Tema 6
Registros y Ficheros

Grado de Ingeniería Informática

Introducción a la Programación

Registros y Ficheros

- Registros
  - 6.1. Definición
  - 6.2. Sintaxis
  - 6.3. Operaciones de acceso
    - 6.3.1 Registros completos (Asignación)
    - 6.3.2 Campos del registro
  - 6.4. Instrucción WITH

Registros y Ficheros

- Ficheros
  - 6.1. Introducción
  - 6.2. Declaración
  - 6.3. Clasificación
  - 6.4. Ficheros como parámetros
  - 6.5. Ficheros de texto
  - 6.6. Ficheros binarios
  - 6.7. Manipulación de ficheros
  - 6.8. Control de errores de E/S
  - 6.9. Operaciones usuales con ficheros

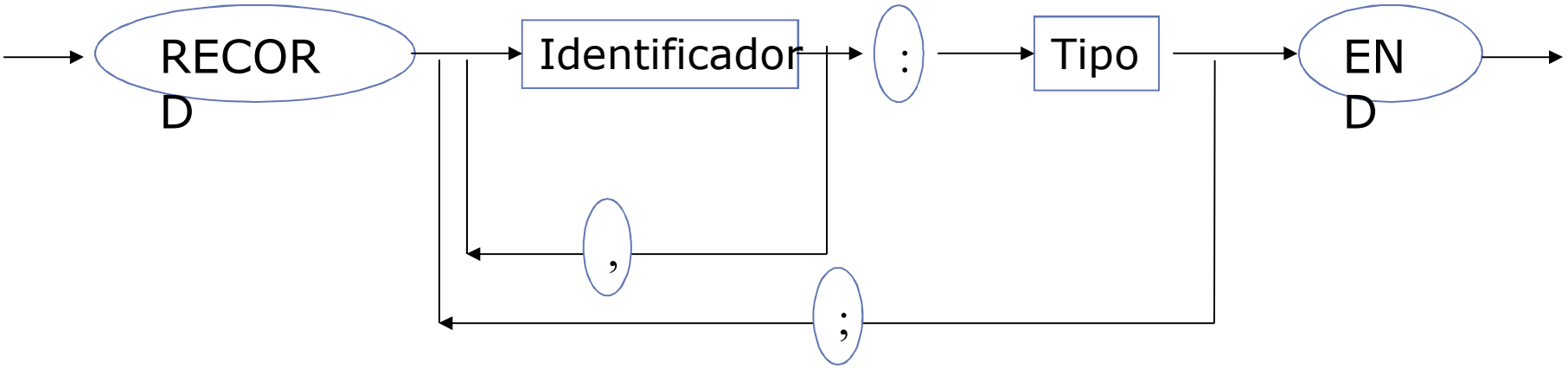
Registros

6.1. Definición

- **Definición**:
  Tipo de datos estructurado que permite almacenar datos heterogéneos
- Cada uno de estos datos se denomina **campo** del registro.
- Cada campo tiene un nombre llamado *identificador de campo.*

6.2. Sintaxis

Diagrama sintáctico



6.2. Sintaxis

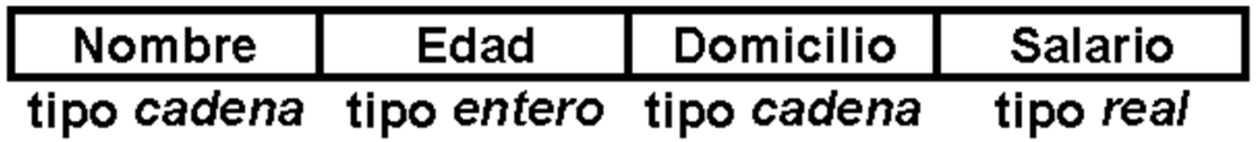
Sintaxis

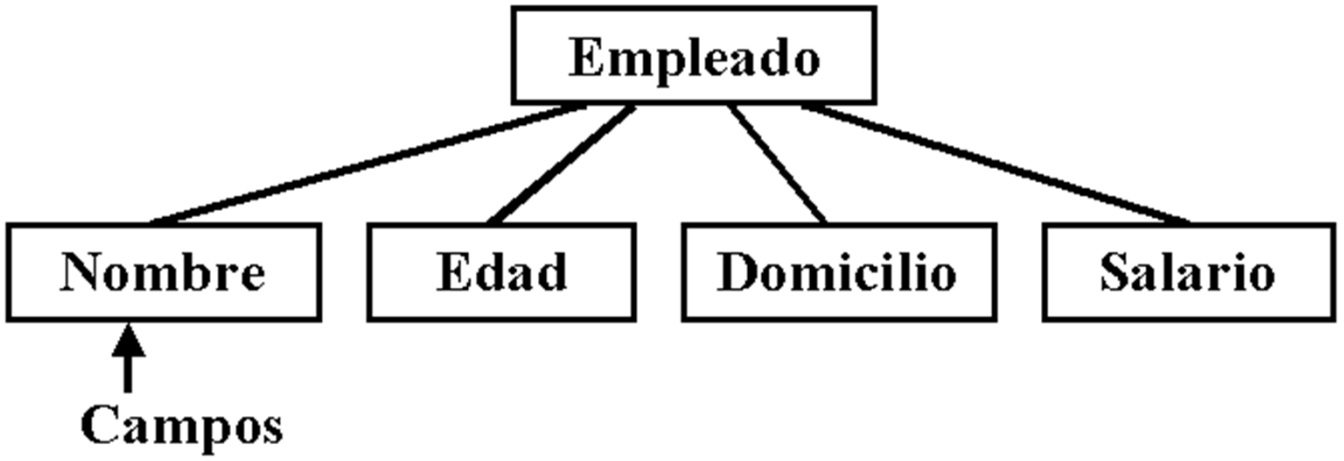
```
TYPE
     TNombreReg = RECORD
          idCampo1 : idTipo1;
          idCampo2 : idTipo2;
                  ...
          idCampoN : idTipoN
     END; {Fin de TNombreReg}
```

