Apontamentos de Programação em Visual Basic

2004/2005

José Manuel Torres

NOÇÕES BÁSICAS DE VISUAL BASIC E VBA-EXCEL

Características do Visual Basic

- Versão moderna da linguagem Basic para o sistema operativo Windows desenvolvida pela Microsoft com capacidade de criação rápida e fácil de programas com ambiente gráfico MS-Windows.
- Linguagem com características específicas para desenvolver rápida e facilmente a parte "Visual" (GUI - *Graphical User Interface* ou Interface Gráfica com o Utilizador) de um programa.
- Pode ser usada em outras aplicações Microsoft (como o MS-Office) tais como MS-Excel ou MS-Access. Nesse caso designada VBA (Visual Basic for Applications).
- Implementa o modelo orientado aos eventos. Muitas das funções escritas respondem a eventos do utilizador, como por exemplo, o utilizador clica num botão de um formulário e como resposta é chamada uma função com uma sequência de instruções em Visual Basic.

Ambiente Integrado de Desenvolvimento (IDE) Editor do Visual Basic

- Operado de forma interactiva e agradável por menus, janelas e caixas de diálogo
- Possui ferramentas de desenho de interfaces gráficas para desenvolvimento rápido
- Editor de texto com grandes potencialidades
- Fácil inserção e teste de correcções e facilidades de "debugging" avançadas
- Várias facilidades para o desenvolvimento de programas extensos de forma modular, em particular através da utilização de "modules"
- Acesso fácil a bibliotecas de objectos (funções e variáveis) possibilitando a programação e desenvolvimento rápido de aplicações Windows usando serviços de outros pacotes de software como por exemplo o MS-Office.
- Possibilidade de programação orientada por objectos

Linguagem Visual Basic

Um programa de Visual Basic obedece às seguintes regras de escrita:

- Escreve-se tipicamente uma instrução por linha sem símbolo de terminação;
- Contudo se a instrução se prolongar por mais do que uma linha usa-se a sequência " _" para fazer a ligação entre linhas;
- Pode-se escrever mais do que uma instrução numa linha, desde que separadas por ":";

- Os comentários são assinalados com o símbolo “”

Exemplo: Procedimento para Calcular Factorial em VBA-Excel

```
Public Sub factorial()
Dim n, termo As Integer
Dim fact As Long
  n = Range("B14").Value ‘ lê valor que está na
                         ‘ célula B14 do MS-Excel
  fact = 1
  termo = n
  Do While termo > 1
    fact = fact * termo
    termo = termo - 1
  Loop
  Range("B15").Value = fact    ‘ coloca resultado na célula
                               ‘ B15 do MS-Excel
End Sub
```

Leitura e Escrita de Dados em VBA-Excel

Vamos considerar ao longo dos algoritmos desenvolvidos os dois modos seguintes de leitura e escrita de dados:

- Leitura e escrita de valores usando células do Excel
- Leitura e escrita de dados usando janelas de diálogo de entrada e saída

Escrita de Dados

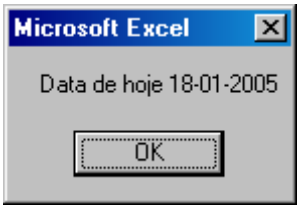
- Escrita de valores em células do MS-Excel usando a função: Range(*referência da célula*).Value = valor (Nota: o operador & serve para ligar ou concatenar o valor de duas variáveis. As células afectadas são as da folha activa de MS-Excel)

```
Range("b1").Value = fact      ‘escreve na célula B1 do MS-Excel
                              ‘o valor da variável fact
Range("b2").Value = “O factorial = ” & fact
Range("b3").Value = "Data de hoje " & DateTime.Date
Range("b4").Value = "Neste momento são " & DateTime.Time
Range("b5").Value = “A soma de 12 + 14 = ” & 12 + 14
Range("b6").Value = 0.1 * 2005     ‘escreve 0.1×2005 na célula B6
```

- Pode-se fazer a escrita de resultados com caixas de diálogo de saída MsgBox

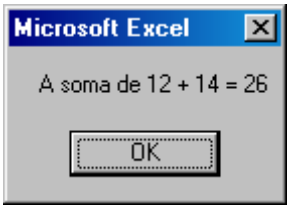
Exemplo 1:

```
MsgBox "Data de hoje " & DateTime.Date
```



Exemplo 2:

```
MsgBox "A soma de 12 + 14 = " & 12 + 14
```



Leitura de Dados

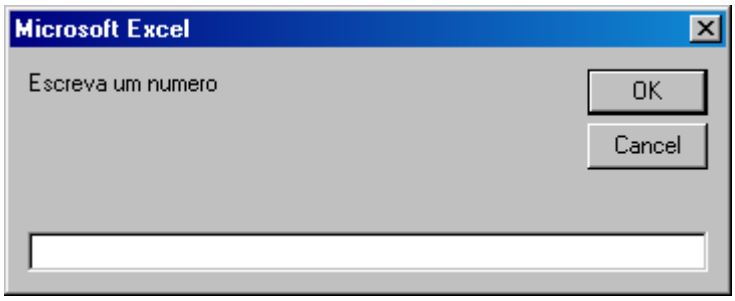
- Leitura de células do MS-Excel usando a função: Range(*referência da célula*).Value = valor

```
Dim n As Integer
n = Range("b1").Value          'lê valor da célula B1 do MS-Excel
                               'e coloca-o na variável n
Exemplo 2:
Dim nome As String
nome = Range("b2").Value
Exemplo 3:
Dim altura As Double
altura = Range("b3").Value
```

- Leitura de dados usando caixas de diálogo de entrada InputBox

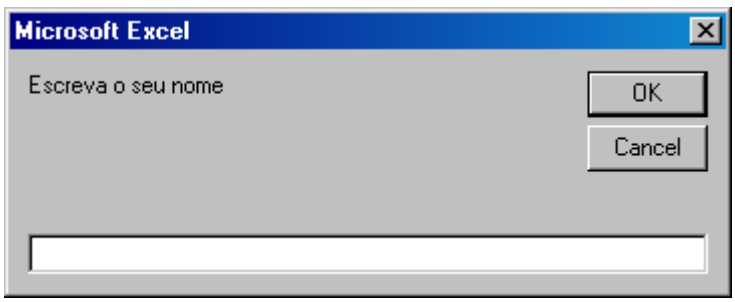
Exemplo 1:

```
Dim n As Integer
n = InputBox("Escreva um numero")
```



Exemplo 2:

```
Dim nome As String
nome = InputBox("Escreva o seu nome")
```



ESTRUTURAS DE CONTROLO

Com o que já foi visto até agora é possível criar programas que "façam alguma coisa", no entanto as suas capacidades estão bastante limitadas pois todos eles são executados como uma sequência simples de instruções.

Num programa, os mecanismos usados para controlar qual a instrução que vai ser executada a seguir, são designados por estruturas de controlo de fluxo, uma vez que controlam o fluxo da execução do programa.

Existem diversos métodos para realizar este tipo de operações que fazem parte das linguagens de programação.

Podem-se identificar duas classes de estruturas de controlo em Visual Basic:

- Estruturas condicionais ou de decisão
  - If ... Then
  - if ... Then ... Else
  - Select Case
- Estruturas repetitivas (ciclos)
  - Do ... Loop
  - For ... Next
  - For Each ... Next

As estruturas condicionais permitem a execução de determinados blocos do código fonte segundo um conjunto de condições que se verificam ou não.

Os ciclos permitem a repetição de um bloco de operações, até que se verifique a condição de saída do ciclo.

