Programación en Pascal. Ficheros

Miguel Ortuño
Escuela de Ingeniería de Fuenlabrada
Universidad Rey Juan Carlos

Marzo de 2024

Contenidos

Introducción a los ficheros

Todos los datos que hemos usado hasta ahora eran volátiles. Se perdían al concluir la ejecución del programa. Por el contrario, un *fichero*:

- Es una colección de datos *persistente*. No se pierde cuando acaba el programa ni cuando se apaga el ordenador.
- Tiene un nombre, basta usar ese nombre para leer o escribir el fichero. P.e `holamundo.pas`.
  - El nombre puede incluir un *trayecto* (*path*) , esto es, una lista de directorios y/o subdirectorios. P.e. `~/fpi/practica01/holamundo.pas`.
    Si el nombre no incluye trayecto, se entiende que el fichero está en el *directorio actual*.

Los ficheros los gestiona el sistema operativo (Linux, Windows, macOS,etc).

Los ficheros pueden contener varios tipos de datos:

- Texto.
  Son los más frecuentes, los únicos que veremos en este curso.
  Trabajaremos con ficheros formados por una secuencia de cadenas de texto, que usan el carácter *salto de línea* como separador.
- Binarios.
  Contienen números reales, enteros o cualquier otro tipo de dato y/o combinación de tipo de dato.

Hay dos formas de acceso a un fichero:

- Acceso secuencial.
  Leemos o escribimos los datos, uno tras otro.
  El único que veremos este curso.
- Acceso aleatorio.
  Leemos o escribimos cualquier posición del fichero. No significa *al azar*, sino cualquier posición que establezca el programa, de forma determinística.

Para usar un fichero hay que:

- Abrirlo, especificando el *modo de apertura*.
  - Modo lectura. El uso que haremos del fichero será leer sus valores.
  - Modo escritura. Borraremos el contenido anterior del fichero para escribir nuevos valores.
  - Otros modos: lectura-escritura, adición, ...
- Leer o escribir.
- Cerrarlo.

Un mismo fichero tiene dos nombres diferentes, es importante no confundirlos.

- El nombre del fichero que maneja el sistema operativo.
  Es el nombre que se verá en el disco, en el gestor de ficheros del sistema operativo, en cualquier otro programa.
  (Llamémosle *nombre externo*, aunque en programación se le suele llamar *nombre de fichero*, a secas).
- El nombre que tendrá el fichero internamente en nuestro programa.
  (Llamémosle *nombre interno*, aunque en programación se le suele llamar *fichero*, a secas, o *descriptor de fichero*). Será una variable:
  - No de ningún tipo básico (entero, real, cadena...) ni compuesto (registro) ni una colección (array).
  - Sino de un tipo nuevo: tipo fichero. En nuestro caso de tipo *text*.

Con frecuencia, nuestro programa manejará estos dos nombre

- Un *string* conteniendo el nombre del fichero (*nombre externo*).
- Un tipo fichero (*text*) (nombre interno).

En la apertura del fichero indicamos el nombre externo del fichero y qué variable (nombre interno) usamos para ese fichero. Y lo normal es que, tras la apertura, el nombre externo ya no lo volvamos a usar más. Usaremos solo el nombre interno.

Apertura de un fichero

En casi todos los lenguajes de programación, la apertura de un fichero se hace con una sentencia o función o método llamado *open*, a la que se pasa como argumento el nombre del fichero y el modo de apertura.
Pero nuestro dialecto de Pascal (Object Pascal) es un poco peculiar.
Para abrir un fichero:

- Declaramos el fichero como una variable de tipo *text* (más o menos como en cualquier otro lenguaje).
- Usamos el procedimiento *assign* para indicar el nombre externo de fichero que tendrá nuestra variable interna.
- Usamos el procedimiento:
  - *reset*
    para indicar que el fichero se abra en modo lectura.
  - *rewrite*
    para indicar que el fichero se abra en modo escritura.

Lectura y escritura de un fichero de texto

La lectura y escritura de un fichero de texto es muy similar a la lectura desde el teclado y escritura en pantalla.

- Usamos los procedimientos *write*, *writeln* y *readln*.
- Lo único que cambia es que añadimos como primer parámetro el nombre interno del fichero.

De hecho, tanto el teclado como la pantalla se consideran ficheros a estos efectos.

- En las lecturas con *readln*, si no se indica el fichero, se entiende que se refiere a leer desde el teclado.
- En las escrituras con *write* y *writeln*, si no se indica el fichero, se entiende que se trata de la pantalla.

