

**ASPER – ASSOCIAÇÃO PARAIBANA DE ENSINO RENOVADO**

# **INTRODUÇÃO À PROGRAMAÇÃO**

**PROF. CÂNDIDO EGYPTO**

**JOÃO PESSOA / PB**

**MARÇO / 2004**

# SUMÁRIO

<b>1. ALGORITMOS.....</b>	<b>4</b>
1.1. DEFINIÇÃO DE ALGORITMO.....	4
1.2. POR QUE PRECISAMOS DE ALGORITMOS?.....	4
1.3. CARACTERÍSTICAS.....	4
1.4. FORMAS DE REPRESENTAÇÃO .....	5
1.5. UM AMBIENTE PARA ESCREVER ALGORITMOS.....	6
1.6. ESTRUTURAS CHAVES DA CONSTRUÇÃO DE ALGORITMOS.....	8
1.7. REFINAMENTOS SUCESSIVOS.....	9
<b>2. CONCEITOS BÁSICOS DE PROGRAMAÇÃO.....</b>	<b>11</b>
2.1. LINGUAGENS DE PROGRAMAÇÃO.....	11
2.2. TRADUTORES.....	11
<b>3. A LINGUAGEM PASCAL .....</b>	<b>12</b>
3.1. VARIÁVEIS .....	12
3.2. IDENTIFICADORES.....	12
3.3. PALAVRAS RESERVADAS .....	12
<b>4. TIPOS DE DADOS .....</b>	<b>13</b>
4.1. SIMPLES .....	13
4.2. ESTRUTURADOS.....	13
4.3. DEFINIDOS PELO USUÁRIO.....	13
<b>5. EXPRESSÕES.....</b>	<b>15</b>
5.1. OPERADORES ARITMÉTICOS.....	15
5.2. FUNÇÕES NUMÉRICAS PREDEFINIDAS.....	15
5.3. OPERADORES RELACIONAIS .....	16
5.4. OPERADORES LÓGICOS.....	16
5.5. PRIORIDADE.....	16
<b>6. FORMATO DE UM PROGRAMA PASCAL .....</b>	<b>18</b>
6.1. DECLARAÇÃO DE USO DE UNIDADES .....	18
6.2. DECLARAÇÃO DE CONSTANTES.....	18
6.3. DECLARAÇÃO DE TIPOS .....	19
6.4. DECLARAÇÃO DE VARIÁVEIS.....	19
6.5. DECLARAÇÃO DE PROCEDIMENTOS E FUNÇÕES .....	19
6.6. ÁREA DE COMANDOS.....	19
6.7. COMENTÁRIOS.....	19

<b>7. COMANDOS BÁSICOS .....</b>	<b>21</b>
7.1. ATRIBUIÇÃO.....	21
7.2. ENTRADA .....	21
7.3. SAÍDA .....	22
<b>8. ESTRUTURAS DE DECISÃO.....</b>	<b>24</b>
8.1. COMANDO IF-THEN -ELSE.....	24
8.2. COMANDO CASE-OF (DECISÃO MÚLTIPLA).....	25
<b>9. ESTRUTURAS DE REPETIÇÃO .....</b>	<b>29</b>
9.1. REPETIÇÃO COM TESTE NO INÍCIO ( WHILE-DO ).....	29
9.2. REPETIÇÃO COM TESTE NO FINAL ( REPEAT-UNTIL ) .....	31
9.3. REPETIÇÃO AUTOMÁTICA ( FOR ).....	32
<b>10. MANIPULAÇÃO DE STRINGS.....</b>	<b>36</b>
10.1. O TIPO DE DADO STRING.....	36
10.2. FUNÇÕES E PROCEDIMENTOS PREDEFINIDOS.....	36
<b>11. ARRAYS .....</b>	<b>41</b>
11.1. VETOR .....	41
11.2. MATRIZ .....	45
11.3. ARRAY MULTIDIMENSIONAL.....	48
<b>12. MODULARIZAÇÃO .....</b>	<b>51</b>
12.1. PROCEDIMENTO.....	51
12.2. FUNÇÃO.....	52
12.3. VARIÁVEIS GLOBAIS E VARIÁVEIS LOCAIS.....	53
12.4. PARÂMETROS .....	54
12.5. UTILIZANDO ARRAYS COMO PARÂMETROS .....	59
12.6. CRIAÇÃO DE UNITS.....	61
<b>13. CONTROLE DO VÍDEO E DO TECLADO.....</b>	<b>63</b>
13.1. CONTROLE DO TECLADO .....	63
13.2. CONTROLE DO VÍDEO .....	63
<b>BIBLIOGRAFIA.....</b>	<b>67</b>

# 1. ALGORITMOS

## 1.1. DEFINIÇÃO DE ALGORITMO

A palavra *algoritmo*, à primeira vista, parece-nos estranha. Embora possua designação desconhecida, fazemos uso constantemente de algoritmos em nosso cotidiano: a maneira como uma pessoa toma banho é um algoritmo. Outros algoritmos freqüentemente encontrados são:

- instruções para se utilizar um aparelho eletrodoméstico;
- uma receita para preparo de algum prato;
- guia de preenchimento para declaração do imposto de renda;
- a regra para determinação de máximos e mínimos de funções por derivadas sucessivas;
- a maneira como as contas de água, luz e telefone são calculadas mensalmente; etc.

São vários os conceitos para algoritmo. Escolhemos alguns para serem apresentados aqui:

*“Um conjunto finito de regras que provê uma seqüência de operações para resolver um tipo de problema específico”*  
[KNUTH]

*“Seqüência ordenada, e não ambígua, de passos que levam à solução de um dado problema”*  
[TREMBLAY]

*“Processo de cálculo, ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, as regras formais para a obtenção do resultado ou da solução do problema”*  
[AURÉLIO]

## 1.2. POR QUE PRECISAMOS DE ALGORITMOS?

Vejamos o que algumas pessoas importantes, para a Ciência da Computação, disseram a respeito de algoritmo:

*“A noção de algoritmo é básica para toda a programação de computadores”.*  
[KNUTH - Professor da Universidade de Stanford, autor da coleção “The art of computer programming”]

*“O conceito central da programação e da ciência da computação é o conceito de algoritmo”.*  
[WIRTH - Professor da Universidade de Zurique, autor de diversos livros na área e responsável pela criação de linguagens de programação como ALGOL, PASCAL e MODULA-2]

A importância do algoritmo está no fato de termos que especificar uma seqüência de passos lógicos para que o computador possa executar uma tarefa qualquer, pois o mesmo por si só não tem vontade própria, faz apenas o que mandamos. Com uma ferramenta algorítmica, podemos conceber uma solução para um dado problema, independentemente de uma linguagem específica e até mesmo do próprio computador.

## 1.3. CARACTERÍSTICAS

Todo algoritmo deve apresentar algumas características básicas:

- ter fim;
- não dar margem à dupla interpretação (não ambíguo);
- capacidade de receber dado(s) de entrada do mundo exterior;
- poder gerar informações de saída para o mundo externo ao do ambiente do algoritmo;
- ser efetivo (todas as etapas especificadas no algoritmo devem ser alcançáveis em um tempo finito).

## 1.4. FORMAS DE REPRESENTAÇÃO

Algoritmos podem ser representados, dentre outras maneiras, por:

### 1.4.1. DESCRIÇÃO NARRATIVA

Faz-se uso do português para descrever algoritmos.

EXEMPLO: Receita de Bolo:

- Providencie manteiga, ovos, 2 Kg de massa, etc.
- Misture os ingredientes
- Despeje a mistura na fôrma de bolo
- Leve a fôrma ao forno
- Espere 20 minutos
- Retire a fôrma do forno
- Deixe esfriar
- Prove

VANTAGENS:

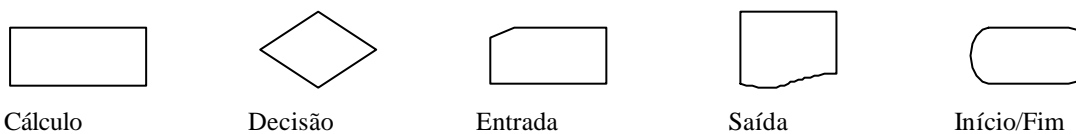
- o português é bastante conhecido por nós;

DESVANTAGENS:

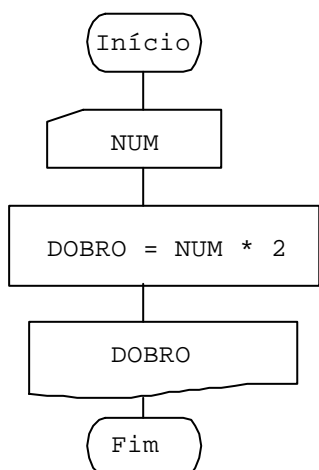
- imprecisão;
- pouca confiabilidade (a imprecisão acarreta a desconfiança);
- extensão (normalmente, escreve-se muito para dizer pouca coisa).

### 1.4.2. FLUXOGRAMA

Utilização de símbolos gráficos para representar algoritmos. No fluxograma existem símbolos padronizados para início, entrada de dados, cálculos, saída de dados, fim, etc.



#### EXEMPLO



#### EXPLICACÃO

Início do algoritmo

Entrada do número

Cálculo do dobro do número

Apresentação do resultado

Fim do algoritmo

VANTAGENS:

- Uma das ferramentas mais conhecidas;
- Figuras dizem muito mais que palavras;
- Padrão mundial

DESVANTAGENS:

- Pouca atenção aos dados, não oferecendo recursos para descrevê-los ou representá-los;
- Complica-se à medida que o algoritmo cresce.

### 1.4.3. LINGUAGEM ALGORÍTMICA

Consiste na definição de uma pseudolinguagem de programação, cujos comandos são em português, para representar algoritmos.

```
EXEMPLO:   Algoritmo CALCULA_DOBRO
            NUM, DOBRO : inteiro
            início
            Leia NUM
            DOBRO ← 2 * NUM
            Escreva DOBRO
            fim
```

VANTAGENS:

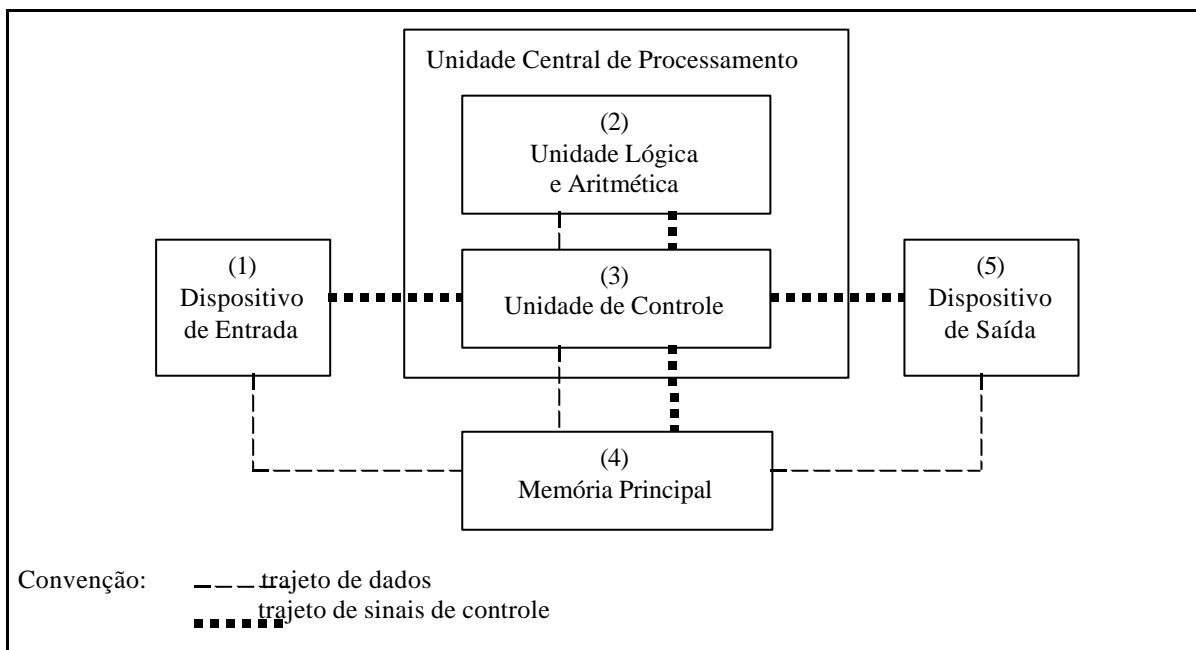
- Usa o português como base;
- Pode-se definir quais e como os dados vão estar estruturados;
- Passagem quase imediata do algoritmo para uma linguagem de programação qualquer.

DESVANTAGENS:

- Exige a definição de uma linguagem não real para trabalho;
- Não padronizado.

### 1.5. UM AMBIENTE PARA ESCREVER ALGORITMOS

Descreveremos uma máquina hipotética para a qual escreveremos nossos algoritmos. O nosso computador hipotético apresentará a seguinte organização:



Cada uma das partes constituintes da figura acima tem os seguintes significados:

- (1) Dispositivo de entrada (o teclado):  
É o meio pelo qual os dados que serão trabalhados pelo algoritmo vão ser introduzidos em nosso computador hipotético;
- (2) Unidade Lógica e Aritmética (ULA):  
Parte responsável pelas operações matemáticas e avaliações lógicas;
- (3) Unidade de Controle:  
Exerce controle sobre as demais partes do nosso computador. É uma verdadeira gerente que distribui tarefas às outras unidades;
- (4) Memória:  
Guarda o algoritmo a ser executado e os dados a serem utilizados pelo mesmo. Todo dado fornecido ao computador e o resultado de suas operações ficam guardados na memória;
- (5) Dispositivo de Saída (vídeo e impressora):  
É o meio que se dispõe para apresentação dos resultados obtidos.

### 1.5.1. FUNCIONAMENTO DO NOSSO COMPUTADOR

Todos os computadores, independentemente dos seus tamanhos, são conceitualmente semelhantes ao esquema da figura anterior (há algumas diferenças, mas não trataremos aqui dos casos especiais).

Resumidamente, podemos afirmar que existem 4 (quatro) operações básicas que qualquer computador pode executar:

- a) **operações de entrada e saída:** ler dados do teclado e escrever dados na tela são exemplos destas operações. Elas servem para introduzir dados na memória do nosso computador e exibir dados que já estejam lá armazenados;
- b) **operações aritméticas:** são utilizadas na realização de operações matemáticas (adição, subtração, multiplicação e divisão);
- c) **operações lógicas e relacionais:** têm aplicabilidade em comparações, testes de condições lógicas ( $2 > 6$  ?  $X=Y$  ?);
- d) **movimentação de dados entre os vários componentes:** as operações aritméticas são executadas na Unidade Lógica e Aritmética, necessitando da transferência dos dados para essa unidade e da volta do resultado final para ser guardado na memória.

### 1.5.2. RESOLVENDO UM PROBLEMA

Suponha que queiramos resolver o seguinte problema: a partir de dois números que serão informados, calcular a adição dos mesmos. Se você fosse encarregado de efetuar essa tarefa, seria bem provável que utilizasse os passos a seguir:

- a) saber quais são os números;
- b) calcular a soma dos números;
- c) responder à questão com o valor do resultado.

Vejamos como seria resolvido esse mesmo problema em termos das operações básicas citadas anteriormente:

- a) operação de entrada de dados dos números ;
- b1) movimento do valor dos números entre a memória e a ULA;
- b2) operação aritmética de somar os 2 números;
- b3) movimentação do resultado da ULA para guardar na memória;
- c) operação de saída do resultado, que está guardado na memória, para o dispositivo de saída desejado.

Deve-se salientar que os passos b1 e b3, normalmente, ficam embutidos na operação matemática, não sendo explicitados.

Em resumo, pode-se dizer que escrever algoritmos ou, em última análise, programar consiste em dividir qualquer problema em muitos pequenos **passos**, usando uma ou mais das quatro operações básicas citadas.

Esses passos que compõem o algoritmo são denominados de **comandos**. Os comandos de uma linguagem de programação podem estar mais próximos da máquina (linguagens de baixo nível) ou serem mais facilmente entendidos pelo homem (linguagens de alto nível). A seqüência de operações básicas, dada anteriormente, para resolver o problema de adicionar dois números, está em uma linguagem de baixo nível para o nosso computador hipotético. Em uma linguagem de alto nível teríamos um resultado assim:

```
Leia X,Y
SOMA ← X + Y
Escreva SOMA
```

## 1.6. ESTRUTURAS CHAVES DA CONSTRUÇÃO DE ALGORITMOS

Existem 3 estruturas básicas de controle nas quais se baseiam os algoritmos: sequenciação, decisão e repetição. Detalharemos cada uma delas:

### 1.6.1. SEQUENCIAÇÃO

Os comandos do algoritmo fazem parte de uma seqüência, onde é relevante a ordem na qual se encontram os mesmos, pois serão executados um de cada vez, estritamente, de acordo com essa ordem. De uma forma genérica, poderíamos expressar uma seqüência da seguinte maneira:

```
Comando-1
Comando-2
Comando-3
:
Comando-n
```

Tem-se uma sequenciação de  $n$  comandos na qual os comandos serão executados na ordem em que aparecem, isto é, o comando de ordem  $i+1$  só será executado após a execução do de ordem  $i$  (o  $3^{\text{a}}$  só será executado após o  $2^{\text{a}}$ ).

Todo algoritmo é uma seqüência. A sequenciação é aplicada quando a solução do problema pode ser decomposta em passos individuais.

