CONCEPTOS DE BASES DE DATOS

CLASE 2

Archivos

Algorítmica clásica

- Operaciones usuales a resolver
  - **Modificar** el contenido actual de un archivo
  - **Agregar** nuevos elementos a un archivo
  - **Actualizar** un archivo maestro con uno o varios archivos detalles
  - **Corte de control**
  - **Merge** de archivos

Archivos

Algorítmica clásica

- Algunas consideraciones generales
  - Las declaraciones de tipos y relaciones de los archivos con el SO (ASSIGN) se encuentran en el programa principal
  - No intentar hacer lecturas en un archivo sin antes chequear que hay registros disponibles (EOF)
  - Cerrar archivos (CLOSE) al finalizar su uso en los algoritmos
  - Para incrementar la eficiencia de los algoritmos, se debe minimizar los accesos a los archivos maestro:
    - Si es posible, no recorrerlos más de una vez
    - Si es posible, no recorrerlos de forma completa

Archivos

Modificación

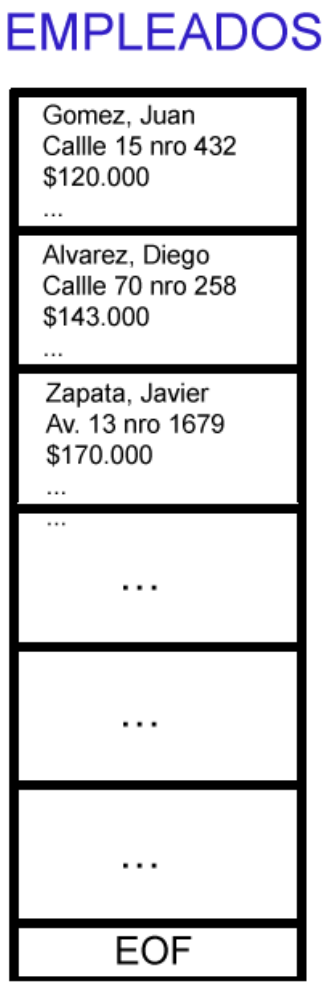
- Ej: actualización de salarios en un archivo de empleados
  - Este caso involucra un archivo de datos previamente generado y consiste en actualizar todos sus elementos
  - El archivo **no se encuentra ordenado** por ningún criterio.
  - El archivo debe ser recorrido de forma completa siguiendo un procesamiento secuencial

Archivos

Modificación

- Se debe procesar registro por registro, del primero al último

- Para la modificación de un registro:
  - Se lee el registro
  - Se realiza la modificación de datos
  - Se vuelve a la posición del registro
  - Se escribe el registro actualizado



Archivos

Modificación

{declaración de tipos de datos en el programa principal, que contiene y utiliza el proceso "actualizar"}

```
Program mainProgram;
type
    registro = record
            nombre: string[20];
            direccion: string[20];
            salario: real;
            (...)
    end;
   empleados = file of registro;
var
    archivoEmp: empleados;
begin
    assign(archivoEmp, "empleados.dat");
     actualizar (archivoEmp, 1.2);
    (...)
```

{se recibe el archivo como parámetro por referencia,

se recibe un valor porcentual de actualización (1.1 → 10%) }

```
Procedure actualizar (var fileEmp:empleados, valor:real);
var
    regEmp: registro;
begin
    reset(fileEmp);
    while not eof(fileEmp) do
    begin
      read(fileEmp, regEmp);
      regEmp.salario := regEmp.salario * valor;
      seek(fileEmp, filepos(Emp) -1 );
      write(fileEmp, regEmp);
    end;
  close(fileEmp);
end;
```

Archivos

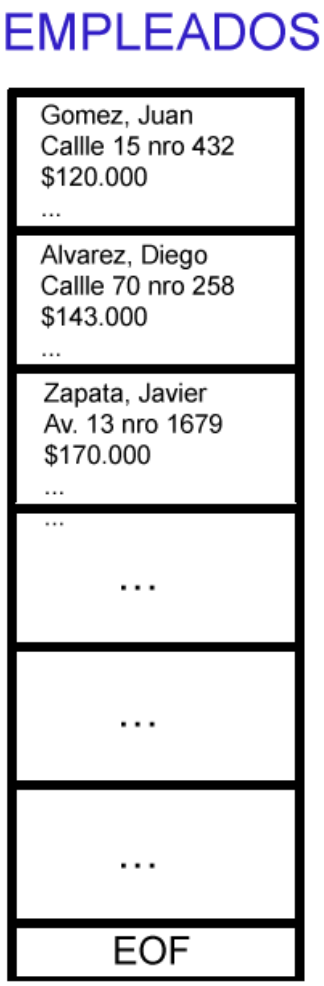
Agregar datos

- Ej: agregar nuevos empleados a un archivo
  - Se procesa un solo archivo de datos
  - El archivo ya contiene información cargada
  - Se le incorporan datos nuevos