Estrutura de Decisão

Estrutura If ... Then

Formato:	Formato alternativo:

If condição **Then** instrução	**If** condição **Then**
	[bloco_de_instruções]
	End If

Exemplo:	Exemplo:

`If (n > max) Then max = n`	`If (n1 > n2) Then`
	`  maior = n1`
	`  menor = n2`
	`End If`

Estrutura If ... Then ... Else

Formato (figura 1):	Formato alternativo:

If condição **Then**
[bloco_de_instruções_1]
[**Else**
[bloco_de_instruções_2]]
End If
	If condição1 **Then**
	[bloco_de_instruções_1]
	[**ElseIf** condition2 **Then**
	[bloco_de_instruções_2]] ...
	[**Else**
	[bloco_de_instruções_n]]
	End If

Exemplo:	Exemplo:

`If (n1 > n2) Then`
`  maior = n1`
`  menor = n2`
`Else`
`  maior = n2`
`  menor = n1`
`End If`
	`If (n1 >= n2) And (n1 >= n3) Then`
	`  maior = n1`
	`  menor =`
	`Excel.WorksheetFunction.Min(n2, n3)`
	`ElseIf (n2 >= n1) And (n2 >= n3) Then`
	`  maior = n2`
	`  menor =`
	`Excel.WorksheetFunction.Min(n2, n3)`
	`Else`
	`  maior = n3`
	`  menor =`
	`Excel.WorksheetFunction.Min(n1, n2)`
	`End If`

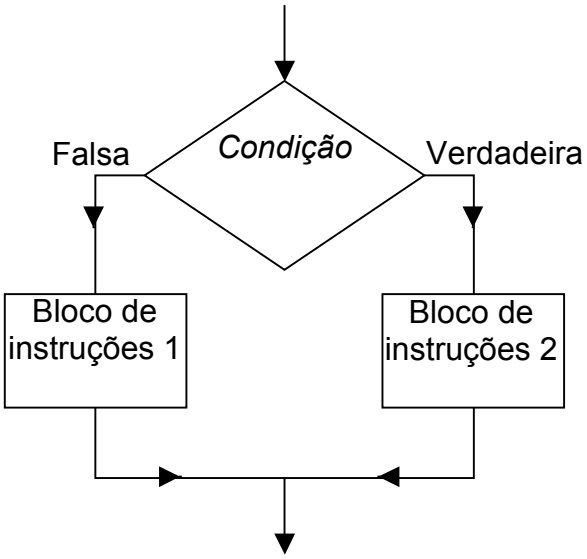


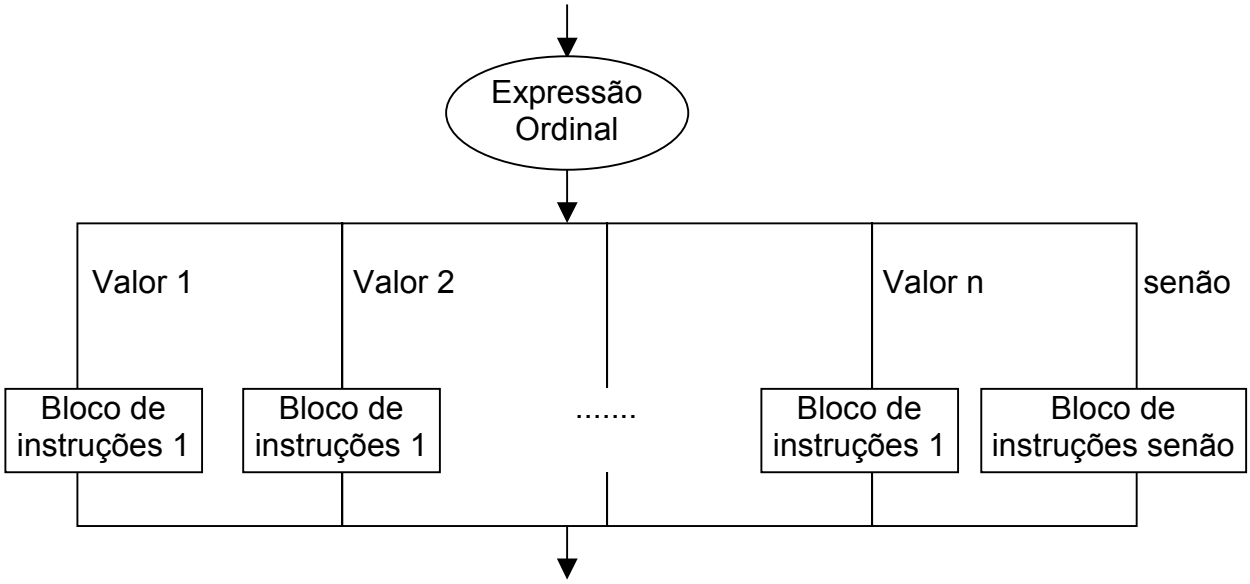
figura 1 – if…then…else… endif

Estrutura Select Case

Permite seleccionar valores ou listas de valores separados por virgulas a partir de uma expressão de teste. Fá-lo de um modo mais elegante e legível do que o If...Then...Else.

Formato:	Exemplo:

Select Case expressão_de_teste
[**Case** valor1
[bloco_de_instruções_1]]
[**Case** valor2
[bloco_de_instruções_2]]
.
.
[**Case Else**
[bloco_de_instruções_n]]
End Select
	`Select Case mes`
	`Case 1, 3, 5, 7, 8, 10, 12`
	`MsgBox "Mês com 31 dias"`
	`Case 4, 6, 9, 11`
	`MsgBox "Mês com 31 dias"`
	`Case Else`
	`MsgBox "Mês de Fevereiro com 28/29 dias"`
	`End Select`



Estrutura Repetitivas ou ciclos

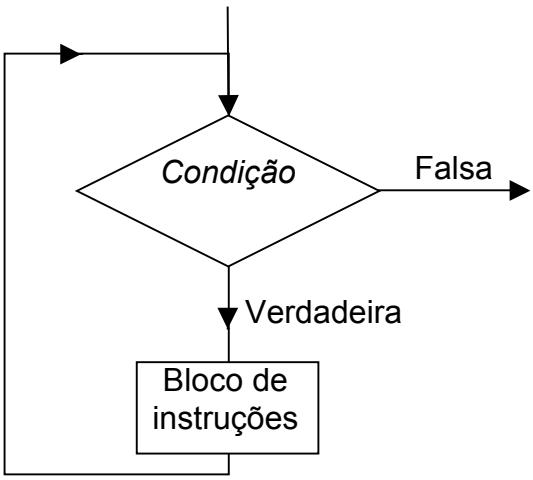
Estrutura Do While

Formato 1:	Formato 2:
Do While condição [bloco_de_instruções] **Loop**	**Do** [bloco_de_instruções] **Loop While** condição
Exemplo:	Exemplo:

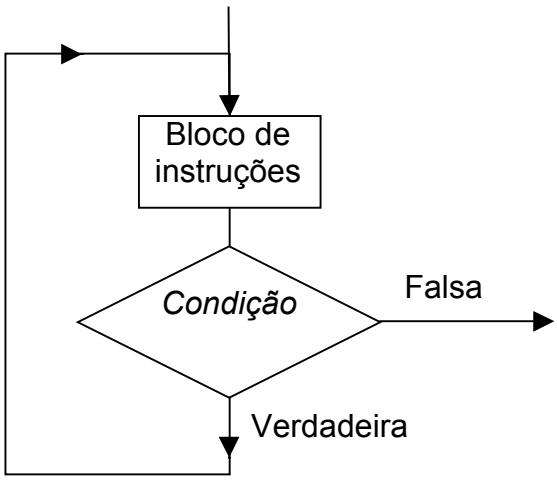
`Do While termo > 1`
`  fact = fact * termo`
`  termo = termo - 1`
`Loop`
	`Do`
	`  fact = fact * termo`
	`  termo = termo - 1`
	`Loop While termo > 1`

Nota: executa o ciclo zero ou mais vezes.	Nota: Garante a execução do ciclo pelo menos uma vez

Formato 1:



Formato 2:



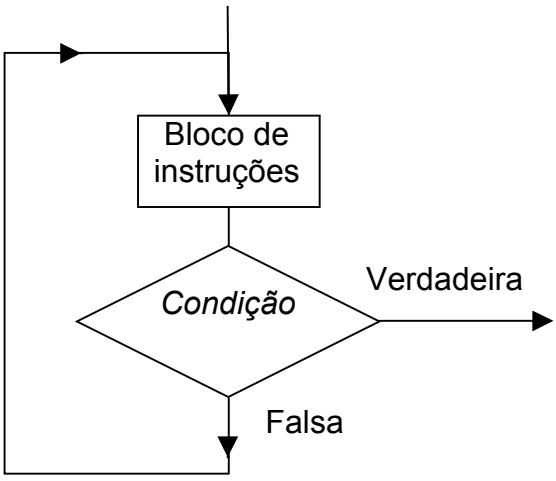
Estrutura Do Until

Formato 1:	Formato 2:
Do [bloco_de_instruções] **Loop Until** condição	**Do Until** condição [bloco_de_instruções] **Loop**
Exemplo:	Exemplo:

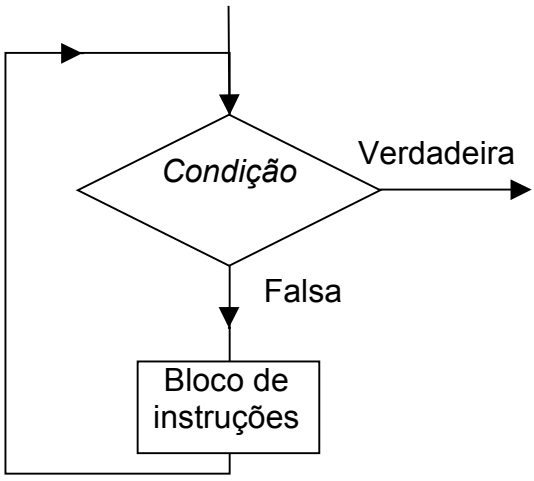
`Do`
`  fact = fact * termo`
`  termo = termo - 1`
`Loop Until termo <= 1`
	`Do Until termo <= 1`
	`  fact = fact * termo`
	`  termo = termo - 1`
	`Loop`

Nota: Garante a execução do ciclo pelo menos uma vez.	Nota: executa o ciclo zero ou mais vezes.

