

- Apostila -

Introdução à Linguagem de Programação Turbo Pascal

por

Leandro Krug Wives

2005

AGRADECIMENTOS

Agradeço aos meus alunos, passados, presentes e futuros, pois sem eles este livro não faz sentido. Ao colega Alexandre de Oliveira Zamberlam, pela utilização do conteúdo, enquanto esboço, em suas aulas e pelas dicas, críticas e contribuições que fez. Aos demais colegas do Grupo de Pesquisa em Tecnologia da Informação, cujos momentos e discussões são sempre fonte de inspiração para a geração e a elaboração de novos conhecimentos.

SUMÁRIO

INTRODUÇÃO	5
1 ESTRUTURA DE UM PROGRAMA EM PASCAL.....	7
2 TIPOS ABSTRATOS DE DADOS	9
2.1 Abstração.....	9
2.2 Tipos abstratos de dados.....	10
3 TIPOS DE DADOS PRIMITIVOS	13
3.1 Exercícios	15
4 OPERADORES.....	16
4.1 Operadores aritméticos.....	16
4.2 Operadores lógicos.....	17
4.3 Operadores relacionais.....	19
4.4 Operadores diversos (outros operadores).....	19
5 COMANDOS E FUNÇÕES DE ENTRADA E SAÍDA DE DADOS (E/S).....	20
5.1 Comandos básicos de Saída	20
5.1.1 Comando Write	20
5.1.2 Comando WriteLn	21
5.2 Comandos e funções básicos de Entrada	22
5.2.1 Comando ReadLn	22
5.2.2 Comando Read.....	22
5.2.3 Função ReadKey	23
5.2.4 Função KeyPressed.....	23
6 ALGUNS COMANDOS E FUNÇÕES INTERESSANTES.....	24
6.1 Funções aritméticas	24
6.2 Comandos de manipulação de tela (utilizar a biblioteca CRT)	25
6.3 Comandos e funções de manipulação de <i>strings</i>	26
6.4 Outros comandos e funções	27
6.5 Exercícios	28

7	COMANDOS DE DESVIO CONDICIONAL	30
7.1	O Comando IF	30
7.1.1	IF ... Then	30
7.1.2	IF ... Then ... Else	32
7.2	O Comando CASE	34
7.3	Exercícios	36
8	COMANDOS DE REPETIÇÃO.....	37
8.1	O Comando 'FOR'.....	37
8.2	O comando 'REPEAT' ... 'UNTIL'	38
8.3	O comando 'WHILE'	39
8.4	Exercícios	40
9	ARRAYS (ARRANJOS OU VETORES).....	42
9.1	Exercícios	45
10	PROCEDIMENTOS E FUNÇÕES.....	46
10.1	Procedimentos	46
10.1.1	Como declarar e criar procedimentos:	47
10.2	Funções.....	51
10.2.1	Como declarar e criar funções	51
10.3	Exercícios	52
11	REGISTROS (ESTRUTURAS) DE DADOS	54
12	TRABALHANDO COM ARQUIVOS	57
12.1	Criando referências (apelidos) para um arquivo.....	57
12.2	Criando e abrindo arquivos	58
12.3	Movimentando-se em um arquivo	59
12.4	Gravando um registro	59
12.5	Lendo um registro	60
12.6	Fechando um arquivo	60
13	TABELA ASCII	61
13.1	Programa-exemplo que imprime a tabela ASCII na tela do seu computador	64
	RESUMO DOS PRINCIPAIS OPERADORES, FUNÇÕES E PROCEDIMENTOS DA LINGUAGEM TURBO PASCAL.....	65
	BIBLIOGRAFIA E LEITURA RECOMENDADA	69

INTRODUÇÃO

A *Linguagem Pascal* foi desenvolvida no início da década de 70 por um professor da Technical University (ETH) de Zurich, na Suíça. Ele se chamava Niklaus Wirth, e projetou esta linguagem a fim de facilitar o aprendizado e o ensino de programação para o curso de computação (CARROW, 1988).

Pascal é uma linguagem baseada no paradigma de *Programação Estruturada*. Esse paradigma foi desenvolvido na década de 60 por E. Dijkstra e C. Haare com o objetivo de melhorar a produtividade do programador e reduzir erros de desenvolvimento. Nele, o programa é estruturado (dividido) em vários módulos e os dados são organizados em estruturas (as *Estruturas de Dados*) a fim de facilitar seu armazenamento e processamento (CARROW, 1988).

Logo, a idéia principal desse paradigma consiste em dividir o programa em diversas e pequenas partes que solucionam cada uma delas uma parte de um problema maior e mais complexo. Cada uma dessas partes (ou módulos) deve ser construída da forma mais independente possível. Os programas são compostos de sentenças que representam comandos ou expressões lógicas ou matemáticas. Estas sentenças devem formar parágrafos que demonstrem rapidamente sua intenção (interação, decisão ou composição) e facilitem a compreensão e a verificação do programa. Recomenda-se também que os módulos não sejam muito grandes, mas, sim, subdivididos e compostos de outros sub-módulos.

Com o tempo, a linguagem Pascal superou as expectativas de seu criador, tornando-se muito popular e poderosa. Atualmente, ela não é mais somente uma linguagem de cunho acadêmico com o objetivo de instruir novos programadores. Um bom exemplo é a

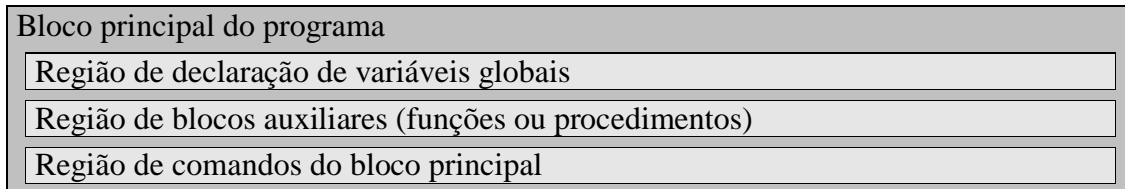
linguagem *Object Pascal* utilizada no ambiente *Delphi* da Borland¹. O Object Pascal é uma variação da linguagem Delphi desenvolvida por Wirth, que introduz diversos recursos, tais como *Orientação a Objetos* (um paradigma de programação mais atual), *eventos* e *componentes de programação visual*, que facilitam muito o desenvolvimento de softwares comerciais e de qualidade.

O objetivo deste livro é demonstrar a você os conceitos, as estruturas básicas e a sintaxe dos comandos disponíveis na linguagem Pascal. Primeiramente será dada uma visão geral de um programa escrito em Pascal e como realizar a abstração de um problema do mundo real para o mundo dos computadores. Após, serão listados e detalhados os tipos de dados (variáveis) que a linguagem padrão oferece, os principais operadores e funções de manipulação dos mesmos, além de uma introdução aos comandos básicos de entrada e saída de dados (E/S), de seleção ou desvio condicional e de repetição. Em seguida, são apresentados os *arrays* ou *vetores*, os *procedimentos*, as *funções* e os *Registros*. Tendo-se uma noção de algoritmos, estes capítulos apresentam tudo o que você precisa para iniciar a fazer programas com esta linguagem.

¹ Site da Borland: <http://www.borland.com>.

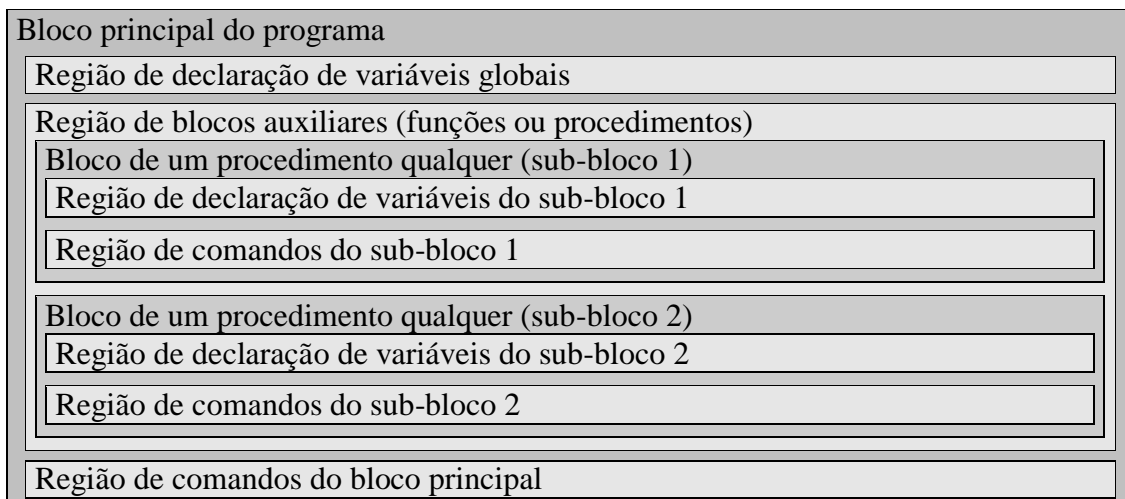
1 ESTRUTURA DE UM PROGRAMA EM PASCAL

Os programas escritos na linguagem Pascal possuem a seguinte estrutura:



Todo programa em Pascal é escrito dentro do bloco principal. Esse bloco possui três sub-regiões principais: região de declaração de variáveis globais, região de blocos auxiliares (sub-blocos correspondentes a funções e/ou procedimentos) e região de comandos do bloco principal.

Cada bloco (sub-bloco) também possui regiões específicas onde se pode declarar o nome do bloco, suas variáveis e seus comandos. O exemplo seguinte demonstra como a região de blocos foi expandida para refletir a inclusão de dois sub-blocos, cada um com suas sub-regiões específicas de variáveis (locais) e comandos:



As variáveis declaradas no bloco principal são ditas globais (escopo global) porque podem ser manipuladas e vistas em todo o programa.

Já as variáveis de um sub-bloco possuem um escopo local, e só podem ser vistas e manipuladas dentro do bloco no qual foram declaradas. Portanto, o sub-bloco 1 só pode enxergar as variáveis do seu bloco e as variáveis globais. Ele não pode enxergar as variáveis do sub-bloco 2.

Inicialmente, para facilitar o entendimento, trabalhar-se-á com programas que só possuem o bloco principal (sem sub-blocos).

A sintaxe de linguagem que permite a definição/construção do bloco principal e suas regiões é a seguinte:

```
Program Exemplo;  
  
Var   idade      : Integer ;  
      total_dias : Integer ;  
  
Begin  
      idade := 23 ;  
      total_dias := idade *365 ;  
End.
```

O programa sempre inicia com a palavra-chave ‘Program’ seguida do *nome do programa* e um *ponto-e-vírgula*. Assim, o compilador sabe que o bloco principal do programa iniciou.

A seguir vem a região de declaração de variáveis (globais). O compilador² sabe que essa região começou quando encontra a palavra-chave ‘Var’. Dentro dessa região são declaradas as variáveis.

Finalmente, encontra-se a região de comandos. Essa região inicia com a palavra-chave ‘Begin’ e termina com a palavra-chave ‘End’ seguida de um *ponto-final*.

² **Compilador** é o componente do ambiente que analisa seu programa em Pascal e o traduz para a linguagem de máquina.

2 TIPOS ABSTRATOS DE DADOS

Tipos abstratos de dados são estruturas especificamente construídas para armazenarem determinados tipos de dados. Estas estruturas especificam operadores que permitirem a manipulação desses dados, oferecendo recursos para que cada tipo possa ser processado junto com outros e até mesmo convertido.

Para compreender melhor essa idéia, vamos primeiro dar uma olhada no que significa *abstração*. Depois, discutiremos novamente a definição dos tipos abstratos de dados e qual sua relação com os tipos primitivos de dados oferecidos pela linguagem Pascal.

2.1 Abstração

Vamos dar uma olhada em uma *definição de abstração*:

- Ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora dessa totalidade (Dicionário Aurélio Eletrônico do Século XXI).

Abstrair tem um sentido muito semelhante aos de *generalizar* ou *determinar*:

- *Generalizar* (conforme o Dicionário Aurélio Eletrônico do século XXI):
 - Extensão de um princípio ou de um conceito a todos os casos a que se pode aplicar;
 - Processo pelo qual se reconhecem caracteres (características) comuns a vários objetos singulares, daí resultando quer a formação de um novo conceito ou idéia, quer o aumento da extensão de um conceito já determinado que passa a cobrir uma nova classe de exemplos
- *Determinar* (conforme o Dicionário Aurélio Eletrônico do século XXI):

- Especificação de características que distinguem um conceito de outro do mesmo gênero, aumentando-lhe a compreensão (conforme Aurélio Eletrônico do século XXI);
- Característica que serve a determinação, tal como uma qualidade, um atributo etc.

A abstração permite com que uma pessoa possa pensar sobre como se dirige um carro qualquer sem a necessidade de que alguém lhe especifique uma marca ou modelo (pois todos os carros são dirigidos basicamente da mesma forma e possuem funcionamento similar).

Logo, a abstração pode ser utilizada para suprimir os detalhes irrelevantes e, ao mesmo tempo, enfatizar os relevantes, facilitando o trabalho de pensar sobre determinado problema a ser resolvido (PREISS, 2001).

2.2 Tipos abstratos de dados

Toda variável de uma linguagem de programação é uma abstração, pois representa virtualmente na memória do computador determinado objeto que existe no mundo real.