Archivos

Agregar datos

- Se debe posicionar el puntero sobre la marca de EOF antes de comenzar a insertar nuevos registros
- Para la inserción de un nuevo registro:
  - Se leen los datos completos desde teclado
  - Se realiza la escritura
- Al finalizar las inserciones, se cierra el archivo



Archivos

Agregar datos

- Ej: agregar nuevos empleados a un archivo

```
Procedure agregar(var fileEmp: empleados)
var
    regEmp: registro;
begin
    reset(fileEmp);
    seek(fileEmp, filesize(fileEmp));
    leer_datos(regEmp);
    while (regEmp.nombre <> ' ') do
    begin
      write(fileEmp, regEmp);
      leer_datos(regEmp);
    end;
    close(fileEmp);
end;
```

Archivos

Agregar datos

- Ej: agregar nuevos empleados a un archivo

```
Procedure leer_datos(var regEmp: registro)
var
    regEmp: registro;
begin
    write(“Ingrese el nombre del empleado:”);
    read(regEmp.nombre);
    if (regEmp.nombre <> ‘ ’)
    begin
       write(“Ingrese su dirección:”);
       read(regEmp.nombre);
       (…)
    end;
end;
```

- El procedimiento **leer_datos()** realiza la lectura desde teclado de un registro de empleado en forma completa, campo por campo.
- Cuando **leer_datos()** devuelve un nombre de empleado vacío, significa que el usuario no desea agregar más empleados.

Archivos

Actualización maestro-detalle

- Este problema involucra utilizar al mismo tiempo varios archivos de datos, de distinto tipo.
  - Se denomina maestro al archivo que resume información sobre un dominio específico. Generalmente estos archivos almacenan datos de:
    - Información sobre algún aspecto importante de las actividades de una organización, como por ejemplo: **CLIENTES**, **PRODUCTOS**, **VENDEDORES**, **PROVEEDORES**, etc.
    - Información que refleja la historia de eventos de la organización, como por ejemplo: **PEDIDOS**, **VENTAS**, etc.

Archivos

Actualización maestro-detalle

- Se denomina detalle al archivo que contiene nueva información generada por las diferentes aplicaciones.
- Cada aplicación crea y maneja sus propios archivos detalle y los utiliza para la operatoria diaria.
- Al finalizar la jornada, se usan para actualizar la información de los archivos maestro.
- En general, las actualizaciones involucran:
  - **1** archivo maestro
  - **N** archivos detalle

Archivos

Un maestro - Un detalle

- Ej 1: actualización de horas trabajadas
  - Si los archivos involucrados estuvieran desordenados, este proceso involucraría recorrer el archivo maestro en más de una ocasión
  - **Precondiciones**
    - Ambos archivos (maestro y detalle) están ordenados por el mismo criterio, en este caso el nombre del empleado
    - En el archivo detalle solo aparecen empleados que existen en el archivo maestro
    - Cada empleado del archivo maestro a lo sumo puede aparecer una vez en el archivo detalle

Archivos

Un maestro - Un detalle

- Se debe recorrer el archivo maestro buscando el primero de los registros del archivo detalle
- Una vez encontrado, se realiza la actualización del registro:
  - Se modifican los datos
  - Se vuelve a la posición del registro
  - Se escribe el registro actualizado
- Se continua con el siguiente registro del archivo detalle
- Cuando no hay más registros en el archivo detalle, se finaliza

Archivos

Un maestro - Un detalle

```
Program actualizar;
type
    empleado = record
            nombre: string[30];
            direccion: string[30];
            cht: integer; {cantidad de horas trabajadas}
    end;
    empDia = record
            nombre: string[30];
            cht: integer;
    end;
    maestro = file of empleado; {archivo que contiene la info completa}
    detalle = file of empDia;  {archivo que contiene la info diaria}
var
    mae1: maestro;
    det1: detalle;
    regm: empleado;
    regd: empDia;
```

