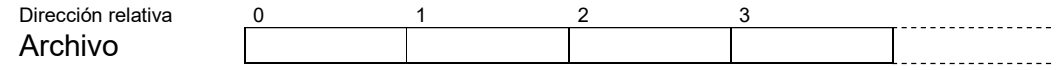
Capítulo 9

1. Archivos con tipo. Características
2. Operaciones
3. Archivos con componentes de tipo simple (real, char, etc), arreglo y registro. Ejemplos.
4. Cortes de Control por 1, 2 y 3 niveles
5. Enfrentamiento de archivos (fusión o mezcla). ABM. Fusión de N archivos
6. Ordenación de Archivos

1. Archivos con tipo. Características

El archivo de **texto**, consiste en una colección de caracteres (puede estar dividido en líneas), puede ser grabado, inspeccionado o modificado mediante el uso de un editor, también un programa Pascal puede escribir o leer del archivo. Se abre para operaciones de lectura o escritura.
El archivo **tipeado** colección de datos de un mismo tipo (simple o estructurado) : enteros, cadena, real, registro, arreglo, etc., sólo puede ser grabado y/o leído desde un programa Pascal, esto se debe a que la información que contiene no es de tipo carácter, sino la imagen de memoria del dato que se almacena (asociada a su tipo, de allí su nombre).
Se los llama tipeados o binarios y consisten en una secuencia de items denominados componentes, el lugar que ocupan respecto al origen del archivo se denomina *dirección relativa*. La primera dirección es la cero porque está en el origen.



Aunque esta descripción los asemeja al tipo arreglo unidimensional (vector), destacamos sus diferencias :

- ✓ El almacenamiento del archivo es en memoria secundaria, con lo cual permanece en memoria después de finalizada la ejecución del programa.
- ✓ El archivo no necesita establecer la cantidad máxima de componentes, dicha cantidad está sólo limitada por el espacio físico del dispositivo.
- ✓ Las componentes del archivo no se referencian mediante un índice.

Los archivos tipeados pueden ser procesados con acceso :
Secuencial, tomando componente a componente en el mismo orden en el que están almacenadas (y fueron grabadas)
Directo o aleatorio, posicionándose en la dirección relativa de una determinada componente (en cualquier lugar del archivo) y accediéndola para leer y/o grabar.

2. Operaciones

Se utilizan las mismas funciones y procedimientos con los que operan los archivos de texto, salvo los cambios que se detallan a continuación.
Recordar que a partir de la apertura del archivo un puntero o "flecha" esta posicionado en una componente del mismo (componente actual).

a. **DECLARAR** el tipo archivo y la variable de tipo archivo

```
Type
TipoArchivo = file of TipoComponente;        {de cualquier tipo, menos archivo}

Var
   ident : TipoArchivo; {es obligatorio describir el tipo, ya que no existe un tipo estándar}
```

b. **ABRIR** archivo

RESET(ident)
{ el archivo debe existir, se posiciona al comienzo para leer o escribir sobre el archivo}

c. **LEER**

READ (ident , variable)
{lee la componente actual sobre la variable, avanza a la próxima componente del archivo}

d. **ESCRIBIR**

WRITE(ident, variable) *{graba la variable en la posición actual del archivo y avanza a la siguiente }*

e. **BORRAR** un archivo que debe estar cerrado

ERASE(ident);

f. **RENOMBRAR** el archivo con un nuevo nombre, que no debe existir y debe estar cerrado.

RENAME(ident, nombre)

g. **Cantidad de componentes** del archivo (función)

FILESIZE (ident) *{devuelve la cantidad de componentes del archivo}*

Si la cantidad de componentes es n las direcciones relativas están en el rango 0..n-1

h. **POSICIONARSE** sobre una determinada componente (procedimiento)

SEEK(ident, direccion);
{Se posiciona en la componente del archivo cuya dirección especifica el parámetro, 0<= dirección < Filesize(ident) }

i. **POSICIÓN ACTUAL** (función)

FILEPOS (ident) *{devuelve la dirección relativa donde esta posicionado el puntero o flecha, componente actual del archivo}*

j. **TRUNCAR** un archivo (debe estar abierto) en la posición actual

TRUNCATE(ident); *{la marca de fin de archivo se graba en la posición de la componente actual, esta y todas las que le siguen se eliminan}*

Notar que:

- ✓ El tipo del archivo debe declararse en la sección Type, pues a diferencia del archivo de texto no es un tipo estandar
- ✓ No se utiliza readln ni writeln, ya que no está organizado por líneas, sino por el tipo de sus componentes.

Función IOResult (se utiliza para los archivos de texto y los tipeados)

Si un archivo se abre para lectura y no existe, se produce un error fatal, interrumpiéndose la ejecución del programa. Para evaluar esta posibilidad y en caso de error permitir que el programa siga corriendo se utiliza la función IOResult, que devuelve 0 si la operación de I/O se realizó con éxito, en caso contrario devuelve un código de error.
Para evitar la interrupción del programa en caso de error, se desactiva el control de I/O con una directiva al compilador.

```
.............
{$I -}      {desactiva el control de I/O}
        Reset (Arch);
        If IOResult <> 0 then
                Writeln('no se pudo abrir el archivo')
        Else
{$I +}  .......
                .......
```

Parámetros de tipo archivo
Los archivos de texto y los tipeados son siempre parámetros variable, independientemente si el subprograma lee o escribe sobre los mismos. La variable de tipo archivo contiene la dirección de la próxima componente a ser leída o grabada y se modifica en cada operación de E/S.

3. Archivos con componentes de tipo simple (real, char, etc), arreglo y registro. Ejemplos.

```
Type

TR = record
        C1 :real;
        C2 : string[5];
End;
TV= array [1..10] of word;
Tarch1 = file of word;          {las componentes son enteros}
Tarch2 = file of TR;            {las componentes son registros}
Tarch3 = file of TV;            {las componentes son arreglos}
```

Var

A1: Tarch1 ; A2 : Tarch2; A3 : Tarch3;

Grabar y listar un archivo de componentes enteras

```
Procedure Graba1(Var A1: Tarch1);
Var
        M: word;
Begin
Rewrite (A1);  Readln(M);
While M <> 0  do
        Begin
        Write(A1, M);  Readln(M)
        End;
Close(A1)
End;
```

```
Procedure Lista1(Var A1: Tarch1);
Var
        M: word;
Begin
Reset (A1);
While Not Eof(A1)  do
        Begin
        Read (A1, M);  Write(M :6)
        End;
Close(A1)
End;
```

```
Begin
Assign(A1,'Datos1.Dat');
Graba1(A1);  Lista1(A1);
End.
```

Grabar y listar un archivo de componentes registro

```
Procedure Graba2(Var A2: Tarch2);
Var
        R: TR;
Begin
Rewrite (A2);  Readln(R.C1);
While R.C1 <> 0  do
        Begin
        Readln(R.C2);  Write(A2, R);
Readln(R.C1)
        End;
Close(A2)
End;
```

```
Procedure Lista2(Var A2: Tarch2);
Var
        R: TR;
Begin
Reset (A2);
While Not Eof(A2)  do
        Begin
        Read (A2, R);  Write(R.C1 :6:2, R.C2)
        End;
Close(A2)
End;
```

```
Begin
Assign(A2,'Datos2.Dat');
Graba2(A2);  Lista2(A2);
End.
```

Importante : La mínima unidad de entrada salida al archivo es una componente completa, en este caso un registro. No es posible leer o escribir por campos.

