Apostila: Bacharelado em Sistemas de informação - BSI
Professor: Wesley A. Gonçalves
Disciplina: Introdução a Programação

Aprendendo Programar em Pascal

Versão 1.5

Professor: Wesley A. Gonçalves

PARACATU – MINAS GERAIS
BRASIL – 2007

Apresentação

Esta apostila destina-se a todos aqueles que desejam aprender a 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 isto, 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, ou seja, o simples fato de ler este pequeno e simples apostila de linguagem Pascal, não basta para aprender a programar em Pascal.

Por fim, estou a disposição de todos, que se aventurem a acompanhar esta apostila, para tirar dúvidas, assim como para receber críticas.

I – INTRODUÇÃO

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 apostilas 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 ensinarmos aos estudantes a formulação sistemática e metódica de algoritmos, através de técnicas que são características da programação.

Como já disse, 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, inclusive por mim, 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 tão pouco 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.
Por outro lado, o processo de compilação é de certa forma moroso, pois deve seguir as seguintes etapas:

1-) Devemos utilizar um editor de textos para escrever e armazenar em disco o nosso programa fonte.
2-) Utilizar um compilador para traduzir o programa fonte para um programa em linguagem de máquina.
3-) 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, pois bem, 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 ele possui numa forma integrada, um editor de textos compatível com o Wordstar, um compilador e um linkeditor. O processo de compilação pode ser feito tanto em disco como em memória, o que faz com que ele seja muito rápido. 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 muito além, pois ele possui inúmeras procedures e funções a mais do que as existentes no padrão da linguagem Pascal.

I.3 - *Equipamento necessário*.

Todos os exemplos e programas contidos nesta apostila, foram escritos num compatível 486DX 50 com dois acionadores de discos de dupla face e alta densidade, 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

Bom, acho que aqueles que nunca tiveram a oportunidade de fazer um programa em Pascal, devem estar muito curiosos para saber como deve ser o seu aspecto. Por isso, antes de prosseguir com os meandros da linguagem Pascal, eu 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 Primeiro_Exemplo; { 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. }

```
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);
```

{o Turbo Pascal possui diversos tipos de variáveis predefinidas, mas também permite definir novos tipos na subárea type }

```
    Var idade           :integer;
    altura              :real;
    nome                :string[30];
    sexo                :char;
    nacionalidade       :n;
```

{ todas as variáveis que forem utilizadas no corpo do programa deverão ser declaradas na subárea Var }

```
Procedure Linha;
```

*{a procedure equivale ao conceito de sub-rotina. Sua estrutura pode se tornar tão complexa como de um programa. Esta procedure, traça uma linha
na posição atual do apostilar }*

```
    Var i:integer;
    Begin
    For i:=1 to 80 do Write('-');
    end;
    Function Soma(x,y:integer):integer;
```

{o Turbo Pascal possui diversas funções pré-definidas, mas o programador também pode definir as suas próprias }

```
    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;
            { estas linhas serão puladas }
nacionalidade:=BRASILEIRA;
Write('Minha nacionalidade e brasileira');
fim:
Write('Prazer em conhece-lo');
End.
```

II.2 - *Estrutura de um programa 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

Darei agora, uma breve explicação de cada subárea, pois mais para frente estudaremos cada uma delas com profundidade. Na subárea Label, devemos declarar todos os labels que forem utilizados no corpo do programa. Os labels são utilizados em conjunto com a instrução goto.
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.
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, po-

demos definir quantas sub-rotinas quisermos. Elas são chamadas durante o programa pelos seus respectivos nomes.

Finalmente, na subárea Functions podemos definir novas funções que depois poderemos utilizar no programa embora o Turbo Pascal possua inúmeras funções pré-definidas. Estas sub-áreas só são obrigatórias caso nós estejamos precisando. 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: + - * / = ^ < > ( ) [ ] { } . , : ; ' # $

Observações:

1-) O Turbo Pascal não faz distinção entre letras maiúsculas e minúsculas, de tal forma que no desenvolvimento desta apostila eu utilizarei os dois tipos da forma que achar mais conveniente.
2-) Embora na maioria das linguagens o sinal de atribuição de valores a variáveis seja o **=**, em Pascal, o símbolo de atribuição é ***:=***,

exemplos:
A = 100 em Basic
A := 100 em Pascal

3-) Dois pontos em seguida (..) indica um delimitador de faixa, exemplo:
1..30 --> todos inteiros entre 1 e 30 inclusive.

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 e não podem ser utilizadas como identificadores de variáveis, procedures, functions etc. Algumas das 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 apostilar e assim por diante.

Constantemente, novas procedures e functions estão sendo criadas pela Borland International (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.

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 formados 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 ser palavra reservada.

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.1.4 - Comentários

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

III.1.5 - Números

No Turbo Pascal, podemos trabalhar com números inteiros e reais, sendo que os números inteiros podem ser representados na forma hexadecimal, para tanto, basta precedê-los do símbolo $. Os números reais também podem ser representados na forma exponencial.
Isso tudo varia de versão para versão do turbo Pascal, citarei aqui as faixas de valores válidas para a versão 7.0:

TIPO	FAIXA	FORMATO
Shortin g	-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.1.6 - Strings

Strings são conjunto de caracteres entre aspas simples.

Exemplos:

'isto é uma string'
'123456' etc.

III.1.7 - Caracteres de controle

Existem alguns caracteres que têm significados especiais. São os caracteres de controle. Exemplos:

Control G => Bell ou beep
Contorl L => Form Feed
etc.

Em Turbo Pascal, também podemos utilizar estes caracteres. Para tanto, eles devem ser escritos pelo seus valores ASCII correspondentes, precedidos do símbolo #, ou então a letra correspondente precedida do símbolo **^**.
Exemplo: ***Control G => #7 ou ^G***

III.2 - Definição de variáveis

Como já dissemos, todas as variáveis que forem utilizadas no corpo do programa, devem ser declaradas numa subárea específica chamada Var.
Para estudarmos 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
- todos os tipos de reais citados acima

Escalares estruturados:

- String
- Array
- Record
- File
- Set
- 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 tais como dígitos, letras e símbolos tais como **&**, #,***** etc.
Os caracteres devem vir entre aspas simples. ('José')

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.

SHORTINT - ***INTEGER*** - ***LONGINT*** - ***BYTE*** - ***WORD***:
Ver tabela acima.

REAL - ***SINGLE*** - ***DOUBLE*** - ***EXTENDED*** - ***COMP***:
Ver tabela acima.

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;
    (* aqui começa o programa *)

  Begin
        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. Lembre-se que 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 linhas 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. 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. *)
```

Observação: *Quando damos os valores que os dados podem assumir através da declaração type, o Turbo Pascal assume, automaticamente, que o valor da direita vale mais que o da esquerda e assim. 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.*

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 e que podemos utilizá-las sem ter que declará-las. São elas:

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

III.3.3 - Constantes tipadas

A declaração de variáveis na subárea Var, apenas reserva espaço de memória para elas, mas não as inicializa, ou seja, até que se atribua valores a elas, seus valores serão desconhecidos. Sob certas circunstâncias, seria interessante que pudéssemos ao mesmo tempo em que declaramos a variável, dar seu valor inicial. Isto é possível com o conceito de constante tipada cuja sintaxe é:

```
Const variável : tipo = valor;
```

Exemplos:

```
const Contador : integer = 100;
    c : char = 'A';
```

Estamos definindo duas variáveis, uma chamada contador que é inteira e vale inicialmente **100**, e outra chamada **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

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.
```

III.4.2 - Operadores lógicos

AND	E	lógico
OR	OU	lógico
XOR	OU	EXCLUSIVO lógico

Estes operadores só aceitam como operandos, valores lógicos, ou seja :
TRUE e FALSE .

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.

Exemplo:

{PROGRAMA UTILIZANDO OS OPERADORES LÓGICOS}

```
Program operadores_logicos;
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 *);
End.
```

III.4.3 - Operadores relacionais

O Turbo Pascal possui ao todo 7 operadores relacionais que são muito utilizados nas tomadas de decisões, são eles:

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

Exemplos:

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

A avaliação será verdadeira se uma ou outra expressão for verdadeira, no caso, como C < D então a resposta é TRUE

III.4.4 - Operadores entre bits

Os operadores entre bits só podem ser aplicados em dados dos tipos *byte* ou *integer* e o resultado é do tipo *integer*. Eles agem bit a bit e podem ser aplicados na notação hexadecimal ou decimal. São eles:

SHL - SHift Left

Desloca **n** bits à esquerda. Durante o deslocamento, os bits à esquerda são perdidos e dígitos zeros preenchem a posição direita. Exemplos:

1-) Se X = 00010101 então

X shl 2 = 01010100
X shl 5 = 10100000

2-) 55 shl 3 = 184

55 = 00110111 deslocando 3 à esquerda ficaria:
10111000 que é igual a 184

3-) $F0 shl 2 = $C0

$F0 = 11110000 deslocando 2 à esquerda ficaria:
11000000 que é igual a $C0

SHR - SHift Right

Desloca n bits à direita. Durante o deslocamento, os bits à esquerda são preenchidos com zeros e os da direita são perdidos. Exemplos:

1-) Se X = 10101100 então

X shr 3 = 00010101
X shr 6 = 00000010

2-) 55 shr 3 = 6

55 = 00110111 deslocando 3 à direita ficaria:
00000110 que é igual a 6

3-) $F0 shr 2 = $3C

$F0 = 11110000 deslocando 2 à direita ficaria:
00111100 que é igual a $3C

Observação.: *Nós operamos na base 10, porque trabalhamos com 10 algarismos, 0..9, certo? Bem na base 2 operamos somente com 2 algarismos, o 0 e o 1. Dessa forma, temos que representar todos os números da base 10 utilizando somente o 0 e 1. Parece complicado ? Nem tanto, veja abaixo a correspondência*:

BASE 10	BASE 2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

8 1000
9 1001
10 1010
11 1011
e assim por diante

Para converter um número da base 10 para a base 2, basta dividir o número, o qual 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 fazer ao contrário:

(10111) => ( )
2 10

4 3 2 1 0
(1 0 1 1 1)

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

NOT

O operador *NOT* nega os bits, isto é os bits iguais a 1 se tornam 0 e os bits zero se tornam 1. Devemos lembrar, no entanto, que os inteiros possuem 2 bytes, portanto, ao se trabalhar com números decimais inteiros
será afetado o byte de mais alta ordem e também o sinal. Exemplo:

NOT (255) = -256

Para suprimir este problema, você deve trabalhar com bytes:

```
Program Exemplo;
Uses CRT;
Var i,j : Byte;
```

```
Begin
  ClrScr;
  i:=255;
  j:=NOT(i);
  Writeln(j);     (* será escrito 0 *)
End.
```

