TP1

Archivos Algoritmia

nahuelArn

Declaracion

```
type
    pepe = record
        a : integer;
        b : integer;
    end;
    archivoNumeros = file of integer;
    archivoPepe = file of pepe;
var
    arch : archivoNumeros;
    archPep: archivoPepe;
begin
    // nombre_fisico = ruta donde se encuentra el archivo o donde queres que se guarde
    assign(arch, nombre_fisico);  //Podes usar un String
    //tambien podes especificar la ruta relativa absluta/ directamente
    assign(archPep, 'D:\Escritorio\Entorno de prubeas de Codigos Fod\');
begin
```

Crear, Abrir, cerrar

```
// Crea un archivo y lo abre, si existe lo sobreescribe
rewrite(nombre_logico);
// Abre un archivo existente
reset(nombre_logico);
// Cierra el archivo. Transfiere los datos del buffer al disco
close(nombre_logico);
```

Lectura y escritura de archivos

Por cada lectura y escritura se mueve el puntero a la siguiente posicion

```
var
  var_dato : registro;

//con archivos normales
// --------Lee el archivo--------
read(nombre_logico, var_dato);
// Escribe el archivo
write(nombre_logico, var_dato);
```

```
var
  var_escritura : txt;
  var_dato : registro;
//------con archivos txt--------
// Lee el archivo, va cargando campo por campo
readln(var_escritura, var_dato.campo1, var_dato.campo2);
// Escribe el archivo,, va escribiendo campo por campo
writeln(var_escritura, var_dato.campo1, var_dato.campo2);
```

Archivos Algoritmia

Operaciones de archivos

```
// Controla el fin de archivo
EOF(nombre_logico);
// Devuelve el tamaño de un archivo
fileSize(nombre_logico);
// Devuelve la posición actual del puntero en el archivo
filePos(nombre_logico);
// Establece la posición del puntero en el archivo
seek(nombre_logico, pos);
// ----------ERASE  Borra un archivo------------
//tp3,primeraParte,eje6.. ejemplo aplicado
var
  f: file;
begin
  assign(f, 'test.txt');
  erase(f);
end;
//------RENAME Cambia el nombre de un archivo-------
var
  f: file;
begin
  assign(f, 'oldname.txt');
  rename(f, 'newname.txt');
end;
```

```
procedure buscarNombresConArroba(var a: file of producto);
begin
  reset(a);
  while not eof(a) do
  begin
    read(a, r);
    if (Length(r.nombre) > 0) and (r.nombre[1] = '@') then
      writeln('Nombre con @: ', r.nombre);
  end;
  close(a);
end;
```

```
Case num of //si num es 1 o 2 o 3 et etc
  1: logica1();
  2: logica2();
  3: logica3();
end;
```

```
if r.cod = cod then begin
    seek(a, filePos(a) - 1);
    r.nombre:= '@' + r.nombre; // Marca de b
    write(a, r);
    writeln('Eliminado con exito');
```

```
detalle = file of empresa;

vectDet = array[1..30]of detalle;
vectDetR = array[1..30] of empresa;
```

```
procedure asignarDirecciones(var vD: vectDet);
var
  i: integer;
  direccionInmutable,casa: string;
begin
  direccionInmutable:= 'binarioYaCargado';
  for i:= 1 to 30 do
    begin
      str(i,casa);
      assign(vD[i], direccionInmutable+casa);
    end;
end;
```

Importar/Exp Archivos

Importar Binario

```
program prueba;
type
  Archivo = file of integer;
var
  n: integer;
  arch: Archivo;
begin
  assign(arch, 'arch.dat');
  rewrite(arch);
  writeln('Ingrese un numero 0 para cortar');
  readln(n);
  while n <> 0 do
    begin
      write(arch, n); //
      writeln('Ingrese un numero 0 para cortar');
      readln(n);
    end;
  close(arch);
  //Si se qusiera abrir el archivo en otro programa
  //solo faltaria el assign
  reset(arch);
  while (not eof (arch)) do
    begin
      read(arch, n);
      writeln('El numero es: ', n);
    end;
  close(arch);
end.
```

Crear/Importar arch Binario

Importar Txt a Binario

```
Importar desde Txt a Binario
procedure importarMaestro(var mae : maestro);
var
  txt : text;
  regM : log;
begin
  assign(mae, 'maestro.dat');
  rewrite(mae);
  assign(txt, 'maestro.txt');
  reset(txt);
  while(not eof(txt))do begin
    readln(txt, regM.numUsuario, regM.cantEmails, regM.nombreUsuario);
    readln(txt, regM.nombre);
    readln(txt, regM.apellido);
    write(mae, regM);
  end;
  writeln('Archivo maestro creado');
  close(mae);
  close(txt);
end;
```

Exportar Binario a Txt

```
procedure exportarATxt(var mae : maestro);
var
  txt : text;
  regM : log;
begin
  assign(txt, 'logs.txt');
  rewrite(txt);
  reset(mae);
  while(not eof(mae))do begin
    read(mae, regM);
    writeln(txt, regM.numUsuario, ' ', regM.cantEmails);
  end;
  close(mae);
  close(txt);
end;
```

TP2

Actualizar Maestro 1Detalle (sin repetidos)

Archivo maestro:

- Resume información sobre el dominio de un problema específico. Ejemplo: El archivo de productos de una empresa.

Archivo detalle:

- Contiene movimientos realizados sobre la información almacenada en el maestro. Ejemplo: archivo conteniendo las ventas sobre esos productos.

1. PRECONDICIONES
   - Existe un archivo maestro.
   - Existe un único archivo detalle que modifica al maestro.
   - Cada registro del detalle modifica a un solo registro del maestro que seguro existe.
   - No todos los registros del maestro son necesariamente modificados.
   - Cada elemento del maestro que se modifica, es alterado por un solo un elemento del archivo detalle.
   - Ambos archivos están ordenados por igual criterio.

```
type
    producto = record
      cod: string[4];
      descripcion: string[30];
      pu: real;  {precio unitario}
      stock: integer;
    end;
    venta_prod = record
      cod: string[4];
      cant_vendida: integer;
    end;
    maestro = file of producto;
    detalle = file of venta_prod;
var
   mae: maestro;  regm: producto;
   det: detalle;  regd: venta_prod;
  begin  { Inicio del programa }
    assign(mae, 'maestro.dat');
    assign(det, 'detalle.dat');
    reset(mae);
    reset(det);
    {Loop archivo detalle}
    while not(EOF(det)) do
      begin
        read(mae, regm); // Lectura archivo maestro
        read(det, regd); // Lectura archivo detalle
        {Se busca en el maestro el producto del       detalle}
      while (regm.cod <> regd.cod) do
        begin
          read(mae, regm);
        end;
        {Se modifica el stock del producto con la
        cantidad vendida de ese producto}
        regm.stock := regm.stock-regd.cant_vendida;
        {Se reubica el puntero en el maestro}
        seek(mae, filepos(mae)-1);
        {Se actualiza el maestro}
        write(mae, regm);
      end;// Fin while archivo detalle
    close(det);
    close(mae);
end.
```

