OS ALGORITMOS,

O PASCAL

E

A MATEMÁTICA

MANUEL PEDRO DE ALMEIDA DUARTE

ÍNDICE

1. PROGRAMAÇÃO ESTRUTURADA

1.1. Noção de Algoritmo

Enunciemos o seguinte problema:

Dada uma sequência de cinco números positivos, determinar qual deles é o maior.

2, 7, 11, 5, 8,

Para a resolução deste problema vejamos quais os passos desenvolvidos mentalmente e que nos conduziram à solução:

- Ler o 1° número;
- Memorizá-lo;
- Ler o 2° número;
- Comparar o 2° número com o que estava na memória;
- Se o que tinha na memória era menor que o 2° número lido, então colocar na memória o 2° e esquecer o 1°;
- Se o que tinha na memória é maior do que o 2° número lido, então esquecer o 2° número e na memória permanecer o 1°;
- Ler o 3° número;
- Comparar o 3° número lido com o que estiver na memória;

... e assim sucessivamente.

É assim que nós "ensinamos" o computador como ele deve proceder para resolver um problema. Podemos dizer que um computador é um escravo extremamente rápido, extremamente obediente e, em certo sentido, extremamente estúpido.

Os computadores são "ensinados" através de algoritmos. Como no exemplo visto, é a **"sequência de passos" que aponta o caminho da resolução de um problema proposto chama-se ALGORITMO.**

Podemos então definir algoritmo como:

> Estabelecimento de uma estratégia (ou "receita") para que o computador resolva o problema proposto;
> ou
> ***Conjunto de instruções para a execução passo a passo de um dado problema.***

São exemplos de algoritmos:

- **Uma receita culinária**: o preparo de um prato complicado é desdobrado em etapas simples que qualquer pessoa saiba cozinhar poderá entender facilmente;
- **A coreografia de um "ballet":** é o desdobramento de uma dança complexa numa sucessão de posições e passos básicos de ballet. O número desses passos e posições básicos é muito pequeno mas, quando reunidas de diversas maneiras, poderemos criar uma variedade infinita de danças;
- **A mudança de um pneu** de um automóvel. Note que há mais do que um procedimento para resolver o problema: mudá-lo o próprio ou mandar chamar quem mude.

Do mesmo modo um computador pode, por meio de algoritmos controlar um processo de fabricação ou a trajectória de um satélite. É evidente que os algoritmos para esses processos em grande escala são muito complexos, mas são constituídos com peças elementares.

O conceito central da programação e da ciência da computação é o do algoritmo. Programar é basicamente construir algoritmos. **Wirth**, o pai da Línguagem Pascal, apresenta a programação estruturada como ***"a arte ou técnica de construir e formular algoritmos de uma forma sistemática".***

Programas são - segundo Wirth - ***"formulações concretas de algoritmos abstractos, baseados em representações e estruturas específicas de dados".***

1.2. Conceito de Acção

> **Definição:**
> A acção é um acontecimento que a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e bem definido.

Vejamos com um exemplo. Escolher um valor numérico L e escrever os termos da sequência de Fibonacci inferiores a L.

> **Sucessão de Fibonacci:**
> Os dois termos iniciais são iguais a 1 e, cada termo seguinte é igual à soma dos dois termos imediatamente anteriores.

Consideremos três situações distintas:

a) Se L= 50 então a resposta será:
1 1 2 3 5 8 13 21 34
b) Se L= 13 então a resposta será:
1 1 2 3 5 8
c) Se L= 1 então "nada se escreve"

A escolha dos valores L=50, 13 ou 1 é totalmente aleatória e portanto não é uma acção. Porém ao escrever os termos da sequência de Fibonacci inferiores a L, é uma acção: partiu-se de um estado inicial (o valor escolhido para L) e, após um período de tempo limitado (alguns segundos), produziu um estado final (o que foi escrito ou não), previsível e bem definido.

Estas três acções são distintas, foram executadas por pessoas diferentes, ocorreram em instantes diferentes, partiram de valores iniciais diferentes e produziram resultados diferentes.

Mas apesar disto, mesmo assim, podemos reconhecer nas três acções distintas um mesmo **Padrão de Comportamento**, a subordinação a uma mesma **Norma de Execução**.

É como se as três acções tivessem sido executadas em obediência ao **Comando**:

"Escreva os termos de Fibonacci inferiores a L."

1.3. Definição de Algoritmo

Algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de acções.

Geralmente, um algoritmo destina-se a resolver um problema. Portanto ***um algoritmo fixa um padrão de comportamento a ser seguido, uma norma de execução a ser trilhado, para se atingir, como resultado final, a solução do problema***.

1.4. Características de Algoritmos

◊ **Não são únicos**. Um problema pode ser resolvido de mais do que uma maneira.

◊ **Finitos.** Qualquer algoritmo quando executado deve terminar após um certo tempo. É fácil fazer programas que nunca mais terminam, a não ser que sejam interrompidos. Devem ter Príncipio e Fim.
Um algoritmo não pode conter um comando tal como :
"Escreva todos os termos da sequência de Fibonacci."

◊ **Definidos**. Cada passo (comando) de um algoritmo deve ser claro, e não ambíguo. Não podem existir instruções tal como:
"Se x for grande então incremente algumas unidades"
"Escreva qualquer coisa"

◊ **Eficaz.** Deve tomar-se em conta a eficiência do algoritmo que tem a ver com o tempo de resposta, ocupação da memória e velocidade de execução.

◊ **Entrada/ Saída** de dados. Um algoritmo recebe normalmente certos dados, trabalha-os (processa-os) e fornece um resultado. Logo deve sempre prever-se a entrada e saída de dados.

Isto leva-nos a falar de:

Estruturas de Dados

Se um programa processa dados, esses dados devem estar na memória do computador. Surgem então duas questões:

-O que é que vamos representar na memória de um computador?

-Como representar alguma coisa no computador?

A primeira resposta que surge é:

-Depende daquilo que nós queremos realizar com os dados.

Daí que seja muito difícil separar o estabelecimento do "algoritmo" da "estrutura de dados" que vai ser utilizada para representação da informação a manipular.

Surge então a definição de programa dada por **Wirth:**

PROGRAMA = ALGORITMO + ESTRUTURA DE DADOS

1.5. Representação de Algoritmos

Existem várias maneiras de representamos algoritmos. Delas destacamos as seguintes:

⇒ Linguagem informal;
⇒ Linguagem de programação;
⇒ Fluxograma;
⇒ Diagrama de Chapin;
⇒ Linguagem Pseudo-algoritmica (LPA - Portugol).

Na construção dos algoritmos iremos utilizar esta última, uma vez que é ela o melhor suporte para a programação estruturada. Usaremos o fluxograma só em casos em que ele seja útil para o entendimento da semântica de alguma estrutura de controlo.

1.6. Refinamentos Sucessivos

Um algoritmo é considerado **completo** se os seus comandos (instruções ou ordens) forem de entendimento do seu destinatário.

Num algoritmo, um comando que não for de entendimento do destinatário terá de ser desdobrado em novos comandos que constituirão um **refinamento** do comando inicial.

Se um algoritmo é formado não apenas por um comando, mas por vários, isto significa que na sua execução não se consideram apenas o **Estado Inicial** e o **Final** de uma acção dele resultante, mas que se consideram também **Estados Intermediários** que delimitam as acções decorrentes de cada comando.

Por exemplo, o comando

```
1) "Escreva os termos de Fibonacci inferiores a L."
```

desdobra-se sucessivamente em:

```
Ref. 2) "Receba o valor L
Processe os 2 primeiros termos
Preocesse os termos restantes."
```

Um algoritmo e os seus refinamentos são formados por comandos que determinam as acções a serem executadas pelo seu destinatário e por estruturas

de controle que determinam a ordem em que os comandos devem ser executados, se devem ser executados ou não e quando devem ser repetidos.

No refinamento 2) temos a **Estrutura Sequencial** segundo a qual os comandos devem ser executados um após a outro na mesma ordem em que foram escritos.

Quando refinar um comando? Quando ele for um "tanto vago"; isto é , quem ovai executar não sabe bem como deve fazê-lo.

```
Ref. 3 - "Processe os 2 primeiros temas."
     Atribua o valor 1 ao 1° termo;
     Se ele for menor  que 1 então
          Escreva-o
     Atribua o valor 1 ao 2° termo;
     Se ele for menor que 1então
          Escreva-o
```

Neste refinamento, aparece a 2ª estrutura de controle: **A Estrutura Condicional:**

```
Se condição então comando
```

Uma 3ª estrutura de controlo, a **Estrutura de Repetição** será necessária ao se desdobrar o comando

```
Ref. 4 - "Processe os termos restantes"
     Repita
     Calcule novo termo somando os dois anteriores
     Se novo o termo menor do que 1 então Escreva-o
     Até que "novo termo maior ou igual a 1"
```

Após estes refinamentos o algoritmo pode ser considerado completo e teremos:

```
Algortitmo (escrita dos termos de Fibonacci < a L)
     Receba o valor L
(Processamento dos 2 primeiros termos)
     Atribua o valor 1 ao primeiro termo
     Se ele for menor do que L então escreva-o
     Atribua o valor 1 ao segundo termo
     Se ele for menor do que L então escreva-o
(Processamento dos termos restantes)
     Repita
          Calcule no termo somando os 2 anteriores
          Se novo termo for menor a L então escreva-o
     Até que novo termo seja supeior ou igual a L
Fim
```

Esta técnica dos refinamentos sucessivos é a defendida pela Programação Estruturada. Do Inglês "**Top Down**". Trata-se da abordagem de um problema em que a sua análise vai sendo feita do geral para o particular.

1.7. A crise do Software

Nos finais dos anos 60, uma altura em que pareciam abrir-se as portas ao desenvolvimento de grandes sistemas de software, ao termos as máquinas e os instrumentos de programação necessárias, constatámos cada vez com maior clareza que havia um problema: o desenvolvimento das aplicações era finalizada sempre com demora, custava mais do que o previsto, e muitas vezes não era útil para as necessidades dos clientes, pelo que com frequência de mais (e foram oferecidas cifras por volta dos 50%) eram abandonadas as aplicações e não chegavam nunca a entrar em funcionamento ou eram dedicados grandes esforços (não previstos nem orçamentados) para as emendar e para as adequar às necessidades. Estavamos em finais dos anos 60; era a crise do Software.

A resposta a esta crise foi o desenvolvimento de uma nova disciplina, chamada "**Engenharia do Software**".

A ideia básica de que parte é que o desenvolvimento do Software não é uma arte (apenas), mas sim uma ciência, com uma série de princípios que utiliza umas metodologias e ferramentas, e dispõe de umas métricas capazes de avaliar diferentes aspectos do processo.

1.8. Programação Estruturada: Princípios Gerais

A Engenharia de Software, desenvolveu um conjunto de princípios gerais para aplicação sistemática à construção de algoritmos. São eles:

- A Decomposição;
- A Abstracção;
- A Capsulação;
- A Ocultação de Dados;
- O Tipado;
- A Baixa Coesão Entre Módulos.

Basicamente, a Programação Estruturada consiste numa metodologia de projecto de algoritmos visando:

- ◊ Facilitar a escrita dos programas (desenvolvimento dos algoritmos);
- ◊ Facilitar a leitura (o seu entendimento) pelos humanos;
- ◊ Antecipar a comprovação da sua correcção (verificação à priori);
- ◊ Facilitar a manutenção e modificação dos algoritmos;
- ◊ Permitir que o seu desenvolvimento possa ser empreendido simultaneamente por uma equipa de pessoas.

Para atingir estes objectivos, a Programação Estruturada preconiza que:

a) Os algoritmos devem ser desenvolvidos por Refinamentos Sucessivos (Top Down), partindo de uma descrição geral e gradativamente mais particular;

b) Cada refinamento deve constituir uma módulo funcional organizado de preferência num sistema hierárquico;

c) Em cada módulo deve ser usado um número limitado de comandos e de estruturas de controle;

d) Todos os programas desenhados utilizando as Estruturas de controle: Sequência; Condicional (alternativa); Repetição (ciclo).

A complexidade de um sistema de software é uma medida do número de seus componentes e do grau de interacção entre eles. Por tal razão Dijkstra, o indiscutível iniciador da Programação Estruturada, afirmava:

> *-"A arte de programar consiste na arte de organizar e dominar a complexidade."*

1.9. Fases da Resolução de um problema

Especificação. Definição rigorosa do problema. Responder às questões:
- o que tenho?
- o que quero?

Projecto. Escolha do algoritmo.
Codificação. Tradução para a Linguagem do Programa.
Verificação. Correção e testes.

Estas são as etapas a seguir quando pretendemos construir algoritmos de acordo com os objectivos da Programação Estruturada, e que utilizam as técnicas por ela defendidas.

Os programadores de hoje não podem mais usar técnicas de programação dos computadores de válvulas, como vemos em muitos casos.

As novas tecnologias do Hardware desenvolveram-se muito mais rapidamente do que as novas técnicas do Software (Programação).

Se a Indútria Automobilística tivesse evoluído como o Hardware, hoje teríamos um carro que custava menos do que 150$00, andaria 1.000.000 de Km com 1 litro de gasolina e atigiria a velocidade de 100.000.000 Km/hora.

2. LINGUAGEM AUXILIAR

2.1 Introdução

A Liberdade de expressão é essencial para o desenvolvimento da criatividade em praticamente todas as actividades humanas. Segundo Wirth, "nós precisamos reconhecer a forte e inegável influência que a nossa linguagem exerce em novos caminhos de pensamento, e que de facto define e delimita o espaço abstracto no qual nós formulamos - damos forma - aos nossos pensamentos."

A tentação seria então usar directamente o Português, na construção de algoritmos. Contudo o uso de uma técnica formal para notação de algoritmos proporciona as seguintes vantagens:

◊ Afasta a possibilidade de ambiguidade de tal modo que dado um algoritmo descrito segundo esta técnica e um estado inicial, a sua execução nos leva sempre ao mesmo resultado, através dos mesmos caminhos;

"... Linguagens naturais, não formalizadas como são, derivam, tanto a sua fraqueza como o seu poder, de sua imprecissão e carácter vago."

◊ Só se aceita um algoritmo quando este é acompanhado de uma argumentação convincente. Só o formalismo de uma linguagem exacta e precisa permite dar essa argumentação.

Estamos, então, diante de um impasse: por um lado, precisamos de liberdade de expressão para formular nossos raciocínios e, por outro necessitamos de uma notação formal para não introduzir ambiguidades.

A solução está na escolha de estruturas adequadas, bem projectadas, com efeitos bem definidos, e que, além disso, não restrinjam a criatividade do programador.

A Linguagem Pseudo Algoritmica (LPA), pseudo código ou PORTUGOL (Português Algorítmico): simbiose de Português com ALGOL e PASCAL, é uma restrição da Língua Portuguesa.

A ideia é permitir que com um conjunto básico de estruturas primitivas seja possível ao projectista pensar no problema e não na máquina que vai

executar o algoritmo e, por outro lado, não fique muito distante desta mesma máquina.

Por outras palavras: que o projectista possa pensar na solução do problema e que esta solução seja facilmente implementada no computador.

Em toda a linguagem, as frases constituídas envolvem dois aspectos:

- A SINTAXE (tem a ver com a forma)
- A SEMÂNTICA (tem a ver com o significado)

Para podermos começar a resolver problemas vamos explicar a Sintaxe e a Semântica da Linguagem Pseudo Algoritmica, sem contudo querermos apresentar normas únicas, nem tão pouco universais. Muitos aspectos de pormenor em Linguagem Pseudo Algoritmica variam de autor para autor relativamente à sintaxe. Contudo a semântica é basicamente a mesma.

2.2. Tipos de Dados: Constantes e Variáveis

Constantes

Na Linguagem Auxiliar, uma constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução do programa.

As constantes, valores atómicos, "entes" que a linguagem máquina de cada computador manipula podem ser de 3 tipos fundamentais:

Numéricos:

Reais:	`REAL`	(-5, 0.5, 3.27, ...)
Inteiros:	`INTEGER`	(1, -5, 0, 327, ...)

Lógica:

Só existem duas constantes deste tipo: TRUE, FALSE. Na linguagem Pascal são designadas de BOOLEAN.

Alfanuméricos:

Do tipo caracteres `CHAR` ("a", "1". "&", "A", ...);

Na maioria das implementações de Pascal já existe o tipo `STRING` que corresponde a uma sequência de caracteres ("Manuel Pedro", "387898", 23/09/56", ...).

Em Pascal, podemos afirmar que ***uma constante é uma macro para algum valor que não muda de valor e cujo valor é definido a seguir a CONST.***

Variáveis

Podemos imaginar uma variável como o nome de um local onde se pode colocar qualquer valor do conjunto de valores possíveis do tipo básico associado.

Toda a variável é identificada por um nome ou **Identificador**: elemento básico da linguagem. É formado por um ou mais caracteres, sendo o 1° obrigatoriamente uma letra, e os caracteres seguintes letras ou dígitos, não sendo permitido o uso de símbolos especiais.

Declaração de variáveis

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo numéricas, lógicas ou alfanuméricas.

Em Pascal o tipo de variável deve ser declarado após VAR.

Exemplos:

```
Var
     X1, K, T48D     :Inteiro;
     Nome, Morada    :String;
```

De um modo geral teremos

```
<Lista de indentificadores > : <nome do tipo>;
```

A Semântica de uma declaração de variáveis corresponde à criação de locais de memória, rótulada com um nome de variável (identificador) e marcada com o tipo de valores que ela pode conter.

Observações:

◊ Os nomes das variáveis devem reflectir a natureza dos valores que nelas estão armazenadas. Ajuda a que o programa seja entendido pelo humano que o ler.

◊ **Palavra-chave** é toda a palavra que tem significado próprio, independente do algoritmo em que esteja inserida. Por isso, palavras-chave não podem ser usadas como identificadoras de variáveis.

Expressões Aritméticas

Expressão aritmética é toda a expressão cujos operadores são aritméticos e cujos operandos são constantes e/ou variáveis do tipo numérico.

Os operadores aritméticos de **Pascal** são:

+	Adição inteira e real;
-	Subtracção inteira e real;
×	Multiplicação inteira e real;
DIV	Divisão inteira;
MOD	Resto da divisão inteira;
/	Divisão real.

Funções

Além das operações básicas anteriormente citadas, podem-se usar nas expressões aritméticas algumas funções comuns da Matemática, já implementadas no Pascal, e na maioria das Linguagens de Programação. São funções de argumento real e resultado real.

Alguns exemplos de funções já implementadas na maioria das linguagens de programação são dados na seguinte tabela:

ABS(X)	$\lvert x \rvert$	TRUNC(3.5)=3
SQR(X)	x^2 x.x	TRUNC(3.8)=3
SQRT(X)	$\sqrt{x}$	TRUNC(3.3)=3
SIN(X)	sen x	ROUND(3.5)=4
COS (X)		ROUND(3.8)=4
ARCTAN (X)		ROUND(3.3)=3
EXP (X)	e^x	TRUNC(-1.5)=-2
Ln(X)	$Log_e x$	ROUND(-1.5)=-1
ROUND (X)	Arrendonda X	***Convertem reais em imteiros***
TRUNC (X)	Trunca X	

Expressões Alfanuméricas

Expresão alfanumérica é formada por operadores alfanuméricos e por operandos que são constantes e ou variáveis do tipo literal.

Temos por exemplo de operação alfanumérica a concatenação. Por exemplo:

Se A= 'Manuel' e B= 'Pedro',

Então A + B = 'ManuelPedro'

O operador de **concatenação** em Pascal é (+). Notar também a necessidade da utilização de pelícas para delimitação da string.

A concatenação é apenas um exemplo pois as operações entre variáveis alfanuméricas são bastante diversificadas e dependem das características de cada linguagem de programação. Normalmente são encontradas sob a forma de Funções e/ou Procedimentos, sendo as mais comuns aquelas que se seguem:

- ◊ O comprimento da variável (número de caracteres);
- ◊ Os *n* primeiros caracteres de uma variável alfanumérica;
- ◊ Os *n* últimos caracteres de uma variável alfanumérica;
- ◊ A posição de um dado carater na variável.

Expressões Lógicas

Expressão lógica é a expressão cujos operadores são lógicos e cujos operandos são relações (ou proposições lógicas), constantes e/ou variáveis do tipo lógico.

Uma relação ou expressão relacional, é uma comparação entre dois valores do mesmo tipo básico.

Os Operadores Relacionais, utilizados na maioria das linguagens de programação (incluindo o Pascal) são:

=, <>, >, <, >=, <=

Embora esteja fora do âmbito deste curso o estudo da **Álgebra de Boole**, torna-se necessário rever alguns conceitos fundamentais.

Uma expressão booleana é uma expressão que só toma um de dois valores lógicos: verdadeiro ou falso.

Exemplos:

2 > 1 é uma expressão sempre verdadeira;

X > 1 é uma expressão verdadeira ... dependendo do valor de x.

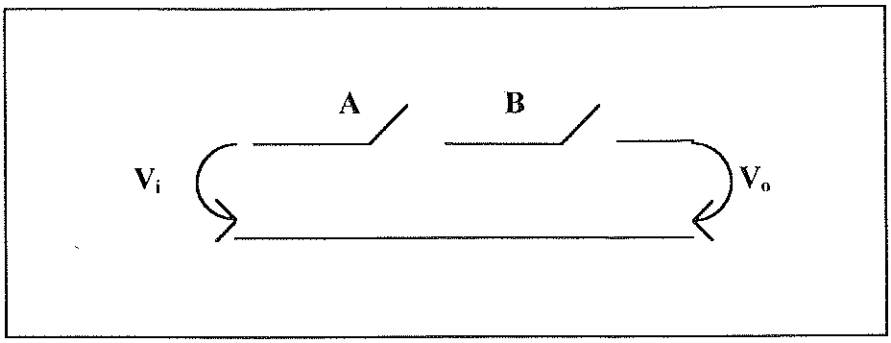
Em Álgebra de Boole, existem os seguintes operadores lógicos: E; Ou; Não; ou exclusivo. No Pascal teremos AND, OR, NOT.

Como em $\Re$ as operações + e × definem um corpo, também as operações **OU** (+) e **E** (×), definem um corpo no conjunto de todas as proposições lógicas.

AND

A ∧ B só é verdadeiro se A e B forem simulteanemente verdadeiras.

A operação conjunção é materializada num circuito **série.**



A tabela de verdade da operação lógica AND

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

Exemplos:

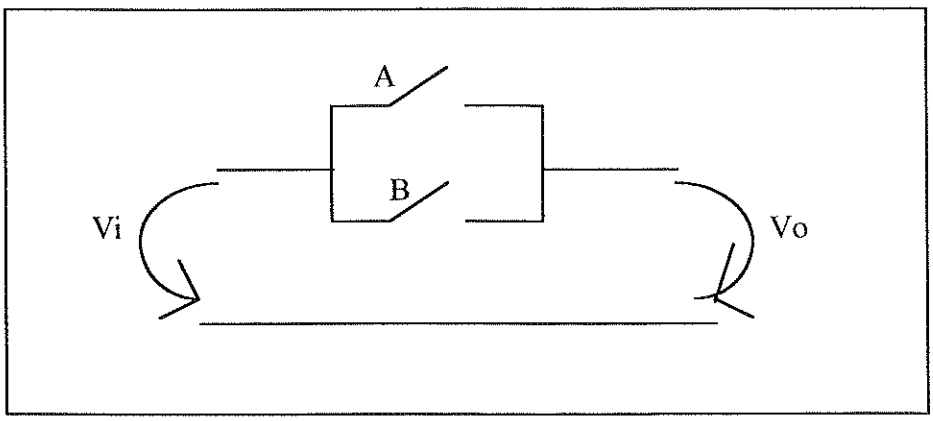
1) $x > 1 \wedge x > 3$ significa que $x > 1 \wedge x > 3$ simultaneamente. Logo, é equivalente a $x > 3$.

2) $x > 1 \wedge x < 0$ significa que $x > 1$ e simultaneamente $x < 0$. Logo, é sempre falso.

OR

A V B é verdadeiro, se pelo menos, uma das proposições o for.

A operação disjunção é materializada num circuito **paralelo.**



A tabela de verdade da operação lógica OR

A	B	A V B
0	0	0
0	1	1
1	0	1
1	1	1

NOT

Não A $\left(\overline{A}\right)$ ***é Verdadeira se esó se A for Falso.***

A sua implementação é um circuito normalmente fechado. Quando acionado o interruptor A abre.

A tabela de verdade da negação

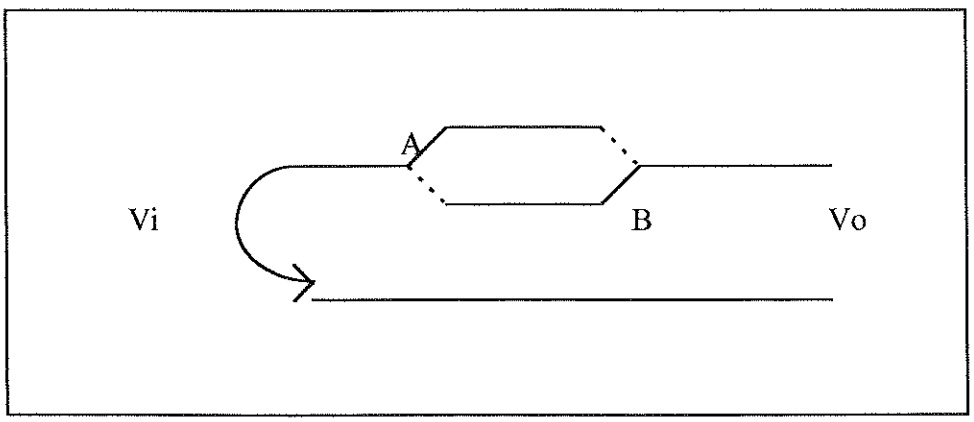
A	$\overline{A}$
0	1
1	0

OU EXCLUSIVO

A ou exclusivo B é verdadeiro se uma e só uma das proposições for verdadeira.

