



# Tema 10. Archivos

- 10.1. Introducción
- 10.2. Definición
- 10.3. Clasificación
- 10.4. Archivos como parámetros
- 10.5. Archivos de texto
- 10.6. Archivos Binarios
- 10.7. Manipulación de archivos y directorios
- 10.8. Control de errores de E/S
- 10.9. Operaciones usuales con archivos



# 10.1. Introducción

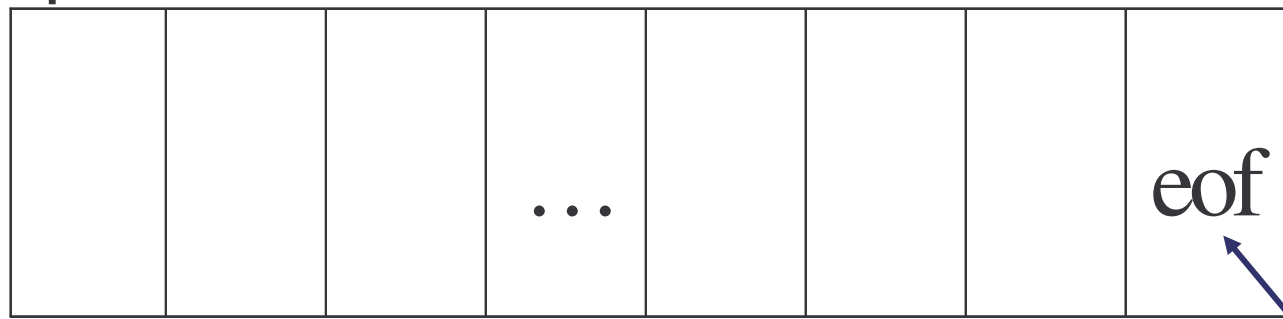
## Definición.

- ✓ Secuencias homogéneas de datos de tamaño no fijado de antemano que facilitan
  - ▶ El manejo de grandes cantidades de datos
  - ▶ La persistencia de los datos, más allá del momento de ejecución de un programa
  - ▶ Uso de datos en diversos programas
- ✓ El almacenamiento de los datos se realiza en memoria externa o auxiliar



# 10.1. Introducción

- ✓ Archivo  $\approx$  Fichero
- ✓ Archivo estándar de entrada (Input)
- ✓ Archivo estándar de salida (Output)
- ✓ Se pueden representar como una fila de celdas en las que se almacenan los datos que componen los archivos





# 10.1. Introducción

- ✓ El tamaño de los archivos
  - ▶ no se declara con anticipación
  - ▶ puede variar durante la ejecución del programa
  - ▶ limitado sólo por el medio de almacenamiento en el que persisten
- ✓ El espacio necesario para almacenar los datos se asigna de forma dinámica



## 10.2. Declaración

- ✓ **Nombre físico** de un archivo: el nombre que tiene asignado en memoria externa (disco, etc.)
- ✓ **Nombre lógico**: identificador válido Pascal asociado a un archivo o nombre físico
- ✓ Es necesario asociar los Archivos Lógicos o nombres lógicos con sus correspondientes Archivos Físicos (identificador en el disco, etc.)



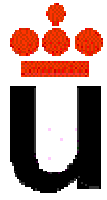
## 10.2. Declaración

- ✓ Relación nombre físico-nombre lógico en Turbo Pascal

```
Assign (ArchivoLogico, ArchivoFisico);
```

- ✓ Ejemplo:

```
Assign (archivoTarjetas, 'C:\tar\datos.txt');
```

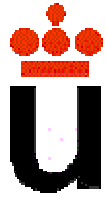


## 10.2. Declaración

- ✓ En Pascal estándar los archivos (nombres lógicos) utilizados por un programa se incluyen en la lista de parámetros de la declaración del programa

```
Program NombrePrograma (input, output,  
    nombreArchivo1, ... nombreArchivoN);  
  
nombreArchivo1 ... nombreArchivoN => nombres lógicos
```

- ✓ En Turbo Pascal no es necesario



## 10.3. Clasificación

### Categorías de Archivos: Criterio

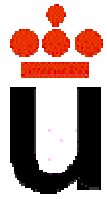
✓ **Forma de almacenamiento** de los datos:

▶ **Archivos de texto**

Son archivos de secuencias de caracteres de longitud variable (líneas) separadas por el carácter de *fin de línea* (EOLN).

