idem anterior: digitar um número inteiro > 10. Imprimir o valor do algarismo das dezenas.

X.1.4 idem anterior: digitar um número inteiro > 100. Imprimir o valor do algarismo das centenas.

X.1.5 digitar um número inteiro com dois algarismos. Imprimir invertido.

X.1.6 digitar dois números inteiros a b. Trocá-los de posição a=b e b=a. Imprimir invertido.

*(Evaristo, p. 44)**

X.1.7 digitar um número real. Imprimir o quadrado, formatado. Use uma função.

X.1.8 digitar uma temperatura em graus C e imprimi-la em grau F, sendo a fórmula de conversão: $F = (9 * C + 160) / 5$

X.1.9 digitar uma temperatura em graus F e imprimi-la em grau C, sendo a fórmula de conversão: $C = (F - 32) * (5 / 9)$

X.1.10 calcular e imprimir o volume de uma lata de óleo, sendo a fórmula: volume = Pi * R^2 * altura. Usar a função e a constante Pi.

(Evaristo, p. 47)

X.1.11 digitar duas frações ordinárias. Somá-las e imprimir o resultado em forma de fração.

X.1.12 uma loja vende em três parcelas. A segunda e terceira parcelas são valores inteiros; a diferença é jogada na primeira parcela. Faça um programa para digitar o valor da venda e imprimir o valor das três parcelas.

X.1.13 digitar um valor em segundos (p. e.: 3850) e imprimir este valor convertido em horas, minutos e segundos.

X.1.14 faça um programa para um caixa eletrônico pagar uma quantia qualquer em reais, dizendo quantas notas de 50, 20, 10, 2, 1. Não há centavos.

Funções predefinidas (Evaristo, p. 43): para uso com a lista de exercícios anterior.

função	*argumento*	*valor*	*Significado*
abs(x)	real, integer	real, integer	valor absoluto de x
ArcTan(x)	Real	real	número cuja tangente é x
chr(x)	Byte	char	caractere ASCII relativo a x
cos(x)	Real	real	cosseno de x
exp(x)	Real	real	exponencial de x (e^x)
frac(x)	Real	real	parte fracionária de x
ln(x)	Real	real	logaritmo natural de x
odd(x)	Integer	boolean	se x é impar
ord(x)	Ordenado	integer	número de ordem de x no tipo de dado
pred(x)	Ordenado	ordenado	predecessor de x: ver succ(x)
round(x)	Real	integer	arredondamento de x
Sin(x)	Real	real	seno de x
sqr(x)	real, integer	real, integer	quadrado de x
sqrT(x)	real, integer	real	raiz quadrada de x
succ(x)	Ordenado	ordenado	sucessor de x: ver pred(x)
trunc(x)	Real	integer	trunca: parte inteira de x
upCase(x)	Char	char	converte x para maiúsculo

** boa parte dos exercícios destas listas foi tirada do livro: Aprendendo a programar – programando em linguagem Pascal – Jaime Evaristo – Ed Book Express*

X.2 – <u>Exercícios – uso de *if* e *case:*</u> faça programas para as tarefas a seguir usando os comandos anteriores mais *if* e *case*.

X.2.1 digitar um número inteiro com quatro algarismos. Imprimir, informando se o ano é bissexto ou não. Faça observando que, na regra conhecida, um ano é bissexto quando é divisível por 4.

X.2.2 idem anterior. Complete a regra acrescentando que: quando o ano é divisível por 100 e não por 400 não é bissexto. Ex: 1900 não é bissexto.
(Evaristo, p. 52, 53)

X.2.3 digitar dois anos. Imprimir quantos anos bissextos existem entre os dois.

X.2.4 digitar três números a b c. Calcular e imprimir a área de um triângulo. Se a b c não forem lados de um triângulo, imprimir mensagem de erro.

X.2.5 digitar um número inteiro. Imprimir dizendo se é par ou ímpar. Usar função.