A implementação da operação lógica ou exclusivo é um circuito do tipo comutador de escadas, e a sua tabela de verdade é a que consta na tabela seguinte:

A	**B**	**A ou exc. B**
0	0	0
0	1	1
1	0	1
1	1	0



Leis de Morgan

Analisemos um caso de Lógica Booleana do dia-a-dia.

a) Condição A - Está Sol
b) Condição B - Ter dinheiro
c) Condição C - Ir à praia

Consideremos agora a seguinte frase: ***"Se estiver sol e tiver dinheiro então vou à praia"***; ou seja,

$$se\ A \wedge B\ então\ C.$$

Pergunta-se: - Em que caso **não** vou à praia?
A resposta evidente será:
-Caso **não** esteja sol **ou** caso **não** tenha dinheiro.

Repare-se que para não ir à praia basta que uma delas não se verifique.

Assim

A negação de um AND é um OR

$$\overline{A \cap B} \equiv \overline{A} \cup \overline{B}$$

E do mesmo modo

A negação de um OR é um AND

$$\overline{A \cup B} \equiv \overline{A} \cap \overline{B}$$

Prioridade dos Operadores

Há expressões em que aparecem operadores de diferentes tipos. Eles têm diferentes prioridades que estabelecem a ordem pela qual as operações devem ser executadas. São as seguintes as prioridades:

1° Aritméticos (dentro destes também há prioridade)
2° Relacionais
3° NOT
4° AND
5° OR

Observação: Quando existem dúvidas deve utilizar-se o parentesis para estabelecer a prioridade desejada para solução do problema.

2.3. Instruções da Linguagem Algoritmica (Comandos)

Atribuição

A atribuição permite fornecer um valor a uma certa variável. A natureza deste valor tem de ser compatível com o tipo da variável na qual está sendo armazenada.

A sua sintaxe é

```
< identificador > ← < expressão >;
```

O símbolo ← tem carácter imperativo.

Exemplos:

```
X ← 3 (leia-se: X passa a ser 3 ou a valor);
DELTA ← B × B - 4 × A × C;
P ← P + 1;
NOME ← 'Manuel Pedro';
RESTO ← N mod 2;
ALFA ← SIN(BETA);
```

Observações:

1) A atribuição **X ← X + 1** (***leia-se X passa a ser X + 1***) significa que a variável X passa a valer + 1 do que valia.

2) A troca de valores entre duas variáveis A e B. Por exemplo, temos

$$A=2 \text{ e } B=3$$

e queremos

$$A=3 \text{ e } B=2$$

$$\left.\begin{array}{l} A \leftarrow B \\ B \leftarrow A \end{array}\right\} \Rightarrow A = 3 \wedge B = 3 \rightarrow Erro$$

Este procedimento está errado. Precisamos de uma variável temporária C, para não perder o primeiro conteúdo.

$$\left.\begin{array}{lll} C & \leftarrow & A \\ A & \leftarrow & B \\ B & \leftarrow & C \end{array}\right\} \Rightarrow C = 2 \wedge A = 3 \wedge B = 2 \rightarrow Correcto$$

3) O resultado da expressão do lado direito de um comando de atribuição deve ser coerente com o tipo declarado para a variável do lado esquerdo.

Exemplos:

X ← A<B Esta instrução só faz sentido se X tiver sido definido do tipo Boolenao;

X ← A/B X deve ser uma variável real;

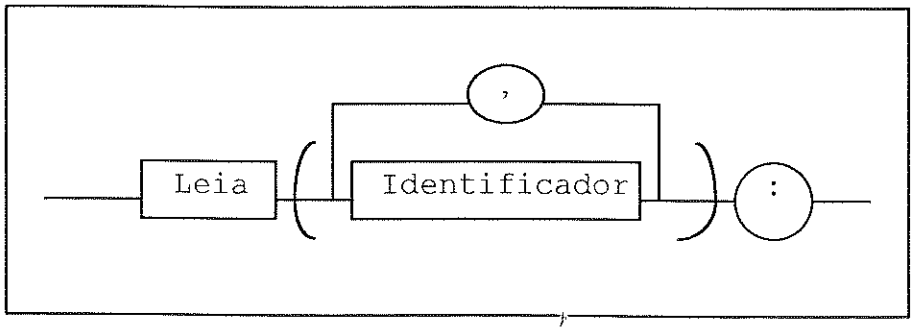
4) As variáveis que constituem numa expressão devem estar previamente definidas.

Leitura (INPUT)

Para escrever algoritmos mais gerais, será necessário obter dados do ambiente externo para o algoritmo (por exemplo, a partir do teclado).

A sintaxe da instrução de leitura é a seguinte:

```
Leia (<lista de variáveis>);
```



Exemplos:

```
Leia (x, y,  c, soma, XIA);
```

Não é aceite uma instrução de leitura que obrigue a um cálculo prévio, como por exemplo `Leia (2*X);`

Escrita (OUT PUT)

Para fornecer dados ao ambiente exterior ao algoritmo é necessário este tipo de comando que estabelece a ligação com um periférico da saída. O standart é o monitor.

A sintaxe da instrução de escrita é a seguinte:

Escreva (<lista de variáveis e/ou expressões>);

Exemplos:

```
Escreva (X, Y, SOMA)
Escreva (" O valor da soma:", SOMA)
Escreva ("O pedro tem", Idade, "Anos")
Escreva (2 * X + y + sin (Z))
```

Para calcular o dobro de um número, podemos construir os dois algoritmos seguintes:

Antigamente:

```
leia (X);
y ← 2*X;
escreva (Y);
```

Actualmente:

```
leia (X);
escreva (2*X);
```

As intruções até aqui apresentadas permitem a implementação de algoritmos constituídos apenas pela **Estrutura de Controlo Sequencial**: **conjunto de comandos que serão executados num sequência linear de cima para baixo**.

A sintaxe desta Estrutura em Linguagem algoritmica será:

```
Início

    < declaração de variáveis>;
    <lista de comandos>;

Fim.
```

Instrução Condicional

Esta instrução permite implementar a Estrutura de Controlo Condicional (ou Alternativa). Utiliza-se quando a acção a ser executada depender de uma inspecção ou teste lógico.

A sua sintaxe é:

Alternativa Simples

```
Se < Expressão Booleana> Então <Instrução>;
```

ou

Alternativa Composta

```
Se <Expressão Booleana> Então <Instrução>
                        Senão < Instrução>;
```

Exemplo:

Desenvolva um algoritmo que actualize os vencimentos dos empregados de um determinada empresa 15%, desde que trabalhem para esta empresa a tempo inteiro há mais de 10 anos.

A solução será um algoritmo que actualize o vencimento em função do resultado lógico de uma condição.

Em L.P.A., poderemos ter o seguinte algoritmo:

```
Início

     Leia (A_SERV);
     Leia (VENC.);
     Se A_SERV>=10 Então VENC ← VENC*1.15;
     Escreva(Venc)

Fim
```

De um modo geral, a sintaxe da instrução condicional em L:P:A, será:

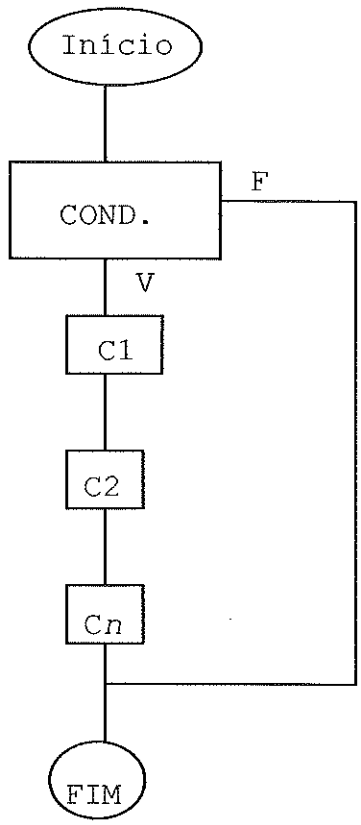
```
Início

      Se <condição>
      Então     C1
                C2
                ...
                Cn
      fimse;

Fim.
```

Note-se que a linha que contem a palavra "`fimse`" não é um comando em si mesmo, mas apenas é usada para delimitação do fim do bloco de instruções que é executado quando a condição é verdadeira. Nas linguagens de programação propriamente ditas esta linha não é necessária.

Na representação de Fluxograma:



Alternativa composta

Exemplo:

Desenvolva um Algoritmo que actualize os salários dos empregados de uma empresa de acordo com o seguinte:

Se o salário é inferior ao salário mínimo nacional deve ser actualizado para o maior dos valores: ou o salário mínimo nacional, ou o salário mais 15%; se o salário é superior ou igual ao salário mínimo nacional, este deve ser aumentado de uma outra taxa 10%.

Resolução:

Numa versão mais simplificada, consideremos em alternativa duas taxas distintas T1 e T2. Em Linguagem Algoritmica teremos:

```
Início

     Leia (V, VMN, T1, T2)
     Se V<VMN  então V←V*(1+T1)
               senão V←V*(1+T2)
     Escreva (V)

fim.
```

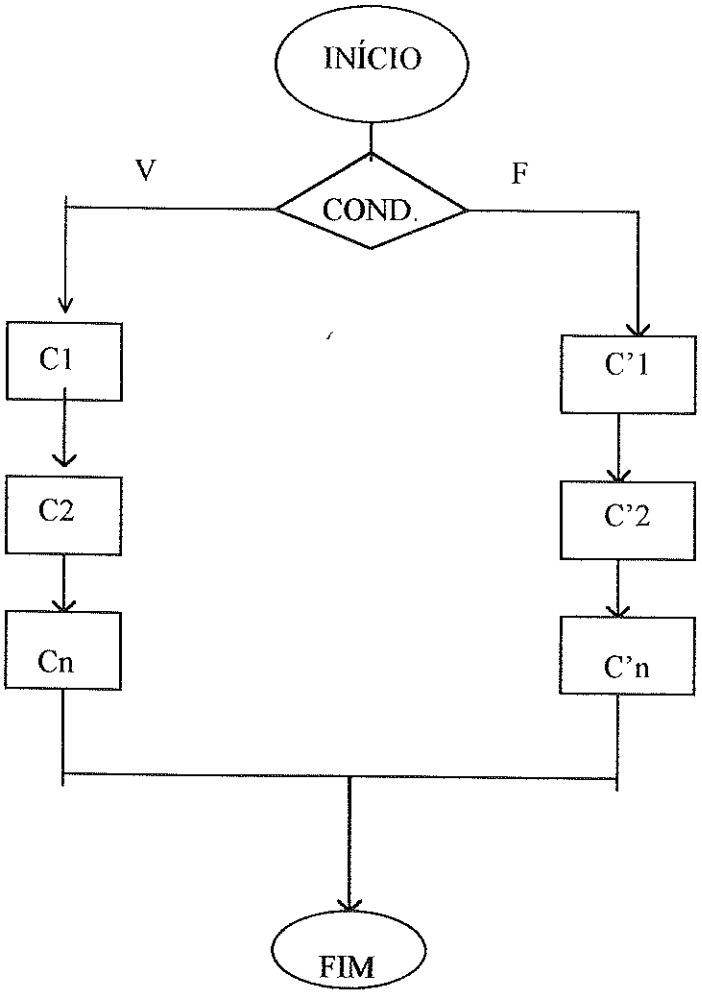
De um modo geral, a sua sintaxe será:

```
Início

     Se   <CONDIÇÃO>
     Então     C1
               C2
               ...
               Cn

     Senão     C'1
               C'2
               ...
               C'n
     fimse;
Fim.
```

Em Fluxograma teremos



A título de exemplo apresentamos a solução do problema da actualização dos salários, mostrando o algoritmo já traduzido em Pascal.

```
Lista das variáveis:
     s - Salário do empregado
     smn - Salário mínimo nacional
     t - Taxa de juro
```

```
Program ActualizaSalarios;

uses
     winCrt;

Type
     Percentagem = 1..100;

Var
     s,smn:     real;
     t:         Percentagem;

{------------Procedimento Pede Valor--------------}

Procedure PedeValores;

begin
     writeln('PROGRAMA DE ACTUALIZAÇÃO DE SALÁRIOS');
     writeln('Diga qual o valor do salário
correspondente ao empregado que pretende calcular.');
     readln(s);
     write('Qual é o valor do salário mínimo
nacional?');
     readln(smn)
     write('Diga qual a taxa de juro que
pretende.(Valor entre 0 e 100)')
     readln(t);

end;

{----------------Programa Principal---------------}

begin

     Pede valores;
     If s<smn then
          begin
               s:=s+s*t/100;
               If s<smn then s: =smn;
               end
               else s:=s+s*t/2/100;
     write('valor do salário actual:',s:10:2,'.');

end.
```

2.4 Instruções de Ciclo

Os próximos três comandos permitem implementar a **Estrutura de Controlo de Repetição.**

Repita

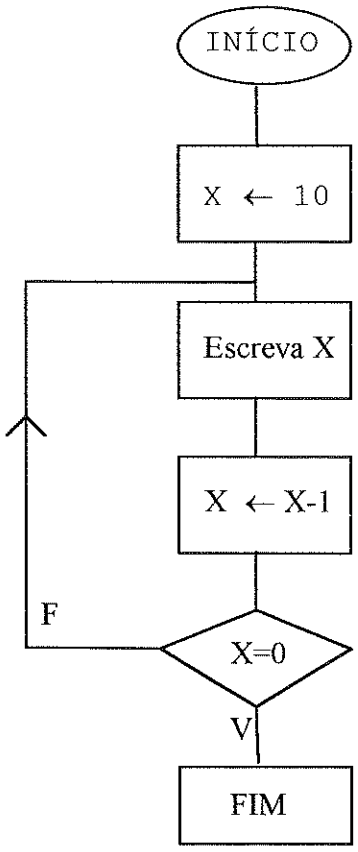
O ciclo condicional REPITA permite que um bloco de instruções seja repetido um número "**indeterminado**" de vezes em função de uma **condição que é testada após cada execução** de bloco. Se a condição for falsa é feita mais uma iteracção, caso contrário o ciclo acaba.

A sua sintaxe é:

```
Repita <instruções> até que <condição>;
```

Exemplo:

```
Início
     X ←10
     Repita
          Escreva(X);
          X ← X-1;
     Até que X=0;
fim.
```



Enquanto

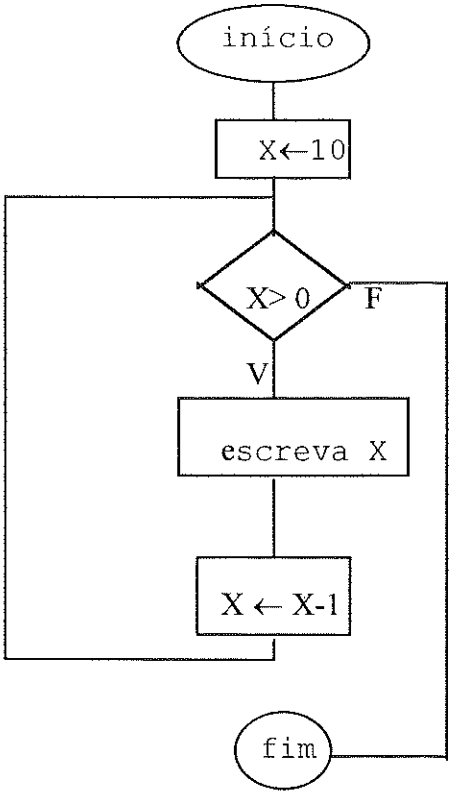
O ciclo condicional ENQUANTO permite que um bloco de instruções seja repetido um numero **"indeterminado"** de vezes em função de uma condição **que é testada à priori**. Se a condição for verdadeira é feita mais uma iteracção; se for falsa o ciclo acaba.

A sua sintaxe é:

```
Enquanto <Expressão Booleana> Faça <instruções>;
```

Exemplo:

```
Início
      X ← 10;
      enquanto X>=0 faça
                escreva(X);
                X ← X-1;
      fim enquanto;
fim.
```



Observação:

Tanto com **Repita** como com **Enquanto**, em cada iteracção, o valor das variáveis que figuram na condição varia. Caso contrário nunca mais sairíamos do ciclo.

Ciclo Para

O ciclo contado **PARA**, permite que um bloco de instruções seja repetido um numero pré-determinado de vezes *(a menos de algumas habilidades pouco recomendáveis).*

Há uma variável inteira (variável de contagem) que é incrementada ou decrementada de uma unidade desde um limite inicial até um limite final.

A sua sintaxe é:

```
Para  I=<Expressão  Inteira>  até  <Expressão  Inteira>
faça <instruções>;
```

Exemplo:

```
Início
      leia(N);
      para I=1 até 10 faça
                escreva(N," x ",I," = ",N*I)
      fim para;
fim.
```

Observações sobre a variável de ciclo I:

1) **I não** deve ser **alterada** dentro de cada ciclo;

   Por exemplo: `I ← I + S;` não é permitido.

2) Depois de sair do ciclo **I** não **está mais definido.**

3) Existe a possibilidade de realizar **"saltos negativos"** e diferentes **de unidade.**

   Por exemplo:
```
para I=3 até -3 com salto -2 faça
      Leia(A);
      Escreva(A+1);
fim para;
```

Comentários

Tudo o que está escrito no programa mas não faz parte dele, é considerado como **comentário**.

A sua utilidade é apenas para "orientar" a leitura e o entendimento do algoritmo pelos humanos.

Convencionamos que tudo o que estiver delimitado por

```
/*      ...xxxxxxx...      */
```

ou por

```
{       ...xxxxxxx...       }
```

Exemplo:

```
...
I←I+2;
/* I acaba de ser incrementado 2 unidades */
Escreva (I);
...
```

Nota: *...Como é obvio este comentário não serve para nada.*

3. A LINGUAGEM PASCAL

3.1. Introdução

A Linguagem Pascal foi desenvolvida com a finalidade de concretizar os princípios básicos da Programação Estruturada. É uma linguagem muito utilizada no ensino da programação de computadores, nomeadamente porque ajuda a criar bons hábitos de programação.

Principais características da Linguagem Pascal

* Formato livre (não há posição fixa para mudar de linha);
* Linguagem de uso genérico
* Estruturada
* Fortemente "tipada"
* Não standartiza (muitas versões)
* Linhas não numeradas

3.2. Estrutura de um programa em Pascal

Um programa em Pascal pode ser dividido nas seguintes partes:

- Um cabeçalho;
- Procedimentos e/ou funções;
- Corpo de programa ou programa principal.

Observações:

1. As 3 partes devem ser escritas por esta ordem;
2. Podem não existir Procedimentos e/ou Funções;
3. O programa principal deve ser geralmente pequeno.

Exemplo:

Ainda que não conhecçamos todos os pormenores da Sintaxe; o mais importante é a identificação dos vários módulos do programa.

```
                  Nome do programa
Program   exemplo(input, output);
VAR       i,j,k : INTEGER;          → Cabeçalho
          x,y,z : REAL;     Declarações
```

```
Function factorial(X:integer):INTEGER; → Tipo de valor Retornado
VAR  i     :INTEGER;
                  Argumentos e seus tipos
BEGIN
     i:=1
     while X>=1 Do
          Begin
               i:=i*x;   Bloco de instruções
               x:=x-1
          End
     factorial :=1
END;
```

```
(*Programa Principal*) → Para abrir e fechar em comentário
BEGIN
     writeln('Escreva um número');
     readln (i)
     writeln ('O factorial de', i,'="factorial(i))
END. → Atenção! End. (com ponto final) marca o fim do programa principal
```

O Cabeçalho

O cabeçalho é sempre constituído por um Título e por uma parte declarativa.

O Título é constituído pela palavra-chave `program` pelo nome do programa e pela indicação dos "ficheiros" que usa.

Na Parte Declarativa, seguem-se as declarações de:

Constantes `CONST`

```
PI   = 3.1415926;
ALFA = 7.248 E-12
```

Tipos `TYPE`

```
IDADE = INTEGER
NOME = ARRAY[1..60] of CHAR
```

Variáveis `VAR`

```
I, J        :INTEGEER
P1, P2      :NOME;
```

Todas as declarações feitas no cabeçalho de um programa são válidas durante toda a sua "vida", e são acessíveis dentro de qualquer procedimento ou função (veremos que nem sempre!...). As variáveis declaradas aí são, por essa razão chamadas variáveis globais do programa.

O programa principal

O programa principal é constituído pelas instruções que são executadas quando o programa corre. Contém geralmente chamadas a procedimentos e/ou funções definidas no mesmo programa.

É iniciado pela palavra-chave *Begin* e termina sempre coma palavra-chave *End.* ("End ponto final").

Um *Bloco de Instruções* pode ser entendido como um conjunto de instruções logicamente relacionadas. Para que o computador possa compreender onde começa e acaba o bloco é necessário delimita-lo pelas palavras *Begin* e *End.* Para separar as instruções dentro de um bloco usa-se o ponto e virgula (;). Na realidade o(;) age como um separador de instruções e a última instrução do bloco não necessita ser finalizada com o (;).

Indendação

A tabulação das linhas que estão compreendidas entre Begin e End é diferente para identificar o bloco. Esta prática não é essencial, mas é fortemente

recomendada como uma boa prática de programação. (Este é o tipo de programação organizada que a linguagem Pascal tem por objectivo fomentar).

Procedimentos e funções

Procedimentos e funções constituem normalmente o grosso do código executável de um programa em Pascal.

Sobre Funções e Procedimentos em Pascal devemos, neste momento, reter o seguinte:

Funções retornam sempre "qualquer coisa", que tem de ser usada por quem a chama. É o conceito matemático de função;

Procedimentos não retornam necessariamente algo, mas podem apenas executar uma tarefa. Por exemplo, escrever uma mensagem no ecran.

Qualquer um destes tipos de "funções" pode receber um número variado de argumentos de diferentes tipos.

A sua estrutura é semelhante à de um programa :

```
Procedure exemplo (X, Y : INTEGER; F1 : REAL);
<Declarações locais>
PROCEDURE ...  Funções e/ou procedimento
                locais à "função"
Begin

      Instruções executadas quando o procedure,
exemplo é chamado

End (Contando geralmente chamados à 'funções' Globais
e/ou locais à própria função ...)
```

3.3. Tipos de Dados

Uma constante pode ser representado por um nome (identificador) ao qual se associa um valor. Todos esses nomes terão de ser previamente declarados na primeira secção de um bloco de programação, (na secção das constantes).

Exemplos:

```
CONST
          neper = 2.71828;
          limite_superior = 100;
          Mensagem = 'Erro no 25';
```

São constantes pré-definidas no Turbo Pascal:

```
          Pi (=3.1415926936),
          Maxint (=32767);
```

A secção da declaração de tipos define com precissão os conjuntos de valores que as variáveis e as expressões podem assumir.

Todas as variáveis tem obrigatoriamente de pertencer a um tipo, e na construção de expressões só podem participar variáveis, constantes e valores pertencentes a tipos compatíveis.

Em síntese, os vários tipos de dados em Pascal estão esquematizados do seguinte quadro:

1. **Tipos simples**
   a) Padrão
      (i) inteiros (integer)
      (ii) reais (real)
      (iii) caracteres (char)
      (iv) Booleanas (boolean)

   b) dados definidos pelo utilizador
      (i) Enumerador
      (ii) gama de valores

2. **Dados de tipo Estruturado**
   a) Quadros ARRAYS
   b) Registos RECORDS
   c) Ficheiros
   d) Conjuntos SETS

3. **Dados de tipo apontador**

Observação: Os dados definidos pelo utilizador podem ser definidos em TYPE ou em VAR, dados do tipo ordinal de duas maneiras
- Por enumeração
- por subintervalo

Os dados do tipo ordinal, (definidos por enumeração ou por subintervalo) não podem ser lidos e nem escritos directamente do monitor ou para o ecran, a menos que se tratem de caracteres ou de inteiros.

Por enumeração, os valores são definidos através da sua listagem explícita na declaração de tipo. Cada um dos valores é representado por um nome (identificador).

Exemplos.:

```
TYPE
     Dia =(Seg, Ter, Qua, Qui, Sex, Sab, Dom)
     Mes =(Jan, Fev, Mar, Abr, Mai, Jun, Jul,
Ago, Set, Out, Nov, Dez)

VAR
     D, D1      : Dia;
     M          : Mês;
     EST        :  (Inverno,  Primavera,  Verão,
Outono);
     V          : ARRAY[ MES ] OF INTEGER;
```

Observação:

A ordem por que se indicam os valores possíveis de um tipo "por enumeração" é usada nas expressões relacionais para comparar dois valores. Por exemplo:

Ter < Sex	é verdadeiro
Dom < Seg	é falso
Qua ≥ Seg	é verdadeiro
Sab ≥ Dom	é falso

Os tipos Subintervalo são constituídos a aprtir de qualquer um dos outros tipos enumeráveis, especificando o seu valor mínimo e o seu valor máximo dentro da gama possível. O limite mínimo e máximo deve ser sepatrado por ***"ponto ponto"(..).***

Exemplos:

```
TYPE
     Pequeno     = 0..255;
     Maiusculas = 'A' ...'Z';
     Dia_Mes     = 1...31;
```

```
VAR
     DM : Dia_Mes;
     DÍGITO : '1' ... '9'
     CHUVA : ARRAY (DIA_MES) of REAL;
```

Há algumas funções pré-definidas no Turbo Pascal que trabalham com dados do tipo ordinal (inteiros, caracteres, e também definidos pelo utilizador).

