

UNIVERSIDADE FERNANDO PESSOA



PROGRAMAÇÃO I
INTRODUÇÃO À ALGORITMIA E ESTRUTURAS DE DADOS

José Vasconcelos

Luís Paulo Reis

Engenharia do Ambiente

Engenharia Civil

Engenharia Informática

Engenharia da Qualidade

1º ANO

Março 2002

Índice

1. ALGORITMOS E A RESOLUÇÃO DE PROBLEMAS	3
1.1. RESOLUÇÃO DE PROBLEMAS	3
1.2. APROXIMAÇÃO DESCENDENTE (<i>TOP-DOWN APPROACH</i>)	4
1.3. NOÇÃO FORMAL DE ALGORITMO.....	6
1.4. CARACTERÍSTICAS DE UM ALGORITMO	6
2. ESTRUTURAS DE DADOS.....	7
2.1. ESTRUTURAS DE DADOS PRIMITIVAS.....	8
2.1.1. <i>Tipo de dados booleano</i>	8
2.1.2. <i>Tipo de dados numérico</i>	8
2.1.3. <i>Tipo de dados alfanumérico</i>	9
2.1.4. <i>Representação dos dados</i>	9
2.2. ESTRUTURAS DE DADOS NÃO PRIMITIVAS	12
2.2.1. <i>Vectores</i>	12
2.2.2. <i>Matrizes</i>	13
3. NOTAÇÃO ALGORÍTMICA.....	14
3.1. PSEUDOCÓDIGO	14
3.1.1. <i>Instrução de atribuição</i>	15
3.1.2. <i>Leitura e escrita de dados</i>	16
3.1.3. <i>Estrutura condicional</i>	17
3.1.4. <i>Instruções de repetição</i>	18
3.1.5. <i>Operações e Expressões Aritméticas</i>	21
3.1.6. <i>Operadores e operações relacionais</i>	21
3.1.7. <i>Operadores e operações lógicas</i>	22
3.2. ALGORITMOS PROPOSTOS	23
3.3. DIAGRAMAS DE FLUXO (FLUXOGRAMAS).....	30
3.3.1. <i>Notação gráfica</i>	30
3.3.2. <i>Fluxogramas vs. Estruturas de controlo</i>	30
3.3.3. <i>Algoritmo em pseudocódigo / Fluxograma</i>	33
4. PROVA E TESTE DE ALGORITMOS.....	34

Algoritmos e a Resolução de Problemas

A programação de computadores é uma disciplina na área das ciências da computação e refere-se essencialmente ao estudo de estruturas de dados e a sua manipulação para a resolução de problemas em diversos domínios do conhecimento.

Um programa de computador envolve a definição de um algoritmo para a resolução de um problema. Um algoritmo é representado através de expressões simbólicas de modo a descrever e a encontrar a solução de problemas do mundo real.

Um algoritmo representa uma sequência finita e não ambígua de instruções elementares bem definidas, conducente à solução de um determinado problema, cada uma das quais pode ser executada mecanicamente numa quantidade finita de tempo.

As estruturas de dados representam de modo simbólico entidades e objectos do mundo real e definem a parte estática de um algoritmo. A manipulação das estruturas de dados através de declarações e instruções precisas de controlo definem a parte dinâmica de um algoritmo. Este conjunto de estruturas de dados e de controlo constituem formalmente um algoritmo para a resolução de problemas.

1.1. Resolução de Problemas

A resolução de um problema através de um algoritmo e consequente programa computacional refere-se ao processo de identificar e analisar um problema do mundo real e desenvolver a sua solução de modo eficiente. Este processo é constituído pelas seguintes fases: (1) identificação e compreensão do problema (e objectivos), (2) conceptualização da solução, (3) definição do algoritmo para a resolução do problema, e (4) implementação (codificação) da solução através de um programa computacional.

A tarefa de escrever um algoritmo pode ser simplificada através da decomposição e análise de subproblemas. O processo de estruturação na resolução de problemas reflecte-se num programa modular constituído por diferentes partes que definem a solução do problema (figura 1).

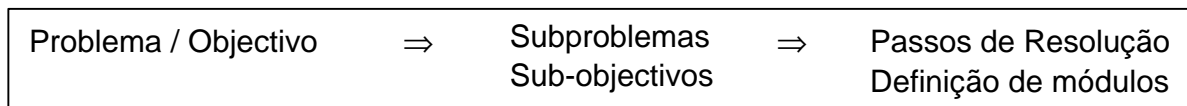


Figura 1: Abordagem para a resolução de problemas

1.2. Aproximação descendente (*top-down approach*)

A aproximação descendente (*top-down*) para a resolução de problemas do mundo real permite raciocinar e estruturar a solução de um problema em linguagem natural (ex.: Português). Este tipo de abordagem facilita o processo de compreensão do problema, assim como a conceptualização do problema/objectivo em subproblemas/sub-objectivos e respectiva solução.

Os próximos exemplos (1 e 2) ilustram dois problemas simples e a notação

Exemplo 1: Substituir uma lâmpada fundida de um candeeiro.

Algoritmo Mudança de Lâmpada

Passo	Descrição
1	Selecione uma nova lâmpada
2	Remova a Lâmpada fundida
3	Insira uma nova lâmpada

1.1	Selecione uma lâmpada da mesma potência da fundida
2.1	Posicione a escada em baixo do candeeiro
2.2	Suba a escada até que possa atingir a lâmpada
2.3	Rode a lâmpada no sentido contrário ao dos ponteiros do relógio até que se solte
3.1	Coloque a nova lâmpada no orifício correspondente
3.2	Rode a lâmpada no sentido dos ponteiros do relógio até que fique presa
3.3	Desça da escada

Definição mais precisa para o passo 1.1

Selecione uma lâmpada candidata à substituição

Se a lâmpada não é da mesma potência da antiga, então **repita** até encontrar uma correcta:

Pouse a lâmpada seleccionada

Selecione uma nova lâmpada

- Por exemplo, para os passos 2.2, 2.3, 3.2 e 3.3 poderiam também derivadas descrições mais precisas e detalhadas, do tipo: Repita ... até ...
- Aumento do detalhe do algoritmo pode continuar quase indefinidamente.
- Grau de detalhe depende das necessidades do agente que vai executar o algoritmo.

Exemplo 2: Encontrar o número de telefone que corresponde a um dado nome numa lista telefónica.