AND

Este operador realiza a operação *E* lógico bit a bit. Relembrando, a operação *E* resulta em 1 *se e somente se* os dois operandos forem iguais a 1, caso contrário, o resultado será igual a 0. Exemplos:

1-) $0F AND $F0 = $0 pois
$0F = 00001111
$F0 = 11110000
00001111 AND 11110000 = 00000000

2-) 255 AND 55 = 55 pois
255 = 11111111
55 = 00110111
11111111 AND 00110111 = 00110111

3-) 34 AND 76 = 0 pois
34 = 00100010
76 = 01001100
00100010 AND 01001100 = 00000000

OR

Este operador realiza a operação *OU* lógico bit a bit. Relembrando, a operação ***OU*** resulta em 1 se um ou os dois operandos forem iguais a 1.
Exemplos:

1-) $0F OR $F0 = $FF pois
$0F = 00001111
$F0 = 11110000
00001111 OR 11110000 = 11111111

2-) 255 OR 55 = 255 pois
255 = 11111111
55 = 00110111
11111111 OR 00110111 = 11111111

3-) 34 OR 76 = 110 pois
34 = 00100010
76 = 01001100
00100010 OR 01001100 = 01101110

XOR

Este operador realiza a operação *OU EXCLUSIVO* lógico bit a bit. Relembrando, a operação OU EXCLUSIVO resulta em 1 se os operandos forem diferentes entre si . Exemplos:

1-) $0F XOR $F0 = $FF pois
$0F = 00001111
$F0 = 11110000
00001111 XOR 11110000 = 11111111

2-) 255 XOR 55 = 200 pois
255 = 11111111
55 = 00110111
11111111 XOR 00110111 = 11001000

3-) 34 XOR 76 = 110 pois
34 = 00100010
76 = 01001100
00100010 XOR 01001100 = 01101110

III.4.5 - Concatenação

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

'Isto é uma ' + 'string' = 'Isto é uma string'

IV - Entrada e saída de dados

IV.1 - Write e Writeln

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

Primeira forma:

Write(parâmetro_1,Parâmetro_2, ...);

Exemplo:

```
Program Exemplo;
Uses CRT;
Var     i : integer;
        r : real;
        c : char;
```

```
        s : string[20];
    Begin
      ClrScr;        (* apaga a tela e coloca o apostilar 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 resultaria 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 apostilar deve ser deslocado à direita, antes do parâmetro ser escrito. Além disso, o parâmetro é escrito da direita para a esquerda, exemplo:

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

Resultaria a seguinte tela:

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

Os pontos representam espaços em branco

Terceira forma:

Write(parâmetro : n : d);

Neste caso, n tem a mesma função que o caso anterior sendo que d representa o número de casas decimais. Obviamente, parâmetro terá queser do tipo Real. Exemplo:

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

resultaria a seguinte tela:

```
3.1415600000E+00
      3.14
```

IV.2 - Read e Readln

Estas procedures são utilizadas para fazer leitura de dados via teclado. A procedure Read lê um dado do teclado até que se pressione a tecla ENTER, sendo que cada tecla digitada é ecoada para o vídeo. Após pressionarmos ENTER, o apostilar permanecerá no mesmo lugar. Já, a procedure Readln faz a mesma coisa só que o apostilar passa para a próxima linha. A sintaxe geral para estas procedures é:

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

Ao se digitar os valores das variáveis pedidas, deve-se separá-los por espaços.

Exemplo 1:

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

Exemplo 2:

```
     Program teste;
     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);
  End.
```

Exemplo 3:

PROGRAMA AREA_DE_TRIANGULOS : calcula área de triângulos

```
Program Area_de_Triangulos;
Uses CRT;
Var Base,       (* base do triângulo   *)
   altura:     (* altura do triângulo *)
         Real;
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);
End.
```

ReadKey:

Lê uma tecla do teclado, sem que seja necessário pressionar a tecla ENTER

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

```
end.
```

IV.3 - 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.4 - Funções e procedures para controle de vídeo

IV.4.1 - ClrScr

Esta procedure tem a finalidade de limpar a tela de vídeo e colocar o apostilar na primeira coluna da primeira linha. A tela de vídeo é dividida em 80 colunas e 25 linhas. O canto superior esquerdo tem coordenadas (1,1) e o inferior direito (80,25).

IV.4.2 - Gotoxy(x,y)

Move o apostilar para a coluna x e linha y.

Exemplo:

```
Program Exemplo;
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');
End.
```

IV.4.3 - ClrEol

Esta procedure limpa desde a posição atual do apostilar até o final da linha.

IV.4.4 - CrtExit

Envia para a tela de vídeo a string de finalização definida na instalação

IV.4.5 - CrtInit

Envia para a tela de vídeo a string de inicialização definida na instalação.

IV.4.6 - Delline

Procedure que elimina a linha em que está o apostilar. As linhas posteriores sobem, ocupando a que foi eliminada.

Exemplo:

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

O programa anterior irá provocar a seguinte tela:

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

Repare que a string 'linha 2' foi eliminada.

IV.4.7 - HighVideo

Coloca o vídeo no modo normal. Esta procedure é equivalente a NormVídeo.

IV.4.8 - InsLine

Esta procedure faz exatamente o contrário de Delline, ou seja, insere uma linha na posição atual do apostilar.

Exemplo

```
Program Exemplo;
Begin
   ClrScr;
   Writeln('linha 1');
```

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

Este Programa provocará a seguinte tela

linha 1
linha 2
teste
linha 3
linha 4

IV.4.9 - LowVideo

Coloca o vídeo em baixa intensidade até que se execute a procedure NormVideo ou HighVideo.

IV.4.10 - NormVideo

O mesmo que HighVideo

IV.4.11 - TextBackGround

Esta procedure seleciona a cor do fundo sobre o qual o texto será escrito. Sua sintaxe geral é:

```
TextBackGround(cor);
```

Tabela de cores

0	Black	Preto
1	Blue	Azul
2	Green	Verde
3	Cyan	Ciano
4	Red	Vermelho
5	Magenta	Magenta
6	LightGray	Cinza-claro

Nós podemos entrar com o número ou o nome da cor em inglês

Exemplo:

```
Program Exemplo;
Uses CRT;
```

```
Begin
   ClrScr;
   WriteLn('teste');
   TextBackGround(7);
   Writeln('teste');
   TextBackGround(Brown);
   Writeln('teste');
End.
```

V.4.12 - TextColor

Esta procedure permite selecionar a cor com que o texto será impresso.

Tabela de cores

0.	Black	Preto
1.	Blue	Azul
2.	Green	Verde
3.	Cyan	Ciano
4.	Red	Vermelho
5.	Magenta	Magenta
6.	Brown	Marrom
7.	LightGray	Cinza-claro
8.	DarkGray	Cinza-escuro
9.	LightBlue	Azul-claro
10.	LightGreen	Verde-claro
11.	LightCyan	Ciano-claro
12.	LightRed	Vermelho-claro
13.	LightMagenta	Magenta-claro
14.	Yellow	Amarelo
15.	White	Branco
16.	Blink	Piscante

Exemplo:

```
Program Exemplo;
Uses CRT;
Begin
```

```
    Clrscr;
    TextBackGround(7);
    TextColor(black);
    writeln('teste');
    TextColor(black+blink);
    write('teste');
End.
```

IV.4.13 - Window

Sintaxe: Window(x1,y1,x2,y2);

Esta procedure tem o poder de definir uma janela de texto cujo canto esquerdo superior é x1,y1 e canto inferior direito é x2,y2. Após esta instrução, as instruções ClrScr, Write Writeln agem somente dentro da janela recém definida. A instrução Gotoxy passa a utilizar como referencial o ponto x1,y1 que passa a ser considerado 1,1.

Exemplo:

```
Program Exemplo;
Uses CRT;
Begin
   Window(10,10,70,20);
   ClrScr;            (* limpa somente a janela *);
   Writeln('teste');    (* escreve 'teste' em 10,10 *)
End.
```

IV.4.14 - WhereX

Função que retorna o número da coluna onde está o apostilar.

IV.4.15 - WhereY

Função que retorna o número da linha onde está o apostilar.

IV.5 - Controle do teclado

IV.5.1 - Kbd

Quando quisermos ler dados do teclado e que não sejam ecoados para o monitor de vídeo até que sejam processados e aceitos, nós podemos utilizar a seguinte sintaxe:

Read(Kbd,Variável);

No caso de números inteiros ou reais, o número só será aceito quando pressionarmos a tecla <enter>, no caso de variáveis do tipo char, o caractere será aceito sem que seja necessário pressionar a tecla <enter>, idem para o tipo string.

Exemplo:

```
Program Exemplo;
Uses CRT;
Var i:integer;
Begin
   ClrScr;
   Write('Entre com um inteiro --> ');
   Readln(Kbd,i);
   Writeln(i);
End.
```

IV.5.2 - BufLen

Buflen é uma variável interna pré-definida em Turbo Pascal cujo valor inicial é 126. Ela contém o número máximo de caracteres aceitos por Read.

Exemplo:

```
Program Exemplo;
Uses CRT;
Var i : Integer;
Begin
   ClrScr;
   Writeln(Buflen);    (* escreve 126 *)
   Buflen:=2;
   Write('Digite um inteiro --> ');
   Readln(i);          (* se você tentar digitar inteiros com mais de dois dígitos, readln não permitirá *)
End.
```

IV.5.3 - Keypressed

O identificador Keypressed é uma função especial do Turbo Pascal que retorna um valor booleano - *TRUE* se uma tecla foi pressionada, ou *FALSE* caso contrário. Ela é muito utilizada para detectar teclas pressionadas no teclado.

Exemplo

```
Program Exemplo;
Uses CRT;
Begin
   ClrScr;
   Write('Pressione uma tecla -> ');
   Repeat until Keypressed; (* repita até que uma tecla seja pressionada. O comando Repeat Until será estudado mais adiante *)
End.
```

V - Comandos para controle do fluxo do programa

V.1 - 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 serem executados, eles deverão vir delimitados pelas palavras Begin e End.

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

No caso acima, se expressão_lógica for *TRUE* então todos comandos inclusos entre as palavras *Begin* e *End* serão executados, caso contrário não.

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 **;** no final de comando_1.

Podemos também escrever:

```
If <expressão> Then Begin
                      Comando_1;
                      Comando_2;
                      . . .
                   End        (* não tem ; *)
              Else Begin
                      Comando_3;
                      Comando_4;
                      . . .
                   End;
```

Exemplos:

```
Program Exemplo_1;
Uses CRT;
Var i : Integer;
Begin
  Clrscr;
  Write('Digite um inteiro maior que 100 --> ');
  Readln(i);
  Writeln;
  Writeln;
  If (i>100)
    Then Writeln('Você conseguiu')
    Else Writeln(i,' não e maior que 100');
End.