### 1.6.2. DECISÃO OU SELEÇÃO

Essa estrutura também é conhecida por estrutura condicional. Há a subordinação da execução de um ou mais comandos à veracidade de uma condição. Vejamos o funcionamento:

```
Se <condição>
    então <seq. de comandos-1>
    senão <seq. de comandos-2>
```

Se a <condição> for verdadeira será executado a <seq. de comandos-1> e, em caso contrário, teremos a execução da <seq. de comandos-2>.

A decisão deve ser sempre usada quando há a necessidade de testar alguma condição e em função da mesma tomar uma atitude. Em nosso dia-a-dia, estamos sempre tomando decisões, vejamos um exemplo:

```
Se tiver dinheiro suficiente, então vou almoçar em um bom restaurante.
Caso contrário (senão), vou comer um sanduíche na lanchonete da esquina.
```

### 1.6.3. REPETIÇÃO OU ITERAÇÃO

Essa estrutura também é conhecida por “looping” ou laço. A repetição permite que tarefas individuais sejam repetidas um número determinado de vezes ou tantas vezes quantas uma condição lógica permita. Vejamos alguns exemplos:

- vou atirar pedras na vidraça até quebrá-la;
- baterei cinco pênaltis;
- enquanto tiver saúde e dinheiro, vou desfrutar a vida.

No exemplo (a), vai-se repetir a ação de atirar pedras na janela até que seja satisfeita a condição de quebrar a janela.

No exemplo (b), haverá a repetição da atitude de bater um pênalti um número determinado de vezes (cinco).

No exemplo (c), a condição que me permitirá continuar desfrutando a vida é ter dinheiro e saúde.

A utilização combinada dessas 3 estruturas descritas vai permitir expressar, usando qualquer que seja a ferramenta, a solução para uma gama muito grande de problemas. Todas as linguagens de programação oferecem representantes dessas estruturas.

## 1.7. REFINAMENTOS SUCESSIVOS

Um algoritmo é considerado completo se os seus comandos forem do entendimento do seu destinatário.

Num algoritmo, um comando que não for do entendimento do destinatário terá que ser desdobrado em novos comandos, que constituirão um **refinamento** do comando inicial, e assim sucessivamente, até que os comandos sejam entendidos pelo destinatário.

Por exemplo, o algoritmo para calcular a média aritmética de dois números pode ser escrito da seguinte forma:

```
Algoritmo CALCULA_MÉDIA
Início
  Receba os dois números
  Calcule a média dos dois números
  Exiba o resultado
Fim
```

Podemos desdobrar o comando “Calcule a média dos dois números” em:

```
Soma os dois números
Divida o resultado por 2
```

Após esse refinamento, o algoritmo pode ser considerado completo, a menos que o destinatário não saiba fazer as operações de adição e divisão, ou não seja capaz de entender diretamente algum comando.

O algoritmo estando completo, podemos reescrevê-lo, inserindo o refinamento na posição do comando que foi refinado. Assim sendo, obtém-se:

```
Algoritmo CALCULA_MÉDIA
Início
  Receba os dois números
  Soma os dois números
  Divida o resultado por 2
  Exiba o resultado
Fim
```

Reescrever um algoritmo completo, com os refinamentos sucessivos inseridos nos seus devidos lugares, permite ter uma visão global de como o algoritmo deve ser executado.

À medida que o algoritmo passa a ser maior e mais complexo, esta visão global torna-se menos clara e, neste caso, um algoritmo apresentado com os refinamentos sucessivos separados oferece uma melhor abordagem para quem precisar entendê-lo.

## EXERCÍCIOS PROPOSTOS

- P1.01. Defina, com suas palavras, o que é algoritmo.
- P1.02. Cite alguns algoritmos que podemos encontrar na vida cotidiana.
- P1.03. De acordo com seu entendimento, qual é a característica mais importante em um algoritmo? Justifique a sua resposta.
- P1.04. Um algoritmo não pode conter um comando como “Escreva todos os números inteiros positivos”. Por quê?
- P1.05. Suponha que temos um robô a nossa disposição. Esse robô chama-se MANNY e precisa ser ensinado a fazer determinadas tarefas. Para ensinar o MANNY, vamos fazer uso do português para passar-lhe as instruções necessárias à execução de cada atividade. Escreva os passos necessários para o nosso robô executar:
- a) encher uma bacia com água;
  - b) trocar uma lâmpada no teto de sua casa;
  - c) trocar o pneu de um carro;
  - d) calcular a sua idade daqui a 20 anos;
  - e) calcular a média de um aluno com 3 notas.
- P1.06. Cite as formas básicas para se representar algoritmos, definindo-as.
- P1.07. Em sua opinião, qual a melhor forma de se representar algoritmos? Justifique sua resposta.
- P1.08. Descreva, com suas próprias palavras, o funcionamento do nosso computador hipotético.
- P1.09. Especifique soluções, em termos das operações básicas do nosso computador, para os itens (d) e (e) do exercício P1.05.
- P1.10. Quais as estruturas básicas de controle dos algoritmos? Explique cada uma delas.
- P1.11. Identifique nas respostas do exercício P1.05 a utilização das estruturas básicas de controle de fluxo.
- P1.12. Escreva o algoritmo solução para o problema de multiplicar dois números (a solução deve ser expressa em alto nível).
- P1.13. Resolva o P1.09 em termos de uma linguagem de alto nível.
- P1.14. Em que consiste a técnica de "refinamentos sucessivos" ?
- P1.15. É comum ouvirmos programadores experientes afirmarem:  
*“algoritmos ... aprendi e nunca usei na prática ... não vejo necessidade...”*.  
Discuta esse tipo de afirmativa.

## 2. CONCEITOS BÁSICOS DE PROGRAMAÇÃO

Para armazenar um algoritmo na memória de um computador e para que ele possa, em seguida, comandar as operações a serem executadas, é necessário que ele seja **programado**, isto é, que seja transcrito para uma **linguagem** que o computador possa entender, direta ou indiretamente.

### 2.1. LINGUAGENS DE PROGRAMAÇÃO

**Linguagem** é uma maneira de comunicação que segue uma forma e uma estrutura com significado interpretável. Portanto, **linguagem de programação** é um conjunto finito de palavras, comandos e instruções, escritos com o objetivo de orientar a realização de uma tarefa pelo computador.

Logicamente, a linguagem que nós utilizamos em nosso cotidiano é diferente da linguagem utilizada pela máquina. A máquina trabalha somente com códigos numéricos (linguagem de máquina), baseados nos números 0 e 1 (sistema binário), que representam impulsos elétricos, ausente e presente.

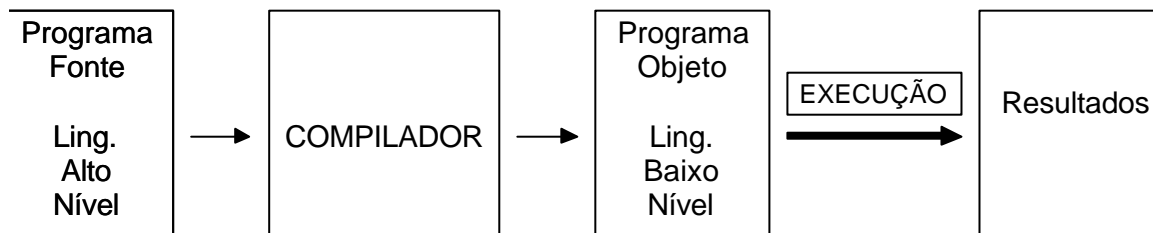
Assim, qualquer linguagem de programação deve estar situada entre dois extremos: o da linguagem natural do homem (muito clara, porém lenta) e o da linguagem de máquina (muito rápida, porém complexa).

Este é o conceito de nível de linguagem: alto nível para as mais próximas da linguagem humana; baixo nível para as mais semelhantes à linguagem de máquina.

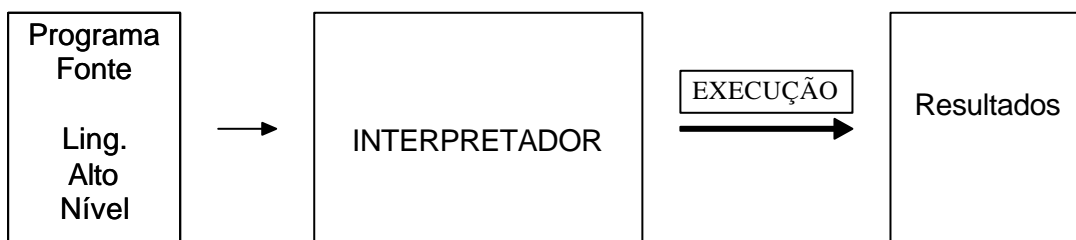
### 2.2. TRADUTORES

Para que um computador possa "entender" um programa escrito em uma linguagem de alto nível, torna-se necessário um meio de tradução entre a linguagem utilizada no programa e a linguagem de máquina. Este meio pode ser de dois tipos: *compilador* e *interpretador*.

**COMPILADOR** - traduz o programa escrito em linguagem de alto nível (programa-fonte) para um programa equivalente escrito em linguagem de máquina (programa-objeto).



**INTERPRETADOR** - traduz e envia para execução, instrução por instrução e o programa permanece na forma fonte.



Exemplos de linguagens de programação: PASCAL, C, CLIPPER, COBOL, HTML, JAVA, etc.

### EXERCÍCIOS PROPOSTOS

- P2.01. Defina, com suas palavras, os seguintes termos:
- programa
  - linguagem de programação
  - tradutor
- P2.02. Qual a diferença entre linguagem de baixo nível e linguagem de alto nível?
- P2.03. Explique a diferença entre compilador e interpretador.

## 3. A LINGUAGEM PASCAL

A linguagem de programação PASCAL foi criada para ser uma ferramenta educacional, isto no início da década de 70 pelo Prof. Niklaus Wirth da Universidade de Zurique. Foi batizada pelo seu idealizador em homenagem ao grande matemático Blaise Pascal, inventor de uma das primeiras máquinas lógicas conhecidas. Foi baseada em algumas linguagens estruturadas existentes na época, ALGOL e PLI.

Apesar de seu propósito inicial, o PASCAL começou a ser utilizado por programadores de outras linguagens, tornando-se, para surpresa do próprio Niklaus, um produto comercial. Contudo, somente ao final de 1983 foi que a empresa americana Borland International lançou o TURBO PASCAL.

O **TURBO PASCAL** é um programa que contém, além do compilador PASCAL, um ambiente completo de programação, com editor de programa, depurador de erros, sistema de ajuda, etc.

Estudaremos, a seguir, os itens fundamentais que compõe a linguagem PASCAL.

### 3.1. VARIÁVEIS

Sabe-se da Matemática que uma variável é a representação simbólica dos elementos de um certo conjunto.

Nos programas destinados a resolver um problema no computador, *a cada variável corresponde uma posição de memória, cujo conteúdo pode variar ao longo do tempo durante a execução de um programa*. Embora a variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

Toda variável é identificada por um nome ou **identificador**. Assim, por exemplo, num programa para calcular a área de um triângulo retângulo pelo teorema de Pitágoras ( $a^2 = b^2 + c^2$ ), os identificadores A, B e C podem representar as posições de memória que armazenam o valor da hipotenusa e dos catetos.

### 3.2. IDENTIFICADORES

São nomes escolhidos para representar constantes, variáveis, tipos, funções, procedimentos, unidades, programas e campos de um registro. Para definirmos um identificador na linguagem Pascal devemos observar o seguinte:

- pode ter qualquer comprimento, mas apenas os sessenta e três primeiros caracteres são significativos;
- deve ter como primeiro caracter uma letra;
- após a primeira letra só pode conter letras, dígitos ou sublinha ( \_ );
- letras maiúsculas e minúsculas são indiferentes;
- não podem haver identificadores repetidos;
- não pode ser uma palavra reservada.

### 3.3. PALAVRAS RESERVADAS

As palavras reservadas da linguagem Pascal são as seguintes:

ABSOLUTE	END	INLINE	PROCEDURE	TYPE
AND	EXTERNAL	INTERFACE	PROGRAM	UNIT
ARRAY	FILE	INTERRUPT	RECORD	UNTIL
BEGIN	FOR	LABEL	REPEAT	USES
CASE	FORWARD	MOD	SET	VAR
CONST	FUNCTION	NIL	SHL	WHILE
DIV	GOTO	NOT	SHR	WITH
DO	IF	OF	STRING	XOR
DOWNT0	IMPLEMENTATION	OR	THEN	
ELSE	IN	PACKED	TO	

### EXERCÍCIOS PROPOSTOS

P2.02. Defina variável.

P2.03. O que são identificadores?

P2.04. Quais as regras básicas para a formação de identificadores?

P2.05. Assinalar os identificadores inválidos, justificando.

a) A1BC

b) XA,1d

c) NomeDoAluno

d) 198aberto

e) TO.begin

## 4. TIPOS DE DADOS

A linguagem Pascal suporta os seguintes tipos de dados:

### 4.1. SIMPLES

**INTEGER** - Envolve os números inteiros. A partir da versão 5.0 do Turbo Pascal passaram a existir também outros tipos de números inteiros: **SHORTINT**, **BYTE**, **WORD** e **LONGINT**.

Tipo	Valor mínimo	Valor máximo	Bytes ocupados
SHORTINT	-128	127	1
BYTE	0	255	1
INTEGER	-32768	32767	2
WORD	0	65535	2
LONGINT	-2147483648	2147483647	4

Exemplos: -45, 1, 138, 0, -2

**REAL** - abrange os números reais. A partir da versão 5.0 passaram a existir também outros tipos de números reais: **SINGLE**, **DOUBLE**, **EXTENDED** e **COMP**.

Tipo	Valor mínimo	Valor máximo	Bytes ocupados	Dígitos Significativos
REAL	$2.9 \times 10^{-39}$	$1.7 \times 10^{38}$	6	11-12
SINGLE	$1.5 \times 10^{-45}$	$3.4 \times 10^{38}$	4	7-8
DOUBLE	$5.0 \times 10^{-324}$	$1.7 \times 10^{308}$	8	15-16
EXTENDED	$3.4 \times 10^{-4932}$	$1.1 \times 10^{4932}$	10	19-20
COMP	$-2^{63} + 1$	$2^{63} - 1$	8	19-20

Exemplos: 4.5, -32.0, .5, 7.8E3, 21E+3, -315E-3

**CHAR** - representa um único carácter, escrito entre apóstrofos ( ' ). A maioria dos computadores utiliza a tabela de códigos ASCII para representar todos os caracteres disponíveis. Exemplos:

'A', 'B', 'a', '1', '@', ' '

**BOOLEAN** - representa um valor lógico. Utiliza apenas duas constantes lógicas: **TRUE** (verdadeiro) e **FALSE** (falso).

### 4.2. ESTRUTURADOS

**STRING** - formado por um conjunto de elementos do tipo **CHAR**. O tamanho máximo é de 255 caracteres. Exemplos:

'CPU', 'Processamento de Dados', '123'

Discutiremos com mais detalhes o tipo string em um capítulo especialmente dedicado a este fim.

Os outros tipos de dados estruturados são: **ARRAY**, **RECORD**, **FILE**, **SET** e **TEXT**. Os dois primeiros serão apresentados no decorrer da apostila, enquanto os demais não fazem parte do escopo deste texto.

### 4.3. DEFINIDOS PELO USUÁRIO

A linguagem Pascal permite que os programadores, além de usarem os tipos predefinidos na linguagem, também possam criar novos tipos de dados. Isto se torna bastante útil quando lidamos com estruturas de dados complexas, como também ajuda a tornar o programa mais legível. Este assunto, pela sua complexidade, será abordado mais adiante.



## 5. EXPRESSÕES

A seguir, relacionamos os operadores aritméticos, relacionais e lógicos do Pascal, como também as funções predefinidas da linguagem.

### 5.1. OPERADORES ARITMÉTICOS

Operador	Operação	Operandos	Resultado
+	Adição	Integer, Real	Integer, Real
-	Subtração	Integer, Real	Integer, Real
*	Multiplicação	Integer, Real	Integer, Real
/	Divisão Real	Integer, Real	Real
DIV	Divisão Inteira	Integer	Integer
MOD	Resto da Divisão	Integer	Integer

EXEMPLOS:

Expressão	Resultado
1 + 2	3
5.0 - 1	4.0
2 * 1.5	3.0
5 / 2	2.5
5 DIV 2	2
5 MOD 2	1

### 5.2. FUNÇÕES NUMÉRICAS PREDEFINIDAS

Função	Finalidade	Tipo do argumento	Tipo do resultado
ABS(X)	Valor Absoluto	Integer, Real	o mesmo do argumento
FRAC(X)	Parte Fracionária	Real	Real
TRUNC(X)	Parte Inteira	Real	Integer
ROUND(X)	Valor Arredondado	Real	Integer
SQR(X)	Eleva ao quadrado	Integer, Real	o mesmo do argumento
SQRT(X)	Raiz quadrada	Integer, Real	Real
LN(X)	Logaritmo Natural	Real	Real
EXP(X)	Exponencial	Real	Real

Como não existe em Pascal um operador nem uma função específica para a operação de Potenciação, podemos conseguí-la utilizando as funções LN(X) e EXP(X). Para calcular o valor de  $X^N$  é suficiente usar:

`EXP ( LN ( X ) * N )`

EXEMPLOS:

Expressão	Resultado
ABS ( - 2 . 5 )	2 . 5
ABS ( 8 )	8
FRAC ( 5 . 234 )	0 . 234
TRUNC ( 2 . 78 )	2
ROUND ( 2 . 78 )	3
SQR ( 2 )	4
SQR ( 1 . 5 )	2 . 25
SQRT ( 4 )	2 . 0
SQRT ( 2 . 25 )	1 . 5
EXP ( LN ( 2 ) * 3 )	8

### 5.3. OPERADORES RELACIONAIS

Operador	Operação
=	igual
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
<>	diferente

EXEMPLOS:

Expressão	Resultado
1 = 2	FALSE
'A' = 'a'	FALSE
5 > 2	TRUE
3 <= 3	TRUE
TRUE < FALSE	FALSE
'JOAO' > 'JOSE'	FALSE
2 + 3 <> 5	FALSE
'comp' <> 'COMP'	TRUE
'11' < '4'	TRUE

### 5.4. OPERADORES LÓGICOS

Operador	Operação
not	não (negação)
and	e (conjunção)
or	ou (disjunção)

EXEMPLOS:

Expressão	Resultado
(1 > 2) and (3 > 2)	FALSE
(1 > 2) or (3 > 2)	TRUE
not (1 > 2)	TRUE

### 5.5. PRIORIDADE

Em uma expressão, a ordem de avaliação dos operadores segue a tabela abaixo:

Prioridade	Operadores
1 <sup>a</sup>	* / DIV MOD
2 <sup>a</sup>	+ -
3 <sup>a</sup>	NOT
4 <sup>a</sup>	AND
5 <sup>a</sup>	OR
6 <sup>a</sup>	= > < >= <= <>

Como a ordem de precedência dos operadores lógicos é maior que a dos operadores relacionais, devem sempre ser usados parênteses quando se escrever uma expressão lógica complexa. Por exemplo:

( A > B ) OR ( B = C )

## EXERCÍCIOS PROPOSTOS

P5.01. O que são expressões aritméticas?