Algoritmo Lista-Telefónica

Passo	Descrição
1	Encontre a página da lista que contém o último apelido do nome
2	Encontre na página determinada no passo 1, o nome procurado

Aumentando o detalhe, obtemos instruções elementares não ambíguas:

1.1	Coloque o marcador D (dedo) ao acaso na lista
1.2	Abra a lista
1.3	Último apelido está contido numa das páginas (esquerda ou direita)? Se sim, siga para o passo 2
1.4	Último apelido precede a página esquerda? Se sim, coloque o marcador atrás da página esquerda; se não, coloque o marcador à frente da página direita.
1.5	Vá para 1.2 (retome a sequência de instruções no passo 1.2)

Eliminando formulações mal definidas (ex.: coloque o marcador atrás):

1.1.1	Torne A igual ao apelido do nome a seleccionar (atribuição à variável A)
1.1.2	Escolha uma posição n ao acaso no intervalo $[1, N]$ (n representa o número de páginas útil da lista)
1.1.3	Torne D igual a n (atribua à variável D o valor n)
1.2	Abra a lista no local seleccionado pelo marcador D
1.3	A está contido numa das páginas (esquerda ou direita)? Se sim, siga para o passo 2.
1.4	A precede o primeiro apelido da página esquerda? Se sim, faça n igual a $(n+1)/2$ (actualização do valor de n); se não, faça n igual a $(N+n)/2$.
1.5	Vá para 1.2 (retome a sequência de instruções no passo 1.2)

1.3. Noção formal de algoritmo

Um algoritmo é um processo discreto (sequência de acções indivisíveis) e determinístico (para cada passo da sequência e para cada conjunto válido de dados, corresponde uma e uma só acção) que termina quaisquer que sejam os dados iniciais (pertencentes a conjuntos pré-definidos).

Um algoritmo é constituído por um conjunto de expressões simbólicas que representam acções (escolher, atribuir, etc.), testes de condições (estruturas condicionais) e estruturas de controlo (ciclos e saltos na estrutura sequencial do algoritmo) de modo a especificar o problema e respectiva solução.

1.4. Características de um Algoritmo

Entradas

Quantidades inicialmente especificadas (por exemplo, através de instruções de leitura).

Saídas

Uma ou mais saídas (habitualmente por instruções de escrita)

Finitude

A execução deve terminar sempre num número finito de passos.

Precisão

Todos os passos do algoritmo devem ter um significado preciso não ambíguo, especificando exactamente o que deve ser feito. Para evitar a ambiguidade das linguagens humanas (linguagens naturais), linguagens especiais (denominadas linguagens de programação) foram criadas para exprimir algoritmos.

Eficácia

Os passos devem conduzir à resolução do problema proposto. Devem ainda ser executáveis numa quantidade finita de tempo e com uma quantidade finita de esforço.

Eficiência

Em muitos casos colocam-se questões de eficiência a um algoritmo.

2. Estruturas de Dados

A resolução de problemas através de algoritmos requer a representação de entidades e objectos reais em itens de dados. As diferentes formas nas quais os itens de dados são logicamente relacionados definem diferentes estruturas de dados (figura 2).

Esta secção apresenta um conjunto de estruturas de dados classificadas como primitivas e não primitivas. Estruturas de dados primitivas são directamente manipuladas em linguagem máquina (binária), enquanto que estruturas de dados não primitivas (ou complexas) representam estruturas de informação em conjuntos (formados por estruturas de dados primitivas) logicamente relacionados.

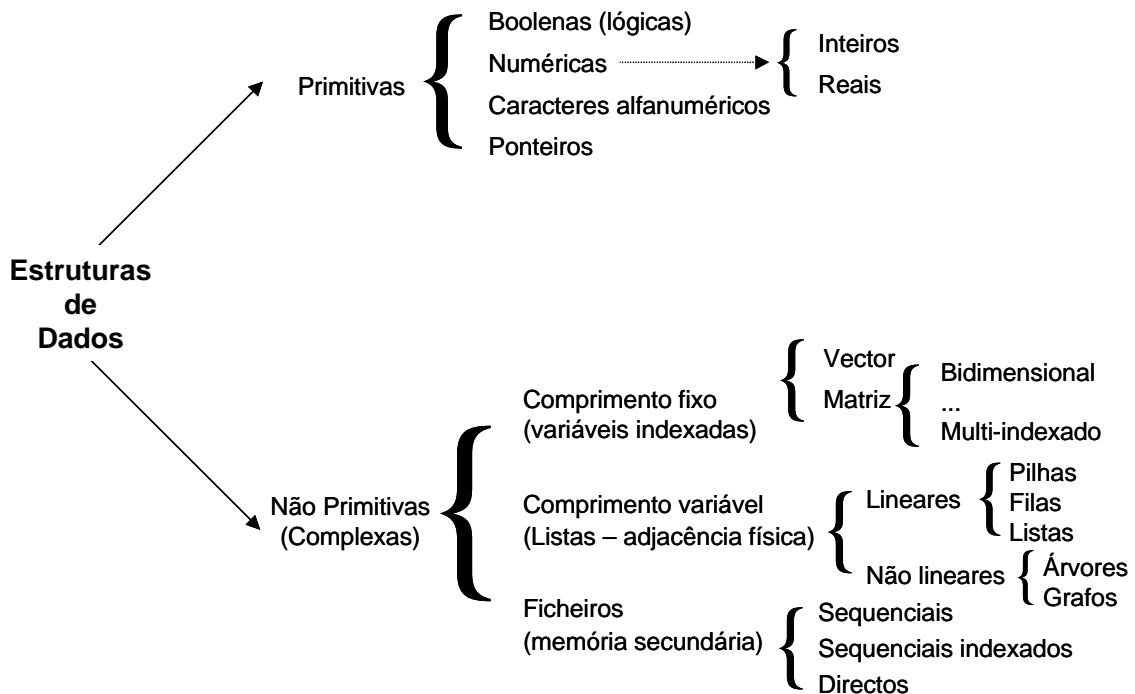


Figura 2: Estruturas de Dados

A representação de uma particular estrutura de dados na memória do computador é designado por estrutura de armazenamento. É fundamental saber distinguir estes dois conceitos: estrutura de dados vs. estrutura de armazenamento. Existem diferentes configurações de armazenamento correspondentes a uma particular estrutura de dados. A preocupação desta disciplina centra-se na forma como representamos entidades e objectos reais através de estruturas de dados e não na forma como os armazenamos fisicamente na memória (primária ou secundária) do computador.

A definição das estruturas de dados a utilizar por um algoritmo tem também subjacente as operações a executar nos dados representados. Diferentes operações podem ser executadas, tais como operações para criar e eliminar estruturas de dados, operações para inserir, alterar e eliminar elementos da estruturas de dados, e operações para aceder a elementos da estrutura de dados.

Em síntese, uma eficiente manipulação das estruturas de dados associadas à resolução de um problema e conseqüente construção de um algoritmo envolve uma análise das seguintes questões:

- a) Compreender a relação entre os dados
- b) Decidir operações a executar nos dados logicamente relacionados
- c) Representar os elementos dos dados de modo a:
 - i. Manter as relações lógicas entre os dados
 - ii. Executar de forma eficiente as operações nos dados
- d) Construir o algoritmo e escolher a linguagem de programação mais apropriada que permita de modo 'natural' e 'expressivo' representar as operações a executar nos dados.