Program Exemplo_2;

    {Programa para determinar 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;
    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');
  End.

Program Exemplo_3;

{Programa para colocar em ordem crescente 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;
  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);
End.
```

V.2 - Labels e Goto

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*.

Exemplos:

```
Program Exemplo_1;
```

{Programa para colocar em ordem crescente 3 números lidos do teclado}

```
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 Exemplo_2;
```

{Programa para determinar se 3 valores lidos do teclados são lados de um triângulo
Observações:
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.

V.3 - For

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 a palavra downto.

Exemplos:

```
Program Exemplo_1;
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 *)
End.

Program Exemplo_2;
Uses CRT;
Var i : Integer;
Begin
  ClrScr;
  For i:=10 downto 1 do Writeln(i);
End.

 Program Exemplo_3;
Uses CRT;
{ Este programa escreve na tela os quadrados dos números de 1 até 20 }
Var i : Integer;
Begin
  ClrScr;
  For i:=1 to 20 do
     Begin
       Write('Valor de i --> ');
       Write(i:3);
       Write('..... quadrado de i = ');
       Writeln(i*i:5);
     End;
End.

Program Exemplo_4;
Uses CRT;
```

{Este programa calcula a soma entre todos os números compreendidos entre dois números lidos do teclado }

```
Label INICIO;
Var i,Numero_1,Numero_2,soma : Integer;
    Tecla                    : Char;
Begin
INICIO:
   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;
   Write('Deseja Continuar ? --> ');
   tecla:=ReadKey;
   If ((Tecla = 'S') OR (Tecla='s')) Then Goto INICIO;
   ClrScr;
   Write('Tchau .........');
End.

Program Exemplo_5;
Uses CRT;
```

{Programa para cálculo de fatorial de um número lido do teclado. Lembrando que fatorial de um número é definido como segue:

N! = 1.2.3.4...N

e 0! = 1}

```
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.
```

V.4 - Repeat Until

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, 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).

Exemplos:

```
Program Exemplo_1;
Uses CRT;

   {Programa exemplo para mostrar o funcionamento da estrutura Repeat Until}

Var i : Integer;
Begin
   ClrScr;
   i:=1;
   Repeat
      Writeln(i);
      i:=i+1;
   Until i=10;
End.

Program Exemplo_2;
Uses CRT;

{Programa que soma os números pares compreendidos entre dois números lidos do teclado}

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 a-
diciona-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);
  end.

  Program Exemplo_3;
  Uses CRT;

{Programa para cálculo de fatorial.}

  Label inicio,fim;
  Var n,i,fatorial:integer;
  Begin
    ClrScr;
  inicio:
    Write('N = (menor que 0 = fim) --> ');
    Readln(n);
    If n<0 then goto fim;
    Writeln;
    fatorial:=1;
    i:=1;
    if n>1
      then Repeat
             i:=i+1;
             fatorial:=fatorial*i;
           Until i=n;
     Writeln('fatorial de ':30,n,' = ',fatorial);
     Writeln;
     goto inicio;
  fim:
  End.
```

V.5 - While Do

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 interação, isto significa que, eventualmente, pode não ocorrer sequer a primeira interação.

A sintaxe de While..Do é:

While <expressão_lógica> Do <comando>;

ou

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

Exemplos:

```
Program Exemplo_1;
Uses CRT;
{Programa exemplo que escreve na tela de 0 até 10}
Var i : Integer;
Begin
  ClrScr;
  i:=0;
  While (i<11) Do
    Begin
      Writeln(i);
      i:=i+1;
    End
End.
Program Exemplo_2;
Uses CRT;
```

{Programa que lê números do teclado e depois informa a média dos números lidos, a quantidade lida, e soma deles}

```
Label INICIO;
Const Quant_de_num : Integer = 0;
      Soma         : Real    = 0;
      Media        : Real    = 0;
Var   Numero       : Real;
      Tecla        : Char;
Begin
INICIO:
  ClrScr;
  Write('Valor numérico (menor que 0=fim) --> ');
  Readln(Numero);
```

```
While (Numero>=0) Do
  Begin
    Soma := Soma + Numero;
    Quant_de_num := Quant_de_num + 1;
    Write('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');
 Writeln;
 Write('Deseja continuar ? ---> ');
 tecla:=ReadKey;
 If (Tecla='s') Or (Tecla='S') Then Begin
                                    Quant_de_num:=0;
                                    Soma := 0;
                                    Goto Inicio;
                                   End;
End.
```

V.6 - 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 comando subsequente será executado.

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 que estiver na frente da cláusula Else será executado.

Exemplos:

```
Program Exemplo_1;
Uses CRT;
```

{Programa exemplo da instrução Case. Calcula a soma, ou a subtração, ou a multiplicação, ou a divisão entre dois números lidos do teclado}

```
    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
        '+' : 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 *)
  End. (* programa *)

  Program Exemplo_2;
  Uses CRT;

{Programa para cálculo de área de figuras}

  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;
'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;
'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;
'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;
          End;
        Until escolha='1';
      End.
```

VI - Tipos de dados estruturados

VI.1 - Introdução

Até o presente instante, nós definimos dados do tipo simples ou não estruturados, como por exemplo: Byte, Integer, Real, Char e Boolean. No entanto, existem outros tipos de dados chamados complexos ou estruturados, String é um deles. Nós já falamos sobre o tipo de dado estruturado String, por ser extremamente utilizado como já salientamos antes. Mas o Turbo Pascal possui outros tipos de estruturas, a saber:

- Array
- Record
- Set
- File
- String (já visto)

O tipo file refere-se a arquivos de discos e será amplamente estudado num capítulo à parte. Os demais serão vistos neste capítulo.

VI.2 - Array

Imagine que nós precisemos declarar 100 variáveis do tipo integer, isso poderia ser feito da seguinte forma:

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

Embora isso pareça uma brincadeira (de mal gosto), 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. Mas para quem acessa BBS, coisa de louco, então o cara pode achar MUITO LEGAL. (:-))

VI.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

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

Após a declaração acima, teríamos definidas 100 variáveis do tipo Integer, cujos nomes seriam:

i[1] - i[2] - i[3] - . . . - i[100]

Exemplo 2:

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

Var
  Arranjo_simples : Arranjo;
```

Após as declarações acima, teríamos definidas 2000 variáveis do tipo char com o nome Arranjo_simples.

Exemplos:

```
Program Exemplo_1;
Uses Crt;

{Lê 10 números inteiros do teclado e os escreve na tela ao contrário do que foram lidos}

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 Exemplo_2;
Uses CRT;
```

{Programa que lê no máximo 100 números reais do teclado e os coloca em ordem crescente}

```
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 um no. menor que 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 Exemplo_3;
Uses CRT;
```

{Programa semelhante ao anterior só que coloca em ordem crescente nomes lidos do teclado}

```
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 um nome = a 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 Exemplo_4;
Uses CRT;
```

{Programa que lê as notas de alunos de uma determinada classe e depois lista os alunos e as respectivas notas menores que 5.0}

```
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.
```

VI.2.2 - Arrays Multidimensionais

No item anterior, trabalhamos com Arrays unidimensionais, ou seja, de uma dimensão. No entanto, é possível trabalhar com arrays de mais de uma dimensão e nesses casos, eles são chamados de multidimensionais.

Exemplos:

```
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, exemplo:

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

Exemplo:

```
Program Exemplo;
Uses CRT;
```

{Programa Matriz => Tem a finalidade de ler uma matriz do teclado e em seguida multiplicar uma coluna ou linha por uma constante. Neste programa, procurei utilizar o maior número possível de conceitos dados até aqui}

```
(* 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 multiplicaçåo *)
  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;
End.
```

VI.3 - Tipo Record

VI.3.1 - Conceito de estrutura heterogênea

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.
Imaginem 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 como podemos reparar, os quatros itens são de tipos diferentes, a saber:

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

Como veremos a seguir, o tipo Record resolver-nos-á o problema.

VI.3.2 - Definição de Records

A definição de uma variável do tipo record, começa com a palavra reservada Record, a qual é seguida pelos campos (variáveis) e os seus tipos. A palavra reservada End seguida de um 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;
```

VI.3.3 - Acesso aos elementos da estrutura

Para acessarmos os elementos da estrutura, ou seja, os campos, nós devemos incluir o nome da variável seguida de um ponto e depois o nome do campo, exemplos:

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

Exemplos:

```
Program Exemplo_1;
```

```
Uses CRT;

  {Lê uma variável do tipo record do teclado e em seguida a mostra no monitor}

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 Exemplo_2;
Uses CRT;
```

{Programa para ler dados de no máximo 20 pessoas. Em seguida é feita uma listagem em ordem alfabética pelo nome}

```
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.
```

VI.3.4 - Declaração With

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 resolver o problema, 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;
```

Exemplo:

```
Program Exemplo_1;
Uses CRT;

                    { lê uma variável tipo record e em seguida a mostra }

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.
```

VI.4 - Tipo Set

VI.4.1 - Definição e declaração

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.
Pois bem, na linguagem Pascal, também podemos utilizar estes conceitos. Na linguagem Pascal, um conjunto é uma coleção de elementos semelhantes. O tamanho do conjunto pode ser variável, sendo que no caso específico do Turbo Pascal, o conjunto pode ter 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*.
Em Pascal, 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
```

Declaração

A forma geral para definição 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;

                etc.
```

VI.4.2 - Operações em tipos Set

Atribuição: (**:=**)

O operador de atribuição é o mesmo utilizado para tipos simples, 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.

Exemplo:

```
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.

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

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.

Exemplos de programas:

```
Program Exemplo_1;
```

```
Uses CRT;

        {Lê uma tecla e a envia para o monitor até que se digite  'S' ou 's' ou 'N' ou 'n'}

Var tecla : Char;

Begin
  ClrScr;
  Repeat
    Read(kbd,tecla);
    Write(tecla);
  Until tecla IN ['s','S','n','N'];
End.

Program Exemplo_2;
Uses CRT;

   {lê uma tecla e diz se é número, letra maiúscula ou letra minúscula até que se leia um '?'}

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 Exemplo_3;
Uses CRT;
{Programa que conta o número de vogais, número de consoantes e de brancos numa frase lida do
                                    teclado}

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 número de caracteres que o
            parâmetro tem *)
   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.
```

VII - Procedures

VII.1 - Definição

Uma das técnicas mais utilizadas e tida como vantajosa na confecção 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 módulos, esse módulo recebe o nome de programa principal, enquanto que os outros são chamados de subprogramas. No sistema Turbo Pascal, existem dois tipos de subprogramas, a saber:

- Procedures (procedimentos)
- Functions (funções)

A procedure é como se fosse um programa. Ela tem a estrutura praticamente igual a de um programa, como veremos mais adiante. A procedure deve ser ativada (chamada) pelo programa principal ou por uma outra procedure, ou até por ela mesma.