Formato 1:



Formato 2:

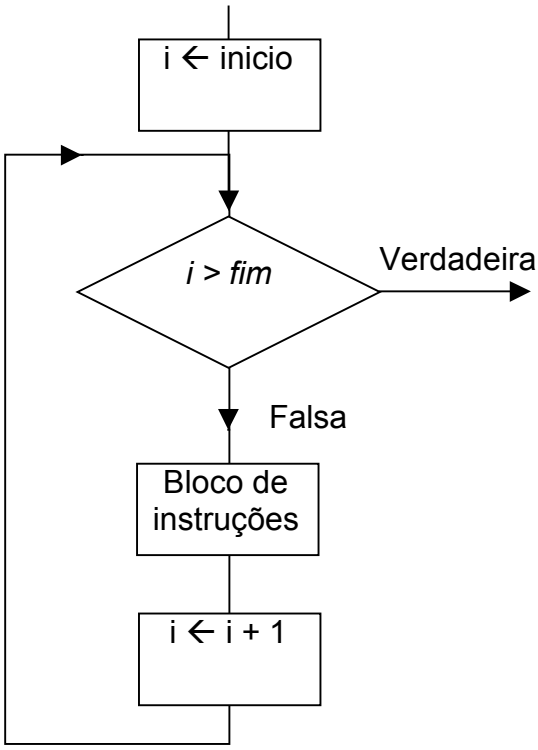


Estrutura For ... Next

Formato:	Exemplo:

For contador = inicio **To** fim [**Step** incremento]
[bloco_de_instruções]
Next [contador]
	`For i = 1 To 100`
	`soma = soma + i`
	`Next i`

Incremento pode ser positivo (apenas se inicio <= fim) ou negativo (apenas se inicio >= fim)



Exemplo de soma de todos os números pares até 100:

```
For i = 2 To 100 Step 2
soma = soma + i
Next i
```

Estrutura For Each... Next

Uso semelhante ao For...Next, mas usado para percorrer vectores ou colecções de objectos. É especialmente útil se não se souber o tamanho da colecção.

Exemplo:

```
Dim soma As Integer
Dim num(10) As Integer
... ' preenchimento do vector num
For Each elemento In num
 soma = soma + elemento
Next elemento
MsgBox "Soma = " & soma
```

TIPOS E DECLARAÇÕES DE DADOS

- Um programa deve definir todas as variáveis utilizadas e quais os seus respectivos tipos (equivalentes aos domínios na matemática N, Z, R, etc)
- Tipos determinam o conjunto de valores admissíveis e as operações aplicáveis
- Nalgumas linguagens (como Visual Basic), os tipos das variáveis podem ser definidos implicitamente. No entanto, também em Visual Basic é conveniente efectuar uma declaração explicita das variáveis usando a instrução: *Dim*

Tipos de Dados nas Linguagens de Programação

Classificação Usual de Tipos de Dados

Primitivos (Pré-definidos)
- Ordinais (inteiro, booleanos):
  - Os tipos Ordinais são baseados no conceito de ordem ou sequência. Para além de ser possível comparar dois valores ordinais, também é possível calcular o predecessor e o sucessor de qualquer valor ordinal.
- Reais
- Datas e Tempos

Definidos pelo Utilizador
- Não Estruturados (um só domínio: meses do anos, naipes de cartas, etc)
- Estruturados (vários domínios: vector, matriz, registo, etc)

Tipos de Dados em Visual Basic

Tipo de Dados	Descrição
Byte	Tipo sem sinal (0 a 255; 1 byte)
Boolean	Tipo Verdadeiro/Falso; por defeito é falso (2 bytes)
Integer	Tipo inteiro normal (-32,768 a 32,767; 2 bytes)
Long	Inteiro longo (-2,147,483,648 a 2,147,483,647; 4 bytes)
Single	Número real de precisão simples (4 bytes)
Double	Número real de precisão dupla (8 bytes)
Currency	Formato moeda (8 bytes)
Date	Datas e tempos (1/1/100 a 31/12/9999; 8 bytes)
Object	Objectos
String	Cadeias de caracteres. Pode ser: -Fixa (tamanho de 1 a aproximadamente 65,400) -Variável (tamanho de 1 a aproximadamente 2 biliões)
Variant	Tipo indefinido (pode assumir qualquer tipo)

Tipo definido pelo utilizador (usando Type)	Exemplo:
	`Type aluno`
	`  Nome as String`
	`  Numero as Integer`
	`End Type`
	
	`Dim aluno1 as aluno`
	`Aluno1.nome = "Ana"`
	`Aluno1.numero = 1234`

Valores especiais no tipo “Variant”: Empty; Null; Error

Fuções de conversão de tipos	Converte a expressão para o tipo
Cbool	Booleano
Cbyte	Byte
Ccur	Currency
Cdate	Date
CDbl	Double
Cint	Integer
CLng	Long
CSng	Single
CStr	String
Cvar	Variant
CVErr	Error

A função VarType(X) devolve o tipo de dados da variável X. Os valores devolvidos podem ser os seguintes:

Constante VB	Valor	Descrição
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency value
vbDate	7	Date value
vbString	8	String
vbObject	9	Object
vbError	10	Error value
vbBoolean	11	Boolean value
vbVariant	12	Variant (used only with arrays of variants)
vbDataObject	13	A data access object
vbDecimal	14	Decimal value
vbByte	17	Byte value
vbUserDefinedType	36	Variants that contain user-defined types
vbArray	8192	Array

Operadores Aritméticos VB

Operadores Unários

op X em que X pode ser uma constante, variável ou expressão

Operador	Operandos	Resultado
SIMÉTRICO: - X	inteiro ou real	Tipo dos operandos

Operadores Binários

X op Y em que X e Y podem ser uma constantes, variáveis ou expressões

Operador	Operandos	Resultado
EXPONENCIAL: X ^ Y	Inteiros ou reais (se X<0 então Y tem de ser inteiro)	Real
ADIÇÃO: X + Y	Inteiros ou reais	Tipo dos operandos
SUBTRACÇÃO: X - Y	Inteiros ou reais	Tipo dos operandos
MULTIPLIÇÃO: X * Y	Inteiros ou reais	Tipo dos operandos
DIVISÃO REAL: X / Y	Inteiros ou reais	Real
DIVISÃO INTEIRA: X \ Y	Inteiros ou reais arredondados	Inteiro
RESTO DIV. INT.: X mod Y	Inteiros ou reais arredondados	Inteiro

Operadores Booleanos e Lógicos (comparação bit a bit) VB

Operador	Sintaxe	Operandos	Resultado
NOT LÓGICO:	**NOT X**	Inteiro	Inteiro
AND LÓGICO:	**X AND Y**	Inteiros	Inteiro
OR LÓGICO:	**X OR Y**	Inteiros	Inteiro
XOR LÓGICO:	**X XOR Y**	Inteiros	Inteiro
EQUIVALÊNCIA LÓGICA (⇔)	**X Eqv Y**	Inteiro	Inteiro
IMPLICAÇÃO LÓGICA (⇒)	**X Imp Y**	Inteiro	Inteiro

X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X Imp Y	X Eqv Y
True	True	False	True	True	False	True	True
True	False	False	False	True	True	False	False
False	True	True	False	True	True	True	False
False	False	True	False	False	False	True	True

Operadores de Comparação VB

Menor: <

Menor ou igual: <=

Maior: >

Maior ou igual: >=

Igualdade: =

Desigualdade: <>

Outros Operadores VB

Operador	Operandos
Concatenação de Strings: X & Y	X e Y são Strings (senão converte a expressão para string)
Endereço de um procedimento: AddressOf X	X é o nome do procedimento

Comparação de uma string S com um padrão P: S Like P	O padrão P pode ser composto pelos símbolos especiais: ?; *; # ; [charlist]; [!charlist].
	Exemplos:
	`"aBBBa" Like "a*a"  ' True.`
	`"F" Like "[A-Z]"  ' True.`
	`"F" Like "[!A-Z]"  ' False.`
	`"a2a" Like "a#a"  ' True.`
	`"aM5b" Like "a[L-P]#[!c-e]"  ' True.`
	`"BAT123khg" Like "B?T*"  ' True.`
	`"CAT123khg" Like "B?T*"  ' False`

VARIÁVEIS INDEXADAS E CADEIAS DE CARACTERES

Vectores: Definição e Utilização

Cálculo dos desvios em torno da média, das notas de 6 alunos:

```
Dim nota1, nota2, nota3, nota4, nota5, nota6 As Integer
...  ' lê valores para estas 6 variáveis
media = (nota1 + nota2 + nota3 + nota4 + nota5 + nota6) / 6
MsgBox (nota1 - media)
...
MsgBox (nota6 - media)
```

Em situações como esta - conjunto ordenado de dados, todos do mesmo tipo - interessa dispor de uma estrutura de dados que permita uma manipulação fácil dos elementos do grupo.