Grabar y listar un archivo de componentes arreglo

```
Procedure Graba3(Var A3: Tarch3);
Var
        V: TV;  Res : char;   i: byte;
Begin
Rewrite (A3);
Write ('Ingresa un arreglo?(S/N)'); Readln(Res);
While Res = 'S'  do
        Begin
        For i := 1 to 10 do
        Readln(V[i]);
    Write(A3, V);
    Write ('Ingresa un arreglo?(S/N)'); Readln(Res);
        End;
Close(A3)
End;
```

```
Procedure Lista3(Var A3: Tarch3);
Var
      V: TV;    i : byte;
Begin
Reset (A3);
While Not Eof(A3)  do
         Begin
         Read (A3, V);
     For i := 1 to 10 do
        Write(V[i] : 6);
         End;
Close(A3)
End;
```

```
Begin
Assign(A3,'Datos3.Dat');
Graba3(A3);  Lista3(A3);
End.
```

Suponiendo que se ha ejecutado Graba1 y Graba3, generado un archivo A1 con 10 componentes enteras y otro A3 con una componente arreglo de 10 enteros respectivamente. Después de asignar y abrir ambos archivos para lectura, las órdenes para recuperar en un arreglo V de tipo TV la información que contiene cada uno de los archivos son:

For i := 1 to 10 do Read(A1, V[i]);	Read (A3, V);

La explicación está en que las componentes de A1 son enteras, mientras que las de A3 son arreglo de 10 enteros. Por lo tanto no es posible recuperar de A3 elementos individuales del arreglo, sino el arreglo completo.

3.1. Archivos con componentes de tipo registro. Definiciones previas

Campo de secuencia : campo con respecto al cual está ordenado el archivo.
Clave primaria : (key o llave) atributo que identifica unívocamente al registro (matrícula en el caso de un registro con datos de alumnos). Puede haber más de un campo clave.
Clave secundaria : atributo que identifica a un conjunto de registros, se trata de un valor (o un conjunto de valores) que se repite (carrera, en archivo con datos de alumnos)

Puede suceder que un atributo sea clave primaria en un archivo y secundaria en otro. Por ejemplo en un archivo que almacena datos de autos (Marca, Patente, Modelo, Precio) de una agencia, tendrá Patente como clave primaria, pero esa clave será secundaria en otro archivo que almacena infracciones (Patente, fecha, tipo de infracción).

E1. Proceso secuencial de un archivo y acceso aleatorio a otro.

Se tiene un archivo Tempe con las marcas térmicas de distintas zonas, con el siguiente diseño de registro:

- Código de Zona (1..50)
- Temperatura Máxima
- Temperatura_Mínima

En otro archivo A Zona se grabaron los nombres de las 50 zonas, y se accede con el código de zona. Se pide recorrer el archivo Tempe y listar *nombre y diferencia* térmica de aquellas zonas con diferencia mayor a 10

```
Program LISTADO;
Type
  Cad20 = string[20];
  TR = record
       Cod : word;
       TMax, TMin : real;
  End;
  TarTtemp = file of TR;    TarZona = file of Cad20;

Procedure Listado (var Temp: TarTemp ; var Zona : TarZona);
  VAR
      R : TR;    Nom : Cad20;
  Begin
```

```
  Reset (Temp); Reset( Zona);
  Writeln('ZONA   DIFERENCIA');
  While not Eof(Temp) do
    Begin
    Read(Temp, R);
    If  R.TMax - R.TMin >10 then
      Begin
      Seek(Zona, R.Cod -1); read(Zona, Nom);
      Writeln( Nom:30 , R.TMax - R.TMin : 8:2);
      End;
    End;
    Close(Temp); Close(Zona);
 End;
```

```
Var
   ATemp: TarTemp;        AZona : TarZona;
Begin{P.P.}
Assign(ATemp,'Tempe.dat');Assign(AZona, 'Zona.dat');
Listado(ATemp, AZona);
End.
```

E2. Consulta individual de registros

Programa que a partir de la matrícula de un alumno muestra sus datos, estos son:

- Legajo (cadena de 5)
- Nombre y Apellido
- Cantidad de materias
- Código y nota de cada una de las materias aprobadas

Como acceder al registro del alumno a partir del legajo?, ya que se requiere la dirección relativa asociada a dicha matricula.
Solución: se arma una tabla con las matrículas de los alumnos, cada vez que ingresa una matrícula se busca en la tabla y el índice es la posición relativa en el registro que permite posicionarse y leerlo.

```
Program ConsultaDirecta;
Type
  St20 = string[20];
  St5=string[5];
  TRM=Record
      CodMat: St5;
      Nota : byte;
  end;

  TV=array[1..30] of TRM        {arreglo de materias}
```

```
  TR=Record
     Legajo: St5; {el legajo ira a una tabla para obtener la dirección del registro en el archivo}
     NyA:St20;
     CantMat :  Byte;
     VecMat :TV;
  End;
  TArch=File of TR;
 TTabla=array[0..1000] of  St5;  {tabla para almacenar legajos}

Procedure CargaTabla(Var A:TArch; Var Tabla:TTabla);
Var
  i:byte;  R : TR;
Begin
Reset(A);  i:=0;
While not eof(A) do
    Begin
     Read(A, R);   Tabla[i]:= R.Legajo;    i:= i+1;
     End;
Close(A);
End    {procedure }
Function Busca(Tabla:TTabla; Legajo:Cad5):word; {supone que siempre se encuentra}
Var
   i,: byte;
Begin
i:=0;
While  Legajo <> Tabla[i]  do
     i:=i+1;
Busca:=i;
End;  {function}

Procedure Consulta(Var A:TArch; Tabla:TTabla);
Var
   i,: byte;    R :TR;   Legajo:St5;
Begin
Reset(A);
Writeln('Ingrese Nro.de Legajo a Consultar; xxxxx fin de consulta'); Readln(Legajo);
While Legajo <>'xxxxx' do
    Begin
    Seek(A, Busca(Tabla, Legajo));   Read(A,R);
    With R do
        Begin
         Writeln('Nombre, Apellido', NyA); Writeln('Cantidad de materias', CantMat);
         Writeln('Codigo de materia    Nota');
         For i := 1 to CantMat do
                Writeln(VecMat[i].CodMat :10,  VecMat[i].Nota: 5);
         End; {With}
```

```
    Writeln('Ingrese Nro de Legajo a Consultar, xxxxx=fin');  Readln(Legajo);
    End; {while}
Close(A);
End; {Procedure}
Begin {P.P.}
Var
    A:TArch;
    Tabla:TTabla;
Begin
Assign(A,'Alumnos.dat');
CargaTabla(A, Tabla)
Consulta(A, Tabla);
End.
```

