

Algorithmi 2.0



Definição da linguagem

(C) António Manso

Luís Lopes – 19055

Raquel Guedes - 19994

Setembro 2019

Índice

[1 GAL – Generic Algorithm Language 5](#_Toc19116838)

[1.1.1 Comentários 5](#_Toc19116839)

[1.1.2 Operador de atribuição 5](#_Toc19116840)

[1.1.3 Terminadores 6](#_Toc19116841)

[1.1.4 Exemplo – Programa mínimo 7](#_Toc19116842)

[1.1.5 Exemplo – Memória Global , Programa Principal e uma Função 8](#_Toc19116843)

[2 Cálculo 9](#_Toc19116844)

[2.1 Tipos de dados 9](#_Toc19116845)

[2.2 Constantes 9](#_Toc19116846)

[2.2.1 Operadores aritméticos 10](#_Toc19116847)

[2.2.2 Operadores relacionais 10](#_Toc19116848)

[2.2.3 Operadores Lógicos 11](#_Toc19116849)

[2.2.4 Funções 11](#_Toc19116850)

[2.3 Expressões computacionais 13](#_Toc19116851)

[2.3.1 Exemplo – expressões numéricas 13](#_Toc19116852)

[2.3.2 Exemplo – Expressões textuais 13](#_Toc19116853)

[2.3.3 Exemplo – Expressões lógicas 14](#_Toc19116854)

[3 Instruções sequenciais 15](#_Toc19116855)

[3.1.1 definir variáveis 15](#_Toc19116856)

[3.1.2 ler variáveis da consola 17](#_Toc19116857)

[3.1.3 Escrever na consola 20](#_Toc19116858)

[3.1.4 Executar instruções 23](#_Toc19116859)

[3.2 Estruturas de controlo 25](#_Toc19116860)

[3.2.1 Decisão condicional 26](#_Toc19116861)

[3.2.2 Repetição com controlo inicial – Enquanto…faz 30](#_Toc19116862)

[3.2.3 Repetição com controlo final – Faz . . .Enquanto 32](#_Toc19116863)

[3.2.4 Iteração de um conjunto de valores 34](#_Toc19116864)

[3.2.5 Saltar dentro de um ciclo 37](#_Toc19116865)

[4 Funções definidas pelo utilizador 39](#_Toc19116866)

[4.1.1 Função com retorno vazio - procedimentos 40](#_Toc19116867)

[4.1.2 Função com parâmetros 41](#_Toc19116868)

[4.1.3 Retornar um valor de uma função 42](#_Toc19116869)

[4.1.4 Memória Global 44](#_Toc19116870)

[5 Arrays 46](#_Toc19116871)

[5.1.1 Exemplo 1 – Calculo e impressão de um vetor 47](#_Toc19116872)

[5.1.2 Calculo e impressão de um vetor bidimensional 48](#_Toc19116873)

[5.1.3 passagem de um vetor para uma função 49](#_Toc19116874)

1. GAL – Generic Algorithm Language

A linguagem algorítmica GAL é uma linguagem baseada em marcadores. Todos os programas são compilados e executados utilizando a GAL. Os marcadores GAL podem ser traduzidos para qualquer linguagem natural.

A linguagem foi definida tendo como principal objetivo o ensino de programação utilizando algoritmos. Os algoritmos são escritos com palavras da língua nativa dos alunos que são traduzidos e executados em GAL.

* + 1. Comentários

Os comentários são sempre introduzidos antes das instruções. Não é possível introduzir comentários no fim nem no meio das instruções.

|  |  |  |
| --- | --- | --- |
| **GAL** | **Descrição** | **Symbolo** |
| **COMMENTS** | Comentários | // |

* + 1. Operador de atribuição

O operador de atribuição serve para atribuir a uma variável definida em memória o resultado de uma expressão.

|  |  |  |
| --- | --- | --- |
| **GAL** | **Descrição** | **Symbolo** |
| **SET** | atribuição | = |

* + 1. Terminadores

Todas as instruções de um algoritmo estão compreendias entre os terminadores e são unidas por uma linha de fluxo.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | BEGIN  MEMORY  FUNCTION  END |

|  |  |
| --- | --- |
| **GAL** | **Descrição** |
| **BEGIN** | Início das instruções do programa principal |
| **MEMORY** | Início da definição das variáveis globais |
| **FUNCTION** | Início das instruções de uma função |
| **END** | Fim de um bloco de instruções. |

Todos os algoritmos têm uma função principal quês está entre os símbolos BEGIN e END.

Os algoritmos também poderão ter variáveis globais que estão definidas entre os símbolos MEMORY e END e funções definidas pelo utilizador que estão definidas entre os símbolos FUNCTION e END

O símbolo END serve para indicar o fim de uma linha de fluxo e tem associado uma classificação que indica a qual terminador inicial se reporta. Esta classificação é opcional e serve unicamente para informação sendo ignorada pelo compilador,

* + 1. Exemplo – Programa mínimo

Algoritmo que mostra o programa mínimo em GAL

|  |
| --- |
| **GAL** |
| BEGIN MAIN\_PROGRAM\_NAME  END MAIN\_PROGRAM\_NAME |

|  |  |
| --- | --- |
| **Fluxograma pt-PT** | **Algoritmo pt-PT** |
|  | inicio ProgramaPrincipal  fim ProgramaPrincipal |

**Nota:**

MAIN\_PROGRAM\_NAME – Nome do programa principal (ProgramaPrincipal).