ORD (X)	retorna o número de ordem do ordinal X
SUCC (X)	retorna o sucessor de X
PRED (X)	retorna o antecessor de X

Assim, nos exemplos anteriores:

```
ORD(Seg)  é igual a 0
ORD(Qua)  é igual a 3

SUCC(Sab) é Dom
PRED(sex) é Qui
```

No caso dos caracteres ORD permite dar o código ASCII:

```
ORD('A') = 65
```

A função inversa de ORD é CHR.

```
CHR(55) é o caracter: '7'
```

A secção de declaração de variáveis serve para definir o nome e o tipo de todas as variáveis utilizadas. Todas as variáveis que apareçam referenciadas num bloco, terão necessariamente de ser declaradas previamente à sua utilização.

Exemplo:

Declaração de constantes, tipos e variáveis. No que se segue apresentamos alguns exemplos de declarações de variáveis não relacionadas

```
Const
     Valor_Maximo = 100

Type
     Dias = (Segunda, Terça, Quarta, Quinta, Sexta,
Sábado, Domingo);
     Dias_de_trabalho =Segunda ... Sexta;
```

```
     Fim_de_semana = Sexta ... Domingo;
     Vencimento = 0 ... Valor_maximo;

Var
     I, J, x    :     Integer;
     Venc       :     Real
     Baixa      :     Boolean;
     Dia_1      :     Dias;
     Dia_2, Dia_3    :     Dias_de_trabalho;
     Dia4_, Dia_5    :     Fim_de_semana;
     Valor_1, Valor_2     :     Vencimento;
     .
     .
     .
```

Observações:

•Duas variáveis são compatíveis se possuirem o mesmo tipo de suporte;

Por exemplo.: `dia_1 dia_2 e dia_4` são compatíveis.

•É possível usar uma variável ou valor do tipo 'integer' onde for legal usar um valor real. Contudo o contrário já não é válido.

3.4. Tradução das Instruções de LPA em Pascal

Entrada e Saída

Para que seja possível efectuar a leitura de uma ou mais variáveis dispomos da instrução **READ** ou **READLN**.

Quer uma quer outra permitem a leitura das variáveis indicadas como argumentos, só que após a leitura READLN faz a mudança de linha no final da especificação da lista de valores.

O utilizador ao introduzir os dados deve separa-los por espaços ou ↵ .

Exemplos:

```
READ(x, y, z);
READLN(X, CH, I);
READ(A);
READLN(A);
```

A escrita do valor de uma expressão é possível com o WRITE ou WRITELN.

É possível formatar a saída indicando a largura do campo e o número de casas decimais após a vírgula, como se vê no último exemplo dos seguintes.

Exemplos:

```
Write ('Soma =', A+B);
Writeln (x, y, z);
Writeln (x: 7 :2);
Writeln;
```

Número de casas decimais

Dimensão do campo

Instrução Condicional

A execução sequencial de um programa pode ser interrompida usando as instruções condicionais.

`Se ... Então` e `Se ... então ... senão.`

A sua tradução para Pascal é:

```
IF condição THEN Instrução;
```

ou

```
IF condição THEN Instrução1 ELSE Instrução2;
```

Se Instrução, Instrução1 e Instrução2 forem não uma instrução, mas um bloco de instruções a escrita da instrução concicional tomará a forma que poderemos ver nos exemplos que se seguem.

Exemplos:

1)
```
If  D>0  THEN  R:=N/D ;
```

2)
```
If CH='S' THEN
    Begin
        Readln (x,y)
        z:= x * y + z;
        Writeln (z)
    End;
```

```
3) If (A<>B) And (I<N) Then I:= I+1
     Else
          Begin
               Writeln (A);
               Readln (B);
               A:= ln(C) - SQRT (EXP(X) * Z + W)
          End;

4) If Not Flag Then
     Begin
          Writeln (' *  * * * *');
          Writeln.
     End
     Else
     Begin
          Writeln ('# # # # # ');
          Writeln
     End;
```

Ciclo Condicional - Enquanto

A sua tradução para Pascal é:

```
While <Condição> Do <Instrução>;
```

Exemplos:

```
1) While (I<N) and (V(I) <> Procurado) Do I:= I+1;

2) While X<>0 Do
     Begin
          S:= S+X
          Readln (X)
     End;
```

O ciclo Condicional - Repita

A sua tradução para Pascal é:

```
Repeat
     <Instrução>
Until <Condição>
```

Exemplos:

```
1)    Repeat
          Write('Quer continuar? (S/N)');
          Readln(CH)
      Until(CH='S'or(CH='s')or(CH='N')or(CH='n');

2)    Repeat
          Readln(X);
          S:= S+X
      Until X=0;
```

Observação:

Observe-se que o bloco de Instruções no ciclo `Repeat-Until`, não é delimitado por *Begin-End*.

O ciclo contado Para

A sua tradução para Pascal é:

```
For I:=<Expressão Inteira> To <Expressão Inteira>
Do <Instrução>
```

Exemplos:

```
1)    For I:=1 To N Do Readln (V[ I ]);

2)    For I:=100 Down To -50 Do
          Begin
              Writeln (I, 2*I, 3*I);
              S:= S+1
          End;

3)    For I:=1 To N Do
          Begin
              For J:=1 To Do Write ('*');
              Writeln
          End;
```

A Atribuição

A atribuição denotada, em Pascal, por :=, permite atribuir a uma variável o resultado de uma expressão.

Exemplos:

```
1) Soma := A+B;
2) X := 2.04;
3) CH := CHR(NC);
4) J := x;
5) A_MAIOR_B :=(A>B);
6) Flag := True;
```

A Instrução CASE

A instrução **If ... Then**, permite a selecção de alternativas em função do resultado de uma condição.

A selecção de alternativas por valor é realizada através da Estrutura CASE, cuja sintaxe é a seguinte:.

```
CASE 'Expressão' of

     caso1 : instrução 1;
     caso2 : instrução 2;
     ...
     caso n: instrução n;
End;
```

Atenção!

O tipo de dados retornado por 'Expressão' tem de ser compatível com os casos dados de 1 a n.

É avaliada 'Expressão'; se o resultado for igual a um dos casos dados, é executado o bloco instrução correspondente.

Caso, um dos "casos" implique a execução de mais do que uma instrução (um bloco) torna-se necessário a aplicação de *Begin e End*.

Exemplo:

```
Case CH   Of
     '*' :  X:=X*Y;
     '_' :  Begin
               X:=X-Y;
               Writeln (X,Y)
               End;
     '+' :  X:=X+Y;
End;
```

Uma construção usando Case pode ser implementada à custa de um **Encadeamento de If -- Then --- Else**. O exemplo dado acima poderia ser escrito:

```
If CH='*' Then X:=X*Y;
          Else
               If CH='-' Then
                    Begin
                         X:=X-Y;
                         writeln(X,Y)
                    End
                         Else
                         If CH='+' Then X:=X+Y;
```

Diferentes valores retornados pela nossa 'Expressão' pode implicar a mesma instrução. Vejamos

Exemplo:

```
Type
     Meses = (jan, fev, mar, abr, mai, jun, jul, ago,
set, out, nov, dez);
     Estações = (Primavera, Verão, Outono, Inverno);
     Naipes   = (Copas, Ouros, Paus, Espadas);
     Cores    = (Preto, vermelho);

Var
     Mes : Meses;
     Naipe : Naipes;
     Estação : Estações;
     Cor, Cores;

     Begin
          ...
          Case Mes of
               Dez, Jan, Fev : Estação:= Inverno;
               Mar, Abr, Mai : Estação:= Primavera;
               Jun, Jul, Ago : Estação:= Verão;
               Set, Out, Nov  : Estação:= Outono
          End;
          ...
          Case Naipe of
          Paus, Espadas : Cor:= Preto
          Otherwise
               Cor:= Vermelho
          End;
```

Para a possibilidade de incluir todas, as restantes alternativas com uma só instrução, utilizamos a instrução *Otherwise*. Em Turbo Pascal *Otherwise* deve ser substituida pela palavra *"Else:"*. Os : (dois pontos) distinguem este Else de um Else relativo a um If ... Then

Exemplo:

```
Program Impostos (input, Output);

Var  Valor           :Longint;
     Categoria       :Byte;
     Taxa, imposto   :Real

Begin
     Write('Introduza o rendimento:');
     Readln(valor);
     If valor ≤ 0 Then
          Readln ('Valor Errado...')
     Else
          Begin
          Categoria:= valor DIV 10000;

          Case categoria of
               0, 1 :taxa:=0.03;        {2 valores}
               2    :taxa:= 0.04;       {1 só valor}
               3..5 :taxa:= 0.05;       {gama val.}
               Else: taxa:= 0.06;
          End;                  {End relativo ao Case}

          imposto:= taxa*valor;
          write('rendimento=', valor);
          writeln(';imposto=', imposto);
          writeln;
          Writeln('Prima Enter.');
          Readln;
          End;             {End relativo ao Else do If}
     End.                  {Fim do Programa}
```

Embutimento

Nada impede que dentro de um bloco de um ciclo ou de uma selecção de alternativas apareçam outros ciclos ou outras selecções de alternativas. Por outras palavras as instruções podem estar embutidas umas nas outras.

No exemplo que se segue vemos como embutir instruções, através de um conjunto de instruções não relacionadas entre si.

Exemplo:

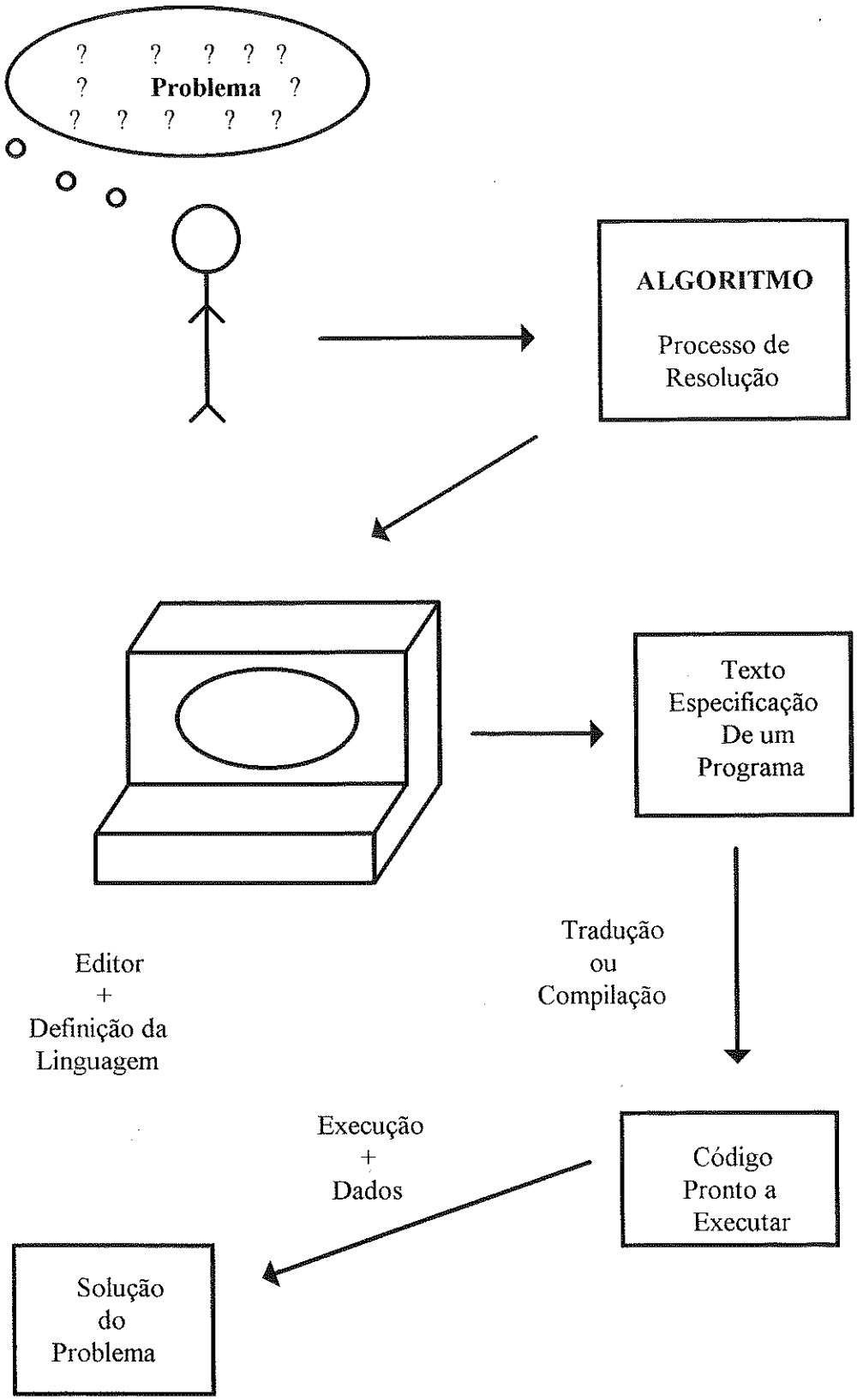
```
Repeat
     Menu;              {chamada a um procedure}
     Read(Escolha);
     Case Escolha of
     'L', 'e'  : For:=1 To N Do Readln (VCIJ);
     'O', 'o'  : Ordena (V);  {chamada a Procedure}
     'E', 'e'  : Begin
                    Writeln ('Vector ordenado');
                    For I:=1 To N Do
                              Writeln(V[I]);
                              End
                    End
Until (Escolha ='F') or (Escolha='f');
     ...
     While X<>0 Do
     Begin
          For I:=1 To X Do write ('*');
          Writeln;
          Readln(X);
     End;
     ...
```

3.5. O Turbo Pascal

O Turbo Pascal é um âmbiente de Programação no qual encontramos todas as ferramentas necessárias para desenvolver e testar todos os algoritmos. Actualmente as linguagens de caracter VISUAL, como é o caso do Visual Basic, ou do Delphi (linguagem que poderiamos designar de Visual Pascal), possuem outras ferramentas que facilitam grandemente o desenvolvimento dos programas. Mas para "vermos" o desenvolver dos nossos algoritmos interessa-nos trabalhar a um nível abaixo da linguagem visual. Caso contrário muitas acções serão transparentes. Quer dizer que automaticamente serão desenvolvidas as listagens dos algoritmos, sem termos participado na sua criação.

As ferramentas essencais para a execução dos algoritmos são:

⇒ Editor;
⇒ Compilador;
⇒ Executor;
⇒ Salvaguarda de Ficheiros.


? ? ? ? ?
? Problema ?
? ? ? ? ?
ALGORITMO
Processo de Resolução
Texto Especificação De um Programa
Editor + Definição da Linguagem
Tradução ou Compilação
Execução + Dados
Código Pronto a Executar
Solução do Problema

Utilização do Turbo Pascal

Para a utilização do **Turbo Pascal** deve Ter em conta que nos manuais se entra toda a formação necessária à sua utilização, não sendo esse assunto conteúdo programático deste curso. Contudo algumas informações úteis para poderem ser dados os primeiros passos.

* Na parte de cima do ecran, encontra-se o Menu Principal;
* Aceder ao menu principal: F10 ou ALT F, seguindo das teclas ←→;
* Na versão 6 é possível a utilização do rato;
* Dentro de cada comando, os sub-comandos são acedidos pelas teclas ↑ e ↓;
* Muitas das opções de sub-menus podem ser acedidos directamente através de teclas de função ou de conjuntos de duas teclas. ALT-X, por exemplo, permite sair directamente do TP; ou F10 permite aceder ao Menu Principal.

Para criar um Programa em Turbo Pascal

Deve utilizar o comando

Uses WINCRT;

A Biblioteca WINCRT permite aceder a todas as rotinas para entrada e saída de dados num ambiente gráfico do Windows 95.

Menu Principal do Turbo Pascal

FILE Permite ao utilizador CARREGAR, GUARDAR e CRIAR programas e ainda ...
Seleccionar directórios de trabalho e sair para o DOS definitivamente ou temporariamente.

EDIT Permite escrever ou alternar linhas de código.
O cursor posiciona-se na "parte principal" do ecran.

SEARCH Permite procurar palavras, declarações ou procedimentos no texto do programa.

RUN Executa o programa que no momento é mostrado no

ecran.

Se o programa ainda não está compilado, o comando RUN procede previamente à COMPILAÇÃO.

COMPILE O programa mostrado no ecran é compilado.
Dá mensagens de erro, e localiza o erro.

Muito Importante:
A compilação pode ser feita para a MEMÓRIA CENTRAL ou para o disco.
Este direccionamento é através do comando: DESTINATION

OPTIONS Permite alterar alguns parâmetros de configuração.

HELP Permite ler uma explicação clara e concisa dos comandos.

WINDOW Permite abrir e manipular janelas. Podemos ter várias janelas abertas simultaneamente, mas apenas uma activa de cada vez.

DEBUG O DEBUGGER permite verificar internamente os programas à medida que vão correndo (sendo executadas). É a análise passo a passo de um programa. Permite a detecção de erros lógicos.

Exercício:

Crie um programa (dos já desenvolvidos em aulas anteriores).

1. Comando EDIT: Digite o código texto de seu programa;

2. Grave o seu texto: Atribua um nome ao seu FILE (A extensão 2por default" será.PAS.

SAVE AS ... um nome é dado ao programa

SAVE ... vai gravando à medida que se escreve (enquanto não falta a luz).

3. Faça a compilação (COMPILE) ... e corrija os erros.

Observação: Destination: Memory on Disk

4. Execute o seu programa

5. Execute o comando COMPILE - DESTINATION - ENTER

6. Execute nova compilação

7. Saia para o DOS

8. Execute o seu programa fora do Pascal(directamente no Dos).

9. Observe a existência em disco dos Files:
NOME.PAS, NOME.EXE e NOME.BAK

3.6. A Função RANDOM e a função RANDOMIZE

A função **Randomize** cuja sintaxe é simplesmente

```
Randomize;
```

É um procedimento que actualiza o sistema da *"semente aleatória"*, que fixa a o valor utilizado pela função Random.

Se for necessário um conjunto repetido de valores aleatórios torna-se necessário executar Randomize antes de cada execução de Random.

Random é uma função cuja sintaxe é:

```
<Valor real>:= Random;

<Valor real>:= Random (Wordvar);
```

No primeiro caso, gera um número aleatório entre]0 e 1[. Notar bem, intervalo aberto.

No segundo caso, a função gera um número aleatório entre 0 e o valor *Wordvar* intervalo aberto.

Se **wordvar** for um número negativo ou zero, na compilação dá erro.

4. ALGORITMOS MATEMÁTICOS SOBRE ESTRUTURAS DE DADOS SIMPLES

4.1. Introdução

Neste capítulo são desenvolvidos algoritmos que resolvem problemas simples de Matemática. Desde a determinação do máximo de uma lista, até ao cálculo de uma série numérica, para um dado valor de *n*, ou à classificação de um triângulo dados os seus lados. O que estes problemas têm de comum é que todos eles usam estruturas de dados simples.

Para cada problema é apresentada a solução, acompanhada das explicações julgadas pertinentes, o algoritmo em Linguagem Algoritmica (L.P.A.), e/ou em Pascal. Nalguns casos são apresentadas várias soluções alternativas.

Problema 4.1

Desenvolver um algoritmo que leia três valores inteiros e escreva o menor deles.

Resolução:

Neste primeiro problema apresentamos o resultado da aplicação da técnica dos refinamentos sucessivos (Top Down).

```
Início
     Defina os tipos das variáveis
     Leia os números
     Determine o menor número
     Escreva o menor número
Fim
```

```
Refinamento "leia os números"
     leia (A,B,C);
Fim do refinamento


Refinamento "Determine o menor número"
     se A<B ∧ A<C
          então MENOR ← A
          senão "Determine o menor dentre B e C"
     fimse;
Fim do Refinamento


Refinamento "Determine o menor dentre B e C"
     se B<C
          então MENOR ← B
          senão MENOR ← C
     fimse;
Fim do Refinamento


Refinamento "Escreva o menor"
     Escreva (Menor);
Fim do Refinamento


Refinamento "Defina os tipos das variáveis"
     declare A, B, C, Menor: inteiro;
Fim do Refinamento
```

Algoritmo Completo em L.P.A.

```
Início
     declare A, B, C, Menor: inteiro;
     leia(A, B, C);
     se A<B ∧ A<C
          então MENOR ← A
          senão
               se B<C
                    então MENOR ← B
                    senão MENOR ← C
               fimse;
     fimse;
     escreva(MENOR);
fim.
```

Outra Versão (... um algoritmo não é único!)

```
Início
     declare A, B, C, Menor: inteiro;
     leia(A, B, C);
     se A<B
     então
          se A<C
          então  MENOR ← A
          senão  Menor ← C
          fimse
     senão
          se B<C
          então MENOR ← B
          senão MENOR ← C
          fimse
     fimse;
     escreva(Menor);
fim.
```

Solução Prob. 4.1 em Pascal

Lista das Variáveis:
x - Variável referente ao primeiro valor
y - Variável referente ao segundo valor
z - Variável referente ao terceiro valor

```
Program MenorNumero;

uses
 WinCrt;

var
 x,y,z:   real;

begin
 writeln('ESTE PROGRAMA PERMITE CALCULAR O MENOR DE 3
VALORES');
 writeln
 write('O primeiro valor é:');
 Readln(x);
 write('O segundo valor é:');
 Readln(y);
 write('O terceiro valor é:');
 Readln(z);
```

```
 ClrScr;
 If(x<y)and(x<z)then
     write('O numero menor é,',x:5:0,'.')
 else If(y<x)and(y<z)then
     write('O numero menor é:'y:5:0,'.')
 else If(z<x)and(z<y)then
     write('O numero menor é,',z:5:0,'.');
end.
```

Problema 4.2

Dada uma sequência de números inteiros positivos e terminada por um número negativo desenvolva um algoritmo que escreva **o maior** desses números.

Resolução:

Algoritmo Completo em L.P.A.

```
Início
     Leia(X);
     M ← X;
     Enquanto X>=0 faça
          Leia (X);
          Se M < X
          então M ← X
          fimse;
     fim enquanto;
     Escreva(M);
fim.
```

Resolução Prob. 4.2 em Pascal

```
Lista das variáveis:
x- Variável referente à introdução dos valores da
sequência
y - Variável atribuída ao maior valo da sequência.


Program Sequencia;

uses
     winCrt;

var
     x, y:     integer;
```

```
begin
     writeln;
     writeln;
     writeln('SE DESEJA SABER QUAL O MAIOR VALOR DE
UMA DETERMINADA SEQUÊNCIA...');
     writeln;
     writeln('Introduza uma sequência de números
terminada com um número negativo.');
     Readln(x);
     y:=x;
     Repeat
          Read(x);
          if x>y then y:=x;
     until x<0;
     Write('O maior número é: ',y,'.');
end.
```

Problema 4.3

Dada uma sequência de números inteiros maiores do que 0 (não sabendo "a priori" quantos) e terminada pelo número 0, desenvolva um algoritmo que calcule a sua soma.

Resolução:

Algoritmo Completo em L.P.A.

```
Início
     Leia(X);
     S ← 0;
     Enquanto X>Q faça
                    S ← S+X;
                    Leia (X);
     fim enquanto;
     Escreva(S);
fim.
```

Resolução Prob. 4.3 em Pascal:

Lista das variáveis :
x - Valores da sequência
s - Soma da sequência

```
Program SomaSequencia;

uses
     winCrt;
var
     x,s: real;

begin
     writeln;
     writeln;
     writeln('ESTE PROGRAMA PERMITE CALCULAR A SOMA
DE UMA SEQUÊNCIA DE VALORES.');
     Writeln;
     write('Introduza uma sequência de valores
terminada por 0. ');
     readln(x);
     s:=0;
     while x>0 do
          begin
               s:=s+x;
               read(x);
          end;
     write('A soma da sequência é: ',s:5:0,'.');
end.
```

Problema 4.4

Desenvolva um algoritmo que conte os números a partir de um dado valor. Deve contar uma quantidade de números dada à nossa escolha.

Por exemplo: Contar os primeiros 25 números acima de 80.
Contar os primeiros N números acima de M.

Resolução:

Algoritmo em L.P.A.

```
Início
     Leia (N,M);
     C ← M;
     Enquanto C < N+M  faça
                    C ← C+1;
                    escreva(C);
     fim enquanto;
fim.
```

Observação:
Para realizar a contagem de 11 a 15 é o algoritmo apresentado como está.
Mas para realizar a contagem de 10 - 15, deveriamos ter a instrução
C ← M-1
E para realizar a contagem de 10 - 14 deveriamos trocar a ordem dos comandos que estão dentro do ciclo.

... Outra versão (*de um modo mais simples*)

```
Início
      Leia (N,M);
      Para I=M até N+M faça
                Escreva (I)
      fim para;
fim.
```

Problema 4.5

Desenvolver um algoritmo que permita calcular a partir do valor de cada factura de uma empresa durante o mês os seguintes valores:

- O valor total facturado;
- O número de facturas inferiores a 5.000$00 e quanto somam;
- O número de facturas superiores a 50.000$00 e quanto somam;
- Qual a percentagem que as pequenas representam sobre o total facturado.

NOTA.: Os dados são: o número de facturas desse mês (**n**) e o valor (**X**) de cada factura.