4. Cortes de Control

Son programas que generan listados a partir de archivos ordenados y cuyo campo de secuencia es clave secundaria. Esto implica que la información se agrupa en "bloques" pertenecientes a un mismo valor de clave y el proceso consiste controlar esos bloques, listando la información obtenida a ese nivel (totales, promedios, máximos).
Si el control se realiza sobre un solo campo de secuencia, se llama corte de 1er. Nivel, si se controlan dos campos (ej: materia y dentro de cada una de ellas por matrícula), corte de 2do. Nivel y así sucesivamente. La información que se obtiene del archivo es en los distintos niveles que se realiza el control

4.1. Corte de 1er. Nivel

E3. Se tiene un archivo MULTAS, con los **importes** por infracciones de tránsito registrados en diferentes **zonas** por distintos vehículos (**patentes**), el diseño de registro es:

#ZONA (cadena de 4) {campo de secuencia, clave secundaria}
#PATENTE
#IMPORTE

Se desea calcular por zona, cantidad de infracciones e importe total de las mismas.

ARCHIVO MULTAS

ZONA	PATENTE	IMPORTE
A34	CAN180	80
A34	CAN180	100
A34	CAN180	90
A34	DEX901	50
A34	DEX901	40
C78	CAN180	30
C78	CAN180	100
C78	RUN100	80
C78	RUN100	100
C78	RUN100	60
ZZZ	------	

➔

LISTADO

ZONA	***CANT.INFRACCIONES***	***IMPORTE TOTAL***
A34	5	360
C78	5	370
TOTAL	*10*	

El bloque correspondiente a la ZONA **A34**, genera una línea de listado con Cantidad de Infracciones 5 y Total 360 y acumula en el Total la cantidad de infracciones.

El proceso consiste en leer y operar los registros de cada bloque, para ello se debe verificar que el campo de secuencia se mantenga con el mismo valor (A34), para lo cual se utiliza una variable de control, si el registro pertenece al bloque se procesa contando la infracción y sumando el importe (según cada caso realizando el proceso pertinente).

Cuando el campo de secuencia no coincide con el de la variable de control, indica que finaliza el bloque y se produce una ruptura o ***corte,*** desencadenando una serie de acciones finales de acuerdo a la índole del problema y asociadas a dicho bloque.

Los archivos deben tener grabado un registro "centinela", con un valor ('ZZZ') mayor al del campo de secuencia del último bloque, la función de este registro (la información que contiene no se procesa) es producir la ruptura o corte del último bloque.

Cada vez que comienza el proceso de un nuevo bloque se inicializan variables (entre ellas la que controla el campo de secuencia).

Se describe, en forma general, lo dicho mediante el siguiente algoritmo:

```
Títulos/ Iniciar variables generales
Read(archivo, registro)
   While  Not  Eof(archivo) do
      Comienzo de bloque : Subtítulos/Inicializar variables de bloque
      While  mismo bloque do
            Procesar registro
            Read(archivo, registro)
      Fin bloque : Acciones finales del bloque (escribir resultados del bloque, comparar, acumular, etc)
Acciones finales del programa (Escribir resultados generales)
```

procesa un bloque

Cuando se especifica que el listado debe tener línea de detalle, debe listarse la información contenida en cada uno de los registros procesados.

```
Program Corte1;
Type
     St3 = string[3]; St6=string[6];
        TReg=record
                Zona: St3;
          Patente:  St6;
                Importe: real;
        End;
     TArch= file of TReg;
Var
        Ar: TArch;
Procedure ListadoCorte1  ( var Ar: TArch);
Var
        R: TReg;     Total, Cont: word;    SumImp: real; ZonaAct : St3;
begin
reset(Ar); read(Ar, R); Total := 0;
writeln('Zona   Cantidad Infracciones     Importe Total');
while not Eof(Ar) do  __ __ __ __ __ __ __ __ __ __ __ __ _
     begin                                                    |
     Cont :=0; SumaImp:=0;                                  {comienza el bloque de ZONA }
     ZonaAct:= R.Zona;  __ __ __ __ __ __ __ __ __ __ __ __ _|
     while  ZonaAct = R.Zona  do
                 begin __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
                 Cont := Cont + 1;                                    |
                 SumaImp := SumaImp + R.Importe;                  {procesa registros ZONA }
                 read(Ar, R);    __ __ __ __ __ __ __ __ __ __ __ __ __ _|
                 end;__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ _
     writeln(ZonaAct, Cont, SumaImp : 8:2 );                  {finaliza bloque ZONA}
     Total ;= Total + Cont;                                             |
     end;          __ __ __ __ __ __ __  __ __ __ __ __ __ __ __ __ __ _|
close(Ar);
writeln('Total =', Total);
end;

Begin
Assign(Ar, 'Multas.dat' );
ListadoCorte1 (Ar);
End.
```

4.2. Corte de de 2do. Nivel

E4. A partir del mismo archivo MULTAS del ejercicio anterior y teniendo en cuenta el siguiente orden :

#ZONA {1er. campo de secuencia, clave secundaria}
#PATENTE {2do. campo de secuencia, **clave secundaria**}
#IMPORTE

Se desea obtener un listado que informe en cada zona el total por patente y la patente con mayor importe de infracciones.

