TP1

Archivos Algoritmia

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;
```

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

Maestro Detalle

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 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.

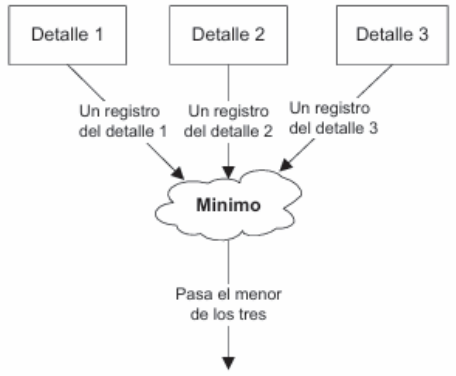
```
procedure leer(var det: detalle; var regD : venta);
begin
    if(not eof(det))then
        read(det, regD)
    else
    regD.codigo := valoralto;
end;
//actualizar Maestro con un archivo ordenado de elementos repetidos
procedure actualizarMaestro(var mae : maestro; var det : detalle);
var
  regD : 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.



1

```
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;
```

2

```
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;
```

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

```
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

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.

- 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.

```
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;
```

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;
```

Baja Fisica

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.
```

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

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

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;
```

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;
```

Compactacion

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 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);
        end;
        leer(mae, aux);
    end;
    Close(mae);
    Writeln('Compactado');
end;
```

Baja Logica y Alta usando Registrs marcados

SIRVE POR 0

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
```

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;
```