P5.02. Qual o resultado das expressões aritméticas abaixo, sabendo-se que os valores de X, Y e Z são, respectivamente, 1, 2 e 5?

- a)  $Z \bmod Y \operatorname{div} Y$
- b)  $X + Y + Z / 3$
- c)  $\operatorname{FRAC}(X / Z) + \operatorname{ROUND}(Z / Y) * \operatorname{TRUNC}(Z / Y)$
- d)  $\operatorname{SQRT}(Z \operatorname{div} Y + X * Y)$
- e)  $Z - \operatorname{ABS}(X - \operatorname{SQR}(Y))$

P5.03. O que são Funções Predefinidas?

P5.04. Escreva o resultado das seguintes funções:

- a)  $\operatorname{ABS}(-4)$
- b)  $\operatorname{ABS}(5.2)$
- c)  $\operatorname{FRAC}(23.0)$
- d)  $\operatorname{FRAC}(-3.1)$
- e)  $\operatorname{TRUNC}(1.8)$
- f)  $\operatorname{TRUNC}(2.2)$
- g)  $\operatorname{ROUND}(1.8)$
- h)  $\operatorname{ROUND}(2.2)$
- i)  $\operatorname{SQR}(1.0)$
- j)  $\operatorname{SQR}(10)$
- k)  $\operatorname{SQRT}(25)$
- l)  $\operatorname{SQRT}(9.0)$

P5.05. Escreva a expressão para se calcular o valor de  $2^5$ .

P5.06. Preencha a Tabela Verdade abaixo:

A	B	A and B	A or B	not A	Not B
TRUE	TRUE				
TRUE	FALSE				
FALSE	TRUE				
FALSE	FALSE				

P5.07. Escreva o resultado das seguintes comparações:

- a)  $1 <> 1.0$
- b)  $'abc' > 'ABC'$
- c)  $' ' = ' '$
- d)  $\operatorname{FALSE} = \operatorname{FALSE}$
- e)  $\operatorname{TRUE} <> \operatorname{TRUE}$
- f)  $'JOSE' > 'JOSEFINA'$
- g)  $'50' < '100'$

P5.08. Qual o resultado das expressões lógicas abaixo, sabendo-se que os valores de A e B são, respectivamente, TRUE e FALSE?

- a)  $\operatorname{not} A \operatorname{and} B \operatorname{or} A \operatorname{and} \operatorname{not} B$
- b)  $\operatorname{not} (\operatorname{not} (A \operatorname{or} B) \operatorname{and} (A \operatorname{or} B))$
- c)  $A \operatorname{or} B \operatorname{and} \operatorname{not} A \operatorname{or} \operatorname{not} B$
- d)  $(A \operatorname{or} B) \operatorname{and} (\operatorname{not} A \operatorname{or} \operatorname{not} B)$

## 6. FORMATO DE UM PROGRAMA PASCAL

Pascal é uma linguagem altamente *estruturada* que possui uma rigidez definida, embora sua estrutura de programa seja flexível. Cada seção ou parte de um programa em Pascal deve aparecer numa seqüência apropriada e ser sistematicamente correta, senão ocorrerá um erro.

Por outro lado, no Pascal não há regras específicas para o uso de espaço, linhas quebradas, requisições e assim os comandos podem ser escritos no *formato livre* em quase todos os estilos em que o programador deseja utilizar.

Um programa escrito em Pascal tem o seguinte formato:

```
PROGRAM <identificador>;  
<bloco>.
```

O <bloco>, por sua vez, está dividido em seis áreas, onde somente a última é obrigatória e devem obedecer a seqüência abaixo. São elas:

- Área de declaração de uso de unidades
- Área de declaração de constantes
- Área de declaração de tipos
- Área de declaração de variáveis
- Área de declaração de procedimentos e funções
- Área de comandos

**Observação:** no Turbo Pascal, a cláusula PROGRAM, bem como a seqüência correta das declarações, não são obrigatórios.

### 6.1. DECLARAÇÃO DE USO DE UNIDADES

Um programa Pascal pode fazer uso de algumas unidades padrão que estão disponíveis no Sistema Turbo, tais como: CRT, DOS, PRINTER, GRAPH, etc.

A área de declaração de uso de unidades possui o seguinte formato:

```
USES <unidade> , ... , <unidade> ;
```

EXEMPLO:

```
USES CRT, PRINTER;
```

### 6.2. DECLARAÇÃO DE CONSTANTES

Serve para associarmos nomes às constantes utilizadas no programa. Possui o seguinte formato:

```
CONST <identificador>=<valor>;...;<identificador>=<valor>;
```

EXEMPLO:

```
CONST  
BRANCO = ' ' ;  
PI = 3.1416 ;  
MAX = 10 ;  
OK = TRUE ;
```

### 6.3. DECLARAÇÃO DE TIPOS

Serve para definirmos novos tipos e estruturas de dados. Não detalharemos esse tópico agora por ser assunto dos próximos capítulos.

### 6.4. DECLARAÇÃO DE VARIÁVEIS

Todas as variáveis que serão utilizadas em um programa devem ser declaradas dentro do mesmo.

A declaração de uma variável tem como finalidade:

- especificar o tipo de dado que poderá ser armazenado na variável;
- alocar um espaço na memória onde possa ser armazenado o conteúdo da variável;
- dar um nome (identificador) à variável.

Possui o seguinte formato:

```
VAR
    <lista-de-identificadores> : <tipo>;
    ...
    <lista-de-identificadores> : <tipo>;
```

Onde:

<lista-de-identificadores> é uma lista de variáveis de um mesmo tipo, separadas por vírgula.  
<tipo> é o nome de um dos tipos predefinidos ou criado pelo programador.

EXEMPLO:

```
VAR
    X, Y, Z : REAL;
    I, J : INTEGER;
    ACHOU : BOOLEAN;
    LETRA : CHAR;
    PALAVRA, FRASE : STRING;
```

### 6.5. DECLARAÇÃO DE PROCEDIMENTOS E FUNÇÕES

Nesta área são definidos os procedimentos e funções utilizados pelo programa. Também é um tópico que será detalhado mais adiante.

### 6.6. ÁREA DE COMANDOS

É nesta área onde é inserido o algoritmo do programa. Os comandos são separados entre si pelo delimitador ponto-e-vírgula. A forma geral é:

```
BEGIN
    <comando> ;
    ... ;
    <comando>
END
```

### 6.7. COMENTÁRIOS

Um comentário na linguagem Pascal é idêntico à nossa linguagem algorítmica, utilizamos chaves:

```
{ comentário }
```

**EXEMPLO:**

```
program REAJUSTE_SALARIO;
  {Finalidade: Calc. o reajuste de um salario em 20%}

uses
  Crt; {unidade que contém os comandos de tela}

const
  IND = 0.20; {indice do reajuste}

var
  SAL_ATUAL {salario atual},
  SAL_NOVO {novo salario},
  AUMENTO {valor do aumento} : Real;

begin

  clrscr; {limpa a tela}

  {leitura do salario atual}
  write('Digite o salario atual: ');
  readln(SAL_ATUAL);

  {calculo do reajuste}
  AUMENTO := SAL_ATUAL * IND;
  SAL_NOVO := SAL_ATUAL + AUMENTO;

  {exibicao do resultado}
  writeln('Novo Salario = ', SAL_NOVO:10:2);
  readkey;

end.
```

**EXERCÍCIOS PROPOSTOS**

- P6.01. Qual o formato básico de um programa Pascal?
- P6.02. Quais as áreas que podem existir em um bloco do Pascal? Qual delas é obrigatória?
- P6.03. Qual a finalidade de declararmos uma variável ?
- P6.04. Qual a finalidade de um comentário dentro de um programa? Como deve ser escrito em Pascal?
- P6.05. Dê um exemplo para cada uma das áreas abaixo:
- Declaração de Uso de Unidades
  - Declaração de Constantes
  - Declaração de Variáveis
- P6.06. Escreva os comandos necessários para declarar:
- uma variável que receba uma frase qualquer
  - três variáveis que recebam as 3 notas de um aluno
  - uma variável que receba a idade de uma pessoa

## 7. COMANDOS BÁSICOS

### 7.1. ATRIBUIÇÃO

A operação de atribuição permite que se forneça um valor a uma certa variável. Se for atribuído uma expressão à variável, será armazenado o resultado daquela expressão. Se for atribuído uma outra variável, será armazenado o conteúdo daquela variável. Para a operação de atribuição, utilizaremos a seguinte sintaxe:

O comando de atribuição tem a forma:

```
<identificador> := <expressão>
```

Exemplos:

```
A := 2           NOME := 'João'       A := B + C
B := A           SENHA := 'Y9'       NOTA := NOTA - 1
NOTA := 10       C := 1 / 3         X := 2.5
```

No comando de atribuição, a variável e a expressão devem ser do mesmo tipo, exceto nos seguintes casos:

- a) a variável sendo *real*, a expressão pode ser *integer*
- b) a variável sendo *string*, a expressão pode ser *char*

Exemplos:

```
Var
  I : Integer;
  R : Real;
  S : String;
  C : Char;
Begin
  I := 5;
  R := I;
  C := 'A';
  S := C
End.
```

### 7.2. ENTRADA

Um comando de entrada serve para que o programa solicite dados no momento em que o mesmo está sendo executado. Esses dados fornecidos serão armazenados em variáveis na memória. Em geral a unidade de entrada é o teclado, podendo também ser uma memória auxiliar como o winchester.

Considerando a unidade de entrada padrão, o teclado, o comando seria:

```
READ (<identificador-1>, ... <identificador-n>)
```

ou

```
READLN (<identificador-1>, ..., <identificador-n>)
```

Com *READ* o cursor permanece na mesma linha após a execução do comando; com o *READLN* o cursor muda para a próxima linha.

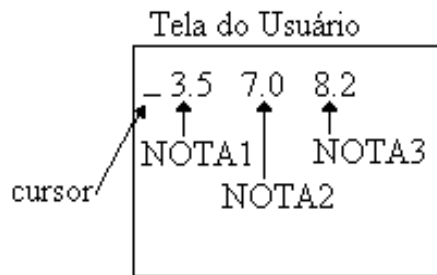
**Observação:** No Turbo Pascal, o comando *READ* só deve ser utilizado para a leitura de arquivos. Portanto, para a leitura de variáveis, devemos sempre utilizar o comando *READLN*.

EXEMPLOS:

1) Se o programa deve solicitar as três notas de um aluno, teríamos:

```
readln (NOTA1, NOTA2, NOTA3); ...
```

No momento da execução do comando acima, o programa mostra a tela do usuário e o cursor aparece esperando a digitação dos três valores que deve ser separada por, pelo menos, um espaço em branco.



3.5 será armazenado na variável `NOTA1`, o 7.0 em `NOTA2` e o 8.2 em `NOTA3`.

2) Se o programa deve solicitar o nome e o salário de um funcionário teríamos:

```

readln (NOME);
readln (SALÁRIO);
...
  
```

### 7.3. SAÍDA

Um comando de saída serve para que o programa mostre ao usuário os resultados desejados. A unidade de saída padrão é o monitor de vídeo, podendo ser também a impressora ou uma memória auxiliar como o disco.

Considerando a unidade de saída padrão, o monitor de vídeo, o comando seria:

```

WRITE (<expressão-1>, ..., <expressão-n>)
ou
WRITELN (<expressão-1>, ..., <expressão-n>)
  
```

Com `WRITE` o cursor permanece na mesma linha após a execução do comando; com `WRITELN` o cursor muda para a próxima linha.

EXEMPLO: `A:=1; B:=2;`  
`writeln ('Soma de ',A,' e ',B,' = ',A+B);`

No caso de variáveis do tipo `REAL` os valores são mostrados na notação exponencial, num campo de 16 posições, a menos que uma formatação seja especificada.

EXEMPLO: `MEDIA := (8.0 + 2.0) / 2`  
`writeln (MEDIA); {saída → 5.0000000000E+00}`  
`writeln (MEDIA:5:2); {saída → 5.00}`

Na formatação, se a variável for *real* especificamos o total de posições ocupadas e a quantidade de casas decimais



. Se *inteira*, só o total de posições.

Se desejarmos que a saída seja através da impressora e não do monitor de vídeo, devemos especificar no começo da lista da saída o parâmetro `LST` e a unidade `PRINTER` com o `USES`.

EXEMPLO: `Uses PRINTER;`  
`:::`  
`writeln (LST, 'Media = ', MEDIA:5:2);`

No instante da solicitação de dados, podemos usar junto com o `READ` ou `READLN` um comando de saída com a finalidade de emitir mensagens que orientem o usuário na digitação dos dados.

EXEMPLOS: `writeln('Digite o Nome:');`  
`readln(NOME);`

`writeln('Digite as 3 Notas:');`  
`readln(NOTA1,NOTA2,NOTA3);`

`writeln('Digite o salário do funcionário:');`  
`readln(SALARIO);`

## EXERCÍCIOS PROPOSTOS

- P7.01. Exemplifique o uso dos comandos de Atribuição, Entrada e Saída.
- P7.02. Qual a diferença entre os comandos WRITE e WRITELN ?
- P7.03. Como podemos direcionar a saída de um programa para a impressora? Dê exemplos.
- P7.04. Como podemos orientar o usuário na digitação dos dados? Exemplifique.
- P7.05. Escreva os comandos necessários para ler:  
 a) as 3 notas de um aluno  
 b) o nome, o peso e altura de uma pessoa
- P7.06. Escreva os comandos necessários para exibir:  
 a) o conteúdo da variável X  
 b) o resultado da expressão 2+3
- P7.07. Determine os valores finais de A, B e C após a execução do trecho do programa abaixo:

A := 0
B := 1
C := A + B
A := A + 1
B := A + B + C

A	B	C

- P7.08. A ordem das atribuições é importante?  $A := B$  e  $C := A$  tem o mesmo efeito de  $C := A$  e  $A := B$  ?
- P7.09. Em quais dos seguintes pares é importante a ordem dos comandos?  
 a)  $X := Y$       b)  $X := Y$       c)  $X := Z$       d)  $Z := Y$   
        $Y := X$              $Z := X$              $X := Y$              $X := Y$
- P7.10. Escreva um programa que leia um número inteiro positivo e exiba o dobro do mesmo.
- P7.11. Escreva um programa para calcular a área de um triângulo, sendo dados a sua base e a sua altura.  

$$ÁREA = \frac{BASE \cdot ALTURA}{2}$$
- P7.12. Escreva um programa para calcular e exibir o comprimento de uma circunferência, sendo dado o valor de seu raio.  

$$C = 2\pi R$$
- P7.13. Escreva um programa para ler uma temperatura dada na escala Fahrenheit e exibir o equivalente em Celsius.  

$$C = \frac{5}{9}(F - 32)$$
- P7.14. Escreva um programa para calcular e exibir a média ponderada de duas notas dadas.  
 (nota1 = peso 6 e nota2 = peso 4)
- P7.15. Escreva um programa que leia duas variáveis inteiras e troque o conteúdo entre elas.
- P7.16. Escreva um programa para calcular e exibir o valor de  $x^y$ , sendo dados a base (x) e o expoente (y).
- P7.17. Escreva um programa para ler o nome e o sobrenome de uma pessoa e escrevê-los na seguinte forma:  
 sobrenome seguido por uma vírgula e pelo nome.

Exemplo:        entrada: "Antonio","Soares"  
                   saída:     Soares, Antonio

## 8. ESTRUTURAS DE DECISÃO

As estruturas de decisão (condicionais) são utilizadas para tomar uma decisão baseada no resultado da avaliação de uma condição de controle e seleciona uma ou mais ações possíveis (comandos) para serem executados pelo computador.

No Pascal, existem três tipos de estrutura de decisão: O comando IF, que pode ser utilizado de duas formas: simples ou composto; e o comando CASE, que é utilizado para uma decisão seletiva.

### 8.1. COMANDO IF-THEN-ELSE

O comando IF é equivalente ao comando SE da linguagem algorítmica, e deve ser utilizada da seguinte forma:

```
IF <condição> THEN
  <comando1>
[ ELSE
  <comando2> ]
```

Neste caso, se a <condição> resultar no valor TRUE, será executado o <comando1>; caso contrário, será executado o <comando2>, se o mesmo existir, já que é opcional.