X.2.6 digitar três números a b c. Imprimir ordenado do maior para o menor.

X.2.7 digitar dois horários (em horas e minutos). Imprimir a diferença entre os dois. Faça a digitação e a impressão no formato hh:mm. Se o segundo horário for menor do que o primeiro, assuma que o segundo horário se refere ao dia seguinte.

X.2.8 calcular as raízes de uma equação de segundo grau digitando os coeficientes a b c. Se a=0, imprimir mensagem dizendo que não é do segundo grau. Se o determinante (b2-4ac < 0), imprimir mensagem dizendo que as raízes não são reais.

X.2.9 digitar o número de um mês m. Informar quantos dias tem o mês. Use o conhecimento de ano bissexto do problema 2. Teste se 0 < m < 13. Faça o programa usando Ifs aninhados e depois use uma estrutura case.

X.2.10 apurar a aprovação de alunos de uma escola que tem os seguintes critérios de aprovação:

- há quatro notas bimestrais n1, n2, n3 n4
- se a média => 7, o aluno está aprovado: imprima a média e "aprovado"
- se a média < 5, o aluno está reprovado: imprima a média e "reprovado"
- se 5 <= média < 7, o aluno pode fazer uma prova de recuperação:
    - digitar a nota da prova de recuperação, n5;
    - a nota final será: a soma de n1 n2 n3 n4 e n5 (esta com peso = 6)
    - se a nota final >= 6, o aluno está aprovado, senão está reprovado
    - imprima a nota final e a mensagem "aprovado" ou "reprovado"
- use Ifs aninhados primeiro, depois uma estrutura case.

X.2.11 apurar os salários reajustados (s2) de uma empresa que deu aumentos escalonados segundo o seguinte critério:

- 13% para salários (s1) =< 200
- 11% para sálários entre 200 e 400 (inclusive)
- 9% para sálários entre 400 e 800 (inclusive)
- e 7% para os demais salários
- imprima "o salário atual: s1 vai para: s2"
- use Ifs aninhados primeiro, depois uma estrutura case.

X.2.12 no caso anterior, para evitar que um empregado que ganhe 201,00 passe a receber menos do que um ganhava 200,00, dê aumentos cumulativos. Ex.: um salário de 1000,00 terá reajuste de: 13% s/200, mais 11% s/200 (=400 – 200), 9% s/400 (=800 – 400) e 7% s/200 (=1000 – 800). Use uma estrutura case.

X.2.13 digitar a idade de um nadador, classificando-o em uma das categorias: infantil A: 5 a 7 anos, infantil B: 8 a 10 anos, juvenil A: 11 a 13 anos, juvenil B: 14 a 18 anos, senior: maior de 18 anos. Imprimir a idade e a categoria.

X.2.14 digitar uma data, após 1/1/2000. Imprima o dia da semana em que cai, admitindo (hipoteticamente) que o dia 1/1/2000 foi um domingo e todos os meses têm 30 dias.

X.2.15 no mesmo problema anterior, passe a considerar a contagem correta dos dias.

X.2.16 digitar a data de nascimento de uma pessoa (d1, m1, a1) e a data de hoje (d2, m2, a2). Calcular a sua idade em anos, meses e dias. Admita que todos os meses têm 30 dias.

X.2.17 no mesmo problema anterior, passe a considerar a contagem correta dos dias.

X.2.18 digitar os lados de um triângulo. Imprimir informando se é retângulo ou não.

X.3 – <u>Exercícios – uso de for, while e repeat:</u> faça programas para as tarefas a seguir usando os comandos anteriores mais *for, while* e *repeat*:

X.3.1 reescrever o programa de colocar três números em ordem (LblGoTo1, apostila, p. 30), sem usar labels e goTo.