Ela possui algumas características que a auxiliam a representar da melhor forma possível o objeto em questão. Essas características ou atributos são (PREISS, 2001):

- **Nome:** nome da variável, normalmente especificado pelo programador;
- **Valor ou conteúdo:** uma representação do objeto que ela armazena (geralmente em código binário);
- **Tipo:** especifica o conjunto de valores que a variável pode assumir, isto é, denota quais valores podem ser atribuídos ou armazenados no atributo *valor* da variável, descrito acima.
- **Escopo:** o contexto ou local de validade de uma variável. A variável somente é visível, isto é, só pode ser manipulada em determinados trechos do programa. O que varia é esse trecho, que pode ser todo o programa (variável global) ou determinadas rotinas ou blocos (variável local). Somente os comandos pertencentes a esses trechos (o escopo) podem ver a variável.

- **Endereço:** toda variável é armazenada em algum lugar da memória do computador. Este lugar possui um índice, um endereço para que saibamos onde olhar caso desejemos manipulá-la;
- **Tamanho:** a quantidade de memória que a variável ocupa ou necessita para armazenar seu valor. Este valor é geralmente atrelado ao tipo da variável, ou seja, cada tipo de variável possui um determinado tamanho de memória que é fixo, independente do seu valor;
- **Duração:** o intervalo de tempo em que a variável existe durante a execução do programa (algumas podem valer durante toda a execução do programa, outras somente durante a execução de alguma tarefa – bloco ou procedimento).

Disso tudo, o que você necessita saber é quais são os *tipos* de dados que a linguagem Pascal oferece, qual o *tamanho* ocupado por cada um destes tipos e como você pode atribuir um *nome* a uma variável. A idéia de *escopo*, que é o local de validade de uma variável, também é importante compreender.

O *endereço* de uma variável geralmente é definido pelo *compilador*, e, na maioria das linguagens, não pode ser definido ou facilmente modificado. Portanto, não necessário se preocupar com isso no momento. Porém, em programas mais complexos, é necessário conhecer a localização de determinadas variáveis ou dados na memória, e o Pascal, assim como a grande maioria das linguagens, oferece mecanismos para se obter essa localização (*endereço*).

Alguns desses atributos são (e devem ser) definidos no momento que o programador escreve o programa: é o caso do nome, do tipo e do escopo de uma variável. Nesse caso, diz-se que essa definição é *estática*, pois não muda. Já outros são (ou podem ser) definidos no momento em que o programa é executado. Neste caso diz-se que a definição destes atributos é *dinâmica*, pois muda ou pode mudar a cada execução ou momento de execução (é o caso do *endereço* e do *valor* de uma variável).

Os atributos estáticos de uma variável são definidos na programação, e a esse ato dá-se o nome de *declaração*. *Declarar uma variável* significa especificar como a variável se chamará no código-fonte do programa que você está desenvolvendo e definir qual o tipo

dos dados que ela pode armazenar e manipular. Cada linguagem possui uma forma de se fazer isso.

Vamos agora estudar quais são os tipos de dados que a linguagem Pascal oferece. Estes tipos de dados que já vêm definidos na linguagem são chamados de tipos primitivos. Através deles o programador pode definir e criar outros tipos ou estruturas de dados derivados. Isto será visto mais adiante (no Capítulo 11, que aborda *registros*).

3 TIPOS DE DADOS PRIMITIVOS

A linguagem Pascal oferece seis tipos de dados abstratos³ primitivos. Para cada um desses tipos, há um limite de valores que eles podem assumir e um conjunto de operadores que os manipulam ou permitem que eles sejam convertidos ou utilizados em conjunto com os outros tipos.

Os tipos que a linguagem oferece são: INTEGER, BYTE, REAL, BOOLEAN, CHAR e STRING. Desses, os tipos BYTE e STRING não constavam na definição inicial do Pascal e podem não ser encontrados em alguns compiladores.

- **Integer:** armazenam números inteiros (naturais) cujos valores estejam entre $-(MAXINT+1)$ e $MAXINT$. $MAXINT$ é uma constante predefinida que pode variar de sistema para sistema ou de compilador para compilador. No ambiente Turbo Pascal, por exemplo, $MAXINT$ vale 32768. Isto significa que os números neste compilador variam entre -32767 e 32768, ocupando dois bytes (16 bits, que geram 65536 combinações possíveis de armazenamento);
- **Byte:** subconjunto do tipo *Integer* que armazena valores entre 0 e 255. Este tipo ocupa 1 byte (8 bits, que geram 256 combinações);
- **Real:** esse tipo armazena números reais positivos e negativos (incluindo frações). No turbo Pascal este tipo ocupa 6 bytes de memória (48 bits), mas o número não é armazenado como nos outros tipos. Neste caso, o número é armazenado num formato conhecido por formato científico, onde ele é

³ Um tipo é dito abstrato quando sua faixa de valores e seus operadores (ou relações) com os outros tipos (ao menos primitivos) são definidos. Desta forma, podemos, através do tipo, pensar em todas as suas propriedades sem que ele (o objeto) realmente exista. Podemos, assim, usá-los ou raciocinar sobre eles sem precisarmos saber como eles e suas operações são realmente implementados no computador (PREISS, 2000).

dividido em duas partes: a *mantissa* e o *expoente*. A mantissa contém números significativos do número real automaticamente normalizadas pelo computador para um valor fracionário na faixa entre 1 e 10. O expoente é um valor inteiro positivo ou negativo que indica a posição da vírgula no número. Para calcular o número armazenado, deve-se aplicar a seguinte fórmula (isso é feito automaticamente pelo computador):

$$\text{número} = \text{mantissa} * 10^{\text{expoente}}$$

Veja o exemplo de alguns números:

$$10.40 = 0.1045 * 10^{+2} \text{ ou } 1.0450000000\text{E}+01$$

$$0.00056993 = 0.56993 * 10^{-3} \text{ ou } 5.6993000000\text{E}-04$$

- Esse tipo de representação pode ocasionar alguns erros ou perdas, principalmente quando se está manipulando números com muitas casas decimais ou muito grandes, pois se o número de casas de uma soma, por exemplo, ultrapassar o número máximo de casas que o computador usa para armazenar o número, ele será arredondado;
- **Char:** abreviação da palavra inglesa “character”, que significa caractere. Como o próprio nome indica, serve para armazenar um único caractere (uma letra, dígito ou símbolo). Ocupa 1 byte de memória, o que significa que pode armazenar 256 combinações de bits. Esse é exatamente o tamanho da tabela ASCII (que significa *American Standard Code for Information Interchange* ou tabela Americana de Códigos Padrão para a Troca de Informações), que contém uma lista de 256 caracteres que variam entre caracteres de controle, letras, dígitos e símbolos. Cada um deles possui um código (um número) fixo. Através do número é possível descobrir o caractere correspondente na tabela e vice-versa;
- **String:** Armazena cadeias de caracteres. No fundo é o mesmo que um vetor de Chars, cujo tamanho máximo é o de 255 caracteres. Possui um byte (na posição 0) que indica quantas letras a string possui armazenada dentro dela;
- **Boolean:** armazena valores lógicos que variam entre “true” (verdadeiro) ou “false” (falso). São muito utilizadas em variáveis do tipo *flag* e em testes condicionais.

3.1 Exercícios

1. Leia com atenção esse capítulo e defina com as suas palavras o que é abstração. Dê um exemplo que não conste no texto, construído a partir do que você entendeu e conhece.
2. Cite os tipos de dados abstratos e primitivos da linguagem Pascal e tente identificar exemplos de dados e informações para os quais eles seriam mais adequados (qual seria, por exemplo, o tipo de dado mais indicado para armazenar um número telefônico, um CPF, um RG, um CEP?) e por quê?

4 OPERADORES

Neste capítulo você vai aprender os principais operadores da linguagem Turbo Pascal. Eles podem ser divididos em: aritméticos, lógicos, relacionais e operadores diversos.

4.1 Operadores aritméticos

São utilizados na realização de operações matemáticas.

+	<u>Soma</u> dois números e devolve o resultado no mesmo tipo dos operandos. Se eles forem de tipos diferentes, o resultado é do tipo mais complexo.
-	<u>Subtrai</u> dois números e devolve o resultado no mesmo tipo dos operandos. Se eles forem de tipos diferentes, o resultado é do tipo mais complexo.
*	<u>Multiplica</u> dois números e devolve o resultado no mesmo tipo dos operandos. Se eles forem de tipos diferentes, o resultado é do tipo mais complexo.
/	<u>Divide</u> dois números reais e devolve um número real como resultado.
div	<u>Divide</u> dois números inteiros e devolve um número inteiro correspondente ao quociente da divisão.
mod	<u>Divide</u> dois números inteiros e devolve um número inteiro correspondente ao resto da divisão.

Programa exemplo:

```
Program Exemplo_Operadores_1;  
Uses CRT;  
Var Dividendo, Divisor, Resultado1, Resto : Integer;  
    Resultado2: Real;  
Begin
```

```

Clrscr;
Num1 := 10; Num2 := 3;
Resultado1 := num1 div num2; { resultado inteiro da divisão: 3 }
Resultado2 := num1 / num2;    { resultado real da divisão: 3.33 }
Resto      := num1 mod num2; { resto da divisão inteira: 1      }
WriteLn('10 div 3 = ', resultado1);
WriteLn('10 mod 3 = ', resto);
WriteLn('10 / 3   = ', resultado2:5:2);
WriteLn('10 + 3 = ', 10 + 3); { soma e imprime o resultado }
WriteLn('10 - 3 = ', 10 - 3); { subtrai e imprime o resultado }
WriteLn('Pressione [ENTER] para sair...');
ReadLn;
End.

```

4.2 Operadores lógicos

São utilizados em expressões lógicas, normalmente utilizadas em comandos de seleção ou de repetição condicional (ver Capítulos 7 e 8).

NOT	<u>Negação, não.</u> Inverte o resultado de uma expressão booleana.																
	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>valor</th> <th>resultado após operador NOT</th> </tr> </thead> <tbody> <tr> <td>True</td> <td>False</td> </tr> <tr> <td>False</td> <td>True</td> </tr> </tbody> </table>		valor	resultado após operador NOT	True	False	False	True									
valor	resultado após operador NOT																
True	False																
False	True																
AND	<u>E.</u> Retorna verdadeiro se os dois operandos (dois lados) forem verdadeiros.																
	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>valor esquerdo</th> <th>valor direito</th> <th>resultado após operador AND</th> </tr> </thead> <tbody> <tr> <td>False</td> <td>False</td> <td>False</td> </tr> <tr> <td>True</td> <td>False</td> <td>False</td> </tr> <tr> <td>False</td> <td>True</td> <td>False</td> </tr> <tr> <td>True</td> <td>True</td> <td>True</td> </tr> </tbody> </table>		valor esquerdo	valor direito	resultado após operador AND	False	False	False	True	False	False	False	True	False	True	True	True
	valor esquerdo	valor direito	resultado após operador AND														
	False	False	False														
	True	False	False														
False	True	False															
True	True	True															

OR	<u>OU</u> . Retorna verdadeiro se qualquer um dos lados da expressão booleana for verdadeiro.															
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">valor esquerdo</th> <th style="text-align: center;">valor direito</th> <th style="text-align: center;">resultado após operador OR</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">False</td> <td style="text-align: center;">False</td> <td style="text-align: center;">False</td> </tr> <tr> <td style="text-align: center;">True</td> <td style="text-align: center;">False</td> <td style="text-align: center;">True</td> </tr> <tr> <td style="text-align: center;">False</td> <td style="text-align: center;">True</td> <td style="text-align: center;">True</td> </tr> <tr> <td style="text-align: center;">True</td> <td style="text-align: center;">True</td> <td style="text-align: center;">True</td> </tr> </tbody> </table>	valor esquerdo	valor direito	resultado após operador OR	False	False	False	True	False	True	False	True	True	True	True	True
	valor esquerdo	valor direito	resultado após operador OR													
	False	False	False													
	True	False	True													
False	True	True														
True	True	True														
XOR	<u>OU exclusivo</u> . Retorna verdadeiro se somente um dos lados da expressão for verdadeiro (nem todos os compiladores implementam este operador).															
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">valor esquerdo</th> <th style="text-align: center;">valor direito</th> <th style="text-align: center;">resultado após operador XOR</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">False</td> <td style="text-align: center;">False</td> <td style="text-align: center;">False</td> </tr> <tr> <td style="text-align: center;">True</td> <td style="text-align: center;">False</td> <td style="text-align: center;">True</td> </tr> <tr> <td style="text-align: center;">False</td> <td style="text-align: center;">True</td> <td style="text-align: center;">True</td> </tr> <tr> <td style="text-align: center;">True</td> <td style="text-align: center;">True</td> <td style="text-align: center;">False</td> </tr> </tbody> </table>	valor esquerdo	valor direito	resultado após operador XOR	False	False	False	True	False	True	False	True	True	True	True	False
	valor esquerdo	valor direito	resultado após operador XOR													
	False	False	False													
	True	False	True													
False	True	True														
True	True	False														

Exemplo:

```

Program Exemplo_Operadores_2;
Uses CRT;
Var a, b : Boolean;
Begin
  Clrscr;
  A := True; { atribui verdadeiro à variável }
  B := False; { atribui falso à variável }
  WriteLn('A = ', A); WriteLn('B = ', B);
  WriteLn('Inverso de A = ', NOT A);
  WriteLn('Inverso de B = ', NOT B);
  WriteLn('A e B = ', A AND B);
  WriteLn('A ou B = ', A OR B);
  WriteLn('A ou exclusivo B = ', A XOR B);
  WriteLn('A ou o inverso de B = ', A OR NOT B);
  Writeln('Pressione [ENTER] para sair...');
  ReadLn;
End.