ARCHIVO MULTAS		
ZONA	PATENTE	IMPORTE
A34	CAN180	80
A34	CAN180	100
A34	CAN180	90
A34	DEX901	50
A34	DEX901	40
C78	CAN180	30
C78	CAN180	100
C78	RUN100	80
C78	RUN100	100
C78	RUN100	60
ZZZ		

➔

LISTADO

ZONA A34

PATENTE	*TOTAL*
CAN180	270
DEX901	90

Patente con mayor importe CAN180

ZONA C78

PATENTE	TOTAL
CAN180	130
RUN100	240

Patente con mayor importe RUN100

Dentro de cada "bloque zona" se forman los "bloques patente" donde se agrupan las infracciones de la misma patente en dicha zona. El control de una zona implica el control de los bloques de patente que contiene. Por lo tanto se repite la necesidad de utilizar una variable que permita verificar para cada registro si se trata de la misma patente.

Como se indica en el ejemplo, para cada campo de secuencia que se desea controlar se tratan los bloques que se forman, implementando tres etapas : inicialización, proceso y finalización. En el caso de querer controlar más de un campo de secuencia estas etapas se anidan. Dentro del proceso del 1er. campo de secuencia se realiza la inicialización, proceso y finalización del bloque determinado por el 2do. campo de secuencia.

```
Program Corte2;
Type
  St3 = string[3]; St6=string[6];
  TReg=record
     Zona: St3;
     Patente: St6;
     Importe: real;
     End;
TArch= file of TReg;
```

```
Var
    Ar:TArch;
Procedure ListadoCorte2  ( var Ar: TArch);
Var
    R: TReg;    Max,  SumImp: real;    ZonaAct :St3;
    PatenteAct,  MaxPatente: St6;

begin
reset(Ar); read(Ar, R);
while not Eof(Ar) do
    begin
    writeln('ZONA ',  R.Zona);
    writeln('ZONA      TOTAL');
    Max := 0;                                                    {comienza el bloque de ZONA}
    ZonaAct:= R.Zona;
    while  ZonaAct = R.Zona  do
        begin
        PatenteAct:= R.Patente;              {comienza bloque PATENTE}
        SumImp:=0;
        while  (ZonaAct = R.Zona)  and  (PatenteAct = R.Patente) do
                    begin
                    SumImp := SumImp + R.Importe; {procesa registro PATENTE}
                    read(Ar, R);
                    end;
        writeln(PatenteAct, SumImp )                    {procesa bloque ZONA}
        if Max < SumImp then                  {finaliza bloque PATENTE}
              begin
              Max:= SumImp;
              MaxPatente:= PatenteAct;
              end;
        end;
    writeln (' Patente con mayor importe', MaxPatente);          {finaliza bloque ZONA}
    end;
close(Ar):
end;

Begin
Assign(Ar, 'Multas.dat' );
ListadoCorte2 (Ar);
End.
```

Como se indica en el ejemplo, cada campo de secuencia (bloque) que se controla tiene una inicialización, un proceso y una finalización. En el caso de considerar más de un campo de secuencia (bloques anidados) esos procesos se anidan. Dentro del proceso del 1er. campo de secuencia se realiza la inicialización, proceso y finalización del bloque determinado por el 2do. campo de secuencia.

4.3. Corte de de 3er. Nivel Contempla el uso de un segundo archivo al que se accede directamente, un arreglo acumulador y una tabla de consulta.

E5. Se tiene un archivo IMPUEST, con datos de inmuebles, para el cobro de impuestos inmobiliarios en distintas provincias. El diseño de registro es :
#COD. de PROVINCIA (1..24) {1er. campo de secuencia, clave secundaria}
#COD. de PARTIDO (1..500) {2do. campo de secuencia, clave secundaria}
#COD. de ZONA (1..20) {3er. campo de secuencia, clave secundaria}
#COD. de INMUEBLE
#SUPERFICIE
#TIPO de INMUEBLE (1..4)

Este último campo indica 1-BALDIO ; 2-VIVIENDA ; 3-COMERCIAL ; 4-SERVICIOS (hospital, escuela..)Se cuenta además con un arreglo PRECIOS, de 4 elementos reales, donde se almacena el precio del impuesto por m^2 , según el tipo de inmueble.
Los nombres de las provincias se encuentran grabadas en un archivo PROVINCI y se accede por COD. de PROVINCIA. Se pide procesar secuencialmente el archivo y generar el siguiente listado :

LISTADO DE IMPUESTOS

Provincia de Buenos Aires

Partido 2

Zona	Importe
3	$99
5	$100
...	...
Total Partido	$2000

Partido 5

Zona	Importe
1	$29
8	$10
...	...
Total Partido	$400

TOTALES PROVINCIALES POR TIPO DE INMUEBLE

BALDIO	$999
VIVIENDA	$9999
COMERCIAL	$999
SERVICIOS	$999

Provincia de Córdoba

Partido 3

...	...

PROVINCIA CON MAS RECAUDACION : CORDOBA
RECAUDACION GENERAL (todas las provincias) $9999999

```
program CORTE3;
TYPE
  TR = record
    cod_pro: 1..24;
    cod_par: 1..500;
    cod_zon:1..20;
    sup: real;
    cod_inm: string[10];
    tipo: 1..4;
  end;

  CAD20= string[20];

  TA1= file of TR;
  TA2= file of CAD20;
  TV = array[1..4] of real;

CONST
  PRECIOS : TV = (2.3, 1.08, 5.8, 0.34);
procedure LIST_CORTE3(var A:TA1);
VAR
  vec:TV ;       nom, max: CAD20;
  R:TR;          B:  TA2;
  max, tot_gen, tot_part, tot_zon, imp, aux : real;
  i, prov_act, part_act, zon_act : word;

begin
assign(B, 'PROVINC.DAT');
reset(B);
reset(A);
read(A, R);
max:= 0;
tot_gen:= 0;
writeln('LISTADO DE IMPUESTOS');  {una sola vez, fuera del ciclo}
```

```
while not eof(A) do
    begin                       {comienza provincia}
    for i:=1 to 4 do
          vec[i] := 0;
    seek(B, R.cod_pro); read(B,nom);
    writeln('Provincia', nom);
    prov_act := R.cod_pro;
    while   (prov_act = R.cod_pro) do
        begin                       {comienza partido}
        tot_part:=0;
        writeln('Partido', R.cod_part);
        writeln('Zona      Importe');
        part_act:=R.cod_part;
        while (prov_act = R.cod_pro) and (part_act =R.cod_part) do
              begin                 {comienza  zona}
              tot_zon :=0;
              zon_act:=R.cod_zon;
              while (prov_act = R.cod_pro) and (part_act =R.cod_part) and (zon_act =R.cod_zon) do
                    begin                   { procesa registros del bloque }
                    imp:= R.sup * PRECIOS[R.tipo];
                    tot_zon:= tot_zon + imp;
                    vec[R.tipo]:= vec[R.tipo] +imp;
                    read(A, R);
                    end;
              witeln(zon_act, tot_zon);     {fin zona}
              tot_part:= tot_part + tot_zon;
             end;
         writeln('Total Partido  ', tot_part);          {fin partido}
         end;
     aux:= vec[1] +vec[2] + vec[3] + vec[4];                {fin provincia}
     tot_gen:=tot_gen + aux;
     if aux > max then
              begin
              max:= aux;
              max_prov:= nom;
              end;
    writeln ('TOTALES PROVINCIALES POR INMUEBLE');
    writeln('BALDIO', vec[1]);
    writeln('VIVIENDA', vec[2]);
    writeln('COMERCIAL', vec[3]);
    writeln('SERVICIOS', vec[4]);
    end;                { fin de archivo}
writeln('PROVINCIA CON MAS RECAUDACION: ', max_prov);
writeln('RECAUDACION GENERAL (todas las provincias) ', tot_gen);
close(A); close (B);
end;
```

```
VAR
        A:TA1;
 begin
   assign(A, 'IMPUEST.DAT');
   LIST_CORTE3(A);
 end.
```