2.1. Estruturas de dados primitivas

Esta secção apresenta em conjunto de exemplos simples representativos de tipos de estruturas de dados primitivos: tipo *booleano* (ou binário), tipo numérico (inteiros e reais) e tipo alfanumérico para o tratamento de cadeias de caracteres e conseqüente tratamento de texto.

2.1.1. Tipo de dados booleano

Este tipo de estrutura de dados permite representar dois (e só dois) estados: 'verdadeiro' e 'falso'. Este tipo pode ser representado através dos dois estados existentes na codificação binário: 1 – 'verdadeiro'; 0 – 'falso'. Este tipo de dados é usualmente aplicado em situações reais que unicamente denotam dois estados possíveis.

2.1.2. Tipo de dados numérico

Tipo de dados representativo de valores numéricos no domínio dos números inteiros e reais. Por exemplo, 37 é um dado do tipo numérico **inteiro**. $\sqrt{2}$ (raiz quadrada de 2) é um dado do tipo **real**.

2.1.3. Tipo de dados alfanumérico

Um sistema de codificação designado por ASCII (*American Standard Code for Information Interchange*) foi criado para representar informação do tipo 'character'. O conjunto de caracteres ASCII permite representar:

- 1) Alfabeto (letras minúsculas e maiúsculas) {a,c,d,e,f,g,h, ...,z, A,B,...,Z}
- 2) Caracteres numéricos decimais {0,1,2,3,4,5,6,7,8,9}
- 3) Operadores e caracteres especiais {+,-,*,/,>,<,#,\$,% ,..., @,&,(,),...}
- 4) Caracteres de controlo. Por exemplo, *DEL – Delete, CR – Carriage Return, HT- Horizontal Tab*, etc.

2.1.4. Representação dos dados

Os tipos de dados referidos anteriormente são representados na memória principal do computador de diferentes formas. Os sistemas de computação digital utilizam vulgarmente três tipos de dados:

- Dados simbólicos
- Dados Numéricos
- Informação Contextual

Representação Simbólica

Símbolos	Letras	(a a z, A a Z)
	Dígitos Decimais	(0 a 9)
	Sinais de Operação	+ - * /
	Sinais de Pontuação	, . ; : ? !
	Símbolos Especiais	“ # & % \$ ‘

Os símbolos são codificados em binário com um código de comprimento n. Os códigos usuais têm 8 bits (n=8, 256 símbolos representáveis), por exemplo o ASCII ou EBCDIC.

Representação Numérica

Os dados representam quantidades numéricas no sistema de numeração binário (ou num outro sistema codificado em binário).

0101 1011 =

$$= 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 64 + 16 + 8 + 2 + 1 = 91$$

Informação Contextual

Conteúdo informativo (significado) de um dado, depende do contexto. Por exemplo, o dado 0100 0001, pode representar:

65	Numa operação utilizando binário
41	Numa operação utilizando BCD (“Binary Coded Decimal”)
A	Símbolo (letra) ASCII
MOV A,B	Instrução do microprocessador INTEL 8080

O contexto pode depender do uso no problema particular, da posição numa sequência de dados ou da posição física no sistema de computação.

Tipo de dados vs. Variáveis vs. Constantes

A compreensão dos conceitos anteriores é de fundamental importância para a eficiente construção de algoritmos. Os seguintes exemplos descrevem a diferença entre estes conceitos e o relacionamento lógico entre eles.

Exemplo 3

$X \leftarrow 47$

Esta expressão algorítmica simples significa que à variável `X` foi atribuído o valor (ou constante) 47. Esta operação de atribuição tem por pressuposto (em termos algorítmicos) o facto que `X` é uma variável do tipo inteiro.

Exemplo 4

$CAR \leftarrow 'G'$

Esta expressão algorítmica simples significa que à variável `CAR` foi atribuído o valor (ou constante) 'G'. Esta operação de atribuição tem por pressuposto (em termos algorítmicos) o facto que `CAR` é uma variável do tipo alfanumérico. Por convenção os valores alfanuméricos são apresentadas entre plicas (ex.: $CAR1 \leftarrow '7'$ ou $CAR2 \leftarrow 'J'$).

Nomes de variáveis

Uma variável é uma entidade que representa um determinado valor e o seu nome é escolhido de forma a reflectir o valor que representa (ex.: a variável MAX pode representar o valor máximo de um conjunto de valores numéricos).

Por convenção, um nome de uma variável não deve conter espaços, não deve iniciar por números e não deve conter determinados caracteres especiais, tais como * , = ? , / * ; : . , } [] { , etc.

O nome de uma variável deve iniciar sempre por uma letra, seguido de um conjunto de caracteres incluindo letras, números e alguns caracteres especiais, como é o caso do carácter '_' que pode ser utilizado para separar palavras que constituem o nome da variável (ex.: NUM_ALUNO).

As próximas variáveis são consideradas válidas:

NUM

N_ALUNO

NOMEALUNO

NOME_ALUNO

X1

Y2

As próximas variáveis são consideradas inválidas:

7NUM

Y-Z

T/4

U*VAR

DUAS-PAL

DUAS PALAVRAS

2.2. Estruturas de dados não primitivas

As estruturas de dados não primitivas são definidas através de conjuntos de estruturas de dados primitivas. Conforme a figura 2, existem estruturas de comprimento fixo e de comprimento variável a nível da memória principal (ou primária) do computador, e existem estruturas de dados na forma de ficheiros de dados que são manipuladas na memória secundária (ou permanente) do computador.

No contexto desta disciplina de introdução à programação, serão estudadas as estruturas de dados de comprimento fixo (vectores e matrizes) e alguns exemplos de ficheiros de dados e suas aplicações.

2.2.1. Vectores

Elementos de um vector (*array*) são representados através de conjuntos de variáveis de um determinado tipo de dados. Um vector contém um nome ao qual está associado um tipo de dados em função dos dados a manipular pelo vector, um índice do tipo inteiro, e uma dimensão do tipo inteiro. Um vector é apresentado através de um nome e de um índice entre parêntesis rectos. A figura seguinte ilustra a representação e a manipulação dos elementos de um vector.