```

4.3 Operadores relacionais

Também são utilizados em expressões lógicas e indicados para os comandos de seleção e de repetição.

=	Igualdade
<>	Diferença
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

Exemplo:

```
Program Exemplo_Operadores_3;
Uses CRT;
Var a, b: Integer;
Begin
  Clrscr;
  A := 10; B := 30;
  WriteLn('A = ', A);   WriteLn('B = ', B);
  WriteLn('B <= A ? ', B <= A);
  WriteLn('A > B ? ', A > B);
  WriteLn('A = B ? ', A = B);
  WriteLn('A <> B ? ', A <> B);
  WriteLn('Pressione [ENTER] para sair...');
  ReadLn;
End.
```

4.4 Operadores diversos (outros operadores)

+	concatena (junta) duas strings.
:=	atribuição

5 COMANDOS E FUNÇÕES DE ENTRADA E SAÍDA DE DADOS (E/S)

Neste capítulo você vai aprender quais são os comandos que a linguagem Turbo Pascal oferece para realizar a saída de dados (ou seja, para mostrar dados na tela do computador) e a entrada de dados (ler dados a partir de um dispositivo de entrada, tal como o teclado).

5.1 Comandos básicos de Saída

Comandos de saída são aqueles que permitem ao programador enviar dados para dispositivos periféricos de saída de dados (o monitor e a impressora, por exemplo) ou de armazenamento (tais como disquetes e discos rígidos).

5.1.1 Comando Write

O Comando Write permite ao programador escrever alguma coisa na tela (ou outro dispositivo qualquer de saída).

Sintaxe⁴:

```
Write(<string> | <variável> [, <string> | <variável>]*);
```

Onde:

<string> é uma seqüência de caracteres quaisquer, entre apóstrofes.

<variável> é uma variável qualquer (no caso, será impresso o conteúdo da variável).

⁴ **Sintaxe** é a forma de utilização de um comando, operação ou instrução. Normalmente é descrita através de uma gramática simples, onde os elementos entre colchetes são opcionais e os que possuem um asterisco

Exemplo:

```
Program Exemplo_Saida_Dados;
Var a, b: Integer;
    c, d: Real;
Begin
    write('Olá, como vai?');
    a := 10; b:= 20;
    c := 20.567; d := 23;
    write(a);
    write('A variável b vale: ', b);
    write('c: ', c, ' d: ', d);
    write('c + d = ', c + d);
End.
```

Note que você pode especificar o tamanho mínimo do que está sendo impresso colocando o sinal : após o que você quer imprimir. Isto é extremamente útil em variáveis do tipo Real, cujo padrão de impressão é o formato científico. Para mudá-lo, especifique o tamanho máximo em dígitos do número desejado e, após, o tamanho da mantissa.

Veja o exemplo:

```
Program Exemplo;
Var d: Real;
Begin
    d := 10.5;      { Atribui o valor 10.5 à variável d }
    { O comando seguinte mostra o conteúdo da variável d com 5
      dígitos (incluindo o ponto), sendo dois deles
      após o ponto }
    write(d:5:2);  { mostrará 10.50 na tela }
End.
```

5.1.2 Comando Writeln

Similar ao anterior, porém, muda de linha após imprimir.

podem ser repetidos mais de uma vez. O sinal '[' significa 'ou' e os que estão entre os sinais de menor e maior '<>' devem ser substituídos. Os demais devem ser escritos exatamente na forma em que aparecem.

5.2 Comandos e funções básicos de Entrada

Comandos de entrada são aqueles que permitem ao programador coletar dados a partir de dispositivos periféricos (geralmente o teclado) ou de armazenamento (tais como disquetes e discos rígidos).

5.2.1 Comando Readln

O comando *Readln* permite ao programador solicitar que o usuário informe algum dado para ser colocado em uma variável. Após ler o dado, o cursor passa para o início da próxima linha.

Sintaxe:

```
Readln(<variável>);
```

Onde:

<variável> é uma variável qualquer onde os dados lidos serão colocados.

Exemplo:

```
Program Exemplo_Entrada_Dados;
Var a: Integer;
    b: Real;
Begin
    write('Digite um número inteiro: ');
    readln(a);
    write('Digite um número real: ');
    readln(b);
    write('a / b = ', (a / b):5:2);
End.
```

5.2.2 Comando Read

Similar ao anterior, porém, não muda de linha após ler. Além disso, normalmente o [ENTER] pressionado pelo usuário não é retirado do *buffer* do teclado e a próxima leitura pode não ser realizada corretamente.

5.2.3 Função ReadKey

ReadKey (do Inglês "ler tecla") é uma função que espera o usuário digitar uma única tecla (qualquer) e devolve-a para o programador. Este, geralmente coloca-a em uma variável do tipo Char. Para usá-la você precisa incluir a biblioteca CRT.

Exemplo:

```
Program Exemplo;  
Uses CRT;  
Var tecla: Char;  
Begin  
    write('pressione qualquer tecla!');  
    tecla := ReadKey;  
    write('A tecla digitada foi: ', tecla,  
        ' seu código ASCII é: ', ord(tecla));  
End.
```

5.2.4 Função KeyPressed

KeyPressed (do Inglês "tecla pressionada") é uma função que devolve *true* (verdadeiro) se o usuário pressionou uma tecla e *false* (falso) se ele não a pressionou. Essa função não espera o usuário digitar uma tecla. Ela somente informa se o usuário pressionou uma no momento em que foi chamada. Esta função também necessita da biblioteca CRT.

Exemplo:

```
Program Exemplo;  
Uses CRT;  
Var tecla: Char;  
Begin  
    Repeat  
        writeln('Pressione uma tecla!');  
    Until keypressed; { repete até que seja pressionada uma  
                        tecla! }  
End.
```

6 ALGUNS COMANDOS E FUNÇÕES INTERESSANTES

Neste capítulo são apresentadas algumas funções e comandos que podem ser utilizados para converter dados, preparar a tela e realizar outras tarefas úteis.

6.1 Funções aritméticas

Estas funções realizam transformações e operações matemáticas.

int(x)	Retorna a parte inteira de um número Real (a que vem antes da vírgula). O resultado é um número real.
frac(x)	Retorna a parte fracionária de um número Real (a que vem depois da vírgula). O resultado é um número real.
abs(y)	Retorna o valor absoluto de um número inteiro (ou seja, o número sem seu sinal).
sqr(y)	Retorna o quadrado de um número.
sqrt(y)	Retorna a raiz quadrada de um número.
round(x)	Retorna a parte inteira de um número, arredondando-o. O resultado é um número inteiro.
trunc(x)	Retorna a parte inteira de um número Real, sem arredondá-lo. O resultado é semelhante ao da função int, com exceção que o resultado é um número inteiro.
cos(x)	Retorna o co-seno de um número (em radianos).
sin(x)	Retorna o seno de um número (em radianos).
exp(x)	Retorna o número e (2.71....) elevado ao valor solicitado.
Ln(x)	Retorna o logaritmo natural de x.

Exemplo:

```
Program Exemplo_Funcoes;  
Uses CRT;
```

```

Var a: Real;
Begin
  Clrscr;
  A := -10.6;
  WriteLn('A = ', A:5:2);
  WriteLn('INT(A) = ', INT(A):5:2); { resultado: -10.00 }
  WriteLn('FRAC(A) = ', FRAC(A):5:2); { resultado: -0.60 }
  WriteLn('ABS(A) = ', ABS(A):5:2); { resultado: 10.60 }
  WriteLn('ROUND(A) = ', ROUND(A)); { resultado: -11 }
  WriteLn('TRUNC(A) = ', TRUNC(A)); { resultado: -10 }
  Writeln('Pressione [ENTER] para sair...');
  ReadLn;
End.

```

6.2 Comandos de manipulação de tela (utilizar a biblioteca CRT)

Estes comandos são utilizados para mudar as cores da tela (texto e fundo), apagá-la e posicionar o cursor.

clrscr	Apaga a tela.
clreol	Apaga da posição atual do cursor até o final da linha.
delline	Apaga a linha onde está o cursor.
insline	Insere uma linha na posição atual do cursor.
gotoxy(c,l)	Posiciona o cursor na coluna (c) e linha (l) especificados.
textcolor(c)	Altera a cor do texto para a cor especificada (c).
textbackground(c)	Altera a cor de fundo do texto para a cor especificada (c).
Window(ci, li, cf, lf)	Define uma janela de texto nas coordenadas especificadas por ci (coluna inicial), li (linha inicial), cf (coluna final) e lf (linha final). O posicionamento do cursor, a partir deste comando, dá-se dentro das coordenadas especificadas. O valor normal é 1,1, 80, 25.
WhereX	Retorna a coluna atual do cursor.
WhereY	Retorna a linha atual do cursor.

Exemplo:

```

Program Exemplo_Cmd_Tela;
Uses CRT;
Begin

```

```

TextColor(YELLOW); { muda a cor do texto para Amarelo }
TextBackGround(BLUE); { muda a cor de fundo para Azul }
Clrscr; { apaga a tela nas cores especificadas }
GotoXY(10, 10); { move o cursor para a coluna 10, linha 10 }
WriteLn('Olá mundo!');
GotoXY(10, 11); { move cursor para coluna 10, linha 11 }
Write('Pressione [ENTER] para inserir uma linha acima...');
ReadLn;
GotoXY(10, 11);
Inline;
GotoXY(10, 13);
WriteLn('Pressione [ENTER] para sair...');
ReadLn;
End.

```

6.3 Comandos e funções de manipulação de *strings*

Estes comandos servem para você manipular *strings* (cadeias de caracteres).

Length(str)	Retorna o tamanho (número de caracteres) de uma string.
Insert(s1, s2, p)	Insere uma string (s1) em outra (s2), na posição especificada por <i>p</i> .
Copy(s, i, t)	Retorna uma parte de uma string, a partir da posição inicial (i), no tamanho especificado (t).
Delete(s, i, t)	Apaga parte de uma string, iniciando na posição desejada (i) e indo até o tamanho especificado (t)
Concat(s1, s2)	Retorna a concatenação de s1 + s2.
Pos(s1, s2)	Retorna a posição de s1 em s2.
Str(v, s)	Converte um número (v) para uma string (s).
UpCase(c)	Retorna a versão maiúscula do caractere especificado
Val(s, n, erro)	Converte uma string (s) para um número (n), se for possível. Se não for possível, erro é diferente de zero.

Exemplo:

```

Program Exemplo_Strings;
Uses CRT;
Var nome, sobrenome, nomeCompleto, nomeCitacoes: String;
    posEspaco : Integer;
Begin

```

```

Clrscr;
Write('Digite seu nome e sobrenome (Ex: fulano silva): ');
posEspaco := pos(' ', nomeCompleto);

{ seleciona o primeiro nome }
nome := copy(nomeCompleto, 1, posEspaco-1);

{ seleciona o sobrenome }
sobrenome := copy(nomeCompleto, posEspaco+1,
                  length(nomeCompleto) - (posEspaco));

nomeCitacoes := sobrenome + ', ' + nome;

WriteLn('Abreviado: ', nome[1], '. ', sobrenome[1], '.');
WriteLn('Citações: ', nomeCitacoes);
Readln;

End.

```

6.4 Outros comandos e funções

random(x)	Retorna um número aleatório entre 0 e o número informado.
sizeof(var)	Retorna o tamanho em bytes de uma variável ou tipo de variável.
chr(n)	Retorna o caractere correspondente ao código (n) na tabela ASCII.
ord(c)	Retorna o código do caractere (c) na tabela ASCII.
Inc(x)	Incrementa x.
Dec(x)	Decrementa x.
Succ(x)	Retorna o sucessor de x.
Pred(x)	Retorna o predecessor de x.
Randomize	Inicializa o gerador de números aleatórios.
Sound(x)	Gera som na frequência especificada por x.
Nosound	Encerra a geração de som iniciada por sound.
Delay(x)	Cria um atraso (espera) de x milissegundos.
Halt	Sai do programa.
Exit	Sai do bloco atual ou vai para o final do bloco corrente.

Exemplo:

```

Program Exemplo_outros;
Uses CRT;
Var Num  : Integer;
    Tecla: Char;
Begin
    Clrscr;
    Num  := Random(10); { sorteia um número entre 0 e 10 }
    Tecla := 'A';
    WriteLn('Número sorteado: ', num);
    WriteLn('Conteúdo da variável "Tecla": ', tecla);
    WriteLn('Tamanho (em bytes) da variável "Tecla": ',
            SizeOf(tecla));
    WriteLn('Código do caractere ', Tecla, ' :', Ord(Tecla));
    WriteLn('Caractere cujo código é 78: ', Chr(78));
    WriteLn('Tamanho de uma variável "Integer": ', SizeOf(Integer));
    Writeln('Pressione [ENTER] para sair...');
    ReadLn;
End.