* + 1. Exemplo – Memória Global , Programa Principal e uma Função

Algoritmo que mostra as a definição da memória global, do programa principal e de uma função

|  |
| --- |
| **GAL** |
| MEMORY GLOBAL\_MEMORY\_NAME  END GLOBAL\_MEMORY\_NAME  BEGIN MAIN\_PROGRAM\_NAME  END MAIN\_PROGRAM\_NAME  FUNCTION VOID funcao1 ROUND\_OPEN ROUND\_CLOSE  END funcao1 |

|  |  |
| --- | --- |
| **Fluxograma pt-PT** | **Algoritmo pt-PT** |
|  | memoria MemoriaGlobal  fim MemoriaGlobal  inicio ProgramaPrincipal  fim ProgramaPrincipal  funcao vazio funcao1()  fim funcao1 |

**Nota:**

GLOBAL\_MEMORY\_NAME – Nome da memória global (memoria).

MAIN\_PROGRAM\_NAME – Nome do programa principal (ProgramaPrincipal).

1. Cálculo

O cálculo em GAL é feito através de variáveis, constantes, operadores e funções.

* 1. Tipos de dados

A linguagem GAL é tipada e todas as variáveis em memória possuem um tipo. O tipo VOID é utilizado para definir uma função que não calcula valores (procedimento).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GAL** | **Descrição** | **Valores** | **pt-PT** | **en-GB** |
| **INTEGER** | Números inteiros com 64 bits | -9223372036854775808  9223372036854775807 | inteiro | integer |
| **REAL** | Números reais com 64 bits | 1.797693134862315E308  4.9E-324 | real | real |
| **LOGIC** | Valores lógicos- | verdadeiro e falso | Logico | Logic |
| **TEXT** | Conjunto de caracteres da tabela unicode | caracteres definidos entres aspas “” | Texto | Text |
| **VOID** | vazio |  | vazio | void |

* 1. Constantes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GAL** | **Descrição** | **Valor** | **pt-PT** | **en-GB** |
| **CONST\_PI** | Número PI | 3.141592653589793 | \_PI | \_PI |
| **CONST\_E** | Numero de Nepper | 2.718281828459045 | \_E | \_E |
| **CONST\_TRUE** | Valor verdadeiro | verdadeiro | \_V | \_T |
| **CONST\_FALSE** | Valor falso | falso | \_F | \_F |

* + 1. Operadores aritméticos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GAL** | **Descrição** | **Prioridade** | **Símbolo** | **Exemplo** |
| **POWER** | Potência | 33 | ^ | 2 ^ 3 = 8  2.0 ^ 3 = 8.0 |
| **MOD** | Resto da divisão | 32 | % | 3 % 2 = 1  4 % 2 = 0 |
| **DIV** | Divisão | 32 | / | 3 / 2 = 1  3.0 / 2 = 1.5 |
| **MULT** | Multiplicação | 32 | \* | 2 \* 3 = 6  2.5 \* 2 = 5.0 |
| **SUB** | Subtração | 31 | - | 3 - 2 = 1  2.5 – 1 = 1.5 |
| **SUM** | Adição | 31 | + | 3 + 2 = 5  2.5 + 1 = 3.5 |

* + 1. Operadores relacionais

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GAL** | **Descrição** | **Prioridade** | **Símbolo** | **Exemplo** |
| **LESS\_OR\_EQUAL** | Menor ou igual | 21 | <= | 3 <= 2 = Falso  1 <= 1 = Verdadeiro |
| **LESS\_THAN** | Menor | 21 | < | 3 <= 2 = Falso  1 <= 1 = Verdadeiro |
| **GREATHER\_OR\_EQUAL** | Maior ou igual | 21 | >= | 3 >= 2 = Verdadeiro  1 >= 1 = Verdadeiro |
| **GREATHER\_THAN** | Maior | 21 | > | 3 > 2 = Verdadeiro  1 > 1 = Falso |
| **DIFFERENT** | Diferente | 20 | != | 3 != 2 = Verdadeiro  1 != 1 = Falso |
| **EQUAL** | Igual | 20 | == | 3 <= 2 = Falso  1 <= 1 = Verdadeiro |

* + 1. Operadores Lógicos

|  |  |  |  |
| --- | --- | --- | --- |
| **GAL** | **Descrição** | **Prioridade** | **Símbolo e Tabela de Verdade** |
| **AND** | Disjunção – OU | 12 | |  |  |  | | --- | --- | --- | | **&&** | **V** | **F** | | **V** | V | F | | **V** | F | F | |
| **OR** | Disjunção – OU | 11 | |  |  |  | | --- | --- | --- | | **||** | **V** | **F** | | **V** | V | V | | **V** | V | F | |
| **NOT** | Disjunção – OU | 13 | |  |  |  | | --- | --- | --- | | **!** | **V** | **F** | |  | **F** | **V** | |

* + 1. Funções

As funções têm prioridade máxima nas expressões computacionais. As funções são sempre executadas antes dos operadores.