Archivos

Un maestro - Un detalle

```
begin
    assign (mae1, 'maestro.dat'); {se asocian y se abren los archivos}
    assign (det1, 'detalle.dat');
    reset (mae1);
    reset (det1);
    while (not eof(det1)) do   {mientras haya elementos en el detalle}
    begin
      read(mae1, regm);
      read(det1,regd);
      while (regm.nombre<> regd.nombre) do   {se busca el empleado en el maestro}
            read (mae1,regm);
      regm.cht := regm.cht + regd.cht;   {al encontrarlo se actualiza}
      seek (mae1, filepos(mae1)-1);
      write(mae1,regm);
    end;
    close(det1); {se cierran los archivos}
    close(mae1);
end.
```

Archivos

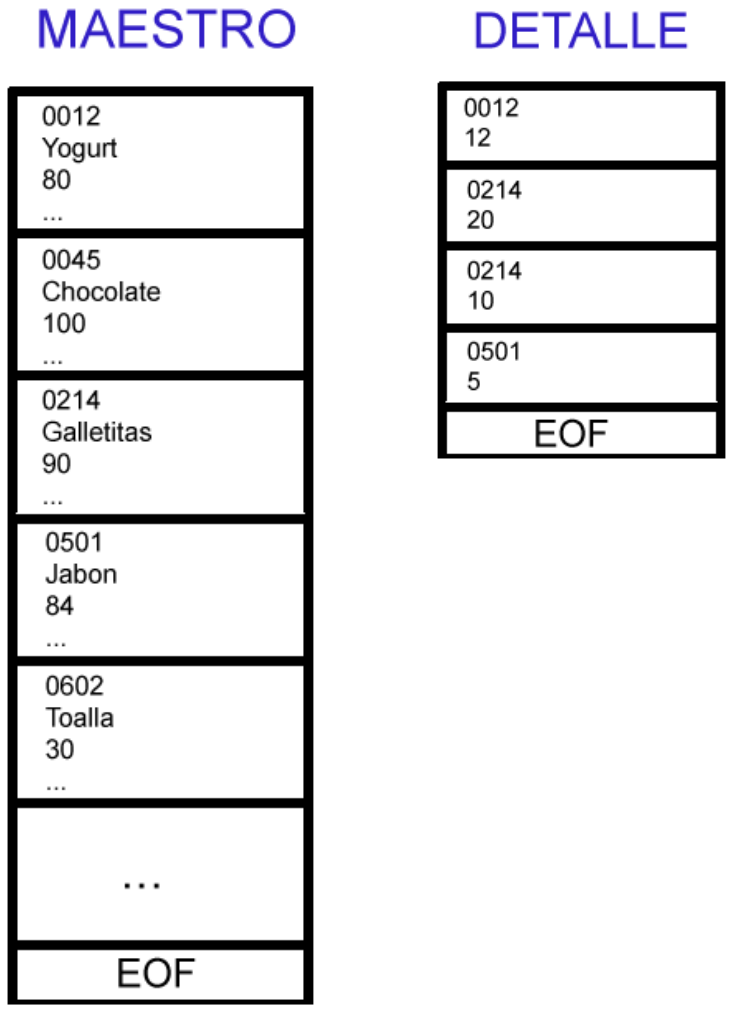
Un maestro - Un detalle

- Ej 2: actualización de stock de productos
  - Si los archivos involucrados estuvieran desordenados, este proceso involucraría recorrer el archivo maestro en más de una ocasión
  - **Precondiciones**
    - El archivo maestro no está vacío
    - Ambos archivos están ordenados por código de producto
    - En el archivo detalle solo aparecen productos que existen en el archivo maestro
    - Cada producto puede ser, a lo largo del día, vendido más de una vez → en el archivo detalle pueden existir varios registros correspondientes al mismo producto

Archivos

Un maestro - Un detalle

- Se procesan todos los registros del archivo detalle con igual código.
- Se encuentra el registro en el archivo maestro y se realiza la actualización:
  - Se modifican los datos
  - Se vuelve a la posición del registro
  - Se escribe el registro actualizado
- Se continua con el siguiente registro del archivo detalle
- Cuando no hay más registros en el archivo detalle, se finaliza



Archivos

Un maestro - Un detalle

```
program actualizar;
const
    valoralto='9999';
type
    str4 = string[4];
    prod = record
            cod: str4;
            descripcion: string[30];
            cant: integer;
            pu: real;
            (…)
    end;
    v_prod = record
            cod: str4;
            cant_vendida: integer;
    end;
    maestro = file of prod;
    detalle = file of v_prod;
```

```
var
    mae1: maestro;
    det1: detalle;

    regm: prod;
    regd: v_prod;

    total: integer;
    aux: str4;
```