```

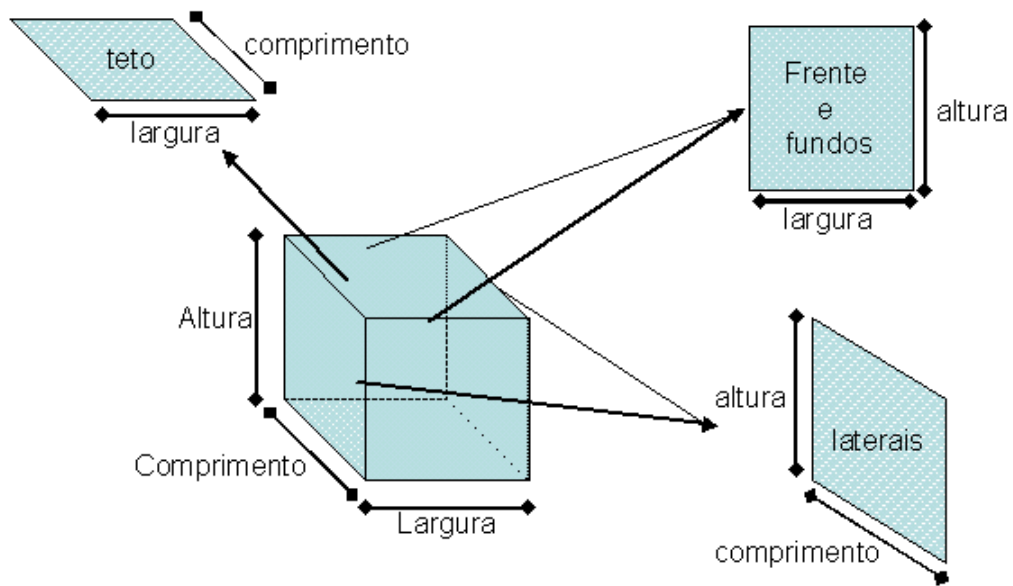
6.5 Exercícios

3. Um professor necessita calcular as notas de seus alunos. Ele ficou sabendo que você é bom programador e contratou você para implementar um programa que solicite três notas e informe a média aritmética delas.
4. Um investidor está preocupado com o cambio do dólar e necessita de um programa que o auxilie a calcular a variação cambial entre dois dias consecutivos. Implemente um programa que solicite dois valores de dólar (preço) e informe a diferença absoluta e a diferença relativa entre eles.

$$\text{dif_absoluta} = \text{val_ontem} - \text{val_hoje}$$

$$\text{dif_relativa} = (\text{dif_absoluta} * 100) / \text{val_ontem}$$
5. Faça um programa que solicite um número e após informe sua raiz quadrada e o seu quadrado.
6. Faça um programa leia uma palavra e informe quantas letras ela possui.
7. Ler a idade de uma pessoa e informar o número de dias e de meses que ela viveu.

8. Um pintor necessita saber quantas latas de tinta são necessárias para pintar determinada peça de uma casa e o custo delas. Desenvolva um programa que solicite a largura, o comprimento e a altura da peça. Após calcular a área a ser pintada (correspondente a quatro paredes e um teto), informe a quantidade de latas e o custo necessário para pintar a peça, levando em conta que cada lata de tinta custa R\$ 23.20, contém cinco litros de tinta e cada litro pinta 3m^2 . Ver figura seguinte para compreender as variáveis do programa e sua relação.



7 COMANDOS DE DESVIO CONDICIONAL

Neste capítulo são apresentados comandos que permitem ao programador desviar ou alterar o caminho de execução de um programa de acordo com o resultado de uma expressão lógica ou matemática, que pode ser dependente dos valores informados pelo usuário ou obtidos a partir de um dispositivo de entrada ou saída de dados.

7.1 O Comando IF

O comando IF tem duas possibilidades:

- IF ... Then (Se ... Então ...)
- IF ... Then ... Else (Se ... Então ... Senão ...)

7.1.1 IF ... Then

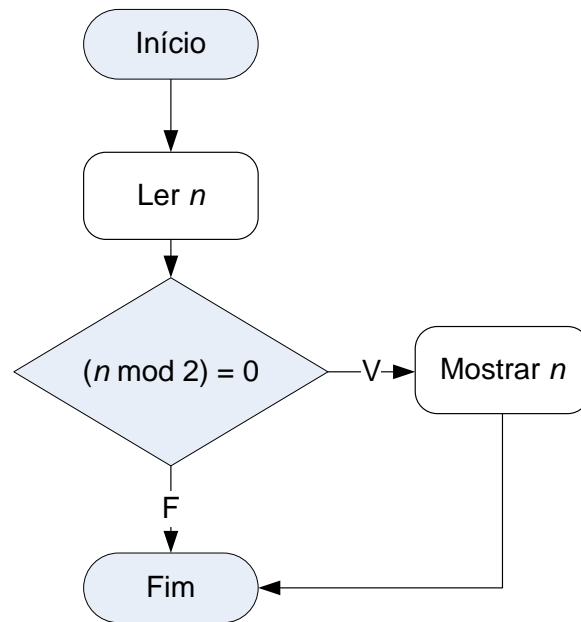
Sintaxe:


```
IF <condição_verdadeira> THEN <comando_a_ser_executado>;
```

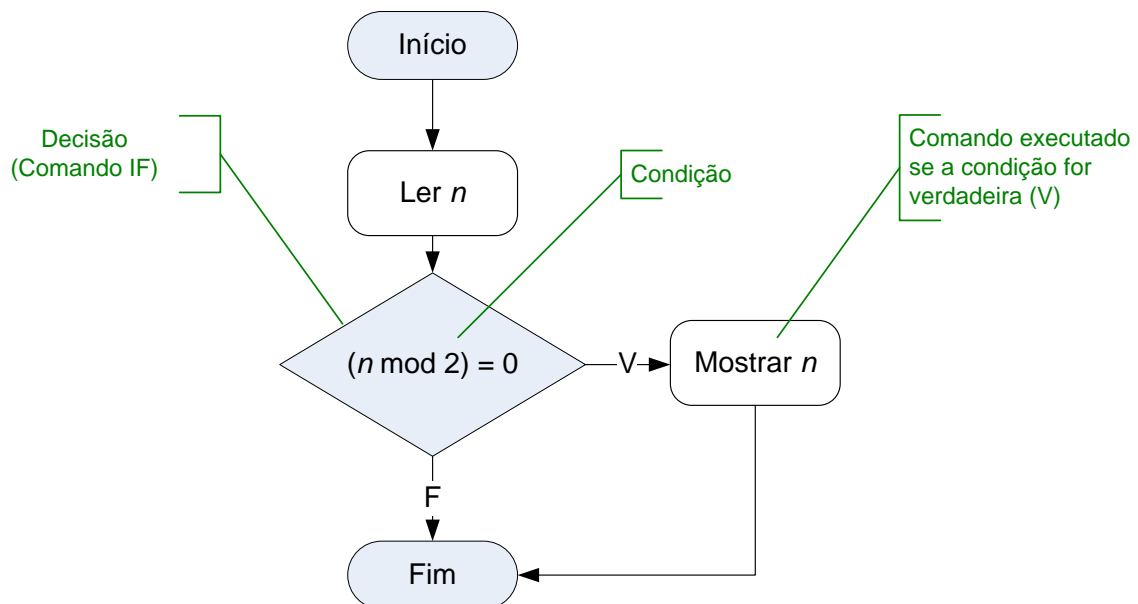
Em português:

```
SE <condição_verdadeira> ENTÃO <comando_a_ser_executado>;
```

Utilizado quando o programador necessita realizar um ou mais comandos caso determinada condição aconteça. Veja o exemplo na figura:



Este programa lê um número e se este número for par (divisível por 2) ele é mostrado na tela. O símbolo  representa um comando de decisão (no caso, o comando IF). Dentro dele vai a **condição** que deve ser testada. Caso a condição seja verdadeira (caso o resultado do teste seja verdadeiro), a execução do programa é desviada para o comando **Mostrar n**. Se ela for falsa a execução não sofre o desvio e o programa acaba.



Em pascal, este programa seria implementado da seguinte forma:

```
Program Testa_Par;
```

```

Var n: Integer;
Begin
  Readln(n);
  IF (n MOD 2) = 0 THEN Writeln(n);
End.

```

Se você deseja executar mais de um comando dentro do desvio, basta inseri-los dentro de um bloco. Um bloco é delimitado pelas palavras BEGIN e END:

```

IF (n MOD 2) = 0                { * Condição * }
THEN Begin                    { * início de bloco de comandos * }
  Clrscr;
  Writeln('o número digitado ', n, ' é par');
  Readln;
End;                        { * fim de bloco de comandos * }

```

Eles (os comandos dentro do bloco) só serão executados se a condição for verdadeira.

7.1.2 IF ... Then ... Else

Sintaxe:

```

IF <condição_verdadeira> THEN <comando_a_ser_executado>
ELSE <comando_a_ser_executado_se_condição_falsa>;

```

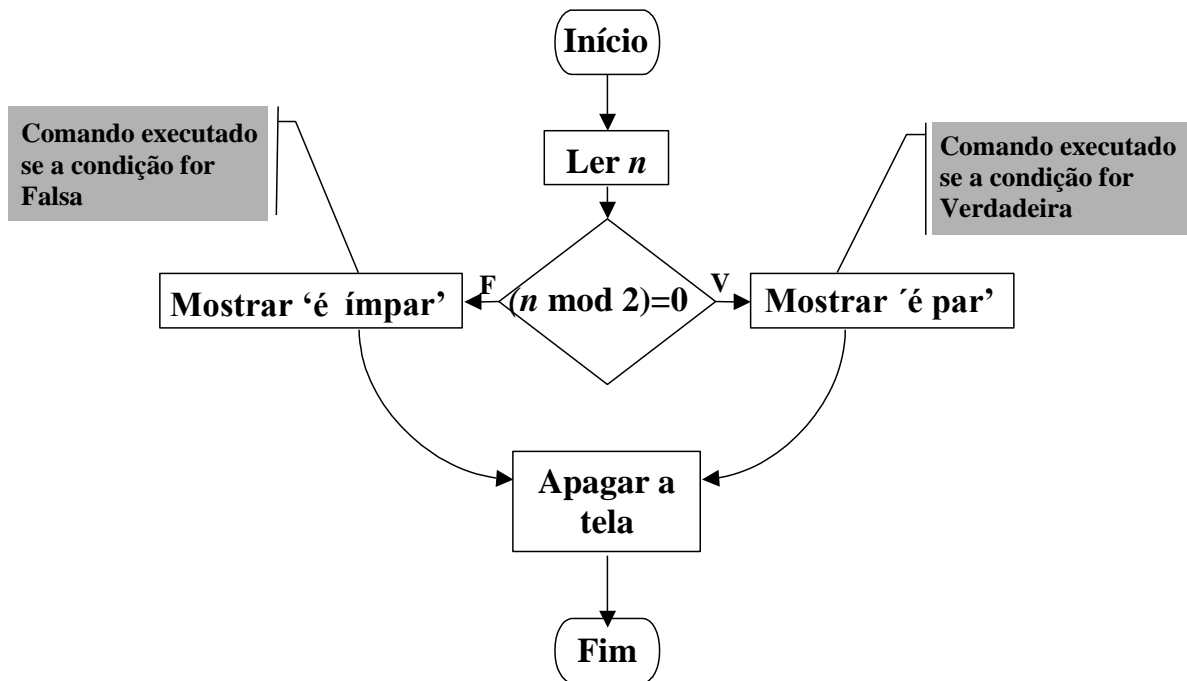
Em português:

```

SE <condição> Então <comando_a_ser_executado_se_condição_verdadeira>
SENÃO <comando_a_ser_executado_se_condição_falsa>;

```

Nesse caso o programador também pode estabelecer um caminho a ser seguido se a condição for falsa. Veja a figura:



Em pascal, esse programa seria implementado da seguinte forma:

```

Program Testa_Par;
Var n: Integer;
Begin
  Readln(n);
  IF (n MOD 2) = 0 THEN Writeln('é par') ELSE Writeln('é ímpar');
End.
  