Respecto a la lectura de ficheros de texto:

- Como su nombre indica, solo contienen texto. Si vamos a interpretar ese texto como números, tendremos que usar el procedimiento *val*.

Respecto a la escritura de ficheros de texto:

- Solo podremos escribir directamente tipos básicos: reales, enteros, caracteres, cadenas y booleanos.
- Si necesitamos llevar a fichero tipos más complejos (registros, vectores, arrays, etc) tendremos que escribir cada tipo básico por separado.
- El propio procedimiento *write* / *writeln* se encarga de convertir estos tipos básicos en texto.

Ejemplo:

```
readln(fichero,linea);
    // Lee en la cadena 'linea' una linea desde el fichero de
    // nombre interno 'fichero'

writeln(fichero, 'hola, mundo');
    // escribe en el fichero de nombre interno 'fichero' la constante
    // cadena 'hola, mundo'

writeln(fichero, linea);
    // escribe en el fichero de nombre interno 'fichero' la cadena
    // contenida en el string 'linea'
```

Cierre de un fichero

Es muy sencillo, basta con invocar al procemiento *close* y pasarle como argumento el nombre (interno) del fichero.

```
close(fichero);
```

- Una buena práctica es que la apertura y el cierre del fichero estén la misma función o procedimiento, en líneas prácticamente contiguas o muy próximas (que se vea en la misma pantalla).
- El procesamiento del fichero irá en una función o procedimiento aparte (a menos que sean unas pocas línea: 8, 10...).

```
assign(fichero, nombre_fichero);
reset(fichero);
procesa_fichero(fichero);
close(fichero);
```

Un programa siempre debe cerrar todos sus ficheros (aunque olvidarlo no suele ser catastrófico, normalmente lo hará el sistema operativo por nosotros cuando el programa concluya).

Escritura de una línea de texto

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program escritura_01;
var
    fichero: text;
begin
    assign(fichero, 'texto.txt');    // Ponemos nombre al fichero
    rewrite(fichero);                // Apertura en modo escritura
    writeln(fichero, 'hola,mundo'); // Escribimos en el fichero
    close(fichero);
end.
```

Uso de write y writeln en ficheros

Recuerda: *write* y *writeln* se usan en ficheros de texto igual que en pantalla. Basta añadir como primer argumento el nombre (interno) del fichero
Ejemplo en pantalla

```
    write('Área del triángulo: ');
    writeln(area:0:3);
```

Ejemplo en fichero:

```
    write(fichero, 'Área del triángulo: ');
    writeln(fichero, area:0:3);
```

(además de, por supuesto, abrir el fichero antes y cerrarlo después)

El siguiente programa escribe en pantalla:

```
Base:12.430 altura: 5.910
Área del triángulo: 73.461
```

Y exactamente lo mismo en el fichero *area_triangulo.txt*:

```
Base:12.430 altura: 5.910
Área del triángulo: 73.461
```

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

function area_triangulo(base,altura:real):real;
begin
    result := base * altura;
end;
```

```
procedure escribe_area_pantalla(base, altura, area: real);
begin
    write('Base:');
    write(Base:0:3);
    write(' altura: ');
    writeln( Altura:0:3);
    write('Área del triángulo: ');
    writeln(area:0:3);
end;

procedure escribe_area_fichero(base, altura, area: real;
↪  nombre_fichero:string);
var
    fichero : text;
begin
    assign(fichero, nombre_fichero);   // Ponemos nombre al fichero
    rewrite(fichero);                  // Apertura en modo escritura
    write(fichero, 'Base:');
    write(fichero, Base:0:3);
    write(fichero, ' altura: ');
    writeln(fichero, Altura:0:3);
    write(fichero, 'Área del triángulo: ');
    writeln(fichero, area:0:3);
    close(fichero)
end;
```

```
const
    Base = 12.43;
    Altura = 5.91;
    NombreFichero = 'area_triangulo.txt';
var
    area : real;

begin
    area := area_triangulo(base, altura);
    escribe_area_pantalla(base, altura, area);
    escribe_area_fichero(base, altura, area, NombreFichero);
end.
```

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

Lectura de una lína de texto

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program lectura_01;

var
    fichero: text;
  // Nombre interno de nuestro fichero. Una variable de tipo
  // text, esto es, de tipo 'fichero de texto'
    linea: string;
begin
    assign(fichero, 'texto.txt');   // Ponemos nombre al fichero
    // 'texto.txt' es un string con el nombre externo del fichero