5. Enfrentamiento de archivos (fusión o mezcla). ABM.

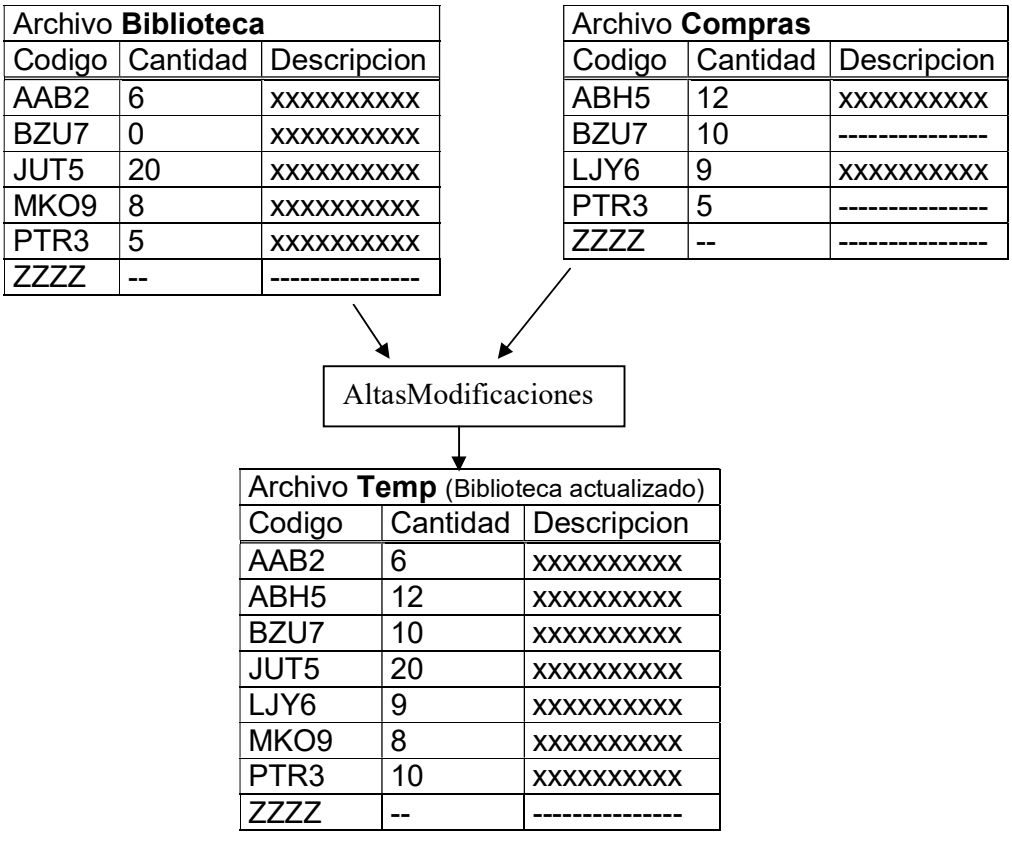
Es un proceso por el cual dos o más archivos, ordenados por el mismo campo de secuencia (clave primaria o secundaria), se enfrentan registro a registro para alguna tarea específica, obteniéndose como resultado listados y/o archivos.

Un proceso típico de esta metodología es denominado ABM y permite actualizar la información de un archivo, contempla:

- ALTAS: proceso que permite incorporar nuevos registros en un archivo
- BAJAS: proceso que permite eliminar registros de un archivo
- MODIFICACIONES : proceso que permite modificar la información almacenada en registros de un archivo (salvo la clave)

E6. Se tienen los datos de los ejemplares de una biblioteca, grabados en un archivo **Biblioteca** y en otro archivo **Compras** los datos de los títulos adquiridos (ambos del mismo tipo y orden)

#Código de Libro {campo de secuencia, clave primaria}
#Cantidad
#Descripción



Se genera un tercer archivo **Temp,** con la información actualizada de Biblioteca, que reemplazará al original.

Proceso de AltasModificaciones utiliza los archivos Biblioteca y Compras

El proceso de enfrentamiento de archivos se asemeja al de intercalación de dos arreglos ordenados, donde se compara un elemento de cada uno y el de menor valor es el procesado, incrementando el índice del arreglo del cual provino. Esto se repite hasta que uno o ambos arreglos finalicen. En el primer caso los elementos del arreglo que no termino serán procesados en su totalidad.

ARREGLOS ERA ASÍ		
V1	V2	V3
I :=1	J:=1	K:=0

```
WHILE (I<=N) AND (J< = M) DO
   BEGIN
   K:=K+1
```

Este proceso aplicado a dos archivos requiere que los mismos hayan sido ordenados con el mismo criterio (campo de secuencia). Los archivos se leen secuencialmente, la memoria almacena un registro de cada uno, el de clave menor es procesado (grabado en un archivo de salida, listando, etc) y se avanza leyendo otro registro el archivo del cual provino. Esto se repite hasta que uno o ambos archivos alcancen el final. En el primer caso los registros del archivo que no finalizo deben ser procesado en su totalidad.

Para simplificar el enfrentamiento y resolver dentro de un único ciclo el proceso de la totalidad de los registros de ambos archivos, se graba un **registro "centinela"** al final de los archivos, con un valor de clave mayor a todas. De esta forma al finalizar un archivo no se detiene el ciclo.

El archivo que finaliza primero deja en memoria su registro centinela con clave mayor a todas las pendientes de proceso en el otro archivo, dando "paso" a éstos hasta agotarlos (siempre se procesa el registro con clave menor y se avanza en el archivo asociado).