Usamos então um tipo estruturado, como por exemplo um vector, variável indexada por um único índice (“one dimensional array”):

Estrutura: lei de correspondência

$$v: i \rightarrow \upsilon$$

$$i \rightarrow v[i] \qquad \text{(alternativamente: v(i) ou } v_i \text{)}$$

índice ↑ ↑ componente do vector

No exemplo anterior :

- Domínio : i
- Contradomínio : [0.20] $\subset$ z

Os v[i] constituem uma sequência de variáveis, todas do mesmo tipo, referida genericamente por “ vector v[] “ ou vector v() na notação de Visual Basic

Declaração de variável indexada

Usando um vector podemos operar com uma sequência arbitrariamente grande de variáveis do mesmo tipo:

```
Public Sub estatistica_de_notas()
Const max_alunos = 50
Dim alunos, n_alunos As Integer
Dim media As Double
Dim notas(max_alunos) As Integer
Do
n_alunos = InputBox("Escreva o número de alunos")
Loop Until n_alunos <= max_alunos
     media = 0
     For aluno = 0 To (n_alunos - 1)
         notas(aluno) = InputBox("Escreva a nota do aluno número: "
& aluno)
         media = media + notas(aluno)
     Next aluno
     media = media / n_alunos
     MsgBox "A média = " & media
End Sub
```

Outros aspectos do uso de vectores :

Se o tamanho do vector não for declarado entre parênteses curvos então diz-se que o vector é um vector dinâmico:

```
Dim curva() As Double
```

A qualquer altura esse vector pode ser dimensionado com

```
ReDim curva(-100 to 100)
```

Uma variável indexada pode ser usada em qualquer expressão, como qualquer outra variável. O índice, por sua vez, pode ser qualquer expressão cujo valor esteja dentro da gama de índices.

Exemplos:

```
curva(-10) :=m + 20.0
curva(-i) :=m + 2.5 * curva(i)
```

Não seria valido:

```
x := curva (4.2)
```

Os elementos de um vector estão armazenados em posições consecutivas na memória, inicialmente com valor indefinido.

Exemplo:

```
Dim x(10 to 15) as integer
X(10) =2
for i =11 to 13
     x(i) = x(i-1) + 3
Next i
```

originaria os seguintes conteúdos : 2, 5, 8, 11, ?, ?

Operações com índices

a) - Expressão ordinal sem funções.

Exemplo do cálculo da diferença de médias dos valores de ordem par e ímpar de um vector:

```
Public Sub op_indice_a()
        Dim x(1 To 20) As Integer
        Dim i, mi, mp As Integer
        For i = 1 To 20
        x(i) = InputBox("Introduza nota do aluno " & i)
        Next i
        mi = 0: mp = 0
        For i = 1 To 2
            mp = mp + x(2 * i)
            mi = mi + x(2 * i - 1)
        Next i
        MsgBox ((mp - mi) / 10)
End Sub
```

b) - Expressão ordinal com funções.

Exemplo do calculo do preço de venda de um motor.

```
Public Sub escalao()
'Determina o preço de venda de um motor de acordo com a formula:
' venda=custo+tax
'A taxa depende do escalão de potência:
' 10 a 19 cv, 20 a 29 cv,........, 50 a 59 cv
Const n_escalões = 5
Dim taxa(1 To n_escalões) As Double
Dim custo, potencia As Double
        ...
        taxa(1) = 0.5: taxa(2) = 0.5: taxa(3) = 0.55
        taxa(4) = 0.57: taxa(5) = 0.6
        ...
        custo = InputBox
        potencia = InputBox
        MsgBox ("Preço de venda: " & custo * (1 + taxa(Int(potência
/ 10))))
End Sub
```

Neste exemplo a fórmula de geração de índices é conhecida. Quando tal fórmula é desconhecida ou não existe, teremos de usar um outro tipo de resolução.

c) - Expressão ordinal com "arrays".

Exemplo de calculo de desvios em torno da média de subconjuntos arbitrários de 5 valores, de um conjunto de 40 valores:

```
Public Sub calc_desvios()
```

```
    Dim x(1 To 40) As Double
    Dim índice(1 To 5) As Integer
    Dim i, media As Integer
        ...
        For i = 1 To 40
        x(i) = InputBox("Valor de " & i)
        Next i
        MsgBox ("Que elementos selecciona (5 indices) ?")
        For i = 1 To 5
          indice(i) = InputBox("Valor de " & i)
        Next i
        media = 0
        For i = 1 To 5
        media = media + x(indice(i))
        Next i
        media = media / 5
        For i = 1 To 5
        MsgBox ("Desvio " & i & " : " & Abs(x(indice(i)) - m))
        Next i
End Sub
```

Atenção: : A utilização de variáveis indexadas e, em particular vectores, deve restringir-se à situação em que se torna necessário "guardar" uma dada sequência de elementos.
O uso desnecessário de variáveis indexadas, tendência frequente nos alunos que se iniciam na programação, é ineficiente em espaço de memória e geralmente promotor de soluções pouco elegantes, confusas ou mesmo erradas. O aluno deve procurar resistir à tentação de "guardar elementos num "array"...antes que "fujam" !...

d) - Não necessidade de utilização de "arrays"

Exemplo : Calcular a maior diferença absoluta de valores consecutivos de inteiros positivos, lidos do teclado. A leitura termina com um valor negativo e são digitados pelo menos 2 valores válidos.

```
Public Sub P1()   'Fraca solução com dois vectores
Dim x(1 To 100), d(1 To 100) As Integer
Dim i, n, max As Integer
    n = 0
    Do
     n = n + 1
     x(n) = InputBox
    Loop Until x(n) < 0
    max = -maxint
    For i = 1 To n - 2
    d(i) = Abs(x(n)) - x(n + 1)
    Next i
    For i = 1 To n - 2
     If d(i) > max Then max = d(i)
    MsgBox ("Máximo = " & max)
End Sub
```

```
Public Sub P2()   'solução recomendada, sem vectores
    Dim x, xi, d, max As Integer
    max = -maxint
    xi = InputBox
    Do
      x = InputBox
      d = Abs(x - xi)
      If (x <> -1) And (d > max) Then max = d
      xi = x
    Loop Until x = -1
    MsgBox (max)
End Sub
```

Ordenação de vectores

Estima-se que, em média, cerca de metade do tempo de operação dos computadores é gasto em operações de ordenação e pesquisa de dados.

Estas operações vão ser ilustradas para vectores mas generalizam-se facilmente para outras estruturas de dados.

Tipos mais usuais de algoritmos de ordenação:

Selecção - baseia-se em inspecções sucessivas sequenciais de partes do vector, seleccionando o menor (ou maior) elemento que é então colocado na posição correcta.

"bubble sort" - baseia-se em sucessivas trocas de dois elementos consecutivos sempre que estes estão fora de ordem.

Fusão - baseia-se na fusão de subconjuntos de elementos previamente ordenados.

Árvore - baseia-se na construção e travessia de árvores (estrutura de dados complexa)

Partição e troca - baseia-se em partições sucessivas da estrutura de dados que depois são ordenadas separadamente.

Ordenação por selecção

Trata-se de inspeccionar o vector sequencialmente, seleccionando repetidamente o elemento que obedece a uma dada condição.