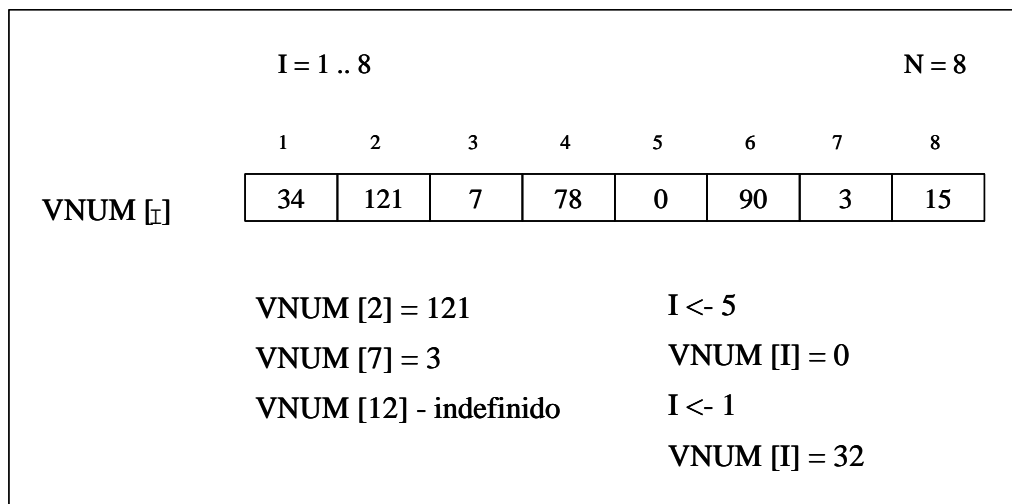


Figura 3: Representação e manipulação de vectores

O vector `VNUM [I]` representado na figura anterior é um vector do tipo numérico (inteiro) com uma dimensão de 8 (variável `N`) elementos. O acesso a elementos (valores) do vector são efectuados através da utilização de um índice (no exemplo anterior representado na variável `I`). Por exemplo, o valor associado à posição 6 do vector `VNUM` é igual a 90.

2.2.2. Matrizes

Uma matriz pode ser interpretada como um vector (*array*) bidimensional (com dimensão 2). A próxima figura ilustra como representar e manipular elementos de uma matriz. Uma matriz é representada através de um nome e dois índices que permitem indexar e aceder os elementos que constam na matriz.

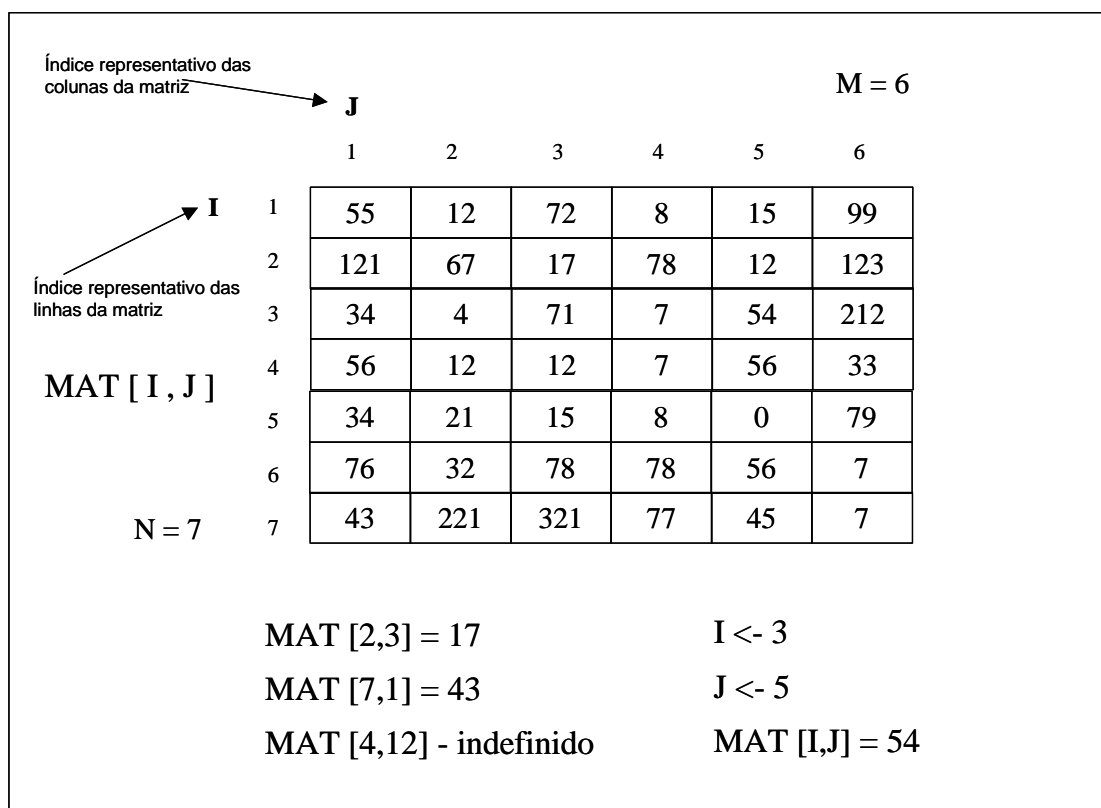


Figura 4: Representação e manipulação de matrizes

A matriz $MAT[I, J]$ ilustrada na figura anterior contém 7 linhas e 6 colunas. O índice I representa as linhas e o índice J as colunas da referida matriz. A inserção, acesso e a actualização dos elementos da matriz efectua-se através da referência da linha e da coluna. Por exemplo, $MAT[3, 6] = 212$.

Este tipo de estruturas de dados (vectores e matrizes) é aplicado com notas explicativas em diversos exercícios e exemplos propostos nas secções do próximo capítulo.

3. Notação algorítmica

Esta secção apresenta a notação algorítmica a utilizar na construção de algoritmos para a resolução de problemas (exercícios propostos). Na fase de desenvolvimento de um algoritmo podemos utilizar:

- Linguagem auxiliar (pseudo-código) – linguagem de representação de alto nível que pode ser traduzida em qualquer linguagem de programação.
- Diagrama de Fluxo (fluxograma) - notação gráfica.
- Linguagens de Programação (ex.: C, C++, Java, Pascal, Basic, Fortran, ADA, etc.).

3.1. Pseudocódigo

O pseudocódigo é uma notação algorítmica muito próxima da linguagem natural.

Um algoritmo é identificado por um nome que deverá ser elucidativo do problema em causa. Por convenção, o nome do algoritmo é seguido por uma breve descrição (comentário introdutório) das tarefas executadas pelo algoritmo. Esta descrição poderá também conter os nomes e tipos das variáveis utilizadas no algoritmo.

Um algoritmo é construído através de uma sequência de passos numerados (exemplo 5). Cada um destes passos deverá conter um comentário explicativo da tarefa a executar no contexto global do algoritmo.

O próximo exemplo apresenta um algoritmo que permite calcular o factorial de um número inteiro. Este algoritmo utiliza uma notação que será utilizada nos restantes exemplos:

1. Nome do algoritmo em letras maiúsculas.
2. Breve comentário das tarefas a desenvolver pelo algoritmo.
3. Conjunto de passos numerados e comentados.
4. Estruturas de dados, funções e subrotinas em letra maiúscula.
5. Estruturas de controlo: primeira letra em maiúscula.
6. O algoritmo termina com a instrução `Exit` (fim lógico). O símbolo representa o fim físico do algoritmo.

As próximas subsecções apresentam instruções, estruturas de controlo, operações e operadores (notação algorítmica) que podem ser utilizadas na construção de algoritmos.

Exemplo 5

Algoritmo FACTORIAL. Este algoritmo calcula o factorial de um número inteiro.