6.2. Sintaxis

- El tipo de los campos puede ser cualquier tipo predefinido de Pascal o uno definido previamente por el programador

Empleado:





Ejemplo 6.1

```
CONST
        N = 50;
        INI = 16;
        FIN = 65;
TYPE
        TEdades = INI..FIN;
        TCadena = string[N];
        TEmpleado = RECORD
                Nombre    : TCadena;
                Edad      : TEdades;
                Domicilio : TCadena;
                Salario   : real;
        END;
```

Ejemplo 6.2

- El tipo de los campos también puede ser de tipo registro:

```
CONST
   N = 30;
   M = 10;
TYPE
   TCadena = string[N];
   TMatricula = string[M];
   TCoche = RECORD
       Marca, Modelo : TCadena;
   END; {TCoche}
   TAutomovil = RECORD
       Matricula : TMatricula;
       Nombre    : TCoche; {Reg. Anidados}
       Precio    : real
   END; {TAutomovil}
```

6.2. Sintaxis

- No puede utilizarse el mismo identificador para especificar dos campos distintos de un mismo registro.
- Dos campos de registros distintos pueden tener el mismo identificador.

Ejemplo 6.3

- Ejemplo:

```
CONST
  N = 30;
TYPE
  TCadena = string[N];
  TEmpleado = RECORD
    Nombre : TCadena; {Nombre del empleado}
    DNI    : integer;
    Nombre : TCadena; {Nombre de la empresa}
                                {ERROR}
  END; {TEmpleado}
```

Ejemplo 6.4

```
CONST
   N = 30;
   D = 10;
   INI = 1;
   FIN = 8;
TYPE
   TNombre = string[N];
   TDNI = string[D];
   TAsignaturas = INI..FIN;
   TNotas = array [TAsignaturas] OF real;
   TEmpleado = RECORD
      Nombre, Apellido : TNombre; {Nombre}
      DNI              : TDNI;{del empleado}
   END; {TEmpleado}
   TAlumno = RECORD
      Nombre : TNombre; {Nombre del alumno}
      DNI    : TDNI;
      Notas  : TNotas; {Notas}
   END; {TAlumno}
```

6.2. Sintaxis

- Ejercicio:
- Escribir una declaración de tipos que permita almacenar la información relativa a los trabajadores de una organización de forma cómoda y controlada.
- La información relativa a cada trabajador será la siguiente: nombre, dirección, edad, número de DNI y sueldo.

Ejemplo 6.5

```
CONST
  N = 50;
  IPOS = 1;
  FPOS = 8;
  INI = 18;
  FIN = 65;
TYPE
  TEdad = INI..FIN;
  TDigito = '0'..'9';
  TPosicion = IPOS..FPOS;
  TNombre = string[N];
  TDNI = array [TPosicion] OF TDigito;
```

Ejemplo 6.5 (y 2)

```
[...]

  TEmpleado = RECORD
     Nombre, Direccion : TNombre;
     Edad      : TEdad;
     DNI       : TDNI;
     Sueldo    : real;
  END; {Fin de TEmpleado}

VAR
 empleado, supervisor : TEmpleado
```

6.3. Operaciones

- **Asignación:**
- A las variables de tipo registro se les puede asignar **registros completos** si las dos variables son de **tipos idénticos**

Ejemplo 6.6

- Basándonos en la definición de tipos del ejemplo 6.5:

```
VAR
  Empleado1, Empleado2 : TEmpleado;

  BEGIN
  ...

      Empleado2 := Empleado1;
```

6.3. Operaciones

- **Acceso a los campos de un registro**:
- Para **acceder** a un **campo** de una variable de tipo registro se utiliza el **operador punto** (.)
  - NombreVariableRegistro.NombreCampo
- Donde **.NombreCampo** =>Selector de Campo

Ejemplo 6.7

```
VAR
  Empleado1, Empleado2: TEmpleado;

BEGIN
  ...

  Empleado1.Nombre:= 'Juan Pérez';
  readln(Empleado2. Direccion);
  Empleado1.Edad:= Empleado2.Edad;

  ...
```

6.3. Operaciones

- **Acceso a los campos de un registro**:

- La vigencia de un nombre de campo es la misma que la del registro en el que se ha definido.

- Con un campo de una variable de tipo registro se pueden usar todos los operadores que se utilizan con variables de su mismo tipo.

6.3. Operaciones

- **Acceso a los campos de un registro**:
- **Asignación**:
- La asignación de valores a los campos de una variable de tipo registro se hace igual que la asignación en variables del tipo de datos del campo selector.
- **Lectura y Escritura**:
- Las operaciones de **lectura** y **escritura** de una variable de tipo registro se hacen **campo a campo**.

