Instituto
Politécnico do Porto

Instituto Superior de Engenharia
Departamento de Engenharia Informática (DEI)

Introdução à Informática
e
Algoritmia

Alberto Sampaio e Isabel Sampaio

Texto de apoio

ÍNDICE

I INTRODUÇÃO À INFORMÁTICA 1

1. CONCEITOS GERAIS 1
2. EVOLUÇÃO HISTÓRICA DOS COMPUTADORES 2
3. REPRESENTAÇÃO DA INFORMAÇÃO 3
4. EXERCÍCIOS 4

II INTRODUÇÃO À PROGRAMAÇÃO 5

5. RESOLUÇÃO DE PROBLEMAS EM INFORMÁTICA 5
6. ALGORITMOS 6
7. TIPOS DE DADOS, VARIÁVEIS, OPERAÇÕES ELEMENTARES E EXPRESSÕES 9
8. ALGUMA NOTAÇÃO 11
9. ESTRUTURAS DE CONTROLO DE FLUXO 11
   - *9.1 Sequência 12*
   - *9.2 Decisões 14*
   - *9.3 Repetições 18*
10. TRAÇAGEM 22
11. EXERCÍCIOS 23

III SUBROTINAS 24

12. INTRODUÇÃO 24
13. FUNÇÕES 24
14. PROCEDIMENTOS 25

IV VECTORES 26

1. INTRODUÇÃO 26
2. DEFINIÇÃO 27
3. EXERCÍCIOS 28

ISEP

I INTRODUÇÃO À INFORMÁTICA

1. CONCEITOS GERAIS

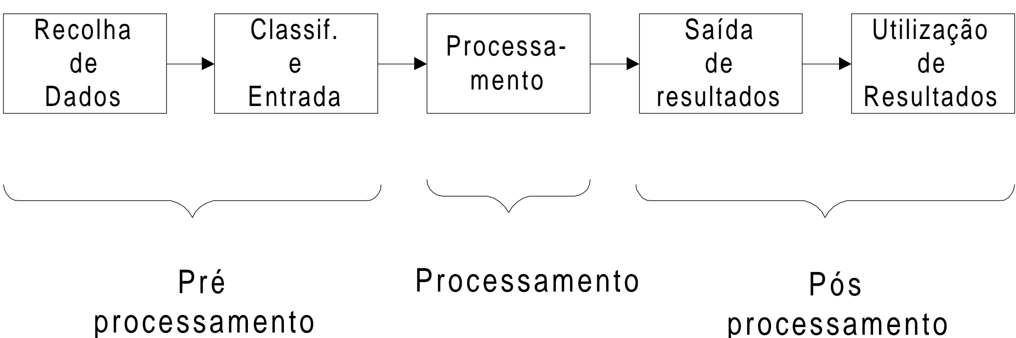
- Dados: Informação automatizável. Diz-se automatizável, se pode ser sujeita a um conjunto de operações repetitivas.

  A informação ou é útil ou não interessa: Para a informação ser útil, é essencial que os dados recolhidos nos conduzam a resultados utilizáveis.

DADOS ⇄ OPERAÇÕES SOBRE DADOS ⇄ RESULTADOS

- Definição de Informática: A informática (Informação Automática) é a ciência do tratamento racional, nomeadamente por processos automáticos, da informação, considerada como suporte do conhecimento humano e da comunicação no domínio técnico, económico e social.

- Ciclo de informação



- O que caracteriza os dados?

- conteúdo
- estrutura
- tipo

Quanto à estrutura e tipo, é necessário proceder a um estudo da sua composição e interligações.

2. EVOLUÇÃO HISTÓRICA DOS COMPUTADORES

1ª Geração

- válvulas
- programação em linguagem máquina
- velocidade tratamento - 10^{-3} s (ms)
- grande aquecimento
- fraca fiabilidade

Exemplo: ENIAC - primeiro computador

2ª Geração

- Transístores e circuitos impressos
- Programação linguagem simbólica
- Velocidade - 10-6 s (μs)
- Memória por anéis de ferrite
- Aumento da fiabilidade

Exemplo: IBM 1401

3ª Geração

- Circuitos integrados
- Velocidade - 10-9 s (ns)
- Aumento da potência e fiabilidade
- Tratamento de dados à distância (telemática)
- generalização das linguagens evoluídas

Exemplo: IBM 360

4ª Geração

- Memória electrónica
- Velocidade - 10-12 s (pico segundo)
- Miniaturização (LSI e VLSI)
- Microprocessadores
- Queda de preços

Exemplo: Microcomputadores (PC)

5ª Geração

ISEP

- Desencadeada por um programa japonês
- Massificação de sistemas e inteligência artificial
- Velocidade tratamento muito elevada
- Arquitecturas paralelas e distribuídas

3. REPRESENTAÇÃO DA INFORMAÇÃO

A representação da informação no computador está de acordo com o modo de funcionamento dos computadores. Como sabe, um programa é executado a partir da memória pricipal do computador. Sendo esta um dispositivo digital de dois estados, ligado e desligado, então o sistema de numeração que naturalmente se adequa para a representação da informação é o binário em virtude de ser constituído por apenas dois dígitos, o 0 e o 1, a que chamamos bits.

Antes de prosseguirmos, e para aqueles que estão mais "presos" ao sistema de numeração decimal (0 a 9), informamos que a <u>única</u> diferença entre estes dois sistemas é o número de dígitos que os compôem.

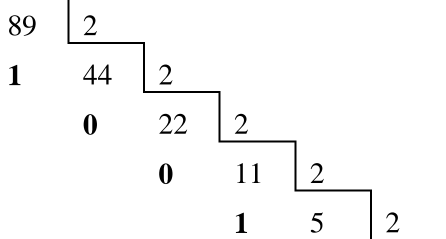
Vejamos a numeração nos dois sistemas:

Decimal	Em qualquer sistema, esgotados os algarismos	**Binário**
0	disponíveis, constituem-se agrupamentos como	0
1	forma de representar valores ainda maiores	1
2	*Note o agrupamento*	10
3		11
4	*Agora com três*	100
5		101
6		110
7		111
8	*Agora com quatro*	1000
9		1001
10	*Note o agrupamento*	1010
...		**...**

ISEP

Também é possível realizar a conversão de valores de um sistema para o outro. Comecemos por ver um exemplo.

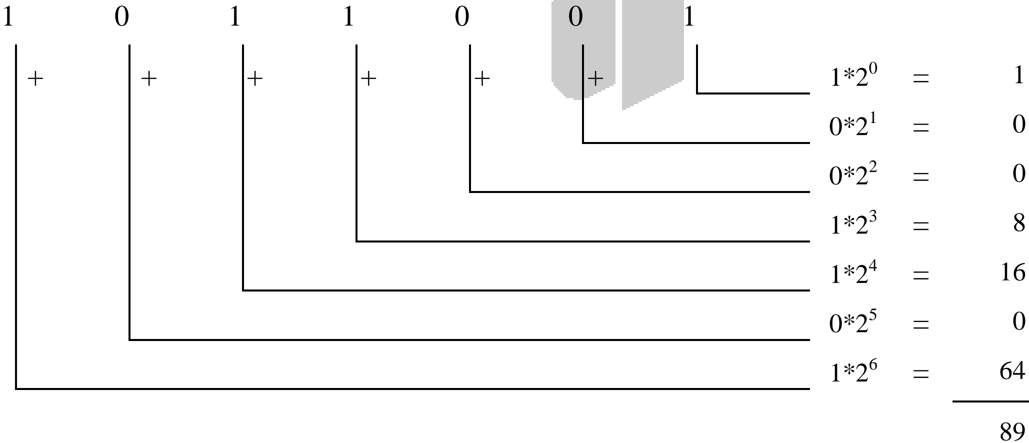
Número a converter: 89_{10} (sistema de numeração decimal).



1 2 | 2
0 1 | 2
1 0

O número binário contrói-se com os restos obtidos, do último para o primeiro. Assim o correspondente de 89_{10} em binário é 1011001_2.

A conversão de binário para decimal do valor obtido deve resultar em 89.



A conversão para o sistema de numeração decimal a partir de outros, e vice-versa, obedece exactamente às mesmas regras, substituindo-se o valor *2* pelo do novo sistema.