Resolução:

Lista das variáveis :
n - Número de facturas
x - Valor de cada factura
st - Variável somatória
npf - Variável contadora das pequenas facturas
spf - Variável somadora das pequenas facturas
ngf - Variável contadora das grandes facturas
sgf - Variável somadora das grandes facturas
pp - percentagem das pequenas facturas
pg - Percentagem das grandes facturas

Algoritmo em LPA

```
Início

     Leia(N);
     st←0;
     sfp←0;
     nfp←0;
     sfg←0;
     nfg←0;
     Para I=1 até n faça
          ler(X);
          st ← st+x;
          Se x<5000
          então
               sfp←sfp+x;
               nfp←nfp+1;
          Senão
               se x>50000
               então
                    sfg←sfg+x;
                    nfg←nfg+1;
               fimse;
          fimse;
     fim para;
     pg←(sfg/st)*100;
     pp←(sfp/st)*100;
     Escreva(st, sfg, nfg, sfp, nfp, pg, pp);

fim.
```

Resolução Prob. 4.5 em Pascal

```
Program Facturas;

uses
  WinCrt;

var
  n,i,npf,ngf:          integer;
  x,st,spf,sgf,pp,pg:   real;


{--------Procedimento Iniciar Variaveis----------}

Procedure IniciarVariaveis;
begin
  st:=0;
  npf:=0;
  spf:=0;
  ngf:=0;
  sgf:=0;
end;


{---------Procedimento Pequenas Facturas--------}

Procedure PequenasFacturas;
begin
  npf:=npf+1;
  spf:=spf+x;
end;


{------Procedimento Grandes Facturas------------}

Procedure GrandesFacturas;
begin
  ngf:=ngf+1;
  sgf:=sgf+x;
end;
```

```
{------------Programa Principal-----------------}

begin
  writeln;
  writeln;
  writeln('FACTURAMENTO MENSAL');
  writeln;
  Write('Diga o número de facturas que vai
introduzir. ');
  readln(n);
  IniciarVariaveis;
  For i:=1 to n do
    begin
      write('Valor da ',i,'ª factura: ');
      readln(x);
      st:=st+x;
      if x<5000 then PequenasFacturas
                else if x>50000 then GrandesFacturas;
    end;
  pp:=(spf/st)*100;
  pg:=(sgf/st)*100;
  writeln;
  writeln('O total facturado é: ',st:5:0,'.');
  write('As facturas inferiores a 5000 escudos são
',npf,', somam ',spf:1:0);
  writeln(' e são uma percentagem de ',pp:1:0,'% do
total.');
  write('As facturas superiores a 50000 escudos são
',ngf,', somam ',sgf:1:0);
  writeln(' e são uma percentagem de ',pg:1:0,'% do
total.');
end.
```

Problema 4.6

Desenvolver um algoritmo destinado a escrever os termos da sucessão de Fibonacci inferiores a um dado valor L.

Resolução:

lista das variáveis :
L - Varíavel que limita a sequência
x, y, z - variáveis dos termos da sucessão

```
Início
     Receba o valor L
     Atribua o valor 1 ao primeiro termo
     Se ele for <1 então escreva-o
     Atribua o valor 1 ao segundo termo
     Se ele for <L então escreva-o
     Repita
          Calcule novo termo somando os dois
          anteriores
          Escreva-o
     Até que "novo termo superior ou igual a L"
Fim
```

Resolução em LPA *(utilizando o ciclo Repita)*

```
Início
     Leia(L);
     x← 1;
     y←1;
     se x<L
     então     escreva(x, y)
     fimse;
     Repita
          z←x+y
          se z<L
          então     escreva(z)
          fimse;
          x←y;
          y←z;
     até que z ≥ L;
fim.
```

Resolução em LPA *(utilizando o ciclo Enquanto)*

```
Início
     Leia(L);
     x←1;
     y←1;
     se x<L
     então     escreva(x,y)
     fimse;
     z←1;
     Enquanto z<L faça
          z←x+y
          Se z<L
          então escreva(z)
          fimse;
          x←y;
          y←z;
     fim enquanto;
fim.
```

Resolução Prob. 4.6 em Pascal

```
program Fibonacci;
uses
     winCrt;
Var
     x,y,z,l:  integer;
begin
     writeln;
     writeln;
     writeln('SUCESSÃO    DE    FIBONACCI    ATÉ    UM
DETERMINADO VALOR');
     writeln;
     write('Até que valor deseja a sequência. ');
     readln(l);
     x:=1;
     y:=1;
     z:=x+y;
     if z<l then write('A sequência é: ',x,y,z);
     while z<l do
          begin
               x:=y;
               y:=z;
               z:=x+y;
               write(z);
          end;
end.
```

Problema 4.7

Desenvolva um algoritmo destinado a determinar e imprimir o primeiro termo da **sucessão de Fibonacci**; imediatamente superior a um dado valor L.

Resolução:

NOTA.: A variável contadora *C* é introduzida para saber qual o número de ordem do termo pretendido.

Resolução em LPA *(utilizando o ciclo Repita)*

```
Início
      Leia(L);
      x←1;
      y←1;
      C←2;
      Repita
            z←x+y;
            C←C+1;
            x←y;
            y←z;
      Até que z<L;
      Escreva(z, C);
fim.
```

Resolução em LPA *(utilizando o ciclo Enquanto)*

```
Início
      Leia(L);
      x←1;
      y←1;
      C←2;
      z←0,
      Enquanto z ≤ L faça
            z←x+y;
            C←C+1;
            x← y;
            y← z;
      fim enquanto;
      escreva(z, C);
fim.
```

Resolução Prob. 4.7 em Pascal

```
Program fibonacciSuperior;

Uses
     WinCrt;

var
     L,x,y,z  :integer;

begin
     writeln;
     writeln;
     writeln('VALOR   DA   SUCESSÃO   DE   FIBONACCI
IMEDIATAMENTE SUPERIOR A UM VALOR DADO');
     writeln;
     write('Introduza um Valor.');
     readln(l);
     x:=1;
     y:=1;
     z:=0;
     while z<=l do
          begin
               z:=x+y;
               x:=y;
               y:=z;
          end;
     write('O valor da sucessão de fibonacci que se
segue a ',l,' é ',z,'.');
end.
```

Problema 4.8

Desenvolver um algoritmo que calcule o valor do capital acumulado num depósito bancário, sendo dados o capital inicial a taxa de juro anual e à quantos anos o depósito foi realizado.

Resolução:

Lista das variáveis :
c - Capital inicial
ta - Taxa de juro anual
n - Número de anos do depósito
i - Variável definida no ciclo *para*

Algoritmo em L.P.A.

```
Início
     Leia(c, tj, n);
     Para i=1 até n faça
               c←c+c*tj/100
     fim para;
     Escreva('Capital= ', c);
fim.
```

Resolução prob. 4.8 em Pascal

```
Program Capital_acumulado;

uses
     WinCrt;

type
     percentagem=1..100;

Var
     c          :real;
     ta         :percentagem;
     n,I        :integer;

{------------Procedimento Pede Valores-------------}
Procedure PedeValores;
begin
     writeln;
     writeln('CAPITAL ACUMULADO');
     writeln;
     write('Introduza o seu capital inicial ');
     readln(c);
     write('Diga qual a taxa de juro anual.(valor
entre 0 e 100) ');
     readln(ta);
     write('À quantos anos fez o seu depósito? ');
     readln(n);
end;

{---------------Programa Principal----------------}

begin
     PedeValores;
     For i:=1 to n do c:=c+c*ta/100;
  write('O catital acomulado é: ',c:5:2,'.');
end.
```

Problema 4.9

Conhecidos o capital inicial e a taxa de juro anual, desenvolver um algoritmo que determine quantos anos deve o depósito ser mantido para que o capital acumulado atinja um determinado valor M (Por exemplo.:10.000 contos).

Resolução:

Lista das variáveis :
c - Capital inicial
m - Valor que pretende atingir
ta - Taxa de juro anual
n - Número de anos do depósito

Em linguagem natural (português corrente), a resposta será:

- **Enquanto** o capital acumulado for menor que 10.000 contos deixe o dinheiro no banco e vá contando os anos;

ou

- Mantenha o dinheiro em depósito e vá contando os anos **até que** o capital acumulado seja superior ou igual a 10.000 contos.

É a aplicação das instruções do **Ciclo Enquanto** e **Repita**

Algoritmo em LPA *(Ciclo Enquanto)*

```
Início
     Leia(c,tj);
     n ←0;
     Enquanto c<10.000 faça
                    c← c+c*tj/100;
                    n←n+1;
     fim enquanto;
     Escreva(n, 'anos');
fim.
```

Algoritmo em LPA (*Ciclo Repita*)

```
Início
     Leia(c,tj);
     n←0;
     Repita
          c←c+c*tj/100;
          n←n+1;
     Até que c≥10.000;
     Escreva(n, 'anos');
fim.
```

Resolução Prob. 4.9 em Pascal

```
Program AnosDeposito;
uses
     WinCrt;
type
     Percentagem=1..100;
Var
     c, m       :real;
     ta         :percentagem;
     n          :integer;
{--------------Procedimento Pede Valores-----------}
Procedure PedeValores;
begin
     writeln('programa que permite saber quantos anos
são necessarios para atingir um determinado valor
pretendido.');
     write('Diga o seu capital inicial. ');
     readln(c);
     write('Taxa anual? (valor entre 0 e 100) ');
     readln(ta);
     write('Qual o valor que pretende atingir? ');
     readln(m);
end;
{-----------------Programa Principal--------------}
begin
     PedeValores;
     n:=0;
     while c<m do
          begin
               c:=c+c*ta/100;
               n:=n+1;
          end;
     write('São precisos ',n,' anos para que possa
     obter o valor ',m:5:0,'.');
end.
```

Problema 4.10

Desenvolver um algoritmo que calcule o factorial de um dado número n.

Resolução:

Lista das variáveis :

n - valor fornecido

f - Factorial de n

```
Início
      Leia(n);
      Se n<0 OR n<> TRUNC(n)
      então
            escreva('ERRO')
      senão
            se n=0
            então
                  escreva(n,'! = ',1)
            Senão
                  f←1
                  Para I=2 até n faça
                        f←f*I
                  fim para;
                  Escreva (n,' != ',f);
            fimse;
      fimse;
fim.
```

...Outra Versão *("à prova de burro")*

```
Início
      Repita
            leia(n);
            se n≥0 and n=TRUNC(n)
            então
                  f←1
                  Se n>0
                  então
                        Para I=1 até n faça
                              f←f*I
                        fim para;
                        Escreva(n,' != ',f);
                  fimse;
            Senão  Escreva('Numero Errado');
            fimse;
      Até que n≥0 and TRUNC(n) = n;
fim.
```

...Outra Versão
("à prova de burro" e de acordo com a P. E.)

```
Início
     Repita
          Leia(n);
          Se n<0 OR n<>TRUNC(n)
          então     escreva('Numero errado');
          fim se;
     Até que n>=0 and n = TRUNC(n)
     f←1;
     Se n>0
     então
          Para I=1 até n Faça
                    f←f*I
          fim para;
     fimse;
     escreva(n,' != ',f);
fim.
```

Resolução Prob. 4.10 em Pascal

```
Program Factorial;
uses
     winCrt;
var
     n,f,i:    integer;
{------------Procedimento Pede Valor--------------}
Procedure  PedeValor;
begin
     Writeln('programa que calcula o factorial de um
número');
     write('Introduza um valor. ');
end;
{-----------Procedimento À Prova de Burro----------}
Procedure ProvaDeBurro;
begin
     repeat
          Read(n);
          if (n<0) or (n<>trunc(n)) then
                                write('Número errado');
     until (n>=0) and (trunc(n)=n);
end;
{-----------Procedimento Saida de Resultados--------}
Procedure SaidaDeResultados;
begin
     write('O factorial de ',n,' é: ',f,'.');
end;
```

```
{--------------Programa Principal-----------------}
begin
      PedeValor;
      ProvaDeBurro;
      f:=1;
      if n<>0 then   for i:=1 to n do f:=f*i;
      SaidaDeResultados;
end.
```

Problema 4.11

Desenvolver um algoritmo que determine numa sequência de 30 números distintos, quais são aqueles que são maiores do que os seus vizinhos e diga quantos são.

Resolução:

Lista das variáveis :
x - Valor de um termo da sequência
xa - Valor do termo anterior a x
xs - Valor do termo seguinte ao de x
c - Variável contadora

Considere a sequência de valores

2 3 1 4 5 6 3 7 2 ...

A_1 A_2 A_3 A_4 A_5 ... A_{30}

Existem três situações distintas: no início, no meio e, no fim da sequência. Com efeito

$A_1 > A_2 \Rightarrow$ deve ser guaradado e contado

$A_2 > A_1 \wedge A_2 > A_3 \Rightarrow$ deve ser escrito e contado

$A_3 > A_2 \wedge A_3 > A_4 \Rightarrow$ guardar ou não?

...

$A_{30} > A_{29} \Rightarrow$ deve ser escrito.

Necessitamos de Ter em mewmória tr~es valores simultaneamente: O valor corrente, o anterior e o seguinte.

Algoritmo em LPA

```
Início

     Leia(x, y);
     S ← Q;
     Se x>y então
          escreva(x, "o",1,"°");
          s ← s+1
     fimse;
     Para i=3 até 30 Faça
          Leia (z);
          Se y>x ∧ y>z
          então
               escreva(y, "Na posição", i-1);
               s ← s+1;
          fimse
          x←y,
          y←z;
     fim para;
     Se z>y
     então
          escreva (z, "o último");
          s← s+1;
     fimse;
     Escreva (s);

Fim.
```

Resolução do Prb. 4.11 em Pascal

```
Program Vizinhos;

uses
     WinCrt;

var
     xa, x, xs, i, c     :integer;

begin

     write('Introduza o 1° valor: ');
     readln(xa);
     write('Introduza o 2° valor: ');
     readln(x);

     if xa>x then
          begin
          writeln('O valor ',xa,' é maior que ',x);
          c:=c+1;
          end;

     for i:=3 to 30 do
          begin
               write('Introduza o ',i,'° valor: ');
               read(xs);
               if (x>xa) and (x>xs) then
                    begin
                    writeln('O  valor  ',x,'  é  maior
               que ',xs);
                    c:=c+1;
                    end;
               xa:=x;
               x:=xs;
          end;

     if xs>xa then
          begin
          writeln('O valor ',xs,' é maior que ',xa);
          c:=c+1;
          end;
          writeln('Os Numeros que são maiores que os
     vizinhos são: ',c);

end.
```

Problema 4.12

É lida uma sequência de 20 números diferentes.

Desenvolva um algoritmo que detecte o número de sequências ascendentes.

Resolução:

Exemplo:

Cosideremos a seguinte sequência de números

2 4 3 8 9 7 5 6 9

Nela dectetamos 3 sequência s ascendentes. Por hipótese uma sequência obriga a mais do que um número

Com efeito, temos (2 4) (3 8 9) (7) (5 6 9). Ora (7)não é uma sequência.

Algoritmo em LPA

```
Início
     Leia(N);                    /*No exemplo será 20*/
     F←0;
     S← 0;
     Leia(X);
     Para I=2 até N faça
          Leia (XS);
          Se X<XS
          então      Se F=0
                     então F←1
                     fimse
          Senão      Se F=1
                     então
                          S←S+1;
                          F←0
                     fimse;
          fimse;
          X←XS;
     fim para;
     Se F=1
     então S←S+1
     fimse;
     Escreva(S);
Fim.
```

Resolução Prob. 4.12 em Pascal

```
Program Numero_de_Sequencias_Ascendentes;

Uses WinCrt;

var
   n, i, x, xs, c, f: integer;

begin
     write('Introduza o tamanho da sequência: ');
     readln(n);
     f:=0;
     c:=0;
     write('Introduza o 1° numero: ');
     readln(x);
     for i:=2 to n do
          begin
               write('Introduza o ',i,'° numero: ');
               readln(xs);
               if x<xs   then f:=1
                         else
                              if f=1 then
                                        begin
                                             c:=c+1;
                                             f:=0;
                                        end;
               x:=xs
          end;
     if f=1 then c:=c+1;
     writeln;
     writeln('São ',c,' sequências ascendentes!!!');
end.
```

Problema 4.13

Desenvolva um algoritmo que determine os maior e menor números de uma sequência de números positivos, assim como o número de vezes que estes aparecem repetidos na sequência.

A sequência é terminada por zero ou por um número negativo.

Resolução:

Lista das variáveis :
MA - Representa o maior número
MI - Representa o menor número
CMA - Variável contadora; conta o número de vezes que o maior número aparece repetido.
CMI - Variável contadora; conta o número de vezes que o menor número aparece repetido.

Algoritmo em LPA

```
Início;
Leia(X);
MA←X;
MI←X;
CMAX←1;
CMIN←1;
Enquanto X>0 faça
     Leia(X);
     Se X = MA
     então CMAX←CMAX+1
     fimse;
     Se X>MA
     então
          MA←X;
          CMAX←1
     fimse;
     Se X=MI
     então CMIN←CMIN+1
     fimse;
     Se X<MI and X>0
     então
          MI←X;
          CMIN←1
     fimse;
fim enquanto;
Escreva ("O MAIOR É", "MA", "APARECE", CMAX ,"VEZES")
Escreva ("O MENOR É", "MI", "APARECE", CMIN ,"VEZES")
Fim.
```

Resolução Prob. 4.13 em Pascal

```
Program Maior_Menor_Numero;

Uses WinCrt;

Var
     x, ma, mi, cma, cmi, c: integer;

Begin
     read(x);
     ma:=x;
     mi:=x;
     cma:=1;
     cmi:=1;
     While x>0 do
          begin
               readln(x);
               if x=ma then cma:=cma+1;
               if x>ma then
                    begin
                         ma:=x;
                         cma:=1;
                    end;
               if x=mi then cmi:=cmi+1;
               if (x<mi) and (x>0) then
                    begin
                         mi:=x;
                         cmi:=1;
                    end;
          end;
     writeln('O  numero  maior  é:  ',ma,'  Aparece
repetido ',cma,'x');
     writeln('O  numero  menor  é:  ',mi,'  Aparece
repetido ',cmi,'x');
end.
```

Problema 4.14

Desenvolva um algoritmo que calcule as raizes de uma equação do 2° grau, prevendo a situação de raizes imaginárias.

Resolução Matemática:

$$ax^2+bx+c=0 \Rightarrow x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Algoritmo em LPA

```
Início
Leia (A,B,C);
D← B*B - 4*A*C;
Se D<0
então
     escreva ("RAÍZES IMAGINÁRIAS")
Senão     Se D=0
          então
               X ← -B/2/A;
               Escreva ("RAIZ DUPLA=", X);
          Senão
               X1← (-B-SQRT(D))/2/A;
               X2← (-B + SQRT (D))/2/A;
               Escreva ("X1=", X1);
               Escreva ("X2=", X2);
          fimse;
fimse;
Fim.
```

Nota.: Raízes Imaginárias

$$x1=\frac{-B+\sqrt{-D}i}{2A} \quad \text{e} \quad x2=\frac{-B-\sqrt{-D}i}{2A}$$

Como fazer aparecer o resulatado desta expressões?
Executando os comandos em LPA:

```
Escreva("X1=",-B/2/A,"+",SQRT(-D)/2/A,"Xi");
Escreva("X1=",-B/2/A,"-",SQRT(-D)/2/A,"Xi");
```

Resolução Prob. 4.14 em Pascal

```
Program Raizes_Equacao;

Uses WinCrt;

Var
a, b, c, d, x, x1, x2    :real;

Begin
     write('Introduza o valor de A: ');
     readln(a);
     write('Introduza o valor de B: ');
     readln(b);
     write('Introduza o valor de C: ');
     readln(c);
     d:=b*b-4*a*c;
     if d<0 then
          begin
          writeln('Raizes Imaginárias: ');
          writeln
          (' - X1= ',-b/2/a,'+',sqrt(-d)/2/a,'i');
          writeln
          (' - X2= ',-b/2/a,'-',sqrt(-d)/2/a,'i');
          end
          else
               if d=0    then
                         begin
                         x:=-b/2/a;
                         writeln('Raiz única: ',x);
                         end
                         else
                         begin
                         x1:=(-b+sqrt(d))/2/a;
                         x2:=(-b-sqrt(d))/2/a;
                         writeln('Duas raízes: ',x1);
                         writeln('             ',x2);
                         end;
end.
```

Problema 4.15

Desenvolver um algoritmo que dado um número de 3 algarismos, calcule a soma dos seus dígitos e escreva-os por ordem inversa.

Resolução:

Torna-se necessário decompôr o número, individualizando os seus dígitos.

Exemplo:

Seja N=724. Teremos

C = ROUND(7,24) =7
D = ROUND(72,4)-7*10= 72-70= 2
U = 724-700-20= 4

```
Início
     Leia(N);
     C←TRUNC(N/100);
     D←TRUNC(N/0)-C*10;
     U←N-C*100-D*10;
     S←C+D+U;
     Escreva (S, "é a SOMA DOS DÍGÍTOS");
     Escreva (U, " ", D, " ", C);
Fim
```

Resolução Prob. 4.15 em Pascal

```
Program Ordem_Inversa;

Uses WinCrt;

var
     n, s, c, d, u  :integer;
Begin
     Write('Introduza um numero com 3 algarismos: ');
     readln(n);
     c:=trunc(n/100);
     d:=trunc(n/10)-c*10;
     u:=n-c*100-d*10;
     s:=c+d+u;
     writeln('Digitos ordem inversa:',u,' ',d,' ',c);
     writeln('A soma é: ',s);
end.
```

Problema 4.16

Dados 3 valores X, Y, Z, verificar se eles podem ser os comprimentos dos lados de um triângulo e, se forem, verificar se é um triângulo equilátero, isósceles ou escaleno.

Resolução:

Lembrar de Geometria:

A condição que três comprimentos sejam os lados de umtriângulo: ***O comprimento de um lado qualquer de um triângulo é menor do que a soma dos comprimentos dos outros dois.***

Classificação de triângulos quanto aos lados:

Equilátero: com os lados todos iguais.
Isósceles: com os 2 lados iguais.
Escaleno: com os 3 lados diferentes.

Nota: bem que o equilátero é também isósceles.

```
Refinamento 1:
Início
     Leia (A,B,C);
     Se "existe triângulo"
     então "classificar o triângulo"
     senão escrever ("NÃO É UM TRIÂNGULO")
     fimse;
Fim

Refinamento "existir triângulo"
```

{A condição "Existe triângulo" pode ser implementada como:}

```
se   A<B+C ∧ B<A+C ∧ C<A+B
então é triângulo
fimse;

Refinamento "classificar triângulo"
Se "3 lados iguais"
então     Escrever(" Equilátero")
Senão
     Se "2 lados iguais"
     então Escreva ("Isosceles")
     Senão Escreva ("Escaleno")
fimse;
```

```
Refinamento "3 lados iguais"
     A=B ∧ B=C

Refinamento "2 lados iguais"
     A=B ou B=C ou A=C
```

Algoritmo em LPA

```
Início
Leia (A,BC);

Se A<B+C ∧ B<A+C ∧ C<A+B
Então

     Se A=B ∧ B=C
     Então
          escreva("Equilátero")
     Senão

          Se A=B ou  B=C ou A=C
          Então
               escreva("Isósceles")
          Senão
               escreva("Escaleno")
          fimse;
     fimse;
Senão
     escreva ("Não é triângulo")
fimse;

Fim.
```

Resolução Prob. 4.16 em Pascal

```
Program Triangulo;

uses WinCrt;

var
     x,y,z     :real;

begin

     ClrScr;
     write('Digite o valor de X:');
     readln(x);
     write('Digite o valor de y:');
     readln(y);
     write('Digite o valor de z:');
     readln(z);

     if (x<y+z) and (y<x+z) and (z<x+y) then

          begin

               if (x=y) or (x=z) or (z=y) then

                         write('Isósceles')

               else

                    if(x=y) or (y=z) then

                              write('equilátero')
                    else write('escaleno');

          end

     else write('Nao é triangulo!!!');

end.
```

Problema 4.17

Desenvolva um algoritmo para determinar se um dado número I lido, é primo, ou não.

Resolução:

Definição: *Número primo é todo o número divisível apenas por si próprio e pela unidade.*

A estratégia geral de resolução será:

- ◊ Ler o número I
- ◊ Testar a divisibilidade do número I lido por todos os valores de 2 até I-1;
- ◊ Se I for divisível por um qualquer destes números é porque não é primo, caso contrário é primo.

```
Início
     Leia(I);
     J←2;
     Enquanto (J≤I) e (J não divide I) faça
          J←J+1
     fim enquanto;

     Se (J divide I)
     então
          escreva(I, "NÃO É PRIMO")
     Senão
          escreva(I, "É PRIMO");
     fimse;

Fim.
```

Observação:

"J divide I" implementa-se através das expressões

(I MOD J)= 0 ou I/J = TRUNC (I/J)

Para tornar o algoritmo mais eficiente observemos que

I nunca é divisível por J para valores tais $J \in]I/2, I[$

Por outras palavras,

I nunca é divisível por J para I/2 < J< I

Basta então testar os divisores de 2 a TRUNC (I/2)

Para uma versão ainda mais eficiente do algoritmo, observemos que

se **J divide I** então **existe um k : I=J*K**

Assim sendo, e dado que **J ou k não excedem** $\sqrt{I}$, basta testar os divisores de 2 até $\sqrt{I}$.

Exemplo: Se tivermos I = 47 $\Rightarrow$ $\sqrt{47} = 6,...$, basta testar os divisores 2, 3, 4, 5 e 6, para ver que as divisões são reais e, portanto concluir concluir que I é primo.

Então, finalmente, teremos

Algoritmo em LPa

```
Início
     Leia(I);
     J←2;
     K←TRUNC(SQRT(I));

     Enquanto (J≤K) e (I/J <> TRUNC(I/J)) faça
               J←J+1
     fim enquanto;

     Se J>K
     então
          escreva(I, "É PRIMO")
     Senão
          escreva (I, "NÃO É PRIMO")
     fimse;
Fim.
```

Problema 4.18

Desenvolva um algoritmo que determine todos os números primos entre 2 e um dado valor N que é lido a partir do teclado.

