Tema 4. Ficheros

1. Noción y definición de fichero
2. Soportes, organización y acceso a ficheros
3. Nombres externos e internos: asociación de un fichero a un programa
4. Operaciones primitivas de acceso a ficheros
5. Operaciones no elementales:
   - 5.1. Creación
   - 5.2. Recorrido
   - 5.3. Búsqueda
6. Ficheros Ordenados
   - 6.1. Introducción
   - 6.2. Operaciones elementales para ficheros ordenados
   - 6.3. Fusión de ficheros ordenados
   - 6.4. Actualización de un fichero ordenado

4.1. Noción y definición de fichero

- Puede existir la necesidad de:
  - almacenar información de forma permanente (p.e. entradas para un programa o salidas de un programa),
  - guardar gran cantidad de información de forma fraccionada (p.e. todos los datos del listín telefónico, organizados por usuarios)
- Los ficheros cubren ambas necesidades

Def. **Fichero**

Colección de informaciones, denominadas registros, todas del mismo tamaño, que **se almacenan en memoria secundaria**

- Aunque los registros suelen ser tipos estructurados no es imprescindible
  - Guardar los datos de una matriz o recuperarlos → se necesitaría un fichero de datos numéricos
  - Guardar los resultados que se muestran por pantalla → se necesitaría un fichero de caracteres / texto

En general tendremos

<decl_fichero> ::= <descriptor> : fichero de <tipo>

Ejemplos:

entrada, salida: fichero de numérico;
texto: fichero de caracteres;
historial : fichero de PACIENTES;

¿Qué diferencias existen entre un fichero de registros de tipo T y un vector de registros de tipo T?

- El número de registros de un fichero no es conocido
- El tamaño del fichero puede ser muy grande
- El fichero se guarda en memoria secundaria y el vector en memoria principal

4.2. Soportes, organización y acceso a ficheros

- ¿Cómo se almacena físicamente la información?
  - soportes secuenciales

    la información se almacena en posiciones consecutivas de memoria, y sólo se puede acceder de forma secuencial (para llegar al registro i, hay que pasar por los i-1 registros anteriores)
  - soportes direccionables

    cada posición del soporte tiene una dirección, que se utiliza para leer o escribir información;

    para acceder a un registro no es necesario acceder a los anteriores, sino que se utiliza su clave para calcular directamente su dirección
- La organización de la información en un fichero puede ser secuencial, secuencial indexada o directa

  Nos centraremos en los ficheros y en los accesos secuenciales.

4.3. Nombres externos e internos: Asociación de un fichero a un programa

- En el sistema de ficheros, cada fichero tiene un nombre único que lo identifica: es su **nombre externo (nombre físico)**

  Ejemplos: datos.txt, resultados.txt, d:\proyectos\memoria.doc, c:_docencia\ficheros.ppt

- Dentro de un programa, se hará referencia al fichero a través de su **nombre interno (nombre lógico)**

- Varios programas o varios algoritmos de un programa pueden usar distintos nombres internos para un mismo nombre externo (fichero con los datos)

- Antes de usar un fichero, es **necesario asociar al nombre interno un nombre externo**

Asociación de un fichero a un programa

¿Cómo se asocian nombres internos/externos?

- **Abrir_f** (<nombre_interno>, <nombre_externo>, <modo>)

  <nombre_interno>: fichero de <tipo>; {parámetro resultado}

  <nombre_externo>: cadena; {parámetro dato}

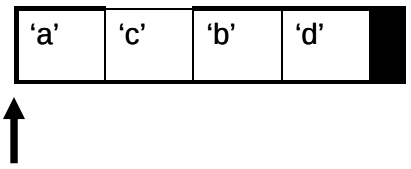
  <modo>: cadena; {par. dato; constante: “l”, “e”, “a”}

  – Se establece una asociación entre los nombres internos/externos
  – El programa pasa a tener el control del fichero
  – El cabezal de L/E se coloca en la posición inicial