    // En lo sucesivo siempre usamos el 'nombre interno'

    reset(fichero);                 // Lo abrimos en modo lectura
    readln(fichero, linea);         // Leemos una línea del fichero
    close(fichero);                 // Cerramos el fichero
    writeln(linea);                 // Escribimos en pantalla la línea
end.
```

Escritura de varias líneas en un fichero

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program escritura_02;
var
    fichero : text;
    i : integer;
    iteraciones : integer = 3;
begin
    assign(fichero, 'texto.txt');   // Ponemos nombre al fichero
    rewrite(fichero);               // Apertura en modo escritura
    for i := 1 to iteraciones do begin
        writeln(fichero, i, ' hola,mundo');
    end;
    close(fichero);
end.
```

Contenido del fichero:

```
1 hola,mundo
2 hola,mundo
3 hola,mundo
```

Lectura de todas las líneas de un fichero

Es muy frecuente que queramos leer un fichero completo, de forma secuencial.

- El tamaño de un fichero no es constante, no podemos usar un bucle `for` como en los arrays.
- Disponemos de una función `eof()` (end of file), que recibe como argumento el nombre (interno) del fichero y devuelve TRUE si hemos llegado hasta el final y por tanto no se puede seguir leyendo.

Lo habitual es usar una sentencia `while`:

```
while not eof(fichero) do
begin
    readln(fichero, linea);
    procesa_linea(linea);
end
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program lectura_02;

var
    fichero: text;   // Tipo de datos 'fichero de texto'
    linea: string;
begin
    assign(fichero, 'texto.txt');   // Ponemos nombre al fichero
    reset(fichero);                 // Lo abrimos en modo lectura

    while not eof(fichero) do
    begin
        readln(fichero, linea);
        writeln(linea);
    end;
    close(fichero);                // Cerramos el fichero
end.
```

El siguiente ejemplo (`lectura_03`) es un caso típico de procesamiento de un fichero línea a línea.

- Un procedimiento (o tal vez una función) recorre un fichero completo en modo lectura.
- Un procedimiento (o tal vez una función) procesa cada línea.
- En este ejemplo el procesamiento de la línea consiste en escribirla sin más, pero podría ser cualquier otra cosa.

Este esquema nos servirá para resolver la mayoría de los problemas comunes relacionados con el procesamiento de ficheros.

- En particular, todos los de este curso.
- Lo único necesario sera reescribir el subprograma *procesa_linea*.

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program lectura_03;
procedure procesa_linea(linea:string);
begin
    writeln(linea);
end;

procedure procesa_fichero(var fichero:text);
var
    linea: string;
begin
    while not eof(fichero) do
    begin
        readln(fichero, linea);
        procesa_linea(linea);
    end;
end;
```

```
const
    Nombre_fichero = 'texto.txt';
var
    fichero : text;
begin
    assign(fichero, Nombre_fichero);    // Ponemos nombre al fichero
    reset(fichero);                     // Lo abrimos en modo lectura

    procesa_fichero(fichero);

    close(fichero);                     // Cerramos el fichero
end.
```

Observaciones:

- Para pasar un fichero como parámetro a un subprograma, es necesario hacerlo por referencia.

```
procedure procesa_fichero(var fichero:text);
```

- Es necesario que exista el fichero en el directorio actual. En este caso, `texto.txt`. Si no, se disparará un error de ejecución (*runtime error*).
- Un programa *real* de mínima calidad no debería generar ningún error de ejecución.
  - O bien comprobaríamos que el fichero realmente existe, antes de intentar abrirlo.
  - O bien capturaríamos el *runtime error* (capturaríamos una excepción).

En los diseños más sencillos, un subprograma:

- Recibirá el nombre externo del fichero como parámetro.
- Declarará el nombre interno como variable local.
- Abrirá, procesará y cerrará el fichero.

En casos no tan básicos:

- Un subprograma abrirá el fichero.
- Pasará el nombre interno a un segundo subprograma. Solo el interno, el externo ya no lo necesitará.
- El segundo subprograma recibirá (por referencia) el nombre interno del fichero y lo procesará.
- El subprograma inicial cerrará el fichero.

Escritura, lectura y proceso

Ejemplo:

- Escribimos n números aleatorios en un fichero.
- Leemos todo el fichero y calculamos la media aritmética de sus valores.

https://gsyc.urjc.es/~mortuno/fpi/escritura_lectura_01.pas

Formato de datos en ficheros de texto

Hay muchas formas de organizar los datos dentro de un fichero de texto. En cada momento de la historia de la informática ha ido cambiando el formato que solía ser *más popular*. (Aunque siempre se han usado los formatos *antiguos*).

- Campos de ancho fijo.
  El único que veremos aquí. Muy sencillo. Aún lo siguen usando algunas aplicaciones bancarias.
- CSV (y variantes).
  comma-separated values.
- XML
  eXtensible Markup Language.
- JSON
- YAML
- ... y muchos otros.

Campos de ancho fijo

Un fichero de texto con campos de ancho fijo tiene un aspecto como por ejemplo este:

```
AAA -17.353-145.510Anaa Airport
AAB -26.693 141.048Arrabury Airport
AAC  31.073  33.836El Arish International Airport
AAD   6.096  46.638Adado Airport
AAE  36.822   7.809Rabah Bitat Airport
AAF  29.728 -85.027Apalachicola Regional Airport
AAG -24.104 -49.789Arapoti Airport
AAH  50.823   6.186Aachen-Merzbrück Airport
AAI -13.025 -46.884Arraias Airport
AAJ   3.899 -55.578Cayana Airstrip
```

- Posición 1 a 3: código IATA.
- Posición 4 a 11: latitud.
- Posición 12 a 19: longitud.
- Posición 20 hasta fin de línea: nombre del aeropuerto.

Para procesar estos ficheros:

- Los leemos línea a línea.
- Descomponemos cada línea en sus campos.
  La función `copy(linea, i, n)`
  acepta como argumentos:
  - Una cadena de texto.
  - Un entero `i`.
  - Un entero `n`.

  y devuelve la subcadena que empieza en la posición `i`, con longitud de `n` caracteres.

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program descomposicion_01;
    // Descomponemos una línea de texto en campos de longitud fija.
    // Los números están como texto, los convertimos en números
    // con el procedimiento val

const
    Pc01 = 1;   // principio del campo01
    Pc02 = 3;   // principio del campo02

var
    lc01, lc02: integer;  // Longitud de los campos 01 y 02
    linea,  campo01, campo02: string;
    codigo : integer; // para val
    nombre : string;
    dorsal : integer;
```

```
begin
    linea := '01Juan Garcia';
    lc01 := Pc02 - Pc01;
    lc02 := length(linea) - Pc02 + 1;
    campo01 := copy(linea, Pc01, lc01);
    campo02 := copy(linea, Pc02, lc02);
    val(campo01, dorsal, codigo);
    if codigo <> 0 then begin
        writeln('Valor de entrada incorrecto');
        halt;
    end;
    nombre := campo02;
    writeln(dorsal, ' ', nombre);   // 1 Juan García
end.
```

En los campos de tipo cadena normalmente tendremos que eliminar los espacios a la derecha.

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program descomposicion_02;
function quita_espacios_dcha(cadena: string): string;
var
    i : integer;
begin
    i := length(cadena);
    while cadena[i] = ' ' do
        i := i-1;
    result := copy(cadena,1, i);
end;

const
    Pc01 = 1;   // principio del campo01
    Pc02 = 6;   // principio del campo02
    Pc03 = 26;  // principio del campo03
    Eol = 46; // fin de linea (end of line)

var
    lc01, lc02, lc03: integer; // Longitud campos 01, 02, 03
    linea,  campo01, campo02, campo03: string;
    cp, poblacion, provincia : string;
```

```
begin
        //          1         2         3         4
        //  12345678901234567890123456789012345678901234567890123456
    linea := '28943Fuenlabrada          Madrid                ';

    lc01 := Pc02 - Pc01;
    lc02 := Pc03 - Pc02;
    lc03 := Eol - Pc03;

    campo01 := copy(linea, Pc01, lc01);
    campo02 := copy(linea, Pc02, lc02);
    campo03 := copy(linea, Pc03, lc03);

    cp := quita_espacios_dcha(campo01);
    poblacion := quita_espacios_dcha(campo02);
    provincia := quita_espacios_dcha(campo03);

    write(cp, ' ');
    write(poblacion,', ');
    writeln('(', provincia, ')');
end.
```

Escritura de campos de ancho fijo

Para escribir campos de ancho fijo, creamos subcadenas de ancho fijo y luego las concatenamos.

- Las subcadenas de los campos numéricos los creamos con el procedimiento `str`.
- Para las subcadenas de los campos de texto:
  - Podemos usar nuestra propia función.
  - Podemos usar la función `padright()` de la librería `strutils`.

https://gsyc.urjc.es/~mortuno/fpi/composicion_01.pas
https://gsyc.urjc.es/~mortuno/fpi/composicion_02.pas
https://gsyc.urjc.es/~mortuno/fpi/composicion_03.pas