|  |  |  |
| --- | --- | --- |
| **GAL** | **Descrição** | **Definição** |
| **SIN** | Seno | real **sin**( real ) |
| **COS** | Coseno | real **cos**( real ) |
| **TAN** | Tangente | real **tan**( real ) |
| **ARC\_SIN** | Arco Seno | real **asin**( real ) |
| **ARC\_COS** | Arco Coseno | real **acos**( real ) |
| **ARC\_TAN** | Arco Tangente | real **atan**( real ) |
| **HIPERBOLIC\_SIN** | Seno hiperbólico | real **sinh**( real ) |
| **HIPERBOLIC\_COS** | Coseno hiperbólico | real **cosh**( real ) |
| **HIPERBOLIC\_TAN** | Tangente hiperbólica | real **tanh**( real ) |
| **RANDOM** | Valor aleatório no intervalo [0,1[ | real **random**( real ) |
| **ABS** | Valor absoluto | inteiro **abs**( inteiro )  real **abs**( real ) |
| **SQRT** | Raiz quadrada | real **sqrt**( real ) |
| **EXP** | Exponencia | real **exp**( real ) |
| **INT** | Valor inteiro | inteiro **int**( real ) |
| **POW** | potência | inteiro **pow**( inteiro, inteiro )  real **pow**( real, real ) |
| **MIN** | Valor mínimo | inteiro **min**( inteiro, inteiro )  real **min**( real, real ) |
| **MAX** | Valor máximo | inteiro **max**( inteiro, inteiro )  real **max**( real, real ) |
| **LOG** | Logaritmo base 10 | real **log**( real ) |
| **LN** | Logaritmo natural | real **ln**( real ) |
| **ELEMENTS\_OF** | Número de elementos | inteiro **lenght**( array )  inteiro **lenght**( texto ) |

* 1. Expressões computacionais

As expressões são um conjunto de variáveis, constantes, operadores e funções. As expressões são avaliadas da esquerda para a direita, respeitando a prioridade dos operadores, e os parêntesis curvos alteram essa prioridade.

* + 1. Exemplo – expressões numéricas

Ó resultado de uma expressão numérica é sempre um inteiro se todos os operandos forem inteiros e real se pelo menos um deles for real.

|  |  |  |
| --- | --- | --- |
| **Matemática** | **Computação** | **Tipo** |
|  | * **(2+3)/4** | **inteiro** |
|  | * **(2+3)/4.0** | **real** |
|  | * **a\*a + b\*b\*b** * **a^2 + b^3** * **pow(a,2)+pow(b,3)** | **real ou inteiro** |
|  | * **\_PI \* r \* r** * **\_PI \* r ^2** * **\_PI \* pow(r,2)** | **real** |
|  | * **(-b + sqrt( b\*b – 4\*a\*c))/(2\*a)** * **(-b + ( b^2 – 4\*a\*c)^0.5)/(2\*a)** | **real** |

* + 1. Exemplo – Expressões textuais

As expressões textuais têm apenas o operador de soma definido que possui o significado de concatenação.

|  |  |
| --- | --- |
| **Computação** | **Resultado** |
| **“OLA” + “MUNDO”** | **“OLAMUNDO”** |
| **“Var = “ + 5** | **“Var = 5”** |
| **“Var = “ + 5 \* 2** | **“Var = 10”** |
| **“Var = “ + 5 + 2 + 1** | **“Var = 521”** |
| **2+3\*4+“ Var = “ + 5** | **“14 Var = 5”** |
| **“Var = “ + (5 + 2 + 1)** | **“Var = 8”** |

* + 1. Exemplo – Expressões lógicas

As expressões lógicas possuem variáveis lógicas, operadores relacionais ou operadores lógicos.

|  |  |
| --- | --- |
| **Computação** | **Resultado** |
| **1 > 2** | **Falso** |
| **1 < 2 && 2 > 0** | **Verdadeiro** |

1. Instruções sequenciais

As instruções sequenciais são intercaladas nas linhas de fluxo. Executam-se e passam para a instrução seguinte.

* + 1. definir variáveis

Define uma ou mais variáveis em memória. Diz-se variável porque o seu conteúdo pode variar, mas nunca o seu nome ou o seu tipo.

A definição de uma variável obriga a fazer a sua inicialização com uma expressão que seja compatível com o tipo de dados da variável, a multi-instrução veio permitir que num só comando se possam repetir múltiplas definições e inicializações destas.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **DEFINE <TYPE> <var> [= <expression>][, <var> [= <expressão>] …]** |

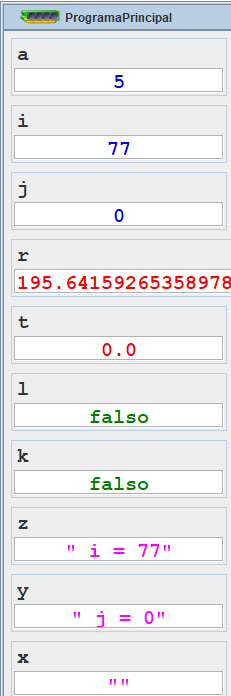
* **<TYPE>** – Tipo de dados (INTEGER | REAL | LOGIC | TEXT)
* **<var>** – Nome da variável sujeita às seguintes regras
  + Não pode ser uma palavra reservada
  + Tem de começar por uma letra (a-z; A-Z)
  + Pode conter dígitos (0-9)
  + Deve ser significativo daquilo que armazena
* **[<expression>]** - expressão computacional cujo tipo de resultado é compatível com o tipo de dados da variável, esta é opcional pois se não for inserida a variável irá tomar o valor por defeito. [**Opcional]**

#### Exemplo – Definir variáveis

Algoritmo que mostra as a definição de variáveis dos tipos presentes no Algorithmi.