  Abrir_f (fich_entrada, “entrada.txt”, “l”);

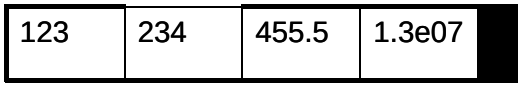
  Abrir_f (fich_salida, “salida.txt”, “a”)

datos.txt

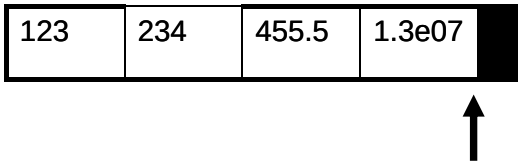


f: fichero de carácter;
Abrir_f (f, "datos.txt", "l");

datosnum.txt



g: fichero de numérico;
Abrir_f (g, "datosnum.txt", "e");



g: fichero de numérico;
Abrir_f (g, "datosnum.txt", "a");

- **Cerrar_f (<nombre_interno>)**

  <nombre_interno>: fichero de <tipo>; {parámetro dato-resultado}

  Se añade una marca de FIN DE FICHERO tras el registro actual (si se está escribiendo).

  Se libera el fichero para que lo puedan usar otros programas.

4.4. Operaciones primitivas de acceso a ficheros

Leer_f (f, V)

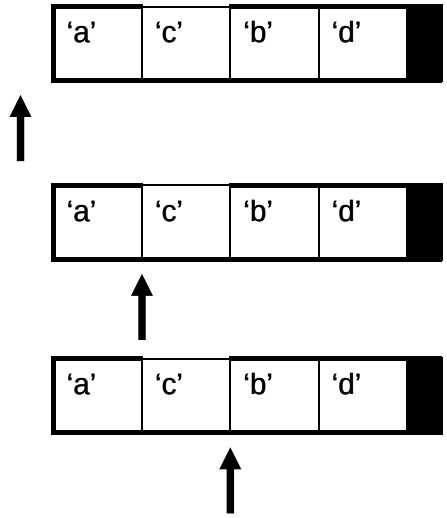
f: fichero de <tipo>; {parámetro dato}

V: <tipo>; {parámetro resultado}

Si hay datos disponibles, tras leer_f(), V contiene el siguiente registro de f. En caso contrario da un error.

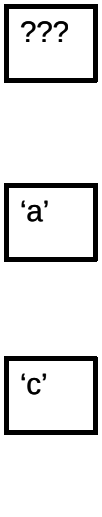
El cabezal de L/E avanza un registro y apunta al siguiente registro que se puede leer.

datos.txt



f: fichero de carácter;
V:carácter;
Inicio
Abrir_f (f, "datos.txt", "l");
Leer_f (f, V);
Leer_f (f, V);
Cerrar_f(f);