Archivos

Un maestro - Un detalle

```
procedure leer (var archivo:detalle; var dato:v_prod);
begin
    if (not eof(archivo))
      then read (archivo,dato)
      else dato.cod:= valoralto;
end;

begin   {programa principal}
    assign (mae1, 'maestro.dat'); {se asocian y se abren los archivos}
    assign (det1, 'detalle.dat');
    reset (mae1);
    reset (det1);
    read(mae1,regm); {siempre se debe preguntar por EOF antes de leer → en
    este caso el enunciado del ejemplo dice que no está vacío}
    leer(det1,regd);
```

Archivos

Un maestro - Un detalle

```
while (regd.cod <> valoralto) do   {se procesan todos los registros del archivo det1}
begin
  aux := regd.cod;
  total := 0;
  while (aux = regd.cod ) do   {se procesan códigos iguales del detalle}
  begin
        total := total + regd.cant_vendida;
        leer(det1,regd);
   end;
  while (regm.cod <> aux) do   {se busca el registro detalle en el maestro}
        read (mae1,regm);
  regm.cant := regm.cant - total;   {se modifica el stock del producto}
  seek (mae1, filepos(mae1)-1);   {se reubica el puntero y actualiza}
  write(mae1,regm);
end;
close(det1);   {se cierra los archivos}
close(mae1);
```

Archivos

Un maestro - Un detalle

```
{se visualiza en pantalla el stock de cada producto}
reset(mae1);

while (not eof(mae1)) do
begin
  read (mae1,regm);
  writeln(regm.cod, regm.cant);
end;

close(mae1);

end.
```

Archivos

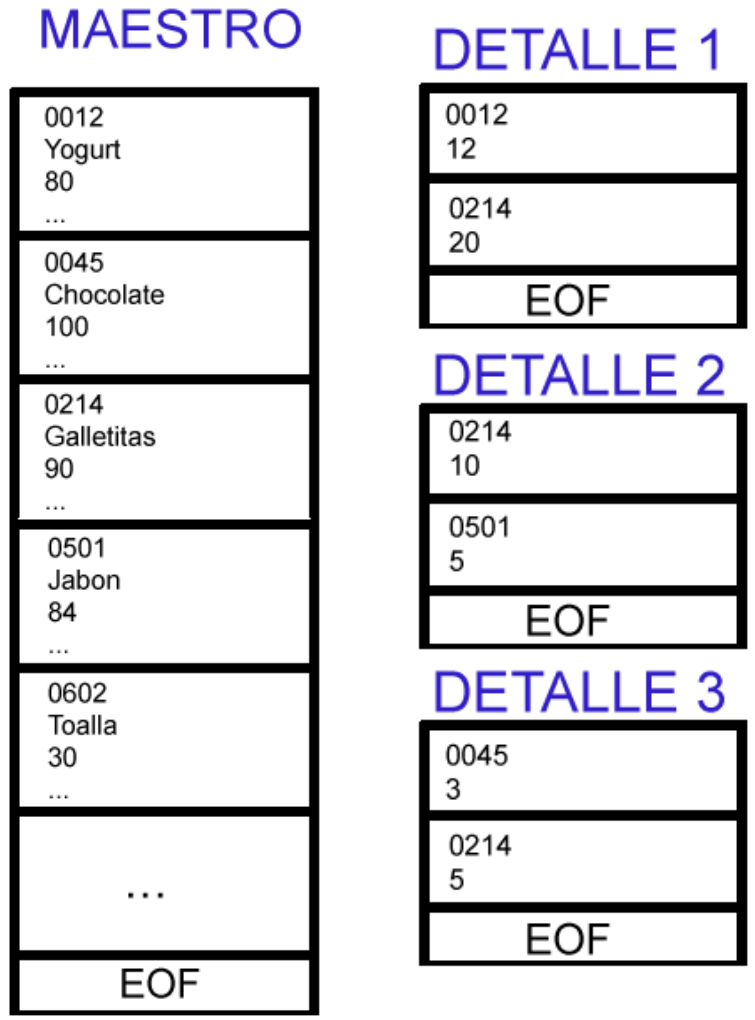
Un maestro - N detalles

- Ej 3: actualización de stock de productos <N detalles>
  - Si los archivos involucrados estuvieran desordenados, este proceso involucraría recorrer el archivo maestro en más de una ocasión
  - **Precondiciones**
    - Mismas precondiciones que para el ejemplo anterior
    - Además, el archivo maestro ahora se actualiza a partir de tres archivos detalle (N=3)