▶ **Archivos binarios**

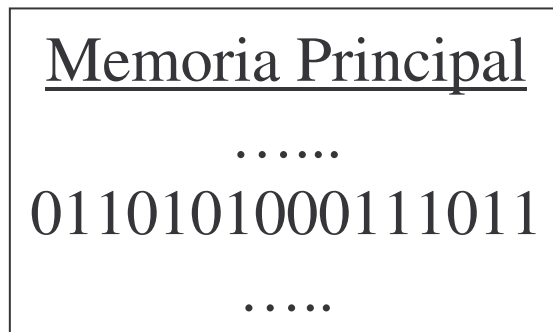
Archivos cuyos componentes se almacenan en la representación interna de la máquina; es decir, con los mismos patrones de bits con los que se almacenan en memoria principal



# 10.3. Clasificación

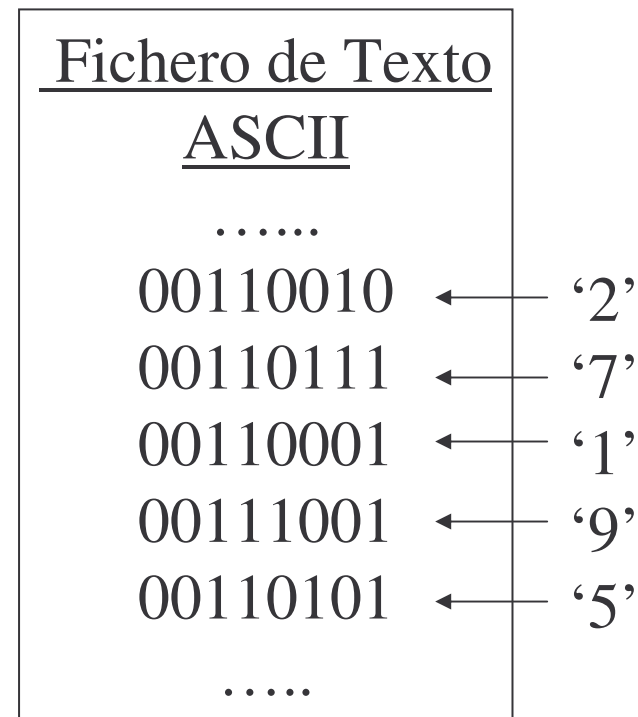
## Archivos de Texto

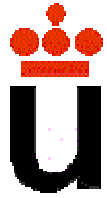
N:=27195



En Texto

‘2’‘7’‘1’‘9’‘5’