V



Operaciones primitivas de acceso a ficheros

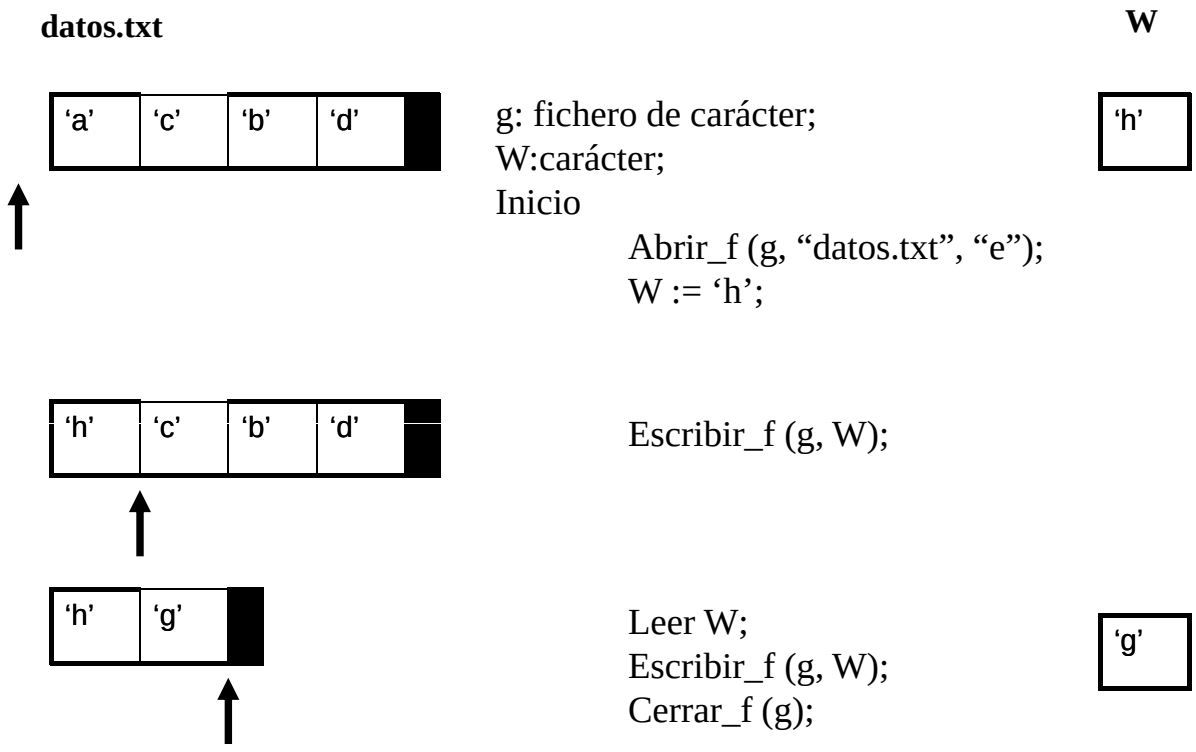
Escribir_f (g, W)