Archivos

Un maestro - N detalles

- Se procesan todos los registros de los archivos detalle con igual código.
- Se encuentra el registro en el archivo maestro y se realiza la actualización:
  - Se modifican los datos
  - Se vuelve a la posición del registro
  - Se escribe el registro actualizado
- Se continua con el siguiente registro de los archivos detalle
- Cuando no hay más registros en los archivos detalle, se finaliza



Archivos

Un maestro - N detalles

```
program actualizarN;
const
    valoralto='9999';
type
    str4 = string[4];
    prod = record
            cod: str4;
            descripcion: string[30];
            cant: integer;
            pu: real;
            (…)
    end;
    v_prod = record
            cod: str4;
            cant_vendida: integer;
    end;
    maestro = file of prod;
    detalle = file of v_prod;
```

```
var
    mae1: maestro;
    det1, det2, det3: detalle;

    regm: prod;
    min, reg1, reg2, reg3: v_prod;

    aux: str4;
```

Archivos

Un maestro - N detalles

```
procedure leer (var archivo:detalle; var dato:v_prod);
begin
    if (not eof(archivo))
      then
          read (archivo,dato)
      else
          dato.cod:= valoralto;
end;
```

Archivos

Un maestro - N detalles

```
procedure minimo (var r1, r2, r3:v_prod;  var min:v_prod
                                  var det1, det2, det3: detalle);
begin
  if (r1.cod <= r2.cod) and (r1.cod <= r3.cod)
    then begin
      min := r1;
      leer(det1,r1)
    end
    else if (r2.cod <= r3.cod)
          then begin
            min := r2;
            leer(det2,r2)
          end
          else begin
            min := r3;
            leer(det3,r3)
          end;
end;
```

Archivos

Un maestro - N detalles

```
begin    {programa principal}
    assign (mae1, 'maestro.dat');   {se asocian y se abren los archivos}
    assign (det1, 'detalle1.dat');
    assign (det2, 'detalle2.dat');
    assign (det3, 'detalle3.dat');
    reset (mae1);
    reset (det1);
    reset (det2);
    reset (det3);

    read(mae1,regm); {siempre se debe preguntar por EOF antes de leer → en este caso
    el enunciado del ejemplo dice que no está vacío}
    leer(det1, regd1);
    leer(det2, regd2);
    leer(det3, regd3);

    minimo(regd1,regd2,regd3,min,det1,det2,det3);   {se obtiene el minimo}
```

Archivos

Un maestro - N detalles

```
while (min.cod <> valoralto) do
begin
  while (regm.cod <> min.cod) do   {busca el mínimo en el maestro}
        read(mae1, regm);
  aux := min.cod;
  while (aux = min.cod ) do   {procesa el mínimo}
  begin
        regm.cant := regm.cant - min.cantvendida;
        minimo(regd1, regd2, regd3, min, det1, det2, det3);
  end;
  seek (mae1, filepos(mae1)-1);   {se reubica el puntero y actualiza}
  write(mae1, regm);
end;
close(mae1);   {se cierran los archivos}
close(det1); close(det2); close(det3);
```

Archivos

Un maestro - N detalles

```
{se visualiza en pantalla el stock de cada producto}
reset (mae1);

while (not eof(mae1)) do
begin;
  read (mae1, regm);
  writeln (regm.cod, regm.cant);
end;

close(mae1);

end.
```

Archivos

Corte de control

- Algorítmica clásica de corte de control
  - Este tipo de algoritmos permite analizar la información almacenada en archivos y generar reportes que incluyan valores promedios, subtotales, totales, etc.
  - Precondición: el archivo se encuentra ordenado por uno o más criterios, y son los mismos criterios por los que se quiere agrupar la información en el reporte

Archivos

Corte de control

- Algorítmica clásica de corte de control
  - El algoritmo consiste en ir recorriendo la información, de forma tal que cada vez que se produce un cambio en alguno de los campos correspondiente a un criterio de ordenamiento, se finaliza el cálculo relacionado y se comienza el siguiente
  - Se utilizan bucles anidados que van incrementando la cantidad de condiciones a verificar
  - El resultado es un reporte que respeta un formato predeterminado

Archivos

Corte de control

- Ej: reporte censo
  - **Precondiciones**
    - El archivo esta ordenado por: Provincia, Partido y Ciudad
    - El formato del reporte debe ser el que se visualiza a la derecha →