Maestro Detalle(con Repetidos) y corte de control

2. PRECONDICIONES

- Existe un archivo maestro.
- Existe un único archivo detalle que modifica al maestro.
- Cada registro del detalle modifica a un registro del maestro que seguro existe.
- No todos los registros del maestro son necesariamente modificados.
- Cada elemento del archivo maestro puede no ser modificado, o ser **modificado por uno o más elementos del detalle**.
- Ambos archivos están ordenados por igual criterio.

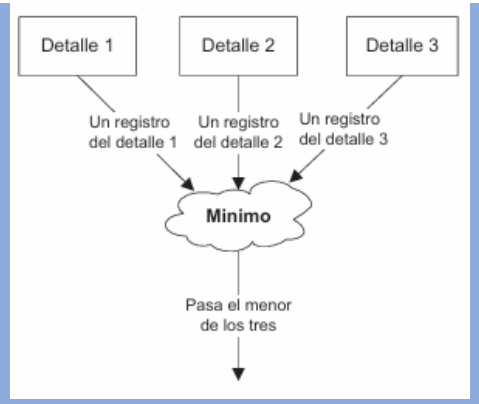
```
procedure leer(var det: detalle; var regD : venta);
begin
    if(not eof(det))then
        read(det, regD)
    else
    regD.codigo := valoralto;
end;
//actualizar Maestro con un archivo ordenado de elementos repetidos
procedure actualizarMaestro(var mae : maestro; var det : detalle);
var
  regD : venta; //registro detalle
  cantTotal, codigo : integer;
  regM : producto; //registro maestro
begin
  reset(mae);
  reset(det);
  read(mae, regM); //leo maestro
  leer(det, regD); //leo detalle
  //itero sobre los detalles + Corte de control
  while(regD.codigo <> valorAlto)do
    begin
      codigo := regD.codigo;
      cantTotal := 0;
      while (regD.codigo = codigo) do begin  //Acumulo la cantidad total
        cantTotal := cantTotal + regD.cantUnidades;
        leer(det, regD);
      end;
      while(regM.codigo <> codigo)do  {se busca el producto del detalle en el maestro}
        begin
          read(mae, regM);
        end; {se modifica el stock del producto con la
        cantidad total vendida de ese producto}
        regM.stockActual := regM.stockActual - cantTotal;
        {se reubica el puntero en el maestro}
        seek(mae, filepos(mae)-1);
        write(mae, regM); {se actualiza el maestro}
        if(not eof(mae))then  {se avanza en el maestro}
          read(mae, regM);
    end;
  writeln('Maestro actualizado');
  close(mae);
  close(det);
end;
```

Actualizacion de un archive maestro Con N Archivos Detalles

se plantea un proceso de actualización donde, ahora, la cantidad de archivos detalle se lleva a *N* (siendo *N* > 1) y el resto de las precondiciones son las mismas.

3. PRECONDICIONES
   - Existe un archivo maestro.
   - Existe varios archivos detalle que modifica al maestro.
   - Cada registro del detalle modifica a un registro del maestro que seguro existe.
   - No todos los registros del maestro son necesariamente modificados.
   - Cada elemento del archivo maestro puede no ser modificado, o ser modificado por uno o más elementos del detalle.
   - Ambos archivos están ordenados por igual criterio.

El objetivo del procedimiento `minimo` es determinar el menor de los tres elementos recibidos de cada archivo (para poder retornarlo como el más pequeño) y leer otro registro del archivo desde donde provenía ese elemento.




```pascal
program actualizacionNdetalles;
const
  cant_archivos_detalles = 100;
  valorAlto = 9999;
type
  maestro = record
    cod_orden: integer;
    stock_disponible: string;
  end;

  detalle = record
    cod_orden: integer;
    cant_vendida: integer;
  end;

  archivo_maestro = file of maestro;
  archivo_detalle = file of detalle;

  vectorDetalles = array[1..100]of archivo_detalle;
  vectorDetallesRegistros = array[1..100]of detalle;
```

1

```pascal
procedure minimo(var vD: vectorDetalles;var min: detalle; var vDetallesReg: vectorDetallesRegistros);
var
  i, pos: integer;
begin
  min.cod_orden:= valorAlto;
  for i:= 1 to cant_archivos_detalles do
    begin
if( vDetallesReg[i].cod_orden) < (min.cod_orden))then  //saco el min de todos los detalles en ese[i]
      begin
        min:= vD[i];
        pos:= i;
      end;
    end;
  if(min.cod_orden <> valorAlto)then
    begin
      min:= vD[pos];
      leer(vD[pos], vDetallesReg[pos]);
    end;
end;
```

2

Registros

Filtras en registros,
Para Avanza en el detalle

por cada campo que tengas ordenado tenes que agregar una condicion en la que consideras que los anteriores campos son iguales

Actualizacion de un archive maestro Con N Archivos Detalles

```pascal
procedure abrirDetalles(var vDetalles : vectorDetalles;var vDregistros : vectorDetallesRegistros);
var  i: integer;
begin
  for i := 1 to cant_archivos_detalles do //leo x5 detalle
    begin
      Reset(vDetalles[i]);
      leer(vDetalles[i], vDregistros[i]);
    end;
end;

procedure cerrarDetalles(var vDet : vectorDetalles);
begin
  for i := 1 to cant_archivos_detalles do
    begin
      Close(vDetalles[i]);
    end;
end;
```

