Programación en Pascal. Arrays

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Enero de 2021

Contenidos

Arrays

Un array, también llamado vector, es

- una colección de elementos
- del mismo tipo
- ordenados. A cada elemento le corresponde un índice, normalmente un número natural

Otros nombres para esta estructura son: tabla, colección indexada

Empecemos por un ejemplo muy básico. Funciona aunque necesita mejoras

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_01;

var
    vector : array[1..3] of string;  // Número mágico (de momento)

begin
    vector[1] := 'Primero';
    vector[2] := 'Segundo';
    vector[3] := 'Tercero';

    writeln(vector[1]);
    writeln(vector[2]);
    writeln(vector[3]);
end.
```

```
var
    vector : array[1..3] of string;  // Número mágico (de momento)
```

- Declaramos `vector` como `array`
- Entre corchetes indicamos el ordinal de su primer elemento y el ordinal del último, separados por `..`
- Como es costumbre en Pascal, en este ejemplo el primer elemento es el 1. Podríamos usar el 0, pero resulta menos *idiomático*. También podríamos usar cualquier otro número positivo o negativo, con tal de que sea entero
- Despues de la palabra reservada `of`, indicamos el tipo de datos que tendrá cada elemento del array
- Para referirnos a un elemento del array, escribimos el nombre del array y añadimos entre corchetes el índice

En este ejemplo usamos un índice que no va de 1 a n

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_02;

var
    vector : array[-1..1] of string;

begin
    vector[-1] := 'Sotano';
    vector[0] := 'Planta baja';
    vector[1] := 'Primero';

    writeln(vector[-1]);
    writeln(vector[0]);
    writeln(vector[1]);
end.
```

Normalmente usaremos un bucle para recorrer el array

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_03;

var
    vector : array[1..3] of string;
    i : integer;

begin
    vector[1] := 'Primero';
    vector[2] := 'Segundo';
    vector[3] := 'Tercero';

    for i := 1 to 3 do
        writeln(vector[i]);
end.
```

También podemos inicializar un array (variable o constante) enumerando todos sus elementos, entre paréntesis

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_03bis;

var
    vector : Array[1..3] of string = ('Primero', 'Segundo', 'Tercero');
    i : integer;

begin
    for i:= 1 to 3 do
        writeln(vector[i]);
end.
```

Si intentamos acceder a una posición fuera de rango, obtendremos un error de compilación (gracias a las directivas de compilación que ponemos en cada programa)

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_mal_01;

var
    vector : array[1..3] of string;

begin
    vector[4] := 'Cuarto'; //  ¡Mal!
end.
```

Resultado:

```
Compiling array_mal_01.pas
array_mal_01.pas(8,11) Error: range check error while evaluating constants
(4 must be between 1 and 3)
array_mal_01.pas(10) Fatal: There were 1 errors compiling module, stopping
Fatal: Compilation aborted
```

Pero si no ponemos directivas al compilador para que detecte los desbordamientos de rango, el programa compilará (con un *warning*) e incluso se ejecutará.
Será un programa vulnerable a todo tipo de problemas de seguridad, una bomba de relojería en potencia

```
// Sin directivas de compilación
program array_mal_02;

var
    vector : array[1..3] of string;

begin
    vector[4] := 'Cuarto';
    writeln(vector[4]);
end.
```

Resultado:

```
Cuarto
```

En los ejemplos anteriores, declarábamos la variable con un número mágico. Esto es incorrecto, deberíamos usar una constante

- En otros casos podríamos declarar y definir esta constante así:

```
const
    TamanioVector : integer = 3;  // ¡Mal en este caso!
```

- Pero como esta constante la usaremos para declarar una variable (será el tamaño del array) tenemos que definirla (darle su valor) pero no declararla (indicar su tipo):

```
const
    TamanioVector = 3;   // Correcto
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_04;
const
    TamanioVector = 3;

var
    vector : array[1..TamanioVector] of string;
    i : integer;

begin
    vector[1] := 'Primero';
    vector[2] := 'Segundo';
    vector[3] := 'Tercero';

    for i := 1 to TamanioVector do
        writeln(vector[i]);
end.
```

Hasta ahora declarábamos las variables directamente como arrays del tamaño correspondiente.

```
var
    vector : array[1..TamanioVector] of string;
```

Pero esto es un problema en potencia. Si necesitamos cambiar ese tipo de datos, tendremos que tocar muchas variables y muchos parámetros. Lo adecuado es declarar un nuevo tipo de datos. En este caso, `TipoVector`

- Por convenio, estos nombres de tipo empezarán por el prefijo *Tipo* y estarán escritos en *NotacionCamello* [1]

```
const
    TamanioVector = 3;
type
    TipoVector = array[1..TamanioVector] of string;
var
    vector : TipoVector;
```

[1] Juntar dos palabras, de forma que cada una empiece por mayúscula

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_05;
    // Declaramos un tipo para el array

const
    TamanioVector = 3;

type
    TipoVector = array[1..TamanioVector] of string;

var
    vector : TipoVector;
    i : integer;

begin
    vector[1] := 'Primero';
    vector[2] := 'Segundo';
    vector[3] := 'Tercero';

    for i := 1 to TamanioVector do
        writeln(vector[i]);
end.
```

Usemos un vector de enteros para almacenar las tiradas de un dado

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_06;  // Guardamos la salida de un dado
uses crt;
const
    TamanioVector = 5;
type
    TipoVector = array[1..TamanioVector] of integer;
// Definimos tira_dado() como en el tema 7
var
    i: integer;
    vector : TipoVector;
const
    Caras_dado = 6;
```

```
begin
    delay(800);
    randomize;

    for i := 1 to TamanioVector do begin
        vector[i] := tira_dado(Caras_dado);
    end;

    for i := 1 to TamanioVector do begin
        write(vector[i], ' ');
    end;
    writeln;
end.
```

Resultado:

```
4 5 2 5 3
```

Clasificación de problemas

Recapitulemos:

- En los temas 2 y 3, vimos cómo resolver problemas cuya solución era directamente una expresión (una *fórmula*)
- En el tema 4 trabajamos en la resolución de problemas donde era necesario considerar distintos casos por separado
- En el tema 7 tratamos los bucles, estructura básica para poder trabajar con arrays. Los problemas con arrays a su vez podemos subdividirlos en
  1. Problemas de acumulación de valores
  2. Problemas de búsqueda
  3. Problemas de maximización

Esta taxonomía es completa. Aplicando esta familia de técnicas (y sus combinaciones) podremos resolver cualquier problema de programación que se nos plantee

Problemas de acumulación

Calculemos la media aritmética de un vector de reales

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program media_vector;
const
    TamanioVector = 3;
type
    TipoVector = array[1..TamanioVector] of real;

function suma_vector(vector: TipoVector): real;
var
    suma: real;
    i: integer;
begin
    suma := 0.0;
    for i := 1 to TamanioVector do begin
        suma := suma + vector[i];
    end;
    result := suma;
end;
```

```
function media(vector: TipoVector): real;
begin
    result := suma_vector(vector) / TamanioVector
end;
const
    Vector_prueba: TipoVector = (115.4, 121.9, 111.9);
begin
    writeln(media(Vector_prueba) :0:2)
end.
```

Resultado:

```
116.40
```

Problemas de búsqueda

Vamos a buscar un valor concreto en nuestro array

- Una función, `busca_duracion` recorrerá todo el array comparando cada valor con el deseado
- Si lo encuentra, devolverá su posición
- Si no lo encuentra, devolverá un valor especial. Un valor que sabemos que nunca puede ser una posición, por lo que podemos darle el significado *no encontrado*
  - Como estamos empezando a contar nuestros arrays desde 1, el 0 es una posición *imposible*. Por tanto podemos darle el significado *valor no encontrado*
  - En otras circunstancias podríamos usar -1 o constantes especiales como `nil`

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program busqueda_real_mal;
    // Búsqueda ERRONEA de un real en un array

const
    TamanioVector = 3;
type
    TipoVector = array[1..TamanioVector] of real;

function busca_valor(vector: TipoVector; valor: real): integer;
    // Esta función busca un valor en un vector y devuelve su
    // posición. Si no existe, devuelve 0
var
    i: integer;
begin
    result := 0;
    for i := 1 to TamanioVector do begin
        if vector[i] = valor then
            result := i
    end
end;
```

```
const
    vector_prueba: TipoVector = (115.4, 121.9, 111.9);
    tiempo_a_buscar: real = 121.900001;
begin
    writeln(busca_valor(vector_prueba, tiempo_a_buscar))
end.
```

Resultado:

0

El programa no ha encontrado un valor exactamente igual al buscado, aunque hay otro apenas una millonésima menor

Observa que en la función de búsqueda NO hemos hecho

```
if LO_ENCUENTRO then
    result := LA_POSICION
else
    result := 0
```

Sino que hemos seguido este otro esquema, que es muy normal en problemas de búsqueda:

```
De momento no lo he encontrado
Busco por todo el array
    Si aparece, lo he encontrado
```

En caso de que no se encuentre nunca, la sentencia *de momento no lo he encontrado* se convierte en definitiva, porque ninguna otra altera el valor

El ejemplo anterior, *busqueda_real_mal* sería correcto para buscar enteros, cadenas, enumerados, etc

- Pero no podemos buscar así un número real
- Es muy frecuente que errores en el redondeo de las expresiones matemáticas provoquen errores muy pequeños, del orden de 1^{-12} o inferiores
- Estos errores son suficientes para que dos números reales se consideren distintos cuando en la práctica son iguales

Para buscar un número real, siempre tenemos que comprobar que la diferencia entre el número a considerar y el número objetivo sea menor a cierto margen

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program busqueda_real;
    // Buscamos un valor real en un vector
const
    TamanioVector = 4;
type
    TipoVector = array[1..TamanioVector] of real;

function busca_valor(vector: TipoVector; valor, margen: real): integer;
    // Esta función busca un valor real en un vector, permitiendo
    // cierto margen. Devuelve su posición, o 0 si no existe
var
    i: integer;
begin
    result := 0;
    for i := 1 to TamanioVector do begin
        if abs(vector[i] - valor) <= margen then
            result := i
    end
end;
```

```
const
    Vector_prueba: TipoVector = (93.0, 86.9, 0, 86.2);
    Tiempo_a_buscar: real = 86.873;
    Margen: real = 0.03;
begin
    writeln(busca_valor(Vector_prueba, Tiempo_a_buscar, Margen))
end.
```

Resultado:

2

El ejemplo anterior ya puede ser aceptable, pero se puede mejorar

- En todos los casos recorríamos todos los valores
- Pero si encontramos un valor que se corresponda con lo solicitado, ya no es necesario seguir buscando

Este tipo de mejoras solo merecen la pena si estamos seguros de que el ahorro de tiempo es importante (hay muchos datos). En otro caso, el tiempo de programador suele ser más valioso

- A continuación veremos una versión que comete un error al intentar aplicar esta mejora

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program busqueda_real_mal_02;
    // Buscamos en un vector un valor real lo baste próximo a un
    // valor pedido. Si lo encontramos, dejamos de buscar.
    // Versión errónea, olvidamos el caso de que no exista

const
    TamanioVector = 4;
type
    TipoVector = array[1..TamanioVector] of real;
```