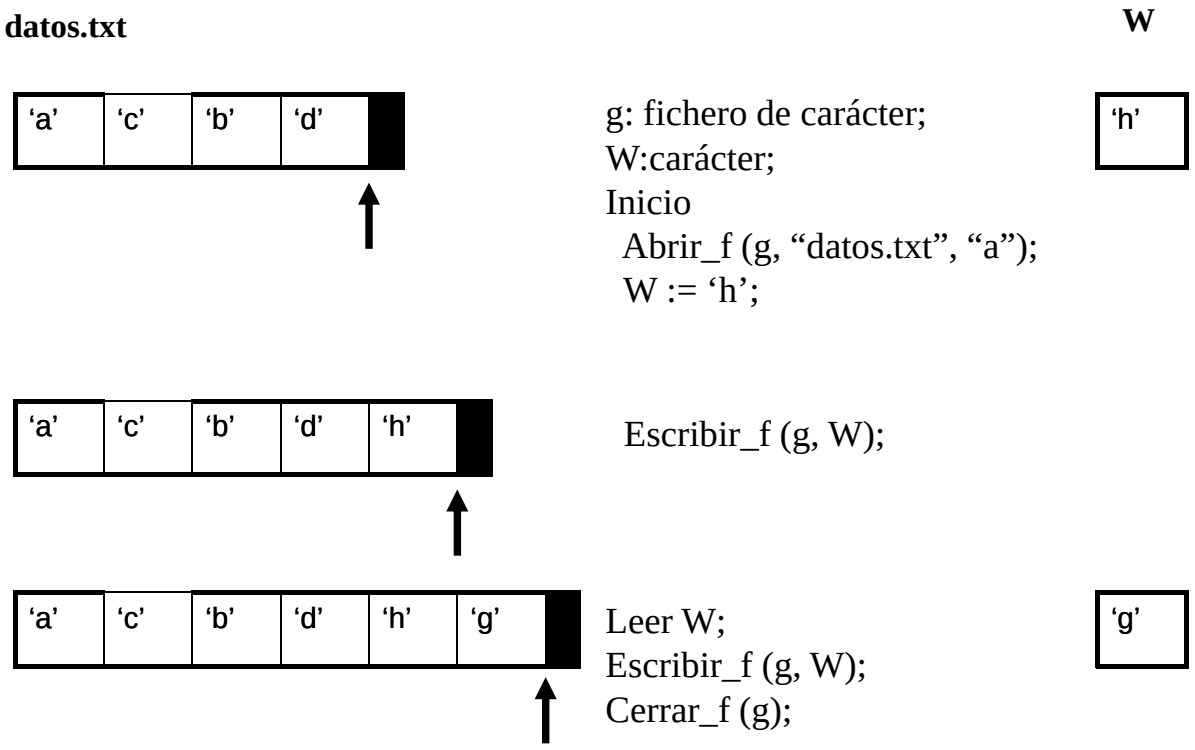
g: fichero de <tipo>; {parámetro dato}

W: <tipo>; {parámetro dato}

Escribe en la posición en la que esté el cabezal de L/E el contenido del registro W, que no se modifica.

El cabezal de L/E avanza un registro y apunta a la siguiente posición donde se pueda escribir.





Operaciones primitivas de acceso a ficheros

- ¿Cómo sabes cuántos registros hay o cuándo se terminan?
- Al terminar de escribir se coloca la marca de FIN DE FICHERO.
- **SIEMPRE** ANTES DE leer_f() **ES OBLIGATORIO** comprobar que no se ha llegado AL FIN DE FICHERO

Fin_f (f, resp)

f: fichero de <tipo>; {parámetro dato}

resp: lógico; {parámetro resultado: cierto si el siguiente registro es la MARCA DE FIN DE FICHERO}

Usar fin_f() no supone que se desplace el cabezal de L/E.

4.5. Operaciones no elementales con ficheros

4.5.1. Creación de un fichero

```
Algoritmo Creación (nombre: cadena) es
F: Fichero de T;
V: T;
hay_más: lógico;
Inicio
  Abrir_f(F, nombre, “e”);
  obtener_datos (V, hay_mas);
  mientras hay_mas = cierto hacer
     escribir_f (F, V);
     obtener_datos (V, hay_mas);
finmientras;
Cerrar_f(F);
Fin
```

4.5.2. Recorrido secuencial en un fichero

```
Algoritmo Recorrido_Secuencial (nombre:cadena) es
F: Fichero de T;
V: T;
resp: lógico;
Inicio
 Abrir_f(F, nombre, “l”);
 Fin_f (F, resp); {Fin de fichero puede ser cierto}
  mientras not resp hacer
       Leer_f (F, V); {Fin de fichero es falso antes de realizar la lectura}
       tratar_registro(V);
       fin_f (F, resp); {Fin de fichero puede ser cierto}
finmientras;
Cerrar_f(F);
Fin
```

4.5.3. Búsqueda en un fichero

```
Algoritmo Búsqueda_Secuencial (nombre, W, pos) es
nombre : cadena ; {p. dato}
W: T; {p. dato}
pos: numérico; {p. resultado}
f: Fichero de T;
V: T;
eof, encontrado: lógico;
Inicio
    Abrir_f(f, nombre, “l”);
    Fin_f (f, eof);
    encontrado := falso;
    pos := 0;
```

Búsqueda en un fichero (cont.)

```
mientras eof = falso y encontrado = falso hacer {Fin fichero es falso}
        Leer_f (f, V);
        iguales (V, W, encontrado);
        si encontrado = falso entonces
                Fin_f (f, eof);
                pos := pos + 1;
        finsi;
finmientras;
si encontrado = cierto entonces
    tratar_encontrado (f, V, W);
sino
    tratar_ausencia (f, W);  {pos := -1;}
finsi;
Cerrar_f(f);
Fin
```

4.6. Ficheros ordenados

4.6.1. Introducción

Existen muchos casos donde los registros de un fichero están almacenados siguiendo un cierto orden.