Conversão de um número composto por *m* digítos numa qualquer base *b* $(n_m\ n_3\ n_2\ n_1)_b$, para base 10:

$$N_{10}, = n_m * b^{m-1} + n_3 * b^3 + n_2\, b^2 + n_1 * b^0$$

Em conclusão, a conversão entre dois sistemas de numeração quaisquer faz-se em dois passos; primeiro conversão para decimal e só depois de decimal para o sistema pretendido.

4. EXERCÍCIOS

Exercício 1: Realize as seguintes conversões entre sistemas de numeração.

- Do número 101010011111_2 para o sistema Hexadecimal(16).
- Do número $9AF_{16}$ para o sistema binário(2).
- Do número 101010011111_2 para o sistema Octal(8).

ISEP

II INTRODUÇÃO À PROGRAMAÇÃO

ISEP

5. RESOLUÇÃO DE PROBLEMAS EM INFORMÁTICA

Uma afirmação como "calcule a nota mais elevada da turma na disciplina de Introdução à Computação", especifica o que pretendemos, mas é demasiado ambígua para poder ser resolvida por um computador pois faltam detalhes tais como: que notas, onde estão, devem ou não ser incluídos alunos ausentes, etc.[1]

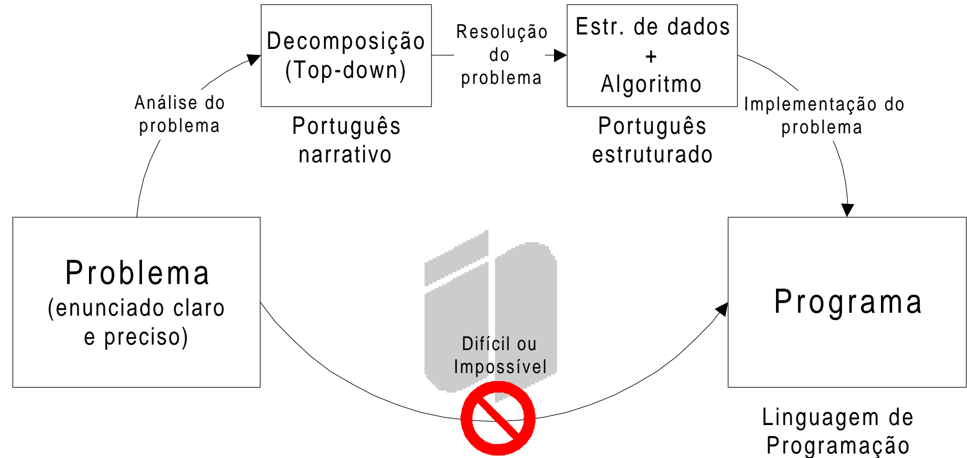
Deste modo a programação de computadores pode tornar-se difícil. No entanto, podemos torná-la mais fácil. Tal pode ser feito se subdividirmos um problema sistematicamente em partes mais pequenas e menos complexas chegando a um ponto em que compreendemos claramente cada uma das partes. A partir daqui podemos, mais facilmente, indicar sem ambiguidade os passos (algoritmo) para a solução do problema.

1. Analisar o problema
   a) Conhecer o problema: ouvir o problema, entendê-lo, perceber qual o objectivo.
   b) Descrever o problema: subdividir o problema (esquematizar), detalhar.

2. Resolver o problema: escrever passo a passo o raciocínio da solução do problema; verificar se não existe ambiguidade.

3. Implementar: esta fase acontece apenas após o problema estar resolvido e consiste em implementar o algoritmo numa linguagem de programação.

[1] Para um estudo mais detalhado ver: "Ciência dos Computadores" de Tremblay e Bunt, McGraw-Hill. Este capítulo baseia-se nessa obra.



O "top-down" assume frequentemente uma de duas formas: gráfica, através de caixas dispostas hierarquicamente; ou narrativa indentada. Os algoritmos também são representados de uma de duas formas: português estruturado indentado; ou em notação gráfica, o chamado fluxograma. No texto que se segue serão mostradas estas quatro formas de representação.

A dificuldade maior encontra-se nas fases de análise e resolução do problema, já que a implementação numa linguagem de programação é "directa" desde que estejamos de posse da resolução do problema (sob a forma das estruturas de dados e algoritmos).

6. ALGORITMOS

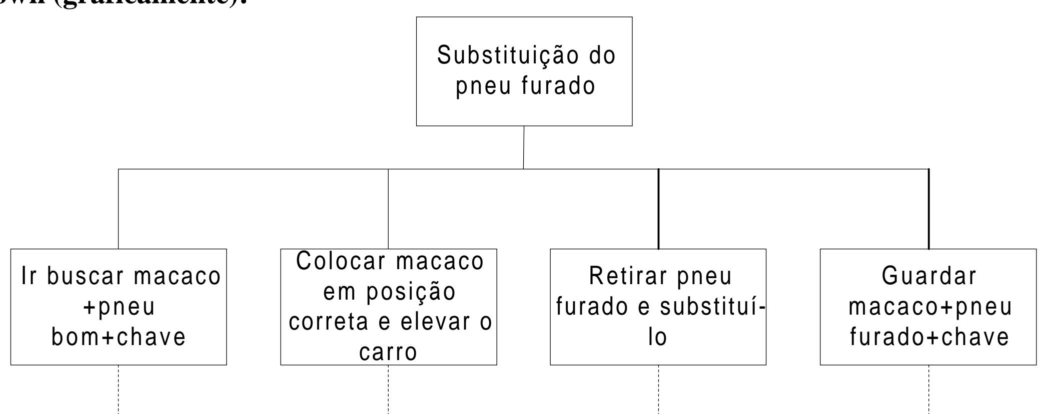
Os computadores apenas fazem aquilo que mandamos, pelo que não deve existir qualquer ambiguidade nas instruções que damos ao computador. Assim, um algoritmo pode ser definido como uma sequência ordenada, e sem ambiguidade, de passos que levam à solução de um dado problema. Por exemplo, as indicações dadas para se chegar a uma dada rua constituem um algoritmo para se encontrar a rua.

Mas é importante que os algoritmos tenham as seguintes propriedades: passos simples e sem ambiguidade, devendo estar numa ordem cuidadosamente definida; devem também ser eficazes, i.e., devem resolver o problema num número finito de passos.

ISEP

Exemplo 1: Elabore um algoritmo que permita a um robot efectuar a mudança do pneu furado de uma viatura.

1ºTopDown (graficamente):



2º Definir as operações elementares:

```
dar passos
abrir ou fechar
pegar ou largar
rodar
deslocar no sentido ...
capacidade de decisão
```

3º Narrativa Identada: [Substituição pneu furado]

```
1. [Buscar macaco, chave e pneu sobresselente]
1.1 Repetir até chegar ao carro
1.1.1 Dar passos
1.2 Abrir a mala
1.3 Pegar macaco, chave e pneu bom
1.4 Repetir até chegar ao pneu furado
1.4.1 Dar passos

2. [Colocar o macaco em posição correta e elevar o carro]
2.1 Largar o pneu e a chave
2.2 Encaixar macaco
2.2.1 Deslocar o macaco no sentido do carro até encaixar
2.3 Elevar o carro
2.3.1 Repetir até carro elevado
2.3.1.1 Rodar manivela no sentido horário
3.[Retirar o pneu furado e substituí-lo]
3.1 Pegar na chave
3.2 Repetir quatro vezes
3.2.1 Deslocar a chave até encaixar na porca
3.2.2 Repetir até porca cair
3.2.2.1 Rodar sentido anti-horário
3.3 Largar chave
3.4 Pegar no pneu furado
3.5 Deslocá-lo sentido contrário ao carro
3.6 Largar pneu furado
3.7 Pegar no pneu bom
3.8 Deslocar pneu no sentido do carro
3.9 Encaixar pneu
3.10 Pegar na chave
3.11 Repetir quatro vezes
3.11.1 Pegar na porca
3.11.2 Encaixar porca
3.11.3 Pegar chave
3.11.4 Repetir até ficar apertado
3.11.4.1 Rodar no sentido horário
3.12 Largar chave

4. [Guardar o macaco, chave e o pneu furado]
4.1 Baixar o carro
4.1.1 Rodar manivela no sentido contrario ao horário
4.2 Desencaixar macaco
4.3 Pegar pneu furado+macaco+chave
4.4 Repetir até à mala
4.4.1 Dar passos
```

ISEP

```
4.5 Largar chave+macaco+pneu furado
4.6 Fechar mala
[FIM]
```

Exemplo 2: Elabore um algoritmo que permita a um robot efectuar a mudança de uma lâmpada fundida.