3

```pascal
//se manda un maestro cargado y un vector de detalles cargado
procedure actualizarMaestroNdetalles(var mae : archivo_maestro; var vDetalles : vectorDetalles);
var
  vDregistros: vectorDetallesRegistros;
  min : archivoDetalle;
  regM : maestro;
begin
  reset(mae);
  read(mae, regM);
  abrirDetalles(vDetalles,vDregistros);
  minimo(vDetalles,min,vDregistros);  //le mando el vector de registro
  while(min.cod_orden <> valorAlto)do
    begin
      cod_actual:= min.cod_orden;
      acumulador_cant_vendida:= 0;
      while(cod_actual = min.cod_orden)do  //acumulo todos los detalles del mismo cod
        begin
          acumulador_cant_vendida:= acumulador_cant_vendida + min.cant_vendida;
          minimo(vDetalles,min,vDregistros);
        end;
      while (regM.cod_orden <> cod_actual) do //busco en el maestro
        read(maestro, regM);
      regM.stock_disponible:= regM.stock_disponible - acumulador_cant_vendida;
      Seek(maestro, filepos(maestro)-1);
      write(maestro,regM);
      if(not eof (maestro))then
        read(maestro,regM);
    end;
  writeln('Maestro actualizado');
  close(mae);
  cerrarDetalles(vDtalles);
end;
```

4

Actualizacion de un archive maestro Con N Archivos Detalles |CritriosOrdenes|

```pascal
//el mas simple, un solo criterio de orden
//Un criterio: Se compara cod.
procedure Leer(var det: archivo; var reg:registro);
begin
    if (not eof(det)) then
      read(det, reg)
    else
      reg.cod:=valorAlto;
end;

procedure Minimo(var vD: vDetalles;var vDR: vDetR; var min: registro);
var
  i, pos:integer;
begin
  min.cod:=valorAlto;
  for i:= 1 to dimF do
    begin
      if (vDR[i].cod < min.cod) then
        begin
          min:= vDR[i];
          pos:= i;
        end;
    end;
  if (min.cod <> valorAlto) then
    Leer(vD[pos], vDetR[pos]);
end;
```

1

por cada campo que tengas ordenado tenes que agregar una condicion en la que consideras que los anteriores campos son iguales

```pascal
// 2 criterios de orden
//Dos criterios: Se comparan fecha y codigo.
procedure Leer(var det:detalle; var reg: datoDetalle);
begin
    if(not eof(det)) then
        read(det, reg)
    else
      begin
        reg.fecha:=valorAlto;
        reg.codigo:=valorAlto;
      end;
end;
procedure Minimo (var vD:vDetalles; var vDR:vDetR; var min:registro);
var
  i, pos:integer;
begin
  min.fecha:=valorAlto;
  min.codigo:=valorAlto;
  for i:= 1 to df do
    begin
      if(vDR[i].fecha < min.fecha) or ((vDR[i].fecha = min.fecha) and (vDR[i].codigo < min.codigo)) then
        begin
          min:= vDR[i];
          pos:= i;
        end;
    end;
  if(min.fecha <> valorAlto) then
    Leer(vD[pos], vDR[pos]);
end;
```

2

```pascal
//3 o mas criterios de orden
//Tres o más criterios: Se comparan codFarmaco, fechaAnho y fechaMes.
procedure leer(var det: detalle; var reg:registro);
begin
    if(not eof(det)) then
        read(det, reg)
    else
      begin
        reg.codFarmaco:=valorAlto;
        reg.fechaAnho:=valorAlto;
        reg.fechaMes:=valorAlto
      end;
end;

procedure minimo(var v: vector; var vr: vector_far; var min: farmaco);
var
  i, pos: integer;
begin
  min.cod_farmaco := valorAlto;
  min.fechaAnho := valorAlto;
  min.fechaMes := valorAlto;
  for i := 1 to df do
    if (vr[i].codFarmaco < min.codFarmaco) or ((vr[i].codFarmaco = min.codFarmaco) and (vr[i].fechaAnho < min.fechaAnho)) or
      ((vr[i].codFarmaco = min.codFarmaco) and (vr[i].fechaAnho = min.fechaAnho) and (vr[i].fechaMes < min.fechaMes)) then
      begin
        min := vr[i];
        pos := i;
      end;
  if (min.cod_farmaco <> valorAlto) then
    leer(v[pos], vr[pos]);
end;
```

3

Corte de control en Actualizacion de Maestro

```pascal
procedure leer(var a: detalle; var r: infoDet);
begin
  if(not eof(a))then
    read(a,r)
  else
    begin
      r.destino:= valorAltoString;
      r.fecha:= valorAltoString;
      r.horaDeSalida:= valorAlto;
    end;
end;

//orden Destino > fecha > hora de salida
procedure minimo(var vD: vecDet;var vDR: vecDetR;var min: infoDet);
var
  pos,i: integer;
begin
  min.destino:= valorAltoString;
  min.fecha:= valorAltoString;
  min.horaDeSalida:= valorAlto;
  for i:= 1 to 2 do
    begin
      if(vDR[i].destino < min.destino) or (vDR[i].destino = min.destino) and (vDR[i].horaDeSalida < min.horaDeSalida) or (
        vDR[i].destino = min.destino and vDR[i].horaDeSalida = min.horaDeSalida and vDR[i].horaDeSalida < min.horaDeSalida) then
        begin
          min:= vDR[i];
          pos:= i;
        end;
    end;
  if(min.destino <> valorAltoString)then
    begin
      min:= vDR[pos];
      leer(vD[pos],vDR[pos]);
    end;
end;
```

```pascal
procedure actualizarMaestro(var arch: maestro; var vD: vecDet; var l: lista);
var
  vDR: vecDetR;
  min: infoDet;
  rM: infoVuelos;
  destinoActual, fechaActual: string;
  horaDeSalidaActual,cantTotAsien,cantidad: integer;
begin
  reset(arch);
  leerPrimeraVez(vD,vDR);
  minimo(vD,vDR,min);
  read(arch,rM);
  Writeln('Ingrese un cantidad ');
  readln(cantidad);
  while(min.destino <> valorAlto)do
    begin
      destinoActual:= min.destino;
      while (destinoActual = min.destino) do
        begin
          fechaActual:= min.fecha;
          while(destinoActual = min.destino and fechaActual = min.fecha)do
            begin
              horaDeSalidaActual:= min.horaDeSalida;
              cantTotAsien:= 0;
              while(destinoActual = min.destino and fechaActual = min.fecha and horaDeSalidaActual = min.horaDeSalida)do
                begin
                  cantTotAsien:= cantTotAsien+min.cantAsientosComp;
                  minimo(vD,vDR,min);
                end;
              while(rM.destino <> destinoActual or rM.fecha <> fechaActual or rM.horaDeSalida<> min.horaDeSalida)do
                begin
                  if(rM.cantAsientosDisponibles < cantidad)then
                    agregarAdelante(l,rM);
                  read(arch,rM);
                end;
              rM.cantAsientosDisponibles:= rM.cantAsientosDisponibles - cantTotAsien;
              seek(arch, filePos(arch)-1);
              write(arch,rM);
              if(rM.cantAsientosDisponibles < cantidad)then
                agregarAdelante(l,rM);
              if(not eof(arch))then
                read(arch,rM);
            end;
        end;
    end;
    while not eof(arch)do //por si me quedan data que leer todavia en el maestro
      begin
        if(rM.cantAsientosDisponibles < cantidad)then
          agregarAdelante(l,rM);
        read(arch,rM);
      end;
    close(arch);
    cerrarDetalles(vD);
end;
```