```
function busca_valor(vector: TipoVector; valor, margen: real): integer;
    // Busca un valor en el vector, permitiendo un margen. Devuelve
    // su posición, o 0 si no existe
var
    i: integer;
    sal: boolean;  // Cuando sea cierto, salgo del bucle
begin
    result := 0;
    i := 1;
    sal := FALSE;
    repeat
        if abs(vector[i] - valor) <= margen then begin
            result := i;
            sal := TRUE
        end;
        i := i+1;
    until sal   // ¡Mal! Si no existe el valor, i se desborda
end;
```

```
const
    vector_prueba: TipoVector = (93.0, 87.4, 0, 86.2);
    tiempo_a_buscar: real = 95;
    margen: real = 0.6;
begin
    writeln(busca_valor(vector_prueba, tiempo_a_buscar, margen))
end.
```

Resultado:

```
Runtime error 201 at $00000000004010D8
  $00000000004010D8 line 23 of busqueda_real_mal_02.pas
  $000000000040118F line 36 of busqueda_real_mal_02.pas
  $000000000040104C
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program busqueda_real_02;
    // Búsqueda de un real, versión mejorada.
    // Si lo encontramos, dejamos de buscar.

const
    TamanioVector = 4;
type
    TipoVector = array[1..TamanioVector] of real;
```

```
function busca_valor(vector: TipoVector; valor, margen: real): integer;
    // Esta función busca un valor en el array, permitiendo
    // cierto margen. Devuelve su posición en el vector. Si no existe,
    // devuelve 0
var
    i: integer;
    sal: boolean;  // Cuando sea cierto, salgo del bucle
begin
    result := 0;
    i := 1;
    sal := FALSE;
    repeat
        if abs(vector[i] - valor) <= margen then begin
            result := i;
            sal := TRUE
        end;
        if i= TamanioVector then
            sal := TRUE
        else
            i := i+1
    until sal
end;
```

```
const
    Tiempos_prueba: TipoVector = (93.0, 87.4, 0, 86.2);
    Tiempo_a_buscar: real = 95;
    Margen: real = 0.6;
begin
    writeln(busca_valor(Tiempos_prueba, Tiempo_a_buscar, Margen))
end.
```

Resultado:

0

Presentaremos ahora un ejemplo donde queremos comprobar si **todos** los valores cumplen cierta condición. Este tipo de problemas se resuelve así:

- En una variable booleana indicamos si se cumple la condición o no
- Inicialmente almacenamos en esta variable que sí se cumple
- Ahora nuestro objetivo es encontrar un caso donde no se cumpla. Basta con uno
- Recorremos todo el array en un bucle
  - Si aparece un caso que incumple, ya podemos marcar que la condición no se cumple y forzar la salida del bucle
  - Si llegamos al final del array, salimos del bucle

Tras la busqueda, si se mantiene la suposición provisional inicial de que la condicion se cumple, podemos afirmar que definitivamente se cumple. Y si no, no

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program todos_en_margen;
// Queremos saber si todos los valores de un vector cumplen
// la condición de estar en el intervalo definido por un
// valor de referencia +- cierto margen

const
    TamanioVector = 4;
type
    TipoVector = array[1..TamanioVector] of real;
```

```
function comprueba_margen(
    vector: TipoVector; referencia, margen: real): boolean;
    // Esta función comprueba si todos los vector están dentro
    // de un valor promedio, más menos cierto margen
var
    i: integer;
    sal: boolean;  // Cuando sea cierto, salgo del bucle
begin
    result := TRUE; // De momento supongo que todos cumplen
    i := 1;
    sal := FALSE;   // De momento indico que no acabe el bucle
    repeat
        if abs(vector[i] - referencia) > margen then begin
            // Valor fuera del intervalo
            result := False;
            sal := TRUE
        end;
        if i= TamanioVector then
            sal := TRUE
        else
            i := i+1
    until sal;
end;
```

```
const
    Margen: real = 5 ;
    Vector_prueba_1 : TipoVector = (115.4, 110.9, 111.9, 114.1);
    Vector_prueba_2 : TipoVector = (115.4, 124.2, 111.9, 120.7);
var
    valor_referencia : real = 113.0;
begin
    writeln(comprueba_margen(
        Vector_prueba_1, valor_referencia, Margen));  // TRUE

    writeln(comprueba_margen(
        Vector_prueba_2, valor_referencia, Margen))  // FALSE
end.
```

Comprobemos que todos los tiempos están dentro de cierto margen, expresado porcentualmente:
https://gsyc.urjc.es/~mortuno/fpi/en_margen_relativo.pas

Máximo y Mínimo

Queremos buscar los valores mínimo y máximo de un vector

- Provisionalmente, tomamos el primer valor como máximo y como mínimo
- Recorremos todo el array desde la segunda posición, cuando algún valor sea mayor que el máximo provisional o menor que el mínimo provisional, actualizamos el máximo / mínimo provisional

Otra enfoque posible podría ser:

- Tomar provisionalmente como mínimo, un valor que sabemos que será mejorado por cualquiera. Un valor más alto que cualquier valor posible
- Tomar provisionalmente como máximo, un valor que sabemos que será mejorado por cualquiera. Un valor menor que cualquier valor posible

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program minimo_maximo_01;
    // Muestra los valores mínimo y máximo de un array
const
    TamanioVector = 4;
type
    TipoVector = array[1..TamanioVector] of real;

procedure minimo_maximo(
    vector: TipoVector; var minimo, maximo : real);
var
    i : integer;
begin
    minimo := vector[1];
    maximo := vector[1];
    for i := 2 to TamanioVector do begin
        if vector[i] < minimo then
            minimo := vector[i]
        else if vector[i] > maximo then
            maximo := vector[i];
    end;
end;
```