Ejemplo 6.8

```
VAR
  Empleado1, Empleado2:TEmpleado;

BEGIN
  ...
  Empleado1.Sueldo := 1250.0;
  Empleado1.Nombre := 'Juan Jose';
  Empleado2.Sueldo := Empleado1.Sueldo +
500.0;
  Empleado2.Edad   := 31;
  Empleado2.DNI    [3] := '5';
  ...
```

Ejemplo 6.9

- **Lectura**:

```
VAR
    Empleado1: TEmpleado;

BEGIN
  ...
  readln(Empleado1.Nombre);
  readln(Empleado1.Direccion);
  readln(Empleado1.Edad);
  FOR i:=Ini TO Fin DO
     read(Empleado1.DNI[i]);
  readln(Empleado1.Sueldo);
  ...
```

Ejemplo 6.9

- **Escritura**:

```
VAR
  Empleado1: TEmpleado;
BEGIN
 ...
  writeln('El nombre es: ', Empleado1.Nombre);
  writeln('vive en: ', Empleado1.Direccion);
  writeln('Tiene ',Empleado1.Edad, ' años');
  writeln('Su documento nacional de identidad
  es:');
  FOR i:=INI TO FIN DO
     write(Empleado1.DNI[i]);
  writeln('Gana ',Empleado1.Sueldo,'€/mes');
 ...
```

6.3. Operaciones

- **Variables de tipo registro como parámetros:**
- Se pueden utilizar como parámetros de subprogramas.
- El tipo de los parámetros formales debe coincidir con el tipo de los parámetros reales.
- CUIDADO!! $\Rightarrow$ Las funciones **NO** pueden devolver una variable de tipo registro.

Ejemplo 6.10

```
TYPE
  TEjemplo = RECORD
     X : integer;
     Y : real;
  END;
VAR
  uno, dos : TEjemplo;
PROCEDURE Prueba1(a: TEjemplo; VAR
  b:TEjemplo);
BEGIN
  ...
END;
FUNCTION Prueba2(m: TEjemplo) : TEjemplo;
```

6.4. Instrucción WITH

- **Instrucción WITH:**
- Evita la repetición del nombre de la variable de tipo registro cuando utilizamos seguidamente varios campos de un mismo registro.

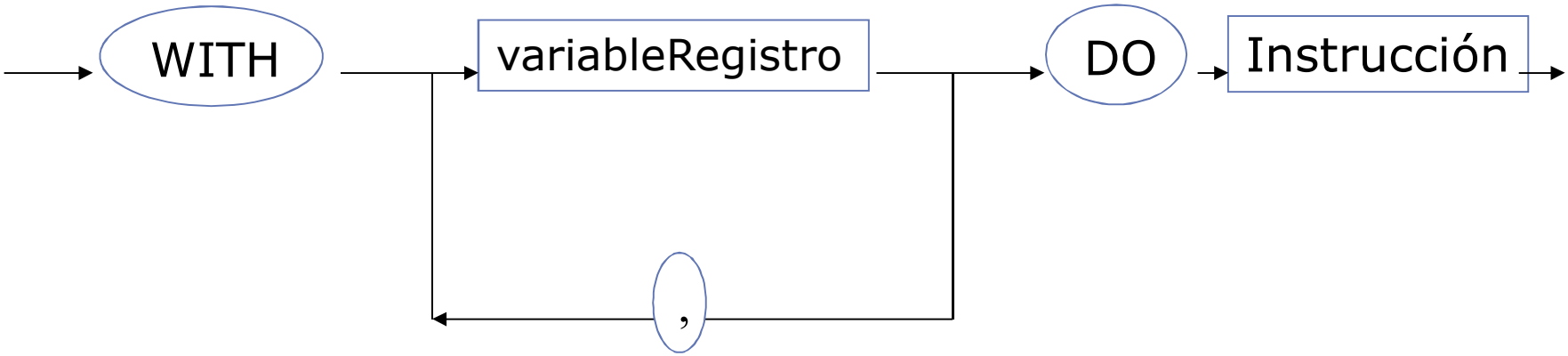
- **Sintaxis:**

WITH variableRegistro **DO**

Instrucción | secuencia de instrucciones

6.4. Instrucción WITH

Diagrama sintáctico



Ejemplo 6.11

```
CONST
     N = 30;
     INI = 18;
     FIN = 65;
TYPE
     TCadena = string[N];
     TEdades = INI..FIN;
     TFicha = RECORD
          Nombre: TCadena;
          Edad: TEdades;
          Sueldo: real;
     END;
VAR
  Profesor : TFicha;
```

Ejemplo 6.11 (y 2)

```
...

BEGIN
   ...

   WITH Profesor DO
      writeln(Nombre,Edad,Sueldo);
                  ||


   writeln (Profesor.Nombre,
      Profesor.Edad, Profesor.Sueldo);
```

6.4. Instrucción WITH

- Los identificadores que no sean selectores de campo se tratan como si no existiese la instrucción WITH.
- No debe usarse la instrucción WITH con variables que tengan campos con identificadores iguales.