OR PARA BUSCAR EN EL MAESTRO
CUANDO TENGO VARIOS CRITERIOS

Otra variante de lo de arriba

```
procedure minimo(var d1, d2:archivo ; var min, regD1, regD2:vuelo);
begin
    if (regD1.destino < regD2.destino) or
    ((regD1.destino = regD2.destino) and (regD1.fecha < regD2.fecha)) or
    ((regD1.destino = regD2.destino) and (regD1.fecha = regD2.fecha) and (regD1.horaSalida < regD2.horaSalida) ) then
    begin
        min:= regD1;
        leer(d1, regD1);
    end
    else
    begin
        min:= regD2;
        leer(d2, regD2);
    end;
end;

procedure entraLista(var l, ultimo:lista;regM:vuelo;asientos:integer);
begin
    if(regM.asientos<asientos)then
        agregarAtras(l, ultimo, regM);
end;

var
    min, regM, regD1, regD2:vuelo;
    asientos:integer;
    ult:lista;
```

```
begin
    writeln('Ingrese la cantidad de asientos que desea filtrar: ');
    readln(asientos);
    reset(m);
    reset(d1);
    reset(d2);
    leer(m, regM);
    leer(d1, regD1);
    leer(d2, regD2);
    minimo(d1, d2, min, regD1, regD2);
    while(min.destino <> valorAlto) do
    begin
        while(regM.destino <> min.destino) or (regM.fecha <> min.fecha) or (regM.horaSalida <> min.horaSalida)do begin
            entraLista(l,ult,regM,asientos);
            leer(m, regM);
        end;
        while(regM.destino = min.destino) and (regM.fecha = min.fecha) and (regM.horaSalida = min.horaSalida)do
        begin
            regM.asientos:= regM.asientos - min.asientos;
            minimo(d1, d2, min, regD1, regD2);
        end;
        seek(m,FilePos(m)-1);
        write(m,regM);
    end;
    entraLista(l,ult,regM,asientos);
    while(not eof(m)) do begin
        read(m,regM);
        entraLista(l,ult,regM,asientos);
    end;
    close(m);
    close(d1);
    close(d2);
end;
```

Actualizacion de un archive maestro
Con N Archivos Detalles

```
procedure leerPrimeraVez(var vD: vecDetalle; var vDR: vecRegistros);
  i: integer;
begin
  for i:= 1 to N do
    begin
      reset(vD[i]);
      read(vD[i], vDR[i]);
    end;
end;

procedure leer(var d: vDetalle; var min: venta);
begin
  if(not eof(d))then
    read(d,min)
  else
    min.codProducto:= valorAlto;
end;

procedure minimo(var vD: vecDetalle;var vDR: vecRegistros;var min: venta);
var
  i,pos: integer;
begin
  min.codProducto:= valorAlto;  //si tengo mas criterios de orden, tengo q iniciarlo en valor alto el min
  for i:= 1 to N do //se encarga de iterar en el vec registro de detalles
    begin
      if(vDR[i] < min.codProducto)then  //filtra al minimo
        begin
          min:= vDR[i];
          pos:= i;
        end;
    end;
  if(min.codProducto <> valorAlto)then //avanza en 1 del minimo encontrado
    begin
      min:= vDR[pos];
      leer(vD[pos], vDR[pos]);
    end;
end;
```

```
procedure Leer(var det:detalle; var registro: datoDetalle);
begin
    if(not eof(det)) then
        read(det, registro)
    else begin
        registro.fecha:=valorAlto;
        registro.codigo:=valorAlto;
    end;
end;
procedure Minimo (var detalles:arrDet; var registros:arrReg; var min:datoDetalle);
var
    i, pos:integer;
begin
    min.fecha:=valorAlto; min.codigo:=valorAlto;
    for i:= 1 to df do
        if(registros[i].fecha < min.fecha) or ((registros[i].fecha = min.fecha) and (registros[i].codigo < min.codigo)) then begin
            min:=registros[i];
            pos:=i;
        end;
    if(min.fecha <> valorAlto) then
      Leer(detalles[pos], registros[pos]);
end;
```

Merge Con Repetidos

El segundo grupo de problemas presentado en este capítulo está vinculado con la generación de un archivo que resuma información de uno o varios archivos existentes. Este proceso recibe el nombre de *merge* o unión, y la principal diferencia con los casos previamente analizados radica en que el archivo maestro no existe, y por lo tanto debe ser generado.

Segundo ejemplo

Como segundo ejemplo se presenta un problema similar, pero ahora los elementos se pueden repetir dentro de los archivos detalle, modificando de esta forma la tercera precondición del ejemplo anterior. El resto de las precondiciones permanecen inalteradas.

```
procedure asignarDetalles(var vD: vecDetalle);
var
  direccionMutable,casa: string;
begin
  for i := 1 to N do
    begin
      str(i, casa); //entero... parseo a casa a String
      direccionMutable :=  'detalle'+casa;
      assign(vD[i], direccionMutable);
    end;
end;
```

```
program actualizacionNdetalles;
const
  cant_archivos_detalles = 100;
  valorAlto = 9999;
type
  maestro = record
    cod_orden: integer;
    stock_disponible: string;
  end;

  detalle = record
    cod_orden: integer;
    cant_vendida: integer;
  end;

  archivo_maestro = file of maestro;
  archivo_detalle = file of detalle;

  vectorDetalles = array[1..100]of archivo_detalle;
  vectorDetallesRegistros = array[1..100]of detalle;

procedure leer(var det : archivo_detalle; var regD : detalle);
begin
  if(not eof(det))then
    read(det, regD)
  else
    regD.provincia := valorAlto;
end;
```