Resolução:

Basta desenvolver um algoritmo que repita o algoritmo do problema anterior para todos os valores de 2 a N.

```
Início
     Leia(N);
     Para I=2 até N faça
          J←2;
          K←TRUNC (SQRT(I));
          Enquanto (J≤K) and (I MOD J <> Q) faça
               J←J+1
          fim enquanto;
          Se J>K
          então
               escrever (I, "É PRIMO")
          fimse;
     fim para;

Fim
```

Resolução Prob. 4.18 em Pascal

```
Program numeros_primos;
Uses WinCrt;
Var
     c, i, x, num   :integer;
     n              :real;
begin
     ClrScr;
     Writeln('*** Programa Numeros Primos ***');
     Writeln;
     Write('Introduza um numero: ');
     read(x);
     c:=0;
     for i:=1 to x do
     begin
          n:=x/i;
          if n<>1 then
          begin
               num:=trunc(n);
               if (num*i=x) and (num<>x) then c:=1
```

```
        end;
    end;
    if c=1    then
        writeln('  O numero ',x,' não é primo!!!')
            else
        writeln('  O numero ',x,' é primo!!!');
end.
```

Problema 4.19

Desenvolva um algoritmo que permita arredondar um número qualquer X, para um dado número de casas décimais pretendido N.

Resolução:

Com efeito, a expressão

$$TRUNC\ (X * 10^n + 0.5)/10^n$$

arredonda **X** para **n** casas decimais.

Exemplos:

1) Seja $X = -5.2576$, e queremos arredondar para 2 casas décimais:

$y = TRUNC\ (-5.2576 * 10^2 + 0.5) / 10^2$
$y = TRUNC\ (-525.76 + 0.5) / 10^2$
$y = TRUNC\ (-525.26) / 10_2$
$y = -526 / 10^2$
$y = -5.26$

2) Seja $X = 23.21458$, e queremos arredondar para 3 casas décimais:

$y = 23.215$

Com efeito esta expressão funciona. Então basta implementá-la. Mas primeiro é necessário calcular as potências naturais de 10.

Algoritmo em LPA

```
Início

     Leia(X);
     PND←1
     Se N>0
     então
          Para I=1 até N faça
               PND←PND*10
          fim para;
     fimse;

     Y←TRUNC (X * PND+0.5)/PND;

     Escreva (Y);

Fim.
```

Resolução Prob. 4.19 em Pascal

```
Program Truncagem;

Uses WinCrt;

Var
     x, num     :real;
     n          :integer;

Begin
     ClrScr;
     Write('Introduza um numero: ');
     readln(x);
     write('Introduza o numero de casa décimais: ');
     readln(n);
     num:=trunc(x*10*exp(n)+0.5)/10*exp(n);
     writeln('O numero arredondado é: ',num:n);
end.
```

Problema 4.20

Desenvolver um algoritmo que dado um número positivo no sistema decimal, o represente no **sistema sexagesimal.** Por exemplo um tempo, que queremos conhecer em horas minutos e segundos.

Resolução:

Exemplo: Dado X=7422 horas, o programa deve escrever 7h 25min 19,2 seg

Algoritmo em LPA

```
Início
     Leia(X)
     H←TRUNC (X)
     y←(X-H)*60
     M←TRUNC(y)
     S←(y-M)*60
     Escreva (H, "H", M, "MIN", S, "SEG")
Fim
```

Problema 4.21

Desenvolva um algoritmo para calcular e imprimir o valor de S dado pela expressão

$$S = \frac{1}{N} + \frac{2}{N-1} + \frac{3}{N-2} + \ldots\ldots\ldots + \frac{N-1}{2} + \frac{N}{1}$$

Resolução:

Algoritmo em LPA

```
Início
     Leia(N);
     S←0;
     Num←1;
     Enquanto NUM≤N faça
          DEN ← N-NUM+1;
          S←S+NUM/DEN;
          NUM←NUM+1;
     fim enquanto;
     Escreva ('O valor de S=', S);
Fim.
```

Problema 4.22

Calcular o valor das séries para um dado valor *n:*

a) $1+\frac{1}{2}+1/3+1/4+\ldots+1/n$

b) $\frac{1}{2}-1/2^2+1/2^3-1/2^n+\ldots+(-1)^{n+1}\,1/2^n$

Resolução:

Algoritmo da a) em LPA

```
Início
      Leia(N);
      S←0;
      Para I=1 até N faça
            S←S+1/I
      fim para;
      Escreva(S);
Fim.
```

Algoritmo da b) em LPA

```
Início
      Leia(N);
      S←0;
      D←1;
      SI←-1;
      Para I=1 até N faça
            D←D*2;
            SI←(-1)*SI;
            S←S+SI*1/D;
      fim para;
      Escreva(S);
Fim.
```

Outra versão para alínea b)

```
Início
      Leia(N);
      S←0;
      D←1;
      SI←-1;
      Para I=1 até N faça
            D←D*2;
            Se I/2 = TRUNC(I/2)
            então     S←S-1/D
            Senão     S←S+1/D
            fimse;
      fim para;
      Escreva(S);
Fim.
```

Problema 4.23

Desenvolva um programa que gere um número inteiro aleatório entre 0 e 100 e nos peça que o adivinhemos.

O programa deve responder se o número que é digitado como palpite é maior ou menor que o número gerado.

O programa deve parar ao fim de *n* tentetivas (por exemplo 10), quando o jogador acertar, ou quando este desistir.

Resolução:

Para gerar um número aletóri o usamos as funções Random e randomize, conforme já explicado no capítulo anterior;

```
X Randomize←Random (100)
Segredo ← ROUND(X)
```

Numa primeira versão em LPA teremos o algoritmo da página seguinte. Na página 90 apresentamos o algoritmo implementado em Pascal. Chamamos particular atenção para a estrutura do programa em Pascal, na qual se recorre à programação desenvolvida por módulos (procedimentos) independentes, que são chamados apartir do programa principal.

Algoritmo Prob. 4.23 em LPA

```
Início
Segredo←Número gerado;
C←0
Leia(N)
FLAG←FALSE
Enquanto NOT FLAG ("não acabar") faça
      Leia(Palpite);
      C←C+1;
      Se Palpite=Segredo
      então
            FLAG←TRUE
            "Acabar"
      senão
            Se Palpite>Segredo
            então     escreva ("O MEU NUMERO É MENOR")
            Senão     escreva ("O MEU NÚMERO É MAIOR")
            fimse;
      fimse;
      Se C=N
      então
            FLAG←TRUE;
            "Acabar";
      fimse;
      Escreva ("Quer acabar? (S ou N)
      Leia (Desisto)
      Se Desisto ="N"
      então
            "Acabar"
            FLAG←TRUE
fim enquanto;

Se palpite=Segredo
então
      escreva("Parabéns!Você    acertou    após",    C,
"Tentativas")
Senão
      Se C=N
      então
            escreva("BURRO")
            escreva("O MEU NÚMERO =", SEGREDO)
      Senão
            escreva("FRACO","DESISTIU APÓS", C)
            escreva("O MEU NUMERO=",SEGREDO)
      fimse;
fimse;
Fim.
```

Resolução Prob. 4.23 em Pascal

```
program Adivinha;

uses
     WinCrt;

var
     segredo, palpite, conta  :integer
     Flag, Fim                :boolean
     tentar, NovoJogo, c      :char;

{---------- Function GeraNumero--------------------}

function GeraNumero :integer;

var
 x    :real

begin
     randomize;
     x:=random(100);
     GeraNumero:=round(x);
end;

{---------------Procedure PedeValor----------------}

procedure PedeValor;

begin
     ClrScr;
     writeln;
     writeln('PENSEI NUM NÚMERO ENTRE 0 E 100');
     writeln;
     writeln('Tem 10 tentativas para o descobrir!');
     writeln;
     conta:=conta+1;
     write('digite o seu',conta,'palpite');
     readln(palpite);
     if conta ≥10 then flag:=true;
     writeln;
     writeln;
end;
```

```
{----------------Procedure Vitoria-----------------}

procedure Vitoria;
begin
     ClrScr;
     writeln;
     writeln('P A R A B É N S ! ! !');
     writeln;
     writeln('Acertou após', conta,'tentativas');
     writeln;
     if conta ≤ 5 then writeln (' Tem M U S I C A !!!');
     writeln;
end;


{----------------Procedure Fraqueza----------------}

procedure Fraqueza;
begin
     ClrScr;
     writeln;
     writeln(' ÉS MESMO FRACO ... ');
     writeln;
     writeln(' Tentaste',' conta,'vezes e desististe
logo!');
     writeln;
     writeln('O meu número segredo era: ',segredo);
     writeln;
end;


{-------------------Procedure Burro---------------}

procedure Burro;
begin
     ClrScr;
     writeln;
     writeln(' Burro !!!!! ');
     writeln;
     writeln(' Tentaste 10 vezes e NÃO conseguiste
acertar');
     writeln;
     writeln('O meu número segredo era: ',segredo);
     writeln;
end;
```

```
{-----------------Programa Principal----------------}

begin
     Fim:=False;
     repeat
          ClrScr;
          Segredo:=GeraNumero;
          conta:=0;
          PedeValor;
          Flag:=False;
          while not Flag do
               begin
            if segredo >palpite then writeln('O meu
número é maior!')
             if segredo < palpite then writeln('O meu
número é menor!')
          if segredo <>palpite then
               begin
                    writeln;
                    writeln
                    writeln('Desiste? [digitar "d" e
"enter"]');
                    writeln;
                    write('Para continuar[digitar 2
vezes"enter"]');
                    readln(tentar);
                    if tentar<>'d' then PedeValor
Else Flag:=True;
               end
           else Flag:=True;
     end;       {fim do ciclo while}

{É feita a avaliação porquê saiu do ciclo Enquanto}

     if segredo=palpite then Vitoria else if conta ≥
10 then Burro;


{É feita a avaliação se se pretende um novo jogo}

     write('NOVO JOGO? [s ou n]');
     readln(NovoJogo);
     if Novojogo='n' then Fim:=True;
     until Fim
end.
```

5. VARIÁVEIS INDEXADAS: VECTORES E MATRIZES

5.1. Variáveis Indexadas

Problema 5.1

No dia 31 de Dezembro de cada ano, uma firma comercial pretende conhecer, a partir do valor das vendas de cada mês os seguintes valores:

a) O valor total das vendas nesse ano;
b) A média mensal das vendas;
c) A diferença mensal entre as vendas desse mês e a média mensal.

Resolução:

Para as alíneas a) e b) não há dificuldade, sendo o algoritmo que se segue suficiente para respondes às quetstões destas duas alíneas.

```
Início
S←0
Para I=1 até 12 faça
          Leia (V)
          S←S+V
fim para
MV←S/12
Escreva (S e MV)
Fim.
```

O problema da alínea c) é que a média só se calcula no fim de todas as leituras e nesse momento já não possuimos em memória os valores das vendas mensais.

A solução é "guardar em memória" os valores das vendas mensais desde Janeiro até Dezembro em memória, para que estes ainda estejam disponíveis quando quisermos calcular o desvio em cada mês.

Problema 5.2

Escrever um algoritmo que leia do teclado os nomes de todos os alunos de uma turma (cerca de 40) e que os escreva no monitor todos os nomes por ordem inversa da inicial.

Resolução:

O Algoritmo para esta listagem de nomes nomes, com os conhecimentos já adquiridos, poderia ser o seguine:

```
Início

      Variáveis  Nome_1, Nome_2, ...Nome_32

      Leia(Nome_1);
      Leia(Nome_2);
      Leia(Nome_3);
      ...
      Leia(Nome_40);
      ...
      Escreva(Nome_40);
      Escreva(Nome_39);
      Escreva(Nome_38);
      ...
      Escreva(Nome_2);
      Escreva(Nome_1);
      Escreva("Ufa"!Terminei');
fim
```

Os inconvenientes deste algoritmo são:

1°) Torna-se muito extenso se houver um grande número de nomes;
2°) É impossível estabelecer processamento com base em ciclos, visto que as diferentes variáveis "NOME" são independentes umas das outras.

Para eliminar os inconvenientes deste tipo de problemas criaram-se as, as chamadas

Variáveis Indexadas

que se classificam em **Vectores** e **Matrizes.**

Podemos referir ambos pela palavra **ARRAY** (do Inglês) que significa **Quadro** e que é a palavra utilizada pelo Pascal, para definir este tipo de dados.

5.2. Dados de Tipo Estruturado - Vectores e Matrizes

Array unidimensional (vector): É uma variável dividida em células todas do mesmo tipo e numeradas, que podem conter valores diferentes uns dos outros, os quais podem ser acessidos individualmente pelo seu índice.

Array Multidimensdional (Matriz): tal como um vector é uma indexada, só que possui mais do que um índice; se tiver dois índices será uma matriz bidimensional, se tiver três índices será tridimensional, etc.

Uma matriz bidimensional pode ser entendida como uma tabela, organizada em linhas e colunas, escrita sobre uma folha de papel. Cada quadrado da tabela é um elemento da matriz, podendo ser acedido por indicação do número da linha e do número da coluna.

Tanto **vectores como matrizes são blocos homogéneos de dados**; isto é: cada um dos elementos do vector ou da matriz são de um mesmo tipo. Daí serem chamados estruturas de dados homogéneas (uni ou multi dimensionais).

Apenas com referência, refira-se que existem outras as estruturas de dados heterogéneas, nomeadamente:

Record : Um bloco constituido por diferentes campos sendo que, em cada campo podemos ter dados de diferentes tipos.

Set : Um bloco de dados cujo conteúdo pode aumentar ou diminuir, consoante seja necessário.

Object : Um bloco heterogéneo de dados que pode assimilar componentes de outros blocos. Este tipo de dados permite a programação orientada para objectos (POO).

O objectivo da POO é encontrar características comuns entre entidades e criar "Estruturas de Dados" (Objectos) que aprendam essas características.

Definição dos tipos e Declaração das variáveis Tipo Array

No Pascal é possível o uso de variáveis indexadas onde o acesso aos vários elementos de uma variável mais genérica é feito através da indicação dos índices apropriados. Os vectores são variáveis indexadas unidimensionais (acesso feito por um índice) e as matrizes multidimensionais (normalmente bidimensionais).

A declaração das variáveis indexadas é feita na parte declarativa do programa.

Exemplos:

```
Type
     Vector     = Array [ 1 .. N] of real;
     Matriz     = Array [ 1..L, 1..C] of integer;

Var
     V1, V2     :Vector;
     VL         : Array [ 1..N of Char];
     M1, M2     : Matriz;
     MB         : Array [1..L, 1..C] of Boolean;
```

O acesso a um dado elemento da variável indexada é feito pela avaliação do(s) índice(s).

Exemplos:

1)

```
     V1[I]:= V2[J]-V2[J-1];
     writeln(M[3*I+1, 2*J-1]);
     X:=M[K,M]-y
     Read(VL[I]);
```

2)

```
     Var  A:   Array[ -4..20] of Integer;
```

É declarado A como um vector com 25 elementos em que cada elemento é um número inteiro. Por sua vez os índices são do tipo inteiro de -4 a 20.

```
     A[-4],      A[-3],      A[0],       A[20]
```

são números inteiros

3)

```
     Var C1 : Array ['A' ..'H'] of Registo
```

É declarada a variável C_1 como um vector com 8 elementos, sendo cada elemento uma entidade do tipo "Registo" definido pelo utilizador previamente em Type.

Por sua vez, os índices são do tipo Char de 'A' a 'H'

```
     C1['A'], C1['B'], ...
```

4)

```
     Var MD     :Array[0..9] of Array [0...5] of Integer
```

ou

```
     Var MD     :Array[0..9,0..5] of Integer;
```

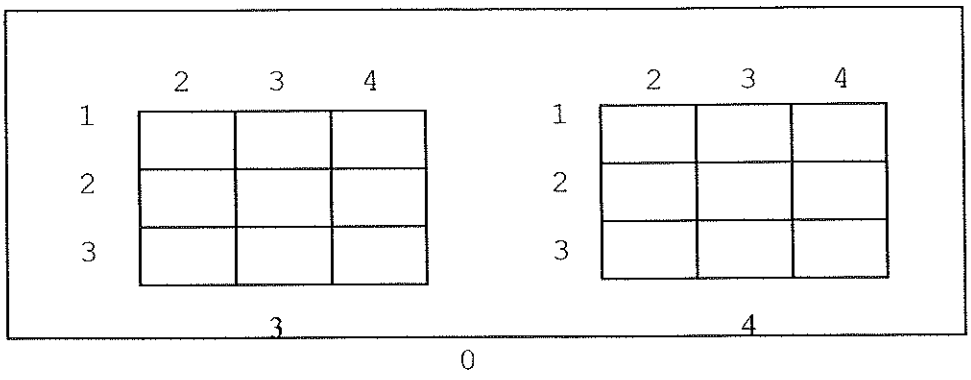
Este exemplo declara MD como uma matriz com 10 linhas e 6 colunas, sendo cada elemento

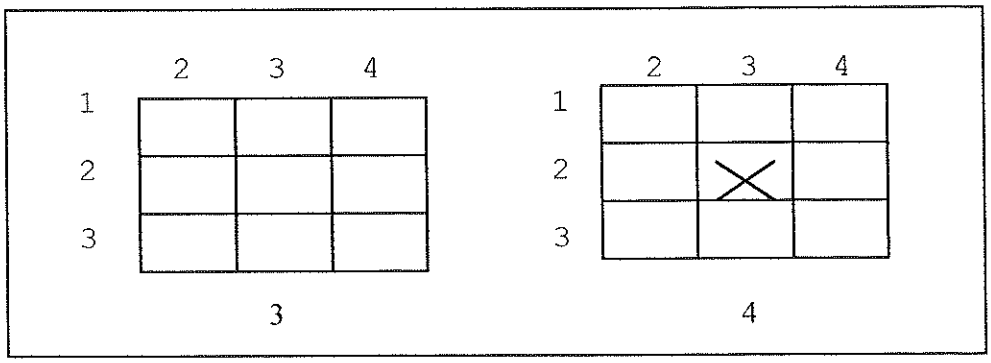
```
MD[I,J]      ou     MD [I][J]
```

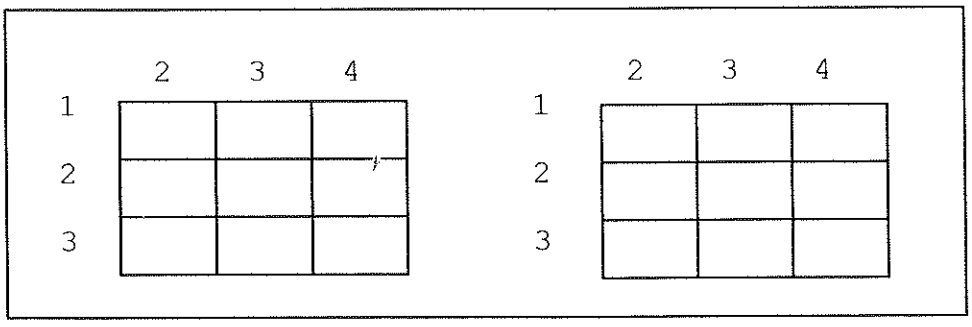
Exercício:

Considere a estrtutura de dados definida por:

```
Type M4 = Array [1..3, 2..4, 3..4, 0..2] of Char;
Var  Quartos   :M4
```







Indique com X o elemento Quarto [2,3,4,1]:= X

Introduzindo o conceito de variável indexada, apresentamos o algoritmo que implementa a solução do problema 5.1, que abre este capítulo

Algoritmo Prob. 5.1 em LPA

```
Início

Soma←0;
Para I=1 até 12 faça
     Leia (A[I]);
     Soma←Soma+ A[I];
fim para;
Média ← Soma/ 12
Para I=1 até 12 faça
     Dif←A[ I]- Média;
     Escreva (Dif)
fim para;
Fim
```

Resolução Prob. 5.1 em Pascal

```
program Desvios (input, output)

uses

     WinCrt;

Var

     i                          : integer;
     TotVend, Media, Desvio     : real;
     VenMes                     : array[1..12] of real;
     Bandeira                   : boolean;
     nim                        : char;
```

```
begin
     Bandeira:= False
     repeat
     ClrScr;

     begin {Leia Vendas Mensais e calcule o Total
Anual e a Média}

          TotVend:=0
          for i:=1 to 12
               begin
                    write('O valor das vendas de ');
                    case i of
                         1    :write('Janeiro:     ');
                         2    :write('Fevereiro:   ');
                         3    :write('Março:       ');
                         4    :write('Abril:       ');
                         5    :write('Maio:        ');
                         6    :write('Junho:       ');
                         7    :write('Julho:       ');
                         8    :write('Agosto:      ');
                         9    :write('Setembro:    ');
                         10   :write('Outubro:     ');
                         11   :write('Novembro:    ');
                         12   :write('Dezembro:    ');
                    end;
                    readln(VenMes [i]);
                    TotVend:= TotVend+VenMes[i];
               end
               Media:=TotVend/12;
               ClrScr;
               writeln; writeln;
               writeln(TOTAL  ANUAL  DAS  VENDAS:  ',
TotVend);
               writeln; writeln;
               writeln('A  Media  Mensal  das  Vendas
é:', media);
               writeln; writeln;

     end; {Fim do calculo da media e do valor Total}
```

```
     begin {Calculo do Valor do Desvio em relação à
Media}
          for i:=1 to 12 do
               begin
                    Desvio:= VenMes[i]-Media;
                    write('Mes de ');
                    case i of
                    1    : write('Janeiro:     ');
                    2    : write('Fevereiro:   ');
                    3    : write('Março:       ');
                    4    : write('Abril:       ');
                    5    : write('Maio:        ');
                    6    : write('Junho:       ');
                    7    : write('Julho:       ');
                    8    : write('Agosto:      ');
                    9    : write('Setembro:    ');
                    10   : write('Outubro:     ');
                    11   : write('Novembro:    ');
                    12   : write('Dezembro:    ')
               end;
     end;
     writeln;
     write('Calculo para um novo ano? [s/n]');
     readln(nim);
     if (nim='n') or (nim='N') then Bandeira := true;
     until Bandeira
end.
```

Problema 5.3

Escreva um algoritmo que leia o nome e a idade de cada aluno de toda a turma e faça uma listagem com os nomes dos alunos cuja idade seja superior à média.

Resolução:

Vai ser necessário definir 2 vectores:

- um para guardar em memória os nomes;
- outro para as idades:

Assim teremos, já em Pascal o seguinte algoritmo:

Resolução Prob. 5.3 em Pascal

```
program MediaIdades;

uses
     WinCrt;

Var
     Nome       : Array [1..32] of String [80];
     Idade      : Array [1..32] of Integer;
     Soma, i    : Integer;
     Média      : Real;

Begin
     Soma := 0;
     For i:=1 to 32 do
     Begin
          readln (Idade [i], Nome [i]);
          Soma:=Soma+Idade[i];
     End;
     Média:=Soma/32
     For i:=1 to 32 do
     Begin
          If Idade[i]>Média Then writeln (Nome [i])
     End;
End.
```

5.3. Processamento de Strings

Uma String pode ser vista como **um vector de caracteres**. No Pascal Standart não estavam definidas funções ou procedimentos para o tratamento de strings. Contudo, as implementações mais recentes do Pascal (Turbo Pascal) já se preocupam com o tratamento de strings, embora não se verifique a existência de normalização nesta área.

As operações mais comuns sobre strings são:

- ◊ extracção de uma sub-string;
- ◊ concatenação;
- ◊ eliminação de uma sub-string num string;
- ◊ pesquisa de uma sub-string numa string;
- ◊ conversão de uma string em número e vice-versa
- ◊ avaliação do comprimento.

Strings no Turbo Pascal

As variáveis deste tipo são usadas para guardar sequências de caracteres. Podem-se declarar strings sem nenhum tamanho especifico ou com um tamanho máximo.

```
Var

     Nome : String [35];
     Morada : String[40];
     Mensagem : String;
```

Em Turbo Pascal a variável pode guardar até 255 caracteres. Para atribuir um valor a uma string, usa-se uma sequência de caracteres entre pelicas.

Se tentarmos atribuir uma sequência com mais caracteres que os permitidos pela declaração da variável string, os que estiverem a mais são ignorados, mas não ocorre erro algum, na execução do programa.

Exemplo:

```
Programa Exemplo;

Var
     String 10: String [10];
     String 5: String [5];

Begin
     String 10 := 'Muito Bom';
     String 5 := String 10;
     writeln (String 5);
     writeln;
     readln;
End.
```

Este programa escreverá no monitor apenas: 'Muito_'.

Funções e Procedimentos sobre Strings em Turbo Pascal

Atribuição Idade:='Quinquagésimo';

Concatenação

Linha:='Muitas Felicidades no teu' + idade + 'aniversário'

Procedimentos Pré-Definidos:

Delete (St, Pos, Num)

```
St:= 'ABCDEFG';
Delete (st, 2, 4);
writeln(St);     → AFG
```

Insert (st_1, st_2, Pos)

```
Insert ('xx', st, 3);
writeln (st);     → ABXXCDEFG
```

Funções Pré-definidas

Copy (t, Pos, Num)

→ O resultado é **uma string**

```
STA:= Copy (ST, 4, 2);
writeln (STA) →DE
```

Lenght (ST)

→ O resultado é **um inteiro**

```
I:=lenght (ST) →          *I:=7*
```

Pos (obj, st)

→ O resultado é **um inteiro**

Retorna a posição de obj em ST. Se o obj não for encontrado em ST, POS retorna 0.

Problema 5.4

Desenvolver um programa em Pascal que crie uma lista de nomes completos e permita a pesquisa e selecção dos nomes que contenham um dado nome.

Por exemplo: numa lista de 20 nomes, seleccionar aqueles que se chamam António.