# 10.3. Clasificación

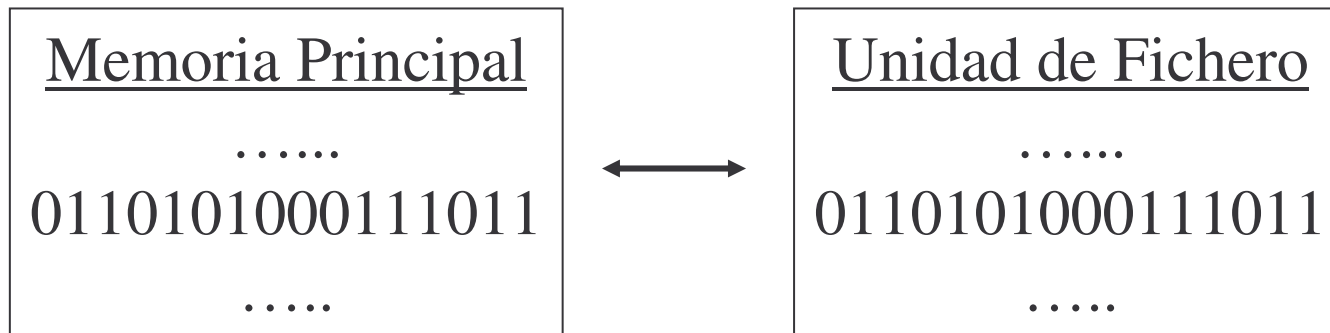
## Archivos Binarios

Entero

N:=27195

En binario

N:=0110101000111011





## 10.3. Clasificación

### Categorías de Archivos: Criterio

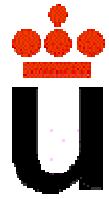
- ✓ **Método de acceso a los datos:**
  - ▶ **Acceso secuencial:** para acceder a una determinada información del fichero hay que acceder previamente a la información precedente
  - ▶ **Acceso directo** (No en Pascal estándar) requiere de soportes direccionables y permite acceder directamente a una posición concreta del soporte



## 10.4. Archivos como Parámetros

### Paso de Archivos como Parámetros

- ✓ Se pasan **siempre por Referencia**
- ✓ Si se pasaran por valor equivaldría a asignar una variable de tipo archivo a otra variable de tipo archivo: NO está permitido en Pascal



## U 10.5. Archivos de Texto

**Definición.** La información se almacena utilizando un alfabeto de texto (ASCII, EBCDIC, ...)

- ✓ Se declaran como variables del tipo predefinido **text**

VAR

```
archivoTexto : text;
```

(Ejemplo 10.1)



## 10.5. Archivos de Texto

- ✓ Los archivos *Input* y *Output* son archivos de tipo texto
- ✓ Todos los archivos de texto están organizados igual que los archivos *Input* y *Output*:
  - ▲ Son secuencias de caracteres
  - ▲ Están divididos en líneas
  - ▲ Cada línea termina con una marca de fin de línea (EOLN)
  - ▲ El archivo completo termina con una marca de fin de archivo (EOF)



## 10.5. Archivos de Texto

- ✓ Todo fichero antes de ser procesado debe ser "abierto" y una vez procesado debe ser "cerrado"
- ✓ Los modos de apertura varían en función del tipo de operaciones (lectura, escritura) que se realizarán con el fichero



## 10.5. Archivos de Texto

### Apertura para Lectura

`Reset (VarArchivoTexto) ;`

- ✓ Este procedimiento abre el archivo sólo para lectura; no se puede escribir en él
- ✓ Presupone que el fichero existe, en caso contrario se produce un error



## 10.5. Archivos de Texto

### Apertura para Lectura

- ✓ Cada llamada a Reset coloca el apuntador de los datos al principio del archivo
- ✓ No es necesario abrir el archivo *Input*
- ✓ En caso de que el archivo no exista y dado que se presupone su existencia al intentar abrirlo, se producirá un error de ejecución
- ✓ Hay formas de evitarlos con directivas especiales (se verá más adelante 10.8)

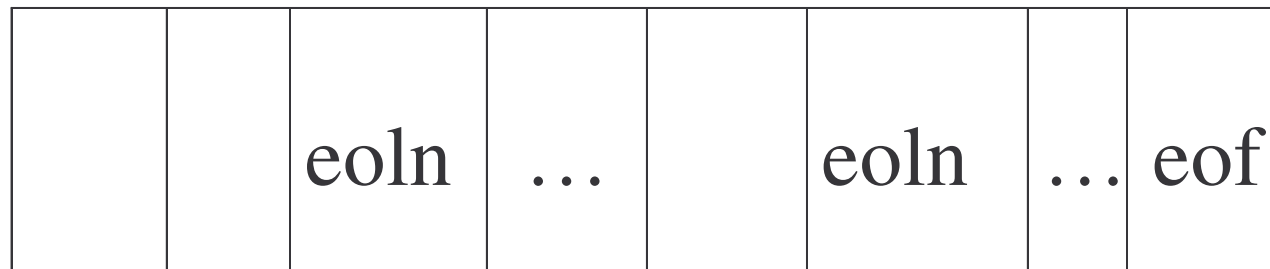


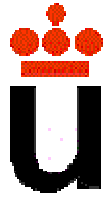
## 10.5. Archivos de Texto

### Lectura

Se lleva a cabo mediante `Read` o `ReadLn`  
(utilizando el formato explícito)

```
Read (VarArchivoTexto, listaParametros);  
ReadLn (VarArchivoTexto, listaParametros);
```