Provincia: AAAA			
Partido: xxxx			
Ciudad	#Hom.	#Muj.	#Des.
aaa	.....	.....	.....
bbb	.....	.....	.....
ccc	.....	.....	.....
Total Partido:			
Partido: yyyy			
Ciudad	#Hom.	#Muj.	#Des.
.....	.....	.....	.....
Total Partido:			
Total Provincia:			
Provincia: BBBB			
(…)			

Archivos

Corte de control

```
program corteDeControl;
const
    valoralto='zzzz';
type
    str10 = string[10];
    prov = record
            provincia: str10;
            partido: str10;
            ciudad: str10;
            cant_hombres : integer;
            cant_mujeres : integer;
            cant_desocupados :integer;
    end;
     datos_censo = file of prov;
```

Archivos

Corte de control

```
var
    censo: datos_censo;
    regm: prov;
    t_hombres, t_mujeres, t_desocupados: integer;
    t_prov_hom, t_prov_muj, t_prov_des: integer;
    ant_partido: str10;
    ant_prov: str10;

procedure leer (var archivo: datos_censo; var dato:prov);
begin
    if (not eof( archivo ))
    then  read (archivo,dato);
    else   dato.provincia := valoralto;
end;
```

Archivos

Corte de control

```
begin
    assign (censo, 'censo.dat' );
    reset (censo);
    leer (censo, regm);

    writeln ('Provincia: ', regm.provincia);   {encabezados}
    writeln ('Partido: ', regm.partido);
    writeln('Ciudad',‘Hombres','Mujeres','Desocupados');

    t_hombres := 0;   {inicialización contadores totales partidos }
    t_mujeres := 0;
    t_desocupados := 0;

    t_prov_hom := 0;   {inicialización contadores totales provincias }
    t_prov_muj := 0;
    t_prov_des := 0;
```

ARCHIVO CENSO						
provincia	BA	BA	BA	BA	SF	. . .
partido	LP	LP	LM	LM	RS	. . .
ciudad	LP	CB	RM	SJ	RS	. . .
cant_hom	5	7	3	5	10	. . .
cant_muj	10	8	4	5	15	. . .
cant_des	15	9	5	5	10	. . .

RESET

Archivos

Corte de control

```
while ( regm.provincia <> valoralto)do
begin
  ant_prov := regm.provincia;
  ant_partido := regm.partido;

  while(ant_prov=regm.provincia)and(ant_partido=regm.partido)do
  begin
        write (regm.ciudad, regm.cant_hombres, regm.cant_mujeres, regm.cant_desocupados);

        t_hombres := t_hombres + regm.cant_hombres;
        t_mujeres := t_mujeres + regm.cant_mujeres;
        t_desocupados:=t_desocupados+regm.cant_desocupados;

        t_prov_hom := t_prov_hom + regm.cant_hombres;
        t_prov_muj := t_prov_muj + regm.cant_mujeres;
        t_prov_des := t_prov_des+ regm.cant_desocupados;
        leer (censo, regm);
  end;   {end while <partido>}
```

Archivos

Corte de control

```
        write ('Total Partido:', t_hombres,t_mujeres,t_desocupados);
        writeln;
        t_hombres := 0;
        t_mujeres := 0;
        t_desocupados := 0;
        ant_partido := regm.partido;
        if (ant_prov <> regm.provincia)   {si era el ultimo partido de la provincia}
        then begin
                writeln ('Total Provincia:', t_prov_hom, t_prov_muj, t_prov_des);   {encabezado}
                t_prov_hom := 0;
                t_prov_muj := 0;
                t_prov_des := 0;
                writeln ('Provincia: ', regm.provincia); {encabezado}
        end;
        writeln ('Partido: ', regm.partido);   {encabezados}
        writeln('Ciudad','Hombres','Mujeres','Desocupados');
    end;    {end while principal}
end.
```

Archivos

Merge

- El proceso de merge (unión) involucra un conjunto de archivos con contenido similar, el cual debe resumirse en un único archivo
  - Se debe crear un archivo nuevo con la información resumida
- Precondiciones generales
  - Todos los archivos detalle tienen igual estructura
  - Todos los archivos detalle están ordenados por igual criterio

Archivos

Merge 3 archivos

- Ej 1: generación de listado de alumnos
  - Un instituto inscribe a los alumnos que tomarán un determinado curso en tres sucursales por separado
  - En cada una de las sucursales se genera un archivo con los datos personales de los estudiantes. Luego son ordenados físicamente por otro proceso
  - El problema consiste en generar un único archivo maestro con los alumnos del curso