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER a SET 2 SUM 3  DEFINE INTEGER i SET 2 SUM 3 MULT 5 POWER 2 ,j SET 0  DEFINE REAL r SET CONST\_PI SUM 2.5 MULT i ,t SET 0.0  DEFINE LOGIC l SET FALSE ,k SET FALSE  DEFINE TEXT z SET " i = " SUM i ,y SET " j = " SUM j ,x SET ""  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | **inicio ProgramaPrincipal**  **definir inteiro a = 2 + 3**  **definir inteiro i = 2 + 3 \* 5 ^ 2, j**  **definir real r = \_PI + 2.5 \* i, t**  **definir logico l = falso, k**  **definir texto z = " i = " + i, y = " j = " + j, x**  **fim ProgramaPrincipal** |

**Resultado:**



* + 1. ler variáveis da consola

Lê uma ou mais variáveis definida na memória a partir da consola. A instrução também permite definir novas variáveis para a memória e cujo valor é inicializado pela consola. É também possível definir uma mensagem para ser apresentada quando o valor for pedido.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **READ [<TYPE>] <var>[,<var> …] [<message>]** |

* **[<TYPE>]** –Tipo de dados (INTEGER **| REAL | LOGIC | TEXT**) **[Opcional]**
* **<var>** – Nome da variável definida ou a definir em memória. O nome da variável deve seguir as seguintes regras:
  + Não pode ser uma palavra reservada
  + Tem de começar por uma letra (a-z; A-Z)
  + Pode conter dígitos (0-9)
  + Deve ser significativo daquilo que armazena
* **[<message>] –** mensagem a ser apresentada antes da leitura das variáveis **[Opcional]**

#### Exemplo – Ler variáveis

Algoritmo que mostra algumas versões da instrução de leitura:

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER a SET 0, b SET 0  READ a  READ INTEGER c  READ a "Insira a: "  READ INTEGER h "Insira h: "  READ a COMMA b  READ INTEGER d COMMA e  READ a COMMA b "insira a e b: "  READ INTEGER f COMMA g "insira f e g: "  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | **inicio ProgramaPrincipal**  **definir inteiro a = 0,b = 0**  **ler a**  **ler inteiro c**  **ler a "Insira a: "**  **ler inteiro h "Insira h: "**  **ler a,b**  **ler inteiro d,e**  **ler a,b "insira a e b: "**  **ler inteiro f,g "insira f e g: "**  **fim ProgramaPrincipal** |

**Resultado:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

* + 1. Escrever na consola

Escreve o valor de uma ou mais expressões na consola. As expressões podem ser constituídas por variáveis simples, e neste caso escreve o seu conteúdo, ou uma expressão complexa e neste caso escreve o resultado depois de ser convertido para texto

Tabela - Multi-Instrução com o símbolo write

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **WRITE <expression>[, <expression> …]** |

* **<expression>**– expressão computacional

#### Exemplo – Escrever na consola

Algoritmo que exemplifica a instrução de escrever em multi-instrução e sem multi-instrução: Escreve uma constante (“olá”) uma variável (txt) e uma expressão computacional (“ “ + version + “!”).

Tabela - Exemplo de algoritmo de multi-instrução com o símbolo write

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE TEXT txt SET "Algorithmi"  DEFINE REAL version SET 2.0  WRITE "Olá " COMMA txt COMMA " " SUM version SUM "!\n\n"  WRITE "Olá "  WRITE txt  WRITE " " SUM version SUM "!"  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | **inicio ProgramaPrincipal**  **definir texto txt = "Algorithmi"**  **definir real version = 2.0**  **escrever "Olá ", txt, " " + version + "!\n\n"**  **escrever "Olá "**  **escrever txt**  **escrever " " + version + "!"**  **fim ProgramaPrincipal** |

**Resultado:**

Tabela - Resultado em memória e consola depois de executado o exemplo de multi-instrução do símbolo write.

|  |  |
| --- | --- |
|  |  |

* + 1. Executar instruções

Serve para executar cálculos em expressões computacionais e guardar o seu resultado numa variável definida em memória.

Também serve para executar procedimentos, isto é, função que não calculam valores.

Este com a multi-instrução permite fazer múltiplos cálculos numa só forma.

Tabela - Multi-Instrução com o símbolo execute

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **EXECUTE <function>** | **<var> = <expression>[, <function> | <var> = <expression> …]** |

* **<var>** – variável definida em memória
* **<expression>** – expressão computacional
* **<function>** – função definida pelo utilizador

O símbolo “|” representa “ou”, isto significa que pode ser **<function>** ou **<var> = <expression>**

#### Exemplo – Definir, ler, calcular e imprimir

Algoritmo que exemplifica solicita ao utilizador o raio de um círculo e que de seguida calcula o perímetro, área e diâmetro e imprime o resultado na consola.

Tabela - Exemplo de algoritmo de multi-instrução com o símbolo execute

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE REAL perimetro SET 0.0 ,area SET 0.0 ,diametro SET 0.0  READ REAL raio "Insira o raio do circulo: "  EXECUTE perimetro SET 2 MULT CONST\_PI MULT raio COMMA area SET CONST\_PI MULT POW ROUND\_OPEN raio COMMA 2 ROUND\_CLOSE COMMA diametro SET 2 MULT raio  WRITE "raio: " SUM raio SUM "\n" COMMA "perimetro: " SUM perimetro SUM "\n" COMMA "area: " SUM area SUM "\n" COMMA "diametro: " SUM diametro  END | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | **inicio ProgramaPrincipal**  **definir real perimetro = 0.0,area = 0.0,diametro = 0.0**  **ler real raio "Insira o raio do circulo: "**  **executar perimetro = 2 \* \_PI \* raio , area = \_PI \* pow(raio , 2) , diametro = 2 \* raio**  **escrever "raio: " + raio + "\n" , "perimetro: " + perimetro + "\n" , "area: " + area + "\n" , "diametro: " + diametro**  **fim ProgramaPrincipal** |