A <condição> deve ser uma expressão lógica. O <comando1> e <comando2> podem ser um comando simples ou um comando composto. Um comando composto é formado por dois ou mais comandos, separados por ponto-e-vírgula e delimitados por BEGIN e END.

EXEMPLO 1:

```
program EXEMPLO_DE_IF_THEN;
  {Ler um número inteiro e exibí-lo se for positivo}
var
  N : integer;

begin
  readln(N);
  if N > 0 then
    writeln(N)
end.
```

No exemplo acima, o comando WRITE só será executado se a condição  $N > 0$  for verdadeira. Caso contrário, nenhuma ação será executada.

EXEMPLO 2:

```
program EXEMPLO_DE_IF_THEN_ELSE;
  {Lê um número e determinar se é maior que zero ou não}
var
  N : integer;

begin
  readln(N);
  if N > 0 then
    writeln (N, ' é maior que zero' )
  else
    writeln (N, ' não é maior que zero')
end.
```

Neste exemplo, a mensagem que será exibida dependerá do resultado da expressão lógica  $N > 0$ . Se for verdadeira, será executado o comando WRITE que sucede a palavra THEN. Caso contrário, será executado o WRITE que sucede a palavra ELSE. Em nenhuma hipótese serão executados os dois comandos.

## EXEMPLO 3:

```

program EXEMPLO_DE_IFS_ANINHADOS;
  {Determinar se um número é maior, menor ou igual a zero}
var
  N : integer;

begin
  readln(N);
  if N > 0 then
    writeln (N,' é maior que zero' )
  else
    if N < 0 then
      writeln (N,' é menor que zero')
    else
      writeln (N,' é igual a zero')
  end.
end.

```

Pode-se observar que diversas linhas deste programa terminaram sem o ponto-e-vírgula, isto porque o ponto-e-vírgula só é utilizado para separar comandos e/ou estruturas.

Deve-se tomar cuidado quando da utilização de IF's aninhados, pois a cláusula ELSE é sempre relacionada ao último IF. Se, dentro de algum programa, precisarmos contornar este fato, podemos fazê-lo com os delimitadores BEGIN e END.

## EXEMPLO:

LINGUAGEM ALGORITMICA	LINGUAGEM PASCAL
<pre> se COND1 então   se COND2   então     comando1 senão   comando2 </pre>	<pre> if COND1 then   begin     if COND2 then       comando1     end   else     comando2 </pre>

## 8.2. COMANDO CASE-OF (DECISÃO MÚLTIPLA)

Utilizada quando se deseja executar um entre vários comandos (ou uma entre várias seqüências de comandos) dependendo do resultado de uma expressão.

A estrutura de seleção (decisão múltipla) do Pascal é o CASE, e obedece a seguinte sintaxe:

```

CASE <expressão> OF
  <lista-de-constantes-1> : <comando-1>;
  <lista-de-constantes-2> : <comando-2>;
  ...
  [ELSE <comando-n>]
END

```

A <expressão> deve resultar um tipo escalar (outros tipos que não o REAL e o STRING). A <lista-de-constantes-x> devem conter uma ou mais constantes (separadas por vírgula), e devem ser do mesmo tipo da <expressão>. O <comando-x> pode ser um comando simples ou composto.

O resultado de <expressão> é comparado com cada constante da <lista-de-constante> para verificar igualdade. Caso a igualdade seja verificada, o <comando> correspondente é executado e a estrutura finalizada. Caso nenhuma igualdade seja verificada, o <comando> correspondente ao ELSE (optativo) será executado.

## EXEMPLO:

```
program EXEMPLO_DE_DECISAO_MÚLTIPLA;  
  {Simulador de uma calculadora básica de números inteiros}  
uses  
  CRT;  
var  
  X,Y : integer;  
  OP : char;  
begin  
  clrscr;  
  write('Digite os operandos: ');  
  readln(X,Y);  
  write('Digite o operador: ');  
  readln(OP);  
  case OP of  
    '+' : writeln(X + Y);  
    '-' : writeln(X - Y);  
    '*', 'x', 'X' : writeln(X * Y);  
    '/' : writeln(X div Y);  
    else   writeln('operação inválida');  
  end {case};  
  readkey;  
end.
```

Neste exemplo, a mensagem que será exibida dependerá do conteúdo da variável OP. Se for igual a uma das constantes especificadas, será executado o comando WRITELN correspondente. Se nenhuma constante for igual ao conteúdo de OP, será executado o WRITELN do ELSE.

Podemos também escrever o mesmo programa acima sem utilizar a estrutura CASE, apenas utilizando IF's aninhados.

## EXEMPLO:

```
program EXEMPLO_DE_DECISAO_MÚLTIPLA_2;  
  {Simulador de uma calculadora básica de números inteiros}  
uses  
  CRT;  
var  
  X,Y : integer;  
  OP : char;  
begin  
  clrscr;  
  write('Digite os operandos: ');  
  readln(X,Y);  
  write('Digite o operador: ');  
  readln(OP);  
  if OP='+' then  
    writeln(X + Y)  
  else  
    if OP='-' then  
      writeln(X - Y)  
    else  
      if (OP='*') or (OP='x') or (OP='X') then  
        writeln(X * Y)  
      else  
        if OP='/' then  
          writeln(X div Y)  
        else  
          writeln('op.inválida');  
        end  
      end  
    end  
  end  
  readkey;  
end.
```

**EXERCÍCIOS PROPOSTOS**

P8.01. Qual a utilidade da estrutura de decisão?

P8.02. Qual a diferença entre a estrutura de decisão simples e a composta?

P8.03. Quais são os comandos de decisão existentes no Pascal ?

P8.04. Em que situações é mais indicado o uso da estrutura CASE-OF ? Quando não podemos utilizá-la ?

P8.05. Os comandos (i) e (ii) são equivalentes? Explique sua resposta.

```
(i)  A := B = C           (ii) if B = C then
                                A := TRUE
                                else
                                A := FALSE
```

P8.06. Observe o trecho de programa abaixo, considerando L1, L2 e L3 como variáveis booleanas.

```
...
if L1 then
  write('A')
else
  if L2 then
    if L3 then
      write('B')
    else
      begin
        write('C');
        write('D');
      end
  else
    write('E');
...

```

Agora, responda as seguintes questões:

- Se forem lidos V, V e V, o que será escrito pelo algoritmo?
- Se forem lidos F, V e V, o que será escrito pelo algoritmo?
- Se forem lidos F, V e F, o que será escrito pelo algoritmo?
- Que valores deveriam ser lidos para que fosse escrito apenas "E"?

P8.07. Escreva um programa que leia dois números e exiba o maior deles.

P8.08. Escreva um programa que leia dois números e exiba-os em ordem crescente.

P8.09. Escreva um programa que leia um número inteiro e determine se ele é par ou ímpar.

P8.10. Deseja-se calcular a conta de consumo de energia elétrica de um consumidor. Para isto, escreva um programa que leia o código do consumidor, o preço do Kw e a quantidade de Kw consumido, e exiba o código do consumidor e o total a pagar.

- total a pagar = preço x quantidade

- total a pagar mínimo = R\$ 11,20

P8.11. Escreva um programa para ler três números inteiros distintos e determinar o menor deles.

P8.12. Faça um programa que, dado as três notas de um aluno, determine e exiba a sua média final e o seu conceito, sabendo-se que:

- a média final é calculada pela média aritmética das 3 notas;

- o conceito é determinado de com base na tabela abaixo:

MÉDIA FINAL	CONCEITO
≥ 8,0	A
≥ 5,0 e < 8,0	B
< 5,0	C

- P8.13. Escreva um programa que determine o grau de obesidade de uma pessoa, sendo fornecido o peso e a altura da pessoa. O grau de obesidade é determinado pelo índice da massa corpórea ( $\text{Massa} = \text{Peso} / \text{Altura}^2$ ) através da tabela abaixo:

MASSA CORPÓREA	GRAU DE OBESIDADE
< 26	Normal
$\geq 26$ e < 30	Obeso
$\geq 30$	Obeso Mórbido

- P8.14. O Botafogo Futebol Clube deseja aumentar o salário de seus jogadores. O reajuste deve obedecer a seguinte tabela:

SALÁRIO ATUAL (R\$)	AUMENTO
0,00 a 1.000,00	20%
1.000,01 a 5.000,00	10%
acima de 5.000,00	0%

Escrever um programa que leia o nome e o salário atual de um jogador, e exiba o nome, o salário atual e o salário reajustado.

- P8.15. Faça um programa para calcular a conta final de um hóspede de um hotel, considerando que:

- serão lidos o nome do hóspede, o tipo do apartamento utilizado (A, B, C ou D), o número de diárias utilizadas pelo hóspede e o valor do consumo interno do hóspede;
- o valor da diária é determinado pela seguinte tabela:

TIPO DO APTO.	VALOR DA DIÁRIA (R\$)
A	150,00
B	100,00
C	75,00
D	50,00

- o valor total das diárias é calculado pela multiplicação do número de diárias utilizadas pelo valor da diária;
- o subtotal é calculado pela soma do valor total das diárias e o valor do consumo interno;
- o valor da taxa de serviço equivale a 10% do subtotal;
- o total geral resulta da soma do subtotal com a taxa de serviço.
- escreva a conta final contendo: o nome do hóspede, o tipo do apartamento, o número de diárias utilizadas, o valor unitário da diária, o valor total das diárias, o valor do consumo interno, o subtotal, o valor da taxa de serviço e o total geral.

- P8.16. Deseja-se calcular o imposto de renda de um contribuinte. Para isto, escreva um programa que:

- leia os seguintes dados do contribuinte: CPF, nome, rendimento anual, imposto retido na fonte, contribuição previdenciária, despesas médicas, número de dependentes;
- é deduzido o valor de R\$ 1.080,00 por cada dependente;
- cálculo do valor total das deduções: contribuição previdenciária + despesas médicas + dedução dos dependentes;
- cálculo da base de cálculo: rendimento anual – total das deduções;
- com base na tabela abaixo:

Base de Cálculo	Alíquota	Parcela a Deduzir
até 10.800,00	Isento	-
De 10.800,01 até 21.600,00	15%	1.620,00
acima de 21.600,00	25%	3.780,00

Cálculo do imposto devido:  $((\text{base de cálculo} * \text{alíquota}) - \text{parcela a deduzir})$

- haverá imposto a pagar se a diferença entre o imposto devido e o imposto retido na fonte for positiva; caso contrário, haverá imposto a restituir.
- exiba todos os dados lidos e calculados.

## 9. ESTRUTURAS DE REPETIÇÃO

A estrutura de repetição permite que uma seqüência de comandos seja executada repetidamente enquanto uma determinada condição seja satisfeita.

No Pascal, existe três tipos de estrutura de repetição: com teste no início (WHILE), com teste no final (REPEAT) e automática (FOR).

### 9.1. REPETIÇÃO COM TESTE NO INÍCIO ( WHILE-DO )

A estrutura de controle WHILE permite que um comando simples ou composto seja executado repetidamente, enquanto uma condição de controle seja VERDADEIRA. É equivalente ao comando ENQUANTO da linguagem algorítmica. A forma geral do WHILE é:

```
WHILE <condição> DO <comando>
```

A <condição> deve ser uma expressão lógica. O <comando> pode ser um comando simples ou um comando composto.

Como o teste da <condição> é realizado no início do laço, o <comando> será executado zero ou mais vezes, dependendo da avaliação da <condição>.

EXEMPLO:

```
program EXEMPLO_DE_WHILE;  
{escrever os números inteiros de 1 a 100}  
  
var  
  N : integer;  
  
begin  
  
  N := 1;  
  
  while N <= 100 do  
  begin  
    writeln(N);  
    N := N + 1  
  end  
  
end.
```

Neste exemplo, o comando escreva será executado repetidas vezes enquanto a variável N possuir um valor igual ou inferior a 100. O algoritmo terá como saída a seqüência dos números inteiros de 1 a 100.

Toda estrutura de repetição deve possuir uma condição de parada. Quando isso não acontece, ocorre o que chamamos de "**loop infinito**". Existem basicamente dois tipos de controle de estruturas de repetição: controle por contador e controle por entrada (flag).

#### 9.1.1. CONTROLE POR CONTADOR

Quando o número de repetições for previamente conhecido, devemos utilizar um contador para controlar o número de repetições do laço.

Um contador é uma variável do tipo inteiro que deve receber inicialmente (antes do laço) o valor 1 e dentro do laço (no final dele) deve ser incrementada em 1, ou seja, adicionar o valor 1 ao conteúdo da própria variável. A condição para interromper o laço será: contador  $\leq$  número de repetições.

Exemplo: este programa lê um conjunto 100 números inteiros e exibe a soma dos mesmos.

```
program LACO_CONTADO;

var
  NUM,SOMA,CONT : integer;

begin

  SOMA := 0;
  CONT := 1;

  while CONT <= 100 do
    begin
      write('Digite um número inteiro: ');
      readln(NUM);
      SOMA := SOMA + NUM;
    end

    writeln('A soma é ',SOMA);

  end.
```

### 9.1.2. CONTROLE POR ENTRADA (FLAG)

Quando não conhecemos o número de repetições e este for determinado por um valor que será lido (que chamamos de **flag**), devemos utilizar um controle de repetições por entrada, também conhecido como controle por sentinela.

Para controle por flag, a melhor estratégia é utilizar um "loop infinito" com a condição de parada sendo aplicada através do comando BREAK, dentro de um comando IF logo após a leitura da variável a ser comparada com o flag.

Obs: o comando BREAK funciona com uma "saída forçada" de qualquer comando de repetição. Porém, só é recomendado o seu uso dentro de um "loop infinito", senão pode quebrar a forma estruturada que é própria dos comandos de repetição.

Exemplo: este programa lê um conjunto de números inteiros e exiba o valor médio dos mesmos.

Obs: a condição de saída do laço será a leitura do valor 0 (flag).

```
program LACO_COM_FLAG;

var
  NUM,CONT,SOMA,MEDIA : integer;

begin

  SOMA := 0;
  CONT := 0;

  while TRUE do {loop infinito}
    begin
      write('Digite um número inteiro (0 para encerrar): ');
      readln(NUM);
      if NUM = 0 then
        break
      SOMA := SOMA + NUM;
      CONT := CONT + 1;
    end;

  MEDIA := SOMA div CONT;
  Writeln('Média = ',MEDIA);

  end.
```

## 9.2. REPETIÇÃO COM TESTE NO FINAL ( REPEAT-UNTIL )

A estrutura de controle REPEAT permite que um comando simples ou composto seja executado repetidamente até que uma condição de controle seja FALSA. A forma geral do REPEAT é:

```
REPEAT <comando> UNTIL <condição>
```

A <condição> deve ser uma expressão lógica. O <comando> pode ser um comando simples ou um comando composto. Não há a necessidade dos delimitadores BEGIN e END no comando composto em um REPEAT.

Como o teste da <condição> é realizado no final do laço, o <comando> será executado uma ou mais vezes, dependendo da avaliação da <condição>.

EXEMPLO:

```
program EXEMPLO_DE_REPEAT;  
{escrever os núm. inteiros de 1 a 100}  
  
var  
  N : Integer;  
  
begin  
  N := 1;  
  repeat  
    writeln(N);  
    N := N + 1  
  until N > 100  
end.
```

O exemplo anterior é equivalente ao exemplo do WHILE, onde o comando WRITELN será executado repetidas vezes até que a variável N possua um valor superior a 100.

A estrutura REPEAT também é bastante utilizada para repetirmos um programa diversas vezes, até que o usuário deseje sair do mesmo.

EXEMPLO:

```
program EXEMPLO_DE_REPEAT_2;  
{calcula a média de 2 números dados repetidas vezes}  
  
uses  
  CRT;  
  
var  
  N1,N2,MEDIA : real;  
  RESP : char;  
  
begin  
  clrScr;  
  repeat  
    write('Digite os dois números: ');  
    readln (N1,N2);  
    MEDIA := (N1+N2)/2;  
    writeln (MEDIA);  
    write ('Deseja repetir o programa (s/n)? ');  
    RESP := readkey;  
  until (RESP='N') or (RESP='n')  
end.
```

### 9.3. REPETIÇÃO AUTOMÁTICA ( FOR )

A estrutura de controle FOR permite que um comando simples ou composto seja repetido um número específico de vezes. A sua forma geral é:

```
FOR <var> := <vi> TO <vf> DO <comando>
```

Onde <var> é uma variável de controle, do tipo inteira, que assumirá inicialmente o valor inicial <vi> e será **incrementada** do valor 1 após cada repetição do laço. A repetição será finalizada quando o conteúdo de <var> for superior ao valor final <vf>. O <comando> também pode ser simples ou composto.

EXEMPLO:

```
program EXEMPLO_DE_FOR;
{escreve os números inteiros de 1 a 100}

var
  N : integer;

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

Observe, no exemplo acima, que não foi necessário utilizar um comando para atribuir um valor inicial a variável N, nem também um outro comando para incrementá-la com o valor 1. Isto é feito automaticamente pela estrutura FOR.

A estrutura de repetição FOR é especialmente indicada para quando o número de repetições é previamente conhecido. Caso contrário, devemos utilizar o WHILE ou o REPEAT, dependendo do caso.

Uma outra forma da estrutura FOR é a seguinte:

```
FOR <var> := <vi> DOWNTO <vf> DO <comando>
```

Neste caso, a variável de controle <var> será **decrementada** do valor 1 após cada repetição do laço e a repetição será finalizada quando o conteúdo de <var> for inferior ao valor final <vf>.

EXEMPLO:

```
program EXEMPLO_DE_FOR_COM_DOWNTO;
{escreve os números inteiros de 100 a 1}

var
  N : integer;

begin
  for N := 100 downto 1 do
    writeln(N)
  end.
```

## EXERCÍCIOS PROPOSTOS

- P9.01. Qual a utilidade da estrutura de repetição?
- P9.02. Em que consiste o controle de repetições por contador?
- P9.03. Em que consiste o controle de repetições por flag?
- P9.04. Quais são as estruturas de repetição existentes no Pascal ?
- P9.05. Qual a principal diferença entre o WHILE-DO e o REPEAT-UNTIL ?
- P9.06. Em que situações é mais indicado o uso da estrutura FOR ?
- P9.07. Em que situações não podemos utilizar a estrutura FOR ?
- P9.08. Dado o trecho de programa abaixo:

```

...
readln(N)
R := 1;
I := 2;
while I <= N-1 do
begin
    R := R * 2;
    I := I + 1;
end;
write(R);
...
    