Archivos

Merge 3 archivos

- Ej 1: generación de listado de alumnos
  - **Precondiciones**
    - El proceso recibe tres archivos con igual estructura
    - Los archivos están ordenados por nombre de alumno
    - Un alumno solo aparece una vez
  - **Postcondición**
    - Se genera el archivo maestro del curso ordenado por nombre de alumno

Archivos

Merge 3 archivos

```
program unionArchivos;
const
    valoralto = 'zzzz';
type
    str30 = string[30];
    str10 = string[10];
    alumno = record
            nombre: str30;
            dni: str10;
            direccion: str30;
            carrera: str10;
    end;
    detalle = file of alumno;
var
    det1, det2, det3, maestro: detalle;
    min, regd1, regd2, regd3: alumno;
```

Archivos

Merge 3 archivos

```
procedure leer (var archivo:detalle; var dato:alumno);
begin
    if (not eof( archivo ))
    then   read (archivo, dato)
    else   dato.nombre := valoralto;
end;

procedure minimo (var r1,r2,r3:alumno; var min:alumno; var det1,det2,det3: detalle);
begin
    if (r1.nombre<r2.nombre) and (r1.nombre<r3.nombre)
    then begin
            min := r1;
            leer(det1,r1)
    end
    else if (r2.nombre<r3.nombre)
    then begin
            min := r2;
            leer(det2,r2)
    end
    else begin
            min := r3;
            leer(det3,r3)
    end;
end;
```

Archivos

Merge 3 archivos

```
begin
    assign (det1, 'det1.dat');      {se asocian y se abren los archivos}
    assign (det2, 'det2.dat');
    assign (det3, 'det3.dat');
    assign (maestro, 'maestro.dat');
    reset (det1);
    reset (det2);
    reset (det3);
    rewrite (maestro);

    leer(det1, regd1);   {se lee el primer elemento y se determina el minimo}
    leer(det2, regd2);
    leer(det3, regd3);
    minimo(regd1, regd2, regd3, min, det1, det2, det3);
```

Archivos

Merge 3 archivos

```
while (min.nombre <> valoralto) do   {se procesan todos los archivos}
begin
  write (maestro,min);
  minimo(regd1, regd2, regd3, min,  det1, det2, det3);
end;
close (maestro);
close (det1); close (det2); close (det3);

reset (maestro);   {se visualiza en pantalla el nombre y dni de los alumnos inscriptos}
while (not eof(maestro)) do
begin
  read (maestro,min);
  writeln (min.nombre,min.dni);
end;
close (maestro);
end.
```

Archivos

Merge 3 archivos

- Ej 2: monto acumulado en las ventas realizadas por cada vendedor en un comercio
  - **Precondiciones**
    - El proceso recibe tres archivos con igual estructura
    - Los archivos están ordenados por código de vendedor
    - Cada vendedor puede haber realizado varias ventas → en los archivo detalle pueden existir varios registros correspondientes al mismo vendedor

Archivos

Merge 3 archivos

```
program unionArchivos3;
const
    valoralto='9999';
type
    str4 = string[4];
    str10 = string[10];
    vendedor = record
            codVendedor: str4;
            producto: str10;
            montoVenta: real;
    end;
    ventas = record
            codVendedor: str4;
            total: real;
    end;
    maestro = file of ventas;
    detalle = file of vendedor;
```

```
var
    min, reg1, reg2, reg3: vendedor;
    regm: ventas;

    det1, det2, det3: detalle;
    mae1: maestro;

    aux: str4;
```

Archivos

Merge 3 archivos

```
procedure leer (var archivo:detalle; var dato:vendedor);
begin
    if (not eof( archivo ))
    then   read (archivo, dato)
    else   dato.codVendedor := valoralto;
end;

procedure minimo (var r1,r2,r3: vendedor; var min:vendedor;  var det1,det2,det3:detalle);
begin
    if (r1.codVendedor <= r2.codVendedor) and (r1.codVendedor <= r3.codVendedor)
    then begin
            min := r1;
            leer(det1,r1)
    end
    else   if (r2.codVendedor <= r3.codVendedor)
            then begin
              min := r2;
              leer(det2,r2)
            end
            else begin
              min := r3;
              leer(det3,r3)
            end;
end;
```