## 10.5. Archivos de Texto

### Lectura

- ✓ El funcionamiento de `Read` y `Readln` con respecto a los archivos de texto es similar al que presentan con respecto a la entrada estándar (teclado)
- ✓ El procedimiento `Readln` se puede utilizar con ficheros de texto, pero no con ficheros binarios
- ✓ El procedimiento `Readln` lee el contenido del archivo hasta la marca de fin de línea (eoln) incluida



## 10.5. Archivos de Texto

### Lectura

Cuando se ejecuta una instrucción de entrada:

- ✓ se leen secuencias de caracteres del archivo especificado
- ✓ en caso necesario se convierten al tipo de las variables expresadas en `listaParametros`

```
Read (VarArchivoTexto, listaParametros);  
ReadLn (VarArchivoTexto, listaParametros);
```



## 10.5. Archivos de Texto

- ✓ Hay dos funciones asociadas a la detección de la marca de fin de línea y de fin de fichero

**EOLN (varArchivo)**

**EOF (varArchivo)**



## 10.5. Archivos de Texto

✓ **FUNCTION EOLN (varArchivo: boolean**

devuelve valor TRUE si el apuntador o puntero de `varArchivo` está sobre la marca de fin de línea y FALSE en caso contrario

✓ **FUNCTION EOF (varArchivo) : boolean**

devuelve valor TRUE si el apuntador o puntero de `varArchivo` está sobre la marca de fin de archivo y FALSE en caso contrario



## 10.5. Archivos de Texto

### Lectura

- ✓ Si se quieren leer todas las líneas del fichero habrá que incluir en el cuerpo de un bucle la instrucción de lectura:

```
ReadLn (VarArchivoTexto, listaParametros);
```

- ✓ La condición de salida del bucle se producirá cuando se detecte el fin del fichero

**(Ejemplo 10.3)**



## 10.5. Archivos de Texto

### Apertura para Escritura

`Rewrite (ArchivoTexto);`

- ✓ Este procedimiento abre el archivo sólo para escritura, no para lectura, y coloca el apuntador al principio del archivo
- ✓ Cada llamada a `Rewrite` destruye el contenido previo del archivo (si es que existe) y, una vez ejecutada, el archivo queda preparado para escribir en él
- ✓ No es necesario abrir el archivo *Output* para escribir en él



## 10.5. Archivos de Texto

### Escritura

Se lleva a cabo mediante `Write` o `WriteLn`  
(utilizando el formato explícito)

```
Write (ArchivoTexto, listaParametros);
```

```
WriteLn (ArchivoTexto, listaParametros);
```



## 10.5. Archivos de Texto

### Escritura:

- ✓ El funcionamiento de `Write` y `Writeln` con respecto a los archivos de texto es similar al que presentan con respecto a la salida estándar (pantalla)
- ✓ El procedimiento `Writeln` se puede utilizar con ficheros de texto, pero NO con ficheros binarios



## 10.5. Archivos de Texto

### Escritura:

- ✓ Cuando se ejecuta una instrucción de salida:
  - ▲ se escribe el contenido de `listaParametros`
  - ▲ si la instrucción es `writeln` se añade una marca de fin de línea después de escribir el último parámetro

(Ejemplo 10.4)



## 10.5. Archivos de Texto

### Cierre de ficheros (Turbo Pascal)

```
Close (ArchivoTexto);
```

- ✓ Asegura que todos los datos del "buffer" de entrada o de salida quedan leídos o escritos correctamente.
- ✓ Se inhabilita la relación establecida con "Assign"



## 10.5. Archivos de Texto

Otro tipo de apertura de Ficheros de Texto

✓ `Append (var f: text);`

Abre el archivo de texto en modo escritura para poder añadir más texto al final del mismo.