VII.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, 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.

Exemplo:

```
Program Exemplo_1;  (* 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 *)
      End.
```

O programa acima, pura e simplesmente faz o seguinte:

1-) Apaga a tela e coloca o apostilar em 1,1
2-) Ativa a procedure linha
3-) Escreve a palavra teste
4-) Ativa novamente a procedure linha.

Por sua vez, a procedure linha traça uma linha a partir da posição atual do apostilar. Uma observação importantíssima 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.

VII.3 - Passagem de parâmetros

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

Exemplo:

```
Program Exemplo;
      Uses CRT;
      Var i,j:integer;
      Procedure soma(x,y:integer);
      Begin
        writeln(x+y);
```

```
end;
Begin
  ClrScr;
  soma(3,4);
  i:=45;
  j:=34;
  soma(i,j);
end.
```

Como podemos reparar, a procedure soma depende de dois parâmetros inteiros, e ao ativarmos esta procedure, devemos fornecer os dois parâmetros. Esses parâmetros podem ser dois números inteiros ou duas variáveis inteiras, obviamente deve haver compatibilidade entre os parâmetros passados. Podemos também passar parâmetros de tipos diferentes, senão vejamos:

```
Program Exemplo_1;
Uses CRT;
Var i,j:integer;
Procedure soma(x,y:integer;h,g:real);
Begin
  writeln(x+y);
  writeln(h/g:10:2);
end;
Begin
  ClrScr;
  i:=34;
  j:=35;
  soma(i,j,3.4,4.5);
End.
```

{Nos exemplos acima, houve passagem de parâmetros para as procedures, mas elas também podem passar dados de volta para o programa chamador, exemplo:}

```
Program exemplo;
Uses CRT;
Var i : Integer;
Procedure Soma(x,y:Integer;Var z:Integer);
Begin
  z:=x+y;
End;
Begin
  ClrScr;
  Soma(3,4,i);
  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 parâmetro z é passado para i.

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

VII.4 - A declaração forward

Suponha o programa abaixo:

```
Program exemplo;
Uses CRT;
Procedure Soma(x,y:Integer);
Begin
  linha;
  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. No entanto, a procedure linha está declarada mais à frente e portanto, ao compilarmos o programa, o compilador irá "reclamar" que não conhece o identificador *Linha* e com justa razão, isto 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.

Exemplo:

```
Program exemplo;
  Uses CRT;
  Procedure Linha; Forward;
  Procedure Soma(x,y:Integer);
  Begin
    linha;
    Writeln(x+y);
  End;
  Procedure Linha;
  Var i:integer;
  Begin
    For i:=1 to 80 do Write('-');
```

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

Agora sim, podemos compilar o programa sem erro.

VII.5 - O escopo de objetos num programa

Reparem o Exemplo abaixo:

```
Program Exemplo;
    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 Sub_teste;        (* a procedure
                                Sub_teste é local
                                à procedure Teste *)
    Begin
      Write('Estive em sub_teste');
    End;
    Begin
      Sub_teste;
      Writeln;
    End;
    Begin
      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 o próprio programa. O espaço para tais elementos é criado durante a compilação. Já,

os elementos declarados dentro de uma procedure, só existem dentro da procedure, exemplo: 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, se repararmos bem no exemplo, veremos 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. Mas 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.

VIII - Functions.

VIII.1 - Definição

As funções são muito parecidas com as procedures. A principal diferença é que o identificador de uma função assume o valor de retorno da função. Uma função deve sempre retornar um valor e em Turbo Pascal, este valor é retornado no nome da função.

VIII.2 - 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 Nome_da_função(parâmetros) : Tipo_da_função;
    < área de declarações >
    Begin
        corpo da função
    End;
```

A formação do nome da função deve seguir as mesmas regras para formação 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 Procedures e Functions. Devemos lembrar que tais elementos só poderão ser utilizados dentro do corpo da função, pois são locais a ela. Abaixo, temos o exemplo de uma função.

```
Program Exemplo;
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;                   (* reparem que o valor da funçåo é retornado p. seu nome *)
End;
Begin
  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.
```

Devemos lembrar que o Turbo Pascal possui inúmeras funções de procedures pré-definidas, que iremos ver no decorrer do apostila.

Exemplos:

```
Program Fat;
Uses CRT;
{Programa para calcular o fatorial de um número lido do teclado, usando o conceito de Function}
Label inicio,fim;
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;
inicio:
  Write('Valor de n (menor que 0 = fim) --> ');
  Readln(n);
  Writeln;
  If n<0
    Then Begin
           Write('Não existe fatorial de numeros negativos');
           Goto fim;
         End
```

```
      Else Writeln('Fatorial de n = ',fatorial(n):10:0);
   Writeln;
   Goto inicio;
Fim:
End. (* do programa *)
```

```
Program Fibonacci;
Uses CRT;
```

{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)}

{Como podemos ver, o elemento atual é determinado pela soma dos dois elementos anteriores}

```
Label inicio;
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;
Begin
   ClrScr;
inicio:
   Write('Fib(');
   Read(numero);
   Writeln(') = ',fib(numero));
```

```
  Writeln;
  Write('Deseja continuar ? --> ');
  Readln(tecla);
  writeln;
  writeln;
  If tecla='s' Then goto inicio;
End.
```

VIII.3 - Recursividade

A linguagem Pascal e o Turbo Pascal permitem 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, só que utilizando o conceito de recursividade.

```
Program Fatorial;
Uses CRT;
Label inicio,fim;
Var n : Integer;
    tecla : char;
Function Fat(n:integer):real;
Begin
  if n=0
    Then Fat:=1
    Else Fat:=n*Fat(n-1);   (* repare que estamos chamando novamente a funçåo Fat *)
End;
Begin
  ClrScr;
inicio:
  Write('Valor de n (menor que 0 = fim) --> ');
  Readln(n);
  Writeln;
  If n<0
    Then Begin
           Write('Não existe fatorial de números negativos');
           Goto fim;
         End
    Else Writeln('Fatorial de n = ',fat(n):10:0);
  Writeln;
  Goto inicio;
Fim:
End.


Program Fibonacci;
Uses CRT;
```

```
Label inicio;
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
   ClrScr;
inicio:
   Write('Fib(');
   Read(numero);
   Writeln(') = ',fib(numero));
   Writeln;
   Write('Deseja continuar ? --> ');
   Readln(tecla);
   writeln;
   writeln;
   If tecla='s' Then goto inicio;
End.
```

IX - Arquivos em disco.

IX.1 - O tipo File

O tipo file ou arquivo, é uma estrutura constituída de elementos do mesmo tipo dispostos seqüencialmente. Essa estrutura é utilizada para comunicação com o meio externo, principalmente com discos magnéticos.

IX.1.1 - Definição do tipo File

A sintaxe geral para definir uma variável com esse tipo de estrutura é:

```
Type Arquivo = File of <Tipo>;

Var a : Arquivo
```

Após as declarações acima, a variável 'a' passa a representar um arquivo de elementos do tipo <Tipo>.

Exemplos:

Exemplo 1: Arquivo com números inteiros:

```
Type Arq = File Of Integer;
Var  Arquivo : Arq;
```

Ou

```
Var Arquivo : File Of Integer;
```

Exemplo 2: Arquivo de números reais:

```
Type Arq = File Of Real;
Var  Arquivo : Arq;
```

Exemplo 3: Arquivo de records:

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

Var Arquivo : File Of Pessoa;
```

e assim por diante...

IX.2 - Procedimentos para operações em arquivos

O acesso a arquivos sempre segue a mesma seqüência, a saber:

1-) Abertura do arquivo
2-) Leitura e/ou escrita de dados no arquivo
3-) Fechamento do arquivo

Para tanto, existem diversas procedures para executar tais operações e que passaremos a examinar agora.

IX.2.1 - Assign

Esta procedure tem a finalidade de atribuir um nome lógico ao arquivo físico, ou seja, ao nome do arquivo em disco. Sintaxe:

```
Assign(Variável_do_tipo_file,Nome_do_arquivo);
```

Exemplo:

```
Program Exemplo;
  Uses CRT;
  Type Arquivo = File Of Integer;
  Var Arq : Arquivo;
```

```
        Begin
            Assign(Arq,'B:EXEMPLO.DTA');
```

(a partir desse instante, todas as operações de escrita ou leitura que forem realizadas com a variável Arq, será automaticamente feitas no arquivo EXEMPLO.DTA no drive B *)*

```
            . . .
            . . .
    End.
```

IX.2.2 - Abertura de arquivos (Rewrite e Reset)

Para abrir arquivos, dispomos de duas procedures, a saber:

Rewrite(<Arq>);

Esta procedure apaga o arquivo em disco associado à variável Arq e cria um novo arquivo.
Exemplo:

```
        Program Exemplo;
        Uses CRT;
        Type Arquivo = File Of Integer;
        Var Arq : Arquivo;
        Begin
            Assign(Arq,'B:EXEMPLO.DTA');
            Rewrite(Arq);
            (* Após estas declarações, teremos um novo arquivo no drive  B com o nome
'EXEMPLO.DTA' *)
            . . .
            . . .
        End.
```

Reset(Arq);

{Esta procedure abre o arquivo em disco associado à variável Arq para leitura ou escrita. Esta procedure parte do princípio que o arquivo exista em disco, caso ele não exista, haverá erro. }

Exemplo:

```
        Program Exemplo;
        Uses CRT;
        Type Arquivo = File Of Integer;
        Var Arq : Arquivo;
        Begin
            Assign(Arq,'B:EXEMPLO.DTA');
            Reset(Arq);
```

```
(* Após estas declarações, o arquivo no drive B com o nome 'EXEMPLO.DTA' está aberto e pronto
para as operações de entrada e saída *)
                . . .
                . . .
        End.
```

IX.2.3 - Escrevendo e lendo dados no arquivo (Write,Read)

A procedure utilizada para escrever dados em um arquivo é Write. Sua sintaxe é:

Write(Arq,var);

Os dados são gravados seqüencialmente no arquivo, ou seja, um após o outro e isto é feito automaticamente pela procedure Write. Para tanto a linguagem Pascal mantém um apontador de registro de arquivo que aponta sempre para o número de registro, onde será gravado ou lido um dado.

Exemplo:

```
Program Exemplo;
    Uses CRT;
    Type Arquivo = File Of Integer;
    Var Arq : Arquivo;
       i   : Integer;
    Begin
        Assign(Arq,'B:EXEMPLO.DTA');
        Rewrite(Arq);
        For i:=1 to 100 do Write(Arq,i);
(* com a instrução acima, teríamos escrito seqüencialmente no arquivo B:EXEMPLO.DTA os números de 1 a 100 *)
                . . .
                . . .
        End.
```

Como já dissemos anteriormente, a linguagem Pascal mantém um apontador de registro que indica o próximo registro que será lido ou escrito, e toda vez que fazemos uma leitura ou escrita num registro, o apontador é incrementado de um, isto é automático. No entanto, dispomos de uma procedure que nos permite alterar o valor desse apontador e portanto, nos permite acessar qualquer registro que quisermos. Essa procedure chama-se *Seek*. A propósito, o número do primeiro registro é zero. A sintaxe desta procedure é:

```
Seek(Arq,número_do_registro);
```

Para ler dados do arquivo,dispomos da procedure Read cuja sintaxe é:

```
Read(Arq,Var);
```

Exemplo:

```
Program Exemplo;
        Uses CRT;
        Type Arquivo = File Of Integer;
        Var Arq : Arquivo;
            i   : Integer;
        Begin
              Assign(Arq,'B:EXEMPLO.DTA');
              Rewrite(Arq);
              For i:=1 to 100 do Write(Arq,i);
              Seek(Arq,0);  (* posiciona o apontador de registro no registro número 0 *)
              Read(Arq,i);  (* a variável i fica igual ao conteúdo do registro número 0 que no presente exemplo valeria 1,a propósito,
                                 o apontador de registro já está valendo 1 *)
              Read(Arq,i);  (* i agora está valendo 2 *)
              . . .
              . . .
        End.
```

IX.2.4 - Fechamento do arquivo

Como já foi visto, após a abertura do arquivo, leitura e/ou escrita de dados, devemos fechá-lo. Para tanto, dispomos da procedure close cuja sintaxe é:

Close(Arq);

Exemplos de programas:

Exemplo 1:

PROGRAMA Grava --> Lê números do teclado e em seguida os grava num arquivo em disco AUTOR : Thelmo J. M. Mesquita

```
Program grava;
Uses CRT;
Var arquivo : File Of Integer;
    i       : Integer;
Begin
  ClrScr;
  Assign(arquivo,'arquivo.dta');
  ReWrite(arquivo);
  Repeat
    Write('Numero --> ');
    Readln(i);
    Write(arquivo,i);
  Until i=0;
  Close(arquivo);
```

```
End.

  (* O próximo programa lê os números gravados pelo programa anterior *)

PROGRAMA : Le.pas ---> Le numeros de um arquivo em disco
  AUTOR    : Thelmo J. M. Mesquita

Program le;
Uses CRT;
Var arquivo : File Of Integer;
    i       : Integer;
Begin
  ClrScr;
  Assign(arquivo,'arquivo.dta');
  Reset(arquivo);
  Repeat
    Read(arquivo,i);
    Writeln(i);
  Until i=0;
  Close(arquivo);
End.
```

Exemplo 2:

```
Program Exemplo_2;
    Uses CRT;
{Programa que grava num arquivo em disco, o quadrado dos números de 0 a 100 e depois permite consulta através da instrução seek}

    Var Arq : File of Real;
        i : Integer;
        s : real;
    Begin
      Assign(Arq,'Arquivo.dta');
      Rewrite(Arq);
      For i:=0 to 100 do Begin
                           s:=i*i;
                           Write(Arq,s);
                         End;
      Close(Arq);
      Reset(Arq);
      ClrScr;
      While i>=0 do
      Begin
        Write('Numero --> ');
        Readln(i);
        if (i>=0) And (i<=100)
          Then Begin
```

```
                seek(Arq,i);
                Read(Arq,s);
                Writeln;
                Writeln(s:10:0);
                Writeln;
              End;
  End;
  Close(Arq);
End.
```

Exemplo 3:

ARQUIVO.PAS : Este programa tem a finalidade de gerenciar um arquivo em disco cujos registros contém dois campos, a saber:

NOME : 20 caracteres
IDADE : integer

O programa apresenta inicialmente o seguinte menu:

1 -) Sair do programa
2 -) Entrar com registros
3 -) Listar todos os registros
4 -) Pesquisa por nome

($I-*) (* esta diretiva de compilação tem a finalidade de indicar ao compilador que os erros de I/O (entrada/saída) serão verificados pelo programador, ou seja, se houver algum erro de I/O durante a execução do programa, o programa não irá abortar. Para que o programador saiba se uma determinada operação de I/O funcionou corretamente, ele deverá verificar o valor da variável IORESULT. Se ela for diferente de zero, então ocorreu algum erro e o programador deverá então tomar alguma providência *)*

```
Program Exemplo_De_Arquivo;
Uses CRT;
Type Pessoa = Record
                Nome   : String[20];
                Idade  : Integer;
              End;
     Frase  = String[80];
Var  Arquivo    : File Of Pessoa;
     P          : Pessoa;
     escolha    : Integer;
Procedure Linha; (* traça uma linha na posição atual do
                    apostilar *)
Var i : Integer;
Begin
   For i:=1 to 80 Do Write('-');
End;
Procedure Centro(S:Frase); (* centra string S na tela *)
Var x:integer;
Begin
```

```
    x:=40+(Length(S)) DIV 2; (* lenght retorna o número de
                                caracteres do parâmetro *)
    Writeln(S:x);
  End;
  Procedure InReg; (* procedimento p/ incluir registros *)
  Var resposta:char;
  Begin
    ClrScr;
    Linha;
    Centro('ROTINA PARA ENTRAR REGISTROS');
    Reset(arquivo);
  (*Neste trecho do programa, iremos utilizar uma função nova : FILESIZE(arq) retorna quantos registros possui o arquivo "arq"     *)
    Seek(arquivo,FileSize(arquivo));  (* posiciona o apontador de registros no final do arquivo *)
    resposta:='s';
    Linha;
    While resposta='s' Do
    Begin
      gotoxy(1,5);clreol; (* limpa até final da linha *)
      gotoxy(1,6);clreol;
      gotoxy(1,5);
      Buflen:=20; (* estou limitando o buffer do teclado  em 20 caracteres, o normal é 126 *)
      Write('Nome da pessoa ---> ');
      Readln(P.Nome);
      Buflen:=2;
      clreol;
      Write('Idade da pessoa --> ');
      Readln(P.Idade);
      Linha;
      Write(arquivo,P);
      Write('Deseja Continuar ? -->':50);
      Readln(resposta);
    end;
    close(arquivo);
    Buflen:=126;
  End;
  Procedure LiReg; (* procedimento para listar os registros na tela *)
  Begin
    Reset(arquivo);
    Clrscr;
    Linha;
    writeln('NOME':15,'IDADE':18);
    linha;
    While not eof(arquivo) do
    Begin
      read(arquivo,P);
```

```
      Writeln(P.nome:21,' - - - ',P.idade);
    end;
    Linha;
    Close(arquivo);
    Write('Digite uma tecla --> ');
    repeat until keypressed;
  End;
  Procedure PeNo;  (* pesquisa por nome *)
  Var nome : string[20];
  Begin
    Reset(arquivo);
    nome:='1';
    While nome<>'0' Do
    Begin
      Clrscr;
      Linha;
      Centro('PESQUISA POR NOME');
      linha;
      Write('Nome (0=fim) --> ');
      Readln(nome);
      if nome<>'0'
      Then Begin
             linha;
             seek(arquivo,0);
             While not eof(arquivo) do
             Begin
               read(arquivo,P);
               if Pos(nome,P.nome)<>0
                  Then Writeln(P.nome:21,' - - - ',P.idade);
             End;
             Linha;
             Write('Digite uma tecla --> ');
             repeat until keypressed;
           End;
    End;
    close(arquivo);
  End;
```

(* aqui começa o nosso programa, inicialmente devemos verificar se o arquivo "arquivo.dta" existe, se não existir, então ele deverá ser criado *)

```
  Begin
    Assign(arquivo,'arquivo.dta');
    Reset(arquivo);
    If IOresult <> 0 Then ReWrite(arquivo);
    Close(arquivo);
    Repeat
      ClrScr;
```

```
      Linha;
      Writeln('..... Programa para gerenciar um arquivo contendo nomes e');
      Writeln('      idades de pessoas');
      Writeln('..... Escrito em 06/09/93 por Thelmo J.M.Mesquita');
      Linha;
      Gotoxy(24,12);Writeln('1 - Sair do programa');
      Gotoxy(24,14);Writeln('2 - Entrar com registros');
      Gotoxy(24,16);Writeln('3 - Listar todos os registros');
      Gotoxy(24,18);Writeln('4 - Pesquisar por nome');
      Gotoxy(33,10);LowVideo;
      Writeln('SUA ESCOLHA :');NormVideo;
      Repeat
        Gotoxy(47,10);
        read(escolha);
      Until (escolha > 0 ) and (escolha < 5);
      Case escolha of
         2 : InReg;
         3 : LiReg;
         4 : PeNo;
      end;
   Until escolha=1;
   ClrScr;
   Gotoxy(33,12);Writeln('T C H A U . . . . . ');
End.
```

Exemplo 4:

{Este programa tem a finalidade de gerenciar um arquivo em disco com a seguida estrutura:}

```
            Nome        : frase;
            Idade       : Integer;
            Sexo : Char;
            Altura      : Real;
Program Arquivo;
Uses CRT;
Type Frase  = string[20];
     Pessoa = Record
                Nome   : frase;
                Idade  : Integer;
                Sexo   : Char;
                Altura : Real;
              End;
Var Arq : File Of Pessoa;
    escolha : char;
    p : pessoa;
    s : frase;
Procedure tecla;
```