X.3.2 reescrever o programa de cálculo Case1 (apostila, p. 22), para que ele volte a pedir a digitação dos dois números:

- usando while... do na primeira versão
- usando repeat... until na segunda versão

X.3.3 escrever um programa que: (1) digita dois números, lados de um terreno retângulo; (2) calcula a área para terrenos formados variando do menor lado ao maior; (3) imprime uma tabela com duas entradas (linha e coluna) com as áreas calculadas (linha x coluna). Reescreva em seguida o programa para voltar e pedir para digitar os lados novamente.

X.3.4 escrever um programa que: (1) digita duas taxas de juro de mora, ao mês (a diferença máxima entre elas é de 3%); (2) calcula e imprime uma tabela com os juros de mora: nas colunas devem variar as taxas de 0,5% em 0,5% e nas linhas devem variar os dias (1 a 10 dias). Reescreva em seguida o programa para voltar e pedir para digitar as taxas novamente.

X.3.5 escrever um programa que: (1) digita os seguintes dados de uma progressão: a_1 (primeiro termo), n (número de termos, n<=10) e r (razão); em seguida digita se é PA ou PG; (2) calcula e imprime uma tabela com as colunas: i, termo a_i e soma S_i; (3) volta depois a pedir para digitar as variáveis novamente.

X.3.6 uma loja vende a prazo em 1 2 3 4 ou 5 prestações, sem juros, aceitando entradas de 10% 15% 20% 25% e 30% do valor da compra. Escreva um programa para imprimir uma tabela variando as entradas no cabeçalho das colunas e o número de prestações variando nas linhas.

X.3.7 altere o programa anterior, pedindo para o usuário digitar o intervalo das entradas (de – até) e a quantidade máxima de prestações.

(Evaristo, p.72, 81, 86)

X.3.8 digitar duas temperaturas (de—até, inteiros) em graus Farenheit. Incrementar de 1 em 1, convertendo para C = 5 * (Farenheit – 32) / 9. Imprimir uma tabela com as duas temperaturas.

X.3.9 digitar um número inteiro n. Imprimir, dizendo se é primo ou não, voltando depois o programa a pedir para digitar um número novamente. Refaça depois o programa para: se n não for primo, informar seu menor e seu maior divisores.

X.3.10 digitar um número inteiro = número de elementos de um conjunto (n < 6). Imprimir os pares do produto cartesiano A x A.

X.3.11 digitar um número inteiro = número de elementos de um conjunto (n < 6). Imprimir os pares do produto cartesiano A x A, sendo que não pode haver elementos repetidos, por ex.: (x, x) não pode pois x = x e (y, x) não pode se já houver o (x, y).

X.3.12 digitar dois números. Calcular o máximo divisor comum entre os dois. Pode usar o algoritmo de Euclides ou outro que você invente.

X.3.13 digitar dois números. Calcular o mínimo múltiplo comum entre os dois. Pode usar o algoritmo ensinado nas aulas de aritmética ou outro que você invente.

X.3.14 escreva um programa para calcular a soma dos n primeiros termos da seqüência 1/2 3/5 5/8 ...

X.3.15 escreva um programa p/ calcular a soma dos n primeiros termos da seqüência 1 -1/2 1/3 -1/4...

X.3.16 O número 3025 possui a seguinte característica: 30 + 25 = 55 e 55^2 = 3025. Faça um programa que liste todos os números com 4 algarismos com esta propriedade.

X.3.17 Escreva o programa de urna eletrônica para o grêmio da escola:

- são dois candidatos: 33-Fala Sério e 44-Só Promessas; se digitar 00-voto nulo; se digitar 99-fim do programa; qualquer outro número anula o voto
- cada votação deve ser confirmada, inclusive voto nulo; se não coincidir a segunda digitação, manda repetir o voto
- no final o programa deve informar a quantidade de votos das três modalidades e informar o candidato eleito.

X.3.18 Escreva um programa que escreva todos os subconjuntos com três elementos do conjunto {1, 2, 3... }, n dado.