Merge

```
procedure minimo(var vD: vectorDetalles;var min: detalle; var vDetallesReg: vectorDetallesRegistros);
var
  i, pos: integer;
begin
  min.cod_orden:= valorAlto;
  for i:= 1 to cant_archivos_detalles do
    begin
      if((vD[i].cod_orden) < (min.cod_orden))then  //saco el min de todos los detalles en ese[i]
        begin
          min:= vD[i];
          pos:= i;
        end;
    end;
  if(min.cod_orden <> valorAlto)then
    begin
      min:= vD[pos];
      leer(vD[pos], vDetallesReg[pos]);
    end;
end;

procedure abrirDetalles(var vDetalles : vectorDetalles;var vDregistros : vectorDetallesRegistros);
var  i: integer;
begin
  for i := 1 to cant_archivos_detalles do //leo x5 detalle
    begin
      Reset(vDetalles[i]);
      leer(vDetalles[i], vDregistros[i]);
    end;
end;

procedure cerrarDetalles(var vDet : vectorDetalles);
begin
  for i := 1 to cant_archivos_detalles do
    begin
      Close(vDetalles[i]);
    end;
end;
```

Merge

```
//se manda un maestro cargado y un vector de detalles cargado
procedure actualizarMaestroNdetalles(var mae : archivo_maestro; var vDetalles : vectorDetalles);
var
  vDregistros: vectorDetallesRegistros;
  min : archivoDetalle;
  regM : maestro;
begin
  rewrite(mae); //CREO EL MAESTRO
  abrirDetalles(vDetalles,vDregistros);
  minimo(vDetalles,min,vDregistros);  //le mando el vector de registro  /LEO XN
  while(min.cod_orden <> valorAlto)do
    begin
      cod_actual:= min.cod_orden;
      acumulador_cant_vendida:= 0;
      while(cod_actual = min.cod_orden)do  //acumulo todos los detalles del mismo cod
        begin
          acumulador_cant_vendida:= acumulador_cant_vendida + min.cant_vendida;
          minimo(vDetalles,min,vDregistros);
        end;
      regM.cod_orden:= cod_actual;
      regM.stock_disponible:= acumulador_cant_vendida;
      write(maestro,regM);
    end;
  writeln('Maestro actualizado');
  close(mae);
  cerrarDetalles(vDtalles);
end;
```

Merge con data Disjunta

- Se tiene información en tres archivos detalle.
- Esta información se encuentra ordenada por el mismo criterio en cada caso.
- La información es disjunta; esto significa que un elemento puede aparecer una sola oportunidad en todo el problema. Si el elemento 1 está en el archivo detalle1, solo puede aparecer una vez en este y no podrá estar en el resto de los archivos.

Primer ejemplo

El primer ejemplo plantea un problema muy simple. Las precondiciones son las siguientes:

- Se tiene información en tres archivos detalle.
- Esta información se encuentra ordenada por el mismo criterio en cada caso.
- La información es disjunta; esto significa que un elemento puede aparecer una sola oportunidad en todo el problema. Si el elemento 1 está en el archivo detalle1, solo puede aparecer una vez en este y no podrá estar en el resto de los archivos.

Merge

```
program ejemplo_3_5;

  const valoralto='9999';

  type str4 = string[4];

       producto = record
         codigo: str4;
         descripcion: string[30];
         pu: real;
         cant: integer;
       end;

       detalle = file of producto;
var

 min, regd1, regd2,regd3: producto;
 det1, det2, det3, mae1: detalle;

procedure leer (var archivo:detalle; var dato:producto);
begin

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

```
procedure minimo (var r1,r2,r3,min:producto);
begin
  if (r1.codigo<=r2.codigo) and (r1.codigo<=r3.codigo) 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;

{programa principal}
begin
  assign (mae1, 'maestro');
  assign (det1, 'detalle1');
  assign (det2, 'detalle2');
  assign (det3, 'detalle3');

  rewrite (mae1);
  reset (det1);
  reset (det2);
  reset (det3);

  leer(det1, regd1);
  leer(det2, regd2);
  leer(det3, regd3);

  minimo(regd1,regd2,regd3,min);

  {se procesan todos los registros de los archivos detalle}
   while (min.codigo <> valoralto) do
    begin
      write(mae1, min);
      minimo(regd1,regd2,regd3,min);
    end;

  close (det1);
  close (det2);
  close (det3);
  close (mae1);

end.
```

Corte de Control

1

```
while (reg.provincia <> valor_alto)do begin
   writeln('Provincia:', reg.provincia);
   prov := reg.provincia;
   totProv := 0;
   while (prov = reg.provincia) do begin
      writeln('Ciudad:', reg.ciudad);
      ciudad := reg.ciudad;
      totCiudad := 0;
      while (prov = reg.provincia) and
            (ciudad = reg.ciudad) do begin
         writeln('Sucursal:',reg.sucursal);
         sucursal := reg.sucursal;
         totSuc := 0;
```

2

```
while (prov = reg.provincia) and
      (ciudad = reg.ciudad) and
      (sucursal = reg.sucursal) do begin

   write("Vendedor:", reg.vendedor);
   writeln(reg.monto);
   totSuc := totSuc + reg.monto;
   leer(archivo, reg);
end;
```

3

```
            writeln("Total Sucursal",totSuc);
            totCiudad := totCiudad + totSuc;
         end;{while (prov = reg.provincia) and
            (ciudad = reg.ciudad)}
         writeln("Total Ciudad", totCiudad);
         totProv := totProv + totCiudad;
      end;{while(prov = reg.provincia)}
      writeln("Total Provincia", totProv);
      total := total + totProv,
   end;{while(reg.provincia <> valor_alto)}
   writeln("Total Empresa", total);
   close(archivo);
end.
```

```
procedure corteDeControl(var mae: maestro);
var
    prov, actual: provincia;
    total, totalProvincia, totalLocalidad, provActual, localidadActual: integer;
begin
    reset(mae);
    leer(mae, prov);
    total:= 0;
    while(prov.codProv <> valoralto) do begin
        writeln();
        writeln('CodigoProv: ', prov.codProv);
        totalProvincia:= 0;
        provActual:= prov.codProv;
        while(provActual = prov.codProv) do begin //Misma provincia
            writeln('CodigoLocalidad          Total de Votos');
            localidadActual:= prov.codLocalidad;
            totalLocalidad:= 0;
            while((provActual = prov.codProv) and (localidadActual = prov.codLocalidad)) do begin //Misma localidad
                totalLocalidad:= totalLocalidad + prov.cantVotos;
                leer(mae, prov);
            end;
            writeln(localidadActual, '                          ', totalLocalidad);
            totalProvincia:= totalProvincia + totalLocalidad;
        end;
        writeln('Total de votos Provincia: ', totalProvincia);
        total:= total + totalProvincia;
    end;
    writeln();
    writeln('Total General de Votos: ', total);
    close(mae);
end;
```

Marcado para Baja Fisica