**Resultado:**

Tabela - Resultado em memória e consola depois de executado o exemplo de multi-instrução do símbolo execute.

|  |  |
| --- | --- |
|  |  |

* 1. Estruturas de controlo

De seguida são apresentadas as instruções que permitem desviar o fluxo do algoritmo através de uma expressão condicional. Uma expressão condicional é uma expressão que devolve um valor lógico. Usualmente estas expressão contêm operadores relacionais, operadores lógicos ou variáveis lógicas

* + 1. Decisão condicional

Permite dividir o fluxo e dois baseado numa condição

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **IF <logic expression>**  **<instructions>**  **[ ELSE ]**  **<instructions>**  **END** |

* **<logic expression>** – expressão lógica
* **<instructions>** – instruções do algoritmo

O simbolo ELSE da estrutura de decisão é opcional e as estruturas de decisão podem ser encadeadas

Exemplo 1 – Valor absoluto

Algoritmo que solicita um número inteiro ao utilizador e imprime o seu valor absoluto

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  WRITE "Introduza um número :"  READ INTEGER n  IF n LESS\_THAN 0  EXECUTE n SET n MULT -1  END IF  WRITE " valor absoluto = " SUM n  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  escrever "Introduza um número :"  ler inteiro n  se n < 0 então  executar n = n \* -1  fim se  escrever " valor absoluto = " + n  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Introduza um número :-5  valor absoluto = 5  ::::::::::::::::::::::::::::::::::::::::: |

Exemplo 2 – Par ou ímpar

Algoritmo que solicita um número inteiro ao utilizador e imprime se o número é ímpar ou par.

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  WRITE "Introduza um número :"  READ INTEGER n  IF n MOD 2 EQUAL 0  WRITE "número par"  ELSE  WRITE "número ímpar"  END IF  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  escrever "Introduza um número :"  ler inteiro n  se n % 2 == 0 então  escrever "número par"  senão  escrever "número ímpar"  fim se  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Introduza um número :5  número ímpar  ::::::::::::::::::::::::::::::::::::::::: |

Exemplo 3 – Positivo negativo ou nulo

Algoritmo que solicita um número inteiro ao utilizador e imprime se o número é positivo negativo ou nulo

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  WRITE "Introduza um número :"  READ INTEGER n  IF n EQUAL 0  WRITE "valor nulo"  ELSE  IF n GREATHER\_THAN 0  WRITE "Valor positivo"  ELSE  WRITE "Valor negativo"  END IF  END IF  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  escrever "Introduza um número :"  ler inteiro n  se n == 0 então  escrever "valor nulo"  senão  se n > 0 então  escrever "Valor positivo"  senão  escrever "Valor negativo"  fim se  fim se  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Introduza um número :5  Valor positivo  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Repetição com controlo inicial – Enquanto…faz

Permite repetir um conjunto de instruções baseadas numa condição que é avaliada no início da repetição.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **WHILE <logic expression>**  **<instructions>**  **END** |

* **<logic expression>** – expressão lógica
* **<instructions>** – instruções do algoritmo

Exemplo – contar por ordem decrescente

Algoritmo que imprime na consola os números inteiros entre 10 e 1

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER conta SET 10  WHILE conta GREATHER\_THAN 0  WRITE conta SUM " "  EXECUTE conta SET conta SUB 1  END WHILE  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  definir inteiro conta = 10  enquanto conta > 0 faz  escrever conta + " "  executar conta = conta - 1  fim enquanto  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  10 9 8 7 6 5 4 3 2 1  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Repetição com controlo final – Faz . . .Enquanto

Permite repetir um conjunto de instruções baseadas numa condição que é avaliada no final da repetição.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **DO**  **<instructions>**  **WHILE <logic expression>** |

* **<logic expression>** – expressão lógica
* **<instructions>** – instruções do algoritmo

Exemplo 1 – Contar por ordem crescente

Algoritmo que imprime na consola os números inteiros entre 1 e 10

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER conta SET 10  WHILE conta GREATHER\_THAN 0  WRITE conta SUM " "  EXECUTE conta SET conta SUB 1  END WHILE  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  definir inteiro conta = 1  faz  escrever conta + " "  executar conta = conta + 1  enquanto conta <= 10  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  1 2 3 4 5 6 7 8 9 10  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Iteração de um conjunto de valores

Permite iterar um conjunto de valores dentro do intervalo [ex1 , ex2[ onde os valores estão separados por ex3. O limite inicial de iterador está incluído e o limite final não. A instrução permite definir uma variável de iteração ou utilizar uma variável numérica já existente.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **ITERATE [<Type>] <var> FROM <ex1> TO <ex2> STEP <ex3>**  **<instructions>**  **END** |

* **[<Type>]** – tipo de variável numérica ( **INTEGER | REAL**) - Opcional
* **<var>** - nome da variável de iteração
* **<ex1>** – expressão computacional para determinar o valor inicial
* **<ex2>** – expressão computacional para determinar o valor final
* **<ex3>** – expressão computacional para determinar a separação entre os valores do intervalo

Exemplo 1 – Contar por ordem crescente

Algoritmo que imprime na consola os números inteiros entre 1 e 10

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  ITERATE INTEGER i FROM 1 TO 11 STEP 1  WRITE i SUM " "  END ITERATE  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  iterar inteiro i de 1 até 11 passo 1  escrever i + " "  fim iterar  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  1 2 3 4 5 6 7 8 9 10  ::::::::::::::::::::::::::::::::::::::::: |

NOTA:

Não existe nenhuma variável definida no final do ciclo porque o iterador i é interno ao ciclo.