1ºTopDown: Substituição da lâmpada fundida

```
1. Preparar acessos à lâmpada fundida
2. Retirar lâmpada fundida
3. Escolher lâmpada nova
4. Colocação da lâmpada nova
5. Arrumar escada
```

2º Definir as operações elementares:

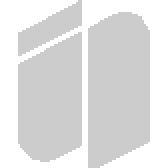
```
dar passos
subir/descer degraus
pegar/largar objectos
rodar objectos sentido directo/indirecto
capacidade de decisão
```

3º Narrativa Identada (Algoritmo):

```
1. [Preparar acessos à lâmpada fundida]
1.1 Repetir enquanto não chegar à escada
1.1.1 Dar passos
1.2 Pegar objecto (escada)
1.3 Repetir enquanto não chegar debaixo lâmpada fundida
1.3.1 Dar passos

2. [Retirar lâmpada fundida]
2.1 Repetir enquanto não chegar à lâmpada fundida
2.1.1 Subir degraus
2.2 Repetir enquanto não soltar a lâmpada fundida
2.2.1 Rodar objecto(L.F.) no sentido indirecto
2.3 Repetir enquanto não chegar ao chão
2.3.1 descer degraus

3.[Escolher lâmpada nova]
3.1 Repetir enquanto não chegar junto gaveta das lâmpadas novas
1.3.1 Dar passos
3.2 Repetir enquanto houver lâmpadas novas ou potência L.N. diferente de
potência L.F.
3.2.1 Pegar objecto (L.N.)
3.2.2 Se potência L.N. = Potência L.F.
3.2.2.1 Então largar a L.F.
3.2.2.2 Senão largar a L.N.

4. [Colocação da lâmpada nova]
4.1 Repetir enquanto não chegar à escada
4.1.1 Dar passos
4.2 Repetir enquanto não chegar ao casquilho
4.2.1 Subir degraus
4.3 Repetir enquanto não firmar a lâmpada nova
4.3.1 Rodar objecto(L.N.) no sentido directo
```

ISEP

```
4.4 Repetir enquanto não chegar ao chão
4.4.1 descer degraus
5. [Arrumar escada]
5.1 Pegar objecto(escada)
5.2 Repetir enquanto não chegar ao sítio da escada
5.2.1 Dar passos
5.3 Largar objecto (escada)
6. [FIM]
```



7. TIPOS DE DADOS, VARIÁVEIS, OPERAÇÕES ELEMENTARES E EXPRESSÕES

Os nossos programas de computador vão manipular dados. Estes podem variar de tipo. Por exemplo, o nome de um aluno é um dado alfabético, enquanto as suas notas são do tipo numérico. No computador tipos de dados diferentes têm representações também diferentes sendo manipulados de forma distinta. Assim podemos considerar duas grandes classes de tipos de dados:

- A doa numéricos, que podem ser inteiros ou reais
- As cadeias de caracteres, agrupamento contendo caracteres alfabéticos, algarismos ou outros caracteres. As cadeias são representadas entre aspas.

Geralmente é necessário guardar os dados manipulados pelo algoritmo (ou programa), durante a sua execução. Para o efeito são utilizadas variáveis. Estas caracterizam-se por um nome, que se inicia sempre por uma letra e deve ser o mais sugestivo possível. No computador, a cada variável corresponde um determinado espaço da memória do computador.

Os dados podem ser guardados nas variáveis, através da operação de atribuição (representada pelo operador =), ou introduzidos directamente pelo utilizador (aquele que executa ou usa o algoritmo ou programa que o programador desenvolveu) através de uma instrução de leitura (representada pelo instrução LER). A atribuição faz-se da direita para a esquerda, isto é, o valor do lado direito da atribuição é atribuído à variável que se encontra do lado esquerdo.

```
Media = 5      [coloca o valor 5 na variável media]
```

O texto entre parêntesis rectos é comentário, ou seja, não é executado, o que significa que tem apenas interesse para quem escreve o algoritmo.

Na leitura, a variável indicada recebe o dado introduzido pelo utilizador; é uma forma de o utilizador poder interagir com o programa.

```
Ler(idade)          [guarda na variável idade o valor introduzido pelo utilizador]
```

Estas duas expressões são destrutivas, ou seja, o valor anterior na variável é substituído pelo actual.

Num algoritmo não basta ler valores do mundo exterior, é também necessário fornece-los. Para tal é utilizada a instrução ESCREVER.

Exemplo: Neste exemplo são adicionadas duas idades introduzidas pelo utilizador e visualizado o resultado.

```
Escrever("Introduza duas idades:")     [pede para o utilizador introduzir
duas idades]
```

ISEP

```
Ler(idade1, idade2) [assim que o utilizador introduzir as idades, estas são guardadas em idade1 e idade2]
Soma=idade1+idade2  [adiciona as idades e atribui o resultado à variável soma]
Escrever(“A soma das idades:”, idade1, “e “, idade2, “é igual a”, soma) [Comunica o resultado]
```

Supondo que as idades introduzidas eram 24 e 30 respectivamente, então o resultado da instrução *Escrever* seria:

```
A soma das idades: 24 e 30 é igual a 54
```

Operações

Para além da atribuição, do exemplo anterior, existem outras operações agrupadas por:

Operador	Descrição
Operador Atribuição	
=	atribui o valor à direita à variável à esquerda
Operadores Aritméticos:	
+	adição
-	subtracção
*	multiplicação
/	divisão
%	módulo; resto da divisão de dois inteiros
Operadores Relacionais:	
=	igualdade
<=	menor que
>=	maior que
<>	desigualdade
Operadores Lógicos:	
NEGAR	negação
E	conjunção
OU	disjunção

Nas expressões aritméticas a avaliação faz-se dos operadores com maior precedência para os de menor. A ordem de precedência dos operadores anteriores é a seguinte:

- * /
- + -
- = < > <= >=
- NEGAR E OU
- =

A utilização de parêntesis permite alterar a ordem de avaliação das expressões, dando prioridade às que se encontram entre parêntesis.

ISEP

Para além destes operadores existem também funções embutidas, isto é, que fazem parte da linguagem, como por exemplo a função *SQRT(valor).*

8. ALGUMA NOTAÇÃO

Nos exemplos anteriores os algoritmos forma escritos em português narrativo ainda muito pouco estruturado. Mas a linguagem de escrita de algoritmos que pretendemos ver utilizada convém que seja um pouco mais estruturada, isto é mais próxima de uma linguagem de programação. Vamos aumentar o formalismo e passar a utilizar um pseudo código (português estruturado), que iremos apresentando ao longo da resolução dos problemas que se seguem.

No nosso pseudo código os algoritmos começam e terminam sempre pelas palavras chave INÍCIO e FIM respectivamente. Enquanto em fluxograma se utiliza para ambas as situações um rectângulo de cantos arredondados.

Palavras chave/símbolos	Significado
Pseudo código:	
INICIO	representa o inicio do algoritmo
FIM	representa o fim do algoritmo
LER	operação de leitura
ESCREVER	operação de escrita
Fluxograma:	
INÍCIO FIM	o início e o fim respectivamente
Data Document	a leitura e escrita respectivamente[1].

Entre as duas palavras *INÍCIO* e *FIM* colocam-se as instruções do algoritmo.

A ligação entre os símbolos faz-se através de setas, indicando qual a sequência de execução das instruções (o fluxo de controlo).

9. ESTRUTURAS DE CONTROLO DE FLUXO

Existe um teorema em informática que diz que se pode escrever qualquer programa recorrendo a apenas três formas de controlo do fluxo de execução de um programa; a sequência, a decisão e a repetição. Os exemplos anteriores utilizavam a sequência e a repetição. Não vamos explicar neste momento cada uma delas, deixando antes a sua apresentação depender da necessidade da sua utilização para a resolução dos problemas que irão surgir.

[1] Por vezes usaremos apenas o primeiro para ambas as operações, mas com a indicação de qual a operação.

