Programación en Pascal. Arrays

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Noviembre de 2019

Contenidos

Arrays

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

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

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

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

- Declaramos `tabla` 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, aunque podríamos usar el 0 al estilo de otros lenguajes. O 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
    tabla : array[-1..1] of string;

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

    writeln(tabla[-1]);
    writeln(tabla[0]);
    writeln(tabla[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
    tabla : array[1..3] of string;
    i : integer;

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

    for i := 1 to 3 do
        writeln(tabla[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
    tabla : Array[1..3] of string = ('Primero', 'Segundo', 'Tercero');
    i : integer;

begin
    for i:= 1 to 3 do
        writeln(tabla[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
    tabla : array[1..3] of string;

begin
    tabla[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
    tabla : array[1..3] of string;

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

Resultado:

```
Cuarto
```

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

- Normalmente declararíamos y definiríamos esta constante así:

```
const
    TamanioArray : 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
    TamanioArray = 3;   // Correcto
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_04;
const
  //   TamanioArray : integer = 3;
  //   No se puede declarar el tipo de esta constante porque
  //   usaremos la constante para declarar otro tipo
    TamanioArray = 3;

var
    tabla : array[1..TamanioArray] of string;
    i : integer;

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

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

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

```
var
    tabla : array[1..TamanioArray] 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 correcto es declarar un nuevo tipo de datos. En este caso,

`TipoArray`

```
const
    TamanioArray = 3;

type
    TipoArray = array[1..TamanioArray] of string;

var
    tabla : TipoArray;
```

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

const
    TamanioArray = 3;

type
    TipoArray = array[1..TamanioArray] of string;

var
    tabla : TipoArray;
    i : integer;

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

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

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program array_06;  // Guardamos la salida de un dado
uses crt;
const
    Tamanio_array = 5;
type
    TipoArray = array[1..Tamanio_array] of integer;
// Definimos tira_dado() como en el tema 7
var
    i: integer;
    tabla : TipoArray;
const
    Caras_dado: integer = 6;
begin
    delay(1000);
    randomize;

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

    for i := 1 to Tamanio_array do begin
        write(tabla[i], ' ');
    end;
    writeln;
```

Matrices

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

- Declaramos los índices separados por comas

```
type
    TipoMatriz = array[1..UltimaFila, 1..UltimaColumna] of
    ↪  integer;
```

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

```
matriz[4,1] = una_variable;
write(matriz[i,j];
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program matriz_01;

const
    UltimaFila = 2;
    UltimaColumna = 3;

type
    TipoMatriz = array[1..UltimaFila, 1..UltimaColumna] of integer;

var
    matriz : TipoMatriz;
    i,j: integer; // Nombres muy cortos, excepcionales, tradicionales
```

```
begin
    matriz[1,1] := 41;
    matriz[1,2] := 43;
    matriz[1,3] := 43;
    matriz[2,1] := 51;
    matriz[2,2] := 51;
    matriz[2,3] := 31;

    for i := 1 to UltimaFila do begin
        for j := 1 to UltimaColumna do
            write(matriz[i,j], ' ');
        writeln;
    end;
end.
```

Resultado:

```
41 43 43
51 51 31
```

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, que nos permitirán trabajar con colecciones de datos (arrays), que estudiamos en el tema actual. Los problemas con colecciones 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

Tenemos una ruta aérea y los tiempos de vuelo, en minuto, de ciertos días concretos

- Almacenamos los tiempos en un array
- Queremos saber el tiempo medio de vuelo

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program tiempo_vuelo_01;
const
    NumVuelos = 3;

type
    TipoTiempos = array[1..NumVuelos] of real;

function suma_tiempos(tiempos: TipoTiempos): real;
var
    suma: real;
    i: integer;
begin
    suma := 0.0;
    for i := 1 to NumVuelos do begin
        suma := suma + tiempos[i];
    end;
    result := suma;
end;
```

```
function media(tiempos: TipoTiempos): real;
begin
    result := suma_tiempos(tiempos) / NumVuelos;
end;

const
    Tiempos_prueba: TipoTiempos = (115.4, 121.9, 111.9);
begin
    writeln(media(Tiempos_prueba) :0:2);   // 116.40
end.
```

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 tiempo_vuelo_02;  // Primer intento, con error
const
    NumVuelos = 3;
type
    TipoTiempos = array[1..NumVuelos] of real;

function busca_duracion(tiempos: TipoTiempos; duracion: real): integer;
    // Esta función busca un vuelo de cierta duración, y devuelve
    // su posición en la tabla. Si no existe, devuelve 0
var
    i: integer;
begin
    result := 0;
    for i := 1 to NumVuelos do begin
        if tiempos[i] = duracion then  // ¡MAL!
            result := i;
    end
end;
```