```

Reescreva-o utilizando:

- a) o comando FOR
- b) o comando REPEAT

- P9.09. Faça o acompanhamento da execução do programa abaixo e preencha a Tabela de Variáveis:

TRECHO DE PROGRAMA	TABELA DE VARIÁVEIS		
<pre> ... N := 0; L := 1; while N &lt;&gt; 6 do begin     L := L * (-1);     N := N + 1;     if L &gt; 0 then         writeln(N);     end; ...                 </pre>	N	L	Saída

- P9.10. Faça um programa que mostre todos os números inteiros pares de 2 a 100.
- P9.11. Faça um programa para gerar e exibir os números inteiros de 20 até 10, decrescendo de 1 em 1.
- P9.12. Faça um programa que leia um número N, some todos os números inteiros de 1 a N, e mostre o resultado obtido.

$$H = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{N}$$

- P9.13. Sendo  $H = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{N}$ , faça um programa p/ calcular H. O número N é lido.
- P9.14. Faça um programa que leia N e uma lista de N números e mostre a soma de todos os números da lista.

- P9.15. Escreva um programa que leia um conjunto de 100 números inteiros positivos e determine o maior deles.
- P9.16. Escreva um programa que leia um número inteiro N e uma lista de N números inteiros positivos e determine o maior número da lista.
- P9.17. Escreva um programa que leia um conjunto de números inteiros positivos e determine o maior deles. A leitura do valor 0 (zero) indica o fim dos dados (flag).
- P9.18. Faça um programa que leia uma lista de números inteiros positivos terminada pelo número 0 (zero). Ao final, o programa deve mostrar a média aritmética de todos os números lidos (excluindo o zero).
- P9.19. Faça um programa que leia uma lista de letras terminada pela letra Z. Ao final, o programa deve mostrar a quantidade lida de cada vogal.
- P9.20. Escreva um programa que calcule o fatorial de um número inteiro lido, sabendo-se que:  
 $N! = 1 \times 2 \times 3 \times \dots \times N-1 \times N$   
 $0! = 1$
- P9.21. Faça um programa que leia 3 números inteiros (N, X, Y) e mostre todos os números múltiplos de N entre X e Y.
- P9.22. Um número é, por definição, primo se ele não tem divisores, exceto 1 e ele próprio. Escreva um programa que leia um número e determine se ele é ou não primo.
- P9.23. Faça um programa que leia dois valores inteiros (X e Y) e mostre todos os números primos entre X e Y.
- P9.24. Faça um programa que leia um número N, calcule e mostre os N primeiros termos da sequência de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, ...). O valor lido para N sempre será maior ou igual a 2.
- P9.25. Faça um programa que, para um número indeterminado de pessoas:
- leia a idade de cada pessoa, sendo que a leitura da idade 0 (zero) indica o fim dos dados (flag) e não deve ser considerada;
  - calcule e escreva o número de pessoas;
  - calcule e escreva a idade média do grupo;
  - calcule e escreva a menor e a maior idade;
- P9.26. Faça um programa que leia a altura de um grupo de 20 pessoas, calcule e exiba:
- a maior altura do grupo;
  - a altura média;
  - o número de pessoas com altura superior a 2 metros.
- P9.27. Num frigorífico existem 90 bois. Cada boi traz preso em seu pescoço um cartão contendo seu número de identificação e seu peso. Faça um programa que escreva o número e o peso do boi mais gordo e do boi mais magro (supondo que não haja empates).
- P9.28. Foi feita uma pesquisa de audiência de canal de TV em várias casas de uma certa cidade, num determinado dia. Para cada casa visitada, é fornecido o número do canal (5, 7, 10 ou 12) e o número de pessoas que o estavam assistindo naquela casa. Fazer um programa que:
- leia um número indeterminado de dados, sendo que o flag corresponde ao canal igual a 0 (zero);
  - calcule e escreva a porcentagem de audiência de cada emissora.
- P9.29. O cardápio de uma casa de lanches, especializada em sanduíches, é dado abaixo. Escreva um programa que leia o código e a quantidade de cada item comprado por um freguês, calcule e exiba o total a pagar. Obs: A leitura do código "X" indica o fim dos itens.

CÓDIGO	PRODUTO	PREÇO (R\$)
H	Hamburger	1,50
C	Cheeseburger	1,80
Q	Queijo Prato	1,00

P9.30. Escreva um programa Pascal que apresente o menu de opções abaixo:

OPÇÕES: 1 - SAUDAÇÃO 2 - BRONCA 3 - FELICITAÇÃO 0 - FIM
---

O programa deve ler a opção do usuário e exibir, para cada opção, a respectiva mensagem:

1 - Olá. Como vai ? 2 - Vamos estudar mais. 3 - Meus Parabéns ! 0 - Fim de serviço.
--

Enquanto a opção for diferente de 0 (zero) deve-se continuar apresentando as opções.

Obs: use como estrutura de repetição o REPEAT e como estrutura condicional o CASE.

P9.31. O Botafogo deseja aumentar o salário de seus 22 jogadores. O reajuste deve seguir a tabela abaixo:

SALÁRIO ATUAL (R\$)	AUMENTO
0,00 a 1.000,00	20%
1.000,01 a 5.000,00	10%
acima de 5.000,00	0%

Escrever um programa que:

- leia o nome e o salário atual de cada jogador;
- exiba o nome, o salário atual e o salário reajustado de cada jogador;
- exiba o total da folha de salários do clube, antes do reajuste.
- exiba o total da folha de salários do clube, após o reajuste.
- exiba o percentual de reajuste sobre o total da folha de salários.

P9.32. Uma certa firma fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um novo produto lançado no mercado. Para isto, forneceu o sexo do entrevistado (M-masculino ou F-feminino) e sua resposta (S-sim ou N-não). Sabendo-se que foram entrevistadas 2.000 pessoas, fazer um programa que calcule e escreva:

- número de pessoas que responderam sim (S);
- número de pessoas que responderam não (N);
- a porcentagem de pessoas do sexo feminino (F);
- a porcentagem de pessoas do sexo masculino (M);

P9.33. Escreva um programa que leia o número de andares de um prédio e, a seguir, para cada andar do prédio, leia o número de pessoas que entraram e saíram do elevador.

Considere que o elevador está vazio e está subindo, os dados se referem a apenas uma subida do elevador e que o número de pessoas dentro do elevador será sempre maior ou igual a zero.

Se o número de pessoas, após a entrada e saída, for maior que 15, deve ser mostrada a mensagem "Excesso de passageiros. Devem sair X", sendo X o número de pessoas que devem sair do elevador, de modo que seja obedecido o limite de 15 passageiros.

Após a entrada e saída no último andar, o programa deve mostrar quantas pessoas permaneceram no elevador para descer.

P9.34. Faça um programa que leia vários códigos do jogador (1 ou 2) que ganhou o ponto em uma partida de pingue-pongue, e responda quem ganha a partida.

A partida chega ao final se um dos jogadores chega a 21 pontos e a diferença de pontos entre os jogadores é maior ou igual a dois. Caso contrário, ganha aquele que, com mais de 21 pontos, consiga colocar uma vantagem de dois pontos sobre o adversário.

## 10. MANIPULAÇÃO DE STRINGS

### 10.1. O TIPO DE DADO STRING

Strings, como trechos de texto, são os tipos de dados mais familiares aos seres humanos. Podemos utilizá-lo de forma completa ou apenas um caracter por vez. Nesse caso, usamos colchetes com um valor indicando a posição do caracter que queremos acessar. Observe o exemplo abaixo:

```
S := 'COMPUTADOR';  
write(s[3]); {será exibida a letra M (3ª letra da string S)}
```

O tamanho de uma string pode variar entre 0 e 255 caracteres. Se na declaração de uma variável string não especificarmos o tamanho máximo da string, será assumido o valor 255. Por exemplo:

```
var  
  S1 : string;  
  S2 : string[10];
```

No exemplo acima, a variável S1 pode conter até 255 caracteres, enquanto a variável S2 pode conter no máximo 10 caracteres. Esse tamanho máximo nós denominamos de comprimento físico da string, que é o que determina o espaço reservado para a variável.

Se na variável S2 for armazenado um string de 4 caracteres, por exemplo, o comprimento físico continua sendo de 10 caracteres, enquanto que o espaço ocupado, no caso 4 caracteres, é o que denominamos comprimento lógico do string. O comprimento lógico de uma string pode variar conforme o valor recebido pela variável durante o programa.

Resumindo, temos então que um string pode ter o seu comprimento físico variando de 1 a 255 caracteres, e o seu comprimento lógico variando de 0 até o valor do comprimento físico.

### 10.2. FUNÇÕES E PROCEDIMENTOS PREDEFINIDOS

O Turbo Pascal dispõe de algumas funções e procedimentos que visam em essência, à otimização do trabalho do programador na parte que se refere à utilização de strings:

- LENGTH
- UPCASE
- CONCAT
- POS
- COPY
- DELETE
- INSERT
- VAL
- STR
- CHR
- ORD

**LENGTH** – Função que retorna o número de caracteres de uma string. Sua sintaxe é:

```
LENGTH (str : string) : byte;
```

Exemplo:

```
tam := length('TURBO PASCAL');  
writeln (tam); {será exibido o valor 12}
```

**UPCASE**– Função que retorna o caractere contido no parâmetro em maiúsculo. Sua sintaxe é:

```
UPCASE (ch : char) : char;
```

Exemplo:

```
letra := 'a';
maiusc := upcase (letra);
writeln (maiusc); {será exibida a letra 'A' (maiúscula) }
```

**CONCAT** – Função que retorna a união de duas ou mais strings passadas como parâmetros. Sua sintaxe é:

```
CONCAT (str1 , str2 , ... , strn : string) : string;
```

Exemplo:

```
pal1 := 'TURBO';
pal2 := 'PASCAL';
uniao := concat (pal1,' ',pal2);
writeln (uniao); {será exibido o string 'TURBO PASCAL' }
```

A função CONCAT tem efeito semelhante ao operador + (operador de concatenação).

Exemplo:

```
pal1 := 'TURBO';
pal2 := 'PASCAL';
uniao := pal1 + ' ' + pal2;
writeln (uniao); {será exibido o string 'TURBO PASCAL' }
```

**POS** – Função que retorna a posição que uma substring ocupa dentro de uma string passadas como parâmetro. Sua sintaxe é:

```
POS (substr , str : string) : byte;
```

Exemplo:

```
frase := 'VAMOS ESTUDAR MAIS';
pesq := 'ESTU';
posicao := pos (pesq,frase);
writeln (posicao); {será exibido o valor 7 }
```

**COPY**– Função que retorna uma substring de uma string passadas como parâmetro, de acordo com sua posição e quantidade de caracteres especificados. Sua sintaxe é:

```
COPY (str:string; pos:byte; quant:byte) : string;
```

Exemplo:

```
frase := 'VAMOS ESTUDAR MAIS';
pedaco := copy(frase,7,4);
writeln (pedaco); {será exibido o string 'ESTU' }
```

**DELETE** – Procedimento que exclui um pedaço de uma string passada como parâmetro, de acordo com uma posição e quantidade de caracteres especificados. Sua sintaxe é:

```
DELETE (var str:string; pos:byte; quant:byte);
```

Exemplo:

```
frase := 'TURBO PASCAL 7.0';
delete (frase,7,7);
writeln (frase); {será exibido o string 'TURBO 7.0' }
```

**INSERT** – Procedimento que permite inserir uma substring dentro de uma string, em uma posição especificada. Sua sintaxe é:

```
INSERT (substr:string; var str:string; pos:byte);
```

Exemplo:

```
frase := 'CURSO DE INFORMATICA';
insert ('MICRO',frase,10);
writeln (frase); {será exibido o string 'CURSO DE MICROINFORMATICA'}
```

**VAL** – Procedimento que converte uma string passada como parâmetro para valor numérico. Caso o conteúdo da string não seja possível de ser convertido, o fato será informado em uma variável de retorno de erro. Se o retorno de erro for diferente de 0 (zero), implica que houve um erro de conversão, e este valor de retorno é a posição onde ocorreu o primeiro erro. Sua sintaxe é:

```
VAL (str:string; var num:integer|real; var erro:integer);
```

Exemplo 1:

```
codigo := '017348';
val (codigo,numero,erro);
writeln (numero); {será exibido o valor 17348}
writeln (erro); {será exibido o valor 0}
```

Exemplo 2:

```
codigo := '12X345'
val (codigo,numero,erro);
writeln (erro) {será exibido o valor 3}
```

**STR** – Procedimento que converte uma variável numérica em um string, determinando o tamanho do string e a quantidade de casas decimais. Sua sintaxe é:

```
STR (num [:tam [:dec]]); var str:string);
```

Exemplo:

```
numero := 12.3;
str (numero:6:2,conv);
writeln (conv); {será exibido o string ' 12.30'}
```

**CHR** – Função que retorna o caracter correspondente ao valor ASCII especificado. Sua sintaxe é:

```
CHR (codigo:byte) : char;
```

Exemplo:

```
codigo := 65;
caracter := CHR(codigo);
writeln (caracter); {será exibido o caracter 'A'}
```

**ORD** – Função que retorna o valor ASCII correspondente ao caracter especificado. Sua sintaxe é:

```
ORD (caracter:char) : byte;
```

Exemplo:

```
caracter := 'A';
codigo := ORD(caracter);
writeln (codigo); {será exibido 65}
```

## EXERCÍCIOS RESOLVIDOS

R10.01. Escreva um programa que leia uma string S e a exiba em letras maiúsculas.

```
program R8_01;

uses
  CRT;

var
  S : string;
  I,TAM : byte;

begin
  clrscr;
  writeln('Digite uma frase:');
  readln(S);
  TAM := length(S);
  writeln('A frase em maiúsculas é:');
  for I := 1 to TAM do
    write(uppercase(S[I]));
  readkey;
end;
```

R10.02. Escreva um programa que leia uma palavra e uma frase S, e calcule e exiba o número de ocorrências da palavra dentro da frase.

```
program R8_02;

uses
  CRT;

var
  PAL, FRASE : string
  I,CONT,TPAL,TFRASE : byte;

begin

  clrscr;
  CONT := 0;

  writeln('Digite uma frase:');
  readln(FRASE);

  writeln('Digite uma palavra:');
  readln(PAL);

  TPAL := length(PAL)
  TFRASE := length(FRASE);

  for I := 1 to (TFRASE-TPAL+1) do
    if copy(FRASE,I,TPAL) = PAL then
      CONT := CONT + 1;

  writeln('A palavra ',PAL,' aparece ',CONT,' vezes na frase');
  readkey;

end;
```

## EXERCÍCIOS PROPOSTOS

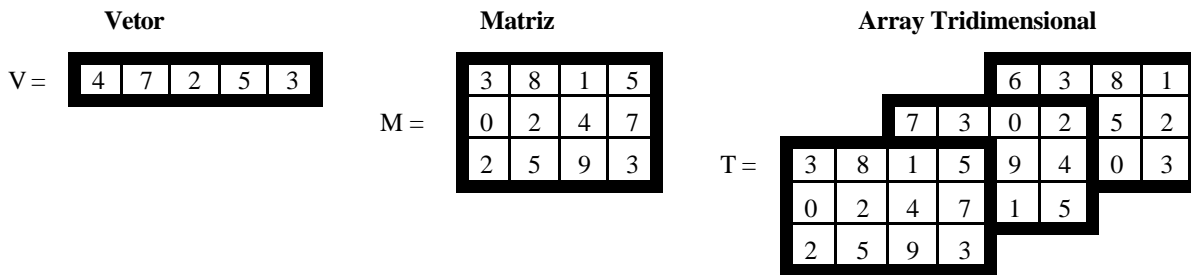
- P10.01. O que é uma string?
- P10.02. Qual o tamanho que uma string pode assumir?
- P10.03. Qual a diferença entre comprimento físico e comprimento lógico? Qual o tamanho mínimo e máximo de cada um deles?
- P10.04. Escreva a finalidade e dê um exemplo de cada um dos comandos abaixo:
- |           |           |        |
|-----------|-----------|--------|
| a) LENGTH | e) COPY   | i) STR |
| b) UPCASE | f) DELETE | j) CHR |
| c) CONCAT | g) INSERT | k) ORD |
| d) POS    | h) VAL    |        |
- P10.05. Escreva um programa que lê uma frase pelo teclado e diz quantos caracteres foram digitados na mesma.
- P10.06. Escreva um programa que receba uma frase pelo teclado e determina a quantidade de cada vogal contida na mesma.
- P10.07. Escreva um programa que receba uma string do teclado e a exiba invertida.  
Exemplo:  
Entrada → MARIA BONITA  
Saída → ATINOBAIRAM
- P10.08. Escreva um programa que leia o nome de uma pessoa e exiba-o conforme o exemplo abaixo. Obs: Suponha que o nome lido não possua nenhuma preposição, artigo, etc.  
Exemplo:  
Entrada → FLAVIO RIBEIRO COUTINHO  
Saída → COUTINHO, F. R.
- P10.09. Escreva um programa que leia três strings: FRASE, S1 e S2. Em seguida, substitua a primeira ocorrência de S1 dentro de FRASE por S2. Depois da modificação, exiba FRASE.
- P10.10. Escreva um programa que leia uma frase de no máximo 20 caracteres e o exiba de acordo com o exemplo seguinte.  
Exemplo:  
Entrada → ABCDE  
Saída → ABCDE  
BCDEA  
CDEAB  
DEABC  
EABCD
- P10.11. Escreva um programa que, a partir da digitação do infinitivo de um verbo regular, faça a conjugação do mesmo no presente do indicativo para as pessoas do singular e plural.  
Exemplo:  
Entrada → CANTAR  
Saída → Eu canto  
Tu cantas  
Ele canta  
Nós cantamos  
Vós cantais  
Eles cantam

## 11. ARRAYS

As *variáveis compostas homogêneas*, mais conhecidas como **arrays**, correspondem a conjuntos de elementos de um mesmo tipo, representados por um único nome.

Os arrays podem variar quanto a sua dimensão, isto é, a quantidade de índices necessária para a individualização de cada elemento do conjunto. O array unidimensional também é conhecido por **vetor**, enquanto o array bidimensional é denominado de **matriz**.

EXEMPLOS:



Cada elemento dos arrays podem ser referenciados através de índices. Exemplos:

V[1] = 4	M[1,1] = 3	T[1,1,1] = 3
V[2] = 7	M[2,3] = 4	T[2,3,2] = 9
V[5] = 3	M[3,1] = 2	T[1,2,3] = 3

### 11.1. VETOR

Vetores são arrays que necessitam de apenas um índice para individualizar um elemento do conjunto.

**Declaração:**

Para definirmos uma variável do tipo vetor, utilizamos a seguinte sintaxe:

`lista-de-identificadores : ARRAY[índice-inicial..índice-final] OF tipo`  
 onde:

<i>lista-de-identificadores</i>	são os nomes das variáveis que se deseja declarar;
<i>índice-inicial</i>	é o limite inferior do intervalo de variação do índice;
<i>índice-final</i>	é o limite superior do intervalo de variação do índice;
<i>tipo</i>	é o tipo dos componentes da variável

O *índice-inicial* e o *índice-final* devem ser do mesmo tipo escalar (inteiro, caracter ou booleano).

EXEMPLO:

Declarar uma variável composta de 8 elementos numéricos de nome NOTA.