ISEP

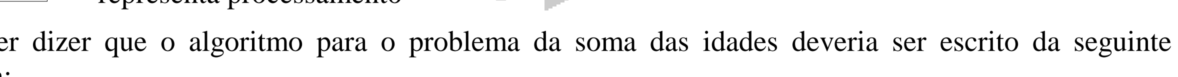
9.1 SEQUÊNCIA

Execução sequencial significa que as instruções de um programa são executadas umas a seguir às outras pela ordem em que se encontram no programa. A notação necessária para a elaboração de algoritmos sequenciais é a seguinte:

Pseudo código:

A apresentada até agora.

Fluxograma:

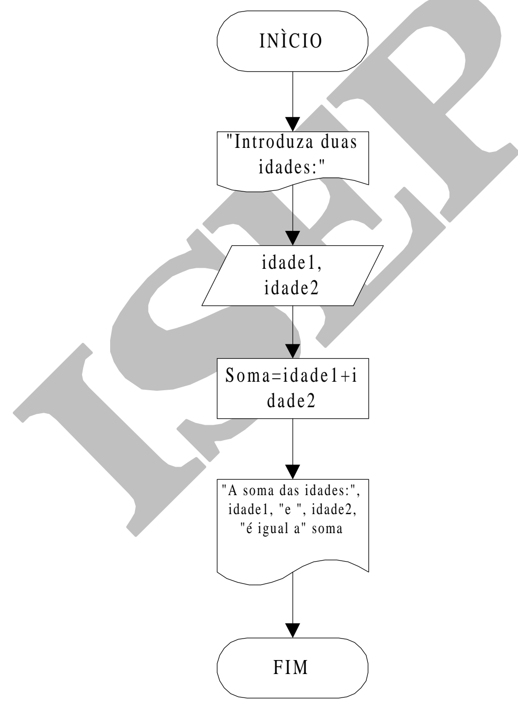


representa processamento

Isto quer dizer que o algoritmo para o problema da soma das idades deveria ser escrito da seguinte maneira:

```
INICIO
Escrever("Introduza duas idades:")     [pede duas idades ao utilizador]
Ler(idade1, idade2)  [guarda-as em idade1 e idade2]
Soma=idade1+idade2   [adiciona-as e atribui o resultado a soma]
Escrever("A soma das idades:", idade1, "e ", idade2, "é igual a", soma)
FIM
```

E em fluxograma:



Exemplo 1: Um artista da nossa praça produz cubos de 2cm de aresta, que depois pinta com uma tinta que adquire ao preço de 500$00 por lata de 1,5dl. Com um dl de tinta o artista consegue pintar $5cm^2$ da superfície de um cubo. Calcule quanto custa pintar um cubo inteiro (seja sistemático). Nota: área do cubo = aresta*aresta*6

Resolução - Para a resolução do problema e partindo dos dados disponíveis,

aresta do cubo (2)

preço da lata (500)

tinta da lata (1,5)

capacidade de pintura por dl (5)

é necessário: determinar a área do cubo, em seguida qual a tinta necessária para pintar o cubo, o número de latas necessárias para pintar o cubo e finalmente o custo de pintar o cubo. Viria então,

área=aresta^2*6 o que dá 24

tinta necessária=24/5 o que dá 4,8

número de latas=4,8/1,5 o que 3,2

custo= 3,2 *500$00 o que resultaria num custo de 1600$00

Exemplo 2: Proceda às alterações ao algoritmo anterior, que entender necessárias para que todos os dados possam ser indicados em cada execução do algoritmo (i.e., sem ter de refazer os cálculos).

Resolução - Consiste em substituir os valores (constantes) pelas variáveis. Assim, de cada vez que o pintor quer calcular o custo, torna-se necessário perguntar quais os valores em causa. As variáveis para os valores de entrada poderiam ser:

aresta - aresta do cubo

preço - preço da lata

tinta_lata - quantidade de tinta da lata

capacidade - capacidade de pintura por dl

O algoritmo resultante:

```
INÍCIO [Custo de pintar um cubo(dados variáveis)]
escrever("Qual a aresta?)
ler(aresta)
escrever("qual o preço de cada lata?")
ler(preço)
escrever("Qual a quantidade de tinta da lata?")
ler(tinta_lata)
escrever("Qual a capacidade de pintura, em cm2, por cada dl?")
ler(capacidade)
área=aresta^2*6
tinta_necessária=área/capacidade
número_latas=tinta_necessária/tinta_lata
custo= número_latas*preço
escrever("O custo de pintar o cubo é de: ", custo)
FIM
```

Relembrar que o texto entre parêntesis rectos é comentário, ou seja, não é executado, o que significa que apenas interessa a quem escreve o algoritmo.

Exemplo 3: Elabore um algoritmo que leia um número inteiro representando segundos, e imprima o seu correspondente em horas, minutos e segundos.

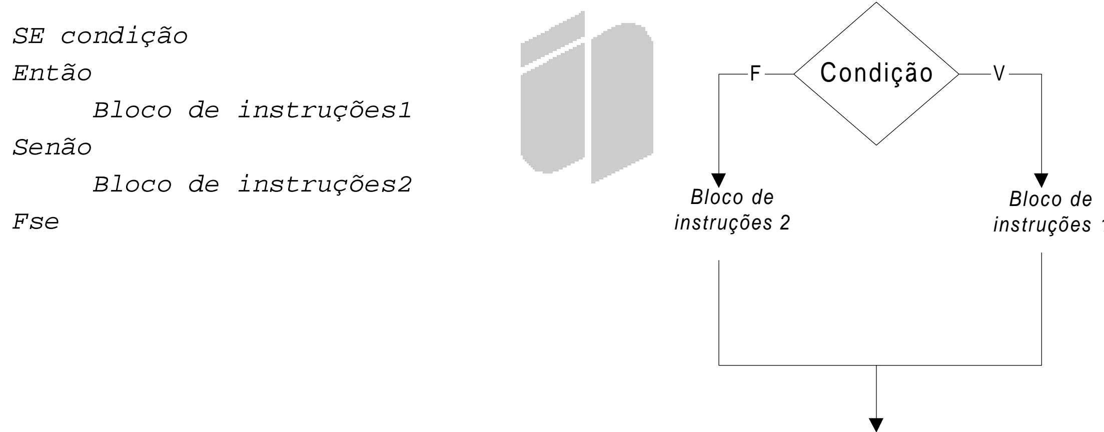
Exemplo: 3662s $\rightarrow$ 1h 1m 2s

ISEP

9.2 DECISÕES

Por vezes e necessário tomar decisões que dependem de determinadas condições. Por exemplo, imprimir os nomes dos clientes de um banco com um saldo superior a um certo valor. As decisões são representadas em português estruturado e fluxograma das seguintes formas:

```
SE condição
Então
     Bloco de instruções1
Senão
     Bloco de instruções2
Fse
```



Em que *Bloco de instruções* representa uma ou mais instruções.

Se a condição for verdadeira executa as instruções do *Bloco de instruções1*, se for falsa executa *Bloco de instruções2*. Portanto executa apenas um dos blocos.

A condição *condição* pode ser simples ou composta (dijunções e/ou conjunções). As disjunções representam-se por *OU* e as conjunções por *E*.

Exemplo 1: Considere agora, que o nosso artista também passou a produzir e pintar cilindros. Reescreva o algoritmo desenvolvido anteriormente para permitir calcular o custo do cubo ou do cilindro, conforme se pretender. Nota: área_da_base=(π*diâmetro^2)/4 ,área_da_superfície=π*diâmetro*altura, área total=2*área_da_base+ área_da_superfície.