```

Como no caso anterior, para cada lado (verdadeiro ou falso), pode-se utilizar blocos de comandos:

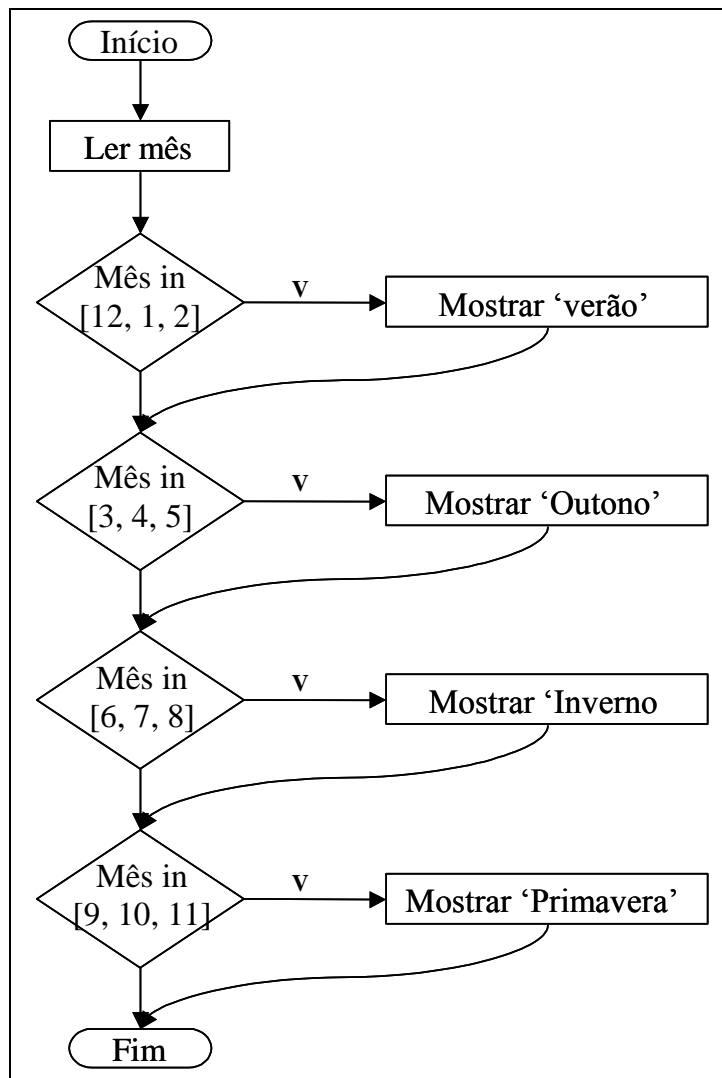
```

IF (n MOD 2) = 0      { * Condição * }
THEN Begin           { * início de bloco de comandos p/verdadeiro * }
  Clrscr;
  Writeln('o número digitado ', n, ' é par');
  Readln;
End                  { * fim do bloco - Não vai ponto-e-vírgula * }
ELSE Begin           { * início de bloco de comandos p/falso * }
  Clrscr;
  Writeln('o número digitado ', n, ' é par');
  
```

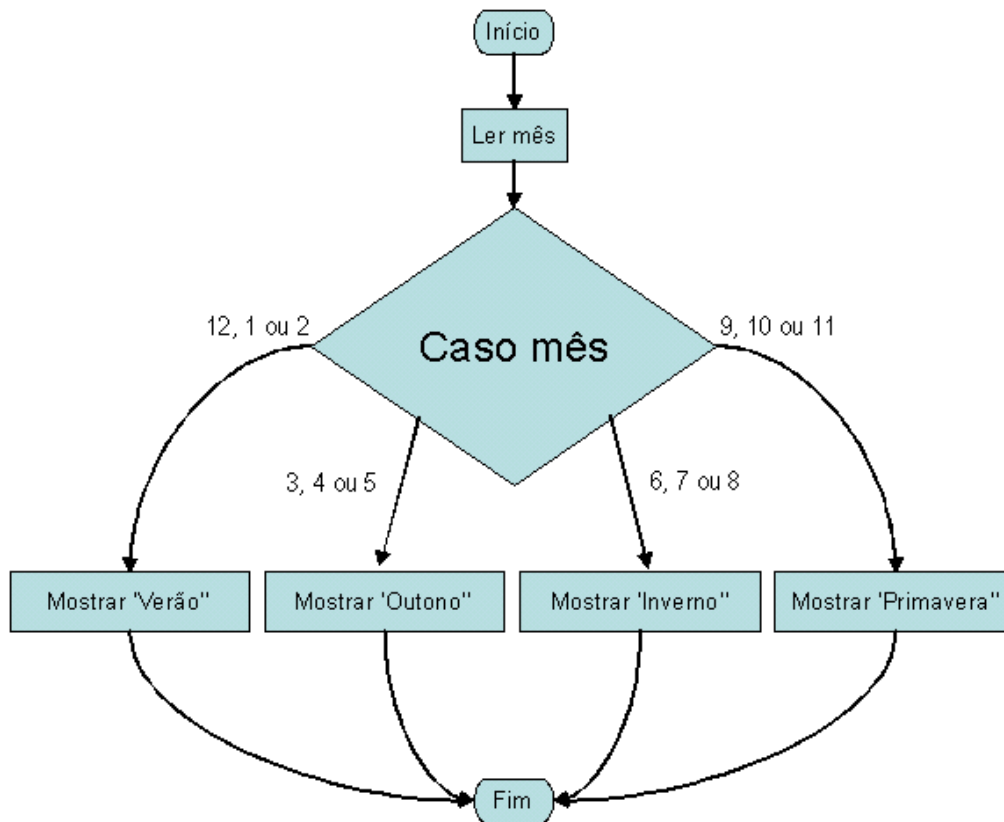
```
Readln;  
End;          { * fim de bloco de comandos * }
```

7.2 O Comando CASE

Imagine que você queira testar se determinado mês está na primavera, verão, outono ou no inverno. Logo, o teste já não é mais binário (verdadeiro ou falso), mas sim composto de 4 valores. Você poderia fazer diversos testes com o comando IF, um para cada intervalo de meses:



Se a condição ou expressão que você necessita testar pode resultar em mais de dois valores, isto é, se o teste não vai ser verdadeiro ou falso, mas sim um conjunto de valores, você pode utilizar o comando case:



Em Pascal:

```

Program Testa_Epoca;
Var mes: Integer;
Begin
  Readln(mes);
  CASE mes OF
    12, 1, 2: Writeln('Verão');    { * caso mês seja 12, 1 ou 2 * }
    3, 4, 5: Writeln('Outono');   { * caso mês seja 3, 4 ou 5 * }
    6, 7, 8: Writeln('Inverno');
    9, 10, 11: Writeln('Outono');
  END;
End.
  
```

Note que o comando Case só funciona para valores ordinais específicos (números inteiros ou caracteres) ou intervalos (faixas entre dois valores), não sendo possível utilizar outros tipos de dados nem expressões lógicas.

7.3 Exercícios

9. Um conhecido seu está fazendo um programa que necessita testar se determinado número digitado pelo usuário pertence à categoria dos números naturais (inteiros) ou dos números reais (fracionários). Implemente um programa que leia um número e informe se ele é Inteiro ou Real.
10. Solicitar ao usuário que digite um número. O programa deve informar se o número digitado é par ou ímpar.

8 COMANDOS DE REPETIÇÃO

No Turbo Pascal 5.5 existem três comandos de repetição:

- FOR ... DO (Para ... Faça ...)
- REPEAT ... UNTIL (Repita ... Até ...)
- WHILE ... DO (Enquanto ... Faça ...)

8.1 O Comando 'FOR'

O 'FOR' é um comando que executa um trecho de código diversas vezes. O número de vezes é determinado por uma variável de controle -- o contador. O Contador é uma variável, geralmente pertencente ao domínio dos números inteiros, que "conta" quantas vezes o laço de repetição já foi executado.

Sintaxe:

```
FOR <variavel_contadora> := <valor_inicial> TO <valor_final> DO  
BEGIN  
    { * trecho de código a ser repetido * }  
END;
```

Lê-se em português:

```
PARA <variavel_contadora> := <valor_inicial> ATÉ <valor_final> FAÇA  
INÍCIO  
    { * trecho de código a ser repetido * }  
FIM;
```

Exemplo:

```
Program Exemplo_Comando_FOR;  
    Var contador: Integer;  
    Begin
```

```

FOR contador := 0 TO 10 DO Begin
    Writeln('Execução: ', contador);
End;
End.

```

Note que o incremento da variável (a cada iteração) é feito automaticamente pelo comando FOR.

O FOR também pode ser utilizado para *decrementar* o valor de uma variável. Desta forma, a contagem passa a ser regressiva ao invés de progressiva:

```

Program Outro_Exemplo_Comando_FOR;
Var contador: Integer;
Begin
    FOR contador := 10 DOWNTO 0 DO Begin
        Writeln('Execução: ', contador);
    End;
End.

```

Note que, nesse caso, é utilizada a palavra-chave *downto*.

8.2 O comando 'REPEAT' ... 'UNTIL'

O conjunto de comandos "REPEAT ... UNTIL" é similar ao comando FOR, pois ele também cria um laço de execução que repete determinado trecho de código diversas vezes. A maior diferença está no fato de que ele executa o laço até que determinada condição tenha sido atingida. Devido a isso, é indicado para situações onde não há limite numérico de iterações a serem feitas.

Sintaxe:

```

REPEAT
    { * trecho de código a ser repetido * }
UNTIL <condição>;

```

Lê-se em português:

```

REPITA
    { * trecho de código a ser repetido * }
ATÉ <condição_ser_verdadeira>;

```

É importante salientar que o conjunto de comandos REPEAT ... UNTIL não exige, obrigatoriamente, uma variável contadora. Decorrente disto, o programador deve, obrigatoriamente e por conta própria, *inicializar* o contador e incrementá-lo manualmente dentro do laço. Veja o exemplo.

Exemplo:

```
Program Exemplo_Comandos_REPEAT_UNTIL;
  Var contador: Integer;
  Begin
    contador := 0;    {* Inicialização *}
    REPEAT
      Writeln('Execução: ', contador);
      contador = contador + 1; {* Incremento *}
    UNTIL contador = 11;
  End.
```

OBS: O incremento NÃO é feito automaticamente pelos comandos REPEAT...UNTIL.

Exemplo 2: usando *repeat ... until* para validar entradas de dados

```
Program Exemplo_Comandos_REPEAT_UNTIL_2;
  Var salario: Real;
  Begin
    clrscr;
    REPEAT
      Writeln('Digite o salário: '); Readln(salario);
    UNTIL salário >= 0;
  End.
```

8.3 O comando 'WHILE'

O comando WHILE também serve para criar laços de execução/iteração. Ele funciona da mesma forma que os comandos 'REPEAT...UNTIL', com exceção de que o teste fica no início do laço. Dessa forma, os comandos do laço são executados **enquanto** a condição é verdadeira. Se a condição, por algum motivo, não for verdadeira antes da primeira iteração, o laço jamais é executado (o que não acontece com o REPEAT...UNTIL).

Sintaxe:

```
WHILE <condição> DO
```

```
Begin
    { * trecho de código a ser repetido * }
END;
```

Lê-se em português:

```
Enquanto <condição_for_verdadeira> faça
Início
    { * trecho de código a ser repetido * }
Fim;
```

Exemplo:

```
Program Exemplo_Comando_WHILE;
Var contador, numeroFinal: Integer;
Begin
    write('Desejas contar até quanto? ');
    readln(numeroFinal);
    contador := 0; { * inicialização * }
    WHILE contador < numeroFinal DO
    Begin
        Writeln(contador);
        contador := Contador + 1; { * incremento * }
    End;
End.
```

8.4 Exercícios

11. Um professor necessita calcular média das notas de seus alunos. Faça um programa que peça para o professor informar o número total de notas e depois solicite cada uma delas (não utilize vetores). Ao final, mostre a média aritmética das notas lidas.
12. Fazer um programa que solicite um número ao usuário e após mostre todos os números ímpares existentes entre 0 e o número informado.
13. Solicitar um número inteiro positivo ao usuário, validando a entrada de dados (informando se ele estiver errado e repetindo a solicitação até que esteja correto). Após o programa deve informar todos os números pares existentes entre 1 e o número fornecido pelo usuário.

Exemplo:

Digite um número inteiro positivo: -8

Valor incorreto!

Digite um número inteiro positivo: 8

Numero digitado: 8

Números inteiros pares entre 1 e 8: 2, 4, 6.

14. Modificar o programa anterior para que ao final ele pergunte ao usuário se ele deseja informar um outro número. Caso positivo, o programa deve ser repetido.
15. Faça um programa que solicite ao o usuário que informe a idade e valide a entrada de dados, ou seja, repita a leitura até que ela esteja correta (maior do que zero). Ao final, mostre a idade.

9 ARRAYS (ARRANJOS OU VETORES)

Imagine que você necessite ler cinco (5) números. Imagine também que você necessite armazenar cada um destes valores em uma variável diferente, pois eles são necessários até o final da execução do programa.

Vamos supor que estes números correspondam aos salários de 5 funcionários de uma empresa e que você tenha que mostrá-los de maneira ordenada (do mais alto ao mais baixo) ao final da leitura.

A maneira mais direta de se fazer isto (sem o uso de vetores) é declarar uma variável para cada salário, lê-los, ordená-los e informá-los.

Exemplo:

```
Program Cinco_salarios;
Var s1, s2, s3, s4, s5: Real;
Begin
  write('Digite o primeiro salário: '); readln(s1);
  write('Digite o primeiro salário: '); readln(s2);
  write('Digite o primeiro salário: '); readln(s3);
  write('Digite o primeiro salário: '); readln(s4);
  write('Digite o primeiro salário: '); readln(s5);
  { processamento da ordenação dos salários...
    Ao final deste, s1 possui o salário mais alto
    e s5 o mais baixo }
  writeln;
  writeln('Salários (do mais alto ao mais baixo):');
  write(s1); write(s2); write(s3); write(s4); write(s5);
End.
```

Imagine agora que o número de funcionários seja 100 ou 1000: o código de leitura, ordenação e impressão desses salários cresceria enormemente! Uma forma de minimizar isto é utilizar *arrays* (também chamados de **vetores** ou arranjos).

Os *arrays* são estruturas que permitem que uma única variável possua diversas posições, podendo então armazenar diversos valores. A cada posição está associado um índice, o que facilita a sua manipulação.