Aplicación de los registros

- Podemos simular arrays parcialmente llenos (APL) mediante un registro.

```
TYPE
 TRango = ...;
 TBase  = ...;
 TProducto = array [TRango] OF TBase;
 TRegArray = RECORD
             tope     : integer;
             producto : TProducto;
 END;
VAR
   almacen : TRegArray;
```

Aplicación de los registros

```
BEGIN {PP}
      ...
  WITH almacen DO
  BEGIN
     tope := tope + 1;
    producto[tope] := producto[i];
   END;
        ...
  FOR j:= 1 TO almacen.tope DO
    Escribe_info(almacen.producto,j);
        ...
END. {PP}
```

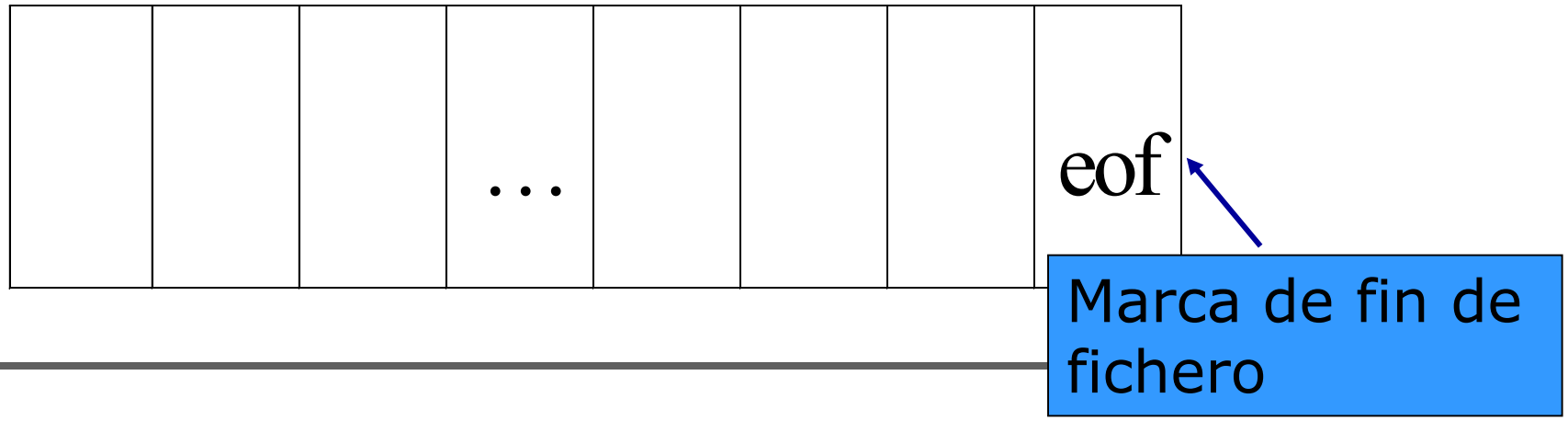
Ficheros

6.1. Introducción

- **Definición**
- ✓ Son 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
  - El uso de los datos en diversos programas
- ✓ El almacenamiento de los datos se realiza en la memoria externa o auxiliar

6.1. Introducción

- Fichero ≈ Archivo
- Fichero estándar de entrada (**input**)
- Fichero estándar de salida (**output**)
- Se pueden representar como una fila de celdas en las que se almacenan los datos que componen los ficheros



6.1. Introducción

- El tamaño de los ficheros
  - No se declara con antelación
  - Puede variar durante la ejecución del programa
  - Solo está limitado por el soporte de almacenamiento en el que persisten
- El espacio necesario para almacenar los datos se asigna de forma dinámica

6.2. Declaración

- **Nombre físico** de un fichero: Es el nombre que tiene asignado en la memoria externa (disco, etc.)
- **Nombre lógico**: Es un identificador válido en Pascal asociado al nombre físico del fichero

- Es necesario **asociar** los **ficheros lógicos** o nombres lógicos con sus correspondientes **ficheros físicos** (identificador en el disco, etc.)

6.2. Declaración

- Para establecer esa relación entre nombre físico y nombre lógico en Turbo Pascal

```
Assign (FicheroLogico,
FicheroFisico);
```

- Ejemplo:

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

6.2. Declaración

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

```
Program NombrePrograma (input,  output, nombreFichero1,
... nombreFicheroN);

nombreFichero1 ... nombreFicheroN => nombres lógicos
```