Exemplo 1 – Contar por ordem decrescente

Algoritmo que imprime na consola os números inteiros entre 10 e 1

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER conta SET 0  ITERATE conta FROM 10 TO 0 STEP -1  WRITE conta SUM " "  END ITERATE  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  definir inteiro conta = 0  iterar conta de 10 até 0 passo -1  escrever conta + " "  fim iterar  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  10 9 8 7 6 5 4 3 2 1  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Saltar dentro de um ciclo

Instrução que permite quebrar ou continuar um ciclo..

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **JUMP\_BREAK**  **JUMP\_CONTINUE** |

A instrução JUMP\_BREAK quebra o ciclo quando é executada, ou seja, sai imediatamente do ciclo.

A instrução JUMP\_CONTINUE continua o ciclo quando é executada, ou seja, salta imediatamente para a condição de controlo.

Exemplo – Solicitar uma hora válida

Algoritmo solicita ao utilizador uma hora válida

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER conta SET 0  ITERATE conta FROM 10 TO 0 STEP -1  WRITE conta SUM " "  END ITERATE  END MAIN\_PROGRAM\_NAME | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio ProgramaPrincipal  definir inteiro hora = 0  enquanto verdadeiro faz  escrever "Hora : "  ler hora  se hora < 0 || hora > 23 então  escrever "Hora errada\n"  continuar  senão  quebrar  fim se  escrever "nunca executado!"  fim enquanto  fim ProgramaPrincipal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Hora : -4  Hora errada  Hora : 111  Hora errada  Hora : 23  ::::::::::::::::::::::::::::::::::::::::: |

1. Funções definidas pelo utilizador

As funções no algorithmi funcionam como sub algoritmos que podem ser evocadas dentro das instruções.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **FUNCTION <type> <name> ([<param>,..,<param>])**  **<instructions>**  **END name** |

* **<type >** – tipo de cálculo executado pela função
  + **INTEGER | REAL | TEXT | LOGIC | VOID**
    - **VOID** significa que não calcula nada
* **< name>** - Nome da função sujeita ás seguintes regras:
  + Não pode ser uma palavra reservada ou de outra função
  + Tem de começar por uma letra (a-z ; A-Z)
  + Pode conter dígitos (0-9)
  + Deve ser significativo daquilo que armazena
* **[<param>]) –** Parâmetros da função (**Opcional**)
  + **<type> –** Tipo de dados do parâmetro
  + **<var>]-** Nome do parâmetros
    1. Função com retorno vazio - procedimentos

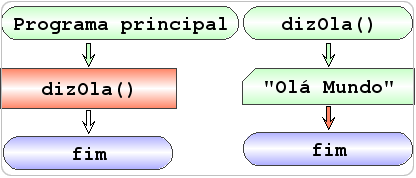
As funções com retorno vazio, usualmente chamadas de procedimentos, servem para executar um conjunto de instruções. Estas funções só podem ser evocadas a partir de um comando de EXECUTE

Exemplo 1 – Dizer Olá

Algoritmo que imprime na consola os Olá mundo através de uma função

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  EXECUTE dizOla ROUND\_OPEN ROUND\_CLOSE  END MAIN\_PROGRAM\_NAME  FUNCTION VOID dizOla ROUND\_OPEN ROUND\_CLOSE  WRITE "O | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio Programa principal  executar dizOla()  fim Programa principal  funcao vazio dizOla()  escrever "Olá Mundo"  fim dizOla |

**Resultado:**

****

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Olá Mundo  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Função com parâmetros

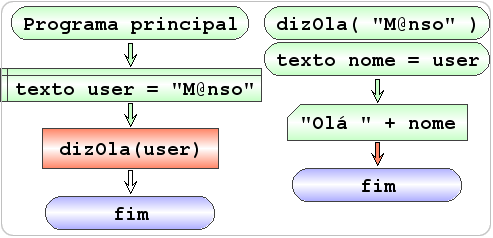
Os parâmetros de uma função funcionam como variáveis locais da função e servem para flexibilizar a sua utilidade.

Exemplo – Dizer Olá com Cálculo

Algoritmo que imprime na consola os Olá mundo através de uma função

|  |  |
| --- | --- |
| **GAL** | |
| BEGIN MAIN\_PROGRAM\_NAME  EXECUTE dizOla ROUND\_OPEN ROUND\_CLOSE  END MAIN\_PROGRAM\_NAME  FUNCTION VOID dizOla ROUND\_OPEN ROUND\_CLOSE  WRITE "O | |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio Programa principal  definir texto user = "M@nso"  executar dizOla(user)  fim Programa principal  funcao vazio dizOla(texto nome)  escrever "Olá " + nome  fim dizOla |

**Resultado:**

****

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Olá M@nso  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Retornar um valor de uma função

As funções podem executar cálculos que são retornados para o local onde a função foi evocada. A Instrução RETURN permite devolver um valor da função. Sempre que esta instrução é executada o função termina.

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **RETURN <expression>** |

* **< expression >** – expressão computacional compatível com o tipo de dados de retorno da função.
  + **Se a função tiver o retorno Vazio , a expressão deve ser vazia**.