```
const
    vector_prueba: TipoVector = (93.0, 87.4, 91.3 , 86.2);
var
    maximo, minimo : real;
begin
    if TamanioVector < 2 then
        writeln('El vector ha de tener al menos dos elementos')
    else begin
        minimo_maximo(vector_prueba, minimo, maximo);
        write('Minimo: ', minimo:0:1);
        writeln(' Máximo: ', maximo:0:1);
    end;
end.
```

Resultado:

```
Minimo: 86.2 Máximo: 93.0
```

Busquemos ahora no el máximo y el mínimo, sino la posición en el vector del máximo y el mínimo

```
procedure minimo_maximo(
    vector: TipoVector; var pos_min, pos_max : integer);
var
    i : integer;
begin
    pos_min := 1;
    pos_max := 1;
    for i := 2 to TamanioVector do begin
        if vector[i] < vector[pos_min] then begin
            pos_min := i;
        end
        else if vector[i] > vector[pos_max] then begin
            pos_max := i;
        end;
    end;
end;
```

https://gsyc.urjc.es/~mortuno/fpi/minimo_maximo_02.pas

Matrices

Podemos usar arrays de dos (o más) índices. Se denominan *matrices*

- Declaramos los índices separados por comas

```
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;
```

- Para acceder a la matriz, escribimos los índices dentro de los corchetes, separados por comas

```
matriz[i,j] = 0;
write(matriz[i,j]);
```

Recuerda que en Pascal normalmente se empieza a contar en 1, mientras que en muchos otros lenguajes, por influencia de C, se empieza por el 0. El criterio de Pascal facilita el procesamiento de vectors y arrays, para N elementos los bucles pueden ir desde 1 hasta N. En otros lenguajes los bucles se recorrería desde 0 hasta $N - 1$, lo que resulta ligeramente más complicado

Este primer ejemplo necesita varias mejoras

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program matriz_01;
const
    Filas = 2;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;
var
    matriz : TipoMatriz;
    i,j: integer; // Estos nombres de variables son muy cortos,
                  // pero son tradicionales para los índices
```

```
begin   // ¡Mal Diseño!
    matriz[1,1] := 11;
    matriz[1,2] := 12;
    matriz[1,3] := 13;
    matriz[2,1] := 21;
    matriz[2,2] := 22;
    matriz[2,3] := 23;
    for i := 1 to Filas do
        for j := 1 to Columnas do
            write(matriz[i,j], ' ');
end.
```

Resultado:

```
11 12 13 21 22 23
```

Observa que *TipoMatriz* es un tipo global, visible en todo el programa por definirse al principio

- Las variables nunca deben ser globales. Sin embargo es muy habitual que los tipos sean globales (aunque puedan ser a veces locales)

La siguiente versión corrige los problemas de la anterior:

- El código se divide en subprogramas
- Añade *writeln* tras la impresión de cada fila, para que se imprima en una nueva línea

```
program matriz_02;
const
    Filas = 2;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

procedure inicia_matriz(var matriz:TipoMatriz);
var
    i,j : integer;
begin
    for i := 1 to Filas do
        for j:= 1 to Columnas do
            matriz[i,j] := i * 10 + j;
end;
```

```
procedure escribe_matriz(matriz:TipoMatriz);
var i,j: integer;
begin
    for i := 1 to Filas do begin
        for j:= 1 to Columnas do
            write(matriz[i,j], ' ');
        writeln;
    end;
end;

var
    matriz : TipoMatriz;
begin
    inicia_matriz(matriz);
    escribe_matriz(matriz);
end.
```

Resultado:

```
11 12 13
21 22 23
```

Observaciones

- El parámetro *matriz* del procedimiento *inicia_matriz* es necesariamente de salida, se pasa por referencia (anteponiendo la palabra reservada *var*)
- El bucle de escritura de las filas ahora tiene dos líneas, por lo que se añade un bloque *begin end*

El siguiente programa genera una matriz donde cada posición es un valor aleatorio

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program matriz_aleatoria;
uses crt;  // Necesario para delay
const
    Filas = 3;
    Columnas = 4;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

function tira_dado(caras_dado:integer):integer;
begin
    result := random(caras_dado) + 1;
end;
```

```
procedure inicia_matriz(var matriz:TipoMatriz);
var
    i,j : integer;
const
    CarasDado = 6;
begin
    for i := 1 to Filas do
        for j:= 1 to columnas do
            matriz[i,j] := tira_dado(CarasDado);
end;

procedure escribe_matriz(matriz:TipoMatriz);
var i,j: integer;
begin
    for i := 1 to Filas do begin
        for j:= 1 to Columnas do
            write(matriz[i,j], ' ');
        writeln;
    end;
end;
```

```
var
    matriz : TipoMatriz;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz);
    escribe_matriz(matriz);
end.
```

Resultado:

```
2 2 3 5
4 6 4 1
2 6 1 4
```

El siguiente programa suma todos los valores de una matriz

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program suma_matriz;
uses crt;  // Necesario para delay
const
    Filas = 2;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
procedure escribe_matriz(matriz:TipoMatriz);
// Iguales a los ejemplos anteriores
```

```
function sumatorio_matriz(matriz:TipoMatriz): integer;
var i,j, sumatorio: integer;
begin
    sumatorio := 0;
    for i:= 1 to Filas do
        for j:= 1 to Columnas do
            sumatorio := sumatorio + matriz[i,j];
    result := sumatorio;
end;

var
    matriz : TipoMatriz;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz);
    escribe_matriz(matriz);
    writeln('Suma de elementos: ',sumatorio_matriz(matriz));
end.
```

Resultado:

```
1 4 6
3 2 1
Suma de elementos: 17
```

Este programa suma los elementos de una matriz, por filas

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program sumatorio_filas;
uses crt;  // Necesario para delay
const
    Filas = 3;
    Columnas = 4;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
procedure escribe_matriz(matriz:TipoMatriz);
// Iguales a los ejemplos anteriores
```