X.4 – Exercícios – uso de functions e procedures:

X.4.1 reescreva o segundo programa da lista anterior (X.3.2) para que ele volte a pedir a digitação dos dois números. Faça agora com que cada operação seja feita por uma função separada: Soma, Subtrai, Multiplica, Divide, devendo cada função devolver o resultado; faça também a digitação em uma procedure separada.

- faça uma primeira versão usando variáveis globais
- faça uma segunda versão passando os parâmetros para a função

X.4.2 escrever um programa que: digita dois números, base e expoente; calcula e imprime a potência (base elevada ao expoente) usando uma função para o cálculo. O programa deve voltar e pedir para digitar os dados novamente; se o usuário digitar zeros, pára.

X.4.3 reescrever o programa que digita os dados de uma progressão: a_1 (primeiro termo), n (número de termos, n<=10) e r (razão) – programa X.3.5 da lista anterior. Calcular e imprimir uma tabela com as colunas: i, termo a_i e S_i. Use a função de potência do X.4.2.

X.4.4 usando a fórmula e o programa que você fez de conversão de temperaturas (C = 5 * (F– 32) / 9), faça um programa em que o usuário digita, num menu:

- se digitar F: digita um valor e converte o valor de C em F, usando uma função,
- se digitar C: digita um valor e converte o valor de F em C, usando uma função,
- se digitar P: pára
- faça uma procedure para digitar os números e a opção desejada.

X.4.5 transforme os programas de cálculo do máximo divisor comum e mínimo múltiplo comum (programas X.3.12 e X.3.13 da lista anterior) em funções. Faça um programa em que o usuário digita, em um menu:

- se digitar M: digita os números e calcula o mMc
- se digitar D: digita os números e calcula o mDc
- se digitar P: pára
- faça uma procedure para digitar os números e a opção desejada.

X.4.6 a fórmula geral de cálculo de juros simples é: j = cin, onde: j: valor do juro, c: valor do capital, i: a taxa de juros, n: número de períodos. Admita que a taxa de juros é dada em % ao mês e n é, então, o número de meses. Faça um programa que calcule qualquer uma destas quatro variáveis, conhecendo as outras três. Assim:

- o usuário deve digitar no menu qual variável deseja calcular: c i n j; se digitar P-pára,
- faça uma função ou procedure para digitar cada variável (que deve ser > zero),
- faça uma função ou procedure para calcular cada variável solicitada.

X.5 – Exercícios – Faça os programas a seguir usando arrays e strings:

X.5.1 Faça um programa para digitar um número qualquer de um mês. Use a function a seguir para imprimir o nome do mês ou a mensagem ‘mês invalido’.

```
function NomeMes(n: integer): string;
const Mes: array[1..13] of string = (‘Janeiro’, ‘Fevereiro’, ‘...’, ‘Dezembro’,’Mes
invalido’);
begin
  if (n > 0) and (n , 13)
     then NomeMes := Mes[n]
     else NomeMes := Mes[13];
end;
```

X.5.2 Faça um programa para digitar um vetor qualquer de números inteiros. Separar em dois vetores, um com os números pares, outro com os números ímpares. Imprimir os dois ao final (JE 127). Use as procedures já estudadas para digitar o vetor e para imprimir.

X.5.3 Escreva um programa que forneça os elementos distintos de um vetor dado. Por ex: se o vetor dado for v = {3, 2, 1, 3, 4, 1, 5, 5, 2}, a procedure deve fornecer v1 = {3, 2, 1, 4, 5} (JE 127). Use as procedures já estudadas para digitar o vetor e para imprimir.

X.5.4 Faça um programa para:

- digitar os pontos de parada do ônibus entre Colatina e Vitória e a distância em km entre cada um e o próximo (admita que há cinco paradas apenas para não dar uma tabela muito grande),
- em seguida, digitar o preço por quilômetro,
- imprimir uma tabela com os preços de Colatina para cada um dos pontos de parada (acumulando a distância),
- repetir o processo para cada ponto de parada depois de Colatina até o final da linha.