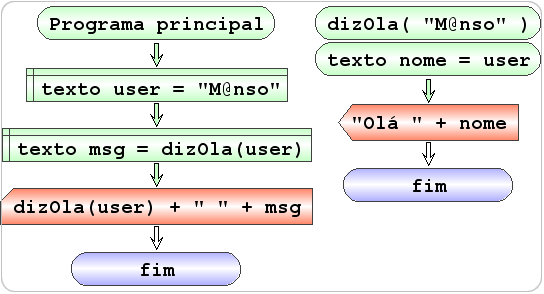
Exemplo 1 – Dizer Olá com parâmetros

Algoritmo que imprime na consola os Olá mundo através de uma função

|  |
| --- |
| **GAL** |
| BEGIN MAIN\_PROGRAM\_NAME  EXECUTE dizOla ROUND\_OPEN ROUND\_CLOSE  END MAIN\_PROGRAM\_NAME  FUNCTION VOID dizOla ROUND\_OPEN ROUND\_CLOSE  WRITE "O |

|  |  |
| --- | --- |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio Programa principal  definir texto user = "M@nso"  definir texto msg=dizOla(user)  escrever dizOla(user)+" "+msg  fim Programa principal  funcao texto dizOla(texto nome)  retornar "Olá " + nome  fim dizOla |

**Resultado:**



|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Olá M@nso Olá M@nso  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Memória Global

Cada função possui a sua memória que pode ser utilizada pelas suas instruções. No entanto o algorithmi permite definir uma memória global que é comum a todas as funções e pode por elas ser utilizada

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **MEMORY**  **<define>DEFINE REAL ver SET 1.0**  **END** |

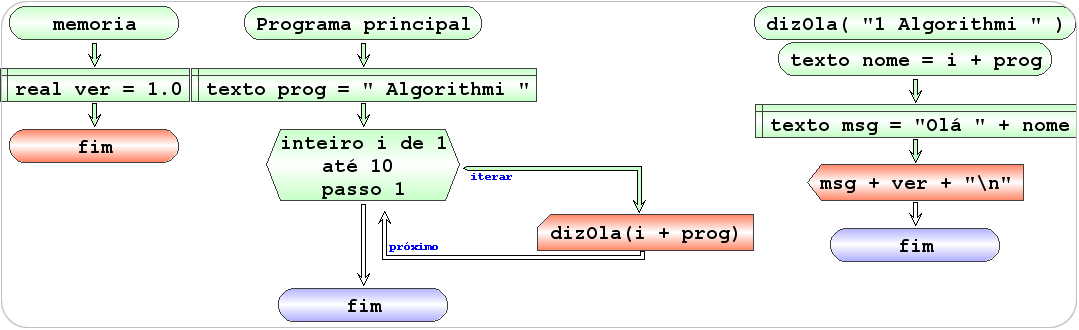
* **<define>** – Definição das variáveis globais.
  + **Ver definição de variáveis**.

Exemplo 1 – Utilização da memória global

|  |
| --- |
| **GAL** |
| MEMORY GLOBAL\_MEMORY\_NAME  DEFINE REAL ver SET 1.0  END GLOBAL\_MEMORY\_NAME  BEGIN MAIN\_PROGRAM\_NAME  DEFINE TEXT prog SET " Algorithmi "  ITERATE INTEGER i FROM 1 TO 10 STEP 1  WRITE dizOla ROUND\_OPEN i SUM prog ROUND\_CLOSE  END ITERATE  END MAIN\_PROGRAM\_NAME  FUNCTION TEXT dizOla ROUND\_OPEN TEXT nome ROUND\_CLOSE  DEFINE TEXT msg SET "Olá " SUM nome  RETURN msg SUM ver SUM "\n"  END dizOlaO |

|  |  |
| --- | --- |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio memoriaGlobal  definir real ver = 1.0  fim memoriaGlobal  inicio Programa principal  definir texto prog = " Algorithmi "  iterar inteiro i de 1 até 10 passo 1  escrever dizOla(i + prog)  fim iterar  fim Programa principal  funcao texto dizOla(texto nome)  definir texto msg = "Olá " + nome  retornar msg + ver + "\n"  fim dizOla |

**Resultado:**

****

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Olá 1 Algorithmi 1.0  Olá 2 Algorithmi 1.0  Olá 3 Algorithmi 1.0  Olá 4 Algorithmi 1.0  ::::::::::::::::::::::::::::::::::::::::: |

1. Arrays

Define uma variável indexada em memória. As variáveis indexadas são conjuntos homogéneos de dados que são acessíveis através de índices.

O Algorithmi permite definir o número de dimensões da variável indexada que o utilizador desejar.

Na definição de um vetor são definidas o número de dimensões e a quantidade de elementos em cada dimensão. As variáveis indexadas são inicializadas com os valores por defeito:

|  |  |
| --- | --- |
| **Fluxograma** | **GAL** |
|  | **DEFINE <TYPE> <var> [<ex>]…[<ex>]** |