```
procedure suma_por_filas(var matriz:TipoMatriz);
var i,j, sumatorio: integer;
begin
    for i:= 1 to Filas do begin
        sumatorio := 0;
        for j:= 1 to Columnas do
            sumatorio := sumatorio + matriz[i,j];
        write('Suma fila ', i);
        writeln(': ', sumatorio);
    end;
end;
```

```
var
    matriz : TipoMatriz;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz);
    escribe_matriz(matriz);
    suma_por_filas(matriz);
end.
```

Resultado:

```
3 4 6 3
5 4 4 1
3 5 3 5
Suma fila 1: 16
Suma fila 2: 14
Suma fila 3: 16
```

Este programa es muy parecido al anterior, pero

- Inicia el sumatorio una vez por cada fila, no una vez para toda la matriz
- El subprograma principal es *suma_por_filas*. Es un procedimiento, no una función. Recuerda que las funciones no deben tener efectos laterales (como escribir en pantalla)

Sumemos ahora las columnas de una matriz

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program sumatorio_columnas;
uses crt;  // Necesario para delay
const
    Filas = 3;
    Columnas = 4;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
procedure escribe_matriz(matriz:TipoMatriz);
// Iguales a los ejemplos anteriores
```

```
procedure suma_por_columnas(var matriz:TipoMatriz);
var i,j, sumatorio: integer;
begin
    for j:= 1 to Columnas do begin
        sumatorio := 0;
        for i:= 1 to Filas do
            sumatorio := sumatorio + matriz[i,j];
        write('Suma columna ', j);
        writeln(': ', sumatorio);
    end;
end;
```

```
var
    matriz : TipoMatriz;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz);
    escribe_matriz(matriz);
    suma_por_columnas(matriz);
end.
```

Resultado:

```
4 1 5 1
4 2 6 4
2 6 4 3
Suma columna 1: 10
Suma columna 2: 9
Suma columna 3: 15
Suma columna 4: 8
```

Observaciones

- Normalmente procesamos *por filas*, esto es, en el orden *de lectura* (occidental): de izquierda a derecha y de arriba a bajo. Fijamos la fila, procesamos cada columna y bajamos a la fila siguiente
- Pero ahora necesitamos procesar por columnas. Así que el primer bucle es el de las columnas, y dentro, el bucle de las filas
- Es recomendable que siempre reserves la variable *i* para las filas y la *j* para las columnas. En este ejemplo algún programador podría tener la tentación de intercambiarlas, pero seguramente induciría confusión
- Es especialmente fácil *despistarse* con estos algoritmos. En cualquier programa es importante probar bien cada funcion, pero en estos casos, mucho más

Generación de matrices

- Para empezar por algo sencillo, en los ejemplos previos hemos escrito en pantalla el resultado de las operaciones deseadas
  - Con un procedimiento, porque las funciones bien escritas ni escriben en pantalla ni tienen ningún otro efecto lateral
- Normalmente no haremos eso, sino que generaremos un nuevo vector o una nueva matriz. Que posteriormente escribiremos en pantalla, si procede
  - Una función nos devolverá el vector o la matriz con el resultado. A menos que sea *demasiado grande*, entonces tendremos que usar un parámetro de salida de un procedimiento

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program sumatorio_filas_02;
uses crt;  // Necesario para delay
const
    Filas = 4;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;
    TipoVector = array[1..Filas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
procedure escribe_matriz(matriz:TipoMatriz);
// Iguales a los ejemplos anteriores
```

```
procedure escribe_vector(var mi_vector:TipoVector);
begin
    for i := 1 to Filas do
        write(mi_vector[i], ' ');
    writeln;
end;

function suma_por_filas(var matriz:TipoMatriz):TipoVector;
var i,j, sumatorio: integer;
begin
    for i:= 1 to Filas do begin
        sumatorio := 0;
        for j:= 1 to Columnas do
            sumatorio := sumatorio + matriz[i,j];
        result[i] := sumatorio;
    end;
end;
```

```
var
    matriz : TipoMatriz;
    vector_suma : TipoVector;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz);
    escribe_matriz(matriz);
    vector_suma := suma_por_filas(matriz);
    writeln('Sumatorio por filas:');
    escribe_vector(vector_suma);
end.
```

Resultado:

```
5 5 1
1 2 1
6 1 1
2 5 5
Sumatorio por filas:
11 4 8 12
```

Generar el vector es muy similar a escribir los resultados, pero

- En vez de escribir con un procedimiento, usamos una función que devuelve un `TipoVector`, asignando a `result[i]` el valor obtenido
  - Si el vector a devolver fuera *grande* habría que usar un procedimiento con un parámetro de salida
- En este caso necesitamos un vector con tantos elementos como filas. Si hiciéramos algún proceso por columnas, el vector tendría tantos elementos como columnas

Suma de matrices

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program adicion_matrices;
uses crt;  // Necesario para delay
const
    Filas = 4;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
procedure escribe_matriz(matriz:TipoMatriz);
// Igual a los ejemplos anteriores
```

```
function suma_matrices(a,b:TipoMatriz):TipoMatriz;
var
    i, j: integer;
begin
    for i:= 1 to Filas do
        for j:=1 to Columnas do
            result[i,j] := a[i,j] + b[i,j];
end;
```

```
var
    matriz_1, matriz_2, matriz_3 : TipoMatriz;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz_1);
    inicia_matriz(matriz_2);
    writeln('Matriz 1');
    escribe_matriz(matriz_1);
    writeln('Matriz 2');
    escribe_matriz(matriz_2);
    matriz_3 := suma_matrices(matriz_1, matriz_2);
    writeln('Matriz suma:');
    escribe_matriz(matriz_3);
end.
```