Resolução: Começar por determinar as estruturas de dados envolvidas (de entrada/saída) vai-nos ajudar a resolver o problema. Para além dos dados anteriores, há a considerar o tipo de objecto, cubo ou cilindro. A indicação do tipo de objecto é da responsabilidade do utilizador, pelo que a resolução do problema passa por perguntar ao utilizador qual o tipo de objecto para que quer calcular o custo de pintura. Consoante o objecto variam os dados a pedir ao utilizador e a forma de cálculo da área. Isto é, após o utilizador introduzir o tipo de objecto é necessário optar por uma das fórmulas de cálculo da área.

```
INÍCIO [Custo de pintar um cubo(dados variáveis)]
Escrever("Qual o objecto?")
Escrever("Introduza: 1 para cubo, 2 para cilindro")
Ler(obj)
Se obj=1 [se utilizador escolheu cubo]
Então    escrever("Qual a aresta?)
ler(aresta)
área=aresta^2*6
Senão    Escrever("Diâmetro e altura?")
         Ler(d, h)
         Pi=3,14
area_da_base =(pi*diâmetro^2)/4
         area_da_superfície=pi*diâmetro*altura
         área= area_da_base+ area_da_superfície
```

ISEP

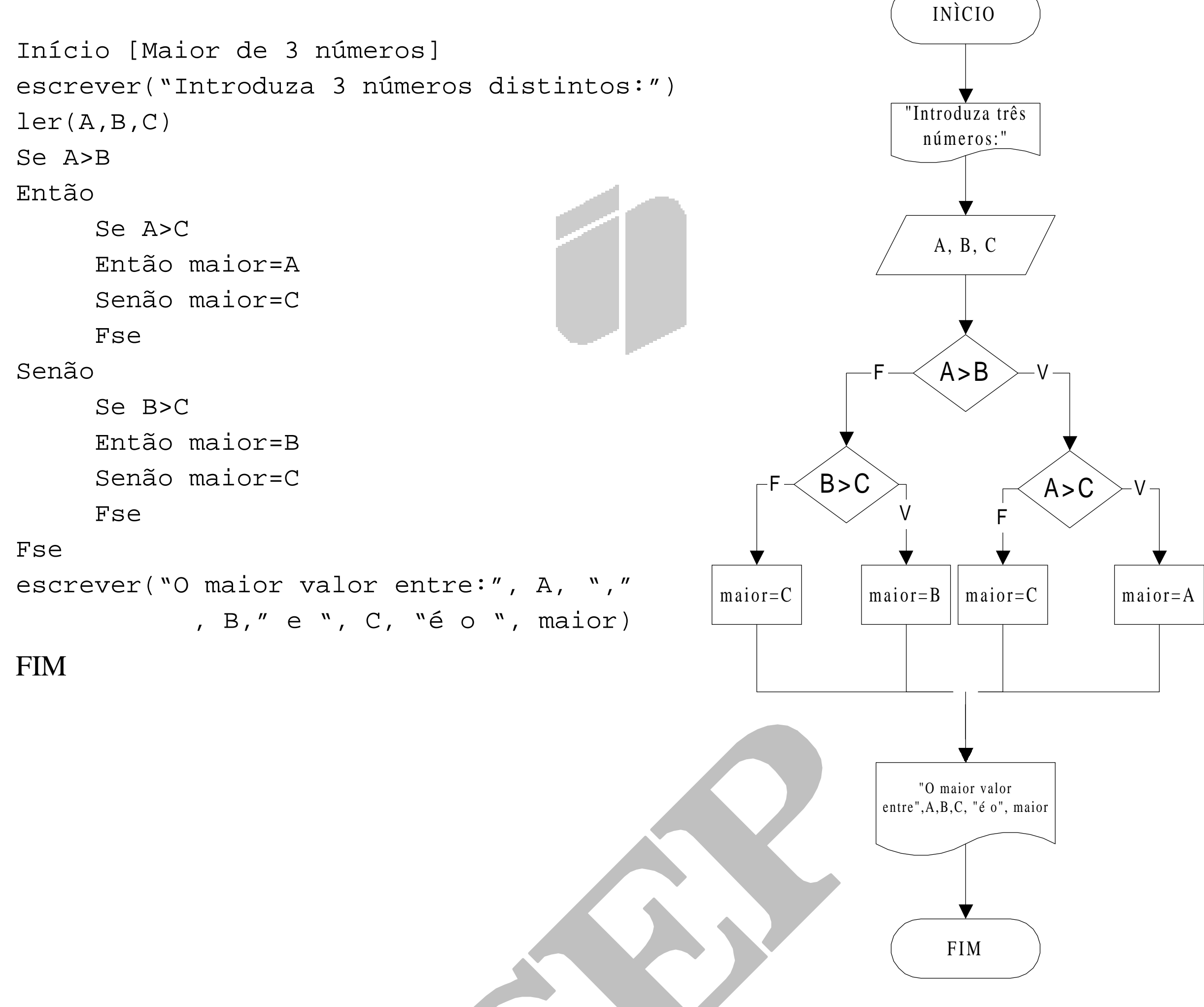
```
Fse
escrever("qual o preço de cada lata?")
ler(preço)
escrever("Qual a quantidade de tinta da lata?")
ler(tinta_lata)
escrever("Qual a capacidade de pintura, em cm2, por cada dl?")
ler(capacidade)
tinta_necessária=área/capacidade
número_latas=tinta_necessária/tinta_lata
custo= número_latas*preço
escrever("O custo de pintar é de: ", custo)
FIM
```

Exercício 1: Este algoritmo pode ser melhorado no sentido de garantir que o custo apenas é calculado nos casos de o utilizador introduzir o valor 1, ou o valor 2 para o tipo de objecto. Outra melhoria poderia ser feita na mensagem com o custo, identificando o objecto para o qual se calculou o custo. Realize as alterações necessárias ao algoritmo para introdução das melhoria sugeridas.

Exemplo 2: Elabore um algoritmo (pseudo código e fluxograma) que, dados 3 números, todos diferentes, determine e imprima o maior dos três.

Resolução: Determinar os dados envolvidos. Neste caso os 3 números e o maior dos três. Em seguida elaborar o algoritmo.

ISEP



```
Início [Maior de 3 números]
escrever("Introduza 3 números distintos:")
ler(A,B,C)
Se A>B
Então
     Se A>C
     Então maior=A
     Senão maior=C
     Fse
Senão
     Se B>C
     Então maior=B
     Senão maior=C
     Fse
Fse
escrever("O maior valor entre:", A, ","
           , B," e ", C, "é o ", maior)
FIM
```


ISEP

Exemplo 3: Elabore um algoritmo (pseudo código e fluxograma) que, dados 3 números, todos diferentes, determine e imprima o maior e o menor deles. Deve procurar determinar o menor enquanto determina o maior.

Resolução: Determinar os dados envolvidos. Neste caso os 3 números, o maior e o menor dos três. Em seguida elaborar o algoritmo.

```
Início [Maior e menor de 3 números]
escrever("Introduza 3 números distintos:")
ler(A,B,C)
Se A>B
Então
   Se A>C
   Então maior=A
         Se B<C
         Então menor=B
         Senão menor=C
         Fse
   Senão maior=C
         menor=B
   Fse
Senão
```

```
   Se B>C
   Então maior=B
         Se A<C
         Então menor=A
         Senão menor=C
         Fse
   Senão maior=C
         menor=A
   Fse
Fse
escrever("O maior entre:", A, ",", B," e ", C, "é o ", maior, "  O menor é
o ", menor)
FIM
```

Exercício 1: Desenvolva um algoritmo (pseudo código e fluxograma) que, dados 4 números, calcule e imprima a diferença entre a soma dos dois maiores e a dos dois menores.

Exemplo 4: Elabore um algoritmo (pseudo código e fluxograma) que permita determinar se um triângulo, dados os seus lados, é ou não rectângulo. Nota: Lembre-se do teorema de Pitágoras.

Resolução: Para um triângulo ser rectângulo é necessário que se verifique a igualdade do teorema de Pitágoras, o que nos obriga a saber qual dos três lados é a hipotenusa e quais são os catetos. Podemos então subdividir o problema em dois:

P1. Determinar a hipotenusa e os catetos

P2. Aplicar o teorema de Pitágoras

Como a hipotenusa será o maior dos três lados, então a resolução do primeiro subproblema passa por determinar o maior dos 3 lados. Os catetos serão os dois lados restantes.

Os dados envolvidos são então: os 3 lados, a hipotenusa e os dois catetos.

```
Início [Triângulo rectângulo]
escrever("Introduza os comprimentos dos 3 lados de 1 triângulo:")
ler(A,B,C)
[P1. Determinar a hipotenusa(o maior) e os catetos(os restantes)]
Se A>B
Então
   Se A>C
   Então HIP=A
         C1=B
         C2=C
   Senão hip=C
         C1=A
         C2=B
   Fse
Senão
   Se B>C
   Então hip=B
         C1=A
         C2=C
   Senão hip=C
         C1=A
         C2=B
```

ISEP

```
   Fse
Fse
[P2. Aplicar o teorema de Pitágoras]
escrever("O triângulo de lados", A, ", ", B, " e ", C)
Se hip^2 = c1^2 + c2^2
Então    escrever(" É Rectângulo)
Senão    escrever(" NÃO É Rectângulo)
FSe
FIM
```

9.3 REPETIÇÕES

Para além das decisões também é muito frequente surgir a necessidade de repetir instruções. Por exemplo, ler as notas dos alunos da turma. As repetições podem assumir diferentes formas, aqui consideraremos duas:

REPETIR ENQUANTO *condição*

Bloco de instruções

FENQ

O *Bloco de instruções* representa uma ou mais instruções.