1. [Leitura do número]
 Read (N)
2. [Caso particular de $N = 0$ ou $N = 1$]
 If $N=0$ Or $N=1$
 Then FACT \leftarrow 1
3. [Outros números]
 Else FACT \leftarrow N
 NUM \leftarrow N - 1
 Do For I = NUM To 1 Step -1
 FACT \leftarrow FACT * I
4. [Impressão do resultado (N!)]
 Print ('O Factorial de ',N,' é igual a ',FACT)
5. [Termina]
 Exit

**3.1.1. Instrução de atribuição**

A instrução de atribuição é representada através do símbolo \leftarrow que é colocado à direita da variável que recebe o valor da atribuição (ex.: FACT \leftarrow 1). Ter atenção a diferença entre o sinal de atribuição \leftarrow e o símbolo = que é utilizado como operador relacional.

Numa instrução de atribuição pode constar variáveis, constantes, operadores e outras funções matemáticas como indica o próximo exemplo.

Exemplo 6

VAR \leftarrow 12 / 7

FACT \leftarrow FACT * NUM

N \leftarrow MOD (NUM1,NUM2)

3.1.2. Leitura e escrita de dados

Perante a notação algorítmica é possível obter (ou ler) valores de variáveis, assim como escrever (ou imprimir) os valores dessas variáveis através de instruções de leitura (entrada de dados) e instruções de escrita (saída de dados).

Leitura de dados

Sintaxe:

Read (<nome da variável>)

Exemplo 7

Read (N_ALUNO)

Exemplo 8

Read (N_ALUNO)

Do For I = 1 to N_ALUNO

 Read (NOME_ALUNO [I])

Saída de dados

Sintaxe:

Write (<nome da variável>) ou

Write (<'texto'>) ou

Write (<nome da variável>, 'texto') ou

Write ('texto', <nome da variável 1>, 'texto' <nome da variável 2>, 'texto', ...)

Exemplo 9

Write (N_ALUNO)

Write ('só texto ...')

Write ('O número de alunos é igual a ', N_ALUNO)

Write ('O número ',N_ALUNO,' refere-se ao n.º de alunos da disciplina de programação')

Exemplo 10

Read (N)

Do For I = 1 to N

 Write (NOME_ALUNO [I])

3.1.3. Estrutura condicional

If Statement – If ... Then ... Else (Se ... Então ... Senão)

Esta declaração representa um teste de condição lógica (se ... então ... senão) na qual é executado um conjunto de instruções consoante a condição especificada for verdadeira ou falsa. A declaração `IF` pode ter uma das seguintes formas:

1. ***If Condition***

Then _____

 ...

2. ***If Condition***

Then _____

 ...
Else _____

 ...

A seguir à clausula `Then` um conjunto de instruções é executado no caso da condição ser verdadeira. Caso contrário (`Else – Senão`), será executado o conjunto de instruções a seguir à clausula `Else`.

Exemplo 11

```
If N > 10
Then X ← (X + 15) / 2
    Print (X)
```

Exemplo 12

```
If N > 10
Then X ← (X + 15) / 2
    Print (X)
Else X ← X * 5
    Print (X)
```

A declaração (If ... Then ... Else) pode conter um conjunto de instruções devidamente indentadas (exemplo 9) de modo a definir um conjunto mais extenso de estruturas condicionais.

Exemplo 13

```

Read (X,Y)
Then X ← (X + 15) / 2
    If X > Y
    Then Print (X - Y)
    Else Print (Y - X)
Else X ← X * 5
    If X > Y/2
    Then Print (X)
        Print (X/Y)
    Else Print (X*Y)

```

3.1.4. Instruções de repetição

Existem três tipos de instruções que permitem controlar iterações ou ciclos de processamento. As instruções de repetição podem ter uma das seguintes formas:

1. Do While <logical condition>
2. Repeat until <logical condition>
3. Do For INDEX = <numerical sequence>

Do While (Fazer enquanto ...)

A instrução `Do While` é utilizada quando é necessário repetir um conjunto de passos em função de uma determinada expressão lógica. Os referidos passos são repetidos enquanto a expressão lógica for verdadeira, ou seja, o ciclo de processamento termina quando a condição se tornar falsa.

Sintaxe:

Do While <logical condition>

```

_____
_____
...

```

Exemplo 14

Read (Z)

$X \leftarrow 15$

Do While $Z \geq 135$

$X \leftarrow X - 5$

$Z \leftarrow (X * 4) / 3$

Print (X,Z)

Repeat Until (Repetir até ...)

A instrução `Repeat Until` é utilizada quando é necessário repetir um conjunto de passos em função de uma determinada expressão lógica. Os referidos passos são repetidos até a expressão lógica se tornar verdadeira, ou seja, o ciclo de processamento termina quando a condição se tornar verdadeira.

Sintaxe:

Repeat Until *<logical condition>*

```
_____
_____
...
```

Exemplo 15

1 [Inicializações]

$K \leftarrow 1$

2. [Repetir até atingir o aluno 75]

Repeat Until $K > 75$

Write (NOME_ALUNO [K])

$K \leftarrow K + 1$

Do For (Fazer para ...)

A instrução Do For é utilizada quando é necessário repetir um conjunto de passos um determinado número de vezes.

Sintaxe:

Do For INDEX = N1 to N2 Step P

```
_____
_____
...
```

INDEX indica o índice que é incrementado em cada ciclo de processamento. N1 e N2 referem dois números inteiros representativos do intervalo inferior e superior da sequência de números a executar. A cláusula Step determina o incremento na sequência numérica. Por defeito, o Step tem valor 1.

Exemplo 16

X ← 10

Do For I = 1 to 70

 X ← X + 5

Print (X)

Exemplo 17

Read (Y)

Do For J = 10 to 100 Step 10

 Y ← Y + 25

 Print (Y)

Exemplo 18 $X \leftarrow 0$

Do For I = 150 to 5 Step -5

Read (A[I])

 $X \leftarrow X + A[I]$

Write (X)

3.1.5. Operações e Expressões Aritméticas

A notação algorítmica inclui operações e funções matemáticas que permitem efectuar variados cálculos aritméticos. Dois tipos de valores numéricos são considerados: reais e inteiros. As regras de precedência têm por base as regras padrão das expressões matemáticas. Deste modo, temos por ordem decrescente, os parêntesis - () - a exponenciação (\uparrow), a multiplicação (*), a divisão (/), a adição (+) e finalmente a subtracção (-).

Determinadas funções matemáticas podem ser utilizadas na definição de expressões computacionais, tais como:

mod(M,N) – função que retorna o resto da divisão de M por N.

int(NUM) – função que retorna o parte inteira de um número real.

sqr(NUM) – função que retorna a raiz quadrada de um número inteiro ou real.