Resultado:

```
Matriz 1
2 4 6
3 5 1
5 3 6
4 3 6
Matriz 2
4 4 3
3 6 5
3 6 5
4 2 3
Matriz suma:
6 8 9
6 11 6
8 9 11
8 5 9
```

Observaciones

- Al escribir la matriz suma, algunos elementos ocupan un espacio y otros dos (según sean menores que 10 o no). Esto queda un poco *feo*
- Podemos solucionarlo indicando que todos los valores ocupen al menos dos espacios, añadiendo :2 en el *write*

```
write(matriz[i,j]:2, ' ');
```

De esta forma el resultado sería

```
 6  8  9
 6 11  6
 8  9 11
 8  5  9
```

Las tres matrices del programa anterior aparecen una debajo de la otra. Sería conveniente mostrarlas una al lado de otra. Para ello

- Necesitamos un procedimiento que escriba la fila *i-ésima* de una matriz
- Para cada fila, invocaremos a este procedimiento, para cada una de las tres matrices

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program matrices_horizontales;
uses crt;  // Necesario para delay
const
    Filas = 4;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
// Igual a los ejemplos anteriores

procedure escribe_matriz(matriz:TipoMatriz);
// Como en los ejemplos anteriores, pero con
//     write(matriz[i,j]:2, ' ');
```

```
procedure escribe_fila(matriz:TipoMatriz; i:integer);
var
    j: integer;
begin
    for j := 1 to Columnas do
        write(matriz[i,j]:2);
        write(' ');
    write('  ');
end;

procedure escribe_3_matrices(
    matriz_1, matriz_2, matriz_3: TipoMatriz);
var
    i : integer;
begin
    for i := 1 to Filas do begin
        escribe_fila(matriz_1, i);
        escribe_fila(matriz_2, i);
        escribe_fila(matriz_3, i);
        writeln;
    end;
end;
```

```
var
    matriz_1, matriz_2, matriz_3 : TipoMatriz;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz_1);
    inicia_matriz(matriz_2);
    inicia_matriz(matriz_3);
    writeln('Matriz 1');
    escribe_matriz(matriz_1);
    writeln('Matriz 2');
    escribe_matriz(matriz_2);
    writeln('Matriz 3:');
    escribe_matriz(matriz_3);
    writeln;
    escribe_3_matrices(matriz_1, matriz_2, matriz_3);
end.
```

Resultado:

```
Matriz 1
 6  5  4
 4  1  3
 4  1  5
 2  1  5
Matriz 2
 3  5  1
 6  5  4
 6  6  2
 3  2  1
Matriz 3:
 1  5  6
 3  2  5
 5  5  1
 2  4  3

 6  5  4     3  5  1     1  5  6
 4  1  3     6  5  4     3  2  5
 4  1  5     6  6  2     5  5  1
 2  1  5     3  2  1     2  4  3
```

Máximos de una matriz

El siguiente programa calcula un vector que contiene el valor máximo de cada fila de una matriz

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program maximos_matriz;
uses crt;  // Necesario para delay
const
    Filas = 4;
    Columnas = 3;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of integer;
    TipoVector = array[1..Filas] of integer;

function tira_dado(caras_dado:integer):integer;
procedure inicia_matriz(var matriz:TipoMatriz);
procedure escribe_matriz(matriz:TipoMatriz);
procedure escribe_vector(var mi_vector:TipoVector);
// Iguales a los ejemplos anteriores
```

```
function maximo_fila(matriz:TipoMatriz; i:integer):integer;
   // Devuelve el valor máximo de la fila i de la matriz
var j,  max: integer;
begin
    max := matriz[i,1];
    for j:= 2 to Columnas do
        if matriz[i,j] > max then
            max := matriz[i,j];
    result := max;
end;

function maximo_por_filas(matriz:TipoMatriz):TipoVector;
// Devuelve un vector con los máximos de cada fila
var
    i : integer;
begin
    for i := 1 to filas do begin
        result[i] := maximo_fila(matriz,i);
    end;
end;
```

```
var
    matriz : TipoMatriz;
    vector_maximos : TipoVector;
begin
    randomize();
    delay(800);
    if Columnas < 2 then
        writeln('La matriz tiene que tener al menos 2 columnas')
    else begin
        inicia_matriz(matriz);
        escribe_matriz(matriz);
        vector_maximos := maximo_por_filas(matriz);
        writeln('Maximos de cada fila:');
        escribe_vector(vector_maximos);
    end;
end.
```

Resultado:

```
2 1 2
6 6 3
2 5 4
1 2 1
Maximos de cada fila:
2 6 5 2
```

Búsqueda en matrices

Todo lo visto sobre búsqueda en vectores es aplicable a búsqueda en matrices

- Como los vectores tienen un índice, hay un bucle. Las matrices tienen dos índices, por tanto basta añadir un segundo bucle, anidado

Exactamente igual que en las búsquedas sobre vectores

- Para buscar cierto valor, suponemos inicialmente que no lo hemos econtrado con un booleano a *False*. Si aparece, lo ponemos a *True*
- Para comprobar si todos los valores cumplen cierta condición, suponemos inicialmente que sí la cumplen (booleano a *True*). Si alguno, basta con uno, la incumple, ponemos el booleano a *False*

¿Hay algún múltiplo de K en la matriz?