Exemplo de ordenação por selecção com permuta de elementos colocando os elementos do vector por ordem ascendente:

x[] inicial	1º passo	2º passo	3º passo	4º passo	5º passo	x[] final
6	**6**	2	2	2	2	2
31	31	**31**	6	6	6	6
10	10	10	**10**	10	10	10
18	18	18	18	**18**	13	13
2	**2**	**6**	31	31	**31**	18
13	13	13	13	**13**	**18**	31

A eficiência do algoritmo em tempo de execução, para um vector em N elementos, depende do numero de comparações efectuado:

$$\text{Total de comparações} = \sum_{\text{passo}=1}^{N}(N - \text{passo}) = N(N-1)/2$$

O algoritmo tem uma eficiência O(N^2). É possível mostrar que o algoritmo “bubble-sort” tem a mesma eficiência, em termos médios.

Implementação em Pascal:

```
program ord_selecção ;
const
        n_elem = 50 ;
var
        x : array [1..n_elem] of real ;
        passo, ind, indmin : integer ;
        aux : real ;
begin
        writeln(“Escreva os elementos do vector”) ;
        for ind:=1 to n_elem do read (x[ind]) ;

        for passo := 1 to n_elem -1 do          {inicio do algoritmo de ordenação}
        begin
                indmin := passo ;
                for ind := passo + 1 to n_elem do
                        if x [ind] < x [indmin] then indmin := ind ;
                if indmin <> passo then
                begin
                        aux := x[passo] ;
                        x[passo] := x[indmin] ;
                        x[indmin] := aux ;
                end ;
        end ;
        for ind := 1 to n_elem do WriteLn ( x[ind] ) ;
end
```

Ordenação por trocas (“bubble sort”)

Sucessivas trocas de dois elementos caso estes estejam fora de ordem. Exemplo de ordenação por “bubble sort”.

x[] inicial	1º passo	2º passo	3º passo	4º passo	5º passo	6º passo	7º passo
6	**6**	**6**	**6**	**6**	**2**	2	2
31	**31**	31	31	31	31	**31**	**10**
10	10	**10**	10	10	10	**10**	31
18	18	18	**18**	18	18	18	**18**
2	2	2	2	**2**	6	6	6
13	13	13	13	13	**13**	13	13

8º passo	9º passo	10º passo	11º passo	12º passo	13º passo	14º passo	15º passo	x[] final
2	2	2	2	2	2	2	2	2
10	**6**	6	6	6	6	6	6	6
31	31	**31**	**18**	**10**	10	10	10	10
18	18	**18**	31	31	**31**	**18**	13	13
6	10	10	**10**	18	**18**	31	**31**	18
13	**13**	13	13	**13**	13	**13**	**18**	31

Implementação em Pascal:

```
program ord_bubble_sort ;
const
        n_elem = 50 ;
var
        x : array [1..n_elem] of real ;
        i1, i2 : integer ;
        aux : real ;
begin
        writeln(“Escreva os elementos do vector”) ;
        for i1:=1 to n_elem do read (x[i1]) ;

        for i1 := 1 to n_elem -1 do          {inicio do algoritmo de ordenação}
                for i2 := i1+1 to n_elem do
                begin
                        if x [i2] < x [i1] then
                        begin
                                aux := x[i1] ;
                                x[i1] := x[i2] ;
                                x[i2] := aux ;
                        end ;
                end ;
        for i1 := 1 to n_elem do WriteLn ( x[i1] ) ;
end
```

Ordenação por fusão

A fusão de duas estruturas de dados ordenadas consiste em criar uma nova estrutura de dados que contêm os elementos das outras duas, também ordenados.

Exemplo para vectores:

x[] 13 21 39
y[] **7** 28 $\rightarrow$ z[] 7

x[] **13** 21 39
y[] 28 $\rightarrow$ z[] 7 13

x[] **21** 39
y[] 28 $\rightarrow$ z[] 7 13 21

x[] 39
y[] **28** $\rightarrow$ z[] 7 13 21

x[] **39**
y[] $\rightarrow$ z[] 7 13 21 28 39

Algoritmo para fundir dois subvectores ordenados, com n1 e n2 elementos, de um vector dado x[], resultando um vector y[]:

1 - Sejam i1 e i2 os índices dos subvectores de x[] e j o índice de y[]
Inicialmente i1 $\leftarrow$ j $\leftarrow$ 1 ; i2 $\leftarrow$ n1 + 1

2 - Enquanto (i1 $\leq$ n1 e i2 $\leq$ n) faça
2.1 Compare x[i1] com x[i2] e transporte o menor para y[i]
2.2 Actualiza i1 ou i2) e j

3 - Copie para y[] os elementos que sobraram de x[] e termine.

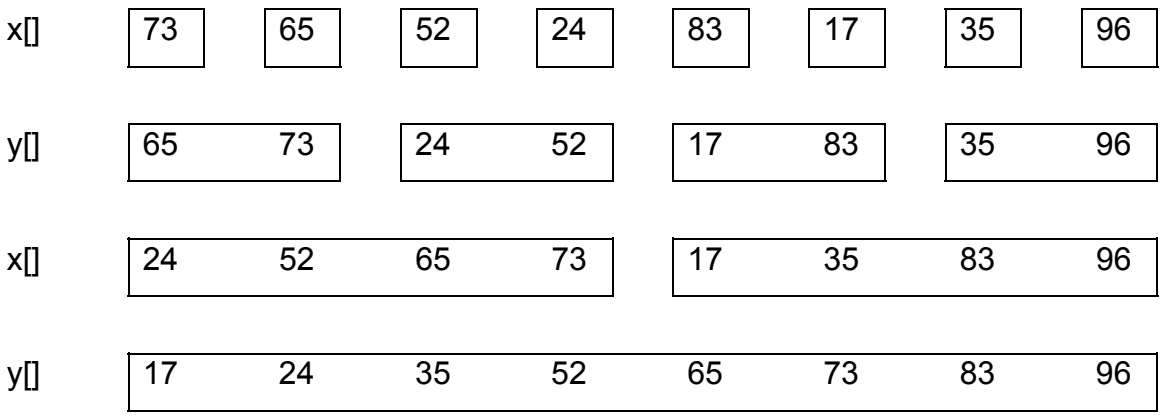
Implementação em Pascal:

```
const
        n=100;
type
        vector = array [1..n] of real ;
        ...
{inicio do algoritmo}
i1 := 1; i2 := n1 + 1; j := 1 ;

While (i1 <= n1) and (i2 <= n ) do          {copia o menor}
        if x[i1] <= x[i2] then
        begin
                y[j] := x[i1]
                Inc(i) ; Inc(j) ;
        end
        else begin
                y [j] := x[i2] ;
                Inc (i2) ; Inc (j) ;
        end ;

{copia os elementos que sobram }
if i1 > n1 then for k := i2 to n do
                begin
                        y[j] := x[k] ; Inc(j)
                end
           else for k := i1 to n1 do
                begin
                        y[j] := x(k) ; Inc (j) ;
                end
```

Este algoritmo de fusão pode ser usado para ordenar um vector à custa de sucessivas fusões conforme se ilustra para o caso de $n = 2^m$:



A fusão dá-se alternadamente do vector original x[] para o auxiliar y[] e vice-versa.

```
program   ordena_por_fusão; { para n= 2^m }
const
        nmax = 50;
type
        vector = array [ 1..nmax ] of real ;
var
        x, y : vector ;
        n, npass, pass, size, subpass : integer ;
        in1, in2, i1, i2, j, k : integer ;
begin
        WriteLn ('Comprimento do vector (2^m) ?');  ReadLn (n) ;
        WriteLn ('Escreva ', n, ' valores') ;
        for k := 1 to n do Read ( x[k] ) ;                  {leitura do vector}
        npass := round ( log (n)/log(2)) ;                  {m, n. o de passagens}
        size := 1 ;                                         {comprim. subvector}
        for pass := 1 to npass do
        begin                                               {prepara uma passagem}
                in1 := 1 ; in2 := in1 + size ;              {indices iniciais}
                subpass := n / (2*size) ;                   {n.o de fusões}
                for k := 1 to subpass do
                begin                                       {inicialize uma fusão simples}
```

```
i1 := in1; i2 := in2; j := in1;
While (i1+1-in1<=size and (i2+1-in2<=size) do
if pass mod 2 = 1 then
if x[i1] <=x[i2] then
        begin                          {de x para y}
                y[j] := x[i1];
                Inc(i1); Inc (j);
        end
        else
        begin
                y[j] := x[i2];
                Inc(i2); Inc(j);
        end
else if y[i1] <= y [i2] then
        begin                          {de y para x}
                x[j] := y[i1];
                Inc[i1]; Inc[j];
        end
        else
        begin
                x[j] := y[i2];
                Inc(i2); Inc(j);
        end ;
                                       {copia elem. que sobram}
if i1+1-in1 > size then
        if pass mod 2=1 then
                for k := i2 in2+size-1 do
                begin
                        y[j] := x[k]; Inc (j)
                end
        else for k := i2 to in2+size-1 do
                begin
                        x[j] := y[k]; Inc(j)
                end
else    if pass mod 2 =1 then
```

```
                                    for k := i1 to in1+size-1 do
                                    begin
                                        y[j] := x[k]; Inc (j)
                                    end
                                else for k := i1 to in1+size-1 do
                                    begin
                                        x[j] := y[k]; Inc(j)
                                    end ;
                            in1 := in2 + size ; in2 := in12 + size;
                            {prepara nova fusão}
                        end;

                        size := 2*size; {prepara nova passagem}
                    end;
                    if npass mod 2 = 1 then
                    for k := 1 to n do x[k] := y[k];
                    WriteLn; WriteLn('Vector ordenado :');
                    for k := 1 to n do WriteLn( x[k] );
end.
```

É possível mostrar que este algoritmo, tal como os baseados em árvores e em partição-troca ("quick sort"), tem uma eficiência, em termos médios, de $O(n\log_2 n)$.