Exemplo 19 $X \leftarrow (A+B) \uparrow 3 - (A-A/3)*B$ $N \leftarrow \text{mod}(20,6)$ $RNUM \leftarrow \text{sqr}(NUM)$ **3.1.6. Operadores e operações relacionais**

Os operadores matemáticos relacionais (=, <, >, ≤, ≥, ≠) têm um conjunto de símbolos correspondentes a nível computacional. A nível computacional temos respectivamente os seguintes operadores relacionais: =, <, >, <=, >=, e <>. As expressões lógicas representam relações entre valores do mesmo tipo de dados.

O resultado da avaliação de uma expressão lógica pode ter um de dois valores possíveis: verdadeiro (*true*) ou falso (*false*).

Exemplo 20

$A \leftarrow 20$ (a variável A tem atribuído o valor numérico 20)

1. $A \leq 10 / 3 + 5$

2. $A <> A + 10$

A relação 1 tem valor *falso* e a relação 2 tem valor *verdadeiro*.

3.1.7. Operadores e operações lógicas

A notação algorítmica também inclui os seguintes operadores lógicos:

<u>Operador</u>	<u>Notação</u>
Negação	not
Conjunção	and
Disjunção	or

Os operadores lógicos são utilizados em conjunto com os operadores relacionais e aritméticos. Deste modo, é considerada a seguinte ordem de precedência entre operadores.

<u>Precedência</u>	<u>Operador</u>
1	Parêntesis
2	Aritmético
3	Relacional
4	Lógico

Considere o seguinte exemplo assumindo que a variável NUM tem o valor 1.

1. $(NUM < 2) \text{ and } (NUM < 0)$

2. $(NUM < 2) \text{ or } (NUM < 0)$

3. $\text{not}(NUM < 2)$

As expressões 1,2,3 têm valores *falso*, *verdadeiro*, e *falso* respectivamente.

A próxima secção propõe um conjunto de algoritmos e respectiva resolução.

3.2. Algoritmos propostos

Algoritmo MAX_DIVISOR_COMUM

Este algoritmo permite calcular o máximo divisor comum – $\text{mdc}(m, n)$ – entre dois números inteiros.

1. [Leitura dos valores (conjunto de valores válidos: inteiros, $\neq 0$)]

```
Read (M,N)
If M < N
Then MIN ← M
Else MIN ← N
```

2. [Inicializações]

```
MDC ← MIN
FLAG ← ∅
Do While FLAG = ∅
    R1 ← mod(M,MDC)
    R2 ← mod(N,MDC)
    If R1 = ∅ and R2 = ∅
    Then FLAG ← 1
    Else MDC ← MDC - 1
```

3. [Escrever o valor do máximo divisor comum]

```
Write('O máximo divisor comum de 'M,'e',N, 'é igual a ',MDC)
```

4. [Termina]

```
Exit
```



Algoritmo MAIOR_VALOR_VECTOR

Este algoritmo determina o maior valor de um conjunto de valores referenciados no vector A. O vector A contém N elementos. O resultado é colocado na variável MAX.

1. [Verificar se o vector contém elementos]
 If $N < 1$
 Then Write('Vector vazio')
 Exit
2. [Inicializações: assumir inicialmente A[1] como o maior elemento do vector]
 $MAX \leftarrow A[1]$
 $I \leftarrow 2$
3. [Verificar todos os elementos do vector]
 Do While $I \leq N$
 If $MAX < A[I]$
 Then $MAX \leftarrow A[I]$
 $I \leftarrow I + 1$
3. [Imprimir o maior elemento do vector A]
 Write('O maior elemento do vector é ',MAX)
4. [Termina]
 Exit

**Algoritmo SOMA_MÉDIA**

Este algoritmo determina o somatório e a média de um conjunto de valores referenciados no vector A. O vector A contém N elementos.

1. [Verificar se o vector contém elementos]
 If $N < 1$
 Then Write('Vector vazio')
 Exit
2. [Inicializações]
 $ACUM \leftarrow 0$
 $MED \leftarrow 0$
3. [Verificar todos os elementos do vector]
 Do For $I = 1$ to N
 $ACUM = ACUM + A[I]$
4. [Determina a média de valores do vector]
 $MED \leftarrow ACUM / N$
4. [Imprime os resultados]
 Write('O somatório é igual a ',ACUM,' e a média igual a ',MED)
5. [Termina]
 Exit



Algoritmo DIAGONAL PRINCIPAL

Este algoritmo efectua a leitura de uma matriz quadrada com N linhas e colunas. A matriz é constituída por números inteiros. Determinar os seguintes valores:

- a) O somatório da diagonal principal da matriz.
- b) O valor mínimo da diagonal principal.
- c) O valor máximo da diagonal principal.

1. [Leitura da matriz quadrada]

Read (N)

Do For I = 1 to N

Do For J = 1 to N

Read (M[I,J])

2. [Inicializações]

SOMA \leftarrow \emptyset

MIN \leftarrow M[1,1]

MAX \leftarrow M[1,1]

3. [Tratar os elementos da primeira diagonal da matriz]

Do For I = 1 to N

[determina o somatório]

SOMA \leftarrow SOMA + M[I,I]

[determina o valor mínimo]

If M[I,I] < MIN

Then MIN \leftarrow M[I,I]

[determina o valor máximo]

If M[I,I] > MAX

Then MAX \leftarrow M[I,I]

3. [Imprimir os resultados]

Write('O somatório dos elementos da primeira diagonal da matriz é igual a ',SOMA)

Write('O menor elemento da primeira diagonal da matriz é igual a ',MIN)

Write('O maior elemento da primeira diagonal da matriz é igual a ',MAX)

4. [Termina]

Exit



Pesquisa e ordenação de vectores

a) Ordenação por selecção

Algoritmo ORDENAÇÃO VECTOR

Dado um vector K com N elementos, este algoritmo ordena o vector por ordem crescente.

1. [Seleccção sucessiva dos valores até ao índice N-1]

Do For PASS = 1 To N - 1

MIN ← PASS

2. [Obter o índice do valor mínimo]

Do For I = PASS + 1 To N

If K[I] < K[MIN]

Then MIN ← I

3. [Trocar os valores]

If MIN <> PASS

Then TEMP ← K[PASS]

K[PASS] ← K[MIN]

K[MIN] ← TEMP

4. [Termina]

Exit



A próxima figura ilustra a forma como um determinado vector é ordenado através da selecção e troca sucessiva de valores. Considere o seguinte vector $K[]$ de números inteiros e as respectivas passagens até obter a ordenação dos seus elementos.