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program algun_multiplo; // Busca en una matriz un múltiplo de k

[...]

function es_multiplo(numero, k: integer): boolean;
begin
    result := (numero mod k ) = 0
    // Si la división entera de un número entre k tiene
    // como resto 0, es que el número es múltiplo de k
end;

function algun_multiplo(matriz: TipoMatriz;k: integer): boolean;
var
    i,j: integer;
begin
    result := False;  // De momento no hay ninguno
    for i:= 1 to Filas do
        for j:= 1 to Columnas do
            if es_multiplo( matriz[i,j] , k) then
                result := True
end;
```

```
var
    matriz : TipoMatriz;
const
    CarasDado  =  24 ;
    K = 7 ;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz, CarasDado);
    escribe_matriz(matriz);
    if algun_multiplo(matriz, K) then
        writeln('Hay algún múltiplo de ',K)
    else
        writeln('No hay ningún múltiplo de ',K)
end.
```

https://gsyc.urjc.es/~mortuno/fpi/algun_multiplo.pas

¿Son todos los reales mayores que K?

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program todos_mayores;
// Indica si en una matriz de reales todos los números son
//mayores que K
uses crt;  // Necesario para delay
const
    Filas = 3;
    Columnas = 5;
type
    TipoMatriz = array[1..Filas, 1..Columnas] of real;


function genera_real(cota_superior:real):real;
begin
    result := random() * cota_superior;
end;
```

```
procedure inicia_matriz(var matriz:TipoMatriz; cota_superior: real);
var
    i,j : integer;
begin
    for i := 1 to Filas do
        for j:= 1 to Columnas do
            matriz[i,j] := genera_real(cota_superior);
end;


procedure escribe_matriz(matriz:TipoMatriz);
var i,j: integer;
begin
    for i := 1 to Filas do begin
        for j:= 1 to Columnas do
            write(matriz[i,j]:6:2);
        writeln;
    end;
end;
```

```
function todos_mayores(matriz: TipoMatriz;k: real): boolean;
var
    i,j: integer;
begin
    result := True;  // De momento todos son mayores
    for i:= 1 to Filas do
        for j:= 1 to Columnas do
            if  matriz[i,j] <= k then
                result := False;
end;
```

```
var
    matriz : TipoMatriz;
const
    CotaSuperior  =  10 ;
    K = 1.0;
begin
    randomize();
    delay(800);
    inicia_matriz(matriz, CotaSuperior);
    escribe_matriz(matriz);
    if todos_mayores(matriz, K) then
        writeln('Todos los valores son mayores que ',K:4:2)
    else
        writeln('No todos los valores son mayores que ',K:4:2)
end.
```

https://gsyc.urjc.es/~mortuno/fpi/todos_mayores.pas

Array de registros

Hasta ahora hemos visto arrays de tipos elementales (enteros, reales, etc) pero es mucho más habitual usar arrays de registros. En el siguiente ejemplo

- Definiremos un registro de TipoAeropuerto, para almacenar código IATA y nombre de un aeropuerto
- Definiremos un array de TipoAeropuerto
- Escribiremos procedimientos para dar valor al array y para imprimirlo

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program aeropuertos_01;
const
    NumAeropuertos = 5;
type
    TipoAeropuerto = record // TipoAeropuerto en singular
        codigo : string;
        nombre : string;
    end;
    TipoAeropuertos = array[1..NumAeropuertos] of TipoAeropuerto;
        // TipoAeropuertos en plural, el array

procedure escribe_aeropuertos(aeropuertos:TipoAeropuertos);
var
    i : integer;
begin
    for i := 1 to NumAeropuertos do begin
        write(aeropuertos[i].codigo, ' ' );
        writeln(aeropuertos[i].nombre);
    end;
end;
```

```
procedure inicia_aeropuertos(var aeropuertos:TipoAeropuertos);
begin
    aeropuertos[1].codigo := 'MAD';
    aeropuertos[1].nombre := 'Adolfo Suárez Madrid Barajas Airport';
    aeropuertos[2].codigo := 'LHR';
    aeropuertos[2].nombre := 'Heathrow Airport';
    aeropuertos[3].codigo := 'CDG';
    aeropuertos[3].nombre := 'Charles de Gaulle Airport';
    aeropuertos[4].codigo := 'CGN';
    aeropuertos[4].nombre := 'Cologne Bonn Airport';
    aeropuertos[5].codigo := 'OVD';
    aeropuertos[5].nombre := 'Asturias Airport';
end;

var
    aeropuertos : TipoAeropuertos;
begin
    inicia_aeropuertos(aeropuertos);
    escribe_aeropuertos(aeropuertos);
end.
```

Resultado:

```
MAD Adolfo Suárez Madrid Barajas Airport
LHR Heathrow Airport
CDG Charles de Gaulle Airport
CGN Cologne Bonn Airport
OVD Asturias Airport
```

Ahora escribiremos un programa para buscar el nombre de aeropuerto correspondiente a un código IATA

- Inicialmente, el resultado provisional será la cadena *Aeropuerto desconocido*
- Recorreremos todo el array, si algún código IATA es igual al buscado, el resultado será el aeropuerto correspondiente
- En el programa *aeropuerto_02* buscamos en todo el array, incluso aunque lo hayamos encontrado
- El programa *aeropuerto_03* está mejorado, porque si encuentra el aeropuerto, deja de buscar

```
programa aeropuertos_02;
[...]
function busca_aeropuerto(aeropuertos:TipoAeropuertos; codigo:string):
↪  string;
var
    i:integer;
begin
    result := 'Aeropuerto desconocido';
    for i := 1 to NumAeropuertos do
        if aeropuertos[i].codigo = codigo then
            result := aeropuertos[i].nombre;
end;

var
    aeropuertos : TipoAeropuertos;
begin
    inicia_aeropuertos(aeropuertos);
    writeln(busca_aeropuerto(aeropuertos, 'LHR'));
    writeln(busca_aeropuerto(aeropuertos,'LPA'));
end.
```

Resultado:

```
Heathrow Airport
Aeropuerto Desconocido
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program aeropuertos_03;
[...]
function busca_aeropuerto(aeropuertos:TipoAeropuertos; codigo:string):
↪  string;
var
    i : integer = 1;
    sal : boolean = FALSE;
begin
    result := 'Aeropuerto desconocido';
    repeat
        if aeropuertos[i].codigo = codigo then begin
            result := aeropuertos[i].nombre;
            sal := TRUE;
        end;
        if i = NumAeropuertos then
            sal := TRUE
        else
            i := i + 1;
    until sal;
end;
```

Ejemplo: normalización de valores

En este ejemplo manejamos una lista de registros, que contiene una fecha de vuelo y una duración de vuelo

- Calculamos el tiempo normalizado de cada vuelo *i-ésimo*, que será tiempo[i] / media(tiempos)
- Añadimos el tiempo normalizado a otro campo del registro

https://gsyc.urjc.es/~mortuno/fpi/normaliza_01.pas
Resultado:

```
Fecha       Tiempo  Tiempo Normalizado
2018.11.18   92.4   1.002
2018.11.19   96.6   1.048
2018.11.20   88.7   0.962
2018.11.21   91.1   0.988

Tiepo medio: 92.200
```

Procesamiento de cadenas

En muchos programas tendremos que tratar cadenas. Una función básica es `length()`, que devuelve el número de caracteres

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program longitud_cadena;
var
    s: string;

begin
    s := 'hola';
    writeln(length(s));      // 4
    // La función se llama length(), ni len(), ni lengh(),
    // ni lenght, ni lenth(), ni...
end.
```

Podemos acceder al caracter *i-ésimo* de una cadena tratándola como un array de caracteres

- De hecho, el tipo básico de cadenas que estamos usando es precisamente eso, un array de char

Vamos a *deletrear* una palabra, escribiéndola letra a letra, con un guión separando las letras

- En el programa *deletrea_01*, escribimos un guión después de cada letra
- En *deletrea_02*, escribimos un guión después de cada letra, excepto en la última

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program deletrea_01;
var
    s: string;
    i: integer;

begin
    s := 'hola';
    for i:= 1 to length(s) do
        write(s[i], '-');
    writeln;
end.
```

Resultado:

```
h-o-l-a-
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program deletrea_02;
var
    s: string;
    i: integer;

begin
    s := 'hola';
    for i:= 1 to length(s)-1 do
        write(s[i], '-');
    writeln(s[i+1]);
        // En otros lenguajes no podemos usar la variable del
        // bucle fuera del bucle, en Pascal sí.
end.
```

Resultado:

```
h-o-l-a
```

Escritura de una cadena al revés

En muchos programas usaremos la *cadena vacía*

```
''
```

- Está formado por una comilla, cero caracteres y otra comilla
- Su longitud es cero
- Es completamente distinta de una cadena con uno o más espacios (cuya longitud sería 1, 2, 3...)
- Es completamente distinta a una cadena no definida

En el siguiente ejemplo, invertiremos una cadena

1. Partimos de la cadena vacía
2. Vamos concatenado cada carácter, empezando por el final

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program invierte_01;

function invierte(s:string):string;
var
    salida : string;
    i: integer;
begin
    salida := '';
    for i := length(s) downto 1 do
        salida := salida + s[i];
    result := salida;
end;

var
    s: string;
begin
    s := 'hola';
    writeln(invierte(s));
end.
```

Resultado:

```
aloh
```

Ahora que sabemos invertir una cadena, vamos a buscar palíndromos

- En el programa *palindromo_01* intentamos buscar palíndromos comparando una cadena con su cadena invertida
  Pero no funciona, porque falta eliminar los espacios
- Para eliminar espacios:
  - Podemos usar una función o un procedimiento. Mostraremos aquí ambas cosas
  - Empezamos por la cadena vacía
  - Vamos concatenando todos los caracteres de la cadena original, excepto los espacios
  - Permutamos la cadena resultante con la original
- En *palindromo_02* detectamos los palíndromos correctamente

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program palindromo_01;
    // Primer intento, erróneo
function invierte(s:string):string;
var
    salida : string;
    i: integer;
begin
    salida := '';
    for i := length(s) downto 1 do
        salida := salida + s[i];
    result := salida;
end;
function palindromo(s:string):boolean;
begin
    result := (s = invierte(s));
end;

begin
    writeln(palindromo('ana'));   // TRUE, ok
    writeln(palindromo('hola, mundo'));  // FALSE, ok
    writeln(palindromo('acaso hubo buhos aca')); // FALSE
        // En el tercer ejemplo fallan los espacios
end.
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program quita_espacios_01;
procedure quita_espacios(var s:string);
var
    salida : string;
    i: integer;
begin
    salida := '';
    for i := 1 to length(s) do
        if s[i] <> ' ' then
            salida := salida + s[i];
    s := salida;
end;

var
    s: string;
begin
    s := 'hola mundo';
    quita_espacios(s);
    writeln(s);
end.
```

Resultado:

```
holamundo
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program palindromo_02;
function invierte(s:string):string;
var
    salida : string;
    i: integer;
begin
    salida := '';
    for i := length(s) downto 1 do
        salida := salida + s[i];
    result := salida;
end;

function quita_espacios(s:string):string;
var
    salida : string;
    i: integer;
begin
    salida := '';
    for i := 1 to length(s) do
        if s[i] <> ' ' then
            salida := salida + s[i];
    result := salida;
end;
```

```
function palindromo(s:string):boolean;
begin
    s := quita_espacios(s);
    result := (s = invierte(s));
end;

begin
    writeln(palindromo('son mulas o civicos alumnos'));
    writeln(palindromo('hola, mundo'));
    writeln(palindromo('la ruta nos aporto otro paso natural'));
end.
```

Resultado:

```
TRUE
FALSE
TRUE
```