El ciclo finaliza cuando ambos archivos alcanzan el final y sus registros centinelas están en memoria.

```
While Not Eof(Biblioteca) or Not Eof(Compras)
        If…  < …then
                …….
        else
                If… > …then
                    ……
                else       {por =}
                    …….
```

```
Program ABM;

Type
St4: string[4];     St20 = string[20];
TR=Record
  Codigo: St4;
  Cantidad:Word;
  Descripcion:String;
End;
TAr = File of TR;
TArB = File of St4; { para almacenar código de libros, utilizado en el proceso de bajas}

Var
Biblioteca, Compras: TAr;
Bajas: TArB;
```

```
Procedure AltasModificaciones(Var Biblioteca,   Compras:TAr;   Nombre: St20);
{se desarrollan en la siguiente hoja}
……..
Procedure Elimina(Var Biblioteca:TAr;   Var Bajas:TArB;    Nombre : St20);
{se desarrollan en la siguiente hoja}
……..

Begin                              {P.P, se presupone que los registros centinela están grabados}
Assign(Biblioteca, 'Biblio.dat');
Assign(Compras, 'Compras.dat');
Assign(Bajas, 'Bajas.dat');
AltasModificaciones(Biblioteca, Compras, 'Biblio.Dat' );
Elimina(Biblioteca, Bajas, 'Biblio.Dat' );
End.
```

Se completa las definiciones de los procedimientos AltasModificaciones y Elimina

```
{se considera compra de nuevos títulos (altas) y reposición (modificación)}
Procedure AltasModificaciones(Var Biblioteca, Compras:TAr;     Nombre: St20);
Var
Temp:Tar;
RB,RC :TR;
Begin
 Assign(Temp,'Temp.Dat');  Rewrite(Temp);
 Reset(Biblioteca);
 read(Biblioteca, RB); read(Compras, RC);
 While not Eof(Biblioteca) or not Eof(Compras) do
      If RB.Codigo < RC.Codigo Then
                Begin
                 write(Temp, RB); read(Biblioteca, RB)
                 End
    Else
                 If  RB.Codigo > RC.Codigo Then   {alta}
                        Begin
                        write(Temp, RC); read(Compras, RC);
                        End
                 Else{ reposición, los códigos son iguales }
                          Begin
                          RB.Cantidad:= RB.Cantidad + RC.Cantidad
                          write(Temp,RB);
                          read(Biblioteca, RB);  read(Compras,RC);
                          End; { fin del if y fin del while }
write(Temp,RB); { para grabar el centinela en el archivo actualizado}
Close(Biblioteca); Close(Compras); Close(Temp);
Erase(Biblioteca);  {elimina el archivo original}
Rename(Temp, Nombre);  {renombra el temporáneo con el nombre del original, que se ha eliminado  previamente}
End;{Procedimiento}
```

El archivo **Bajas** , almacena los códigos de libro que se desean eliminar del archivo biblioteca (sus componentes son de tipo cadena, clave primaria ordenadas como Biblioteca)

Archivo Biblioteca		
Codigo	Cantidad	Descripcion
AAB2	6	xxxxxxxxxx
BZU7	0	xxxxxxxxxx
JUT5	20	xxxxxxxxxx
MKO9	8	xxxxxxxxxx
PTR3	5	xxxxxxxxxx
ZZZZ	--	----------

Archivo Bajas
Codigo
BZU7
MKO9
ZZZZ

Elimina

Archivo Temp. (Biblioteca actualizado)		
Codigo	Cantidad	Descripcion
AAB2	6	xxxxxxxxxx
JUT5	20	xxxxxxxxxx
PTR3	5	xxxxxxxxxx
ZZZZ	--	----------

```
{se eliminan registros }
Procedure Elimina(Var Biblioteca:TAr;Var Bajas:TArB; Nombre : St20);
Var
Temp:TAr;
RB:TR;    R:St4;
Begin
 Assign(Temp,'Temp.Dat');
 Reset(Biblioteca);
 Reset(Bajas);
 Rewrite(Temp);
 read(Biblioteca, RB);
 read(Bajas, R);
 While Not Eof(Biblioteca) or Not Eot(Bajas) do
    If RB.Codigo  < R Then
        Begin
        write(Temp,RB);
       read(Biblioteca, RB);
        end
    Else
        If RB.Codigo > R Then
             read(Bajas,R) { R Baja incorrecta}
        Else                                        {son iguales los códigos, no grabo en TEMP}
            Begin
            Read(Biblioteca, RB);
           Read(Bajas, R);
            End;              {fin del while}
```

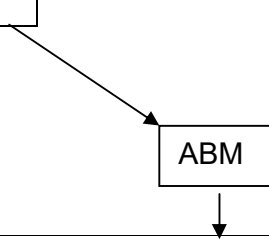
```
  write(Temp,RB); { para grabar el centinela en el archivo actualizado}
  Close(Biblioteca);
  Close(Bajas);
  Close(Temp);
  Erase((Biblioteca);
  Rename(Temp,Nombre);
End;{Procedure}
```

E7. Proceso simultaneo de altas, bajas y modificaciones mediante enfrentamiento de archivos

A partir del archivo Maestro y el de Novedades generar el maestro actualizado

Archivo Maestro	
200	XXX
231	YYY
315	AAA
386	TTT
391	BBB
402	MMM
999	------

Archivo Novedades		
200	M	EEE
230	A	HHH
315	B	------
386	A	ZZZ
390	B	------
999	--	------



Maestro Actualizado	
200	EEE
230	HHH
231	YYY
386	TTT
391	BBB
402	MMM

Procedimiento ABM, enfrenta dos archivos: Maestro y Novedades, generando un tercer archivo actualizado, donde se registran las altas, bajas y modificaciones.

```
Program ActualizaABM;
Type
   TR=record
      Clave : TClave;
      Dato: Tdato;
   end;
TRN = record
   Clave : TClave;
   Cod : char;  {A - B - M}
   Dato: Tdato;
end;
```