X.5.5 Faça um programa para calcular medidas estatísticas: digitar uma série de números; quando digitar valor negativo, cessa digitação. Em seguida, imprimir: a moda (valor com a maior freqüência), a média aritmética, a amplitude (diferença entre o maior e menor valor), o desvio médio (média dos desvios em relação à média) e o desvio padrão (raiz quadrada da média dos quadrados dos desvios. Fazer uma função ou procedimento para cada medida (JE 124). Use as procedures já estudadas para digitar o vetor e para imprimir.

X.5.6 Faça um programa para digitar e imprimir notas de alunos. Cada aluno tem quatro notas (quatro matérias): Pascal, Lógica, Inglês e Matemática, por exemplo. O programa deve:

- digitar os nomes dos alunos, os nomes das matérias e as notas obtidas pelos alunos, quando digitar nome do aluno = brancos, pára a digitação.
- imprimir uma tabela com os nomes dos alunos e a nota de cada matéria; os nomes das matérias ficam na primeira linha e os nomes dos alunos no lado esquerdo,
- imprimir a última coluna com as médias por aluno; imprimir uma linha no final com as médias por matéria.

X.5.7 Uma empresa sorteia prêmios, sendo ganhador o número formado pelas centenas dos cinco primeiros resultados da loteria federal. Por ex.: se a loteria federal deu 23451, 00234, 11935, 24652 e 78712, o ganhador é o número 42967. Faça um programa para digitar os números da loteria e uma função que receba estes números e devolva o número do ganhador. (JE 128)

X.5.8 Faça um programa que:

- digita um vetor v (quantidade de elementos em aberto); use a procedure já estudada,
- cria o vetor v1 com os mesmos elementos só que ordenados (do menor para o maior); crie uma procedure específica para isto,
- imprime o vetor v1 ordenado; use a procedure já estudada.

X.5.9 O programa anterior é um método conhecido de colocar um vetor em ordem. O outro é o programa ExArray2 da página 44. Faça este e o ExArray3 para exercitar o método. Use as procedures de digitar um vetor e imprimir já estudadas. Faça uma procedure separada ordenar.

X.5.10 Faça um programa que (JE 128):

- digita um vetor v (quantidade de elementos em aberto); use a procedure já estudada,
- em seguida, digita um novo elemento x,
- ordena o vetor (use a procedure do anterior), insere o elemento x no vetor mantendo-o ordenado e imprime o vetor com o elemento x inserido (use a procedure já estudada).

X.5.11 Dado o polinômio $P(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + ... + a_{n-2}x^2 + a_{n-1}x^1 + a_n$, escreva um programa que permita (JE 128):

- digitar um vetor v (quantidade de elementos em aberto – use a procedure já estudada) com os coeficientes a_0 até a_n
- em seguida, digitar um valor para x,
- calcular e imprimir y = P(x).

X.5.12 Faça um programa com menu que permita: (1) digitar um número binário qualquer e imprimir o valor convertido para decimal, (2) digitar um número decimal qualquer e imprimir o valor convertido para binário e (3) parar. Use funções para (1) e (2).

X.5.13 Um dos métodos mais usados para cálculo de dígito verificador (dv) é o módulo 11, cuja seqüência de cálculo é a seguinte:

- multiplicar cada dígito da direita para a esquerda por 2, 3, 4, 5..., somando o resultado de cada multiplicação → soma
- dividir a soma por 11; o dv = 11 – resto da divisão; se o dv > 9, dv = 0.

Faça uma função para cálculo deste dv e coloque-o dentro de um programa que digita um número e imprime o dv calculado.

X.5.14 O dv do CPF é composto de dois algarismos calculados pelo módulo 11 em duas vezes:

- na primeira, consideram-se 9 algarismos → gera-se o décimo algarismo = dv1
- na segunda, consideram-se 10 algarismos: os 9 anteriores mais o dv1 → gera-se dv2.