## 10.6. Archivos binarios

- ✓ La información se almacena utilizando la representación binaria interna
- ✓ Los archivos almacenan datos homogéneos: todas sus componentes son del mismo tipo
- ✓ Los procedimientos `Assign`, `Rewrite`, `Read` y `Write`, así como la función `EOF`, trabajan igual con los archivos binarios que con los de texto



## 10.6. Archivos binarios

- ✓ No se utilizan funciones y procedimientos predefinidos relacionados con manejo de “líneas”, ya que no están divididos en líneas.
  - ✓ La función `EOLN` no se utiliza con ficheros binarios
  - ✓ `WriteLn` no se utiliza con ficheros binarios
  - ✓ `ReadLn` no se utiliza con ficheros binarios

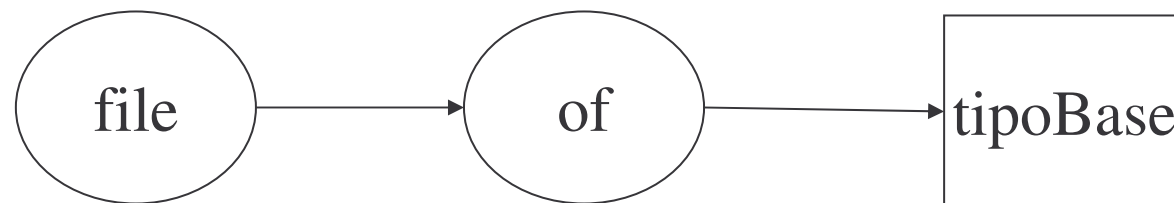


## 10.6. Archivos binarios

### Definición de Tipo Fichero

#### **TYPE**

TipoArchivo = File OF tipoBase;



tipoBase puede ser de cualquier tipo excepto fichero



## 10.6. Archivos binarios

### Declaración:

```
PROGRAM Programa;
```

```
VAR
```

```
    archivo: TipoArchivo;
```

```
    archivo2: FILE OF tComponente;
```

```
{Se puede declarar una variable con tipo anónimo}
```

```
BEGIN
```

```
    assign(archivo, 'c:\Alumnos.dat');
```

```
    . . .
```

```
END;
```



## 10.6. Archivos binarios

### Apertura para Escritura

`Rewrite (ArchivoBinario);`

- ✓ Este procedimiento abre el archivo sólo para escritura, no para lectura, y coloca el apuntador al principio del archivo
- ✓ Cada llamada a `Rewrite` destruye el contenido previo del archivo y, una vez ejecutada, el archivo queda preparado para escribir en él



## 10.6. Archivos binarios

### Escritura

```
Write (varFichero, varComponente) ;
```

- ✓ Escribe el contenido `varComponente`  
**en** `varFichero`



## 10.6. Archivos binarios

### Apertura para Lectura

`Reset (archivoBinario);`

- ✓ es una apertura que con ficheros binarios permite tanto operaciones de lectura como de escritura (no es así para archivos de texto)
- ✓ Cada llamada a `Reset` coloca el apuntador de los datos al principio del archivo



## 10.6. Archivos binarios

### Lectura

`Read (varArchivo, varComponente) ;`

- ✓ Lee el contenido de `varComponente` desde `varArchivo`



## 10.6. Archivos binarios

### **Ficheros de acceso directo** (Turbo Pascal)

- ✓ En Pascal no existe una declaración explícita de la organización de un fichero (secuencial, directa, ...)
- ✓ Los ficheros binarios pueden ser procesados indistintamente como ficheros secuenciales o como ficheros directos
- ✓ La única condición necesaria para que un fichero pueda ser tratado con acceso directo es que esté almacenado en un soporte direccionable



## 10.6. Archivos binarios

### Ficheros de acceso directo (Turbo Pascal)

- ✓ El acceso directo a un registro se realiza situando el puntero del fichero en la posición en la que se quiere hacer la operación de lectura o escritura
- ✓ El posicionamiento del puntero se realiza con el procedimiento **SEEK**



## 10.6. Archivos binarios

### Ficheros de acceso directo (Turbo Pascal)

```
SEEK (varFichero, puntero)
```

varFichero: tipo fichero binario

Puntero: longint (entero largo)

- ✓ Sitúa el puntero en la posición que indica *puntero*.
- ✓ Recuérdese que el primer registro ocupa la posición 0.