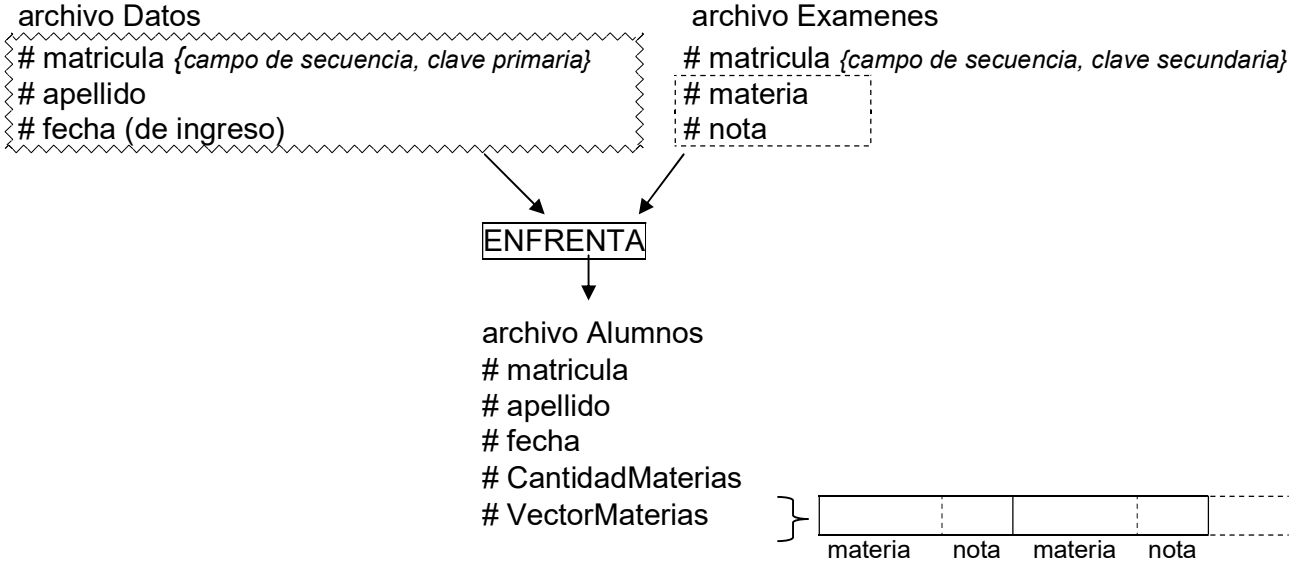
```
TArch = file of TR;
TArchN = file of TRN;

Procedure ABM( Var Ar, ArAct : TArch; Var ArNov, Err : TArchN);
Var
  R, Raux : TR;
  RN : TRN;
Begin
   Reset (Ar);  Reset (ArNov);  Rewrite (ArAct);  Rewrite (Err);
   Read(Ar, R);   Read (ArNov, RN);
   While Not Eof(Ar)  or  Not Eof (ArNov)  do
        If R.Clave = RN.Clave then
                Begin
                If RN.Cod = 'A' then
                        Begin
                        Write(Err, RN);
                        Write(ArAct, R);
                        end
                else
                        if RN.Cod = 'M' then
                                Begin
                                R.Dato := RN.Dato;
                                Write (ArAct, R);
                                end;   {la baja no tiene sentencias}
                Read(Ar, R);   Read (ArNov, RN);
                end
         else
                If R.Clave  > RN.Clave then
                        Begin
                        If RN.Cod = 'A' then
                                Begin
                                RAux.Clave:=RN.Clave;
                                RAux.Dato:=RN.Dato;
                                Write(ArAct, RAux);
                                end
                        else
                                Write(Err, RN);  {baja o modificación}
                        Read (ArNov, RN )
                        end
                else            { R.Clave  < RN.Clave }
                        Begin
                        Write (ArAct, R);
                        Read (Ar, R)
                        End;
   Close( Ar );  Close( ArNov);  Close (ArAct);  Close (Err);
end;
```

```
Var
   Ar, ArAct : TArch;
   ArNov, Err : TArchN

  Begin
  Assign( Ar, 'Ar.Dat'); Assign(ArAct, 'ArAct.Dat');
  Assign(ArNov, 'ArNov.Dat'); Assign(Err, 'Err.Dat');
  ABM(  Ar, ArAct, ArNov, Err);
  End.
```

E8. Dados dos archivos, el primero **Datos** contiene información sobre alumnos y el segundo **Examenes** contiene información sobre las materias aprobadas por los alumnos (hay más de un registro por alumno).
Se pide enfrentarlos y obtener un tercer archivo **Alumnos** que agrupe la información total de cada alumno.




```
Program Alumnos;
Type
St20=String[20];   St8=String[8];   St5=string[5];

TRMareria=Record              {Registros para materias}
   CodigoMateria :St5;
   Nota:Byte;
end;

TRExamen=Record                 {Registro para exámenes aprobados}
  Matricula:St5;                {campo de secuencia, clave secundaria }
  E:TRMateria;
End;

TRDatos=Record                          {Registro para datos de alumno}
  Matricula : St5;                      {campo de secuencia, clave primaria }
  ApelyNom:St20;
  Fecha:St8;
End;
```

```
TV=array[1..30] of TRMateria;
TRAlumno=Record                          {Alumnos}
  Datos:TRDatos;
  CantidadMaterias:Byte;
  VectorMaterias:TV;
End;

TArDatos = file of TRDatos;                    {Tipos Archivos}
TArExamen = file of TRExamenes;
TArAlumnos = file of TRAlumnos;

Procedure Enfrenta(Var Datos:TArDatos; Var Examenes:TArExamenes; Var Alumnos:TArAlumnos);
Var
   i:Byte {para el VectorMaterias}
   RD:TRDatos;
   RE:TRExamenes;
   RA:TRAlumnos;
Begin
   Reset(Datos); Reset(Examenes);          { se supone que están grabados los centinelas}
   Rewrite(Alumnos);
   Read(Datos, RD);Read(Examenes, RE);
   While Not Eof(Datos) or Not Eof(Examenes) do
         If RD.Matricula = RE.Matricula Then   {Alumnos con materias rendidas}
              Begin
i:= 0 ;
              While RD.Matricula = RE.Matricula do
                   Begin
                   i:=i+1 ;
                   RA.VectorMaterias[i]:=RE.E;
                   Read(Examenes, RE);
                   End; {Cuando termina este ciclo el alumno RD.Matricula no tiene más materias}
                   RA.CantidadMaterias:= i ;
                   RA.Datos:=RD; {Termina de cargar el registro RA, RD está en memoria}
                   Write(Alumnos,RA); Read(Datos,RD);
                   End
         Else {Alumnos que no rinden examenes}
                  If RD.Matricula<RE.Matricula Then
                       Begin
                       RA.Datos:=RD;
                       RA.CantidadMaterias:=0;
                       Write(Alumnos,RA);Read(Datos,RD);
                       End
                  Else {Examen que no pertenece a ningun alumno}
                       Read(Examenes,RE); { fin del if y fin del while}
   Close(Datos);Close(Examenes);Close(Alumnos);
End;

Var
Datos:TArDatos;
Examenes:TArExamenes;
Alumnos:TArAlumnos;
```

```
Begin{P.P.}
  Assign(Datos, 'Datos.Dat');
  Assign(Examenes,'Examenes.Dat');
  Assign(Alumnos,'Alumnos.Dat');
  Enfrenta(Datos,Examenes,Alumnos);
End.
```