```
var  NOTA : array[1..8] of real;
```

fará com que passe a existir um conjunto de 8 elementos do tipo real, individualizáveis pelos índices 1,2,3, ...,8.



Outros exemplos de declarações de vetores:

```
var   IDADE : array[1..20] of integer;
      NOME  : array[1..30] of string;
      QUANT : array['A'..'Z'] of integer;
      RECEITA, DESPESA : array[90..96] of real;
      VETOR : array[-5..5] of char;
```

Para processarmos individualmente todos os componentes de um vetor, é aconselhável o uso da estrutura FOR, para que possamos variar o índice do vetor.

## EXERCÍCIOS RESOLVIDOS

R11.01. Dado um vetor A definido com `A : array[1..100] of integer;`

a) preenchê-lo com o valor 30;

```
for I:=1 to 100 do
  A[I]:=30;
```

b) preenchê-lo com os números inteiros 1,2,3,...,100.

```
for I:=1 to 100 do
  A[I]:=I;
```

R11.02. Fazer um programa que leia um vetor A contendo 30 números inteiros, calcule e exiba:

a) o maior elemento;

b) a posição (índice) do maior elemento.

```
program R9_02;

const
  N = 30; {número de elementos do vetor}

var
  A : array[1..N] of integer;
  I, POS, MAIOR : integer;

begin
  {leitura do vetor}
  for I:=1 to N do
    readln(A[I]);
  {inicialização das variáveis}
  MAIOR := A[1];
  POS := 1;
  {comparação dos elementos com a variável MAIOR}
  for I := 2 to N do
    if A[I] > MAIOR then
      begin
        MAIOR := A[I];
        POS := I;
      end;
  {exibição dos resultados}
  writeln(MAIOR, POS);
end.
```

R11.03. Fazer um programa para ler 20 números, calcular a média dos mesmos e exibir os números que estiverem acima da média.

```
program R9_03;

const
  N = 20; {número de elementos do vetor}

var
  NUM : array[1..N] of real;
  I,SOMA : integer;
  MEDIA : real;

begin
  {inicialização da variável SOMA}
  SOMA := 0;
  {leitura e soma dos números}
  for I:=1 to N do
    begin
      readln(NUM[I]);
      SOMA := SOMA + NUM[I];
    end;
  {cálculo da média}
  MEDIA := SOMA / N;
  {exibição dos números maiores que a média}
  for I:=1 to N do
    if NUM[I] > MEDIA then
      writeln(NUM[I]);
  end.
```

R11.04. Fazer um programa que:

- a) leia um vetor VET de 100 números inteiros;
- b) leia um valor inteiro NUM;
- c) determine e exiba a posição de NUM dentro de VET. Caso NUM não seja encontrado dentro de VET, exiba o valor 0 (zero).

```
Program R9_04;

const
  N = 100; {número de elementos do vetor}

var
  VET : array[1..N] of integer;
  NUM,I,POS : integer;

begin
  {leitura dos dados}
  for I:=1 to N do
    readln(VET[I]);
  readln(NUM);
  {determinação da posição}
  POS := 0;
  for I:=1 to N do
    if VET[I] = NUM then
      POS := I;
  {exibição do resultado}
  writeln(POS);
end.
```

## EXERCÍCIOS PROPOSTOS

P11.01. Defina com suas palavras e exemplifique:

- Array
- Vetor
- Matriz
- Índice
- Elemento ou Componente

P11.02. Dar o número de elementos de cada um dos vetores dados abaixo:

- VET : array[-5..5] of integer;
- NOME: array[0..20] of string;
- CONT: array['A'..'Z'] of integer;
- NOTA: array[1..50] of real;

P11.03. Dado o seguinte vetor:

	1	2	3	4	5	6	7	8
VET =	7	4	1	5	8	2	3	6

qual será o seu conteúdo após a execução dos seguintes comandos:

```
for I:= 8 downto 5 do
begin
  AUX := VET[I];
  VET[I] := VET[8-I+1];
  VET[8-I+1] := AUX;
end;
```

P11.04. Dado dois vetores A e B contendo 20 elementos inteiros cada, gerar e exibir um vetor C do mesmo tamanho cujos elementos sejam a soma dos respectivos elementos de A e B.

Exemplo:

	1	2	3	...	19	20
A =	23	37	30	...	45	35
B =	30	32	46	...	33	42
C =	53	69	76	...	88	77

P11.05. Dado um vetor A contendo 100 elementos inteiros, gerar e exibir um vetor B cujos elementos estão na ordem inversa de A.

Exemplo:

	1	2	...	99	100
A =	23	37	...	20	26
B =	26	20	...	37	23

P11.06. Dado dois vetores A e B contendo 25 elementos inteiros cada, gerar e exibir um vetor C de 50 elementos, cujos elementos sejam a intercalação dos elementos de A e B.

Exemplo:

	1	2	3	...	24	25					
A =	23	37	30	...	38	55					
B =	30	32	46	...	43	49					
C =	23	30	37	32	30	46	...	38	43	55	49

- P11.07. Um time de basquete possui 12 jogadores. Deseja-se um programa que, dado o nome e a altura dos jogadores, determine:
- o nome e a altura do jogador mais alto;
  - a média de altura do time;
  - a quantidade de jogadores com altura superior a média, listando o nome e a altura de cada um.
- P11.08. Fazer um programa em Pascal para corrigir provas de múltipla escolha. Cada prova tem 10 questões e cada questão vale 1 ponto. O primeiro conjunto de dados a ser lido será o gabarito para a correção da prova. Os outros dados serão os números dos alunos e suas respectivas respostas, e o último número, do aluno fictício, será 0 (zero). O programa deverá calcular e imprimir:
- para cada aluno, o seu número e sua nota;
  - o percentual de aprovação, sabendo-se que a nota mínima de aprovação é 6.
  - a nota que teve maior frequência absoluta, ou seja, a nota que apareceu maior número de vezes (supondo a inexistência de empates).

A estrutura de dados para este programa de ser a seguinte:

GABARITO	NUMERO	NOTA
<input type="text"/>	<input type="text"/>	<input type="text"/>
RESPOSTAS	APROVADOS	TOTAL
<input type="text"/>	<input type="text"/>	<input type="text"/>
FREQUENCIA	MAIOR	PORCENT
<input type="text"/>	<input type="text"/>	<input type="text"/>

## 11.2. MATRIZ

Matrizes são arrays que necessitam de dois índices para individualizar um elemento do conjunto. O primeiro índice representa as linhas e o segundo as colunas.

### Declaração:

Para definirmos uma variável do tipo matriz, utilizamos a seguinte sintaxe:

```
lista-de-identificadores : ARRAY [índice1-inicial..índice1-final,
                                índice2-inicial..índice2-final] OF tipo
```

onde:

<i>lista-de-identificadores</i>	são os nomes das variáveis que se deseja declarar;
<i>índice1-inicial</i>	é o limite inferior do intervalo de variação do primeiro índice;
<i>índice1-final</i>	é o limite superior do intervalo de variação do primeiro índice;
<i>índice2-inicial</i>	é o limite inferior do intervalo de variação do segundo índice;
<i>índice2-final</i>	é o limite superior do intervalo de variação do segundo índice;
<i>tipo</i>	é o tipo dos componentes da variável

o *índice1-inicial* e o *índice1-final* devem ser do mesmo tipo escalar (inteiro, caracter ou booleano). O *índice2-inicial* também deve ser do mesmo tipo escalar do *índice2-final*.

**EXEMPLO:**

Declarar uma matriz M, de 4 linhas por 3 colunas, constituída de elementos numéricos inteiros.

```
var M : array[1..4,1..3] of integer;
```

fará com que passe a existir uma estrutura de dados agrupada denominada M, com 4x3=12 elementos inteiros, endereçáveis por um par de índices, com o primeiro indicando a linha e o outro, a coluna.

$$M =$$

$m_{11}$	$m_{12}$	$m_{13}$
$m_{21}$	$m_{22}$	$m_{23}$
$m_{31}$	$m_{32}$	$m_{33}$
$m_{41}$	$m_{42}$	$m_{43}$

Outros exemplos de declarações de matrizes:

```
var
  M1 : array[1..4,80..90] of real;
  M2 : array['A'..'E',0..10] of string;
  M3 : array[-3..3,1..3] of char;
```

**EXERCÍCIOS RESOLVIDOS**

R11.05. Fazer um programa para ler uma matriz 3 x 5 de números inteiros e escrevê-la após ter multiplicado cada elemento por 2.

```
Program R9_05;

const
  NL = 3; {número de linhas}
  NC = 5; {número de colunas}
  K = 2; {fator para multiplicação}

var
  M : array[1..NL,1..NC] of integer;
  I,J : integer;

begin
  {leitura da matriz}
  for I:=1 to NL do
    for J:=1 to NC do
      begin
        write('Elemento da linha ',I,' coluna ',J,' : ');
        readln(M[I,J]);
      end;
    end;

  {cálculo da multiplicação}
  for I:=1 to NL do
    for J:=1 to NC do
      M[I,J] := M[I,J] * K;
    end;
  end;

  {exibição da matriz resultante}
  writeln('Resultado:');
  for I:=1 to NL do
    begin
      for J:=1 to NC do
        write(M[I,J], ' ');
      end;
      writeln;
    end;
  end.
end.
```

R11.06. Dada uma matriz de 4 x 5 elementos inteiros, calcular a soma de cada linha, de cada coluna e de todos os seus elementos.

Obs: utilize um vetor para armazenar o resultado da soma de cada linha e outro para a soma de cada coluna.

```
program R9_06;

const
  NL = 4; {número de linhas}
  NC = 5; {número de colunas}

var
  M : array[1..NL,1..NC] of integer;
  L : array[1..NL] of integer;
  C : array[1..NC] of integer;
  I,J,SOMA : integer;

begin

  {leitura da matriz}
  for I:=1 to NL do
    for J:=1 to NC do
      begin
        write('Elemento da linha ',I,' coluna ',J,' : ');
        readln(M[I,J]);
      end;

  {cálculo da soma dos elementos de cada linha}
  for I:=1 to NL do
    begin
      L[I] := 0;
      for J:=1 to NC do
        L[I] := L[I] + M[I,J];
      end;

  {cálculo da soma dos elementos de cada coluna}
  for J:=1 to NC do
    begin
      C[J] := 0;
      for I:=1 to NL do
        C[J] := C[J] + M[I,J];
      end;

  {cálculo da soma de todos os elementos da matriz}
  SOMA := 0;
  for I:=1 to NL do
    for J:=1 to NC do
      SOMA := SOMA + M[I,J];

  {exibição dos resultados}
  for I:=1 to NL do
    writeln('Soma da Linha ',I,' : ',L[I]);
  for J:=1 to NC do
    writeln('Soma da Coluna ',J,' : ',C[J]);
  writeln('Soma da Matriz: ',SOMA);

end.
```

### 11.3. ARRAY MULTIDIMENSIONAL

O Pascal permite a criação de arrays multidimensionais, que necessitam de vários índices para serem manipulados. A maneira de utilizarmos este tipo de array segue a mesma lógica das matrizes, diferenciando apenas no número de índices.

#### EXERCÍCIO RESOLVIDO

R11.07. Faça um programa que monte e exiba um array tridimensional 5 x 7 x 3, onde o conteúdo de cada elemento é igual a soma de seus índices.

```
Program R9_07;

const
  NL = 5; {número de linhas}
  NC = 7; {número de colunas}
  NP = 3; {número de páginas}

var
  A : array[1..NL,1..NC,1..NP] of integer;

begin
  {geração do array}
  for I:=1 to NL do
    for J:=1 to NC do
      for K:=1 to NP do
        A[I,J,K] := I + J + K;

  {exibição do array}
  for K:=1 to NP do
  begin
    for I:=1 to NL do
    begin
      for J:=1 to NC do
        write(M[I,J], ' ');
      writeln;
    end;
    writeln;
  end;

end.
```

#### EXERCÍCIOS PROPOSTOS

P11.09. Dar o número de elementos de cada uma das matrizes abaixo dados abaixo:

- MAT : array[1..3,1..4] of integer;
- CONJ: array[0..2,1..3] of string;
- TAB : array['A'..'E',-1..1] of integer;
- NOTA: array[90..98,0..1] of real;
- A : array[1..2,1..3,1..4] of char;

P11.10. Dado as matrizes M e R abaixo:

M =

O	Q	*	I
E	*	E	S
R	E	U	T
A	*	*	S

R =


qual será o conteúdo de R depois de executado os comandos:

```
for I:= 1 to 4 do
  for J:=1 to 4 do
    R[J,I] := M[I,J];
  AUX := R[1,1];
  R[1,1] := R[4,4];
  R[4,4] := AUX;
  AUX := R[2,2];
  R[2,2] := R[3,3];
  R[3,3] := AUX;
```

P11.11. Dado duas matrizes A e B, com 2 x 3 elementos inteiros cada, gerar e exibir uma matriz C do mesmo tamanho que resultará da soma da matriz A com a matriz B.

P11.12. Faça um programa que leia uma matriz de ordem 3 x 5 de elementos inteiros, calcular e exibir:

- o maior elemento da matriz;
- a soma dos elementos da matriz;
- a média dos elementos da matriz;

P11.13. Dado uma matriz quadrada de ordem N, de elementos inteiros, exibir os elementos da diagonal principal, isto é, os elementos onde  $i = j$ . Obs: N será lido ( $N \leq 10$ ).

P11.14. Dado uma matriz A com 3 x 4 elementos inteiros, gerar e exibir uma matriz B que será a matriz transposta de A.

P11.15. Dado uma matriz A com 2 x 3 elementos inteiros e uma matriz B com 3 x 2 elementos inteiros, gerar e exibir uma matriz C que resultará da multiplicação da matriz A com a matriz B.

P11.16. Faça um programa que leia o nome e as 3 notas dos 50 alunos de uma turma e:

- calcule:
  - a média aritmética de cada aluno;
  - a situação de cada aluno; (aprovado se média superior ou igual a 7.0)
  - o número de alunos aprovados;
  - a média geral da turma;
- exiba:
  - o nome e a situação de cada aluno;
  - o número de alunos aprovados;
  - a média geral da turma;
  - o nome e a média dos alunos com média superior ou igual à média geral da turma.

⇒ Use vetores para armazenar nome, média e situação, e uma matriz para armazenar as notas.

P11.17. A tabela abaixo demonstra a quantidade de vendas dos fabricantes de veículos durante o período de 1993 a 1998, em mil unidades.

Fabricante / Ano	1993	1994	1995	1996	1997	1998
Fiat	204	223	230	257	290	322
Ford	195	192	198	203	208	228
GM	220	222	217	231	245	280
Wolkswagen	254	262	270	284	296	330

Faça um programa que:

- leia os dados da tabela;
- determine e exiba o fabricante que mais vendeu em 1996;
- determine e exiba o ano de maior volume geral de vendas.
- determine e exiba a média anual de vendas de cada fabricante durante o período.

P11.18. Faça um programa que leia e armazene em um array tridimensional contendo os valores do faturamento anual de uma empresa, especificados mês a mês e também por filial. Veja a estrutura do array abaixo:

					ANO DE 1997		
					ANO DE 1996		TOTAL
					ANO DE 1995		TOTAL
ANO DE 1994					TOTAL		
MESES	FILIAL 1	FILIAL 2	FILIAL 3	TOTAL			
Janeiro							
Fevereiro							
Março							
Abril							
Mai							
Junho							
Julho							
Agosto							
Setembro							
Outubro							
Novembro							
Dezembro							
<b>TOTAL</b>							

Após a leitura dos dados faça o seguinte:

- calcule os totais das linhas e das colunas em cada ano;
- crie uma nova página contendo a consolidação dos dados, isto é, a soma de todos os anos (mês a mês e por filial);
- exiba todos os dados lidos e calculados.

## 12. MODULARIZAÇÃO

A modularização consiste num método utilizado para facilitar a construção de grandes programas, através de sua divisão em pequenas etapas, que são os *módulos* ou *subprogramas*. A primeira delas, por onde começa a execução do trabalho, recebe o nome de *programa principal*, e as outras são os *subprogramas* propriamente ditos, que são executados sempre que ocorre uma chamada dos mesmos, o que é feito através da especificação de seus nomes.

### Vantagens da utilização de subprogramas:

- Economia de código: escreve-se menos;
- Desenvolvimento modularizado: pensa-se no algoritmo por partes;
- Facilidade de depuração (correção/acompanhamento): é mais fácil corrigir/detectar um erro apenas uma vez do que dez vezes;
- Facilidade de alteração do código: se é preciso alterar, altera-se apenas uma vez;
- Generalidade de código com o uso de parâmetros: escreve-se algoritmos para situações genéricas.

Há duas espécies de subprogramas: **PROCEDIMENTO** e **FUNÇÃO**.

### 12.1. PROCEDIMENTO

Um subprograma do tipo PROCEDIMENTO é, na realidade, um programa com vida própria, mas que, para ser processado, tem que ser solicitado pelo programa principal que o contém, ou por outro subprograma, ou por ele mesmo.

#### Declaração:

```
PROCEDURE nome;  
  declaração dos objetos locais ao Procedimento  
BEGIN  
  comandos do Procedimento  
END;
```

onde: *nome* é o identificador associado ao procedimento.

#### EXEMPLO:

O programa abaixo calcula a média aritmética entre duas notas, sem o uso de procedimentos.