```
procedure marcarRegistros(var arch: archivo);
var
  codAliminar: integer;
  r: ave;
  encontrado: boolean;
begin
  writeln('Ingrese el código a eliminar (50000 para finalizar): ');
  readln(codAliminar);
  while codAliminar <> 50000 do
    begin
      encontrado:= false;
      while not eof(arch) and not encontrado do //el encontrado no va si existe repediso se tendria que recorrer completo el archivo
        begin
          read(arch, r);
          if(r.cod = codAliminar)then
            begin
              seek(arch, filePos(arch)-1);
              r.cod:= r.cod-1;
              write(arch, r);
              encontrado:= true; //si existen repetidos esto no va
            end;
        end;
        writeln('Ingrese el código a eliminar (50000 para finalizar): ');
        readln(codAliminar);
        Seek(arch, 0);
    end;
end;
```

Baja Fisica

Aca no existe la cabecera!

Compactar,
Este era el algoritmo ineficiente
pero mas fácil de hacer, cumple
va al final del archivo y swapea por uno

```
procedure compactar(var mae:archivo);
var
    pos:integer; aux, cambiazo:registro;
begin
    Reset(mae);
    leer(mae,aux);
    while (aux.num <> valorAlto) do begin
        if (aux. marcado = -1) then
          begin
            pos:=(filepos(mae)-1);      //Guardamos la posicion
            Seek(mae,filesize(mae)-1);  //Vamos al final y agarramos el ultimo
            leer(mae,cambiazo);
            Seek(mae,pos);
            Write(mae,cambiazo);       //Escribimos el reemplazo
            Seek(mae,filesize(mae)-1);
            Truncate(mae);
            Seek(mae,pos);        //Volvemos para atras (chequeamos si el reemplazo estaba eliminado)
          end;
        leer(mae, aux);
    end;
    Close(mae);
    Writeln('Compactado');
end;
```

Crear un nuevo archivo sin los elementos que quisiera eliminar

```
procedure leer(var archivo: archivo_empleados; var reg: empleado);
begin
  if(not eof (archivo))then
    read(archivo, reg)
  else
    reg.nombre:= 'Corte'; //"Carlos Garcia" valor de corte
end;
begin {se sabe que existe Carlos Garcia}
    assign (archivo, 'arch_empleados');
    assign (archivo_nuevo, 'arch_nuevo');
    reset (archivo);
    rewrite (archivo_nuevo);
    leer (archivo, reg);
  {se copian SOLO los registros q no sean Carlos Garcia}
    while (reg.nombre <> 'Corte') do begin
      if(reg.nombre <> 'Carlos Garcia')then
        begin
          write (archivo_nuevo, reg);
        end;
      leer (archivo, reg);
    end;
    close(archivo_nuevo);
    close(archivo);
end.
```

Compactacion/Baja Fisica/Compactar

COMPACTACION ARCHIVO

Quita los registros marcados como eliminados, utilizando cualquiera de los algoritmos vistos para baja física.

- 2 maneras,
  Usando un nuevo archivo guardando los que no esten marcados
- -Swapeando con el ultimo valor del archivo

```
procedure leer(var arch: archivo; var r: registro);
begin
  if(not eof (arch))then
    read(arch,r)
  else
    r.num:= valorAlto;
end;
```

```
procedure compactar (var arc_log,aux:archivo);
var
n:prendas;
begin
    reset(arc_log);
    rewrite(aux);
    leerArc(arc_log,n);
    while(n.cod <> valorAlto) do begin
        if (n.stock >= 0) then begin
            write (aux,n);
        end;
        leerArc(arc_log,n);
    end;
    close (arc_log);
    close(aux);
end;

    actualizar (arc_log,aEliminar);
    compactar (arc_log,arch_nuevo);
    erase (arc_log);
    rename(arch_nuevo,'maestro.dot');
    writeln('NUEVO MAESTRO');
    writeln ('');
    mostrar(arch_nuevo);
end.
```

```
procedure compactar(var mae:archivo);
var
    pos:integer; aux, cambiazo:registro;
begin
    Reset(mae);
    leer(mae,aux);
    while (aux.num <> valorAlto) do begin
        if (aux. marcado = -1) then
          begin
            pos:=(filepos(mae)-1);      //Guardamos la posicion
            Seek(mae,filesize(mae)-1);  //Vamos al final y agarramos el ultimo
            leer(mae,cambiazo);
            Seek(mae,pos);
            Write(mae,cambiazo);       //Lo ponemos en la posicion del marcado
            Seek(mae,filesize(mae)-1);
            Truncate(mae);
            Seek(mae,pos); Vuelve a la pos donde comenzo, puede ser el ultimo
        end;
        leer(mae, aux);
    end;
    Close(mae);
    Writeln('Compactado');
end;
```

Baja Logica

La famosa “Lista Inverta”, usa una cabecera como un falso enlace

Marcar

```
procedure eliminar_asistentes_logico(var archivo_logico: archivo)
var
  a: asistente;
begin
  reset(archivo_logico);
  while (not eof (archivo_logico))do
    begin
      read(archivo_logico, a);
      if(a.nro < 1000)then
        begin
          a.apellido:= '@' + a.apellido;
          seek(archivo_logico, filepos(archivo_logico) - 1);
          write(archivo_logico, a);
        end;
    end;
  close(archivo_logico);
end;
```

Baja, Mauro clean

```
{eliminar solo un elemento}
procedure eliminarDinosaurios (var a : tArchDinos);
var
    codigoEliminar : integer;
    rLectura,cabecera : recorDinos;
begin
    reset(a);
    leer(a,cabecera);

    write ('Ingrese el codigo a eliminar: ');
    readln(codigoEliminar);
    {busco si existe el codigo}
    while ( (reg.codigo <> valorAlto) and (reg.codigo <> codigoEliminar) ) do
        leer(a,reg);
    if (reg.codigo = codigoEliminar) then
    begin
        seek(a,filePos(a)-1); {vuelvo a la posicion anterior}
        write(a,cabecera);    {escribo mi cabecera en la pos a borrar}
        cabecera.codigo := (filePos(a)-1) * -1; {me paso el indice a negativo}
        seek(a,0);
        write(a,cabecera); {actualizo la nueva cabecera}
        writeln('Se elimino correctamente.');
    end
    else writeln('El codigo no existe.');


end;
```

Baja Logica y Alta usando Registrs marcados

Dar de baja involucra
-leo cabecera
-buscar y encontrar
-encontre lo escribo con lo que tenia la cabecera
-me guardo la pos del archivo eliminado
-vuelvo a la cabecera y actualizo con esa posicion

Baja