```
Begin
  Write(chr(7));
  Write('Digite uma tecla --> ');
  Repeat until keypressed;
End;
Procedure Linha;
Var i:byte;
Begin
  For i:=1 to 80 do write('-');
End;
Function Maiuscula(s:frase):frase;
var i:byte;
Begin
  for i:=1 to length(s) do s[i]:=upcase(s[i]);
  maiuscula:=s;
end;
Function Acha_Nome(s:frase):integer;
Label fim;
Begin
  Acha_Nome:=-1;
  While not eof(arq) do
    Begin
      Read(arq,p);
      if pos(s,p.nome) > 0 Then Begin
                                  Acha_Nome:=Filepos(arq)-1;
                                  Goto fim;
                                End;
    End;
fim:
End;
Procedure Consulta;
Var escolha : Char;
Procedure lireg;
Begin
  Seek(Arq,0);
  ClrScr;
  Linha;
  lowvideo;
  Writeln('NOME':18,'IDADE':12,'SEXO':5,'ALTURA':10);
  Normvideo;
  linha;
  While not eof(arq) do
  Begin
    Read(arq,p);
    With p do
      Writeln(nome:22,idade:6,sexo:5,altura:10:2);
  End;
  linha;
```

```
   tecla;
End;
Procedure peno;
label fim;
Begin
   Repeat
      clrscr;
      write('Nome para pesquisa (0=fim) -> ');
      readln(s);
      s:=maiuscula(s);
      if s='0' then goto fim;
      Seek(Arq,0);
      ClrScr;
      Linha;
      lowvideo;
      Writeln('NOME':18,'IDADE':12,'SEXO':5,'ALTURA':10);
      Normvideo;
      linha;
      While not eof(arq) do
      Begin
         Read(arq,p);
         if pos(s,p.nome)>0 then
         With p do
            Writeln(nome:22,idade:6,sexo:5,altura:10:2);
      End;
      linha;
      tecla;
fim:
   until s='0';
End;
Procedure lidade;
label fim;
var i1,i2:byte;
Begin
   Repeat
      clrscr;
      write('Idade no.1 (0=fim) -> ');
      readln(i1);
      if i1=0 then goto fim;
      write('Idade no.2 ---------> ');
      readln(i2);
      Seek(Arq,0);
      ClrScr;
      Linha;
      lowvideo;
      Writeln('NOME':18,'IDADE':12,'SEXO':5,'ALTURA':10);
      Normvideo;
      linha;
```

```
      While not eof(arq) do
      Begin
        Read(arq,p);
        if ((p.idade>=i1) and (p.idade<=i2)) then
        With p do
          Writeln(nome:22,idade:6,sexo:5,altura:10:2);
      End;
      linha;
      tecla;
fim:
  until i1=0;
End;
Procedure lisexo;
label fim;
var s:char;
Begin
  Repeat
    clrscr;
    write('Sexo para pesquisa (0=fim) -> ');
    readln(s);
    s:=maiuscula(s);
    if s='0' then goto fim;
    Seek(Arq,0);
    ClrScr;
    Linha;
    lowvideo;
    Writeln('NOME':18,'IDADE':12,'SEXO':5,'ALTURA':10);
    Normvideo;
    linha;
    While not eof(arq) do
    Begin
      Read(arq,p);
      if p.sexo=s then
      With p do
        Writeln(nome:22,idade:6,sexo:5,altura:10:2);
    End;
    linha;
    tecla;
fim:
  until s='0';
End;
Procedure lialtura;
label fim;
var i1,i2:real;
Begin
  Repeat
    clrscr;
    write('Altura no.1 (0=fim) -> ');
```

```
      readln(i1);
      if i1=0 then goto fim;
      write('Altura no.2 ---------> ');
      readln(i2);
      Seek(Arq,0);
      ClrScr;
      Linha;
      lowvideo;
      Writeln('NOME':18,'IDADE':12,'SEXO':5,'ALTURA':10);
      Normvideo;
      linha;
      While not eof(arq) do
      Begin
        Read(arq,p);
        if ((p.altura>=i1) and (p.altura<=i2)) then
        With p do
          Writeln(nome:22,idade:6,sexo:5,altura:10:2);
      End;
      linha;
      tecla;
fim:
  until i1=0;
End;
Begin
  Repeat
    ClrScr;
    Gotoxy(32,3);LowVideo;Write('MENU DE CONSULTA');NormVideo;
    Gotoxy(23, 6);Write('1 - Voltar ao menu anterior');
    Gotoxy(23,8);Write('2 - Listar todos os registros na tela');
    Gotoxy(23,10);Write('3 - Pesquisa por nome');
    Gotoxy(23,12);Write('4 - Listar Registros  de  pessoas com');
    Gotoxy(27,13);Write('certa idade');
    Gotoxy(23,15);Write('5 - Listar Registros  de  pessoas  de');
    Gotoxy(27,16);Write('determinado sexo');
    Gotoxy(23,18);Write('6 - Listar registros  de  pessoas  de');
    Gotoxy(27,19);Write('certa altura');
    Gotoxy(32,21);Write('SUA ESCOLHA -> ');
    Repeat
      escolha:=readkey;
    Until escolha IN ['1','2','3','4','5','6'];
    Write(chr(7));
    Case escolha of
      '2' : lireg;
      '3' : peno;
      '4' : lidade;
      '5' : lisexo;
      '6' : lialtura;
    End;
```

```
  Until escolha='1';
End;
Procedure Altera;
Var escolha : Char;
Procedure Inreg;
Label fim;
Begin
  seek(Arq,filesize(Arq));
  ClrScr;
  Gotoxy(25,6);LowVideo;
  Write('INTRODUCAO DE UM NOVO REGISTRO');NormVideo;
  Gotoxy(5, 9);Write('NOME (0=fim) --> ');
  Gotoxy(5,11);Write('IDADE ---------> ');
  Gotoxy(5,13);Write('SEXO ----------> ');
  Gotoxy(5,15);Write('ALTURA --------> ');
  Repeat
    Gotoxy(26, 9);write('.....................');
    Gotoxy(26,11);write('...');
    Gotoxy(26,13);write('.');
    Gotoxy(26,15);write('....');
    Gotoxy(26, 9);Read(p.nome);
    If p.nome='0' Then Goto Fim;
    Gotoxy(26,11);Read(p.idade);
    Gotoxy(26,13);Read(p.sexo);
    Gotoxy(26,15);Read(p.altura);
    p.nome:=maiuscula(p.nome);
    p.sexo:=maiuscula(p.sexo);
    Write(Arq,p);
Fim:
  Until p.nome='0';
End;
Procedure Delreg;
label fim;
var r,i:integer;
   resp,resposta:char;
   temp:file of pessoa;
Begin
  seek(arq,0);
  Repeat
    ClrScr;
    Gotoxy(25,2);LowVideo;
    Write('ROTINA PARA DELETAR REGISTROS');NormVideo;
    Gotoxy(10,6);Write('Nome (0=fim) --> ');
    Readln(s);
    s:=maiuscula(s);
    if s='0' then goto fim;
    repeat
      r:=acha_nome(s);
```

```
          if r=-1
          Then Begin
                Gotoxy(40,23);
                lowvideo;
                write('FIM DE ARQUIVO. . ');
                normvideo;
                seek(arq,0);
                tecla;
              End
          Else Begin
                gotoxy(10, 6);clreol;Write('NOME ...: ',p.nome);
                gotoxy(10, 8);clreol;Write('IDADE ..: ',p.idade);
                gotoxy(10,10);clreol;write('SEXO ...: ',p.sexo);
                gotoxy(10,12);clreol;write('ALTURA .: ',p.altura:6:2);
                gotoxy(1,16) ;clreol; write('POSSO DELETAR -->');
                Readln(resposta);
                resposta:=maiuscula(resposta);
                if (resposta='S')
                  Then Begin
                        assign(temp,'tempor');
                        rewrite(temp);
                        seek(arq,0);
                        while not eof(arq) do
                          if filepos(arq)<>r
                            then begin
                                   read(arq,p);
                                   write(temp,p);
                                 end
                             else read(arq,p);
                        close(arq);
                        close(temp);
                        erase(arq);
                        rename(temp,'dados.dta');
                        reset(arq);
                      end
                  else Begin
                        gotoxy(1,16);clreol;
                        write('CONTINUA A PESQUISA ? --> ');
                        readln(resp);
                        resp:=maiuscula(resp);
                        if  (resp='N')
                          Then r:=-1;
                       End;
              end;
       until r=-1;
fim:
   Until s='0';
End;
```

```
Procedure Modreg;
label fim;
var r,i:integer;
    resp,resposta:char;
    temp:file of pessoa;
Begin
   seek(arq,0);
   Repeat
      ClrScr;
      Gotoxy(25,2);LowVideo;
      Write('ROTINA PARA MODIFICAR REGISTROS');NormVideo;
      Gotoxy(10,6);Write('Nome (0=fim) --> ');
      Readln(s);
      s:=maiuscula(s);
      if s='0' then goto fim;
      repeat
        r:=acha_nome(s);
        if r=-1
        Then Begin
               Gotoxy(40,23);
               lowvideo;
               write('FIM DE ARQUIVO. . ');
               normvideo;
               seek(arq,0);
               tecla;
             End
        Else Begin
               gotoxy(10, 6);clreol;Write('NOME ...: ',p.nome);
               gotoxy(10, 8);clreol;Write('IDADE ..: ',p.idade);
               gotoxy(10,10);clreol;write('SEXO ...: ',p.sexo);
               gotoxy(10,12);clreol;write('ALTURA .: ',p.altura:6:2);
               gotoxy(1,16) ;clreol; write('MODIFICA ? ----->');
               Readln(resposta);
               resposta:=maiuscula(resposta);
               if (resposta='S')
                  Then Begin
                         gotoxy(20, 6);
                         read(p.nome);clreol;
                         gotoxy(20, 8);
                         read(p.idade);clreol;
                         gotoxy(20,10);read(p.sexo);
                         clreol;
                         gotoxy(22,12);
                         read(p.altura);clreol;
                         p.nome:=maiuscula(p.nome);
                         p.sexo:=maiuscula(p.sexo);
                         seek(arq,r);
                         write(arq,p);
```

```
                    end
                else Begin
                       gotoxy(1,16);clreol;
                       write('CONTINUA A PESQUISA ? --> ');
                       readln(resp);
                       resp:=maiuscula(resp);
                       if  (resp='N')
                         Then r:=-1;
                      End;
            end;
      until r=-1;
fim:
  Until s='0';
End;
Begin
  Repeat
    Clrscr;
    Gotoxy(27,10);Write('1 - Voltar ao menu anterior');
    Gotoxy(27,12);Write('2 - Entrar com um registro');
    Gotoxy(27,14);Write('3 - Deletar um registro');
    Gotoxy(27,16);Write('4 - Modificar um registro');
    Gotoxy(31,7);Lowvideo;Write('MENU DE ALTERACAO');NormVideo;
    Gotoxy(32,19);Write('SUA ESCOLHA -> ');
    Repeat
      escolha:=readkey;
    Until escolha IN ['1','2','3','4'];
    Write(chr(7));
    Case escolha of
      '2' : Inreg;
      '3' : Delreg;
      '4' : Modreg;
    End;
  Until escolha='1';
End;
Begin
  Assign(Arq,'dados.dta');
  (*$I-*)
  Reset(Arq);
  If IORESULT <> 0 Then Rewrite(Arq);
  (*$I+*)
  Repeat
    ClrScr;
    Gotoxy(29,10);Write('1 - Sair do programa');
    Gotoxy(29,12);Write('2 - Consulta de dados');
    Gotoxy(29,14);Write('3 - Alteracao de dados');
    Gotoxy(33,7);LowVideo;Write('MENU PRINCIPAL');NormVideo;
    Gotoxy(32,17);Write('SUA ESCOLHA -> ');
    Repeat
```