* **<TYPE>** – Tipo de dados ( INTEGER | REAL | LOGIC | TEXT )
* **<var>** – Nome da variável sujeita ás seguintes regras
  + Não pode ser uma palavra reservada
  + Tem de começar por uma letra (a-z ; A-Z)
  + Pode conter dígitos (0-9)
  + Deve ser significativo daquilo que armazena
* **[<ex>]** - expressão computacional cujo resultado é um inteiro positvo
  + 1. Exemplo 1 – Calculo e impressão de um vetor

|  |
| --- |
| **GAL** |
| BEGIN MAIN\_PROGRAM\_NAME  DEFINE INTEGER vetor SQUARE\_OPEN 5 SQUARE\_CLOSE  ITERATE INTEGER i FROM 0 TO ELEMENTS\_OF ROUND\_OPEN vetor ROUND\_CLOSE STEP 1  EXECUTE vetor SQUARE\_OPEN i SQUARE\_CLOSE SET i MULT 10  WRITE "\nVetor[" SUM i SUM "] = " SUM vetor SQUARE\_OPEN i SQUARE\_CLOSE  END ITERATE  END MAIN\_PROGRAM\_NAME |

|  |  |
| --- | --- |
| **Fluxograma pt-PT** | **Pseudocódigo pt-PT** |
|  | inicio Programa principal  definir inteiro vetor[5]  iterar inteiro i de 0 até length(vetor) passo 1  executar vetor[i] = i \* 10  escrever "\nVetor[" + i + "] = " + vetor[i]  fim iterar  fim Programa principal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  Vetor[0] = 0  Vetor[1] = 10  Vetor[2] = 20  Vetor[3] = 30  Vetor[4] = 40  ::::::::::::::::::::::::::::::::::::::::: |

* + 1. Calculo e impressão de um vetor bidimensional

|  |  |
| --- | --- |
| **Fluxograma pt-PT** | |
|  | |
| **Pseudocódigo pt-PT** |
| inicio Programa principal  definir inteiro array[8][4]  iterar inteiro i de 0 até length(array) passo 1  iterar inteiro j de 0 até length(array[i])  passo 1  executar array[i][j] = i \* 10 + j  escrever "\t" + array[i][j]  fim iterar  escrever "\n"  fim iterar  fim Programa principal |

**Resultado:**

|  |  |
| --- | --- |
|  | :::::::::::::::::::::::::::::::::::::::::  0 1 2 3  10 11 12 13  20 21 22 23  30 31 32 33  40 41 42 43  50 51 52 53  60 61 62 63  70 71 72 73  ::::::::::::::::::::::::::::::::::::::::: |

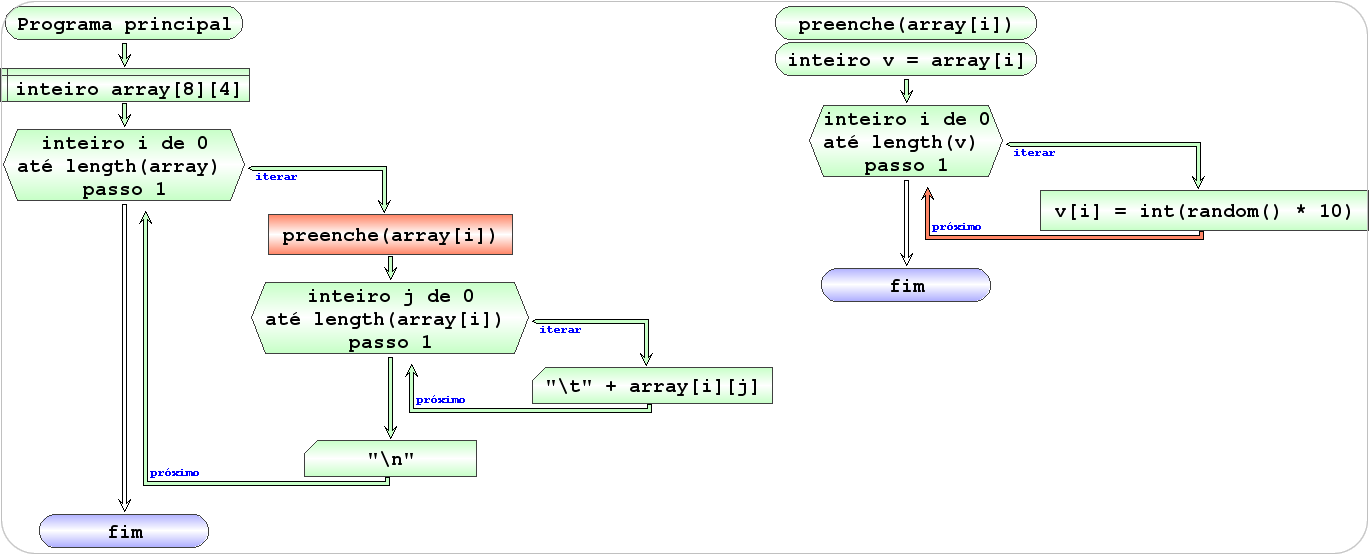
* + 1. passagem de um vetor para uma função

Os vetores são passados para as funções sempre por referência. As alterações feitas no vetor passado como parâmetros refletem-se no vetor principal.

|  |
| --- |
| **Fluxograma pt-PT** |
|  |

|  |
| --- |
| **Pseudocódigo pt-PT** |
| inicio Programa principal  definir inteiro array[8][4]  iterar inteiro i de 0 até length(array) passo 1  executar preenche(array[i])  iterar inteiro j de 0 até length(array[i]) passo 1  escrever "\t" + array[i][j]  fim iterar  escrever "\n"  fim iterar  fim Programa principal  funcao vazio preenche(inteiro v[])  iterar inteiro i de 0 até length(v) passo 1  executar v[i] = int(random() \* 10)  fim iterar  fim preenche |

**Resultado:**

****

|  |  |
| --- | --- |
|  | ::::::::::::::::::::::::::::::::::::::::: :::::::::::::::::::::::::::::::::::::::::  5 3 0 9  1 5 7 1  6 5 2 8  7 8 4 4  7 2 1 2  2 0 3 1  7 7 1 7  ::::::::::::::::::::::::::::::::::::::::: ::::::::::::::::::::::::::::::::::::::::: |