## 10.6. Archivos binarios

### Ficheros de acceso directo (Turbo Pascal)

Funciones predefinidas que facilitan el acceso directo:

✓ `FILEPOS (var_fichero) : longint`

Devuelve el valor del puntero

✓ `FILESIZE (var_fichero) : longint`

Devuelve el número de registros que tiene el fichero.



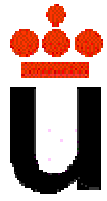
## 10.7. Manipulación de archivos y directorios

Existen otros procedimientos aplicables a todo tipo de archivos (texto y binarios).

✓ `GetDir(D: Byte; var S: String);`

Devuelve el directorio actual de la unidad especificada por D, que toma valores 0 (unidad por defecto), 1 (unidad A), 2 (unidad B),...

El directorio se devuelve en s



## 10.7. Manipulación de archivos y directorios

✓ `ChDir (S: String) ;`

Permite cambiar el directorio actual de trabajo por el indicado en `s`

✓ `MkDir (S:String) ;`

Permite crear un directorio con el nombre indicado en `s`

✓ `Rmdir (S: String) ;`

Permite eliminar el directorio (debe estar vacío) indicado en `s`



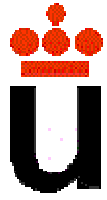
## 10.7. Manipulación de archivos y directorios

- ✓ `Rename (var f; NuevoNombre:String);`  
Permite cambiar el nombre externo del fichero `f` por la cadena `NuevoNombre`
- ✓ `Erase (var F);`  
Permite borrar el fichero asociado a la variable `f`  
Se debe usar con un fichero cerrado



## 10.8. Control errores E/S

- ✓ Turbo Pascal dispone de un sistema de detección de errores de E/S (aperturas, lectura, escritura) que por defecto está activado ( `{ $I+ }` )
- ✓ Errores comunes:
  - ✓ Abrir en modo lectura un fichero que no existe
  - ✓ Intentar leer más allá de la marca EOF
- ✓ Si se produce un error de este tipo el programa detiene su ejecución y muestra un mensaje de error



## 10.8. Control errores E/S

- ✓ Para evitar que el programa detenga su ejecución se puede desactivar el sistema de detección de errores (`{ $I- }`) y realizar ese control desde el propio programa
- ✓ Este control se realiza con la función `IOResult`
  - ✓ Devuelve valor 0 si la última operación de E/S se ha realizado con éxito
  - ✓ Otro valor en caso contrario



# 10.9. Operaciones usuales con ficheros

## Fusión

- ✓ Consiste en agrupar en un fichero los registros de dos o más ficheros que se encuentran ordenados por un determinado campo clave
- ✓ El fichero resultante debe estar ordenado también por el mismo campo clave.
- ✓ Ejemplo. tenemos dos ficheros F1 y F2 y sus campos clave son:

F1: 2, 3, 6, 8, 9, 12, 15

F2: 1, 3, 5, 7, 10

El fichero mezcla de los dos anteriores FF debería contener:

FF: 1, 2, 3, 3, 5, 6, 7, 8, 9, 10, 12, 15

MTP 2006-2007

(Ejemplo 10,12)



# 10.9. Operaciones usuales con ficheros

## Partición

- ✓ División de un fichero en 2 o más ficheros de acuerdo a un determinado criterio
- ✓ Criterios comunes:
  - ✓ **Partición por contenido:** la partición se realiza en función del contenido de uno o más campos de los registros
  - ✓ **Partición en secuencias de longitud N:** se escriben N registros del original en cada fichero destino