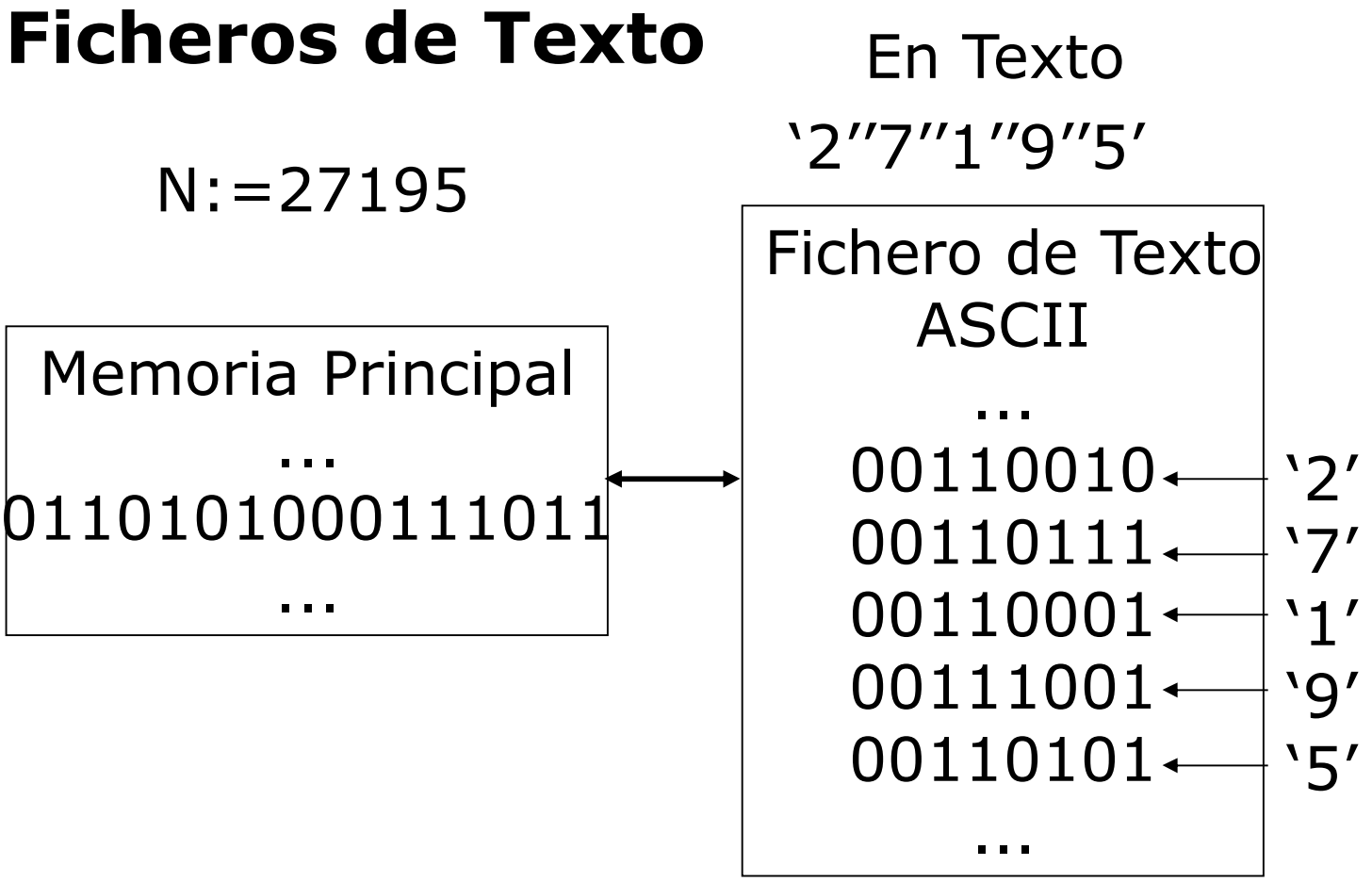
- En Turbo Pascal no es necesario

6.3. Clasificación

- Categorías de ficheros:
- ✓ Atendiendo al criterio de la **forma en que se almacenan** los datos:
  - **Ficheros de texto**
    - Son ficheros de secuencias de caracteres de longitud variable (líneas) separadas por el carácter de **fin de línea** (EOLN).
  - **Ficheros binarios**
    - Son ficheros 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.

6.3. Clasificación

Ficheros de Texto



6.3. Clasificación

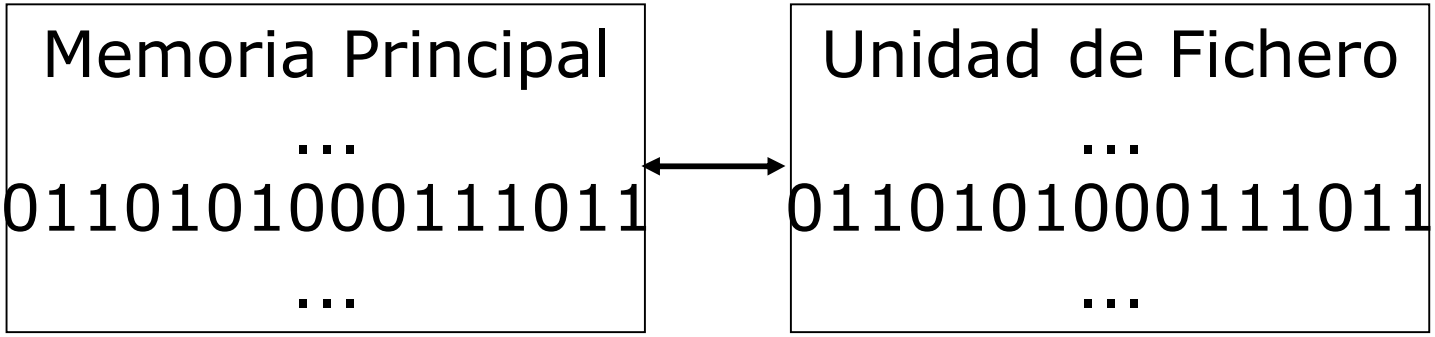
Ficheros Binarios

Entero

N:=27195

En binario

N:=0110101000111011



6.3. Clasificación

- Categorías de ficheros:
- ✓ Atendiendo al criterio del **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.

6.4. Ficheros como parámetros

- Paso de ficheros como parámetros:

  - ✓ Se pasan **siempre** por **referencia**

  - ✓ Si se pasaran por valor equivaldría a asignar una variable de tipo fichero a otra variable de tipo fichero y esto **NO** está permitido en Pascal

6.5. Ficheros de texto

- **Definición**
  - La información se almacena utilizando un alfabeto de texto (ASCII, EBCDIC, ...)
  - Se declaran como variables del tipo predefinido **text**

```
VAR
     ficheroTexto : text;
```

Ejemplo 6.12

```
PROGRAM ProgramaEjemploTexto;

VAR
          archivoTexto : text ;
...
BEGIN

ASSIGN
(archivoTexto,'c:\Fichero.txt');
...
END;
```

6.5. Ficheros de texto

- Los ficheros **input** y **output** son ficheros de tipo texto

- Todos los ficheros de texto están organizados igual que los ficheros **input** y **output**:
  - Son secuencias de caracteres
  - Están divididos en líneas
  - Cada línea termina con una marca de fin de línea (EOLN)
  - El fichero completo termina con una marca de fin de fichero (EOF)

6.5. Ficheros 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 o escritura) que se realizarán con el fichero

6.5. Ficheros de texto

Apertura para lectura

```
Reset (VarFicheroTexto);
```

- Este procedimiento abre el fichero solo para su lectura. No se puede escribir en él.
- Se presupone que el fichero existe y, en caso contrario, se produce un error.