Resolução:

A estratégia geral para construção de um algoritmo poderá ser a seguinte:

- ◊ Lê a lista de nomes;
- ◊ Pede nome a pesquisar;
- ◊ Pesquisa nome :
  - Pesquisa no início
  - Pesquisa no meio
  - Pesquisa no fim

_ _ _ _ _ _ _ _ A N T Ó N I O
28 29 30 31 32 33 34

Resolução do Prob. 5.4 em Pascal

```
Program ProcuraNomes;

uses
     Crt;

type
     NomesCompletos      = String[35]
     Nomes               = String[15];

var

lista, conta,i      :integer;
NomeComp, NomeSel   :Array [1..20] of NomesCompletos;
Nome                :Nomes;
NovoNome, NovaLista :Char;
```

```
{----------Procedure Le Lista de Nomes-------------}

procedure LeLista;
begin
     ClrScr;
     write('Quantos nomes tem a lista?');
     readln(lista);
     For i:=1 To lista Do
          begin
               ClrScr;
               write(i,'O nome:');
               readln(nomecomp[i]);
          end;
     ClrScr;
end;

{-------------Procedure Nome a Pesquisar------------}

procedure PedeNome;

begin
     ClrScr;
     writeln;
     writeln;
     writeln;
     write('Qual o nome que quer pesquisar?');
     readln(nome);
     ClrScr;
end;
```

```
{----------------Procedure Pesquisa-----------------}

procedure Pesquisa;

var
     ncar, cnocomp, avanchar        :integer;
     início, meio, fim, St          : Nomes;
begin
     ncar:=length(nome);
     início:=nome + ' ';
     meio:=nome + ' ';
     fim:= ' ' + nome;
     conta:=0;
     For i:=1 To lista Do
          begin
               cnocomp:=length(nomecomp[i]);
               St:=copy(nomecomp[i], i, ncar+1);
               If St=início Then
                    begin
                         conta:=conta+1;
                         NomeSel[conta]:=nomecomp[i];
                    end;
               avanchar:=2;
               while avanchar<cnocomp-ncar-1 do
                    begin
          St:=copy(nomecomp[i], avanchar, ncar+2);
                    If St=meio Then
                         begin
                         conta:=conta+1;
                         NomeSel[conta]:=nomecomp[i];
                         end;
                    avanchar:=avanchar+1;
                    end;
          St:=copy(nomecomp[i], cnocomp_ncar, ncar+1);
               If St=fim Then
                    begin
                         conta:= conta+1;
                         NomeSel[conta]:=nomecomp[i];
                    end;
               end;
          end;
```

```
{------------Procedure Lista de Nomes-------------}

procedure ListaNomes;

begin
     ClrScr;
     writeln;
     writeln;
     writeln(LISTA DE NOMES);
     writeln(--------------);
     writeln;
     for i=1 To conta Do
          begin
               writeln('     ', NomeSel[i]);
               writeln;
          end;
end;


{----------------Programa Principal-----------------}

begin
     ClrScr;
     repeat;
          leLista;
          repeat
               PedeNome;
               Pesquisa;
               ListaNomes;
               writeln;
               writeln;
               write('Deseja  pesquisar  mais  algum
nome) ["s" ou "n"];
               readln(novoNome='n');
               ClrScr;
               writeln;
               write('Deseja introduzir nova lista de
nomes? ["s" ou "n"];
               readln(Nova Lista);
          until NovaLista='n';
end.
```

Problema 5.5

Escreva um procedimento ao qual é passada uma sequência de caracteres representando uma data escrita na forma DD/MM/AA e que escreve a dada na forma (dia) de (mês) de 19(ano).

Por exemplo: 4/6/61 é escrito na forma
4 de Junho de 1961

NOTA.: O dia e o mês podem ser representados por um ou dois caracteres.

Resolução Prob.5.5 em Pascal

```
program datas(input, output);

uses

     WinCrt;

Var

     data, d1        :string[8];
     dia, mes, ano   :string[2];
     extmes          :string[9];
     extdata         :string[30];
     ndia, nmes      :integer;
     datacorr, fim   :boolean;
     novadat         :char;


{----------------Procedure LeData----------------}

procedure LeData;
begin
     write('Digite a data desejada [dd/mm/aa]:');
     readln(data);
     writeln
end;
```

```
{---------------Procedure TestaData---------------}

procedure LeData;
var
     i,j,ned,nem    :integer;
begin
               {----Primeiro separar os valores----}
     i:=pos('/', data);
     dia:=copy(data,1,i-1);
     d1:=data;
     delete(d1,1,i);
     i:=pos('/', d1);
     mes:=copy(d1,1,i-1);
     ano:=copy(d1, i+1, length(d1)-i);

               {--Segundo: Testar a sua validade---}
     datacorr:=true;
     Val(dia, ndia, ned);
     val(mes, nmes, nem);
     if (ndia>31) or (nmes>12) then
               begin
                    writeln('Data inválida !!');
                    datacorr:= false
               end
          end;

{--------------Procedure ConstroiData-------------}

procedure ConstroiData;
begin
     case nmes of
          1   :  extmes := 'Janeiro';
          2   :  extmes := 'Fevereiro';
          3   :  extmes := 'Março';
          4   :  extmes := 'Abril';
          5   :  extmes := 'Maio';
          6   :  extmes := 'Junho';
          7   :  extmes := 'Julho';
          8   :  extmes := 'Agostoo';
          9   :  extmes := 'Setembro';
          10  :  extmes := 'Outubro';
          11  :  extmes := 'Novembro';
          12  :  extmes := 'Dezembro';
     end;
     extdata := dia+'de'+ extmes + 'de 19'+ ano
end;
```

```
{------------Procedure Escreve Data---------------}

procedure EscreveData;
begin
     ClrScr;
     gotoxy(20,12);
     writeln('A data digitada foi:',data);
     gotoxy(20,15);
     writeln('Por extenso será: ''', extdata, ''')
end;

{---------------Programa Principal-----------------}

begin
     fim:=false;
     repeat
          ClrScr;
          repeat
               LeData;
               TestaData
          until datacorr;
          ConstroiData;
          EscreveData;
          gotoxy(2,25);
          write('Inserir    nova    data?    [N    para
terminar]');
          read(novadata);
          if(novadata='n')   or   (novadata='N')   then
fim:=true
     until fim
end.
```

5.4. Processamento de Matrizes

Até agora analisamos alguns algoritmos respeitantes a vectores (Arrays de apenas um índice).

Um quadro de valores pode ser considerado como um Array de dois índices. Em Pascal o conceito de quadro não está confinado a uma dimensão. Quadros multidimensionais também podem ser definidos.

Já vimos que um quadro de uma dimensão pode ser pensado como uma lista (isto é, uma única linha ou coluna) de elementos.

O quadro multidimensional proporciona uma extensão natural da ideia. Podemos pensar em quadros de duas dimensões como uma tabela de elementos consistindo em linhas e colunas.

A primeira dimensão (isto é, o primeiro índice) poderá referir-se ao número da linha e a segunda dimensão (o segundo índice) ao número da coluna.

Identicamente, um quadro de 3 dimensões pode ser pensado como uma colecção de tabelas, como as páginas de um livro.

$$T = \begin{bmatrix} T[1,1] & T[1,2] & .... & T[1,n] \\ T[2,1] & T[2,2] & .... & T[2,n] \\ .... & .... & .... & .... \\ T[m,1] & T[m,2] & .... & T[m,n] \end{bmatrix}$$

T é uma tabela (matriz) de duas dimensões constitui um quadro de *m* linhas e *n* colunas de elementos pertencentes ao mesmo tipo.

Todos os elementos podem ser colectivamente referidor por um único identificador T (o nome comum da matriz).

Um elemento individual, um único elemento, pode ser referido, especificando o nome do quadro, seguido pelos valores apropriados dos índices, escritos entre parentesis rectos e separados por vírgulas.

Declaração de Matrizes

Uma matriz de 60 linhas e 150 colunas de números reais pode ser definida da seguinte maneira:

```
Var  T:   Array[1..60, 1..150] of real;
```

ou

```
Type
     índice1=1..60;
     índice2=1..150;
Var  T:   Array[índice1, índice2] of real;
```

ou ainda

```
Const     limite1=60;
          limite2=150;
Type      índice1=1..limite1;
          índice2=1..limite2;
Var  T:   Array[índice1, índice2] of real;
```

Observações:

1) Obviamente a primeira definição é a mais simples!

2) Ambos os índices são do tipo gama de valores inteiros mas os elementos do quadro são do tipo real.

3) Os índices num quadro multidimensional não necessitam ser todos do mesmo tipo. A única restrição é que cada índice seja de um tipo ordinal, simples

Organização de Matrizes Bidimensionais em Memória

A quantidade de memória ocupada é

nº de linhas * nº de colunas * nº de bytes por elemento.

A forma como os elementos da matriz se"estendem" sequencialmente pela memória, varia consoante a "linguagem" :Representação por linha ou representação por coluna.

Na representação de matrizes por linha, a sequência dos elementos em memória é a que se encontra ao percorrer uma linha da matriz no sentido da esquerda para a direita; ao último elemento da linha sucede o primeiro elemento da linha seguinte.

Por exemplo, os elementos da matriz

V	T	A	C
I	H	M	B
F	W	P	S

Na representação por linha teriamos a seguinte forma

V	T	A	C	I	H	M	B	F	W	P	S

Na representação de matrizes por coluna, a sequência é a de uma coluna no sentido de cima para baixo.

V	I	F	T	H	W	A	M	P	C	B	S

Processamento de Matrizes Bidimensionais

O processamento de matrizes bidimensionais efectua-se a maior parte das vezes por meio de dois ou mais ciclos encaixados, associando-se cada ciclo à evolução de cada índice. Assim, sendo, podemos classificar o processamento de matrizes em dois grupos:

- **Processamento por linha:** Fixando-nos numa linha avançando todas as colunas;
- **Processamento por coluna:** Fixando-nos numa coluna avançando todas as linhas.

Considerar a seguinte Matriz

I\J	1	2	3	4
1	10	5	7	19
2	15	8	2	5
3	2	17	11	1

```
Var
     T:   Array [1..3, 1..4]
```

Considerar o seguinte programa em LPA

```
Para I=1 até 4 faça Leia(T[1,I]);
```

Este programa pede-nos 4 valores que guardará na variável T e nas posições:

```
1ªlinha, 1ªcoluna
1ªlinha, 2ªcoluna
1ªlinha, 3ªcoluna
1ªlinha, 4ªcoluna
```

Se fornecermos 10, 5, 7, 9 teremos

I\J	1	2	3	4
1	10	5	7	9
2	-	-	-	-
3	-	-	-	-

Nota: o símbolo '-' elemento não definido.

É claro que para preenchermos toda a matriz, podemos de uma forma ordenada ler, para a primeira linha todas as colunas (o que já fizemos); para a segunda linha todas as colunas e finalmente para a terceira também todas as colunas.

O programa anterior não faz isto porque apresenta um índice fixo (1); pelo que, só nos permite ler a primeira linha.

Para podermos preencher as outras linhas, precisamos de **dois ciclos Para**: um para ir "gerando" as linhas e, dentro de cada linha, um outro ciclo para "gerar" as colunas.

```
Para I=1 até 3 faça
          Para J=1 até 4 faça
               Leia(T[i,j]);
          fim para;
fim para;
```

Se digitarmos sequencialmente os valores

10, 5, 7, 9, 15, 8, 2, 5, 2, 17, 11, 1,

obteremos a matriz inicialmente dada.

Exercício:

Suponha a seguinte sequ~encia de valores

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Diga qual a matriz A, após a realização de cada um dos algoritmos.

a)

```
Para I=1 até 3 faça
     Para J=1 até 4 faça
               Leia(A[i,j]);
     fim para;
fima para;
```

b)

```
Para I=1 até 4 faça
     Para J=1 até 3 faça
               Leia (A[j,i]);
     fim para;
fim para;
```

Resposta: a) $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$ b) $\begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$

Vemos então que não é arbitrária a ordem porque se escrevem os índices.

6. FUNÇÕES E PROCEDIMENTOS

6.1. Introdução

Procedimentos e Funções, em Pascal, são blocos de suporte de uma Programação Estruturada.

Permite a criação de programas com módulos independentes.

A modalidade tem duas vantagens principais:

- ◊ O trabalho de equipe;
- ◊ Um procedimento ou função pode ser chamada mais do que uma vez de diferentes locais de um programa principal, tornando o programa mais curto.

6.2. Variáveis globais, Locais e Parâmetros, Passagem por valor e por referência à variável

Nas duas secções que se seguem é mostrado o uso de sub-programas (Funções e Procedimentos). Em qualquer caso, os conceitos desta secção são fundamentais.

Uma *variável global* existe toda a execução do programa e é declarado no bloco declarativo do programa.

Uma *variável local* apenas tem existência dentro de uma função ou de um procedimento, ou ainda dentro de funções ou procedimentos chamados a a partir dos primeiros. As variáveis locais são declaradas no bloco declarativo das respectivas funções ou procedimentos.

Algumas funções e alguns procedimentos necessitam de *argumentos*.

Dentro dessas funções e desses procedimento tais parâmetros agem como se fossem variáveis ou valores normais.

Esses argumentos são designados parâmetros (formais).

No caso das funções os parâmetros recebem, de onde são chamados os valores necessários para o procedimento que devem executar.

Quanto aos procedimentos temos diferentes casos.

Quanto aos procedimentos os parâmetros podem:

1°) Não recebem nenhum valor quando o procedimento é chamado se forem parâmetros destinados a retornar valores;

2°) Recebem um valor que não será alterado agindo nesse caso como se fosse uma constante (quando chamado é copiado o valor da expressão do respectivo argumento para o parâmetro, derivando daí o nome ***chamada "por valor").***

3°) Receber algo que poderá ser modificado dentro do procedimento e cujo valor afecta a variável usada como argumento quando foi feita a chamada (derivando daí o nome *"passagem por referência" ou passagem por variável*). O que na realidade é passado é o endereço da variável usada como argumento quando é feita a chamada.

Resumindo:

Passar um parâmetro *por valor:* não é permitido que o valor do parâmetro actual seja alterado.

Passar um parâmetro *por referência*: se os *parâmetros formais* forem alterados pela função ou procedimento e essa alteração se reflectir nos parâmetros actuais correspondentes.

Para *indentificar* um parâmetro passado *por referência* usa-se a declaração **var.**

Funções

Vimos anteriormente algumas funções pré-definidas do Pascal (como por exemplo.: Cos (x). Ln(x) ...). Interessa agora ver como o utilizador pode (utilizar) definir as suas próprias funções.

Uma função retorna um valor através do seu nome.

Exemplo:

A função que se segue efectua o Somatório dos Primeiros Nel elementos de um vector previamente lido.

```
Function Somatório(S:Vector_Reais;Nel:Integer):Real;

Var
     Soma:Real;
     I:Integer;
Begin
     Soma:=0;
     For I:=1 to Nel Do  Soma:=Soma+S[I];
     Somatório:=Soma
End;
```

Admite-se que Vector_Reais é um tipo definido na parte declarativa do programa. O resultado da função é real e retorna pelo nome da função.

A chamada da função pode ser feita de diversas maneiras.

Exemplo:

```
SM:=Somatório (V, N1);
SM1:=Somatório  (V, 50);
```

ou então

```
writeln (Somatório (V1,N2+1);
```

Procedimentos

Um procedimento difere de uma função porque permite que se retome um número qualquer de valores (0, 1 ou mais) através dos argumentos do procedimento.

A passagem por referência à variável é feita antecedendo o respectivo parâmetro por Var.

As diferenças entre Procedimentos e Funções são basicamente duas:

◊ Não se especifica um tipo para o procedimento, como se faz para uma função, porque um procedimento não devolve nenhum valor;

◊ A chamada a uma função faz parte de uma instrução, enquanto que a chamada a um procedimento é em si mesma uma instrução (comando).

6.3. Aplicações Típicas à Matemática

Neste parágrafo apresenta-se um conjunto de problemas resolvidos, nos quais se clarifica como estes conceitos de Função e de Procedimento permitem a resolução de problemas de Matemática no âmbito da programação.

Problema 6.1

Desenvolva um programa que **calcule as combinações de A elementos B a B**. Utilize para tal a particularidade de implementação de funções definidas pelo utilizador.

Resolução:

Trata-se de construir um algoritmo que calcule o valor da expressão

$$C^{A}_{B} = \frac{A!}{(A-B)!*B!}$$

Como vemos por esta expressão da análise combinatória é necessário calcular o factorial de três valores pelo que vamos fazer uma função capaz de o fazer, para não termos que repetir trêz vezes as instruções que executam esse cálculo.

Resolução Prob. 6.1 em Pascal

```
Program CALCULOCOMBINATORIO(input,output);

uses
     Wincrt;

var
     a,b  :integer;
     comb :real;
```

```
{----------------Função Factorial----------------}

function factorial(x:integer):real;
var
     f    :real;
     j    :integer
begin
     if(x=0) then factorial:=1
     else
          begin
               f:=1;
               for j:=1 To x Do f:=f*j;
               factorial:=f
          end
end;

{----------------Função Combinação----------------}

function Combinação (a,b:integer):real;
var
     c,x  :real;
begin
     c:=factorial(a)/factorial(a-b)/factorial(b);
     Combinação:=c
end;

{----------------Programa Principal----------------}

begin
     ClrScr;
     writeln('Cálculo Combinatório');
     write('Digite o número de elementos do conjunto:
');
     readln(a);
     write('Digite o número de elementos de cada
agrupamento: ');
     readln(b);
     comb:=combinação(a,b);
     writeln;
     writeln('Combinações de ',a,'elementos ',b,'a',
b,'=',com:5:0);
     repeat until keypressed
end.
```

Problema 6.2

Utilizando a estrutura de controlo "Case of", escreva uma função aritmética, que receba dois números reais e o caracter de uma das quatro operações aritméticas (+,-,*,/) e retorne o resultado dessa operação entre aqueles dois números, caso esta seja possível. Caso contrário deve apresentar uma mensagem de erro.

Resolução:

Resolução do Prob. 6.2 Em Pascal

```
program Contas

uses
     WinCrt;
var
     a,b,c      :real;
     operação   :char;
     bandeira   :boolean;

{----------------procedure LeDados-----------------}
begin
     clrscr;
     write('Qual o 1° número?');
     readln(a);
     writeln;
     write('Qual o 2° número?');
     readln(b);
     writeln;
     write('Qual a operação? [+,-,*,/]');
     readln(operação);
     bandeira:=false
end;

{----------------procedure Resultados--------------}
begin
     writeln;
     if not bandeira then
     begin
          writeln('Resultado :');
          writeln;
          writeln(a:3:1,operação,b:3:1,'=',c:3:1)
     end
     else
     writeln('Detectado     Erro     na     function
Aritmética');
end;
```

```
{--------------Function Aritmética-----------------}

function Aritmética (x,y :real; op: char): real;

begin
     Case op Of
          '+' :aritmética:=x+y
          '-' :aritmética:=x-y
          '*' :aritmética:=x*y
          '/' :if y=0 then
               begin
                writeln;
                writeln('divisão por zero');
                bandeira:=true
               end
               else aritmética:=x/y;
     Else:     begin
               writeln;
               writeln('sinal da operação errado');
               bandeira:=true
               end
     end
end;

{------------------Programa Principal---------------}

begin
     LeDados;
     c:=Aritmética(a,b,operação);
     Resultado (c);
end.
```

Problema 6.3

O Banco SóVê Turista, SA, pretende desenvolver um programa para calcular o preço em escudos ou a receber por um cliente, em operações de compra e venda de divisas, respectivamente, dados o câmbio aplicável e a quantidade de divisas.

Sabe-se que sobre as vendas recai uma taxa de 6 por mil, e ainda, para facilitar os trocos, o resultado deve ser arredondado a um múltiplo de 5, sempre a favor do banco.

Resolução:

Na resolução deste problema, por um lado apresenta-se um algoritmo que resolve qualquer tipo de arredondamento (no exemplo arredondar valores a multiplos de 5), para cima ou para baixo, por outro lado procurou utilizar as funções levadas ao extremo de cada uma ser apenas uma linha de programação.

Resolução Prob. 6.3 em Pascal

```
program Cambios(input, output);

uses
     WinCrt;

const
     taxa =0.006;

var
     cambio              :real;
     escudos,divisas     :integer;
     op                  :char;


{--------------Function ArrBaixo-----------------}

function ArrBaixo(n:integer):integer;
begin
     ArrBaixo:=n-n mod 5;
end;


{--------------Function ArrCima-----------------}

function ArrCima(n:integer):integer;
begin
     ArrCima:=ArrBaixo(n+4);
end;


{--------------Function Compra-----------------}

function Compra(c: real; q:integer):integer;
begin
     Compra:=ArrBaixo(round(c*q));
end;


{--------------Function Venda-----------------}

function Venda(c: real; q:integer):integer;
begin
     Venda:=ArrCima(round(c*q*(1+taxa)));
end;
```

```
{--------------------procedureLeDados---------------}
begin
     repeat
          clrscr;
          write('Compra[C] ou Venda[V])');
          readln(op);
     until(op='c')or(op='C')or(op='v')or(op='V');
     write('Quantidade de Divisas:');
     readln(divisas);
     write('Cambio');
     readln(cambio)
end;

{--------------Procedure EscreveVector--------------}

begin
     clrscr;
     writeln('São, escudos:1,$00)
end;

{-----------------Programa Principal----------------}

begin
     LeDados;
     case op of
          'c', 'C':escudos:=Compra(cambios, divisas);
          'v', 'V':escudos:=Venda(cambio,divisas);
     end;
     EscreveValor;
     Repeat until keypressed
end.
```

No problema que se segue, evidencia-se como utilizar funções em programação de modo a conseguir-se um algoritmo mais eficiente. Também se procura utilizar soluções matemáticas menos comuns, mas que pelo facto de usarem matemática no domínio dos números inteiros naturais, simplifica grandemente o algoritmo, muito embora este seja menos evidente.

Problema 6.4

Na disciplina da L.P.I. estabeleceu-se que para dispensar de exame é preciso que a média (não arredondada) entre as notas de dois trabalhos e de um teste seja pelo menos 10. Também se estabeleceu que o teste vale 3 vezes mais que um trabalho.

Pretende-se saber qual a nota do teste que assegura a dispensa, conhecidas as notas dos dois trabalhos. O programa pedirá estas duas notas de maneira interactiva e mostrará no monitor a nota do teste.

Resolução:

Resolução Matemática

$(t1 + t2 + 3 * X)/5 \geq 10$

$t1 + t2 + 3 * X \geq 50 \Leftrightarrow 3 * X \geq 50 - (t1 + t2) \Leftrightarrow$

$\Leftrightarrow X \geq (50 - (t1 + t2))/ 3$

O valor procurado X, a nota mínima de passagem, deve ser o menor valor inteiro maior do que a expressão a que se chegou na inequação acima.

Quanto ao algoritmo, não basta

```
Function Tecto (x:real) : integer;

begin
     Tecto:= succ (Trunc(x))
end;
```

Pois estaríamos a desconsiderar a possibilidade de X ser inteiro (ou melhor: ser um número real com parte decimal nula)

Poderia ser:

```
Function Tecto(x:real) : integer;
begin
     If X=Trunc(x) Then Tecto:=Trunc(x)
                   Else Tecto := Succ(Trunc(x))
End;
```

Mas porque os números reais são representados nos computadores de maneira aproximada, sabemos que dois números reais que aritmeticamente deveriam ser iguais, podem acabar por ser representados diferentemente.

De onde, em vez de se testar a igualdade entre x e Trunc(x), é mais prudente limitarmo-nos a averiguar se dois números estão muito próximos. Eis duas funções para isso:

```
Function Infinitésimo (X:real) : Boolean;
begin
     Infinitésimo := abs(x) <0.000001
end;

Function MuitoProximos (x,y : real) : Boolean;
begin
     MuitoProximos := Infinitesimo (x-y)
end;
```

A função tecto teria então o seguinte aspecto:

```
Function Tecto(x:real) : integer;
begin if MuitoProximos (x, Trunc(x))
          Then      Tecto:=Trunc(x)
          Else      Tecto:= Succ(Trunc(x))
end;
```

Para não repetir três vezes a função `Trunc(x)`

```
Function  Tecto(x:real) : integer;
var
     T    :integer

begin
     T:= Trunc(x)
     If MuitoProximos(x,t)
     Then
          Tecto:=t
     Else
          Tecto:= Succ(t)
end;
```

Tirando partido da função ORD que retorna o número de ordem de ordinal a começar em zero, poderia ter-se

```
Function  Tecto(x:real):integer;

var
     t:integer;

begin
     t:=Trunc (x);
     Tecto:=t+ORD(not MuitoProximos [x,t])
end;
```

E, então a função NotaMin seria:

```
Function NotaMin(T1, T2 : integer):integer;
     begin
          NotaMin := Tecto(150 -(T1+T2))/3)
     end;
```

... Outra Solução ***(Um algoritmo não é único)***

No algoritmo antes desenvolvido (1ª versão), utilizaram-se valores inteiros e reais.

Como programar a função NotaMin usando apenas inteiros? Esta questão é legítima, pois os dados de entrada são inteiros (as notas dos dois trabalhos), e o resultado pretendido também é um inteiro (a nota mínima de passagem).

Ora da Matemática sabe-se que

$$m \; DIV \; n \leq m/n < m \; div \; n+1$$

Por exemplo:

12	DIV	4	=	12/4	<	12	DIV	4+1
	3		=	3	<	3+1		

Ou:

12	DIV	5	<	12/5	<	12	DIV	5+1
	2		<	2.4	<	2+1		

A função NotaMin (T1, T2 : integer) poderá então ser:

```
function NotaMin(T1, T2:integer):integer;
     Var
          d    :integer;

     begin
          d:=50 -(t1 + t2);
          NotaMin := d   div3 + ORD(d MOD 3<>0)
     end;
```

Mas pode ainda considerar-se outro algoritmo que também só manipula números inteiros.