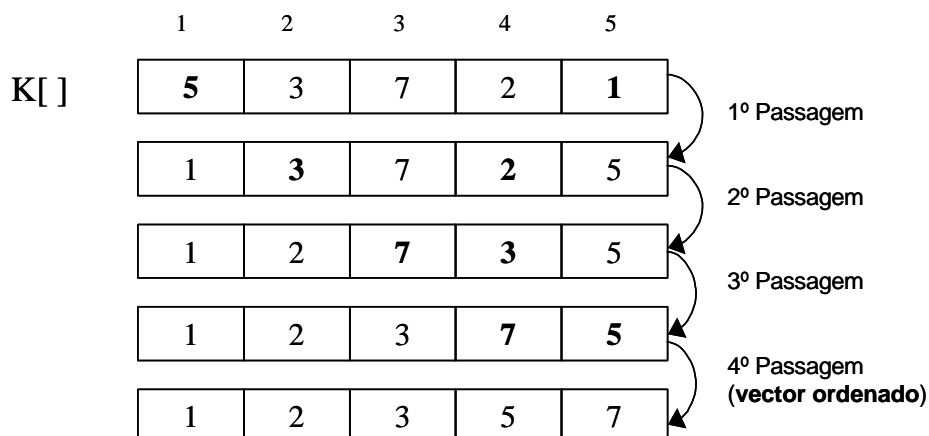


Figura 5: Ordenação dos elementos de um vector por selecção

b) Pesquisa linear

Os próximos três algoritmos representam três versões para a resolução do mesmo problema: pesquisar a existência de um dado elemento num determinado vector.

Pesquisa linear de elementos de um vector (versão 1.0)

Algoritmo PESQUISA_LINEAR_V1.0

Dado um vector K com N elementos ($N \geq 1$), este algoritmo permite verificar a existência de um dado elemento (valor) representado pela variável X.

1. [Iniciar a 'bandeira']

FLAG \leftarrow \emptyset

2. [Pesquisar sucessivamente cada elemento do vector]

Do For I = 1 to N

 If K[I] = X

 Then Write ('O valor ',X, ' foi encontrado na posição ',I)

 FLAG \leftarrow 1

3. [Verificar o sucesso da pesquisa]

If FLAG = \emptyset

Then Write('O valor ',X,' não foi encontrado)

4. [Termina]

Exit



NOTA: esta versão é pouco eficiente, na medida em que é necessário efectuar N testes até encontrar um determinado elemento do vector. Os próximos algoritmos (versão 1.1. e 1.2.) apresentam uma forma mais eficiente de resolver este problema.

Algoritmo PESQUISA_LINEAR_V1.1

1. [Inicializações]

$I \leftarrow 1$
 $FLAG \leftarrow \emptyset$

2. [Pesquisar sucessivamente cada elemento do vector]

Do While $I \leq N$ AND $FLAG = \emptyset$
 If $K[I] = X$
 Then Write ('O valor ',X, ' foi encontrado na posição ',I)
 $FLAG \leftarrow 1$
 $I \leftarrow I + 1$

3. [Verificar o sucesso da pesquisa]

If $FLAG = \emptyset$
Then Write('O valor ',X,' não foi encontrado')

4. [Termina]

Exit

**Algoritmo PESQUISA_LINEAR_V1.2**

1. [Inicializações]

$I \leftarrow 1$

2. [Pesquisar sucessivamente cada elemento do vector]

Do While $I \leq N$ AND $K[I] \neq X$
 $I \leftarrow I + 1$

3. [Verificar o sucesso da pesquisa]

If $K[I] = X$
Then Write ('O valor ',X, ' foi encontrado na posição ',I)
Else Write('O valor ',X,' não foi encontrado')

4. [Termina]

Exit



Algoritmo VECTORES_ALUNOS_INFO

Este algoritmo processa a informação relativa a um conjunto de alunos de um determinado estabelecimento de ensino. Este algoritmo efectua os seguintes procedimentos:

- a) Leitura dos vectores representativos do número, nome, curso e turma de N alunos.
 - b) Ordenação dos alunos e respectivos vectores por ordem alfabética (crescente) do nome do aluno.
 - c) Verificar a existência de um determinado aluno. A pesquisa é efectuada pelo número do aluno.
1. [Leitura de valores]
 - Read (N)
 - Do For I = 1 To N
 - Read(NUM[I], NOME[I], CURSO[I], TURMA[I])
 2. [Ordenação dos alunos por nome]
 - Do For PASS = 1 To N - 1
 - MIN ← PASS
 - Do For I = PASS + 1 To N
 - If NOME[I] < NOME[MIN]
 - Then MIN ← I
 - If MIN <> PASS
 - Then TEMP ← NOME[PASS]
 - NOME[PASS] ← NOME[MIN]
 - NOME[MIN] ← TEMP
 - [trocar os outros elementos dos restantes vectores]
 - TEMP ← NUM[PASS]
 - NUM[PASS] ← NUM[MIN]
 - NUM[MIN] ← TEMP
 - TEMP ← CURSO[PASS]
 - CURSO[PASS] ← CURSO[MIN]
 - CURSO[MIN] ← TEMP
 - TEMP ← TURMA[PASS]
 - TURMA[PASS] ← TURMA[MIN]
 - TURMA[MIN] ← TEMP
 2. [Pesquisa por número de aluno]
 - Read(NALUNO)
 - I ← 1
 - Do While I <= N AND NUM[I] <> NALUNO
 - I = I + 1
 - If NUM[I] = NALUNO
 - Then Write('O aluno existe')
 - Else Write('O aluno não existe')
 2. [Termina]
 - Exit




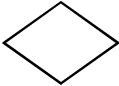
□

3.3. Diagramas de Fluxo (Fluxogramas)

Um diagrama de fluxo representa de uma forma gráfica as instruções e respectivas operações incluídas em determinado algoritmo. O objectivo principal de um diagrama de fluxo (ou fluxograma) é facilitar a compreensão e desenvolvimento de algoritmos para a resolução de problemas através da utilização e composição de símbolos gráficos.

3.3.1. Notação gráfica

De modo a construir um fluxograma é necessário conhecer a notação gráfica a utilizar e respectivo significado. A próxima tabela apresenta os símbolos (e respectivo significado) utilizados na construção de fluxogramas.

Símbolo	Significado
	Início / fim do algoritmo
	Leitura e escrita (entrada e saída) de dados
	Instruções de atribuição
	Estrutura condicional

3.3.2. Fluxogramas vs. Estruturas de controlo

As estruturas de controlo (condicional e ciclos) apresentadas anteriormente podem ser representadas graficamente conforme os seguintes diagramas.