6.5. Ficheros de texto

Apertura para lectura

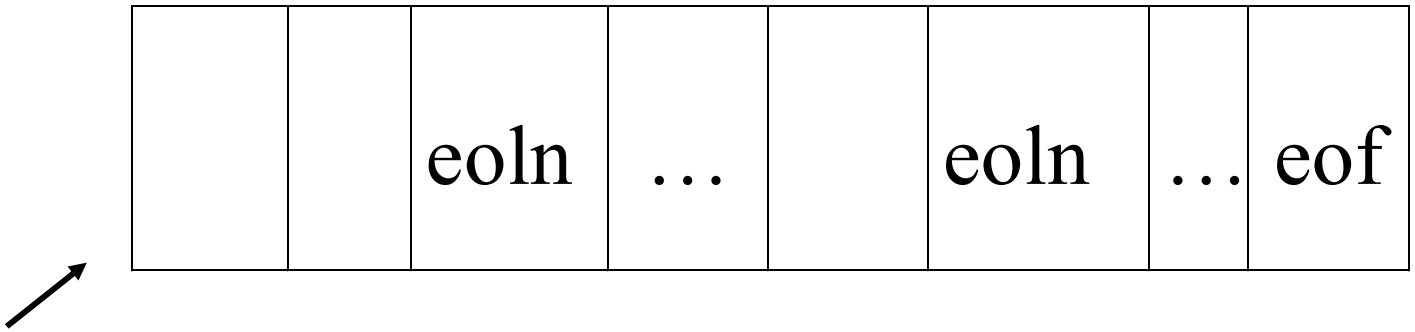
- Cada llamada a **reset** coloca el apuntador de los datos al principio del fichero
- No es necesario abrir el fichero **input**
- En caso de que el fichero no exista, dado que se presupone su existencia al intentar abrirlo, se producirá un error de ejecución
- Hay formas de evitarlos con directivas especiales que se verán más adelante (6.8)

6.5. Ficheros de texto

Lectura

Se lleva a cabo mediante **read** o **readln** (utilizando el formato explícito)

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



6.5. Ficheros de texto

Lectura

- El funcionamiento de **read** y **readln** con respecto a los ficheros 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 fichero hasta la marca de fin de línea (EOLN) incluida

6.5. Ficheros de texto

Lectura

- Cuando se ejecuta una instrucción de entrada:
  - ✓ Se leen secuencias de caracteres del fichero especificado
  - ✓ En caso necesario, se convierten al tipo de las variables expresadas en `listaParametros`

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

Ejemplo 6.13

- Supongamos que tenemos la siguiente información en un fichero de texto:

A 334.33\n
B 33.2\n
C 44.3\n
...

El primer elemento de la línea es un dato de tipo char

El segundo es de tipo real

Ejemplo 6.13 (y 2)

```
VAR
  varArchivo: text;
  c: char;
  num: real;

BEGIN
  ASSIGN (varArchivo,'c:\Fichero.txt');
  {leer una línea del fichero varArchivo}
  READLN (varArchivo, c, num)
```

Lee una línea del fichero `varArchivo`, almacena el primer dato de tipo `char` en `c` y el siguiente que es un `real` en `num`

6.5. Ficheros de texto

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

```
EOLN (varFichero)

EOF (varFichero)
```

6.5. Ficheros de texto

- **`FUNCTION EOLN (varFichero): boolean`**

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

- **`FUNCTION EOF (varFichero): boolean`**

  Devuelve valor TRUE si el apuntador o puntero de `varFichero` está sobre la marca de fin de fichero y FALSE en caso contrario

6.5. Ficheros 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 (VarFicheroTexto,listaParametros);
```

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

Ejemplo 6.14

```
VAR
   varArchivo: text;
   c: char;
   num: real;
BEGIN
   ASSIGN (varArchivo,'c:\Fichero.txt');
   RESET(varArchivo);
   WHILE NOT EOF (varArchivo) DO BEGIN
      READLN (varArchivo, c, num)
      ProcesarLinea (c, num);
   END;
   ...
END.
```

6.5. Ficheros de texto

Apertura para escritura

```
Rewrite (FicheroTexto);
```

- Este procedimiento abre el fichero solo para escribir en él, no para leerlo. Coloca el apuntador al principio del fichero
- Cada llamada a **rewrite** destruye el contenido previo del fichero (si es que existía) y, una vez ejecutado, el fichero queda preparado para escribir en él
- No es necesario abrir el fichero **output** para escribir en él

6.5. Ficheros de texto

Escritura

Se lleva a cabo mediante **write** o **writeln** (utilizando el formato explícito)

```
Write (FicheroTexto, listaParametros);

WriteLn (FicheroTexto,listaParametros);
```

6.5. Ficheros de texto

Escritura

- El funcionamiento de **write** y **writeln** con respecto a los ficheros 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

6.5. Ficheros 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 6.15

Queremos almacenar en un fichero de texto un código de tipo `char` y un valor de tipo `real` a partir de información leída del teclado:

```
BEGIN
    ASSIGN
(varArchivo,'c:\Ftexto\Fichero.txt');
    REWRITE (varArchivo);
    parar:= false;
    REPEAT
    ...
          READLN (c, num);
          WRITELN (varArchivo, c:2, num:10:2);
    ...
    UNTIL parar;
    ...
```

6.5. Ficheros de texto

Cierre de ficheros (Turbo Pascal)