```
procedure eliminar(var mae:maestro);
var
    cabecera, reg:datoMae; cod, pos:integer; encontrado:Boolean;
begin
    Reset(mae);
    Writeln('Introduzca un codigo a buscar');
    Readln(cod);
    encontrado:=False;
    read(mae, cabecera);
    while (not eof(mae)) and (not encontrado) do begin  //lo busco primero
        read(mae,reg);
        encontrado:= (reg.cod = cod);
    end;
    if (encontrado) then  //si encuentro
      begin
        Seek(mae, FilePos(mae)-1);  //vuelvo al nodo ha eliminar
        pos:= (FilePos(mae) * -1);  //me guardo la pos en la que voy a eliminar
        Write(mae, cabecera); //Sobreescribo con los datos de la cabecera
        Seek(mae, 0); //vuelvo a la cabecera
        reg.cod:= pos; //guardo la pos en el campo cod
        Write(mae, reg);     //Escribo el puntero actual en la cabecera
      end;
    Close(mae);
    mostrar(mae);
end;
```

Dar de Alta involucra
-leo la cabecera voy a esa pos si es < 0
-leo la pos esa me guardo los datos
-escribo esa pos con el nuevo elemento
-vuelvo a la cabecera, escribo la cabecera con los datos anteriormente guardados

Alta

```
procedure darDeAlta(var mae:maestro);
var
    aux, cabecera:datoMae; pos:integer;
begin
    Reset(mae);
    Read(mae, cabecera);
    leerNovela(aux);
    if (cabecera.cod = 0) then
      begin
        WriteLn('No hay espacio vacio');
        Seek(mae, FileSize(mae)); //se inserta al final
        Write(mae, aux);
      end
    else
      begin
        pos:= cabecera.cod * -1;//me guardo la pos en la que voy a insertar
        Seek(mae, pos); //voy a la pos
        Read(mae, cabecera);  //leo en la cabecera el contenido de la pos
        Seek(mae, FilePos(mae)-1);  //vuelvo a la pos
        Write(mae, aux);  //escribo el nuevo registro
        Seek(mae, 0); //vuelvo a la cabecera
        Write(mae, cabecera);//actualizo la cabecera
      end;
    Close(mae);
    WriteLn('Alta realizada con exito');
    mostrar(mae);
end;
```

Data de Errores

Pautas de Corrección

Los errores pueden ser inadmisibles, graves, moderados o leves.

Para aprobar un tema no se puede tener ningún error inadmisible, a lo sumo un error grave con no más de dos leves, hasta dos moderados (o hasta uno moderado con no más de tres leves) con no más de uno leve si no se tiene ninguno grave, o hasta cuatro leves si no se tiene ninguno moderado ni grave.

Errores posibles

No usar la lectura especial para generar un código de sucursal máximo o inexistente cuando se termina el archivo: grave. Si no se usa la lectura especial y no se reportan totales del último isbn, el último autor, la última sucursal y el total genera se reporta inconsistente, el error es inadmisible.

No abrir correctamente (con *reset*) ni cerrar el archivo de ventas: grave.

Reportar en pantalla y no en un archivo de texto: grave.

No abrir correctamente (con *rewrite*) el archivo de texto: grave.

No asignar, abrir ni cerrar el archivo de texto: inadmisible.

Abrir, pero no cerrar archivos: moderado.

Abrir archivo de texto, pero imprimir en pantalla: leve.

No validar igualdad de código de sucursal en ciclos de autor y de libro, o de autor en ciclo de libro: moderado (en cada ciclo que cometa un error).

Usar el procedimiento de lectura especial pero no codificarlo: moderado.

No inicializar un contador o el valor de referencia actual (sucActual, autorActual o isbnActual) o no reportar encabezado en el archivo al iniciar un ciclo: grave.

No reportar un total o hacerlo incorrectamente al producirse un corte de control: grave.

Incrementar todos los totalizadores por unidad en lugar de acumulativamente al haber un corte de control: leve.

Data de Errores

Pautas de Corrección

Los errores pueden ser inadmisibles, graves, moderados o leves.

Opciones para aprobar un tema: no tener errores inadmisibles / 1 grave y hasta 2 leves / 4 leves / 2 moderados y hasta 1 leve / 1 moderado y hasta 3 leves. Cualquier otra combinación desaprueba el tema.

Posibles errores:

- Asignar y/o abrir-cerrar el archivo dentro de la función cuando se indica que el archivo ya está abierto y no debe cerrarse: si se abre el archivo con la intención de reposicionarse al comienzo, es un error leve (performance). Si abren y también cierran el error es grave (análisis).

- Posicionamiento en 0 en vez de en 1 para búsqueda de duplicado: error leve (performance). Si no se posicionan en 0 o 1 error muy grave (lógica).
- No buscar duplicado: error grave (análisis).
- Buscar duplicado sin verificar validez de registro: error grave (lógica).
- Encontrar duplicado y seguir leyendo hasta fin de archivo: error moderado (lógica y performance).
- No buscar posición libre y agregar al final: error muy grave (análisis).
- Inconsistencia en el manejo de posiciones libres: muy grave (lógica-algoritmia).

No de admite ningún error muy grave. Se tolera hasta un error grave. Todos los leves son tolerables.

Errores posibles

No asignar y abrir con reset los 5 detalles: inadmisible.

No abrir correctamente (con *reset*) el archivo maestro: grave.

Corte de control principal por eof o valor alto del maestro (procesa registros sin sentido): si la solución es eficaz, moderado, si no, inadmisible.

No realizar un proceso que devuelva mínimo de los detalles: grave.

No inicializar por cada código diferente un contador de revistas: inadmisible.

Volver atrás en detalles: inadmisible.

No buscar en el maestro el registro a actualizar: inadmisible.

No volver atrás en el maestro para escribir: inadmisible.

No actualizar maestro: inadmisible.

No cerrar el maestro: grave.

No procesar en simultáneo detalles y maestro: inadmisible.

Recorrer más de una vez los archivos: inadmisible.

No hacer un procedimiento: leve.

Leer más de una vez al sacar el mínimo (perder registros): inadmisible (solo se avanza en el archivo que tenga el código mínimo).

No utilizar estructura auxiliar para trabajar los 5 detalles y los registros correspondientes a cada detalle: inadmisible, contradice el enunciado, la solución no es escalable.

Cuando usar el procedimiento Leer? Cuando el archive que estas procesando puede contener repetidos

Cosas de listas