ESTRUTURA If ... Then ... Else

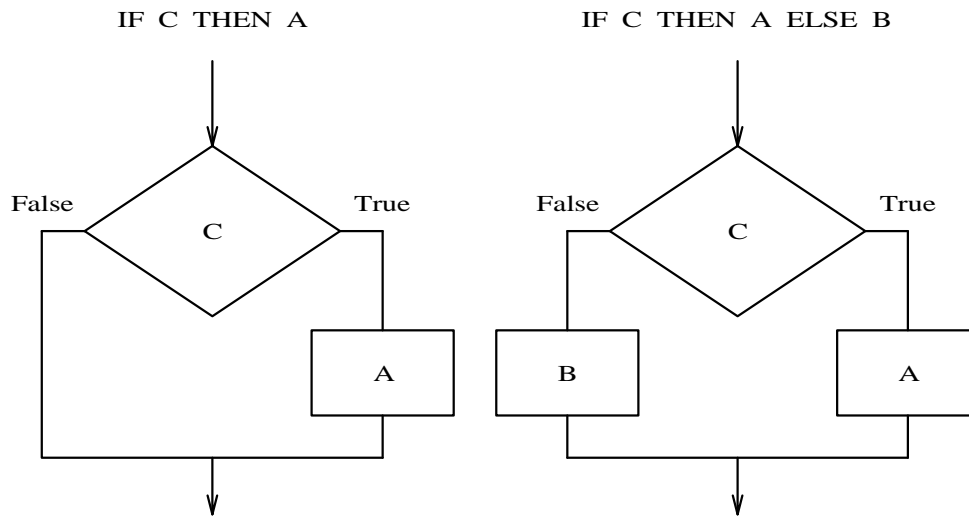


Figura 6: Representação em fluxograma da estrutura If ... Then ... Else

ESTRUTURA Do While

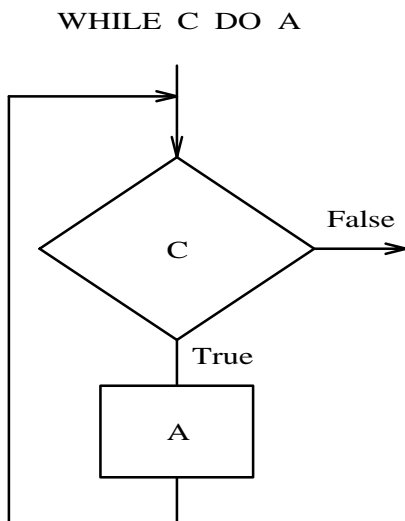


Figura 7: Representação em fluxograma da estrutura de controlo Do While

ESTRUTURA Repeat Until

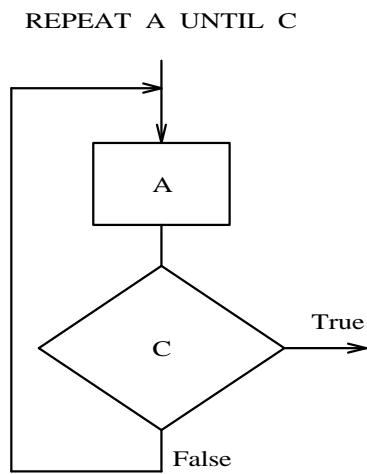


Figura 8: Representação em fluxograma da estrutura de controlo Repeat Until

ESTRUTURA For ... To

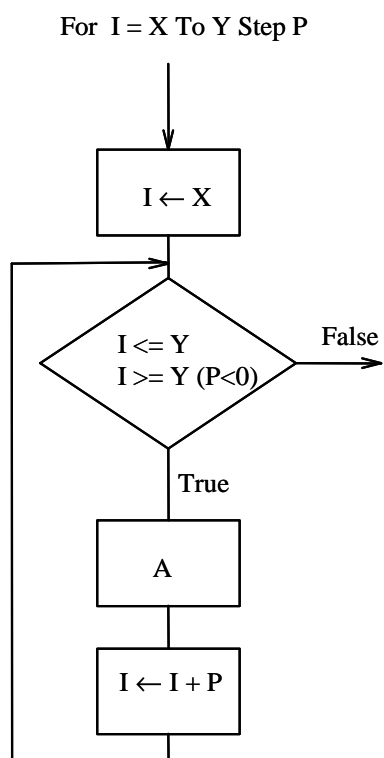


Figura 9: Representação em fluxograma da estrutura de controlo Do For

3.3.3. Algoritmo em pseudocódigo / Fluxograma

Considere os seguinte algoritmo em pseudocódigo e o respectivo fluxograma.

Algoritmo SOMA_VALORES

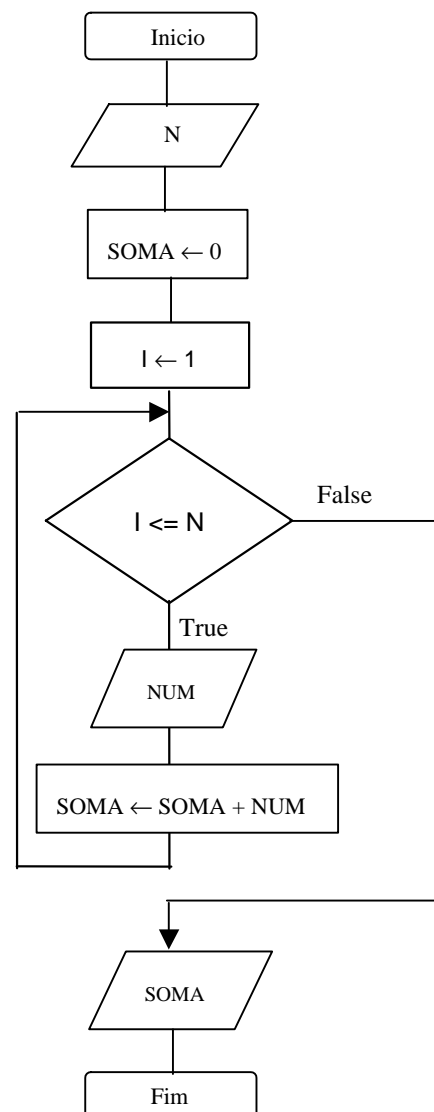
Este algoritmo determina o somatório de um determinado número de valores numéricos a introduzir pelo utilizador.

Pseudocódigo

1. [Leitura de dados (número de valores)]
 Read ('Nº de valores a somar: ', N)
2. [Inicialização]
 SOMA \leftarrow 0
2. [Ciclo para efectuar o somatório]
 Do For I = 1 To N
 Read ('Introduza o Nº:', NUM)
 SOMA \leftarrow SOMA + NUM
3. [Imprimir o resultado]
 Write ('O somatório é igual a ', SOMA)
4. [Termina]
 Exit

□

Fluxograma



4. Prova e Teste de Algoritmos

A prova e teste de algoritmos e programas permite seguir a execução de um algoritmo (ou programa) passo a passo e verificar a evolução de todas as variáveis. Este processo é apresentado através de uma tabela em que deve constar todas as variáveis, testes lógicos efectuados, entrada de dados e saída de resultados.

Seguimento do Algoritmo Máximo Divisor Comum (mdc) para M=8 e N=20

M	N	MIN	mdc	R1	R2	FLAG= Ø	R1=R2	Entrada/Saída
8	20							Leia(8,20)
		8						
			8					
				0				
					4			
			7			V	F	
				1				
					6			
			6			V	F	
				2				
					2			
			5			V	F	
				3				
					0			
			4			V	F	
				0				
					0			
						F	V	Escreve(4)

O seguimento (ou 'traçagem') de algoritmos pode detectar a presença de erros mas não pode comprovar a sua ausência (exceptuando em casos simples).