```
Close (FicheroTexto);
```

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

Ejemplo 6.16

```
Program EjemploArchivo;

  VAR
     ArchivoTexto : text;
     uno, dos : real;

  BEGIN
     ASSIGN (ArchivoTexto, 'prueba.txt');
    RESET (ArchivoTexto);
    READLN (ArchivoTexto, uno, dos);
    WRITELN (uno:6:1, dos:6:1) ;
    CLOSE (ArchivoTexto)
  END.
```

6.5. Ficheros de texto

Otro tipo de apertura (ficheros de texto)

```
Append (var f: text);
```

- ✓ Abre el fichero de texto en modo escritura para poder añadir más texto al final del mismo

6.6. Ficheros binarios

- La información se almacena utilizando la representación binaria interna
- Los ficheros almacenan datos homogéneos, es decir, todas sus componentes deben ser del mismo tipo
- Los procedimientos **`Assign`**, **`Rewrite`**, **`Read`** y **`Write`**, así como la función **`EOF`**, funcionan del mismo modo tanto con los ficheros binarios como con los de texto

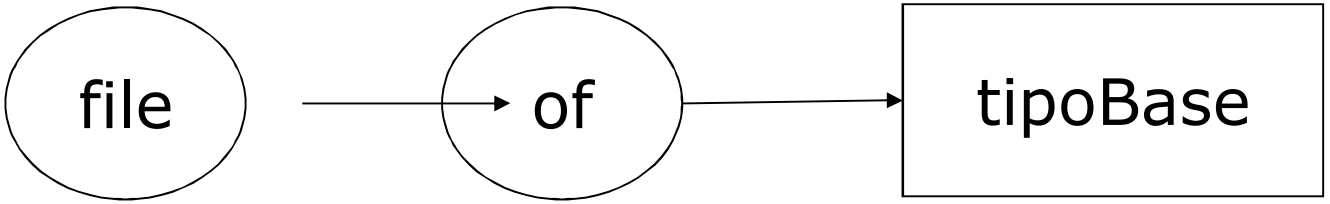
6.6. Ficheros binarios

- No se utilizan funciones y procedimientos predefinidos relacionados con el manejo de líneas puesto que estos ficheros no están divididos en líneas.
  - La función **EOLN** no se utiliza con ficheros binarios
  - **WriteIn** no se utiliza con ficheros binarios
  - **ReadIn** no se utiliza con ficheros binarios

6.6. Ficheros binarios

Declaración de tipo de fichero

```
TYPE
   TipoFichero = File OF tipoBase;
```



`tipoBase` puede ser de cualquier tipo excepto fichero

Ejemplo 6.17

```
PROGRAM Archivos;
TYPE
  tDiasSemana = (Lun, Mar, Mie, Jue, Vie,
                 Sab, Dom);
  tLista = array [1..100] OF integer;
  tCadena = string[25];
  tRegistroEmpleado = Record
                      nombre : tCadena;
                      numero, sueldoHora : real
                      END;
```

Ejemplo 6.17 (y 2)

```
tArchivoNumeros = File OF integer;

tArchivoDias    = File OF tDiasSemana;

tArchivoListas  = File OF tLista;

tArchivoEmpleados = File OF tRegistroEmpleado;

VAR
  archivoCaracteres : text;
  archivoNumeros    : tArchivoNumeros;
  archivoDias       : tArchivoDias;
  archivoListas     : tArchivoListas
  archivoEmpleados  : tArchivoEmpleados;
```

6.6. Ficheros binarios

Declaración

```
PROGRAM Programa;
VAR
     fichero: TipoFichero;
     fichero2: FILE OF tComponente;
{Se puede declarar una variable con tipo
  anónimo}
BEGIN
  assign (fichero,'c:\Alumnos.dat');
  ...
END.
```

6.6. Ficheros binarios

Apertura para escritura

```
Rewrite (FicheroBinario);
```

- Este procedimiento abre el fichero solo para su escritura, no para leerlo, y coloca el apuntador al principio del fichero
- Cada llamada a **rewrite** destruye el contenido previo del fichero y, una vez ejecutado, el fichero queda preparado para escribir en él

6.6. Ficheros binarios

Escritura

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

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

Ejemplo 6.18

```
TYPE
     t_registro= RECORD
          codigo:string[10];
          nombre:string[30];
     end;
     t_fichero = FILE of t_registro;
Var
     f: t_fichero;
     r: t_registro;
```

Ejemplo 6.18 (y 2)

```
BEGIN
    ASSIGN (f,'c:\programas\datos.dat');
    ...
    REWRITE (f);
    parar:= false;
    REPEAT
          ...
          READLN (r.codigo);
          READLN (r.nombre);
          WRITE (f,r);
          ...
    UNTIL parar;
    CLOSE(f);
    ...
```

6.6. Ficheros binarios

Apertura para lectura

```
Reset (ficheroBinario);
```

- Es una apertura que, con ficheros binarios, permite tanto operaciones de lectura como de escritura (no es así para los ficheros de texto)
- Cada llamada a **reset** coloca el apuntador de los datos al principio del fichero

6.6. Ficheros binarios

Lectura

```
Read(varFichero,varComponente);
```

- Lee el contenido de **`varComponente`** desde **`varFichero`**

Ejemplo 6.19

```
BEGIN
 ASSIGN
(f,'c:\programas\datos.dat');
 ...
 RESET(f);
 WHILE NOT EOF(f) DO BEGIN
   READ(f,r);
   WRITEN('El codigo es: 'r.codigo);
   WRITEN('El nombre es: 'r.nombre);
 END; {WHILE}
 CLOSE(f);
 ...
```

6.6. Ficheros 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

6.6. Ficheros 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`**

6.6. Ficheros 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.

Ejemplo 6.20

```
BEGIN
  ...
  Seek (varFichero, 23);
  Read (varFichero, varRegistro);
  ...