Archivos

Merge 3 archivos

```
begin
    assign (det1, 'det1.dat');   {se asocian y se abren los archivos}
    assign (det2, 'det2.dat');
    assign (det3, 'det3.dat');
    assign (mae1, 'maestro.dat');
    reset (det1);
    reset (det2);
    reset (det3);
    rewrite (mae1);

    leer (det1, regd1); {se lee el primer elemento y se determina el minimo}
    leer (det2, regd2);
    leer (det3, regd3);
    minimo(regd1, regd2, regd3, min, det1, det2, det3);
```

Archivos

Merge 3 archivos

```
while (min.codVendedor <> valoralto) do {se procesan los archivos de detalles}
begin
  aux := min.codVendedor;
  regm.codVendedor := min.codVendedor;   {valor para registro del archivo maestro}
  regm.total := 0;
  while (aux = min.codVendedor ) do   {se procesan los registros de un mismo vendedor}
  begin
       regm.total := regm.total + min.montoVenta;
       minimo (regd1, regd2, regd3, min, det1, det2, det3);
  end;
  write(mae1, regm);   {se guarda el registro en el archivo maestro}
end;
close(mae1);   {se cierran los archivos}
close(det1); close(det2); close(det3);
```

Archivos

Merge 3 archivos

```
{se visualiza en pantalla el monto acumulado por producto}
reset (mae1);
while (not eof( mae1 )) do begin
  read (mae1,regm);
  writeln (regm.codVendedor, regm.total);
end;
close(mae1);

end.
```

Archivos

Merge N archivos

- Ej 3: monto acumulado en las ventas realizadas por cada vendedor en un comercio → **N archivos**
  - **Precondiciones**
    - El proceso recibe N archivos con igual estructura
    - Los archivos están ordenados por código de vendedor
    - Cada vendedor puede haber realizado varias ventas → en los archivo detalle pueden existir varios registros correspondientes al mismo vendedor

Archivos

Merge N archivos

```
program unionArchivosN;
const
    valoralto='9999';
    N = 100;
type
    str4 = string[4];
    str10 = string[10];
    vendedor = record
            cod: str4;
            producto: str10;
            montoVenta: real;
    end;
    ventas = record
            cod: str4;
            total: real;
    end;
    maestro = file of ventas;
    detalle = file of vendedor;
    fileN = array[1..N] of detalle;
    regN = array[1..N] of vendedor;

var
    mae1: maestro;
    deta: fileN

    regm: ventas;
    min: vendedor;
    regDeta: regN;

    aux: str4;
    i: integer;
```

Archivos

Merge N archivos

```
procedure leer (var archivo:detalle; var dato:vendedor);
begin

    if (not eof( archivo ))
    then
            read (archivo, dato)
    else
            dato.cod := valoralto;
end;
```

Archivos

Merge N archivos

```
procedure minimo (var reg_deta: regN;  var min:vendedor;
                  var deta:fileN);
var
    indice_min: integer;
begin
    {recorrer el arreglo de registros reg_deta determinando el elemento MINIMO. En la
    variable indice_min guardar la POSICION del elemento mínimo}
    determinarMinimo(reg_deta, indice_min);

     {guardar minimo}
    min = reg_deta[indice_min];

    {leer nuevo elemento del archivo correspondiente}
    leer(deta[indice_min], reg_deta[indice_min]);
end;
```

Archivos

Merge N archivos

```
begin
    {se preparan los archivos detalle}
    for i:= 1 to N
    do begin
      assign (deta[i], concat('det‘, IntToStr(i), '.dat'));
      reset(deta[i]);
      leer(deta[i], reg_det[i]);
    end;
    {se prepara el archivo maestro}
    assign (mae1, 'maestro.dat');
    rewrite (mae1);
    {se determina el minimo elemento}
    minimo (regDeta, min, deta);
```

Archivos

Merge N archivos

```
while (min.cod <> valoralto) do   {se procesan los archivos de detalle}
begin
  aux := min.cod;
  {valores para registro del archivo maestro}
  regm.cod := min.cod;
  regm.total := 0;
  {se procesan los reg. de un mismo vendedor}
  while (aux = min.cod ) do
  begin
        regm.total := regm.total + min.montoVenta;
        minimo (regDeta, min, deta);
  end;
  {se guarda en el archivo maestro}
  write(mae1, regm);
end;
close (mae1);
```

Archivos

Merge N archivos

```
{se visualiza en pantalla el monto acumulado por producto}
reset (mae1);
while (not eof( mae1 )) do begin
  read (mae1,regm);
  writeln (regm.cod, regm.total);
end;
close(mae1);

end.
```