Com efeito, sabemos que:

m DIV n + ORD (m MOD n<>0)	$\Leftrightarrow$	***(m + pred(n)) DIV n***
(i)		***(ii)***

Verifiquemos atrvés de um exemplo numérico:

a)

(i) 13 DIV 3 + (13 mod 3 <> 0) =

= 4 + 1 = 5

(ii) (13 +2) DIV 3 =

= 15 DIV 3 = 5

b)

(i) 12 DIV 4 + (12 mod 4<>0) =

= 3 + 0 = 3

(ii) (12 + 3) DIV 4 =

= 15 DIV 4 = 3

Nas páginas seguintes apresentam-se os programas em Pascal que implementam estes algoritmos.

Resolução Prob. 6.4 em Pascal (versão 1)

```
program NotadePassagem(input, output)

uses
     WinCrt;

var
     n1, n2    :integer;

Function Infinitesimo(x:real) : boolean;
begin
     Infinitesimo := abs(x)<0.000001
end;

FunctionMuitoProximos(x,y:real): boolean;
begin
     MuitoProximos := Infinitesimo(x-y)
end;

Function Tecto(x:real): integer;
var
     t:integer;

begin
     t := trunc(x);
     Tecto := t+ord(not Muitoproximos(x,t))
end;

Function Notamin(t1,t2: integer): integer;
begin
     NotaMin := Tecto((50-(t1+t2))/3)
end;

begin                          {Programa Principal}
     ClrScr;
     write('Nota do trabalho 1:');
     readln(n1);
     write('Nota do trabalho 2:');
     readln(n2);
     writeln;
     writeln;
     Writeln('Nota Minima de Passagem:
',NotaMin(n1,n2):1);
     repeat until keypressed
end.
```

Resolução Prob. 6.4 em Pascal (versão 2)

```
program NotadePassagem(input, output);

uses
     WinCrt;

var
     n1, n2    :integer;


function Div1(m,n: integer): integer;
begin
     Div1 := (m+pred(n)) div n
end;


function Notamin(t1,t2: integer) :integer;
begin
     NotaMin := Div1(50-(t1+t2),3)
end;


begin                                  { Programa Principal}

     Clrscr;
     write('Nota do trabalho 1:');
     readln(n1);
     write('Nota do trabalho2:');
     readln(n2);
     Writeln;
     writeln('Nota Minima de Passagem:
',NotaMin(n1,n2):1;
     repeat until keypressed

end.
```

7. ALGORITMOS MATEMÁTICOS SOBRE VECTORES E MATRIZES

7.1. Introdução

Neste capítulo é apresentado um conjunto de algoritmos que utilizam dados de Tipo estruturado, nomeadamente Vectores e Matrizes. São apresentados algoritmos para resolução de problemas típicos sobre vectores, como por exemplo a ordenação de um vector, e também problemas sobre operações com matrizes, tais como: adição e multiplicação de matrizes; transposição de matrizes; avaliação da simetria de uma matriz; e ,também constução de matrizes de potências a partir de dados de um vector.

Para cada problema é apresentada a solução, acompanhada das explicações julgadas pertinentes, o algoritmo em Linguagem Algoritmica (L.P.A.), e/ou em Pascal. Nalguns casos são apresentadas várias soluções alternativas.

7.2. Ordenação de um Vector

Problema 7.1

Ordenar por ordem crescente uma sequência de números. Admitamos que temos um vector A[I] com elementos que queremos ordenar por ordem crescente.

Resolução:

Há vários algoritmos que nos permitem ordenar, por ordem crescente ou por ordem decrescente, os elementos de um vector. São apresentados neste capítulo dois dos mais conhecidos: **"Replacement Sort"** e **"Bubble Sort"**. No capítulo seguinte é apresentado outro algoritmo de ordenação, mas que utiliza a recursividade, e, portanto, é muito mais eficiente.

Replacemente Sort

A estratégia estabelecida por este algoritmo é a seguinte:

Dada uma sequência de números, por exemplo:

3, 7, 1, 9, 6

deve-se comparar o primeiro elemento com todos os que estão à sua direita. Se ele for maior então trocamo-lo.

Comparando A[1] com todos os seguintes, garante-se que na posição A[1] fica o menor elemento do vector e, não mais se troca este elemento A[1].

Proceda-se do mesmo modo o elemento que está na posição 2. Isto é, vou considerar um vector com menos um elemento: o primeiro.

Vou comparar A[2] com todos os elementos que ficam à sua direita.

E assim sucessivamente...

Para o exemplo numérico acima referido, realizando a execução passo a passo deste algoritmo obter-se-ia oseguinte:

3, 7, 1, 9, 6

1° elemento = 3
2° elemento = 7
Como 3<7 nada acontece

3, 7, 1, 9, 6

1° elemento = 3
3° elemento = 1
Como 3>1 então devem ser trocados

1, 7, 3, 9, 6

1° elemento = 1
4° elemento =9
Como 1<9 nada acontece

1, 7, 3, 9, 6

1° elemento = 1
5° elemento =6
Como 1<6 nada acontece

1, 7, 3, 9, 6

... E terminam-se as comparações do 1° elemento com todos os outros garantindo que em A[1] está o menor de todos eles.

2° elemento = 7
3° elemento = 3
Como 7<3 então devem ser trocados

1, 3, 7, 9, 6

2° elemento = 3
4° elemento =9
Como 3<9 nada acontece

1, 3, 7, 9, 6

2° elemento = 3
5° elemento =6
Como 3<6 nada acontece

1, 3, 7, 9, 6

... E terminam-se as comparações do 2° elemento com todos os que estão à sua direita, garantindo que A[2] está o 2° menor valor.

3° elemento = 7
4° elemento = 9
Como 7<9 nada acontece

1, 3, 7, 9, 6

3° elemento = 7
5° elemento = 6
Como 7>6 então devem ser trocados

1, 3, 6, 9, 7

... E só falta comparar o 4° elemento com o 5°.

4° elemento = 9
5° elemento = 7
Como 9>7 então devem ser trocados, obtendo-se finalmente

1, 3, 6, 7, 9

Para se conseguir construir o algoritmo descrito em linguagem auxiliar, vamos inicialmente colocar em memória os cinco elementos, através de uma instrução de leitura.

Em linguagem auxiliar, geralmente começa-se por estruturas ideias. É o método dos refinamentos sucessivos que permitem, de acordo com a programação estruturada ir abordando um problema do geral para o particular.

Teremos então:

Estratégia geral do algoritmo Replacement Sort:

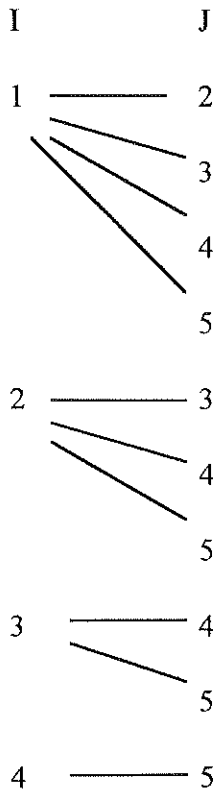
```
1° - Leitura dos valores de A[1];
2° - Comparação de cada elemento com todos os
que lhe ficam à direita e troca sempre que se
encontro um menor;
3° - Escrita dos valores de A[1] já ordenados;
```

As etapas 1ª e a 3ª não apresentam dificuldade. Podem ser respectivamente dois ciclos ***PARA (For Do)***.

Agora, para realizar as trocas de posições são necessários dois ciclos ***PARA,*** um dentro de outro. Isto é: **para cada elemento deve fazer-se a comparação** com todos os que estão à direita.

Então o primeiro ciclo deve variar de 1 até ao penúltimo (seria até 4 no exemplo atrás estudado), pois o último elemento não vai ser comparado com mais nenhum. O segundo ciclo, dentro do primeiro, deve variar "do seguinte" até ao último (5, no nosso exemplo).

Esquematicamente teremos,

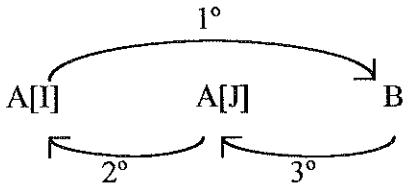


Então, no caso genérico, de um vector com N elementos:

I → Toma valores de 1 até N-1;
J → Toma valores de I+1 até N.

```
Para I=1 até N-1 faça
     Para J=I+1 até N faça
          Efectua a troca se necessário
     fim para;
fim para;
```

Como realizar a troca?



O bloco de instruções para a troca de elementos será:

```
Se A[I]>A[J]
então      {Troca}
     B←A[I];
     A[I]←A[J];
     A[J]←B;
fimse;
```

Finalmente, o algoritmo completo e generalizado a N elementos é apresentado na página seguinte.

Algoritmo Replacement Sort em L.P.A.

```
Início
{------------ Leitura dos Valores -----------}

     Leia(N);
     Para I=1 até N faça
          Leia(A[I])
     fim para;

{----------------- Ordenação -------------}

     Para I=1 até N-1 faça
          Para J=I+1 até N faça
               Se A[I]>A[J]
               então
                    B←A[I]
                    A[I]←A[J]
                    A[J]←B
               fim se;
          fim para;
     fim para;

{-------------- Escrita do vector ordenado --------}

     Para I=1 até N faça
                    Escreva(A[I])
     fim para;
Fim.
```

Bubble Sort

Outro algoritmo para a ordenação de um vector é designado por Exchange Sort ou Buble Sort.

A estratégia deste algoritmo é:

Comparar sempre elementos contíguos e, realizar a troca de elementos sempre que estes não estão por ordem.

Neste algoritmo o maior elemento é que será colocado no local certo (*último lugar*) ao fim da 1ª execução do ciclo.***PARA.***

Esta execução é repetida e, pára quando numa das vezes não fizer nenhuma troca. Por outras palavaras:

repetirá o ciclo PARA, enquanto houver trocas.

Algoritmo Bubble Sort em L.P.A.

```
Início

{------------ Leitura dos Valores -----------}

          Leia(N);
          Para I=1 até N faça
               Leia(A[I]);
          fim para;

{------------------ Ordenação -------------}

          N_corrida←0;
          Flag←True;
          Enquanto Flag=True faça
               Flag←False;
               Para I=1 até N-1-conta faça
                    Se A[I]>A[I+1]
                    então
                         B←A[I]
                         A[I]←A[I+1]
                         A[I+1]←B
                         Flag←True
                    fim se;
               fim para;
               N_corrida←N_corrida + 1;
          fim enquanto;

{--------------- Escrita do vector ordenado --------}

          Para I=1 até N faça
               Escreva(A[I]);
          fim para;

Fim.
```

Procedimento Bubble Sort em Pascal

```
procedure BubbleSort (p:integer; var v:lista);

var
     Ncorrida  :integer;
     Flag      :boolean;

begin
     Ncorrida := 0;
     Flag := True;
     while flag do
          begin
               Flag := False;
               for i := 1 to p-1-Ncorrida do
                    if v[i]>v[i+1] then
                         begin
                              troca(v[i],v[i+1]);
                              Flag := True
                         end;
               Ncorrida := Ncorrida+1
          end;
     end.
```

7.3. A Pesquisa de um Elemento num Vector

Problema 7.2

O problema que nos propomos resolver agora é o de pesquisa de um elemento x num dado vector A [i]. Com as strings poderia ser a pesquisa de um nome numa lista de nomes.

Resolução:

O algoritmo será constituido por três blocos distintos:

```
Refinamento 1:
     1 nível   1° Bloco - Leitura
     2° Bloco - Pesquisa
     3° Bloco - Escrever resultado
fim refinamento 1.
```

A abordagem mais simples poderá ser:

```
Início

     Para I=1 até N faça
               Leia A[i];
     fim para;

     Leia (X);       /* X: Elemento a pesquisar*/;

     Flag←False;
     Para I=1 até N faça
               Se X=A[i]
               então Flag←True;
               fimse;
     fim para;

     Se Flag
     então
          Escreva ('O valor', X, 'existe na lista');
     Senão
          Escreva ('O valor', X, 'ausente da lista');
     fimse;
Fim.
```

A variável Flag é colocada a False no início e só passará a True se o elemento for encontrado. Funciona como uma bandeira.

Contudo, pode-se criticar o algoritmo apresentado dizendo que ele é pouco eficiente, pois após ter encontrado o elemento X (se é que ele existe em A[i]), o algoritmo continua a fazer comparações que já não vão servir para nada. Assim, o ciclo de repetição só deve ser executado Enquanto a Flag está em False e ainda há elementos para comparar. Por outras palavras: o ciclo **PARA** deve ser substituido por um ciclo **ENQUANTO.**

Em LPA e generalizando para N elementos do vector teríamos:

Resolução Prob. 7.2 em L.P.A.

```
Início
     Leia(N);
     Para i=1 até N faça
          Leia(A[i]);
     fim para;

     Leia (X);       {X: valor a pesquisar}
```

```
    Flag←True;
    Enquanto Not Flag and i≤N faça
              Se X=A[i]
              então
                   Flag←True;
              fimse
              I←I+1;
    fim enquanto;

    Se Flag
    então
         escreva(X, "está na lista");
    Senão
         escreva(X, "não está na lista");
    fimse;

Fim.
```

Este algoritmo é muito mais eficiente do que o anterior.

Pesquisa Binária

O problema da pesquisa de um elemento num vector que **já** está **ordenado** torna-se muito mais eficiente se a pesquisa for realizada de acordo com o algoritmo denominado **"Pesquisa Banária"**, cuja estratégia se apresenta a seguir.

Dado um vector ordenado por ordem crescente:

$$A[i] = 1\_\_\_\_\_ N;$$

e um número X, queremos saber se X figura no vector.

A estratégia a seguir é a seguinte:

Comparar x com o elemento que figura no meio do vector.
Se X for menor que esse elemento, procuramos nos elementos que figuram do meio para a esquerda, uma vez que os elementos estão por ordem crescente.
Se X for maior procuramos nos elementos do meio para a direita.

E assim sucessivamente...

Exemplo:

Seja A[i] =

7	*10*	*12*	*20*	*30*	*45*	*70*	*80*	*90*	*105*	*200*
	1°El									**11°El**

Seja o elemento a pesquisar, por exemplo, X=90.

1°) Cálculo do elemento mediano?

$$\frac{11+1}{2} = 6$$

Então A[6]=45

2°) Como, 90>45, a pesquisa continua mas somente entre os 5 elementos que se seguem:

...	***70***	***80***	***90***	***105***	***200***
Posição →	7°	8°	9°	10°	11°

Como calcular agora o meio desta metade do vector?

$$\frac{7+11}{2} = 9$$

Como A[9]=90 e X=90 então "*X existe na lista*"

Procurando agora desenvolver o Algoritmo.

1°) Seja o vector ordenado:

A [i] = _____
1 N

2°) O elemento mediano será sempre dado pela expressão

Trunc((N+1)/2)

Exemplos: Considere os seguintes vectores de elementos, e calcule os seus elementos medianos.

1	*2*	*3*	*4*	*5*		=>	Trunc((5+1)/2) = 3°
1	*2*	*3*	*4*	*5*	*6*	=>	Trunc((6+1)/2) = 3°

Então teremos:

```
     Se X>Trunc((N+1)/2)
     então
          "Procurar elemento X para a direita"
     senão
          "Procurar elemento X para a esquerda"
     fimse;
```

3º) A questão a resolver agora é a seguinte: **como indicar em cada instante o intervalo** no qual quero fazer comparações?

A **resposta** é: teremos de criar duas variáveis (L e R por exemplo) que indiquem em cada situação a ordem do limite inferior e a ordem do limite superior desse intervalo

Seja então o seguinte algoritmo:

```
Início
Flag←False;
L←1
R←N
Enquanto ("não chegar ao fim") faça
          PMédio←Trunc ((L+R)/2)
          Se X=A[PMédia]
          então
               Flag←True;
          senão
               Se X>A[PMédia]
               então L←PMédia;
               senão R←PMédia;
               fimse;
          fimse;
fim enquanto;

Se NotFlag
então
     Se X=A[1] or X=A(N)
     então
          Flag←True;
     fimse;
fimse;
Se NotFlag
então
     Escreva(X, "não existe");
Senão
     Escreva (X,"existe sim senhor");
fimse;
fim.
```

Observação:

A instrução

```
Se NotFlag
então
     Se X=A[1] or X=A(N)
     então
     Flag←True;
     fimse;
fimse;
```

torna-se necessária porque este algoritmo não compara o nosso valor X, nem com o primeiro elemento nem com o último do vector.

Verifiquemos que assim é através de um exemplo analisado passo a passo:

Seja o nosso vector ordenado

A[i] = 1 3 4 5 6 e seja X=2

L =1
R=5 | ⇒ PM = Trunc ((1+5)/2)=3 ⇒ A[3] =4

Como X=2 <4 ⇒R=3

L =1
R =3 | ⇒ PM = Trunc ((1+3)/2) = 2 ⇒ A[2] =3

Como X=2 <3 ⇒ R=2

L=1
R=2 | ⇒ PM = Trunc ((1+2)/2) =1 ⇒ A (1) =1 Como X=2>1 ⇒ L=1

Temos então de novo

L=1
R=2 | (L=R-1)

Reparemos que obtivemos os mesmos valores de L e R e então a condição de saída do ciclo enquanto foi encontrada antes de se comparar X com o primeiro valor.

O mesmo se passaria no extremo superior do vector: não comparava com o último.

Observação:

A condição ***"Não chegar ao fim"*** que termina o ciclo enquanto, traduz-se por

Not Flag and L<> R-1

Por outras palavras: o ciclo enquanto termina quando

F= True ou L= R-1

No sentido de tornar o algoritmo mais eficiente devemos procurar outra condição saída do ciclo enquanto

Por exemplo, impor que L≤ R e não que L<>k-1. Podemos então permitir que o ciclo compare X com A [1].

Mas como fazer com que L seja maior do que R ?
Vejamos que é, fazendo

R←PM -1 ***e*** ***L←PM + 1***

Apresenta-se, na página seguinte, o algoritmo completo da Pesquisa Binária em linguagem auxiliar de programação.

Algorimo da Pesquisa Binária em L.P.A.

```
Início
Flag←False;
L←1;
R←N;
Enquanto NotFlag and L≤R faça
          PM←Trunc((L + R)/2);
          Se X=A [PM]
          então
               I←PM
               Flag←True;
          senão
               Se X>A[PM]
               então
                    L←PM + 1
               Senão
                    R←PM - 1
               fimse;
          fimse;
     fim enquanto;

Se Flag
então
     Escreva(X', "Existe", "na posição",I);
senão
     Escreva("Não encontrei", X, "na lista");
fimse;
Fim.
```

Exemplo:
Analisemos passa a passo com o exemplo que se segue:

A[i] = 1 3 5 7 9; e seja X=2

L=1

R=5 | ⇒ PM= Trunc ((1+5)/2)=3

A[3]=5 Como X=2 <5 ⇒ R=PM - 1 = 3 -1 =2

L=1 | ⇒ PM = Trunc ((1+2)/2) =1
R=2 |

A[1]=1 Como X=2 >1 ⇒ L =PM+1 =1+1 =2

L=2 | ⇒ PM = Trunc ((1+1)/2)=2
R=2 |

A[2]=3 Como X=2<3 ⇒ R=Pm - 1 = 2 -1= 1

E temos neste momento L =2 | ⇒ Fim do ciclo enquanto.
R =1 |

Como ocorreu o fim do ciclo enquanto sem que a variável Flag passasse a True, então o valor de Xnão figura no vector.

Seja agora X=9

L=1 | ⇒ PM =3 ∴ A[3]=5
R=5 |

Como X=9>5 ⇒ L=3+1+=4

L=4 | ⇒ PM =4 ∴ A[4]=7
R=5 |

Como X=9>7 ⇒ L=4+1+=5

L=5 | ⇒ PM =5 ∴ A[5]=9
R=5 |

Como X=9=9 ⇒ Flag←True ⇒ x existe na sequência

7.4. "Merging": Actualização de um Vector a Partir de Outro

Problema 7.3.

Suponhamos que temos um vector A com N elementos e um vector B com M elementos, sendo que os dois vectores estão ordenados pela mesma ordem.

Sejam os vectores

```
A[i]  =
i     =   1°,....., N°

B[j]=
j     =   1°,....., M°
```

Pretende-se construir um vector C[k] constituído pela reunião de A e B, sendo o vector C[k] também ordenado segundo a mesma ordem.

```
C[k]  =
k     =   1°,.....,(M+N)°
```

Observação: Se não houver elementos comuns, o vector C terá M+N elementos; mas caso haja elementos iguais, isso já não acontecerá. Esta situação deve estar prevista.

Exemplo:

Dados os vectores A e B

A[i]= 3 7 9 15 30 40 70 80 500

B[j]= 1 4 5 10 20 24 27

Pretende-se construir o vector:

C[k]= 1 3 4 5 7 9 10 15 20

24 27 30 40 70 80 500

Merging

Técnica de actualização do vector A[i] por intermédio do vector B[j].

Resolução Prob. 7.3:

A primeira abordagem ao algoritmo poderá ser:

```
Início
     Leitura dos vectores A e B;
     Comparação e selecção entre A e B até um
deles acabar;
     Copiar o que falta do outro;
     Escrita do vector C[k];
fim.
```

Observação:

O ciclo **Enquanto** termina:

para I=N+1 e o I fica igual a J;
ou
para J=M+1 e neste caso o I fica igual a J

Então, para terminar de copiar o resto do vector que ainda não chegou ao fim, será necessário um terceiro ciclo ***PARA***

Então teremos | z=J até M
ou
z=I até N

Finalmente, o algoritmo Merge em LPA poderá ser Ter a forma do algoritmo apresentado na página seguinte.

Merging em L.P.A.

```
Início
     Leia(N,M);
     Para i=1 até N faça          { Lê primeiro vector}
          Leia(A[i]);
     fim para;
     Para j=1 até M faça          { Lê segundo vector}
          Leia(B[j]);
     fim para;
     i←1;
     j←1;
     k←1;
     Enquanto I≤N and J≥M faça    {Há Dois vectores}
          Se A[i]<B[j]
          então
               C[k]←A[i]
               i←i+1
               k←k+1
          senão
               C[k]←B[j]
               j←j+1
               k←k+1
               Se A[i]=B[j]
               então
                    i←i+1
               fimse;
          fimse;
     fim enquanto;

     Se I>N
     então     /*Copia o resto de B[j]*/
          Para z=i até N faça
               C[k]←B(z)
               k←k+1
          fim para;
     Senão     /* Copia o resto de A[i]*/
          Para z=i até N faça
               C(k)←A(z)
               k←k+1
          fim para;
     Para i=1 até k-1 faça
          escreva (C[i]);
     fim para;
Fim.
```

7.5. Rotação de Vectores

Problema 7.4

Escreva um procedimento em Pascal para rodar para a esquerda um vector com *n* elementos.

Resolução:

Resolução Prob. 7.4 em Pascal

```
program rodavector;

uses
     WinCrt;

type
     lista=array[1..30] of integer;

var
     vector    :lista;
     n,i       :integer;

{---------------Procedure LeVector-----------------}

procedure LeVector;

begin
     clrscr;
     write('Quantos elementos tem o vector ?
[1..30]');
     readln(n);
     writeln;
     for i:=1 to n do
      begin
          clrscr;
          write('Digite o ',i,' elemento:');
          readln(vector[i]);
     end;
     clrscr;
end;
```

```
{--------------Procedure RodaEsquerda-------------}

procedure RodaEsquerda(n:integer; var v :lista);
var
     t    :integer;

begin
     t:= v[1];
     for i:=1 to n-1 do v[1] := v[i+1];
     v[n]:=t
end;

{--------------Procedure EscreveVector-------------}

procedure EscreveVector;
begin
     clrscr;
     writeln('O VECTOR RODADO');
     writeln;
     writeln;
     for i:=1 to n do write (vector[i],' ')
end;

{-----------------Programa Principal--------------}

begin
     LeVector;
     RodaEsquerda(n,vector);
     EscreveVector;
     repeat until keypressed
end.
```

7.6. Aplicações com Matrizes

Problema 7.5

Dada uma matriz de *ml* linhas e *nc* colunas, desenvolva um algoritmo que nos calcule uma matriz de *m+1 linhas e n+1 colunas*, em que a última linha é formada pelas somas de cada coluna e a última coluna é formada pelas somas dos elementos de cada linha.

Exemplo:

Dada a matriz

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

pretende-se um programa que construa e escreva no monitor a matriz

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 15 \\ 7 & 8 & 9 & 24 \\ 12 & 15 & 18 & 45 \end{bmatrix}$$

Resolução:

O programa terá 4 etapas distintas a que correspondam 4 procedimentos:

- ◊ Lê a matriz;
- ◊ Soma cada linha criando a última coluna;
- ◊ Soma cada coluna criando a última linha;
- ◊ Escreve a tabela final.

Algoritmo do problema 7.5 em LPA

```
Início
Leia(NL);
Leia(NC);
                {Ciclo de leitura da Matriz}
Para l=1 até Nl faça
          Para C=1 até NC faça
               Leia (T[l,c]);
          fim para;
fim para;
          {Cálculo das somas de cada linha}
Para l=1 até NL faça
          T[l, NC+1]←0;
          Para C=1 até NC faça
                    T[l, NC+1]←T[l, NC+1]+T[l,C]
          fim para;
fim para;
```

```
          {Cálculo das somas de cada coluna}
Para C=1 até NC+1 faça
          T[NL+1,C]←0
          Para l=1 até Nl faça
                    T[NL+1,C]← T[NL+1,C] + T[l,C];
          fim para;
fim para;

          {Escrita da Tabela Resultado}
Para l=1 até NL+1 faça
          Para C=1 até NC+1 faça
               Escreva (T[l,C]);
          fim para;
fim para;
Fim.
```

Problema 7.6

Desenvolva um programa que dadas duas matrizes do mesmo tipo, calcule e escreva a matriz soma das duas matrizes.