END;
```

6.6. Ficheros 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.

Ejemplo 6.21 (A)

```
BEGIN
  {Extender (añadir registros a partir del
  último) un fichero}
  ...
  SEEK (varFichero,
  FILESIZE(varFichero));
  WRITE (varFichero, varRegistro);
  ...
END;
```

Ejemplo 6.21 (B)

```
BEGIN
  {Hacer una lectura secuencial a partir de un
   determinado punto del fichero}
  ...
   SEEK (varFichero, puntero)
   WHILE NOT EOF (varFichero) DO BEGIN
      READ (varFichero, varRegistro);
      Procesar (varRegistro);
   END;
  ...
END;
```

6.7. Manipulación de ficheros

1. **`Rename (var f;`**

   `NuevoNombre:String);`

   - ✓ Permite cambiar el nombre externo del fichero **`f`** por la cadena **`NuevoNombre`**

2. **`Erase (var F);`**

   - ✓ Permite borrar el fichero asociado a la variable **`f`**. Se debe usar con un fichero cerrado

6.8. 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

6.8. 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

Ejemplo 6.22

Control de apertura con **`Reset`** de un fichero que no existe

```
FUNCTION existe (var archivo: tArchivo):
  boolean;
BEGIN
  {$I-} {se desactiva para controlar Reset}
  RESET (archivo);
  {$I+} {se activa para que detecte otros
  posibles errores que nosotros no vamos a
  controlar}
  existe := (IOResult= 0);
END;
```

Si no existe podemos mostrar un mensaje o crearlo con **Rewrite** o lo que se considere adecuado

6.9. 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

Ejemplo 6.23

```
TYPE
  Tr = RECORD
     clave: tipo1;
     ...
  END;
  Tf = FILE OF Tr;
```

Ejemplo 6.23 (2)

```
PROCEDURE Leer (VAR f: Tf; VAR r:
  Tr; VAR fin: boolean);
BEGIN
  IF NOT EOF(f) THEN
     READ (f,r)
   ELSE
       fin := true;
END;
```

Ejemplo 6.23 (3)

```
PROCEDURE FusionFicheros (VAR f1,f2,f: Tf, VAR
  parar: boolean);
VAR
  r1,r2: Tr;
BEGIN
  parar:= False; {control existencia de
  ficheros y abrir}
  IF NOT existe (f1) THEN
     parar:= True
  ELSE
     IF NOT existe (f2) THEN
     parar:= True;
  ...
```

Ejemplo 6.23 (4)

```
...
  IF NOT parar THEN BEGIN
  {se puede hacer la fusión}
     REWRITE (f);
     fin1:= false;
      fin2:= false;
       Leer (f1,r1,fin1);
       Leer (f2,r2,fin2);
```

Ejemplo 6.23 (5)

```
WHILE NOT fin1 AND NOT fin2 DO BEGIN
  IF r1.clave <= r2.clave THEN BEGIN
     WRITE (f,r1);
     Leer (f1,r1,fin1);
  END
  ELSE BEGIN
     WRITE (f,r2);
     Leer (f2,r2,fin2);
  END;
END; {WHILE}
```

Ejemplo 6.23 (6)

```
WHILE NOT fin1 DO BEGIN
 Write (f,r1);
 Leer (f1,r1,fin1);
  END;
  WHILE NOT fin2 DO BEGIN
 Write (f,r2);
 Leer (f2,r2,fin2);
  END;
```

Ejemplo 6.23 (y 7)

```
    Close(f);
    Close(f1);
    Close(f2);
END;{IF}
END; {FusionFicheros}
```

6.9. Operaciones usuales con ficheros

Partición

- División de un fichero en dos o más ficheros de acuerdo con 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

Ejemplo 6.24 (A)

```
PROCEDURE ParticionContenido (VAR f, f1,f2: Tf);
VAR
   r:Tr;
BEGIN
...{control existencia de ficheros y apertura}
WHILE NOT EOF(f) DO BEGIN
   READ (f,r);
   IF condicion (r) THEN
       WRITE (f1, r);
   ELSE
       WRITE (f2, r)
END; {WHILE}
Close (f);
Close (f1);
Close (f2);
END; {ParticionContenido}
```

Ejemplo 6.24 (B_1)

```
PROCEDURE ParticionSecuenciasN (VAR f, f1,f2: Tf, n:
   integer);
VAR
   r: Tr; cont: integer; cambio: boolean;
BEGIN
...{control existencia de ficheros y apertura}
cont := 0; cambio:= True;
WHILE NOT EOF(f) DO BEGIN
   READ (f,r);
   IF cambio THEN
       WRITE (f1, r);
   ELSE
       WRITE (f2, r);
   cont:= cont +1;
```

Ejemplo 6.24 (B_2)

```
   IF cont = n THEN BEGIN
      cont:= 0;
      cambio := NOT cambio;
   END;
  END; {WHILE}
  Close(f);
  Close(f1);
  Close(f2);
END; {ParticionSecuenciasN}
```

ascension.lovillo@urjc.es