A condição *condição* pode ser simples ou composta (dijunções e/ou conjunções). As disjunções representam-se por *OU* e as conjunções por *E*.

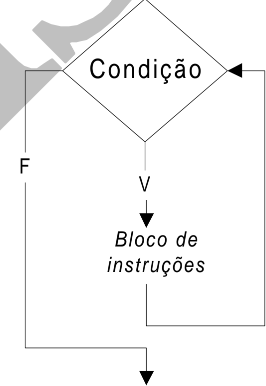
REPETIR PARA *contador=início* ATÉ *fim* PASSO *valor*

Bloco de instruções

FPARA

A segunda forma utiliza-se quando se pretende repetir uma ou mais instruções um número determinado de vezes. O *PARA* é sempre convertível num *ENQUANTO*, o inverso pode não ser.

A repetição representa-se em fluxograma utilizando o símbolo da decisão, como se segue.



Exemplo 1: Escreva um algoritmo (pseudo código e fluxograma) que permita calcular a média das notas da turma na disciplina de Introdução à Computação.

Resolução - Decomposição do problema: saber o número de notas(alunos); somar as n notas; calcular a média das n notas. Dados a considerar: as notas, o número de alunos, a sua soma e a média.

Vamos considerar que o número de alunos é introduzido pelo utilizador do algoritmo que vamos desenvolver. Para tal, necessitamos ler o número de alunos da turma.

```
escrever(" Quantos alunos tem a turma?")
ler(nalunos)
```

Em seguida ler as notas dos NALUNOS uma a uma e somá-la às anteriores. Para isso necessita-se de uma variável que vá guardando as notas já somadas. Chamaremos a essa variável acumuladora *soma* e ela terá que ser inicialmente colocada a zero (inicializada a zero).

```
soma=0
ler(nota)
```

Em seguida somar a nota à variável *soma* e atribuir o resultado a soma novamente.

```
soma=soma+nota [soma e depois atribui]
```

Como já guardamos a nota anterior, podemos utilizar a mesma variável *nota* para guardar a próxima nota a ler.

```
ler(nota)
```

Em seguida somar a nota à variável *soma* e atribuir o resultado a soma novamente.

```
soma=soma+nota
```

E assim sucessivamente até ter lido e somado as *nalunos* notas.

Para saber quando é que já foram lidas todas as notas, é necessário uma variável de controlo, i.e, uma variável que vá tomando valores desde 1 até *nalunos*, por exemplo chamada *i*. O algoritmo seria então:

```
escrever(" Quantos alunos tem a turma?")
ler(nalunos)
soma=0
i=1
ler(nota)
soma=soma+nota [soma e depois atribui]
i=i+1
ler(nota)
soma=soma+nota [soma e depois atribui]
i=i+1
ler(nota)
soma=soma+nota [soma e depois atribui]
......
```

até onde?

Não sabemos, porque o número de alunos só é conhecido no momento em que executarmos o algoritmo. Mesmo que o soubéssemos, como se trata de repetir as mesmas duas instruções não seria melhor que as mandássemos repetir enquanto I fosse menor ou igual a *nalunos*? Vejamos:

```
Repetir enquanto i<=nalunos
   ler(nota)
   soma=soma+nota [soma e depois atribui]
   i=i+1
Fenq
```

Desta forma, testa-se a condição (i<=nalunos) que se for verdadeira manda-se executar o bloco de 3 instruções. Executado este bloco, volta-se a testar a condição. Se ainda for verdadeira, é novamente executado o bloco de 3 instruções. E assim sucessivamente, enquanto a condição for verdadeira, ou seja enquanto não tiverem sido lidas todas as notas. Notar ainda a 3ª instrução do bloco, pois sem ela o valor de *i* permaneceria inalterável.

O algoritmo final:

```
Início [Média das notas]
escrever(" Quantos alunos tem a turma?")
ler(nalunos)
```

ISEP

```
soma=0
i=1
Repetir enquanto i<=nalunos
   escrever("Nota do ", i, "º alunos: ")
   ler(nota)
   soma=soma+nota [soma e depois atribui]
   i=i+1
Fenq
media=soma/nalunos
escrever("A média das notas dos", nalunos, "é de: ", media)
FIM
```

Exemplo 2: Desenvolva o algoritmo (pseudo código e fluxograma) que permita calcular a potência inteira de um número.

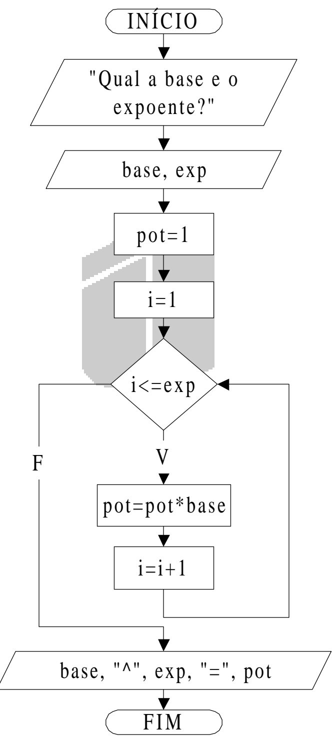
Resolução: Notar que este problema, com execepção da soma e do elemento neutro,é em tudo igual ao anterior, tratando-se agora da acumulação de produtos, o que obriga a iniciá-lo a 1 (elementro neutro da multiplicação).

```
Início [Potência de um número]
escrever("Qual a Base e o Expoente?")
ler(base, exp)
pot=1
i=1
Repetir enquanto i<=exp
   pot=pot*base [multiplica e depois atribui]
   i=i+1
Fenq
escrever(base, "^", exp, " = ", pot)
FIM
```

A versão deste algoritmo usando a construção *PARA* e o respectivo fluxograma, seriam:

```
Início [Potência de um número]
escrever("Qual a Base e o Expoente?")
ler(base, exp)
pot=1
Repetir Para i=1 Até exp Passo 1
     pot=pot*base                  [multiplica e depois atribui]
Fpara
escrever(base, "^", exp, " = ", pot)
FIM
```

ISEP



Exemplo 3: Desenvolva o algoritmo (pseudo código e fluxograma) que permita calcular o factorial de um dado número.

Exemplo 4: Elabore um algoritmo (pseudo código e fluxograma) para determinar se um dado número é ou não primo.

Resolução – Um número é primo se apenas for divisível por ele e pela unidade. Assim para se determinar se um dado número é primo há que verificar se existe outro pelo qual ele seja divisível. Isto implica exprimentar dividi-lo por todos os números entre 2 e o número menos 1. Na realidade basta experimentar entre 2 e a raíz quadrada do número: $\lfloor 2; \sqrt{N} \rfloor$

```
Início [Se número é primo]
escrever("Qual o número?")
ler(num)
D=2           [divisor (1ºvalor do intervalo)]
Resto=1  [serve qualquer valor diferente de 0]
Repetir enquanto D<=SQRT(num) E resto<>0
   Q=num/D
   Resto=num-Q*D
   D=D+1
Fenq
Se resto=0
Então escrever("O número", num, " NÃO é Primo")
Senão escrever("O número", num, " É Primo")
FSe
FIM
```

ISEP

10.TRAÇAGEM

A traçagem consiste em testar um algoritmo para certos valores de entrada, observando o comportamento interno do algoritmo para esses valores e ao longo dos vários passos que compôem o algoritmo. Assim, a primeira etapa consiste em numerar os passos do algoritmo. Em seguida constrói-se uma tabela colocando na primeira linha as entidades que queremos estudar ao longo dos passos do algoritmo. Variáveis e condições, são as entidades que podem variar, e por isso aquelas que devemos estudar. Depois é só executar os passos do algoritmo.