Resolução:

```
1°)Ler A;
2°)Ler B
3°)Calcula C[i,j]= a[i,j]+b[i,j]
4°)Escreva C
```

Resolução Prob. 7.6 em Pascal

```
Programa SomaMatrizes(input, output),
Type
     tabela    :Array[1..20,1..30] of integer;
Var
     a,b,c     :tabela;
     l, nl     :1..20
     c, nc     :1..30;

Procedure LeMatriz (var t: tabela);
     {Lê os elementos de uma matriz}
Begin
     For l:=1 To nl Do
          Begin
          Writeln('Introduza os dados da linha
número', l:2);
          For C:=1 To nc Do read(t[l,c]);
          writeln
          End
End;
```

```
Procedure SomaMat (t1,t2:tabela; Var t3: tabela);
     {Soma duas matrizes}

Begin
     For l:=1 To nl Do
          For C:=1 To nc Do
               t3[l,c]:=t1[l,c]+t2[l,c]
End;

Procedure EscreveMatriz (t:tabela)
     {Escreve os elementos de uma Matriz}

Begin
     writeln('Matriz Soma:')
     writeln;
     For l:=1 to nl Do
     Begin
          For C:=1 To nl Do write(t[l,c]:4)
     End
End;

Begin     {Programa Principal}

     ClrScr
     write('Quantas linhas? (1..20)');
     readln(nl);
     writeln;
     write('Quantas colunas? (1..30)');
     readnl(nc);
     writeln;
     writeln('Primeira Matriz:');
     writeln;
     Lê Matriz(a);
     writeln('Segunda Matriz:');
     Lê Matriz(b);
     SomaMatriz(a,b,c);
     Escreve Matriz(c); repeat until Keypressed;

End.
```

Observação:

Este programa tem a aparticularidade de conter Procedimentos com passagem de parâmetros por referência e por valor.

Problema 7.7

Recorrendo à *Programação Modular*, desenvovla em Pascal, um programa que permita:

Ler duas matrizes quadradas de ordem n, não superior a 10;

Calcular e escrever a matriz resultante da soma da primeira matriz com a matriz transporta da segunda;

A partir da matriz anterior, construir e escrever uma outra matriz tal que os elementos da diagonal principal estejam ordenados por ordem crescente, e os elementos **a[i,j],** acima da diagonal principal sejam trocados pelos simétricos dos elementos **a[i,j]** abaixo da diagonal principal.

Resolução:

Resolução Prob. 7.7 em Pascal

```
program SomaMatrizes(input, output);

uses
     WinCrt;

type
     tabela = array[1..10,1..10] of real;
var
     a,b,c,d        :tabela;
     i,nl,i,nc :1..10;

{-----------------Procedure LeMatriz----------------}

procedure LeMatriz(var t:tabela);

begin
     {lê os elementos de uma matriz}

for i:=1 to nl do
 begin
     writeln('Introduza os dados da linha numero',
1:2);
     for j:=1 to nc do
      begin
          write('elementos[',i,',',j,']');
          read (t[i,j])
      end;
     writeln
   end
end;
```

```
{---------------Procedure SomaMatrizes--------------}

procedure SomaMat(t1,t2 :tabela; vart3 :tabela);

var
     s    :real;

begin

     for i:=1 to nl do {Troca de valores}
          forj:=1 to nc do
               if i≥j then t2[i,j] :=[i,j]
                    else t2[i,j] := -t1[j,i];

     for i:= 1 to nl-1do {Ordenação da Diagonal}
          for j:=i+1 to nl do
               if t2[i,i] > t2[j,j] then
                    begin
                    s:=t2[i,i];
                    t2[i,i]:= t2[j,j];
                    t2[j,j]:=s
               end;
end;

{--------------Procedure EscrevaMat----------------}

procedure EscreveMat(t:tabela);

begin
     {Escreve os elementos de uma matriz}

writeln('Matriz Soma:');
writeln;
for i:=1 to nl do
     begin
          for j:=1 to nc do write(t[i,j]:10:2);
     writeln;
     writeln;
     end
end;
```

```
{----------------Programa Principal----------------}

begin
     clrscr;
     write('Qual a ordem das Matrizes? (1..10)');
     readln(nl);
     nc:=nl;
     writeln('Primeira Matriz:');
     writeln;
     LeMatriz(a);
     writeln('Segunda Matriz:');
     writeln;
     LeMatriz(b);
     SomaMat(a,b,c);
     ClrScr;
     TransformaMat(c,d);
     EscreveMat(c);
     writeln;
     writeln;
     EscreveMat(d);
     repeat until keypressed
end.
```

Problema 7.8

Na disciplina de Linguagens e Programação I, a avaliação dos conhecimentos é feita tendo por base a realização de dois testes e dois trabalhos. Todas estas provas são classificadas numa escala de 0 a 20 valores. A nota final é a média aritmética arredondada a uma casa decimal das três melhores classificações obtidas.

Desenvolva um programa que permita:

Ler do terminal o nome de cada aluno desta turma, bem como as suas 4 notas parciais;

Calcule a nota final de cada aluno;

Apresente uma lista dos nomes e respectivas notas finais, ordenada por ordem decrescente de classificações.

Resolução:

Este problema foi apresentado em prova de exame final escrito na disciplina de Linguagens e programação I do curso de Informática do Instituto superior politécnico Portucalense.

Resolução do Prob. 7.8 em Pascal

```
program Avaliação(input,output);

uses
     WinCrt;
var
     nome              :array [1..30] of string [30];
     notas                  :array [1..30,1..4] of real;
     notaf                  :array [1..30] of real;
     i,j,n                  : integer;
     notamin,soma,temp      :real;
     nometemp               :string [30];

{Aquisição dos dados (Nomes e Notas Parciais)}

begin
     ClrScr;
     write('Quantos alunos tem a turma? [1..30]');
     readln(n);
     for i:=1 to n do
          begin
               clrscr;
               write('Nome do aluno numero ',i,' ');
               readln(nome[i]);
               for j:=1 to 4 do
                    begin
                         write(j,' a nota');
                         readln(notas[i,j])
               end
end;

{Cálculo das 3 melhores notas e da sua média}
for i:=1 to n do
     begin
          notamin:=0;
          soma:=0;
          for j:=1 to 4 do
               begin
                    if notamin =0 then
                              notamin :=notas[i,j];
                    if notas[i,j]<notamin then
                              notamin :=notas [i,j]
               soma :=soma-notamin;
               nota[i] :=soma/3
      end;
```

```
{Ordenação dos nomes e das notas finais}

for :=1 to n-1 do
     begin
          for j:=i+1 to n do
               begin
                    if nota[i]<notaf[j]then
                         begin
                              temp :=notaf[i];
                              notaf[i] :=notaf[j];
                              notaf[j] :=temp;
                              nometemp :=nome[i];
                              nome[i] :=nome[j];
                              nome[j] :=nometemp
                         end
               end
end;

{Escrita de resultados}

clrscr;
for i:=1 to n do
     begin
          write(nome[i],' ');
          writeln(notaf[i]:5:1)
     end;
repeat until keypressed
end.
```

Problema 7.9

Recorrendo à Programação Modular, desenvolva em Pascal, um programa que permita:

Ler duas matrizes quadradas de ordem n, não superior a 10;
Verificar se elas são permutáveis isto é, (A.B=B.A);
O resultado deve ser do tipo booleano;
Em caso afirmativo, a matriz (A.B) deve ser escrita.

Resolução:

Resolução do Prob. 7.9 em Pascal

```
program SimetriaDe Matrizes(input, output);

uses
     WinCrt;

type
     tabela=array [1..10,1..10] of integer;

var

     a,b,c,d          :tabela;
     i,j,z,n          :integer;
     somatório :integer;

{------------------Procedure LeMatriz---------------}

procedure LeMatriz(var t:tabela);

begin {lê os elementos da matriz}
     for i:=1 to n do
      begin
          writeln('Introduza os dados da linha
numero',i:2);
          for j:=1 to n do
           begin
               write('elemento [',i,',',j,']'):
               read(t[i,j])
           end;
          writeln
      end
end;

{---------------Procedure MultiplicaMat------------}

procedure MultiplicaMat(t1,t2:tabela; var t3:tabela);
begin     { Realiza o produto de duas matrizes }
     for i:=1 to n do
          for J:=1 to n do
          begin
               somatório :=0;
               for z:=1 to n do;
               somatório:=somatório+t1[i,z]*t2[z,j];
               t3[i,j] :=somatório
          end
end;
```

```
{-----------------Procedure EscrevaMat-------------}

procedure EscreveMat(t:tabela);

begin     { Escreve os elementos de uma matriz}
 writeln;
 writeln('matriz Produto A.B');
 writeln;
 for i:=1 to n do
     begin
          for j:=1 to n do write(t[i,j] :10);
          writeln;
          writeln;
     end
end;


{-----------------Function Simetria---------------}

function Simetria(t1,t2 : tabela):boolean;

var
 flag     :boolean;

begin

 flag :=true;
 i:=1;
 while (flag) and (i≤n) do
     begin
          j:=1;
          while(flag) and (j≤n) do
               begin
               if t1[i,j]<>t2[i,j] then flag:=false;
               j:=j+1
               end;
          i:=i+1
     end;
Simetria:=flag;

end;
```

```
{------------------Programa Principal--------------}

begin

     clrscr;
     write('Qual a ordem das matrizes parcela?');
     readln(n);
     writeln;
     writeln('Primeira Matriz:');
     writeln;
     LeMatriz(a);
     writeln('Segunda Matriz:');
     writeln;
     LeMatriz(b);
     MultiplicaMat(a,b,c);
     MultipilcaMat(b,a,d);
     if Simetria (c,d)
          then
               begin
                    clrscr;
                    writeln('As matrizes são
                    Simétricas, ou seja, A.B = B.A');
                    EscreveMat(c);
               end
          else
               writeln('As matrizes Não são
               simétricas, ou seja, A.B <>B.A');
     repeat until keypressed
end.
```

Problema 7.10

Recorrendo à Programação Modular, desenvolva em Pascal, um programa que permita:

- ◊ Ler dois vectores de inteiros com m e n elementos respectivamente;
- ◊ Ordenar por ordem decrescente o primeiro vector;
- ◊ Construir uma matriz com m linhas e n colunas, cujos elementos de cada linha sejam as n primeiras potências naturais dos elementos do primeiro vector;
- ◊ Escreva essa matriz.

NOTA.: A potência natural de um número deve ser uma função definida pelo utilizador.

```
{----------Procedure Troca(var a1,a2:integer)-------}

var
     temp :integer;

begin
     temp:=a1;
     a1:=a2;
     a2:=temp
end;

{----Procedure Ordena(x:integer; var v:vector)----}

var
     p,k  :integer;

begin
     for p:=1 to x-1 do
          for k:=p+1 to x do
               if v[p]<v[k] then Troca(v[p],v[k])
end;

{--Function Potência(base, expo:integer):integer)---}

var
     j, produto     :integer;

begin
     if base=0 then Potência:=0
          else if expo=0 then Potência:01
               else
                    begin
                         produto :=1;
                         for j:=1 to expo do
                              produto :=produto*base;
                         Potência:=produto
                    end
end;

{----------------Procedure ConstroiMatriz-----------}

var
     i,j  :integer;

begin
     for i:=1 to m do
          for j:=1 to n do mat[i,j]:=Potência(v1[i],j)
end;
```

```
{---Procedure EscreveMat(nl,nc:integer; t:matriz)---}

var
     i,j       :integer;

begin

     clrscr;
     writeln('Matriz de Potências:');
     writeln;
     for i:=1 to nl do
          begin
               for j:=1 to nc do write(t[i,j]:10);
               writeln;
               writeln;
          end

end;

{----------------Programa Principal-----------------}

begin

     conta:=1;
     NumeroDeElementos(m);
     LeVector(m,v1);
     NumeroDeElementos(n);
     LeVector(n,v2);
     Ordena(m,v1);
     ConstroiMatriz;
     EscreveMat(m,n,mat);
     repeat until keypressed;

end.
```

8. A RECURSIVIDADE, O PASCAL E A MATEMÁTICA

8.1. Introdução

Uma das características mais importantes da linguagem Pascal é a capacidade de um procedimento ou função se chamar a si próprio. Isto é conhecido por Recursividade.

O uso da recursividade é particularmente conveniente para aqueles problemas de Matemática que podem ser definidos em termos naturalmente recursivos (embora tais problemas possam também ser programados usando técnicas não-recursivas). Exemplo típico de tais situações é o cálculo de um factorial.

Então, um sub-programa (procedimento ou função) é dito recursivo quando se invoca a si próprio; directa ou indirectamente.

Ao escrever um procedimento ou função recursivas é essencial que o procedimento (ou função) incluam uma condição de finalização. Isto evita que a recursividade continue indefinidamente. Contudo, enquanto a condição de finalização se mantiver insatisfeita, o procedimento (ou função) simplesmente se chama a si mesmo no local apropriado tal como faria qualquer outro procedimento ou função.

Problema 8.1

Já vimos um programa para cálculo do factorial de uma dada quantidade usando uma função não-recursiva para executar os cálculos. Vejamos agora como este cálculo pode ser efectuado usando a recursividade

Resolução Matemática:

Notar que o factorial de um inteiro positivo n, pode ser definido recursivamente.

$$n! = n.(n-1).(n-2).(n-3) \ldots 4.3.2.1 \Leftrightarrow$$
$$n! = n.(n-1)! \quad \text{sabendo que} \quad 1!=1$$

Estas equações são a base da seguinte função recursiva.

```
Function factorial(n:integer):integer;
     (*Calcula o factorial de n *)

Begin
     If N≤1     Then factorial:1
                Else factorial:=n*factorial(n-1)
end;
```

Note que esta função é muito mais simples do que a resolução anterior.

A correspondência íntima entre esta função e a definição matemática de factorial, em termos recursivos, deve ser imediatamente evidente: Note, também, que a instrução If - Then -Else produz uma condição de finalização que é activada quando o parâmetro n se torna menor ou igual a 1 (n nunca se torna menor do que 1 a não ser que o valor original seja menor do que 1)

O Programa completo de factorial recursivamente calculado

```
Program factorial(input, output);

Var X: integer

Function Factorial (n:integer) : integer;

Begin
     If n≤1
     Then
          factorial := 1
     Else
          factorial :=n*factorial (n-1)
End;

{-----------------Programa Principal----------------}

Begin
     write('Introduza um inteiro positivo:');
     readln(x);
     writeln;
     writeln('x=',X,'X! =, factorial(X));
End.
```

Para calcular o factorial de um único valor n, a função factorial é agora acedida repetidamente; uma vez no bloco principal, e (n-1) vezes por si próprio.

Problema 8.2

Dada uma função definida não recursivamente, pretende-se que ela seja definida de um modo recursivo.

Seja a função

```
Function SSSSS(N: Integer):Integer;

Var
     I,A : Integer;

Begin
     A:=1
     For I:=2 To N Do A:=A+I;
     SSSSS:=A;
End;
```

Resolução Prob. 8.2 em Pascal

```
Function SomaRec(n:integer) : integer;

Begin
     If n<1     Then SomaRec:=0
                Else SomaRec:= n + SomaRec (n-1)
End;
```

Resolução Matemática:

1+2+3+4+5 = (5)+(4+3+2+1) =

= (5+4) + (3+2+1) = (5+4+3) + (2+1) =

= (5+4+3+2) + 1 = (5+4+3+2+1)

= 15

Programa em Pascal da Soma Recursiva

```
program CalculaSomaRec;

uses
     Crt;
Var
     x    :integer;

{--------------------Função Soma--------------------}

Function SomaNumero (n: integer):integer;

var
     soma, i   :integer;

begin
     soma:=0;
     For i:=1 To n Do soma:=soma+i;
     SomaNumeros:=soma
end;

{---------------Função Soma Recursiva---------------}

function SomaRec(n:integer):integer;

begin
     If n<1    Then SomaRec :=0
               Else SomaRec:=n+SomaRec(n-1)
end;

{-------------- Programa Principal------------------}
begin
     ClrScr;
     repeat
          writeln('Este programa soma os primeiros n
números naturais.');
          writeln;
          write('Digite um valor para n:');
          read(x);
          ClrScr;
writeln('Cálculo normal da soma:,SomaNumeros(x));
          writeln;
writeln('Cálculo da Soma Recursiva: ',SomaRec(x));
          repeat until Keypresses;
          ClrScr
     until x=0
End.
```

Problema 8.3

Desenvolver um programa que aplique a recursividade ao cálculo da *x* elevado a *n*.

$$x^n,$$

sendo x um número real e n um inteiro raltivo.

Resolução:

Do ponto de vista matemático, podemos definir recursivamente o cálculo de potências, através da expressão

$$x^n = x \,.\, x^{n-1}$$

Como *n* é inteiro, positivo ou negativo, há quatro situações possíveis:

- ◊ Base=0;
- ◊ Expoente=0;
- ◊ Expoente Negativo: x^{n+1}/x
- ◊ Base e expoente positivos: $x \,.\, x^{n-1}$

Resolução Prob. 8.3 em Pascal

```
program CalculaPotencia;

uses
     WinCrt;

var
     x     :real;
     n     :integer;
```

```
{-----------Função Potência Recursiva-------------}

function PotenciaRec(p:real; q:integer) :real;

begin
     If p=0.0
     Then
          PotenciaRec:=0.0
     Else
          if q=0
          Then
               PotenciaRec:= 1.0
          Else
               if q<0
               Then
                    PotenciRec:=PotenciaRec(p,q+1)/p
               Else
                    PotenciRec:=PotenciaRec(p,q-1)*p
end;

{--------------Programa Principal-----------------}
begin
     ClrScr;
     write('Qual o valor da base?');
     readln(x);
     writeln;
     write('Qual o valor do expoente?');
     readln(n);
     writeln;
     writeln(x:3:1,'elevado a ',n,'=,
                              PotenciaRec(x,n):4:1);
     repeat until keypressed
end.
```

8.2. A recursividade na ordenação de vectores

Problema 8.

Utilizar a recursividade para realizar a ordenação de um vector.

Resolução:

Apresenta-se seguidamente o algoritmo Quick Sort, que utiliza a recursividade para ordenar um vector.

Programa Quick Sort em Pascal

```
program quick_sort;

uses
     WinCrt;

var
     vector     :Array[1..30] of integer;
     n,i        :integer;

{----------------Procedure LeVector----------------}

procedure LeVector;

begin
     ClrScr;
     write('Quantos elementos tem o vector?
     [1..30]');
     readln(n);
     writeln;
     For i:=1 To n Do
          begin
               ClrScr;
               write('Digite o',i,'elemento:');
               readln(vector[i]);
          end;
     ClrScr;
end;

{---------------Procedure EscreveVector-------------}

procedure EscreveVector;

begin
     ClrScr;
     writeln('O VECTOR ORDENADO');
     writeln;
     writeln;
     For i:=1 To n Do write(vector[i],' ')
end;
```

```
{---------------Procedure Quick_Sort----------------}

procedure QUICKSORT(prim, ult:integer);

{Os parametros PRIM e ULT corrspondem, na primeira
chamada, ao primeiro e ao último índices,
respectivamente, do vector considerado.}

var
     dir,esq        :integer; {índices do vector}
     pivot, troca   :integer; {elementos do vector}
begin
     dir:=prim;
     esq:=ult;
                    {elemento central do vector};
     pivot:=vector[(prim+ult)div2];
     repeat
          while vector[dir]<pivot do
               {DIR desloca-se para a direita}
               dir:=dir+1
          while vector[esq]<pivot do
               {ESQ  desloca-se para a direita}
               esq:=esq+1
          if dir≤esq Then
                    begin
                         troca:=vector[dir]
                         vector[dir]:=vector[esq];
                         vector[esq]:=troca;
                         dir:=dir+1;
                         esq:=esq-1;
                    end;
          until dir>esq;
          if prim<esq then QUICKSORT (prim,esq);
          if ult>dir then QUICKSORT (dir,ult)
end;

{-----------------Programa Principal----------------}

begin
     LeVector;
     QUICKSORT(1,n);
     EscreveVector;
     repeat until keypressed
end.
```

8.3. A Recursividade em Jogos de Estratégia

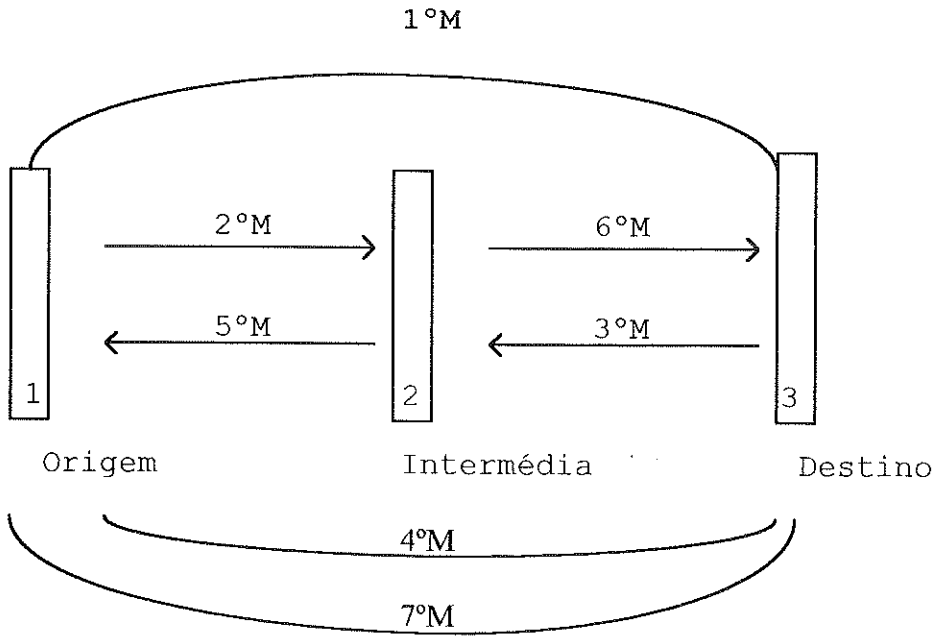
Problema 8.5 **A Torre de Hanoi**

Existem três estacas no chão, sendo que na estaca de origem estão três discos colocados por ordem crescente. Pretende-se mudar todos os discos da estaca origem para a estaca destino, sabendo que só podemos movimentar um disco de cada vez; não e não podemos colocar um disco em cima de outro menor.

Desenvolver um algoritmo que estabeleça uma estratégia para qualquer número de discos.

Resolução:

No esquema que se segue estão indicados os movimentos que se devem realizar para três discos.



Estratégia Geral

1. ***Mudar os (n-1) discos superiores da estaca da esquerda (Origem) para a do meio (Intermédio), usando a da direita (Destino) para armazenamento intermédio.***
2. ***Mudar o disco remanescente (na origem), para a estaca da direita (Destino);***

O Programa da Torre de Hanoi em Pascal

```
program Hanoi;
     uses
          WinCrt;

     Var
          n    :integer;

{-------------- Procedure Transfere -------------}

procedure transfere(n,origem,destino,outra :integer);

     {---------- procedure MoveDisco -----------}
     procedure MoveDisco(origem, destino :integer);

     begin
     writeln('Movimente',origem:1,'para',destino:1);
     end;

begin                {transfere}

     if n>o then
          begin
               transfere(n-1,origem,outra,destino);
               MoveDisco(origem,destino);
               transfere(n-1,outra,destinoorigem)
          end;
end;

{--------------- Programa Pricipal----------}

begin
     ClrScr;
     write('digite o número de discos existentes na
estaca origem:');
     reradln(n);
     writeln;
     transfere(n,1,3,2);
     repeat until keypressed;
end.
```

Problema 8.6

Diga o que executa o seguinte programa em Pascal.

```
Program QueFaçoEu;

var
     mais :char;

procedure troca;     {-- procedimento troca ----}

var
     c    :char;
begin
     read(c);
     if not eoln then troca;
     write(c)
end;

begin          {------ programa principal -----}
     repeat
          writeln('digite uma linha de texto e
prima enter');
          troca;
          writeln('quer tentar de novo?');
          readln(mais);
     until mais='n';
end.
```

Solução:

"asrevni medro rop a-evercse e otxet ed ahnil amu odalcet od êl amargorp O".

A solução foi processada pelo próprio programa.

BIBLIOGRAFIA

1. FARRER, Harry, e outros - *Algoritmos Estruturados.* Editora Guanabara.

2. FINDLAY, W. & D. A. Watt - *Introdução à Programação em Pascal.* Edições CETOP.

3. GUIMARÃES, Ângelo de Moura, LAGES, Newton A. C. - *Algoritmos e Estruturas de Dados.* Livros Técnicos e Cientificos, S. A.

4. GOTTFRIED, Byron S. - *Programação em Pascal.* Schaum, McGraw-Hill.

5. LIMA, Jorge Reis - *Programação de Computadores.* Porto Editora.

6. O'BRIEN, Stephen K. - *Turbo Pascal, Total e Completo.* Osborne/ McGraw-Hill.

7. TREMBLAY, Jean-Paul, BLUNT, R.B., *An introduction to Computer Science: an algorithmic approach.* New York, McGraw-Hill Book Company.

8. TREMBLAY, Jean-Paul, SORENSON, Paul G. - *An introduction to data structures with applications.* McGraw-Hill International Editions.

9. WIRTH, Niklaus - *Algoritmos e Estrutura de Dados.* Prentice Hall do Brasil.

10. Manuais do TURBO PASCAL, que constam na Biblioteca.