```
Program CALCULA_MÉDIA; {sem o uso de procedimentos}  
  
var  
  NOTA1,NOTA2,MEDIA : real;  
  
begin  
  {lê as notas}  
  write('Digite a primeira nota: ');  
  readln(NOTA1);  
  write('Digite a segunda nota: ');  
  readln(NOTA2);  
  {calcula a media}  
  MEDIA := (NOTA1 + NOTA2) / 2;  
  {escreve o resultado}  
  writeln('Media = ',MEDIA,4:1)  
end.
```

Mostraremos agora o mesmo programa, utilizando um procedimento.

```

program CALCULA_MÉDIA; {usando procedimento}
var
    NOTA1,NOTA2,MEDIA : real;

{declaração do procedimento}
procedure LER_NOTAS;
begin
    write('Digite a primeira nota: ');
    readln(NOTA1);
    write('Digite a segunda nota: ');
    readln(NOTA2);
end;

{Programa Principal}
begin
    LER_NOTAS; {ativação do procedimento LER_NOTAS}
    MEDIA := (NOTA1 + NOTA2) / 2; {calcula a media}
    Writeln('Media = ',MEDIA,4:1) {escreve o resultado}
end.

```

## 12.2. FUNÇÃO

As funções, embora bastante semelhantes aos procedimentos, têm a característica especial de retornar ao programa que as chamou um valor associado ao nome da função. Esta característica permite uma analogia com o conceito de função da Matemática.

### Declaração:

```

FUNCTION nome : tipo;
    declaração dos objetos locais à Função
BEGIN
    Comandos da Função
END;

```

onde: *nome* é o identificador associado à função.  
*tipo* é o tipo da função, ou seja, o tipo do valor de retorno.

### EXEMPLO:

O programa abaixo calcula a média dos elementos de um vetor, sem uso de Procedimentos ou Funções.

```

program SOMA_VETOR; {sem o uso de procedimentos ou funções}
const N = 30;
var VETOR : array[1..N] of integer;
    I,SOMA,MEDIA : integer;
begin
    {lê os valores do vetor}
    for I:=1 to N do
        readln(VETOR[I]);
    {calcula a media}
    SOMA := 0;
    for I:=1 to N do
        SOMA := SOMA + VETOR[I];
    MEDIA := SOMA div N;
    {escreve o resultado}
    writeln(MEDIA)
end.

```

Mostraremos agora o mesmo programa, utilizando um procedimento para ler os valores do vetor e uma função para efetuar o cálculo da média.

```

program SOMA_VETOR; {usando uma função e um procedimento}

const
  N = 30;

var
  VETOR : array[1..N] of integer;

{declaração do procedimento}
procedure LER_DADOS;
  var I : integer;
  begin
    for I:=1 to N do
      readln(VETOR[I]);
    end;

{declaração da função}
function MEDIA : integer;
  var I,SOMA : integer;
  begin
    SOMA := 0;
    for I:=1 to N do
      SOMA := SOMA + VETOR[I];
    MEDIA := SOMA div N;
  end;

{Programa Principal}
begin
  {ativa o procedimento LER_DADOS}
  LER_DADOS;
  {escreve o resultado, chamando a função MEDIA}
  writeln(MEDIA)
end.

```

### 12.3. VARIÁVEIS GLOBAIS E VARIÁVEIS LOCAIS

Observe que, no exemplo anterior, declaramos uma variável no programa principal e outras nos subprogramas. Podemos dizer que a variável VETOR, que foi declarada no programa principal é uma *variável global* aos subprogramas, enquanto que a variável I é dita *variável local* ao procedimento LER\_DADOS e as variáveis I e SOMA são locais à função MEDIA. É importante ressaltar que a variável I do procedimento LER\_DADOS é diferente da variável I da função MEDIA, embora possuam o mesmo identificador.

O uso de *variáveis globais* dentro de procedimentos e funções serve para implementar um mecanismo de transmissão de informações de um nível mais externo para um mais interno.

As *variáveis locais* dos procedimentos e funções são criadas e alocadas quando da sua ativação e automaticamente liberadas quando do seu término.

A utilização de *variáveis globais* não constitui, no entanto, uma boa prática de programação. Assim, todos subprogramas devem apenas utilizar as *variáveis locais*, conhecidas dentro dos mesmos, e a transmissão de informações para dentro e fora dos subprogramas deve ser feita através dos *parâmetros* de transmissão, que serão apresentados a seguir.

## 12.4. PARÂMETROS

Quando se deseja escrever um subprograma que seja o mais genérico possível, deve-se usar a passagem de parâmetros.

A passagem de parâmetros formaliza a comunicação entre os módulos. Além disto, permite que um módulo seja utilizado com operandos diferentes, dependendo do que se deseja do mesmo.

Dá-se a designação de *parâmetro real* ou *de chamada* ao objeto utilizado na unidade chamadora/ativadora e de *parâmetro formal* ou *de definição* ao recebido como parâmetro no subprograma.

Dentre os modos de passagem de parâmetros, podemos destacar a *passagem por valor* e a *passagem por referência*.

Na passagem de parâmetros *por valor*, as alterações feitas nos parâmetros formais, dentro do subprograma, não se refletem nos parâmetros reais. O valor do parâmetro real é copiado no parâmetro formal, na chamada do subprograma. Assim, quando a passagem é por valor, isto significa que o parâmetro é de *entrada*.

Na passagem de parâmetros *por referência*, a toda alteração feita num parâmetro formal corresponde a mesma alteração feita no seu parâmetro real associado. Assim, quando a passagem é por referência, isto significa que o parâmetro é de *entrada-saída*.

Na linguagem Pascal, a declaração dos procedimentos e funções com parâmetros se diferencia da forma já apresentada apenas pela inclusão da lista de parâmetros formais no cabeçalho. Esta deve vir entre parênteses e cada parâmetro deve ter o seu tipo especificado. A forma geral é:

```
PROCEDURE nome (lista de parâmetros formais)
FUNCTION nome (lista de parâmetros formais) : tipo
```

A *lista de parâmetros formais* tem a seguinte forma:

```
parâmetro1 : tipo; parâmetro2 : tipo; ...; parâmetro n : tipo
```

Exemplos da lista de parâmetros:

```
procedure PROC (X,Y,Z:integer; K:real)
function FUNC (A,B:real; C:string) : integer
```

Na forma apresentada, a passagem dos parâmetros será por valor. Para se utilizar a passagem por referência, deve-se acrescentar a palavra **VAR** antes do nome do parâmetro.

EXEMPLO:

```
procedure PROC(A:integer; var B,C:integer)
```

Na chamada de procedimentos ou funções utilizando parâmetros, devemos acrescentar após o nome do procedimento ou função uma lista de parâmetros reais (de chamada), os quais devem ser do mesmo tipo e quantidade dos parâmetros formais declarados.

O exemplo a seguir demonstra a diferença entre a passagem de parâmetros por referência e a passagem de parâmetros por valor:

```
program EXEMPLO_PASSAGEM_PARÂMETROS;

var N1,N2 : integer;

Procedure PROC(X:integer; var Y:integer);
begin
    X:=1;
    Y:=1;
end;

begin
    N1:=0; N2:=0;
    PROC(N1,N2);
    writeln(N1); {será exibido o valor 0}
    writeln(N2); {será exibido o valor 1}
end.
```

## EXERCÍCIOS RESOLVIDOS

- R12.01. Escrever uma função chamada MAIOR que receba dois números inteiros e retorne o maior deles. Escrever um programa para ler dois números inteiros e, utilizando a função MAIOR, calcular e exibir o maior valor entre os números lidos.

```
program CALCULA_MAIOR;

var X,Y,M : integer;

function MAIOR (NUM1,NUM2:integer) : integer;
begin
    If NUM1 > NUM2 then
        MAIOR := NUM1
    else
        MAIOR := NUM2;
    end;

begin
    readln(X,Y);
    M := MAIOR(X,Y);
    writeln(M);
end.
```

- R12.02. Escrever um procedimento chamado DOBRA que multiplique um número inteiro (recebido como parâmetro) por 2. Escrever um programa para ler um valor inteiro e, utilizando o procedimento DOBRA, calcular e exibir o dobro do mesmo.

```
program CALCULA_DOBRO;

var X : integer;

procedure DOBRA (var NUM:integer);
begin
    NUM := NUM * 2
end;

begin
    readln(X);
    DOBRA(X);
    writeln(X);
end.
```

R12.03. Escreva um procedimento que receba uma string S e converta o mesmo para letras maiúsculas.

```
procedure MAIUSC (var S:string);
  var
    I,TAM : byte;
  begin
    TAM := length(S);
    for I:= 1 to TAM do
      S[I] := upcase(S[I]);
    end;
```

R12.04. Escreva uma função que retorne o número de ocorrências de um substring SUB dentro de uma string S, passados como parâmetros.

```
function OCORRENCIAS (SUB,S:string) : byte;
  var
    I,CONT,TSUB,TS : byte;
  begin
    CONT := 0;
    TSUB := length(SUB)
    TS := length(S);
    for I:= 1 to (TS-TSUB+1) do
      if copy(S,I,TSUB) = SUB then
        CONT := CONT+1;
    OCORRENCIAS := CONT;
  end;
```

R12.05. Escreva um procedimento que receba uma string S como parâmetro e retire todos os brancos contidos no mesmo.

```
procedure TIRABRANCOS (var S:string);
  var
    I,TAM : byte;
  begin
    TAM := length(S);
    I := 1;
    while I<=TAM do
      if S[I]=' ' then
        begin
          delete(S,I,1);
          TAM := TAM-1;
        end
      else
        I := I+1;
    end;
```

R12.06. Escreva uma função que receba um número real e retorne uma string correspondente ao número recebido, com o mesmo convertido para string com tamanho mínimo e 2 casas decimais, e com uma vírgula no lugar do ponto decimal.

```
function CONVERSAO (X:real) : string;
  var
    P : byte;
    S : string;
  begin
    str(X:0:2,S);
    P := pos('.',S);
    S[P] := ',';
    CONVERSAO := S;
  end;
```

## EXERCÍCIOS PROPOSTOS

- P12.01. Defina modularização.
- P12.02. Cite as principais vantagens da utilização de subprogramas.
- P12.03. Conceitue procedimento e função. Em que eles são semelhantes e em que eles são diferentes?
- P12.04. Que tipo de informação deve ser incluído na declaração de um procedimento? E na declaração de uma função? Se houver diferenças, explique o motivo.
- P12.05. Qual a diferença entre variável global e variável local?
- P12.06. Como deve ser feita a transmissão de informações entre um subprograma e o programa principal?
- P12.07. Qual a diferença entre parâmetro real e parâmetro formal?
- P12.08. Cite os modos de passagem de parâmetros, explicando como funciona cada um deles.
- P12.09. Escreva um procedimento que limpe a tela do micro e exiba o seu nome.
- P12.10. Escreva um procedimento que receba uma string S e um inteiro positivo N e exiba a string S por N vezes seguidas na tela.
- P12.11. Escreva uma função chamada CUBO que receba um valor do tipo real e retorne a potência elevado a 3 do mesmo.
- P12.12. Escreva um procedimento chamado TROCA que receba duas variáveis inteiras (X e Y) e troque o conteúdo entre elas;
- P12.13. Escreva uma função que receba uma string S e retorne o número de brancos existentes na mesma.
- P12.14. Escreva uma função que receba uma string S e um valor inteiro N e retorne os N primeiros caracteres da string S.
- P12.15. Supondo que no Turbo Pascal não existisse a função UPCASE, escreva uma função que simule a mesma.
- P12.16. Idem para a função POS.
- P12.17. Idem para o procedimento DELETE.
- P12.18. Escreva um procedimento chamado SINAL que receba como parâmetro um valor N inteiro e escreva a palavra POSITIVO se N for um número maior que zero, NEGATIVO se N for menor que zero, ou ZERO se N for igual a zero.
- Escreva um programa que leia um número inteiro e, usando o procedimento SINAL, mostre se ele é maior, menor ou igual a zero.
- P12.19. Escreva um procedimento chamado METADE que divida um valor do tipo real (passado como parâmetro) pela metade.
- Escreva um programa que leia um vetor A de 30 elementos reais e, usando o procedimento METADE, divida todos seus elementos pela metade.
- P12.20. Escreva uma função chamada MEDIA que retorne a média de três valores reais (X, Y e Z) passados como parâmetros.
- Escreva um programa que, para um número indeterminado de alunos, faça para cada uma delas:
- ⇒ ler o nome e as três notas do aluno (a leitura do nome FIM indica o fim dos dados - flag);
  - ⇒ calcule a média do aluno (usando a função MEDIA);
  - ⇒ exiba o nome e a média do aluno.

P12.21. Escreva um procedimento chamado AUMENTO que receba dois valores reais X e Y como parâmetros e aumente o valor de X em Y%.

Escreva um programa que leia uma variável K do tipo real e, para um número indeterminado de funcionários de uma empresa, faça para cada uma delas:

⇒ ler a matrícula, o nome e o salário (a leitura da matrícula 0 (zero) indica o fim dos dados - flag);

⇒ aumente o salário em K% (usando o procedimento AUMENTO) e exiba o salário aumentado.

P12.22. Escreva um programa Pascal que leia as três notas e o número de faltas de um aluno, calcule a sua média e determine e exiba a sua situação. Caso o aluno tenha mais de 10 faltas, ele está REPROVADO POR FALTA. Caso contrário, estará REPROVADO se sua média for menor que 5.0 ou APROVADO se sua média for superior a 5.0.

Observações:

- utilize uma função para calcular a média e um procedimento para determinar e exibir a situação do aluno;
- não utilize variáveis globais.

P12.23. Escreva um programa em Pascal que calcule o valor do coseno de X através de 20 termos da série abaixo:

$$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

Observações:

- O valor de x será lido;
- Devem ser implementados em funções independentes o cálculo do fatorial e o cálculo das potências.
- Utilize, como resultado do fatorial e da potência, o tipo EXTENDED, que é um tipo real, mas que permite valores muito grandes. Porém, inclua no início de seu programa (antes da cláusula USES) a diretiva {\$N+}.

P12.24. Escreva uma função chamada SEG para receber uma medida de tempo expressa em Horas, Minutos e Segundos e retornar esta medida convertida apenas para segundos.

Escreva um procedimento chamado HMS para receber uma medida de tempo expressa apenas em segundos e retornar esta medida convertida para horas, minutos e segundos.

Faça um programa que leia duas medidas de tempo (expressas em horas, minutos e segundos) e, usando a função SEG e o procedimento HMS, calcule e exiba a diferença (também em horas, minutos e segundos) entre elas.

P12.25. Escreva uma função chamada NOME\_MES que receba um valor inteiro N (de 1 a 12) e retorne um string contendo o nome do mês correspondente a N.

Faça um programa que leia uma data (no formato dia, mês e ano) e, usando a função NOME\_MES, exiba a data lida no formato abaixo:

EXEMPLO: Entrada: 23 11 1998

Saída: 23 de novembro de 1998

P12.26. Escreva uma função chamada DIAS\_ANO que receba 3 valores inteiros (DIA, MES, ANO) e retorne o número de dias decorridos no ano até o dia/mês/ano fornecido.

Escreva uma função booleana chamada DATA\_VALIDA que receba uma data (DIA, MÊS, ANO) e verifique se a data é válida (considerando os anos bissextos).

Faça um programa que leia 2 datas, no formato dia, mês e ano (as datas devem ter o mesmo ano) verificando se as mesmas são válidas (através da função DATA\_VALIDA), calcule e exiba a diferença de dias entre elas (usando a função DIAS\_ANO).

## 12.5. UTILIZANDO ARRAYS COMO PARÂMETROS

O Turbo Pascal não permite a declaração normal de um array como parâmetro formal em uma função ou procedimento. Porém, podemos utilizar o recurso da criação de novos tipos de dados disponível na linguagem Pascal. Veja o exemplo a seguir:

```
type VETOR = array[1..20] of integer;
```

Essa declaração deve ser colocada na área de declarações do programa, preferencialmente antes da área de declarações de variáveis (var). Após essa declaração, caso você necessite declarar uma variável do tipo array com 20 elementos inteiros, basta fazer o seguinte:

```
var V : VETOR;
```

Assim, para utilizar um array como parâmetro formal em uma função ou procedimento, você deve declará-lo na lista de parâmetros com o nome do tipo criado na declaração **type**.

Da mesma forma, isto também será necessário se o parâmetro formal for uma string com tamanho especificado. Veja o exemplo:

```
procedure PROC (s1:string; s2:string[10]);
```

Essa declaração causaria um erro de compilação por causa do tipo string[10]. Teríamos então que fazer o seguinte:

```
type STRING10 = string[10];
procedure PROC (s1:string; s2:STRING10);
```

## EXERCÍCIO RESOLVIDO

R12.03. Escrever um programa para ler um vetor de 50 elementos inteiros e determinar o valor médio dos seus elementos (utilizando um procedimento para ler um vetor e uma função para calcular a soma dos elementos do vetor).