Exemplifiquemos para o algoritmo que permite determinar se um dado número é, ou não, primo, começando por numerar os seus passos.

```
Início [Se número é primo]
     P1.  escrever(“Qual o número?”)
     P2.  ler(num)
     P3.  D=2        [divisor (1ºvalor do intervalo)]
     P4.  Resto=1    [serve qualquer valor diferente de 0]
     P5.  Repetir enquanto D<=SQRT(num) E resto<>0
     P6.       Q=num/D
     P7.       Resto=num-Q*D
     P8.       D=D+1
          Fenq
     P9.  Se resto=0
     P10. Então escrever(“O número”, num, “ NÃO é Primo“)
     P11. Senão escrever(“O número”, num, “ É Primo“)
     FSe
FIM
```

Exemplificado para o número 5

Passos	**num**	**D**	**Resto**	**D<=SQRT(num) E resto<>0**	**Q**	**Resto=0**	**Saída**
P1							número?
P2	5						
P3		2					
P4			1				
P5				V			
P6					2		
P7			1				
P8		3					
P5				F			
P9						F	
P10							5 é Primo
FIM							

11.EXERCÍCIOS

Exercício 1: Considere a sequência de números de Fibonacci:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Escreva um algoritmo que imprima os primeiros *n* números de Fibonacci.

Exercício 2: Desenvolva um algoritmo, para contar o número de vezes que surge o algarismo zero num qualquer número inteiro introduzido pelo utilizador.

Exercício 3: Elabore um algoritmo que permita determinar, e imprima, o maior número e o menor número de uma lista de *n* números introduzidos pelo utilizador.

Exercício 4: Desenvolva o algoritmo que permita calcular o valor do somatório com *n* introduzido pelo utilizador.

Exercício 5: Escreva um programa que imprima o algarismo que se encontra numa dada posição de um número inteiro. O número inteiro e a posição são introduzidos pelo utilizador.

Exercício 2: Em data remota, cerca de 3 séculos a.C., Euclides desenvolveu um método para **?**, o qual é utilizado no algoritmo seguinte:

```
INÍCIO
 Escrever ("Introduzir dois números: ")
 Ler(n1, n2)
 r = n1 % n2        [ % é o operador que permite obter o resto da divisão entre dois inteiros]
 Repetir enquanto r<>0
      n1 = n2
      n2 = r
      r = n1 % n2
 FENQ
 Escrever(n2)
FIM
```

ISEP

a) Após:

- escrever o respectivo fluxograma, e
- executar a "traçagem" utilizando os valores 25 e 15 como dados de entrada,

diga qual pensa ser a finalidade do método que Euclides desenvolveu.

b) Converta o algoritmo para BASIC.

c) Diga qual dos algoritmos considera mais eficiente, este ou o que desenvolveu nas aulas? E porquê.

III SUBROTINAS

12.INTRODUÇÃO

Vamos estudar dois tipos de sub rotinas, funções e procedimentos. A utilização de sub rotinas permite modularizar os programas e encapsular processamento o que resulta em programas mais simples de desenvolver e ler. Quanto mais independentes os módulos (sub rotinas) mais atentamente nos podemos concentrar sobre cada uma ignorando os restantes.

Com a chamada de uma subrotina é transferido o controlo para essa sub rotina. A diferença entre funções e procedimentos consiste no facto de as primeiras retornarem um valor, e as segundas não.

As sub rotinas executam operações sobre dados que lhes são passados. Então as sub rotinas possuem normalmente parâmetros. Por exemplo, a função embutida *SQRT(valor)* tem definido um parâmetro, o do valor para o qual se pretende calcular a raiz quadrada.

As variáveis existentes nas sub rotinas são criadas no momento em que se inicia a execução da sub rotina e destruídas no momento em que a sub rotina termina a sua execução.

ISEP

13.FUNÇÕES

```
FUNCÃO nome [(lista_parâmetros)]
        [bloco_instruções]
        nome = expressão
        [bloco_instruções]
FIMFUNÇÃO
```

nome — O nome da funcão.

lista_parâmetros — Uma ou mais variáveis que especificam os parâmetros a ser passados à função quando esta é chamada:

expressão O valor a retornar pela função.

Quando uma função termina, “deixa ficar” no programa que a chamou o último valor atribuído ao nome da função.

Exemplo 1: Desenvolva uma função que permita calcular o cubo de um número dado como argumento. Em seguida escreva um programa que use essa função.

Resolução: O parâmetro da função será o número e a função retornará ao programa que a invoque o cubo desse número.

```
FUNÇÃO cubo( num )
   cubo=num*num*num
FIMFUNÇÃO
```

Agora o programa que a usa:

```
INÍCIO
   Escrever(“Introduza um número:”)
Ler(n)
Escrever( “O cubo de ”, n, “ = ”, cubo(num) )
FIM
```

Executemos o programa por exemplo para o valor 3:

Entrada/Saída:

```
Introduza um número: 3
O cubo de 3 = 27
```

14.PROCEDIMENTOS

PROCEDIMENTO nome[(lista_parâmetros)]

[bloco_instruções]

FIMPROC

nome O nome da procedimento.

lista_parâmetros Uma ou mais variáveis que especificam os parâmetros a ser passados ao procedimento quando este é chamado:

Exemplo 1: Escreva um procedimento que imprima um número inteiro pela ordem inversa.

Resolução:

```
PROCEDIMENTO impinvnum( num )
   Repetir Enquanto num>0
         Escrever( num MOD 10)
         num=num/10  [divisão de inteiros]
   FEnq
FIMPROC
```

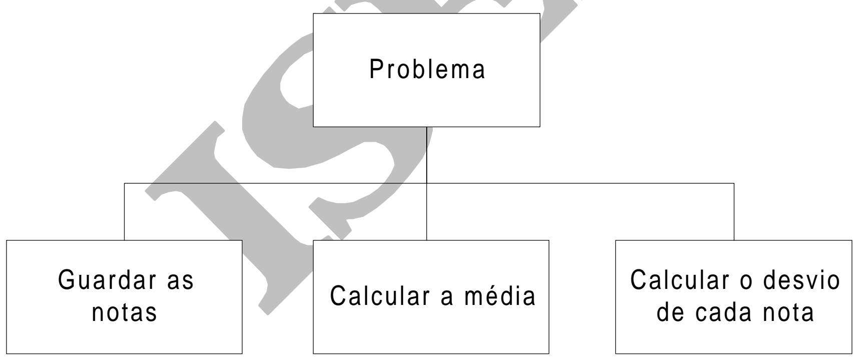
ISEP

IV VECTORES

1. INTRODUÇÃO

Suponha, que se pretendia desenvolver um programa que dadas as notas dos cerca de 4000 alunos do ISEP, calculasse o desvio de cada uma relativamente à média das notas.

Para o cálculo dos desvios é necessário o cálculo prévio da média, o que implica manter as notas após o cálculo da média, ou seja, guardar as notas em variáveis. Vamos começar por decompor o problema, em sub problemas, como se segue:



Uma solução para guardar cada uma das notas, seria definir 4000 variáveis, por exemplo:

nota_1, nota_2, nota_3, nota_4, nota_5, nota_6, nota_7, nota_8, nota_9, . . . , *nota_4000*.

Assim, as instruções para a leitura das notas seriam:

```
Escrever((“Introduza a média do aluno nº 1: “)
Ler(nota_1)
…
```

4000 vezes, o que se revela completamente impraticável. Seria preferível a possibilidade de definir as 4000 variáveis de uma só vez, por exemplo da seguinte forma:

```
nota(1 até 4000)
```

em que *nota(1)* guardaria a nota do aluno 1, *nota(2)* a do aluno 2, e assim sucessivamente fazendo variar o valor do índice até 4000. Mas utilizar constantes como índice continua a obrigar a repetir as 4000 instruções. Então é necessário definir uma variável inteira para índice, por exemplo:

num

Deste modo, para a leitura das 4000 notas, poder-se-ia utilizar um ciclo, como a seguir se ilustra:

```
REPETIR PARA num=1 Até 4000 Passo 1
      Escrever("Introduza a nota do aluno nº ", num)
      Ler(nota(num))
FPara
```

2. DEFINIÇÃO

A solução anteriopr representa uma melhoria extraordinária relativamente à primeira solução. Felizmente a generalidade das linguagens de programação fornece este tipo de dados, chamado vector[1].