Veja o exemplo:

```
Program Cinco_salarios_2;
Var salario: array [1..5] of Real;
Begin
    write('Digite o primeiro salário: '); readln(salario[1]);
    write('Digite o primeiro salário: '); readln(salario[2]);
    write('Digite o primeiro salário: '); readln(salario[3]);
    write('Digite o primeiro salário: '); readln(salario[4]);
    write('Digite o primeiro salário: '); readln(salario[5]);
    { processamento da ordenação dos salários...
      Ao final deste, salario[1] possui o salário mais alto
      e salario[5] o mais baixo }
    writeln;
    writeln('Salários (do mais alto ao mais baixo):');
    write(salario[1]); write(salario[2]);
    write(salario[3]); write(salario[4]); write(salario[5]);
End.
```

Nesse exemplo podemos identificar que a sintaxe de declaração de um *array* é um pouco diferente da declaração de uma variável comum:

```
salario: array [1..5] of Real;
```

Sintaxe:

```
<variável>: array [1..<tamanho>] of <tipo>;
```

Onde:

<variável> é o identificador (nome) da variável;

<tamanho> é o tamanho máximo do array;

<tipo> é o tipo dos elementos do array, e pode ser qualquer um dos tipos existentes.

Exemplos:

```
nome : array [1..25] of Char;
idade : array [1..100] of Byte;
numero: array [1..50] of Integer;
```

Também verificamos no exemplo que a utilização de um elemento do *array* é muito similar a utilização de uma variável qualquer:

```
readln(salario[1]); { lê um valor na primeira posição do array }
```

A diferença é que a posição (ou índice) deve ser informada dentro dos colchetes! Fora isto, eles podem ser utilizados normalmente:

```
salario[2] := 10.30; { coloca 10.30 na segunda posição do array }
media := (salario[1]+salario[2]+salario[3])/3; { utiliza os
elementos 1, 2 e 3 no cálculo de uma média }
```

O exemplo citado, porém, não utiliza todo o potencial de um *array*. Vamos ver um exemplo que aproveita melhor a estrutura de *arrays*, utilizando um índice (variável que indica a posição do vetor desejada no momento):

```
Program Cinco_salarios_3;
Uses CRT;
Var salario: array [1..100] of Real;
    salarioAuxiliar : Real;
    indice, indiceAuxiliar;
Begin
    clrscr;
    { Leitura de 100 salários }
    For indice := 1 to 100 do
    Begin
        write('Digite o ', indice, 'º salário: ');
        readln(salario[indice]);
    End;
    { processamento da ordenação dos salários...
    For indice := 1 to 99 do
        For indiceAuxiliar := indice to 100 Do
            If salario[indice] < salario[indiceAuxiliar] then
```

```

        Begin
            { Inverte }
            salarioAuxiliar := salario[indice];
            salario[indice] := salario[indiceauxiliar];
            salario[indiceauxiliar] := salario[indice];
        End;
    clrscr;
    { Impressão dos 100 salários em ordem decrescente }
    For indice := 1 to 100 do
        writeln(salario[indice]);
    readln;
End.

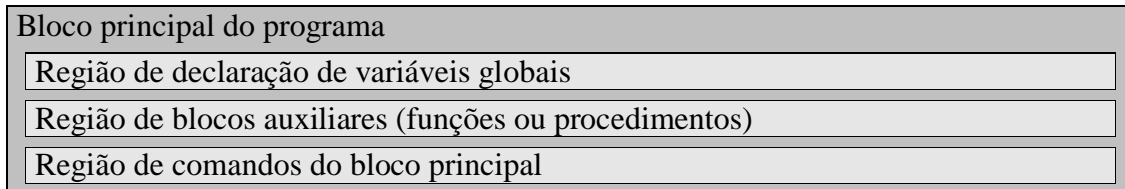
```

9.1 Exercícios

16. Um professor necessita calcular média das notas de seus alunos. Faça um programa que peça para o professor informar 4 notas (utilize vetores). Ao final, mostre a média aritmética das notas lidas.
17. Fazer um programa que pergunte ao usuário quantos números ele deseja informar. Após, cada número par informado deve ir para um *array* chamado "pares" e cada número ímpar informado deve ir para um *array* chamado "impares". Ao final, mostre o conteúdo de cada vetor.
OBS: Crie *arrays* fixos de 255 números e não deixe que o usuário digite mais de 255 números em cada um deles.
18. Ler 20 números (inteiros) em um *array*. Após, ordená-lo por ordem crescente e mostrá-lo.

10 PROCEDIMENTOS E FUNÇÕES

Você lembra da estrutura de um programa em Pascal? Ela era a seguinte:



Já trabalhamos com a maior parte delas menos a região de blocos auxiliares. É justamente ela que vamos estudar agora -- a de procedimentos e funções.

Procedimentos (*procedures*) e funções (*functions*) permitem que o programador sub-divida seu programa em diversas partes (sub-programas), sendo cada uma delas responsável por executar determinada tarefa (um procedimento ou função).

Dessa forma, o desenvolvimento de software fica mais prático, pois o programador pode ater-se à tarefa que o procedimento deve realizar e o seu foco de atenção torna-se bem menor, diminuindo as chances de problemas ocorrerem. Pode-se inclusive distribuir a tarefa de programação para várias pessoas, cada uma encarregada de uma parte do programa.

10.1 Procedimentos

Procedimentos (*procedures*) são trechos de código que executam determinada tarefa ao serem chamados e depois retornam o controle para o programa principal (como o comando 'clrscr' do Pascal, que limpa a tela).

10.1.1 Como declarar e criar procedimentos:

Todo procedimento é um sub-programa em Pascal que possui as mesmas características (a mesma estrutura) de um Programa em Pascal: nome, variáveis, bloco de código. Lembre-se disso.

Os procedimentos devem ser declarados em um arquivo a parte ou no início do programa, logo após a cláusula Var (caso o programa possua variáveis globais) ou USES (ou seja, após a declaração das bibliotecas utilizadas no programa).

Note que, depois de declarado, o procedimento funciona como um comando qualquer. Basta colocar seu nome para que ele seja chamado.

Veja o exemplo:

```
Program exemplo_procedure;  
Uses CRT;  
  
{ declaração do procedimento de inicialização da tela }  
Procedure preparaTela;  
Begin  
    TextColor(BLUE);          { ajusta a cor do texto para Azul }  
    TextBackground(BLACK);    { Preto para a cor de fundo }  
    Clrscr;                   { limpa a tela nas cores especificadas }  
End;  
  
Begin { bloco de comandos do programa principal }  
    preparaTela;              { chama o procedimento definido acima }  
    write('Olá!');  
    readln;  
End.
```

Seu programa pode possuir vários procedimentos e uns podem chamar os outros. A única restrição é que se um procedimento necessita de outro, o que é chamado deve vir antes.

Veja o exemplo:

```
Program exemplo_procedure2;  
Uses CRT;  
  
{ bloco de procedimentos }  
{ declaração do procedimento de cabeçalho }  
Procedure cabeçalho;  
Begin
```

```

write(' Programa exemplo de múltiplos procedimentos');
write('*****');
End;
{ declaração do procedimento de inicialização da tela }
Procedure preparaTela;
Begin
    TextColor(BLUE);
    TextBackground(BLACK);
    Clrscr;

    cabecalho; { chama procedimento de cabeçalho }
End;
{ declaração do procedimento de finalização }
Procedure voltaTelaNormal;
Begin
    TextColor(WHITE);
    TextBackground(BLACK);
    Clrscr;
End;

Begin { bloco de comandos do programa principal }
    preparaTela;      { chama o procedimento }
    write('Olá!');
    readln;
    voltaTelaNormal; { chama o procedimento }
End.

```

Seu procedimento também pode utilizar variáveis locais. Nesse caso, há uma região específica para isso, que fica depois da declaração do nome do procedimento.

Veja o exemplo:

```

Program exemplo_procedure3;
Uses CRT;
Var numero: Integer; { variável global ou do programa principal }
{ declaração do procedimento de inicialização da tela }
Procedure inicializacao;
{ declaração de variáveis do procedimento (locais) }
Var cor: Integer;
Begin
    TextColor(BLUE);
    for cor := 15 downto 0 do
    Begin

```

```

    textbackground(cor);
    clrscr;
    delay(300); { espera 300 milisegundos }
End;
End;

```

```

Begin
    inicializacao();
    write('Digite um número: ');
    readln(a);
End.

```

Se você prestar atenção, a estrutura do bloco de procedimento é muito parecida com a estrutura de um programa.

Veja:

```

Procedure inicializacao; { declaração do procedimento }
Var cor: Integer; { declaração de variáveis locais do procedimento }
Begin { bloco de comandos do procedimento }
    TextColor(BLUE);
    for cor := 15 downto 0 do
    Begin
        textbackground(cor);
        clrscr;
        delay(300);
    End;
End;

```

Um procedimento pode receber parâmetros. Parâmetros são valores que o programador passa para o procedimento quando o chama. O comando gotoxy(10, 10), por exemplo, recebe dois números inteiros como parâmetros: o primeiro correspondente à coluna e o segundo à linha onde o cursor deve ser posicionado.

Vamos supor que você queira criar um comando (procedimento) que limpe a tela com a cor fornecida como parâmetro pelo programador. Você faria isto da seguinte forma:

```

Program exemplo_procedure4;
Uses CRT;
{ declaração de procedimento que recebe parâmetro: }
Procedure limpaTelaColorido(cor: Integer);
Begin
    { utiliza o parâmetro como se fosse uma variável qualquer: }
    TextBackground(cor);
    Clrscr;

```

```

End;
Begin
    limpaTelaColorido(5);
    readln;
End;

```

O parâmetro é uma variável declarada dentro dos parênteses, que indica o tipo do dado que o programador deve passar ao chamar o procedimento. Este dado é automaticamente colocado (atribuído) à variável do parâmetro quando o procedimento começa a ser executado. Este parâmetro é uma variável normal e similar às outras variáveis declaradas dentro do procedimento. A maior diferença é que ela é inicializada com o valor passado como parâmetro.

Note que o parâmetro só tem validade dentro do procedimento, assim como as variáveis locais do procedimento. Logo, ao sair, cor não existirá mais.

Seu procedimento pode receber mais de um parâmetro, basta declará-los, separando-os por ponto-e-vírgula:

```

Program exemplo_procedure5;
Uses CRT;
{ declara procedimento que recebe vários parâmetros: }
Procedure escreveXY(coluna: Integer; linha: Integer; str: String);
Begin
    gotoxy(coluna, linha);
    write(str);
End;
Begin
    escreveXY(5, 5, 'Olá');
    readln;
End;

```

No exemplo anterior, foi criado o procedimento (comando) `escreveXY`. Este procedimento recebe três parâmetros: *coluna*, *linha* e *string*. Note, portanto, que os parâmetros podem ser de tipos diferentes. Na hora de chamar o comando, os parâmetros passados devem seguir a mesma ordem e tipos declarados no cabeçalho do procedimento.

Perceba que todo procedimento é composto de duas partes: o cabeçalho e o corpo. O cabeçalho é a linha que contém o nome do procedimento e seus parâmetros. O Corpo são os comandos que ele executa (o trecho que começa no *Begin* e termina no *End*).

```

Procedure limpaTelaColorido(cor: Integer);
Begin
  TextBackground(cor);
  Clrscr;
End;

```

.....> **cabeçalho**
- - - - -> **corpo**

10.2 Funções

Funções (*functions*) também são trechos de código que executam determinada tarefa e retornam o controle para o programa principal. A diferença está no fato de que as funções podem retornar um valor (como o comando 'ReadKey' do Pascal, que espera o usuário digitar uma tecla e retorna a tecla digitada).

Veja a diferença básica entre um procedimento e uma função:

clrscr;	<p>É um procedimento, pois não retorna valor. Não dá para fazer uma variável receber seu valor:</p> <p style="text-align: center;">a := clrscr; { errado! }</p>
tecla := ReadKey;	<p>É uma função que retorna um valor (no caso o código da tecla pressionada pelo usuário). Este valor pode ser atribuído a uma variável.</p>

10.2.1 Como declarar e criar funções

A forma de declarar uma função é muito parecida com a da declaração de um procedimento:

```

FUNCTION quadrado(numero: Integer): Integer;
Var resultado: Integer;
Begin
  resultado := numero * numero;
  quadrado := resultado;
End;

```

Note que as diferenças estão marcadas em negrito.

A primeira delas é a palavra *function*, que inglês, significa função. Em Pascal tudo é bem detalhado: o programa inicia com o "program", os procedimentos com "procedure" e as funções com "function"! Todos eles possuem estruturas bem similares.

Após vem o nome da função e seus parâmetros entre parênteses. A declaração e o funcionamento dos mesmos são idênticos aos dos procedimentos. Após, vem o tipo de dado que a função retorna. No caso, a função recebe um número e retorna este número ao quadrado. Logo, seu tipo de retorno é Integer. O tipo de retorno é obrigatório e deve vir sempre após os parênteses.

Finalmente, você coloca todos os comandos que executam a função. Quando você já tiver calculado o valor ou quiser retorná-lo por algum motivo, basta atribuir este valor ao nome da função. No exemplo, o comando "quadrado := resultado" diz para a função que o valor de resultado deve ser retornado.