Na prática há alguns tempos extra não contabilizados e verifica-se que só a partir de $N \approx 30$ se torna conveniente não usar ordenação por selecção.

Pesquisa de vectores

Há fundamentalmente dois métodos de pesquisa de estruturas de dados: Linear e Binária

Pesquisa linear

A estrutura é rastreada sequencialmente até encontrar o elemento pretendido:

```
...
v = InputBox("Elemento a encontrar ?")
i = 1
Do While (i <= n) And (x(i) <> v)
 i = i + 1
Loop
If i > n Then ...       'elemento v não foi encontrado
Else ...                'elemento v foi encontrado
```

Este algoritmo tem uma eficiência 0(n) necessitando em média de n/2 comparações.

Pesquisa binária

Aplica-se apenas a estruturas de dados ordenadas.

A pesquisa é efectuada em sucessivas dicotomias da estrutura de dados, de forma análoga ao algoritmo “lista-telefónica” referido na introdução à algoritmia.

Cada dicotomia é referenciada por dois índices : alto e baixo

Exemplo para n = 2^P-1 = 7 elementos:

x[i] : 2 4 9 15 20 27 34

i : 1 2 3 4 5 6 7

elemento a encontrar: v=9

Rastreio do algoritmo

Passo	baixo	alto	médio	x[médio]	próx dicotomia
1	1	7	4	15 > v	Inferior
2	1	3	2	4 < v	Superior
3	3	3	3	9 = v	Final

São necessárias p comparações para n=2^P-1 elementos

Eficiência média da pesquisa binária: $0(\text{trunc}(\log_2(n))-1)$

Implementação em VB:

```
...
Dim x(1 To n) As Double
Dim baixo, alto, medio As Integer
Dim sucesso As Boolean
...
baixo = 1
alto = n
sucesso = False
Do While (baixo <= alto) And (Not sucesso)
    medio = Int((baixo + alto) / 2)
    If v < x(medio) Then alto = medio - 1
    ElseIf v > x(medio) Then baixo = medio + 1
    Else: sucesso = True
    '...
Loop
```

Matrizes

Variáveis indexadas (“arrays”) por mais de um índice.

Exemplos:

```
  Dim page(0 To 24, 0 To 79) As String
  Dim chess(1 To 8, 1 To 8) As figures
  Dim heat(0 To n, 0 To n, 0 To n, 0 To maxTime) As Double
```

A operação com matrizes é análoga à operação com vectores.

Exemplos:

```
        'apaga 5 linhas no topo do ecrã
        For linha = 1 To 5
            For coluna = 0 To 79
                page(linha, coluna) = " "
            Next j
        Next i

        'calcula próximos valores do campo de calor
        t_next = t + 1
        For i = 0 To n
         For j = 0 To n
          For k = 0 To n
           heat(i, j, k, t_next) = Exp(-alfa * heat(i, j, k, t))
          Next k
         Next j
        Next i
```

Vectores que contêm outros vectores

É possível criar um vector tipo Variant e preenche-lo com vectores de diferentes tipos.

```
Public Sub vector_num_vector()
   Dim intX As Integer   'variável contadora
   ' Declara e preenche array de inteiros
   Dim countersA(5) As Integer
      For intX = 0 To 4
         countersA(intX) = 5
      Next intX
   ' Declara e preenche array de strings
      Dim countersB(5) As String
         For intX = 0 To 4
            countersB(intX) = "hello"
         Next intX
   Dim arrX(2) As Variant   ' Declara array com 2 elementos
      arrX(1) = countersA()   ' Preenche array com outros arrays
      arrX(2) = countersB()
      MsgBox arrX(1)(2)   ' Faz Display de membros de cada array
      MsgBox arrX(2)(3)
End Sub
```

Outro exemplo

```
Public Sub temperatures_semana_5cidades()
Dim tempSemana5Cidades(1 To 5) As Variant
Dim temperaturasSemana(1 To 7) As Double
        For i = 1 To 5
            ' temperaturasSemana para cidade i
            ...
            tempSemana5Cidades(i) = temperaturasSemana
        Next i
        ' temperatura da cidade i no dia j
        MsgBox tempSemana5Cidades(i)(j)
End Sub
```

Operações com Matrizes

Operações usuais com matrizes bidimensionais:

Transposição

```
Dim x(1 To n, 1 To m), xt(1 To n, 1 To m) As Double
...
For i = 1 To n
    For j = 1 To m
        xt(i, j) = x(j, i)
    Next j
Next i
```

Soma

```
Dim x(1 To n, 1 To m), y(1 To n, 1 To m) As Double
...
For i = 1 To n
    For j = 1 To m
        x(i, j) = x(j, i) + y(j, i)
    Next j
Next i
```

Multiplicação

```
Dim x(1 To n_max, 1 To m_max), y(1 To n_max, 1 To m_max) As Double
Dim z(1 To n_max, 1 To m_max) As Double
...
For i = 1 To n
    For j = 1 To p
        soma = 0
        For k = 1 To m
            soma = soma + x(i, k) * y(k, j)
        Next k
        z(i, j) = soma
    Next j
Next i
```

CADEIAS DE CARACTERES (“STRINGS”)

Definição, Representação e Operações

Uma String é uma sequência de um ou mais caracteres. O tipo String em VB pode ser usado para declarar strings de tamanho fixo ou de comprimento variável (estrutura dinâmica).

A definição de um tipo *string* é feita da seguinte forma:

```
Dim str1 As String        'string de comprimento variável
Dim str2 As String * 50   'string de comprimento fixo com 50
                          'caracteres (preenchida com espaços)
```

Concatenação

A operação de concatenação é representada pelo símbolo: &

O elemento neutro da concatenação é a string nula: “”

Exemplos :

```
Dim msg(1 To 3) As String
msg(1) = "dia"
msg(2) = "trabalho"
msg(3) = "resultado"
For i = 1 To 3
 MsgBox "Bom " + msg(i)
Next i
```

```
Dim fim_linha As Boolean
Dim alarme As String * 10
alarme = "atenção !"
If fim_linha Then
    MsgBox alarme & " fim da linha"
End If
```

Conversão e atribuição

Uma string VB pode ser directamente atribuida a uma variável numérica desde que o conteúdo da string seja uma quantidade numérica, caso contrário poderá acontecer um erro em tempo de execução:

```
Dim s As String * 10
Dim n As Integer
s = "100"
n = s
```

O processo de alteração de um caracter da string passa pela utilização da função Mid (formato *Mid(str_alterar, inicio[, comprimento]) = nova_sub_str*):

```
Dim s1, s2 As String
s1 = "hoje faz sol"
Mid(s1, 6, 3) = "vem"          's1 fica "hoje vem sol"
s2 = "são 8 horas"
Mid(s2, 5, 1) = "9"            's2 fica "são 9 horas"
```

Comparação de cadeias de caracteres

A comparação de strings ou de strings e caracteres (considerados como strings de dimensão 1) é feita de acordo com os códigos ASCII.

Exemplos:

"A" < "B"	→	True	"MESA" = "MESA"	→	True
"a" < "B"	→	False	"Mesa" = "MESA"	→	False
"a" < "b"	→	True	"MESA" < "MESADA"	→	True

Funções pré-definidas

A seguir apresentam-se as principais funções de manipulação de strings em VB.

Len (string1 as string)

Retorna um número inteiro representando o número de caracteres da string *str*.

```
Public Sub comprimento()
    Dim comp As Integer
    Dim string1 As String
    string1 = "MESADA"
    comp = Len(string1)
    MsgBox comp
End Sub
```

Mid(string1, start[, length])

Retorna uma subcadeia de caracteres da cadeia string1, com início na posição start e com length caracteres.