Faça um programa para digitar um número de CPF completo. O programa confere o dv (2 últimas posições) e imprime uma mensagem: "cpf aceito" ou "cpf incorreto". O programa deve voltar para digitar outro número; caso digite zeros, encerra.

X.5.15 Faça um programa para digitar um valor em reais e imprimir o valor por extenso. Se digitar zeros, pára.

X.5.16 Um banco usa uma chave para criptografar valores que é uma string com 10 letras não repetidas. Faça um programa que permita: (1) digitar a chave a usar no dia, (2) digitar um número e imprimir o valor criptografado, (3) digitar um valor criptografado e convertê-lo para número e (4) parar.

X.5.17 Um recurso comum em editores de texto é contar as palavras. Faça um programa que conta o número de palavras escritas numa frase. Se a frase digitada for vazia, pára (JE 141).

X.5.18 Um banco de dados não pode aceitar nomes grandes. Faça um programa para digitar um nome e simplificar os nomes do meio. Se o nome digitado for vazio, pára.

X.5.19 O programa abaixo (JE 45) implementa um recurso não existente no Pascal: converter uma letra maiúscula para minúscula. Ele usa a função chr(x) – lista de exercícios 1 – e a função ord(x), sabendo que ord('A') = 65 e ord('a') = 97 e as demais letras maiúsculas e minúsculas estão na seqüência.

```
program LowCase;
var minuscula, Maiuscula: char;
begin
    writeln('Digite uma letra maiúscula');
    readln(Maiuscula);
    minuscula := chr(ord(Maiuscula) + 32);
    writeln('LowCase(', Maiuscula, ') = ', minuscula);
end;
```

A função upCase(x) faz o contrário: converte x para maiúsculo. Agora, sabendo que um banco de dados precisa garantir que o campo NomeCliente venha escrito com o primeiro caractere de cada nome em maiúsculo e os demais em minúsculo, faça um programa que

permita digitar o NomeCliente de qualquer forma (maiúsculas ou minúsculas) e o devolva devidamente formatado.

X.5.20 Faça um programa para ajudar dois jogadores no jogo da velha, com as seguintes funções:

- desenhar a tela inicial do jogo: uma matriz 3 x 3 na tela, separada por traços,
- pedir para cada jogador (do Zero ou X) digitar a posição em que joga (1:1 a 3:3); escrever 0 ou X depois na posição assinalada,
- verificar depois de cada jogada: se algum jogador ganhou ou se acabou o jogo e perguntar se deseja jogar novamente.

Funções para utilizar com strings e arrays:

length(x): integer	função: tamanho da string; length(x) = ord(x[0])
sizeOf(x): integer	função: retorna o número de bytes ocupado pela variável x; válida também para arrays
concat(s1, s2…sn: string): string	função: junta strings; também pode somar : s:=s1+s2...
Pos(s1, s2: string): byte	função: retorna a posição de s1 dentro de s2; ou 0 se não existir
copy(s: string; i, n:integer): string	função: retorna a substring de n caracteres iniciada em i
delete(var s: string; i, n: integer)	procedure: exclui n caracteres a partir da posição i; retorna s
insert(s1: string; var s2: string; i:integer)	procedure: insere s1 em s2 a partir da posição i
val(s:string; var v: integer/real; var r: integer)	procedure: converte string s para número em v; r= 0 se operação deu certo ou r = primeiro caractere não numérico encontrado
str(x: integer/real; var s: string)	procedure: converte um número em uma string