Vector pode-se então definir como um conjunto de tamanho fixo de elementos do mesmo tipo ocupando posições contíguas. Antes de se utilizar um vector, deve-se proceder à sua declaração, cuja sintaxe é:

```
DIM nome_vector( inicio ATÉ fim )
```

Em que:

nome_vector é o nome do vector (escolhido pelo programador)

inicio é o valor início do índice

fim é o valor máximo do índice

O número de posições do vector é igual a *fim* menos *início* mais *um*, não sendo obrigatório preencher todas as posições com valores.

Por exemplo,

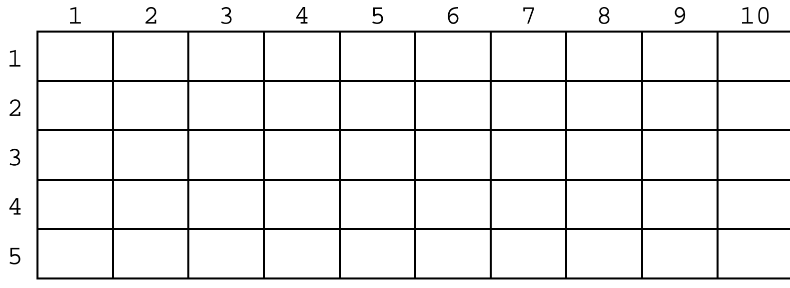
```
DIM notas(1 até 20)
```

Define um vector unidimensional chamado *notas* com 20 posições numeradas de 1 a 20.

Outro exemplo, mas de um vector bidimensional,

```
DIM ecra(1 até 5, 1 até 10)
```

Define um vector bidimensional chamado *ecra* com 5 posições para primeira dimensão e 10 posições para cada uma das 5 posições da primeira dimensão. Vectores bidimensionais são usados frequentemente para representar matrizes matemáticas. Visualizando:



ISEP

[1] Neste texto é usado o também o termo array para significar vector.

Um vector pode ter as dimensões que se pretenderem, fazendo-se a sua separação por vírgulas.

Exemplo 1: Elabore um programa que após determinar se um dado valor existe, ou não, num vector escreva a respectiva mensagem.

Resolução: As estruturas de dados são o vector e o valor a procurar. Em primeiro lugar leem-se valores para o vector, em seguida faz-se a pesquisa. A pesquisa consiste em percorrer o vector até encontrar o valor, se ele existir, ou até se chegar ao fim do vector. Como só faz sentido perguntar se o valor foi encontrado no caso de ainda não se ter chegado ao final do vector, a ordem das condições deverá ser trocada. A condição do ciclo que permite percorrer o vector será então:

não se chegou ao fim do vector E elemento do vector <> valor procurado

Quando uma destas condições for falsa, o ciclo terminará.

```
INÍCIO
DIM val( 1 to 100)
Escrever("Quantos valores?")
Ler(n)
[Leitura de valores para o vector val]
Repetir Para i=1 até n
   Escrever("Valor?")
   Ler(val(i))
FPara
Escrever("Qual o valor que pretende procurar?")
Ler(procurado)
[Pesquisa de procurado]
i=1
Repetir Enquanto i<=n E val(i)<>procurado
   i=i+1
FEnq
Se i<=n
Então    Escrever(procurado, " existe!")
Senão    Escrever(procurado, " Não existe!")
FSe
FIM
```

ISEP

3. ORDENAÇÃO E PESQUISA

A ordenação de vectores e a pesquisa de um dado elemento num vector, são operações muito comuns em programação. Por uma questão de simplificação serão utilizados vectores de números. No entanto estes métodos poder-se-iam adaptar facilmente a vectores de outro tipo de dados.

3.1 ORDENAÇÃO POR SELECÇÃO

O algoritmo do método de ordenação por selecção consiste em seleccionar repetidamente o menor elemento dos que ainda não foram tratados, daí o nome do método. Pretendendo-se uma ordenação por ordem crescente, primeiro selecciona-se o menor elemento do vector e faz-se a sua troca com o elemento

na primeira posição do vector, em seguida selecciona-se o segundo menor elemento e faz-se a sua troca com o elemento na segunda posição do vector, repetindo-se o processo até que todo o vector fique ordenado. Versão BASIC deste método onde *vec* é o vector a ordenar e *num_elem* o número de elementos do vector. Este método é bastante eficiente para pequenos e médios arrays.

```
REM Ordenação por selecção
INPUT "Qual o número de elementos do vector", num_elem
DIM vec(num_elem)
REM Vector definido com num_elem elementos, de 0 a num_elem-1
FOR i=0 to num_elem-2
   FOR j=i+1 to num_elem-1
         If vec(j)<vec(i) Then
               temp=vec(i)
               vec(i)=vec(i)
               vec(i)=temp
         End If
   NEXT j
NEXT i
END
```

3.2 ORDENAÇÃO POR INSERÇÃO

Na ordenação por inserção considera-se um novo elemento de cada vez. Este é inserido no seu devido lugar entre os elementos já considerados, tendo-se o cuidado de os manter ordenados. Em seguida é apresentado o seu algoritmo, no qual se considera que o vector se inicia na posição zero.

```
0. Considerar como parâmetros: o vector, o novo elemento e uma variável que
indique o número de elementos do vector.
1. Testar se vector existe. Isto ,é se o seu número de elementos é maior ou
igual que zero.
1.1      Testar se vector vazio.
1.1.1    Se sim então colocar o novo elemento na primeira posição.
1.1.2    Se não, puxar todos os elementos maiores que novo uma posição
acima e colocar o novo elemento na sua posição. Isto é:
1.1.2.1  Atribuir à posição índice i o elemento da posição índice i-1.
1.1.2.2  Repetir 1.1.2.1 Enquanto não percorrer todo o vector (i>0) e o
novo elemento fôr menor que o elemento na posição índice i.
1.1.2.3  Colocar o novo elemento na posição índice i.
1.2      Actualizar o número de elementos do vector.
2. Fim
```

3.3 PESQUISA SEQUENCIAL

Trata-se do método mais simples de pesquisa, e que consiste em pesquisar sequencial e exaustivamente um vector na procura de um dado valor. O extracto de um programa em BASIC, que executasse este método poderia ter o seguinte aspecto, considerando que *vec* é o vector a pesquisar, *valor* o valor procurado e *num_elem* o número de elementos do vector:

```
...
DIM vec(num_elem)
...
```

ISEP

```
i=0
DO WHILE i<num_elem AND vec(i)<>valor
   i=i+1
LOOP
IF vec(i)=valor THEN
   PRINT "Valor EXISTE no vector"
ELSE
   PRINT "Valor NÃO existe no vector"
END IF
...
```

4. EXERCÍCIOS

Exercício 1: Escreva um programa que permita determinar o maior valor (máximo) existente num vector.

Exercício 2: Como sabe, podemos calcular a massa atómica (peso atómico) de um elemento, MA_e, a partir da massa atómica e abundância relativa de cada um dos seus isótopos, usando a seguinte expressão:

$$MA_e = \frac{\sum_{i=1}^{n} (MA_i * P_i)}{100}$$

MA_e – massa atómica do elemento *e*

MA_i – massa atómica de cada isótopo

P_i – abundância relativa de cada isótopo (1-100)

Em que *i* varia entre 1 e o número de isótopos, *n*.

a) Desenvolva o algoritmo duma sub rotina que realize o somatório do numerador e devolva o resultado. O algoritmo terá como parâmetros: dois vectores (um contendo os MA_i e o outro os P_i) e o seu número de elementos (isótopos), *n*.

b) Escreva um algorimto que calcule o peso atómico de um elemento, MA_e. Para tal, utilize o sub algoritmo que desenvolveu na alínea anterior.

Exercício 3: Pretende-se calcular o mínimo múltiplo comum de dois números inteiros quaisquer.

a) Escreva um sub algoritmo que determine e devolva o mínimo múltiplo comum de dois números dados como argumentos.

b) Desenvolva um programa que, utilizando o sub algoritmo que desenvolveu na alínea anterior, calcule o mínimo múltiplo comum de dois números inteiros introduzidos pelo utilizador.

Exercício 4: Suponha que as notas dos alunos de duas turmas são lidas para dois vectores, um para cada turma. Considere que as notas foram inseridas em ambos os vectores ordenadamente, da menor para a maior. Escreva um programa que "funda" ordenadamente os dois vectores de notas num terceiro.

ISEP