```
const
    Tiempos_prueba: TipoTiempos = (115.4, 121.9, 111.9);
    Tiempo_a_buscar: real = 121.9;
begin
    writeln(busca_duracion(Tiempos_prueba, Tiempo_a_buscar));
end.
```

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, *tiempo_vuelo_02* 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 tiempo_vuelo_03;
    // Buscamos un vuelo cuya duración se aproxime a la duración pedida
const
    NumVuelos = 4;
type
    TipoTiempos = array[1..NumVuelos] of real;

function busca_duracion(tiempos: TipoTiempos; duracion, margen: real):
↪  integer;
var
    i: integer;
begin
    result := 0;
    for i := 1 to NumVuelos do begin
        if abs(tiempos[i] - duracion) <= margen then
            result := i;
    end
end;
```

```
const
    Tiempos_prueba: TipoTiempos = (93.0, 86.9, 0, 86.2);
    Tiempo_a_buscar: real = 87;
    Margen: real = 0.1;
begin
    writeln(busca_duracion(Tiempos_prueba, Tiempo_a_buscar, Margen)) ;
end.
```

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 tiempo_vuelo_04;
    // Versión errónea, olvidamos el caso de que no exista
const
    NumVuelos = 4;
type
    TipoTiempos = array[1..NumVuelos] of real;

function busca_duracion(tiempos: TipoTiempos; duracion, margen: real):
↪  integer;
var
    i: integer;
    sal: boolean;  // Cuando sea cierto, salgo del bucle
begin
    result := 0;
    i := 1;
    sal := FALSE;
    repeat
        if abs(tiempos[i] - duracion) <= margen then begin
            result := i;
            sal := TRUE;
        end;
        i := i+1;
    until sal;      // ¡Mal!  Bucle infinito. ¿Y si no existe?
end;
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program tiempo_vuelo_05;  // Versión correcta
const
    NumVuelos = 4;
type
    TipoTiempos = array[1..NumVuelos] of real;
```

```
function busca_duracion(tiempos: TipoTiempos; duracion, margen: real):
↪  integer;
var
    i: integer;
    sal: boolean;  // Cuando sea cierto, salgo del bucle
begin
    result := 0;
    i := 1;
    sal := FALSE;
    repeat
        if abs(tiempos[i] - duracion) <= margen then begin
            result := i;
            sal := TRUE;
        end;
        if i= NumVuelos then
            sal := TRUE
        else
            i := i+1;
    until sal;
end;
```

Presentaremos ahora un ejemplo donde queremos comprobar si todos los valores cumplen cierta condición

- Es similar al caso anterior, pero empezamos considerando de forma provisional que sí se cumple la condición
- Cuando algún valor lo incumple, marcamos el resultado como falso y forzamos la salida
- Si hemos recorrido todos los valores, salimos

```
program tiempo_vuelo_06;
[...]
function comprueba_margen(tiempos: TipoTiempos; promedio, margen:
↪  real): boolean;
    // Esta función comprueba si todos los tiempos están dentro
    // de un valor promedio, más menos cierto margen
var
    i: integer;
    sal: boolean;  // Cuando sea cierto, salgo del bucle
begin
    result := TRUE;
    i := 1;
    sal := FALSE;
    repeat
        if abs(tiempos[i] - promedio) > margen then begin
            result := False;
            sal := TRUE;
        end;
        if i= NumVuelos then
            sal := TRUE
        else
            i := i+1;
    until sal;
end;
```

```
const
    Tiempos_prueba_1: TipoTiempos = (115.4, 110.9, 111.9, 114.1);
    Tiempos_prueba_2: TipoTiempos = (115.4, 124.2, 111.9, 120.7);
    Valor_promedio: real = 113;
    Margen: real = 5 ;
begin
    writeln(comprueba_margen(Tiempos_prueba_1, Valor_promedio,
    ↪  Margen));  // TRUE
    writeln(comprueba_margen(Tiempos_prueba_2, Valor_promedio,
    ↪  Margen)); // FALSE
end.
```

Comprobemos que todos los tiempos están dentro de cierto margen, expresado porcentualmente:
https://gsyc.urjc.es/~mortuno/fpi/tiempo_vuelo_07.pas

Máximo y Mínimo

Queremos buscar el tiempo mínimo y el tiempo máximo del array de tiempos de vuelos

- 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;
    // Queremos saber el tiempo mínimo y el máximo del array de tiempos
    // de vuelo

const
    NumVuelos = 4;
type
    TipoTiempos = array[1..NumVuelos] of real;

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

```
const
    Tiempos_prueba: TipoTiempos = (93.0, 87.4, 91.3 , 86.2);
var
    maximo, minimo : real;
begin
    minimo_maximo(Tiempos_prueba, minimo, maximo);
    write('Minimo: ', minimo:0:1);
    writeln(' Máximo: ', maximo:0:1);
end.
```

Resultado:

```
Minimo: 86.2 Máximo: 93.0
```

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

- Versión errónea. Olvidamos iniciar los valores
  https://gsyc.urjc.es/~mortuno/fpi/minimo_maximo_02.pas
- Versión correcta
  https://gsyc.urjc.es/~mortuno/fpi/minimo_maximo_03.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 distinto de una cadena con uno o más espacios (cuya longitud sería 1, 2, 3...)
- Es completamente distinto 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:
  - 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
```