```
      escolha:=readkey;
    Until escolha IN ['1','2','3'];
    Write(chr(7));
    Case escolha of
      '2': Consulta;
      '3': Altera;
    End;
  Until escolha='1';
  Close(Arq);
  ClrScr;
End.
```

IX.2.5 - Erase

Esta procedure permite deletar um arquivo em disco. Sintaxe:

```
Erase( Arq : File of tipo);
```

ou

```
Erase( Arq : File );
```

Exemplo:

```
Program Exemplo;
    Uses CRT;
    Var arq : file;
    Begin
      assign(arq,'thelmo.001');
      erase(arq);
        (* após a execução deste trecho de programa, o arquivo 'thelmo.001' seria elimina-
do do disco *)
    End.
```

IX.2.6 - Rename

Procedure utilizada para trocar o nome de um arquivo. Sintaxe:

```
Rename( Arq : File , Novo_Nome);
```

onde
Arq deve ser uma variável do tipo file e
Novo_nome uma string.

Exemplo:

```
Program Exemplo_2;
    Uses CRT;
    Var Arq : File;
    Begin
      Assign(Arq,'teste.001');
```

```
        Rename(Arq,'soma.dta');
(* após a execução deste trecho de programa, o arquivo 'thelmo.001' teria seu nome trocado para 'soma.dta' *)
        End.
```

IX.2.7 - BlockWrite e BlockRead

A procedure BlockRead lê um no. especificado de blocos de 128 bytes de um arquivo não tipado para uma variável. O n.º de registros lidos é retornado numa variável inteira que é opcional. Block-Write Escreve ao invés de ler. Sintaxe:

```
BlockWrite(Arquivo,Variável,No_de_Regs,Resultado);

BlockRead(Arquivo,Variável,No_de_Regs,Resultado);
```

Exemplo:

```
Program Exemplo;
    Uses CRT;

{Programa para copiar um arquivo para outro, em seguida é feita uma verificação se a cópia foi bem feita}

    Const Buf_Regs = 100;   (* Número de blocos de 128 bytes que serão transferidos pelo BlockRead ou pelo Block-Write *)
    Label FIM;
    Var Fonte,              (* Nome do arquivo fonte    *)
      Destino               (* Nome do arquivo destino  *)
            : String[33];
      F,                    (* Nome lógico do arq. fonte *)
      D                     (* Nome lógico do arq. destino *)
            : File;  (* Arquivos não tipados  *)
      No_Regs_restantes,  (* No. de  registros que faltam  para serem transferidos *)
      Regs_para_ler,      (* No. de registros que serão lidos e/ou escritos *)
      i,r,r1                (* Variáveis auxiliares *)
            : Integer;
      Buffer,               (* Variável que receberá o blocos de registros lidos pelo Block-Read *)
      Buffer1               (* Idem ao anterior *)
            : Array[1..12800] Of Byte;
    Procedure Erro(x:integer);
    Begin
      Writeln('. . . . . Problemas com a copia');
      If x=1
        Then Writeln('. . . . . Arquivos de tamanhos diferentes')
        Else Writeln('. . . . . .Arquivos diferentes');
      Writeln('Tente novamente');
    End;
    Begin
```

```
  ClrScr;
  Lowvideo;
  Writeln('Copiador de arquivos':50);
  NormVideo;
  Write('Fonte ----> ');
  Readln(Fonte);
  Assign(F,Fonte);
{$I-}                 (* já explicado em programa anterior *)
  Reset(F);
{$I+}
  If IORESULT <> 0
    Then Begin
           Writeln('..... Este arquivo não existe');
           Writeln('..... Operacao nao realizada');
           Goto FIM;
         End;
  Write('Destino --> ');
  Readln(Destino);
  Assign(D,Destino);
  Rewrite(D);
  No_Regs_Restantes := Filesize(F);
(* FileSize retorna o número de registros que contém o arquivo *)
  While No_Regs_Restantes > 0 do
    Begin
      If Buf_Regs < No_Regs_Restantes
        Then Regs_para_ler := Buf_regs
        Else Regs_para_ler := No_Regs_Restantes;
      BlockRead(F,Buffer,Regs_para_ler);
      BlockWrite(D,Buffer,Regs_para_ler);
      No_Regs_restantes := No_regs_restantes-Regs_para_ler;
    End;
  Close(F);
  Close(D);
  Reset(F);
  Reset(D);
  No_Regs_Restantes := Filesize(F);
  While No_Regs_Restantes > 0 do
    Begin
      If Buf_Regs < No_Regs_Restantes
        Then Regs_para_ler := Buf_regs
        Else Regs_para_ler := No_Regs_Restantes;
      BlockRead(F,Buffer,Regs_para_ler,r);
      BlockRead(D,Buffer1,Regs_para_ler,r1);
      No_Regs_restantes := No_regs_restantes-Regs_para_ler;
      If r<>r1
        Then Begin
               Erro(1);
               Goto FIM;
```

```
            End;
        For i:=1 to 128*r do
          if buffer[i]<>buffer1[i]
            Then Begin
                   Erro(2);
                   Goto FIM;
                 End;
    End;
FIM:
End.
```

IX.2.8 - Truncate

Esta procedure trunca o arquivo a partir do registro corrente.

Sintaxe:

Truncate(Arq);

Exemplo:

```
Program Exemplo;
Uses CRT;
Var a : file of integer;
    i : integer;
Begin
   Assign(a,'Arquivo.Dta');
   Rewrite(a);
   For i:=1 to 100 do write(a,i);
   Close(a);
         (* O arquivo 'Arquivo.Dta' contem 100 números inteiros de 1 até 100 *)
   Reset(a);
   Seek(a,10);
  truncate(a); (* o arquivo foi truncado a partir do registro 10 *)
   Seek(a,0);
   while not eof(a) do   (* eof() está explicado logo abaixo *)
   Begin
     read(a,i);
     writeln(i); (* será escrito de 1 até 10 no vídeo *)
   end;
end.
```

IX.3 - Funções para operações em arquivos

IX.3.1 - Eof()

Esta função retorna um TRUE, caso tenha se alcançado um fim de arquivo, caso contrário, retorna um FALSE. Um exemplo de aplicação foi mostrado no último programa.

IX.3.2 - SeekEof()

Função semelhante ao Eof() exceto que ela pula brancos e tabulações, checando somente o marcador de fim de arquivo (CTRL-Z).

IX.3.3 - FilePos

Retorna o número do registro corrente. Lembramos novamente, que o primeiro registro recebe o número zero. Sintaxe:

```
FilePos(Arquivo);
```

IX.3.4 - FileSize

Retorna o número de registros de determinado arquivo. Retorna zero se o arquivo estiver vazio. Caso o arquivo não seja tipado, então a função FileSize considera que os registros tenham 128 bytes cada um

Sintaxe:
```
FileSize(Arquivo);
```

Esta função em conjunto com a procedure Seek, nos permite colocar o apontador de registros para o final do arquivo. Isto é muito útil quando desejamos adicionar mais registros num arquivo. Para tanto, basta declarar a seguinte instrução:

```
Seek(Arquivo,FileSize(Arquivo));
```

IX.3.5 - IORESULT

IORESULT é uma variável pré-definida no Turbo Pascal que assume determinados valores inteiros, quando algum erro de Entrada/Saída ocorre.

IORESULT pode assumir os seguintes valores:

01 Arquivo não existe
02 Arquivo não foi aberto para entrada Provavelmente, você está tentando ler de um arquivo que ainda não foi aberto.
03 Arquivo não foi aberto para saída Provavelmente, você está tentando escrever num arquivo que ainda não foi aberto.
04 Arquivo não aberto Este tipo de erro costuma acontecer quando tentamos utilizar as procedures BlockRead ou BlockWrite sem antes usarmos Reset ou Rewrite.
16 Erro no formato numérico Quando tentamos ler uma string de um arquivo texto, para uma variável numérica que não está de acordo com o formato numérico.

32 Operação não permitida para um dispositivo lógico. Por exemplo, você tenta ler de um arquivo que foi assinalado para a impressora.
33 Não permitido no modo direto.
34 Não permitido assinalação para arquivos standards.
144 Erro de comprimento de registros. Por exemplo, você tenta ler um registro que contém um número inteiro, e uma string para uma variável de estrutura diferente.
145 Seek dado para uma posição posterior ao final do arquivo.
153 Fim de arquivo foi alcançado antes de se encontrar o CTRL-Z.
240 Erro de escrita no disco.
241 Diretório cheio
242 Overflow do comprimento do arquivo.
243 Arquivo desapareceu.
Imagine que antes de se executar um close() você troque o disco.

IX.3.6 - LongFilePos

Esta função deve ser utilizada em lugar da FilePos quando o arquivo em questão tiver mais de 32K.

IX.3.7 - LongFileSize

Esta função deve ser utilizada em lugar de FileSize quando o arquivo em questão tiver mais de 32K.

IX.3.8 - LongSeek

Esta função deve ser utilizada em lugar de Seek para arquivos maiores de 32K.

IX.4 - Arquivos Textos

IX.4.1 - Definição

Os arquivos textos são utilizados para uma série de aplicações, que não são cobertas pelos arquivos, cujos conteúdos são dados binários. Exemplo: Wordstar, Word, Dbase III etc.
Para declarar um arquivo como sendo do tipo Texto, procedemos da seguinte forma:

```
Var Arquivo : Text;
```

Após esta declaração, o Turbo Pascal aloca um buffer de 128 bytes para as operações de entrada e saída com relação ao arquivo. Nós podemos modificar o tamanho desse buffer da seguinte forma:

```
Var Arquivo : Text[256];
```

Da mesma forma que nos arquivos não textos, devemos assinalar o nome lógico e em seguida, abrir o arquivo para então podermos ler ou escrever dados no arquivo.

IX.4.2 - Procedures e Functions para arquivos texto

IX.4.2.1 - Append

Esta procedure abre um arquivo do tipo texto para inclusão de dados no final do arquivo.

Sintaxe:

```
Append(Arquivo);
```

IX.4.2.2 - Readln

Lê uma linha de dados de um arquivo de texto, isto é, lê dados do arquivo até que se encontre um fim de linha.

Sintaxe:

```
Readln(Arquivo,Variável);
```

Arquivo deverá ser do tipo Text e a variável do tipo String.

IX.4.2.3 - Writeln

Escreve uma linha de dados no arquivo, isto é, escreve dados e mais o fim de linha.

Sintaxe:

```
Writeln(Arquivo,Variável);
```

Arquivo deverá ser do tipo Text e a variável do tipo String.

Exemplo:

```
Program Exemplo;
      Uses CRT;
      Var frase : String[200];
          a     : Text;
      Begin
        ClrScr;
        Assign(a,'arquivo.dta');
        ReWrite(a);
        Frase:=' ';
        While Frase<>'fim' do
        Begin
          Write('Frase --> ');
          Readln(frase);
          Writeln(a,frase);
        End;
        Close(a);
        Reset(a);
        ClrScr;
```

```
  While not eof(a) do
  Begin
    Readln(a,frase);
    Writeln(frase);
  End;
  Close(a);
End.
```

IX.4.2.4 - Eoln

Esta função retorna um TRUE se foi alcançado um fim de linha no arquivo texto. Sintaxe:

```
Eoln(arquivo);
```

Onde arquivo tem que ser do tipo texto.

X - Variáveis dinâmicas

X.1 - Comparação entre variáveis estáticas e variáveis dinâmicas

Até o presente momento, lidamos com variáveis que tiveram de ser criadas antes de se executar um programa. São variáveis que existem o tempo todo, ou seja, são variáveis estáticas. Portanto, a alocação de memória para esse tipo de variável é feita antes da execução do programa. A grande desvantagem desse tipo de variável é o fato de que uma vez criada, o espaço de memória que ela ocupa não pode mais ser alterado. As variáveis dinâmicas podem ser criadas e ou destruídas durante a execução de um programa, e esta, é a grande vantagem delas sobre as estáticas. As variáveis dinâmicas podem ser obtidas através de um tipo pré-definido em Pascal, chamado Pointer.
O Pointer ou apontador, como o próprio nome diz, aponta para um local de memória onde está armazenada uma variável.

X.2 - O tipo Pointer

O procedimento para se declarar uma variável do tipo Pointer é simples, senão vejamos:

```
Var
    p : ^Integer;
```

Após esta declaração, teríamos criado uma variável do tipo Pointer que ocupa 4 bytes (lembre-se que ela aponta um endereço, e como sabemos, no IBM/PC, um endereço é formado pelo Segment e pelo offset, cada um com 2 bytes) e que irá apontar uma variável do tipo Integer. Eu utilizei como exemplo, o tipo Integer, mas poderia ser qualquer outro tipo e até mesmo Records.
Até esse instante, não criamos a tão famosa variável dinâmica, e sim uma variável do tipo Pointer, que irá apontar o endereço de uma variável dinâmica do tipo Integer. Isto parece meio complicado a princípio, mas aos poucos, iremos entender o funcionamento desse novo tipo de variável.
E agora eu pergunto, para onde está apontando a variável recém-criada chamada p ? Simplesmente para nenhum lugar. E isto recebe o nome em Pascal de NIL. Quando escrevemos no meio de um programa a declaração abaixo:

```
p := NIL;
```

Estamos querendo dizer que a variável do tipo Pointer, chamada p, não está apontando para nenhuma variável no momento. Sempre que criamos uma variável do tipo Pointer, ela tem o valor inicial NIL.

X.3 - Criação de variáveis dinâmicas

O próximo passo, é a criação de uma variável dinâmica, para tanto, utilizamos a procedure New. Sua sintaxe é:

```
New(p);
```

Isto faz com que seja alocado um espaço de memória, suficiente para armazenar uma variável do tipo associado a p, no caso integer. Esse espaço de memória fica num local especial chamado HEAP. No caso do IBM/PC, o HEAP é toda a memória não utilizada pelo sistema.
Portanto, a declaração New(p) aloca um espaço de memória no HEAP, suficiente para armazenar uma variável do tipo Integer e retorna o endereço inicial desta região de memória para a variável p. Lembre-se que p é do tipo Pointer.

A grande questão agora é: Como acessamos essa variável dinâmica? Através da seguinte simbologia:

```
p^
```

Está na hora de um exemplo para esclarecer melhor as coisas:

```
Program Exemplo;
      Uses CRT;
      Type  Ponteiro = ^Integer;
      Var p : Ponteiro;
          i : Integer;
(* p é uma variável do tipo Pointer que aponta para variáveis dinâmicas do tipo integer *)
      Begin
        ClrScr;
        If p = NIL Then Writeln('sim');
(* como p acabou de ser criada, ela näo deve estar apontando para algum endereço, ou seja, seu
valor inicial deve ser NIL. Para descobrirmos se isso é verdadeiro, basta compará-la com NIL *)

        New(p);

(* acabamos de criar uma variável dinâmica do tipo Integer, e seu endereço foi colocado no Pointer
p *)

        p^:=100;

          (* estamos atribuindo o valor 100 à variável dinâmica recém-criada *)
```

```
  Writeln(p^);
  i:=200;
  p^:=i;
  Writeln(p^);  (* será escrito 200 *)
(* A função addr(var) retorna o endereço da variável var *)
  p:=addr(i); (* o pointer contém agora o endereço da variável i *)
  p^:=1000;   (* indiretamente estou atribuindo o valor 1000 à variável i *)
  Writeln(i); (* será escrito 1000 *)
End.
```

X.4 - Estruturas de dados com ponteiros

Suponha que você tenha que fazer um programa que terá que ler uma certa quantidade indeterminada de registros do teclado. Você não sabe se serão 10, 100 ou até 1000 registros. A princípio, você poderia super-dimensionar um array, desde que seu computador tenha memória suficiente, mas mesmo assim, corre-se o risco de, no futuro, termos que redimensionar a matriz. Para um caso como este, podemos utilizar o conceito de variáveis dinâmicas. Para tanto, devemos declarar um Pointer para uma variável, cuja estrutura seja constituída de dois campos: um contendo o valor propriamente dito que se quer armazenar e o outro apontando para a próxima variável dinâmica.

Exemplo:

```
Program Exemplo;
    Uses CRT;

{Este programa lê registros com a estrutura abaixo, até que se digite 'fim' quando é perguntado o nome da pessoa. Repare que o programa tem a capacidade de ler um número ilimitado de registros sem a preocupação de se definir um array e sua respectiva dimensão.}

     Nome  : String[30];
     Sexo  : Char;
     Idade : Integer;
     Altura: Real;
    Type
        Pessoa   = Record
                     Nome  : String[30];
                     Sexo  : Char;
                     Idade : Integer;
                     Altura: Real;
                End;
        ponteiro = ^Pessoas;
        Pessoas  = Record
                     Valor : Pessoa;
                     Prox  : Ponteiro;
                End;
    Var
        p,prim : Ponteiro;
    Procedure Linha;
```

```
Var i:integer;
Begin
  For i:=1 to 80 do write('-')
End;
Begin
  Prim:=nil;
  ClrScr;
  Repeat
    Linha;
    New(p);
    Write('Nome da pessoa -----> ');
    Readln(p^.valor.Nome);
    If (p^.valor.Nome<>'fim')
    Then Begin
           Write('Sexo ----------------> ');
           Readln(p^.valor.Sexo);
           Write('Idade ---------------> ');
           Readln(p^.valor.Idade);
           Write('Altura --------------> ');
           Readln(p^.valor.altura);
           p^.Prox:=Prim;
           Prim:=p;
         End;
  Until p^.valor.nome='fim';
  ClrScr;
  Linha;
  p:=prim;
  While p<>nil do
  Begin
    With p^.valor do
      Writeln(nome:30,sexo:5,idade:5,altura:6:2);
    p:=p^.prox;
  End;
End.
```

X.5 - Procedures para variáveis dinâmicas

X.5.1 - Dispose

Esta procedure libera o espaço ocupado pela variável em questão que deve ser do tipo Pointer. Ela não mexe com o resto do HEAP. Sintaxe:

```
Dispose(Var);
```

Podemos dizer que Dispose é contrário a New, pois esta aloca espaço no HEAP para determinado tipo de variável enquanto Dispose libera este espaço.

X.5.2 - Mark e Release

Como vimos, as variáveis dinâmicas são armazenadas num local de memória especial chamado de HEAP. Esse trecho de memória funciona como se fosse uma pilha. E para controlar o topo da pilha, o Turbo Pascal mantém um apontador. Nós podemos alterar o valor do apontador do topo do HEAP. Não podemos esquecer que alterando-o valor deste apontador, todas as variáveis dinâmicas que estiverem acima deste endereço serão perdidas. A procedure que nos permite alterar o valor deste apontador é a Release e sua sintaxe é:

```
Release(Var);
```

Onde Var deve ser uma variável do tipo Pointer e que deve conter o endereço desejado, para se atribuir ao apontador do topo do HEAP.
Já a procedure Mark nos permitem atribuir, a uma variável do tipo Pointer, o valor atual do apontador do topo do HEAP. Sintaxe:

```
Mark(Var);
```

Estas duas procedures em conjunto nos permite controlar e liberar, quando desejarmos, um trecho de memória do HEAP.

X.5.3 - GetMem e FreeMem

Com a procedure New, podemos alocar espaço necessário no HEAP somente para uma variável de determinado tipo. Com o par Mark e Release ou Dispose, podemos liberar tal espaço no HEAP. Já, as procedures GetMem e FreeMem, podemos alocar o número de bytes que desejarmos, sem estarmos presos a um determinado tipo de variável.

Sintaxes:

```
GetMem(Var,i);
```

Onde Var é do tipo Pointer e i Integer.

Após esta declaração, teríamos alocado no HEAP, um espaço de memória no HEAP no tamanho de i bytes. O endereço inicial desse trecho de memória é retornado em Var.

```
FreeMem(Var,i);
```

Esta procedure faz exatamente o contrário da GetMem, ou seja, libera i bytes no HEAP a partir do endereço armazenado em Var.

X.6 - Functions para variáveis dinâmicas

X.6.1 - MaxAvail

Retorna um número inteiro, que corresponde ao número de parágrafos (conjunto de 16 bytes) livres disponíveis no maior bloco de espaço contíguo no HEAP.

X.6.2 - MemAvail

Retorna o número de parágrafos disponíveis no HEAP.