Vamos ver esta função funcionando em um programa-exemplo que calcula o quadrado de um número:

```
Program exemplo_function1;
Var a, r;
{ declara uma função que calcula e retorna o quadrado de um número }
FUNCTION quadrado(numero: Integer): Integer;
Var resultado: Integer;
Begin
    resultado := numero * numero;
    quadrado := resultado;
End;
Begin
    write('Digite um número: '); readln(a);
    { você pode colocar o valor de uma função em uma variável: }
    r := quadrado(a);
    writeln('O quadrado de ', a, ' é: ', r);
    { você pode utilizá-la diretamente em um write: }
    write('O número 2 elevado ao quadrado é: ', quadrado(2));
    { e você pode utilizá-la em expressões matemáticas: }
    r := 10 + 27 DIV quadrado(3);
    write('O resultado da expressão é: ', r);
    readln;
End;
```

10.3 Exercícios

19. Crie um procedimento para inicializar a tela de seu programa. Ele deve limpar a tela com as cores de sua preferência;

20. Crie um procedimento para finalizar o seu programa. Ele deve limpar a tela e ajustar as cores para o padrão (fundo preto, letras brancas);
21. **Desafio:** Modifique os procedimentos de inicialização e finalização (exercícios 1 e 2) de modo que funcionem em conjunto. O primeiro, antes de ajustar a tela para as suas preferências, deve obter informações sobre o estado atual dela. O último deve restaurar as configurações para o estado que ela estava antes de seu programa ter sido executado;
22. Fazer o procedimento `linha(linha, colInicial, colFinal)`. Este procedimento deve desenhar uma linha horizontal na linha especificada pelo parâmetro *linha*, indo da *colInicial* até a *colFinal*;
23. Fazer o procedimento `coluna(coluna, linInicial, linFinal)`. Este procedimento deve desenhar uma linha vertical na coluna especificada, indo da *linInicial* até a *linFinal*;
24. Fazer o procedimento `janela(colInic, linInic, colFin, linFin)`. Este procedimento deve montar uma janela na tela que comece nas coordenadas especificadas pelos parâmetros *colInic* e *linInic* e termine nas coordenadas especificadas por *colFin* e *linFin*.
OBS: Utilize os procedimentos `linha` e `coluna` dos exercícios 4 e 5.
25. **Desafio:** Faça o procedimento `abrirJanela(ci, li, cf, lf)`, que abra uma janela (de dentro para fora) nas coordenadas especificadas. Dica: utilize o procedimento `janela` do exercício anterior.
26. **Desafio:** Faça o procedimento `fecharJanela(ci, li, cf, lf)`, que feche uma janela aberta anteriormente pelo comando `abrirJanela`. Faça com que os dois procedimentos funcionem em conjunto e restaurem o conteúdo que estava no lugar da janela antes de ela ter sido aberta.

11 REGISTROS (ESTRUTURAS) DE DADOS

As variáveis são a maneira mais simples de representar dados e informações no computador. Os tipos de dados oferecidos por uma linguagem permitem ao programador manipular objetos simples, tais como números inteiros (*Integer*), números reais (Real), Caracteres (*Char*), Seqüências de caracteres (*Strings*), valores booleanos (*Boolean*), entre outros. No entanto, para representar e armazenar objetos e entidades mais complexos, tais como pessoas, contas de um banco etc. torna-se necessário combinar os tipos primitivos em outros tipos de dados. O registro (RECORD) é a forma que a linguagem Pascal nos oferece para realizar essas combinações e definir novos tipos de dados compostos e mais complexos.

Para definir um novo tipo combinado, na forma de um registro, na linguagem Pascal, utilizamos a seguinte sintaxe:

```
Type <identificador_do_tipo> = Record
    <identificador_do_primeiro_atributo> : <tipo_do_atributo>;
    <identificador_do_segundo_atributo> : <tipo_do_atributo>;
    <identificador_do_último_atributo>   : <tipo_do_atributo>;
End;
```

Onde:

<identificador_do_tipo> é o nome que você deseja dar ao novo tipo de dados.

<identificador_do_n-ésimo_atributo> é o nome que você deseja dar ao n-ésimo atributo.

<tipo_do_atributo> é o tipo do atributo.

Exemplo:

```

Type Pessoa = Record
    nome   : String[30];
    idade  : Integer;
    sexo   : Char;
End;

```

O trecho de código acima define um novo tipo de variável chamado *Pessoa*, que servirá para armazenar na memória do computador informações sobre pessoas. Esse tipo possui como atributos (sub-variáveis) o *nome*, a *idade* e o *sexo* da pessoa. Note que os atributos foram definidos através de tipos básicos do Pascal: *String* (cadeia de caracteres), *Integer* (números inteiros) e *char* (caracter).

A partir deste momento podem ser criadas variáveis que tenham como *Pessoa* como tipo:

```

Var Funcionario : Pessoa;

```

No exemplo acima é criada a variável “*funcionário*” cujo tipo é “*Pessoa*”. Essa variável pode ser manipulada (acessadas) usando a seguinte sintaxe:

```

<objeto>.<atributo>

```

Ou seja, o ponto ‘.’ é o caractere especial utilizado para indicar que se deseja manipular um atributo do objeto (sub-variável). Coloca-se, portanto, o identificador do objeto, o ponto e o identificador do atributo. Esses atributos podem ser utilizados normalmente, como se fossem variáveis comuns, inclusive por procedimentos e funções de manipulação de variáveis oferecidas pela linguagem.

Veja os exemplos:

```

Funcionario.nome := 'Márcia'; { Atribui um nome ao funcionário }
Funcionario.idade := 24;      { Atribui uma idade ao funcionário }
Funcionario.sexo := 'F';     { Atribui um sexo ao funcionário }
Readln(Funcionario.nome);    { Lê o nome do funcionário }
Writeln(Funcionario.idade);  { Lê a idade do funcionário }

```

Essas variáveis funcionam como se fossem variáveis normais, inclusive são perdidas quando o programa é finalizado. Para que os conteúdos delas sejam salvos, para que

possam ser manipulados em outra execução do programa, é necessário armazená-los em um arquivo. Para tanto, leia o capítulo seguinte que aborda como se trabalha com arquivos.

12 TRABALHANDO COM ARQUIVOS

Neste capítulo você irá aprender como manipular arquivos-binários⁵ na linguagem Turbo Pascal.

12.1 Criando referências (apelidos) para um arquivo

Para que um objeto seja armazenado em um arquivo é necessário primeiro criar um novo arquivo ou abrir um já existente (caso objetos já tenham sido armazenados anteriormente).

Porém, antes de criar (ou abrir) fisicamente o arquivo (no dispositivo) é preciso declarar uma variável de programa que referencie⁶ esse arquivo no dispositivo de armazenamento. Por exemplo, se você deseja criar um arquivo chamado *funcionarios.dat* para armazenar seus registros de funcionários vai precisar de uma variável de arquivo para utilizá-lo. Para declarar uma variável do tipo *referência para arquivo* em Pascal utiliza-se a seguinte notação:

```
<identificador_do_arquivo> : File of <tipo_dos_registros >;
```

Onde *<identificador_do_arquivo>* é o nome da variável, *file of* são palavras especiais que avisam ao Pascal que essa variável é do tipo *referência de arquivo* e o *<tipo_dos_registros>* indica que tipo de objetos será armazenado nesse arquivo.

⁵ Arquivos binários são aqueles em que as informações são gravadas em formato binário, exatamente como estão armazenadas na memória do computador. Além do arquivo binário também existem os arquivos-texto, que são aqueles cujos dados podem ser compreendidos por um ser humano (tais como os arquivos .TXT que você cria com o “Bloco de Notas” do Windows). A criação de um arquivo-texto é muito semelhante a dos arquivos binários. Basta você especificar o tipo do arquivo como sendo do tipo “Text”. Exemplo: Var arquivo: Text;

⁶ Uma referência é como se fosse um apelido para o arquivo.

Vamos supor que o nome (identificador) escolhido para essa variável seja *Arquivo_de_funcionarios*, o código em Pascal para declarar e criar esta variável seria:

```
Var Arquivo_de_funcionarios : File of Pessoa;
```

A partir desse momento você tem um arquivo preparado para armazenar informações (registros) do tipo *Pessoa*. Note que quando você declara um arquivo como sendo de determinado *tipo*, o Pascal só consegue armazenar informações desse tipo, ou seja, não podem ser armazenados registros de outros tipos, mas, sim, somente objetos do tipo *Pessoa*.

Assim, sempre que quisermos utilizar o arquivo *funcionarios.dat* devemos utilizar o apelido dele, ou seja, a sua referência, que no caso é *Arquivo_de_funcionarios*.

Porém, ainda não foi dito que a variável *Arquivo_de_funcionarios* é um apelido (referência) para o arquivo *funcionarios.dat*. É preciso fazer essa ligação entre o apelido e o arquivo fisicamente disposto no dispositivo. Em Pascal, essa ligação (mapeamento) é feita pelo comando *Assign*:

```
Assign(Arquivo_de_funcionarios, 'c:\temp\funcionarios.dat');
```

O trecho de código listado anteriormente liga (mapeia) a variável *Arquivo_de_funcionarios* ao arquivo *funcionarios.dat*. Note que é possível especificar também o *drive* e o *caminho* onde se encontra o arquivo. No caso, a variável *Arquivo_de_funcionarios* foi ligada com o arquivo denominado “*funcionarios.dat*” que se encontra no diretório “*temp*” do drive “*C:*”, e cria uma referência para um arquivo no dispositivo de armazenamento (no disco rígido).

12.2 Criando e abrindo arquivos

O comando utilizado em Pascal para criar um novo arquivo no disco é *rewrite*. Veja o exemplo:

```
Rewrite(<referência_ao_arquivo>);
```

Seguindo o exemplo do arquivo de funcionários, o comando seguinte criaria um novo arquivo, chamado *funcionarios.dat*, no dispositivo:

```
Rewrite(Arquivo_de_funcionarios);
```

Note que, se já existir um arquivo com esse mesmo nome no local especificado, o arquivo será zerado (seu conteúdo apagado). Por outro lado, se você quiser abrir um arquivo existente para consultar suas informações, sem zerá-lo, deve utilizar o comando `Reset`:

```
Reset(Arquivo_de_funcionarios);
```

Esse comando abre o arquivo solicitado sem que as informações contidas nele sejam perdidas. Porém, se o arquivo não existir ele não será criado e nada irá acontecer.

12.3 Movimentando-se em um arquivo

A partir do momento em que um arquivo é criado ou aberto, ele é posicionado no primeiro registro. Caso você queira mover-se para um outro registro basta utilizar o comando `Seek`:

```
Seek(<referencia_ao_arquivo>, <número_do_registro>);
```

Exemplo:

```
Seek(Arquivo_de_funcionarios, 4);
```

Existem outras funções que podem ser utilizadas em conjunto com a função *seek* a fim de facilitar sua utilização. Entre elas encontra-se a função que testa se o arquivo chegou ao seu final, a que devolve a posição atual e função que indica o número total de registros armazenados no arquivo:

- EOF**(<referência>); → Indica se a posição atual é o final do arquivo
- Filepos**(<referência>); → Retorna o registro atual
- Filesize**(<referência>); → Retorna o número total de registros no arquivo

12.4 Gravando um registro

Para gravar um registro no arquivo você deve primeiro se posicionar na região em que você deseja gravá-lo e utilizar o comando de gravação. Lembre-se de que o comando de gravação não insere o registro no arquivo. Isso significa que se você se posicionar em um registro já existente e mandar gravar, os dados do arquivo serão substituídos (alterados). O comando gravar só insere (cria) um novo registro se você estiver posicionado no final do arquivo.

Importante: quando você grava (ou lê) um registro do arquivo você é posicionado automaticamente no próximo registro.

O comando em Pascal que grava um registro no arquivo é o *write*:

```
Write(<referência_ao_arquivo>, <registro>);
```

Exemplo:

```
Write(Arquivo_de_funcionarios, Funcionario);
```

12.5 Lendo um registro

Assim como na gravação, para você ler um registro deve primeiro se posicionar nele (usando a função *seek*).

Importante: Quando você lê (ou grava) um registro do arquivo você é posicionado automaticamente no próximo registro.

O comando de leitura de registros em Pascal é o *Read*:

```
Read(<referência_ao_arquivo>, <registro>);
```

Exemplo:

```
Read(Arquivo_de_funcionarios, Funcionario);
```

Esse comando lê o registro atual do arquivo e coloca-o na variável especificada (no caso, *funcionario*). Essa variável pode ser então manipulada. Lembre-se que essa variável é independente do arquivo. Caso você altere o conteúdo dela, o registro correspondente no arquivo não será alterado. Caso você deseje alterar o arquivo para corresponder às alterações realizadas em memória, você deve se re-posicionar e utilizar o comando *write*.

12.6 Fechando um arquivo

Antes de finalizar (sair) um programa que utilize arquivos você deve fechar o arquivo. O processo é similar a uma gaveta: se você abriu, deve fechar... O comando é *Close*:

```
Close(<referência_ao_arquivo>);
```

Exemplo:

```
Close(Arquivo_de_funcionarios);
```

13 TABELA ASCII

A tabela ASCII, acrônimo de *American Standard Code for Information Interchange* (Código Americano Padrão para o Intercambio de Informações), é um conjunto de valores que representam caracteres e códigos de controle armazenados ou utilizados em computadores. Nessa tabela, cada caractere (letra ou número) possui um código correspondente. Cada código ocupa 1 byte (tipo char), o que nos dá 255 posições.