```
procedure agregarAtras(var L,Ult: lista; d: cosa);
var
  nue: lista;
begin
  new(nue);
  nue^.dato:= d;
  nue^.sig:= nil;
  if(L = nil)do //si es el primer nodo
    L:= nue;
  else      // si no es el primer nodo
    Ult^.sig:= nue;
  Ult:= nue;
end;
```

```
procedure agregarAdelante(var L: lista; d: cosa);
var
  nue: lista;
begin
  new(nue);
  nue^.dato:= d;
  nue^.sig:= L;
  L:= nue;
end;
```

```
procedure insertarOrdenado(var L: lista; d: cliente);
var
  nue: lista;
  ant,act: lista;
begin
  new(nue);
  nue^.dato:= d;
  ant:= L;
  act:= L;
  While(act <> nil)and(d.dni > act^.dato.dni);  // > ascendente | < descendente
    begin
      ant:= act;
      act:= act^.sig;
    end;
    if(act = ant)then //al principio o vacio
      L:= nue
    else
      ant^sig:= nue;
    nue^.sig:= act;
end;
```

Baja Logica y Alta usando Registrs marcados

SIRVE PARA NOSE

Baja

```
procedure opcion_c(var flor: tArchFlores; del: reg_flor); //BAJAAAAA
var
  r: reg_flor;
  c: reg_flor;
  encontrado: boolean;
  pos_ultimo_eliminado: integer;
begin
  Reset(flor);
  read(flor,c);
  encontrado:= false;
  while not eof(flor) and (not encontrado)do
  begin
    read(flor,r);
    if(r.codigo = del.codigo)then
      begin
        encontrado:= true;
        r.codigo:= c.codigo;  //actualizo el codigo del registro con el que tenia la cabecera
        seek(flor,FilePos(flor)-1); //me vuelvo a posicionar donde estaba
        pos_ultimo_eliminado:= FilePos(flor) * -1;// me guardo la posicion del registro eliminado negativo
        write(flor,r);
        Seek(flor,0);
        c.codigo:= pos_ultimo_eliminado;
        write(flor,c);
      end;
  end;
  Close(flor);
end;
```

Leo cabecera

Busco un Dato en el archivo y si lo encuentro hago una baja logica, con lista invertida (cabecera)

guardo en la pos, lo que apuntaba la cabecera

la cabecera queda apuntando al nuevo elemento eliminado

Alta

```
procedure opcion_a(var flor: tArchFlores; flor_reg: reg_flor);
var
  puntero_cabecera: integer;
  cabecera: reg_flor;
  reg_apuntado: reg_flor;
begin
  reset(flor);
  //dimension fisica de un archivo
  read(flor,cabecera);
  if(cabecera.codigo = 0)then
    begin
      Seek(flor,FileSize(flor));
      write(flor,flor_reg);
    end
  else
    begin
      puntero_cabecera:= cabecera.codigo *-1; //saco la posicion del registro a reutilizar
      seek(flor,puntero_cabecera);
      read(flor,reg_apuntado);  //leo el registro apuntado
      Seek(flor,puntero_cabecera);
      write(flor,flor_reg); //escribo el nuevo registro
      seek(flor,0);
      write(flor,reg_apuntado); //actualizo la cabecera
    end;
  Close(flor);
end;
```

si es 0 la cabecera, entonces no hay ninguno marcado, se inserta al final

se usa la cabecera, como falso enlace

```
{
  Se cuenta con un archivo que almacena información sobre especies de aves en vía
de extinción, para ello se almacena: código, nombre de la especie, familia de ave,
descripción y zona geográfica. El archivo no está ordenado por ningún criterio. Realice
un programa que elimine especies de aves, para ello se recibe por teclado las
especies a eliminar. Deberá realizar todas las declaraciones necesarias, implementar
todos los procedimientos que requiera y una alternativa para borrar los registros. Para
ello deberá implementar dos procedimientos, uno que marque los registros a borrar y
posteriormente otro procedimiento que compacte el archivo, quitando los registros
marcados. Para quitar los registros se deberá copiar el último registro del archivo en la
posición del registro a borrar y luego eliminar del archivo el último registro de forma tal
```

Files

main

Go to file

- tp2_maestro_detalle_...
- tp3_bajas_archivos_s...
  - pdfs
  - primeraParte
    - eje1
    - eje2
    - eje3
    - eje4
    - eje5
    - eje6
      - archivo
      - eje6.pas
      - tempCodeRunner...
    - eje7
      - eje7.pas
    - eje8

Fundamentos-De-Organizacion-De-Datos-FOD / Practicas-Resueltas / 1-Archivos / tp3_bajas_archivos_secuenciales / primeraParte / eje7 / **eje7.pas**

"nahuel" orden

Code | Blame | 113 lines (110 loc) · 3.39 KB

```
{
  Se cuenta con un archivo que almacena información sobre especies de aves en vía
de extinción, para ello se almacena: código, nombre de la especie, familia de ave,
descripción y zona geográfica. El archivo no está ordenado por ningún criterio. Realice
un programa que elimine especies de aves, para ello se recibe por teclado las
especies a eliminar. Deberá realizar todas las declaraciones necesarias, implementar
todos los procedimientos que requiera y una alternativa para borrar los registros. Para
ello deberá implementar dos procedimientos, uno que marque los registros a borrar y
posteriormente otro procedimiento que compacte el archivo, quitando los registros
marcados. Para quitar los registros se deberá copiar el último registro del archivo en la
posición del registro a borrar y luego eliminar del archivo el último registro de forma tal
de evitar registros duplicados.
Nota: Las bajas deben finalizar al recibir el código 500000
}
program eje7;
const
  fin = 5000;
type
  str20 = string[20];
  aves = record
    codigo: integer;  //LONGITUD FIJAA, ESTO SOLO SE PUEDE HACER CON ARCHIVOS DE LONGITUD FIJA
    nombre_de_especie: str20;
    familia: str20;
    descripcion: str20;
```