E9. Fusión de una cantidad variable de archivos

El siguiente programa enfrenta N archivos del mismo tipo, con campo de secuencia c1, y genera un nuevo archivo producto de la unión ordenada de los mismos. (intercalación de registros)

```
program FusionMultiple;
type
tr=record
  c1,c2:word;  {c1 campo de secuencia}
end;
tf=file of tr;
tvf=array[1..5] of tf;
tvr=array[1..5] of tr;

procedure asigna( var v:tvf; var n:byte);
var
   nom  :string[5];  i:byte;
begin
   readln(n);
   for i:=1 to n do
     begin
     readln(nom);
     assign(v[i],nom);
     end;
end;

procedure intercal_n(var v:tvf; var salida:tf;n:byte);
var
   r:tvr; min,i:byte;
begin
   rewrite(salida);
   for i:=1 to n do
          begin
          reset(v[i]);       read(v[i],r[i]);
          end;
   while nofin(v,n) do
          begin
          min:=minimo(r,n);
          write(salida,r[min]);
          read(v[min],r[min]);
          end;
 for i:=1 to n do
          close(v[i]);
close(salida);
end;
```

```
function nofin(v:tvf; n:byte):boolean;
var                          {devuelve TRUE si algún archivo no finalizó }
   i:byte;
begin
    i:= 1;
    while  (i < n)  and  eof ( v [i] ) do
        i := i + 1;
    nofin:= not eof ( v[i]);
end;

function minimo(r:tvr;n:byte):byte;
var
   i,pos:byte;
begin
    pos:=1;
    for i:=2 to n do
      if r[pos].c1> r[i].c1 then
         pos:=i;
     minimo:=pos;
end;

var
   v:tvf; salida:tf;n:byte;r:tr;
begin
   assign(salida,'salida.dat');
   writeln('ingese cantidad y nombres');
   asigna(v,n);   {los archivos tienen centinela}
   intercal_n(v,salida,n);
   reset(salida);
   while not eof(salida) do
      begin
      read(salida,r);
      writeln(r.c1:5, r.c2:5);
      end;
   close(salida);
end.
```

6.Ordenación de Archivos (mediante partición y fusión)

En el caso que el tamaño del archivo no permita su ordenación interna (leerlo sobre un arreglo y ordenarlo en memoria, grabándolo finalmente sobre el archivo original) se realiza la clasificación externa que requiere mayor tiempo de ejecución (operaciones de lectura/escritura)
También puede realizarse una combinación de clasificación externa e interna, aprovechando al máximo el espacio de memoria.
A continuación se describen dos métodos de clasificación externa que utilizan archivos auxiliares.

6.1.Clasificación por mezcla directa

El método consiste en realizar sobre los datos del archivo una sucesión de particiones y fusiones que producen secuencias ordenadas de longitud creciente.
La primera partición genera secuencias de longitud 1, que fusionadas sobre el archivo original produce secuencias ordenadas de longitud 2.
En la siguiente partición se duplica la longitud de la secuencia, en cada partición-fusión la duplicación de la longitud conlleva a que la longitud exceda la longitud original, determinando de esta forma el final del proceso de ordenación.

Ejemplo:
Sea F un archivo cuyos campos de secuencia figuran a continuación (no se incluyen los campos restantes). Se utilizan dos archivos auxiliares F1 y F2, con la misma estructura de F para realizar las particiones y fusiones.

F : 15, 18, 7, 75, 14, 13, 43, 40, 51, 93, 75, 26, 64, 27, 13

Partición de secuencias de longitud 1
F1 : 15, 7, 14, 43, 51, 75, 64, 13
F2: : 18, 75, 13, 40, 93, 26, 27

Fusión de secuencias de longitud 1
F : 15, 18, 7, 75, 13, 14, 40, 43, 51, 93, 26, 75, 27, 64, 13

Partición de secuencias de longitud 2
F1 : 15, 18, 13, 14, 51, 93, 27, 64
F2: : 7, 75, 40, 43, 26, 75, 13

Fusión de secuencias de longitud 2
F : 7, 15, 18, 75, 13,14, 40, 43, 26, 51, 75, 93, 13, 27, 64

Partición de secuencias de longitud 4
F1 : 7, 15, 18, 75, 26, 51, 75, 93
F2 : 13, 14, 40, 43, 13, 27, 64

Fusión de secuencias de longitud 4
F : 7, 13, 14, 15, 18, 40, 43, 75, 13, 26, 27, 51, 64, 75, 93

Partición de secuencias de longitud 8
F1 : 7, 13, 14, 15, 18, 40, 43, 75
F2: *13, 26, 27, 51, 64, 75, 93*

Fusión de secuencias de longitud 8
F : 7,13, 13, 14, 15, 18, 26, 27, 40, 43, 51, 64, 75, 75, 93

El proceso termina al detectarse que la longitud (16) de la próxima secuencia (para la partición) es mayor o igual que la cantidad de registros del archivo(15).

6.2.Clasificación por mezcla equilibrada

Es una optimización del método anterior que consiste en realizar la partición tomando secuencias ordenadas de máxima longitud posible y realizando la fusión de dichas secuencias alternativamente sobre dos archivos. O sea que el proceso de partición fusión es simultaneo.
Se utilizan tres archivos auxiliares junto al original, siendo dos de entrada y dos de salida en forma alternativa para la realización del proceso de partición-fusión simultanea.
El proceso termina cuando la partición-fusión deriva todos los registros a uno de los archivos de salida, quedando el otro vacío.

Ejemplo:
Sea F el archivo a ordenar y F1, F2 y F3 los archivos auxiliares con la misma estructura de F.
Al comienzo se realiza por única vez la partición del archivo original en dos archivos.

F : 15, 18, 7, 75, 14, 13, 43, 40, 51, 93, 75, 26, 64, 27, 13
F1:

Partición inicial
F2 : 15, 18, 14, 40, 51, 93, 26, 64, 13
F3 : 7, 75, 13, 43, 75, 27

Primera fusión-partición
F : 7, 15, 18, 75, 26, 27, 64
F1 : 13, 14, 40, 43, 51, 75, 93, 13

Segunda fusión-partición
F2 : 7, 13, 14, 15, 18, 40, 43, 51, 75, 75, 93
F3 : 13, 26, 27, 64

Tercera fusión-partición
F : 7,13, 13, 14, 15, 18, 26, 27, 40, 43, 51, 64, 75, 75, 93
F1 :