Dessas 255 posições, as primeiras 32 (0 a 31) correspondem a códigos de controle que são utilizados para controlar dispositivos (tais como monitores e impressoras). Logo, a maioria desses códigos não produz caracteres quando impressos em algum dispositivo (como a tela do computador). Da posição 32 até a 127 estão alocados os caracteres do conjunto padrão, correspondendo a caracteres do alfabeto latino (sem acentos, maiúsculos e minúsculos), dígitos (0 a 9) e alguns outros símbolos comuns. Os demais códigos (128 a 255) formam o conjunto estendido, e podem variar de região para região.

A seguir é apresentada a tabela ASCII padrão (código 0 a 127), que você encontra na maioria dos computadores.

Códigos de controle	Cód	Caractere	Cód	Caractere
	0	NULL (nulo)	1	SOH (Start of Heading / Início de cabeçalho)
	2	STX (Start of TeXt / Início de Texto)	3	ETX (End of TeXt / fim de texto)
	4	EOT (End Of Transmission / fim de transmissão)	5	ENQ (ENQuiry / inquirição, consulta)
	6	ACK (ACKnowledge / confirmação, entendido)	7	BEL (BELL, BEEP / Campainha)
	8	BS (Backspace / retorno de 1 caractere)	9	HT (Horizontal Tab / Tabulação horizontal)
	10	LF (Line Feed / alimentação, mudança de linha)	11	VT (Vertical Tab / Tabulação vertical)

Cód	Caractere	Cód	Caractere
12	FF (Form Feed / Alimentação de formulário)	13	CR (Carriage Return / retorno ao início da linha)
14	SO (Serial Out / Saída Serial) (Shift Out / deslocamento para fora)	15	SI (Serial In / Entrada Serial) (Shift In / deslocamento para dentro)
16	DLE (Data Link Escape / escape de conexão)	17	DC1/XON (Device Control1/controlado de dispositivo1)
18	DC2 (Device Control 2 / controle de dispositivo2)	19	DC3/XOFF (Device Control3/controlado de dispositivo3)
20	DC4 (Device Control 4 / controle de dispositivo4)	21	NAK (Negative Acknowledge / confirmação negativa)
22	SYN (SYNchronous Idle / espera síncrona)	23	ETB (End Transm. Block/bloco de fim de transmissão)
24	CAN (Cancel / cancelamento)	25	EM (End of Media / Fim do meio ou mídia)
26	SUB (SUBstitute, substituir)	27	ESC (ESCAPE / escape)
28	FS (File Separator / Separador de arquivo)	29	GS (Group Separator / separador de grupo)
30	RS (Request to Send, Record Separator / requisição de envio, separador de registro)	31	US (Unit Separator / separador de unidade)

Cód	Carac	Cód	Carac	Cód	Carac	Cód	Carac	Cód	Carac	Cód	Carac
32	<espaço>	33	!	34	"	35	#	36	\$	37	%
38	&	39	'	40	(41)	42	*	43	+
44	,	45	-	46	.	47	/	48	0	49	1
50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=
62	>	63	?	64	@	65	A	66	B	67	C
68	D	69	E	70	F	71	G	72	H	73	I
74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U
86	V	87	W	88	X	89	Y	90	Z	91	[
92	\	93]	94	^	95	_	96	`	97	a
98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m
110	n	111	o	112	p	113	q	114	r	115	s
116	t	117	u	118	v	119	w	120	x	121	y
122	z	123	{	124		125	}	126	~	127	<delete>

Conjunto padrão

Os códigos seguintes não são padrão e podem variar dependendo da configuração do seu computador, principalmente se você não estiver trabalhando com o MSDOS configurado para o idioma português ou estiver utilizando outro sistema operacional. Nestes casos, procure informar-se sobre o padrão UNICODE (<http://www.unicode.org>), utilizado por sistemas como o Windows, ou utilize o código-exemplo seguinte (após a tabela) que imprime a tabela ASCII na tela do seu computador.

Cód	Carac	Cód	Carac	Cód	Carac	Cód	Carac	Cód	Carac	Cód	Carac
128	Ç	129	ü	130	é	131	â	132	ä	133	à
134	â	135	ç	136	ê	137	ë	138	è	139	ï
140	î	141	ì	142	Á	143	Â	144	É	145	æ
146	Æ	147	ô	148	ö	149	ò	150	û	151	ù
152	ÿ	153	Ö	154	Ü	155	ø	156	£	157	Ø
158	×	159	f	160	á	161	í	162	ó	163	ú
164	ñ	165	Ñ	166	ª	167	º	168	¿	169	®
170	γ	171	½	172	¼	173	ı	174	«	175	»
176	⋮	177	⋮	178	⋮	179		180	†	181	Á
182	Â	183	À	184	©	185	‡	186	‖	187	¶
188	‡	189	¢	190	¥	191	γ	192	⊥	193	⊥
194	⊥	195	†	196	—	197	†	198	ã	199	Ã
200	⊥	201	⊥	202	⊥	203	⊥	204	‡	205	=
206	‡	207	⊥	208	δ	209	Ð	210	Ê	211	Ë
212	È	213	¬	214	Í	215	Î	216	Ï	217	⋮
218	Γ	219	■	220	■	221	ı	222	Ï	223	■
224	Ó	225	β	226	Ô	227	Ò	228	õ	229	Õ
230	Μ	231	Ρ	232	ρ	233	Ú	234	Û	235	Û
236	ý	237	Ý	238	-	239	˘	240	-	241	±
242	≡	243	¾	244	¶	245	§	246	÷	247	,
248	°	249	“	250	.	251	ı	252	³	253	²
254	■	255									

Conjunto estendido (padrão brasileiro/português)

13.1 Programa-exemplo que imprime a tabela ASCII na tela do seu computador

```
Program Tabela_ASCII;
Uses CRT;
Var codigo : Integer;
Begin
  clrscr;
  { começa no 32 para não mostrar os códigos de controle }
  For codigo := 32 to 255 do
    Write(código:3, \:', Chr(código):3, \ ');
  Readln;
End.
```

RESUMO DOS PRINCIPAIS OPERADORES, FUNÇÕES E PROCEDIMENTOS DA LINGUAGEM TURBO PASCAL

Operadores aritméticos / matemáticos	
+	<u>Soma</u> dois números e devolve o resultado no mesmo tipo dos operandos. Se eles forem de tipos diferentes, o resultado é do tipo mais complexo.
-	<u>Subtrai</u> dois números e devolve o resultado no mesmo tipo dos operandos. Se eles forem de tipos diferentes, o resultado é do tipo mais complexo.
*	<u>Multiplca</u> dois números e devolve o resultado no mesmo tipo dos operandos. Se eles forem de tipos diferentes, o resultado é do tipo mais complexo.
/	<u>Divide</u> dois números reais e devolve um número real como resultado.
div	<u>Divide</u> dois números inteiros e devolve um número inteiro correspondente ao quociente da divisão.
mod	<u>Divide</u> dois números inteiros e devolve um número inteiro correspondente ao resto da divisão.

Operadores Relacionais	
=	Igualdade
<>	Diferença
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

Operadores diversos	
+	concatena (junta) duas strings.
:=	atribuição

Operadores lógicos / booleanos	
NOT	<u>Negação, não</u> . Inverte o resultado de uma expressão booleana.
AND	<u>E</u> . Retorna verdadeiro se os dois operandos (dois lados) forem verdadeiros.
OR	<u>OU</u> . Retorna verdadeiro se qualquer um dos lados da expressão booleana for verdadeiro.
XOR	<u>OU exclusivo</u> . Retorna verdadeiro se somente um dos lados da expressão for verdadeiro (nem todos os compiladores implementam este operador).

Funções matemáticas / manipulação numérica	
int(x)	Retorna a parte inteira de um número Real (a que vem antes da vírgula). O resultado é um número real.
frac(x)	Retorna a parte fracionária de um número Real (a que vem depois da vírgula). O resultado é um número real.
abs(y)	Retorna o valor absoluto de um número inteiro (ou seja, o número sem seu sinal).
sqr(y)	Retorna o quadrado de um número.
sqrt(y)	Retorna a raiz quadrada de um número.
round(x)	Retorna a parte inteira de um número, arredondando-o. O resultado é um número inteiro.
trunc(x)	Retorna a parte inteira de um número Real, sem arredondá-lo. O resultado é semelhante ao da função int, com exceção que o resultado é um número inteiro.
cos(x)	Retorna o co-seno de um número (em radianos).
sin(x)	Retorna o seno de um número (em radianos).
exp(x)	Retorna o número e (2.71....) elevado ao valor solicitado.
Ln(x)	Retorna o logaritmo natural de x.

Procedimentos de controle do vídeo e do teclado	
clrscr	Apaga a tela.
clreol	Apaga da posição atual do cursor até o final da linha.
delline	Apaga a linha onde está o cursor.
insline	Insere uma linha na posição atual do cursor.
gotoxy(c,l)	Posiciona o cursor na coluna (c) e linha (l) especificados.
textcolor(c)	Altera a cor do texto para a cor especificada (c).
textbackground(c)	Altera a cor de fundo do texto para a cor especificada (c).

Procedimentos de controle do vídeo e do teclado (continuação)

Window(ci, li, cf, lf)	Define uma janela de texto nas coordenadas especificadas por ci (coluna inicial), li (linha inicial), cf (coluna final) e lf (linha final). O posicionamento do cursor, a partir deste comando, dá-se dentro das coordenadas especificadas. O valor normal é 1,1, 80, 25.
WhereX	Retorna a coluna atual do cursor.
WhereY	Retorna a linha atual do cursor.

Funções e procedimentos de manipulação de *strings* e caracteres

Length(str)	Retorna o tamanho (número de caracteres) de uma string.
Insert(s1, s2, p)	Insere uma string (s1) em outra (s2), na posição especificada por <i>p</i> .
Copy(s, i, t)	Retorna uma parte de uma string, a partir da posição inicial (i), no tamanho especificado (t).
Delete(s, i, t)	Apaga parte de uma string, iniciando na posição desejada (i) e indo até o tamanho especificado (t)
Concat(s1, s2)	Retorna a concatenação de s1 + s2.
Pos(s1, s2)	Retorna a posição de s1 em s2.
Str(v, s)	Converte um número (v) para uma string (s).
UpCase(c)	Retorna a versão maiúscula do caractere especificado
Val(s, n, erro)	Converte uma string (s) para um número (n), se for possível. Se não for possível, erro é diferente de zero.

Funções e procedimentos diversos

random(x)	Retorna um número aleatório entre 0 e o número informado.
sizeof(var)	Retorna o tamanho em bytes de uma variável ou tipo de variável.
chr(n)	Retorna o caractere correspondente ao código (n) na tabela ASCII.
ord(c)	Retorna o código do caractere (c) na tabela ASCII.
Inc(x)	Incrementa x.
Dec(x)	Decrementa x.
Succ(x)	Retorna o sucessor de x.
Pred(x)	Retorna o predecessor de x.
Randomize	Inicializa o gerador de números aleatórios.
Sound(x)	Gera som na frequência especificada por x.
Nosound	Encerra a geração de som iniciada por sound.
Delay(x)	Cria um atraso (espera) de x milissegundos.
Halt	Sai do programa.

Funções e procedimentos diversos (continuação)

Exit	Sai do bloco atual ou vai para o final do bloco corrente.
-------------	---

Manipulação de arquivos

Assign(arq, nom)	Associa o nome de arquivo especificado no parâmetro <i>nom</i> à variável <i>arq</i> .
Append(arq)	Abre o arquivo-texto especificado pela variável <i>arq</i> para inclusão de novas informações no seu fim.
Close(arq)	Fecha o arquivo associado à variável <i>arq</i> , aberto anteriormente.
Eof(arq)	Testa se o final do arquivo associado à variável <i>arq</i> foi encontrado. Retorna <i>true</i> se sim.
Eol(arq)	Testa se o final da linha do arquivo associado à variável <i>arq</i> foi atingido. Retorna <i>true</i> se sim.
Erase(arq)	Apaga um arquivo.
FilePos(arq)	Retorna a posição atual no arquivo (registro corrente).
FileSize(arq)	Retorna o tamanho do arquivo (número de registros).
Read(arq, var)	Lê informações do arquivo e coloca na variável especificada.
ReadLn(arq, var)	Lê informações do arquivo-texto e coloca na variável especificada.
Reset(arq)	Abre um arquivo para leitura a partir do seu início.
Rewrite(arq)	Cria um arquivo para gravação. Se ele já existir, será eliminado.
Seek(arq, pos)	Posiciona-se na posição especificada.
Write(arq, var)	Grava o conteúdo da variável especificada no arquivo.
WriteLn(arq, var)	Grava uma linha de texto no arquivo especificado.

BIBLIOGRAFIA E LEITURA RECOMENDADA

BOENTE, Alfredo. **Aprendendo a Programar em Pascal - Técnicas de Programação**. Rio de Janeiro: Brasport, 2003.

BOENTE, Alfredo. **Lógica de Programação - Construindo Algoritmos Computacionais**. Rio de Janeiro: Brasport, 2003. p. 224.

CARROW, David W. **Programação em Turbo Pascal**. São Paulo: McGraw-Hill. 1988. p. 396.

PREISS, Bruno R. **Estruturas de Dados e Algoritmos: Padrões de projetos orientados a objetos com Java**. Rio de Janeiro: Campus, 2001. p. 566.

MANZANO, José Augusto N. G.; YAMATUMI, Wilson Y. **Estudo Dirigido de Turbo Pascal**. São Paulo: Érica, . p.232.