Se start > Len(string1) é retornada a cadeia nula.

Se length > Len(string1) - start + 1 retorna apenas os restantes caracteres da string1.

Exemplos:

```
Dim s, r As String * 10
s = "ABCDE"
r = Mid(s, 2, 3)  'r = "BCD"
r = Mid(s, 2, 1)  'r = "B"
r = Mid(s, 7, 2)  'r = ""
r = Mid(s, 3, 4)  'r = "CDE"
```

Nota: O resultado de Mid é uma cadeia de caracteres.

InStr([start,]string1, string2[, compare])

Retorna a posição de string2 dentro de string1. Se string2 não existe no interior de string1, retorna zero.

Opcionalmente a função InStr suporta os seguintes argumentos:

- Start: inteiro que determina a posição inicial de pesquisa
- Compare: usada para seleccionar comparação binária ou textual

Exemplos:

```
Dim string1 As String
Dim p As Integer
string1 = "ABCDE"
p = InStr(string1, "AB")        ' p:=1
p = InStr(string1, "CD")        ' p:=3
p = InStr(string1, "C")         ' p:=3
p = InStr(string1, "FGH")       ' p:=0
```

Str (numero)

Converte um valor numérico numero na sua representação por uma cadeia de caracteres.

Exemplos:

```
Dim MyString as String
MyString = Str(459)   ' Returns " 459".
MyString = Str(-459.65)   ' Returns "-459.65".
MyString = Str(459.001)   ' Returns " 459.001".
```

Note-se a diferença interna (em memória) entre as duas representações:

- Um número inteiro é usado para armazenar quantidades numéricas que, por exemplo, podem funcionar como operandos em expressões aritméticas.
- Uma string é representada internamente por uma sequência de caracteres usando uma determinada codificação como por exemplo o ASCII ou o UNICODE.

Val (string1 as string)

Efectua a operação inversa de Str(). A cadeia de caracteres string1 é convertida no valor numérico correspondente que é devolvido pela função.

Se string1 não for uma representação válida de um valor numérico, a operação de conversão pára no primeiro carácter não válido de string1 (ignorando os espaços).

Exemplos:

```
Dim MyValue as Integer
MyValue = Val("2457")    ' Returns 2457.
MyValue = Val(" 2 45 7")    ' Returns 2457.
MyValue = Val("24 and 57")    ' Returns 24.
```

Outras funções

Nome da função	Descrição
StrComp(string1, string2[, compare])	Compara duas strings usando comparaçáo textual (ignora maiúsculas /minúsculas) ou comparação binária.
InstrRev(string1, string2[, start[, compare]])	Mesmo que InStr mas devolve a posição desde o fim da string1. (ver também: StrReverse(string1))
Left(string1, length)	Devolve os length caracteres desde o inicio da string1 (ver também: Right).
Trim(string1)	Remove espaços no inicio e no fim da string1 (ver também: LTrim, RTrim).
Asc(string1)	Código de caracter do primeiro caracter de string1.
Chr(número)	Devolve caracter correspondente ao código de caracter passado.
Replace(expression, find, replacewith[, start[, count[, compare]]])	Substitui em *expression* a string *find* pela string *replacewith*.
Lcase(string1)	Converte *string1* para minúsculas
Ucase(string1)	Converte *string1* para maiúsculas

FUNÇÕES E PROCEDIMENTOS

Noções Básicas

- Para lidar com programas extensos é necessário dividi-los em blocos que possam ser analisados de forma independente.
- Uma tarefa complexa é subdividida em diversas tarefas mais simples.
- Cada uma dessas tarefas pode ainda ser dividida de novo em tarefas mais elementares.
- Procedendo desta forma, obtêm-se no final, tarefas elementares muito facilmente codificáveis.
- Os blocos resultantes são mais facilmente compreendidos, testados, corrigidos e alterados.

Conceito de Bloco ou Rotina

Bloco:

- Conjunto de instruções “encapsulado” (“caixa preta” com um nome)
- Implementa uma parte de um algoritmo (sub-algoritmo)
- Deve ter poucas instruções (em princípio menos de 100)
- Designações possíveis: Sub-rotina, Procedimento, Função.

Atributos Fundamentais:

- **Nome:** Designação pela qual o bloco é chamado.
- **Parâmetros:** Informação fornecida ao bloco ou retornada pelo módulo para o programa principal.
- **Função:** Processamento de dados executado pelo bloco
- **Código:** Instruções que implementam o subalgoritmo correspondente ao bloco
- **Entrada e Saída:** Um bloco deve possuir uma única via de entrada e uma única via de saída.
- **Dados Internos:** Área de trabalho utilizada exclusivamente pelo bloco.

Motivações para o Uso de Blocos:

- **Simplicidade:** Cada sub-problema é mais simples de resolver. No caso de blocos usados repetidamente, só é necessário codificar o bloco uma única vez.
- **Manutenção:** Reduzidos custos e tempos de alteração de um programa. Blocos podem ser substituídos ou alterados independentemente.
- **Divisão de tarefas:** Permite a distribuição das tarefas de programação pelos diferentes programadores (peritos em domínios específicos).
- **Eficiência:** Permite a utilização de blocos existentes em bibliotecas de software.

Tipos de Rotinas disponibilizados pelo VB:

- **Subrotinas ou Procedimentos:** Executa uma determinada acção (não devolvem nenhum valor).
- **Funções:** Implementa uma determinada forma de cálculo de um valor (devolvem um valor).

Módulos de software em VB

- Em VB, as subrotinas e as funções são armazenadas em módulos.
- Cada módulo está associado a um ficheiro, e tem um determinado nome, como por exemplo “ModuloOrdenacao”.
- Existem diversos tipos de módulos: *standard modules*, *class modules* ou *form modules*.
- Para além das rotinas, um módulo contempla ainda uma zona de declaração de dados e tipos (Declaration).

Procedimentos ou Subrotinas

Definição:

Um procedimento ou subrotina é uma parte do programa que executa uma determinada acção, frequentemente baseado num conjunto de parâmetros.

Sintaxe:

```
[Private|Public][Static] Sub nome_procedimento (argumentos)
     instruções
End Sub
```

- O conjunto de instruções correspondente ao bloco (procedimento) é referenciado por um único identificador.
- O procedimento comunica com o exterior através de um conjunto de parâmetros (opcional). Estes são os valores passados para o procedimento pelo procedimento que o chama.
- Em VB pode-se distinguir entre procedimentos generalistas ou procedimentos que respondem a eventos específicos (como por exemplo o clicar de um botão na aplicação)
- Pode ser chamado usando Call *nome_procedimento*(*arg1*, *arg2*, ...)
- Ou alternativamente: *nome_procedimento arg1*, *arg2*, ...

Mecânica de Execução:

Exemplo: Procedimento para calcular o factorial (podendo, por exemplo, funcionar como uma macro MS-Excel).

```
Public Sub factorial()
Dim n, termo As Integer
Dim fact As Long
  n = Range("B14").Value ' lê valor que está na
                         ' célula B14 do MS-Excel
  fact = 1
  termo = n
  Do While termo > 1
    fact = fact * termo
    termo = termo - 1
  Loop
  Range("B15").Value = fact    ' coloca resultado na célula
                               ' B15 do MS-Excel
End Sub
```

Exemplo: O procedimento *calc_notas()* chama o procedimento *estatistica_de_notas()*.

```
Const max_alunos = 50

Public Sub calc_notas()
Do
 num_alunos = InputBox("Introduza o número de alunos")
Loop While num_alunos > max_alunos
Call estatistica_de_notas(num_alunos)
'ou poderia ser: estatistica_de_notas num_alunos
End Sub
```

```
Public Sub estatistica_de_notas(n_alunos As Integer)
Dim alunos As Integer
Dim media As Double
Dim notas(max_alunos) As Integer
media = 0
For aluno = 0 To (n_alunos - 1)
 notas(aluno) = InputBox("Escreva a nota do aluno número: " & aluno)
 media = media + notas(aluno)
Next aluno
media = media / n_alunos
MsgBox "A média = " & media
End Sub
```

- Existem diversos procedimentos pré-definidos pela linguagem: Procedimento Intrínsecos (Exemplos: Open, Close, etc.).
- Procedimentos definidos pelo utilizador designam-se por Procedimentos Extrínsecos.

Embutimento de Procedimentos e Alcance de Identificadores

As variáveis (dados) declarados na zona de declaração dos módulos designam-se por variáveis (dados) globais.