```
program CALCULA_MÉDIA;

const
  N = 50;

type
  VETOR = array[1..N] of integer;

var
  VET : VETOR;
  MEDIA : integer;

procedure LEIA_VETOR (var V:VETOR);
var
  I : integer;
begin
  for I:=1 to N do
    readln(V[I]);
  end;

function SOMA_VETOR (V:VETOR) : integer;
var
  I,S : integer;
begin
  S := 0;
  for I:=1 to N do
    S := S + V[I];
  SOMA_VETOR := S;
end;

begin
  LEIA_VETOR(VET);
  MEDIA := SOMA_VETOR(VET) div N;
  writeln(MEDIA);
end.
```

## EXERCÍCIOS PROPOSTOS

- P12.22. Escreva uma função que receba um vetor  $V$  de 30 elementos inteiros, e retorne o maior elemento do vetor  $V$ .
- P12.23. Escreva uma função que receba um vetor  $V$  de 30 elementos inteiros, e retorne a moda do vetor, isto é, o elemento que mais ocorre dentro do vetor  $V$ .
- P12.24. Escreva uma função que receba um vetor  $V$  de 30 elementos inteiros, e retorne a quantidade de números positivos do vetor  $V$ .
- P12.25. Escreva uma função que receba um valor  $X$  do tipo inteiro e um vetor  $V$  de 80 elementos inteiros, e retorne o número de ocorrências de  $X$  dentro do vetor  $V$ .
- P12.26. Escreva uma função que receba um valor  $X$  do tipo inteiro e um vetor  $V$  de 80 elementos inteiros, e retorne o valor lógico TRUE se  $X$  existir dentro de  $V$  ou o valor lógico FALSE caso contrário.
- P12.27. Escreva um procedimento chamado PAR\_IMPARG que receba um vetor de 100 elementos inteiros e retorne a quantidade de números pares e de números ímpares contidas no mesmo.
- Faça um programa que leia 100 valores inteiros (armazenando em um vetor) e, usando o procedimento PAR\_IMPARG, determine e exiba a quantidade de números pares e de números ímpares.
- P12.28. Escreva um procedimento chamado LEIA que leia um vetor de 50 elementos inteiros (passado como parâmetro).
- Escreva uma função chamada MAIOR que receba um vetor de 50 elementos inteiros e retorne o maior elemento do mesmo.
- Escreva uma função chamada SOMA que receba um vetor de 50 elementos inteiros e retorne a soma dos elementos do vetor.
- Faça um programa que:
- ⇒ Usando o procedimento chamado LEIA, leia a idade de 50 pessoas;
  - ⇒ usando a função MAIOR criada, determine e exiba a maior idade;
  - ⇒ usando a função SOMA criada, calcule e exiba a idade média das 50 pessoas;
- P12.29. Modifique o programa da questão P8.08, criando os seguintes subprogramas:
- a) um procedimento que leia o gabarito da prova;
  - b) um procedimento que leia as respostas de um aluno.
  - c) Uma função que receba as respostas de um aluno e retorne a sua nota;
  - d) Uma função que receba a frequência de cada nota e retorne a nota que teve maior frequência absoluta, ou seja, a nota que apareceu maior número de vezes (supondo a inexistência de empates).

## 12.6. CRIAÇÃO DE UNITS

Uma **unit** é uma coleção de constantes, tipos de dados, variáveis, procedimentos e funções. Cada unit é como um programa Pascal separado. Ela é uma biblioteca de declarações que permite dividir seu programa e compilá-lo em partes separadas. Ela pode ter um corpo principal o qual é chamado antes do seu programa ser iniciado para preparar as "inicializações" necessárias.

Todas as declarações em uma unit estão normalmente relacionadas. Por exemplo, a unit CRT contém todas as declarações de rotinas relativas à tela do computador. O Turbo Pascal possui algumas units predefinidas: System, Overlay, Graph, Dos, Crt, Printer...

### 12.6.1. ESTRUTURA DE UMA UNIT

Toda unit deve iniciar pela palavra UNIT seguida do identificador da unit (que deve ter obrigatoriamente o mesmo do arquivo que será gravado em disco).

Em seguida aparece a seção INTERFACE, onde deve ser colocado tudo que será tornado público, isto é, tudo que os programas ou as outras units que usarem esta terão disponíveis.

A outra seção é a IMPLEMENTATION, onde deve ser colocado a implementação das rotinas que foram declaradas na INTERFACE e também tudo que for privado, isto é, que será local a esta unit, não estando disponíveis aos outros programas ou units que usarem esta.

Veja o formato geral:

```
UNIT <identificador>; {identificador deve ser o mesmo nome do arquivo}
INTERFACE
  uses <lista de units>      {opcional}
  <declarações públicas>    {só cabeçalho}

IMPLEMENTATION
  uses <lista de units>      {opcional}
  <declarações privadas>

  <implementação de proc. e funções>
  {corpo das funções e proc.}
End.
```

EXEMPLO:

```
Unit ROTINAS; {salva em disco com o nome ROTINAS.PAS}

Interface
  Function COMP_CIRC (R:real) : real;
  Function AREA_CIRC (R:real) : real;

Implementation
  Const
    PI=3.1416;

  Function COMP_CIRC (R:real) : real;
  {retorna o comprimento de uma circunferência}
  Begin
    COMP_CIRC := 2 * PI * R;
  End;

  Function AREA_CIRC (R:real) : real;
  {retorna a área de uma circunferência}
  Begin
    AREA_CIRC := PI * SQR(R);
  End;
End. (final da Unit)
```

## 12.6.2. UTILIZAÇÃO DE UNITS

O programa abaixo, localizado em outro arquivo, utiliza esta unit.

```
program TESTA_UNIT;

uses
  CRT,ROTINAS;

var
  RAIIO,COMP,AREA : integer;

begin
  clrscr;
  write('Digite o raio da circunferência: ');
  readln(RAIIO);
  COMP := COMP_CIRC(RAIIO);
  AREA := AREA_CIRC(RAIIO);
  writeln('Comprimento = ',COMP:0:2);
  writeln('Área = ',AREA:0:2);
  readkey;
end.
```

Assim que este programa for executado, o Turbo Pascal irá gerar o arquivo ROTINAS.TPU, que nada mais é do que o arquivo objeto que contém a unit ROTINAS compilada.

### Observação:

Problemas poderão ocorrer em virtude da localização do arquivo da unit. O Turbo Pascal permite definirmos o diretório (pasta) onde estarão armazenadas todas as units necessárias, através do comando Options...Compiler...Directories.

## EXERCÍCIO PROPOSTO

P12.30. Criar uma unit chamada BIBLIOT contendo os seguintes subprogrmas:

- uma procedimento chamado LEIA que receba um string S e uma variável inteira V, exiba o string S e, em seguida, leia a variável V;
- um procedimento chamado PAUSA que exiba a mensagem "Pressione qualquer tecla para continuar", seguida de um comando de espera por uma tecla digitada;
- uma função chamada MENOR que receba 3 valores inteiros e retorne o menor deles;
- uma função chamada MEDIA que receba 2 valores inteiros e retorne a média (inteira) deles.

P12.31. Faça um programa que leia as três notas (do tipo inteiro) dos 50 alunos de uma turma, elimine a nota mais baixa e calcule e exiba a média das duas notas restantes. Obs: faça uso dos subprogramas da unit BIBLIOT que você achar possível.

P12.32. Acrescente a unit BIBLIOT um procedimento chamado FINAL que exiba a mensagem "Pressione qualquer tecla para retornar ao Turbo Pascal", seguida de um comando de espera por uma tecla digitada. Substitua no seu programa o procedimento PAUSA pelo procedimento FINAL.

## 13. CONTROLE DO VÍDEO E DO TECLADO

Os procedimentos e funções mais usados que controlam o vídeo e o teclado no Turbo Pascal são:

- |              |          |                  |
|--------------|----------|------------------|
| – KEYPRESSED | – CLRSCR | – CLREOL         |
| – READKEY    | – GOTOXY | – TEXTCOLOR      |
| – DELAY      | – WHEREX | – TEXTBACKGROUND |
| – WINDOW     | – WHEREY |                  |

Todos esses comandos estão localizados na Unit CRT, sendo necessário para usá-los a declaração:

```
uses CRT
```

Detalharemos, a seguir, cada um destes comandos.

### 13.1. CONTROLE DO TECLADO

**KEYPRESSED** – Função que retorna o valor lógico TRUE caso tenha sido pressionada alguma tecla. Sua sintaxe é:

```
KEYPRESSED : boolean
```

**READKEY** – Função que retorna o valor (do tipo caracter) de uma tecla pressionada. Bastante utilizada quando queremos receber um caracter pelo teclado sem que o usuário precise teclar ENTER. Sua sintaxe é:

```
READKEY : char
```

Observação: As teclas de função (F1, F2, ...) nos retornam dois códigos, o primeiro sendo o caracter zero da tabela ASCII, e o segundo o da própria tecla de função.

**DELAY** – Procedimento que provoca uma pausa num determinado intervalo de tempo antes de ser executado o próximo comando. O intervalo de tempo especificado é sempre em milisegundo. Sua sintaxe é:

```
DELAY (tempo : word)
```

### 13.2. CONTROLE DO VÍDEO

**WINDOW** – Procedimento que nos permite definir o tamanho útil da tela. Quando definimos uma window, as coordenadas de referência de linha e coluna ficam relativas à nova window e sempre o canto superior esquerdo da tela é a posição (1,1), estando os procedimentos de vídeo também vinculados a esta nova janela. Sua sintaxe é:

```
WINDOW (x1, y1, x2, y2 : byte)
```

onde:

x1,y1 – são as coordenadas do canto superior esquerdo da janela

x2,y2 – são as coordenadas do canto inferior direito da janela

**CLRSCR** – Procedimento que limpa a tela e automaticamente coloca o cursor no canto superior esquerdo da mesma. Sua sintaxe é:

```
CLRSCR
```

**GOTOXY** – Este procedimento posiciona o cursor em um ponto determinado da tela, referenciado pelos eixos X e Y, ou seja, coluna e linha. Sua sintaxe é:

```
GOTOXY (x, y : byte)
```

**WHEREX** – Função que retorna a coluna em que se encontra o cursor (em relação à window atual). Sua sintaxe é:

```
WHEREX : byte
```

**WHEREY** – Função que retorna a linha em que se encontra o cursor (em relação à window atual). Sua sintaxe é:

```
WHEREY : byte
```

**CLREOL** – Procedimento que apaga todos os caracteres de uma linha do vídeo, que se encontram à direita do cursor. Sua sintaxe é:

```
CLREOL
```

**TEXTCOLOR** – Procedimento que determina a cor do texto que aparecerá no vídeo. Sua sintaxe é:

```
TEXTCOLOR (cor : byte)
```

As cores são representadas pelos valores inteiros de 0 a 15, que corresponde as seguintes cores:

0 → Preto	8 → Cinza escuro
1 → Azul	9 → Azul claro
2 → Verde	10 → Verde claro
3 → Ciano	11 → Ciano claro
4 → Vermelho	12 → Vermelho claro
5 → Magenta	13 → Magenta claro
6 → Marrom	14 → Amarelo
7 → Cinza claro	15 → Branco

Além destas 16 cores disponíveis, podemos somar a qualquer uma delas 128 para que o texto fique piscante.

**TEXTBACKGROUND** – Procedimento que permite selecionar a cor de fundo da tela. Sua sintaxe é:

```
TEXTBACKGROUND (cor : byte)
```

As cores são representadas pelos valores inteiros de 0 a 7, que corresponde as seguintes cores:

0 → Preto
1 → Azul
2 → Verde
3 → Ciano
4 → Vermelho
5 → Magenta
6 → Marrom
7 → Cinza claro

## EXERCÍCIOS RESOLVIDOS

R13.01. Escreva um procedimento que receba um string S e dois valores inteiros X e Y, e exiba o string S na coluna X e linha Y da tela.

```
procedure EXIBA (S:string; X,Y:byte);
begin
  gotoxy(X,Y);
  write(S);
end;
```

R13.02. Escreva um procedimento que receba um string S e um valor inteiro LIN e exiba o string S centralizado na linha L da tela. Obs: suponha que a "window" ativa está ocupando toda a tela.

```
procedure CENTRA (S:string; LIN:byte);
var
  TAM,COL : byte;
begin
  TAM := length(S);
  COL := ((80-TAM) div 2) + 1;
  gotoxy(COL,LIN);
  write(S);
end;
```

R13.03. Escreva um programa que leia uma frase (máximo de 30 caracteres) e faça a mesma "passear" pela tela do computador, isto é, faça a frase movimentar-se horizontalmente coluna a coluna, iniciando na coluna 1 e linha 1, e quando chegar à última coluna de cada linha, passar para a linha seguinte, até a última linha da tela. Crie também uma alternativa do programa parar ao se pressionar qualquer tecla.

```
program PASSEIO;
uses
  CRT;
const
  TEMPO = 100;
var
  FRASE : string[30];
  TAM,COL,LIN : byte;
begin
  clrscr;
  write('Frase: ');
  readln(FRASE);
  TAM := length(FRASE);
  COL := 1;
  LIN := 1;
  repeat
    clrscr;
    gotoxy(COL,LIN);
    write(FRASE);
    delay(TEMPO);
    COL := COL+1;
    if COL >= (80-TAM+1) then
      begin
        LIN := LIN+1;
        COL := 1;
      end;
  until (LIN > 25) or keypressed;
end.
```

## EXERCÍCIOS PROPOSTOS

P13.01. Escreva a finalidade e dê um exemplo de cada um dos comandos abaixo:

- a) KEYPRESSED
- b) READKEY
- c) DELAY
- d) WINDOW
- e) CLRSCR
- f) GOTOXY
- g) WHEREX
- h) WHEREY
- i) CLREOL
- j) TEXTCOLOR
- k) TEXTBACKGROUND

P13.02. Escreva um procedimento que receba uma string S e um valor inteiro C, e exiba a string S com a cor da fonte C.

P13.03. Escreva um procedimento que limpe uma área retangular da tela, sendo passados como parâmetros as coordenadas (coluna esquerda, linha superior, coluna direita, linha inferior).

P13.04. Escreva um procedimento que exiba uma janela na tela, sendo passados como parâmetros as coordenadas da janela (coluna esquerda, linha superior, coluna direita, linha inferior), e a cor de preenchimento da janela.

P13.05. Escreva um procedimento que exiba uma janela centralizada na tela, sendo passados com parâmetro o número de colunas e o número de linhas da janela, e a cor de preenchimento da mesma.

P13.06. Escreva um procedimento para exibir uma janela com sombra, isto é, uma janela de uma cor sobreposta a outra janela de cor diferente, com um pequeno deslocamento entre as duas. São passados como parâmetro as coordenadas da janela, a cor de preenchimento da janela e a cor da sombra.

P13.07. Escreva um programa para transformar a tela do computador em um verdadeiro arco-íris, ou seja, colocar em cada linha uma cor de fundo diferente.

P13.08. Escreva um programa que exiba uma tabela de conversão de graus Celsius para Fahrenheit, no intervalo de 1 a 100, variando de 1 em 1, dispostos em 5 colunas na tela.

## BIBLIOGRAFIA

FARRER, Harry et al. *Algoritmos estruturados*. Guanabara Dois, 1989.

FARRER, Harry et al. *Pascal estruturados*. Guanabara Dois, 1986.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. *Lógica de programação*. 2.ed. Makron Books, 2000.

GUIMARÃES, Ângelo de Moura. *Algoritmos e estruturas de dados*. LTC, 1985.

MANZANO, José Augusto N. G.; YAMATUMI, Wilson Y. *Programando em Turbo Pascal 7.0*. Érica.

PAIVA, S. R. *Algoritmos, Técnicas de Programação e Estruturas de Dados*. Apostila da ASPER, 1995.

RINALDI, Roberto. *Turbo Pascal 7.0: comandos e funções*. Érica, 1993.

SCHIMTZ, Eber A.; TELES, Antonio A. S. *Pascal e técnicas de programação*. LTC, 1985.

TREMBLAY, Jean-Paul; BUNT, Richard B. *Ciência dos computadores: uma abordagem algorítmica*. McGraw-Hill, 1983.

WIRTH, Niklaus. *Programação Sistemática em Pascal*. Campus, 1989.

WORTMAN, Leon A. *Programando em Turbo Pascal com Aplicações*. Campus, 1988.