X.6 – Comandos e atalhos para uso do Turbo Pascal e para uso do debug

X.6.1 Procedimentos para executar o Pascal

- Criar um atalho para executar o Pascal: o nome do programa que executa o Turbo Pascal é *tpx.exe*. Se foi adotado o diretório default para a instalação do TP, o programa estará no diretório *C:\tp\bin*. Para criar o atalho, copia-se o programa (clicar no programa e apertar as teclas *Ctrl C*); depois clica-se com o botão direito na janela do desktop e escolhe-se o menu "colar atalho".
- Sempre que entrar no TP, o primeiro comando a executar é "File / Change dir"; em seguida, fazer o caminho até a sua pasta, dando clique duplo e depois clicando em ok. Isto fará com que todos os arquivos que você salvar sejam gravados na sua pasta.
- Depois de digitado o programa, grave-o antes de testá-lo, pois o computador pode travar e causar a perda do conteúdo. Para salvar o programa: menu "File / Save". O arquivo será gravado na pasta assinalada pelo "File / Change dir".
- Abrir um arquivo: após executar o "File / Change dir", para abrir um arquivo, use o menu "File / Open". Surgirá uma janela com todos os arquivos com terminação *.pas*. Clique em cima do que deseja abrir e depois em Open.
- Para executar o programa use o menu Run / Run, ou aperte as teclas *Ctrl F9*. Se desejar apenas compilar, sem executar, use o menu Compile / Compile, ou aperte as teclas *Alt F9*.

- Na maioria das vezes um programa é parecido com algum anterior que já fizemos. Abra o programa anterior (File / Open) e salve-o com outro nome: menu “File / Save as...”, e altere este novo programa.
- Trechos de programa que se repetem podem ser copiados de um lugar para o outro. Para copiar um trecho: selecione-o com o mouse ou com as setas e use o menu “Edit / Copy”. Para colar o texto em outro lugar: clique no lugar desejado e use o menu “Edit / Paste”. Se quiser apagar um trecho do programa, selecione-o e use o menu “Edit / Clear”. Se apagar inadvertidamente um trecho, pode tentar recuperá-lo usando o menu “Edit / UnDo”. Para apagar uma linha inteira use as teclas *Ctrl Y*. Para apagar uma palavra à sua direita use as teclas *Ctrl T*.
- Você pode abrir mais de um programa e copiar trechos de uma janela de programa para outra. Para alternar de uma janela para outra basta clicar na janela desejada ou usar o menu Window / Next (seguinte) ou Previous (anterior). Podemos fechar a janela que não estiver sendo usada: menu Window / Close.
- Para localizar uma palavra (ou texto) no programa, use o menu Search / Find; para trocar um texto por outro use o menu Search / Replace.
- Para não esquecer de gravar o programa alterado, você pode marcar a seguinte opção: menu Options / Environment / Preferences / Autosave: marque a opção Editor files. Sempre que você executar o programa, o Pascal gravará o arquivo antes.

X.6.2 Procedimentos para efetuar Debug

Às vezes examinamos um programa, achamos que está todo correto e, no entanto, o resultado produzido pelo programa vem errado. Nestes casos, podemos usar um recurso interessante chamado “debug”: colocamos pontos de parada dentro do programa e executamos trechos do programa passo a passo, conferindo o valor de variáveis chave. Os recursos de debug são:

- para inserir um breakpoint (ponto de parada): clicar antes na linha em que se deseja parar; depois, menu “debug / add breakpoint”;
- para executar o programa usando debug:
    - F7: executa passo a passo (uma instrução de cada vez)
    - F8: idem F7, porém pula chamadas a procedures e functions
    - Ctrl F9: executa até o próximo breakpoint
- para conferir o valor de uma variável: menu “debug / evaluate/modify”: digite o nome da variável ou expressão e clique em “evaluate”;
- para eliminar os breakpoints: menu “debug / breakpoints”; clica-se naquele breakpoint que se quer excluir e depois em “delete”. “Clear all”: limpa todos os breakpoints;
- estando em um breakpoint, para parar a execução do programa podemos usar o menu “Run / program reset”.

Para parar a execução de um programa que entrou em loop usamos as teclas “Ctrl break”. Se não der certo, terá de ser usado o Gerenciador de Tarefas do Windows: aperte “Ctrl Alt Delete” e em seguida “mate” o programa Turbo Pascal; neste caso, você perderá as alterações feitas no seu programa fonte que ainda não foram salvas (a menos que você tenha marcado a opção AutoSave).