As variáveis (dados) declarados num bloco (procedimento ou função) designam-se por Variáveis (dados) locais.

Vantagens do uso de variáveis locais:

- Maior legibilidade do programa
- Facilidade de detecção de erros
- Optimização do espaço em memória

Exemplo:

```
Const max_alunos = 50
Private num_alunos as Integer       'num_alunos - variável global
                                    'valida apenas neste módulo

Public media As Double              'media - variável global
                                    'publica para todos os módulos

Public Sub calc_notas()
Do
 num_alunos = InputBox("Introduza o número de alunos")
Loop While num_alunos > max_alunos
estatistica_de_notas(num_alunos)
End Sub
```

```
Public Sub estatistica_de_notas(n_alunos As Integer)
Dim aluno As Integer     'variável local
Dim notas(max_alunos) As Integer
media = 0
For aluno = 0 To (n_alunos - 1)
 notas(aluno) = InputBox("Escreva a nota do aluno número: " & aluno)
 media = media + notas(aluno)
Next aluno
media = media / n_alunos
MsgBox "A média = " & media
End Sub
```

Parâmetros de um Procedimento

O cabeçalho de um procedimento ou função pode especificar uma lista de parâmetros formais na forma seguinte: (param, param, ... param)

Cada parâmetro pode estar numa das seguintes formas:

- ByVal *identifier* as type ‘Parâmetro passado por valor
- *identifier* as type ‘Passado por referência (mesmo que ByRef)
- Optional *identifier* as type ‘Parâmetro opcional

Quando se declara um parâmetro opcional, todos os seguintes devem também ser opcionais.

Passagem de Parâmetros por Referência para a Variável

- Utiliza-se quando o parâmetro actual é uma variável.
- Neste caso, para dentro do bloco é passado o endereço da variável.
- Utilização como **parâmetro de entrada** e **parâmetro de saída**
- Obrigatoriamente utilizado para **parâmetros de saída**

Exemplo:

```
Public Sub macro_troca()
Dim a As Integer, b As Integer
a = InputBox("Introduza o valor de a")
b = InputBox("Introduza o valor de b")
troca a, b          ' troca valor de a com b
MsgBox "Valor de a = " & a
MsgBox "Valor de b = " & b
End Sub
```

```
Public Sub troca(x As Integer, y As Integer)
 Dim aux As Integer
 aux = x
 x = y
 y = aux
End Sub
```

Passagem de Parâmetros por Valor

- Utiliza-se para transferir valores para o interior de um procedimento.
- Utilização unicamente como **parâmetro de entrada**
- Podem ser utilizadas expressões como parâmetros actuais.

Exemplo:

```
Public Sub ppm(ByRef metros As Double, ByVal pes As Integer _
, ByVal poleg As Integer)
poleg = 12 * pes + poleg
metros = poleg / 39.39
End Sub


Public Sub macro_usa_ppm()
Dim poleg As Integer
Dim m As Double
poleg = 10
ppm m, 1, poleg
MsgBox "Valor de poleg = " & poleg      'Escreve 10, pois o valor
                                        'de poleg não é alterado
MsgBox "Valor de m = " & m    'Escreve 0.558…, i.e. (12*1+10)/39.39)
End Sub
```

A passagem de parâmetros por valor utiliza dentro do procedimento, uma cópia interna dos parâmetros:

Neste caso, a variável *poleg* utilizada internamente no procedimento é uma cópia da variável de entrada definida exteriormente:

Utilização de Variáveis Indexadas como Parâmetros

A passagem também é feita por referência

Exemplo:

```
Public Sub macro_de_notas()
Const n_alunos = 3
Dim aluno As Integer
Dim notas(n_alunos) As Integer
Dim n_alunos As Integer
```

```
media = 0
For aluno = 0 To (n_alunos - 1)
 notas(aluno) = InputBox("Escreva a nota do aluno número: " & aluno)
Next aluno
mult_por_5 notas          'elementos do vector serão
                          'todos multiplicados por 5
For aluno = 0 To (n_alunos - 1)
 media = media + notas(aluno)
Next aluno
media = media / n_alunos
MsgBox "A média = " & media
End Sub

Public Sub mult_por_5(notas_alunos() As Integer)
Dim i As Integer
For i = LBound(notas_alunos) To UBound(notas_alunos)
    notas_alunos(i) = 5 * notas_alunos(i)
Next i
End Sub
```

Funções

Definição e Forma de Utilização:

- Uma Função é um bloco que retorna um valor
- A função ao ser executada, retorna um valor que deve ser utilizado numa expressão.
- O nome da função é um identificador de uma variável dependente.
- Deve existir uma única instrução de atribuição de um valor a esse identificador.

```
Public Function menor(x As Integer, y As Integer) As Integer
If x < y Then
    menor = x
Else
    menor = y
End If
End Function

Public Sub macro_usa_menor()
Dim a As Integer, b As Integer, menor_quadrado As Integer
a = InputBox("Introduza o valor de a")
b = InputBox("Introduza o valor de b")
menor_quadrado = menor(a, b) ^ 2
MsgBox "Menor valor ao quadrado = " & menor_quadrado
End Sub
```

- Funções pré-definidas na linguagem: funções intrínsecas (Exemplos: Sin(x), Abs(x), Len(str), etc);
- Funções definidas pelo utilizador: funções extrínsecas

Sintaxe:

```
[Private|Public][Static] Function nome_função (argumentos) [As tipo]
     instruções
End Function
```

Diferenças entre Funções e Procedimentos:

- O bloco duma função exprime uma regra de cálculo do valor de saída da função. Logo a chamada à função tem de ser utilizada dentro do cálculo de uma expressão no programa principal.
- Dentro do bloco da função deve existir pelo menos uma instrução de atribuição de valor ao identificador da função.
- A função, tal como qualquer variável possui um tipo (As *tipo*)

Exemplos:

```
…
b = 5 * media (h, j, media (Sin(k), 2 * k, 2#))
a = media (1.5, 3 * x, m(4, i))
…

Public Function media(x As Double, y As Double, z As Double) As
Double
        media = (x + y + z) / 3#
End Function

Public Function elevado(x As Integer, y As Integer) As Long
        elevado = Exp(y * Log(x))
End Function
```

Efeitos Colaterais de Funções

Dentro de uma função podem ser incluídas instruções não directamente relacionadas com o cálculo do seu valor (tais como escrita no ecrã, leitura de dados, alteração de variáveis globais, etc). Designam-se por efeitos colaterais de funções.

Exemplo:

```
Dim v As Integer     'variável global

Public Function f(x As Integer) As Integer
        v = v * x
        f = 2 * v + x
End Function

…
z = f(a) + v
…
```

Supondo inicialmente v=2 e a=3, então v=2*3=6 f(a)=12+3=15 z=15+6=21

O resultado esperado seria naturalmente: z=15+2=17

Para evitar efeitos colaterais de funções, todos os parâmetros de entrada devem ser passados por valor e devem se utilizadas unicamente variáveis locais dentro da função.

Recursividade

- Propriedade de um algoritmo de poder invocar a si próprio directamente ou através de outros:
- O compilador guarda numa pilha os valores dos parâmetros relativos a cada chamada recursiva.
- Deve existir uma condição que interrompa a sequência de chamadas (senão esta será infinita).

Exemplos:

```
Public Sub Inverte(ByVal n As Integer)
        MsgBox n Mod 10
        If n \ 10 <> 0 Then   'divisão inteira por 10
        Inverte (n \ 10)
        End If
End Sub

Public Function factorial(n As Integer) As Integer
 If n = 0 Then
  factorial = 1
 Else
  factorial = n * factorial(n - 1)
 End If
End Function
```

Nota: Esta solução do factorial, quer em espaço quer em tempo de execução é menos eficiente do que a solução iterativa habitual.

Problema clássico: **Torres de Hanói**: Mover n discos da haste esquerda para direita, utilizando a média como auxiliar, de forma a que nunca fique um disco maior sobre um menor. O algoritmo baseia-se em três passos:

- Mover n-1 discos da haste inicial para a media, utilizando a final como auxiliar.
- Mover o último disco para a haste final.
- Mover n-1 discos da haste auxiliar para a final, utilizando a inicial como auxiliar.

```
Public Enum haste
 esquerda    'primeira constante tem valor zero por defeito
 intermedia
 direita
End Enum

Public Sub move(n As Integer, inicial As haste _
                , auxiliar As haste, final As haste)
    If n = 1 Then
        Call escreve(inicial, final)
    Else
        Call move(n - 1, inicial, final, auxiliar)
        Call escreve(inicial, final)
        Call move(n - 1, auxiliar, inicial, final)
    End If
End Sub
```