Fichero ordenado:

Se dice que **un fichero F está ordenado** principalmente por el campo M, en orden secundario 1 por el campo m1, en orden secundario 2 por el campo m2, etc. si:

- los campos M, m1, m2,... son campos de los registros de F,
- los registros aparecen en F según el orden de los valores de M,
- para un mismo valor de M aparecen ordenados por el valor de m1,
- en general, para un mismo valor de m_i los registros aparecen ordenados según los valores del campo m_{i+1}.

En esta asignatura no se darán algoritmos de ordenación de ficheros (lo que se conoce como ordenación externa, para distinguirlo de la ordenación interna que se aplica a los arrays, por ejemplo).

Suponemos que los ficheros ya vienen ordenados.

Si el fichero está ordenado y se supone conocido un valor máximo V_{Max} para el tipo genérico T para cualquier fichero, se pueden simplificar las operaciones de lectura y recorrido de un fichero ordenado. Para ello se define el siguiente algoritmo:

```
Algoritmo Leer_ord (f, v) es
f: fichero de T; {par. dato}
v: T; {par. resultado}
fin: lógico; {var. interna}
Inicio
    Fin_f (f, fin);
    si not fin entonces
            Leer_f (f, v);
    sino
            v.clave := Máx_M; {Valor que toma la clave M > VMax}
    fin si
Fin
```

Si se encuentra fin de fichero, V tomará el valor MAX_M. Por lo tanto, ya no hace falta preocuparse porque **fin o eof** sean ciertos, sino por v.clave = MAX_M.

4.6.2. Operaciones sobre un fichero ordenado

- **Recorrido**

```
Algoritmo Recorrido_fichero_ordenado (nombre) es
nombre: cadena; {par. dato}
f: fichero de T; {var. interna}
v: T; {var. interna}
Inicio
    abrir_f (f, nombre, "l");
    leer_ord (f, v);
    mientras v.clave < MAX_M hacer
            tratar (v);
            leer_ord (f, v);
    fin mientras;
    cerrar_f (f);
Fin
```

- **Búsqueda**

```
Algoritmo Búsqueda_secuencial_ord (fich, x, encontrado) es
Fich: cadena; {parámetro dato, tiene el nombre del fichero}
X: T; {parámetro dato, dato buscado}
Encontrado: lógico;  {parámetro resultado}
F: fichero de T; {var. interna}
V: T; {variable interna}

    Inicio
        Encontrado := falso;
        Abrir_f (F, fich, "l");
        Leer_ord (F, V);
        Mientras V.clave < X.clave {AND V.clave < MAX_M} hacer
                Leer_ord (F, V);
        Fin mientras
        Si V.clave = X.clave entonces
                Encontrado := cierto;
                Tratar_elemento_hallado (X, V);
        Sino
                Encontrado := Falso;
                Tratar_ausencia();
        Finsi
        Cerrar_f(f);
    Fin
```

- **Fusión de dos ficheros ordenados**

  Dados dos ficheros ordenados, obtener un nuevo fichero ordenado que contenga la información de los dos anteriores.

  Ejemplo:

  F1:

3A	6X	10B	12C	EOF

  F2:

4T	5Y	7M	10Z	11K	EOF

  Fichero Resultado:

3A	4T	5Y	6X	7M	10B	10Z	11K	12C	EOF

Principio básico:

Se realizan dos lecturas iniciales de F1 y F2, guardándolas en V1 y V2, respectivamente.
Se escribe en FR el registro cuyo campo clave sea menor.
Se lee el fichero de donde se obtuvo el valor que se ha escrito.
Se repite el proceso anterior con los nuevos valores, hasta que ambos ficheros se hayan leído en su totalidad.

```
Algoritmo Fusión (nombre1, nombre2, nombre_r) es
nombre1, nombre2, nombre_r: cadena; {nombre de los ficheros}
f1, f2, fr: fichero de T;
v1, v2: T;
Inicio
    Abrir_f (f1, nombre1, "l");
    Abrir_f (f2, nombre2, "l");
    Abrir_f (fr, nombre_r, "e");
    Leer_ord (f1, v1);
    Leer_ord (f2, v2);
    mientras v1.clave < máximo o v2.clave < máximo hacer
      Si v1.Clave < v2.Clave entonces
          {v1.clave < máximo si f2 llega a fin de fichero}
          escribir_f (fr, v1);
          Leer_ord (f1, v1);
      sino {v1.Clave>=v2.Clave; v2.Clave < máximo si f1 llega a fin de fichero}
          escribir_f (fr, v2);
          Leer_ord (f2, v2);
      finsi;
    finmientras;  {v1.c = máximo y v2.c=máximo;}
    Cerrar_f(f1);
    Cerrar_f(f2);
    Cerrar_f(fr);
Fin
```

Ejercicio: Plantea un algoritmo de fusión de 2 vectores ordenados.

4.6.4. Actualización de un fichero ordenado

Disponemos de un fichero ordenado, cuyos registros están ordenados por el campo clave:

T = registro de
    Clave: numérica;
    Resto: T1;
Fin registro;

Existen varios ficheros, ordenados por la misma clave, que contienen las actualizaciones a realizar, y que pueden ser: altas, modificaciones o bajas.

TM = registro de
    Mvto: carácter; {puede ser altas, bajas, modificaciones}
    Info: T;
Fin registro;

Para actualizar el fichero original en otro nuevo, hay que recorrer todos en paralelo e ir realizando las actualizaciones que concurran en cada caso.

- **Actualización de un ficheros ordenados, con 2 ficheros de movimientos**

Fichero original

3A	4T	5Y	6X	7M	9B	10Z	11K	12C	EOF

Fichero movimientos #1

A-2X	M-4V	B-10	EOF

Fichero movimientos #2

M-3C	M-4Y	B-4	A-8W	M-11Q	A-13X	EOF

Fichero final

2X	3C	5Y	6X	7M	8W	9B	11Q	12C	13X	EOF

```
{VERSIÓN PARA ACTUALIZAR UN FICHERO CON DOS DE MOVIMIENTOS}
Algoritmo Actualización (antiguo, m1, m2, nuevo) es
Antiguo, m1, m2, nuevo: cadena; {parámetros dato, nombres de ficheros}
Fa, fn: fichero de T; {ficheros con datos antiguos y nuevos}
Fm1, fm2: ficheros de TM; {ficheros que contienen los movimientos}
Va, vn: T; {para leer de los ficheros fa, fn}
Vm1, vm2: TM; {para leer de los ficheros fm1, fm2}
Min: numérico;
Inicio
{Inicializar adquisición + obtener datos}
    Abrir_f (fa, antiguo, "l"); Abrir_f (fn, nuevo, "e");
    Abrir_f (fm1, m1, "l"); Abrir_f (fm2, m2, "l");
{Obtener primer elemento}
    Leer_ord (fa, va);
    Leer_ord (fm1, vm1);
    Leer_ord (fm2, vm2);
    minimo (va, vm1, vm2, min); {min determina la próxima clave a tratar}
    mientras min < MAX hacer {clave < MAX}
        {va >= min, vm1 >= min, vm2 >= min}
        tratar_claves_iguales (min, va, vn, vm1, vm2, fa, fn, fm1, fm2);
        {va > min, vm1 > min, vm2 > min}
        {obtener elemento siguiente se realiza dentro de tratar_claves_iguales}
        minimo (va, vm1, vm2, min);
    fin mientras;
    Cerrar_F (fa);  Cerrar_F (fn); Cerrar_f (fm1); Cerrar_f (fm2);
Fin
```

```
{Quiénes forman parte de un bloque:
        0 ó 1 registros de fa + 0 ó n registros de fm1 + 0 ó m registros de fm2}
{Objetivo: Crear un registro vn a partir de la información:
        va, vm11, vm12,...,vm1n, vm21,mv22,...vm2m}

Algoritmo tratar_claves_iguales (min, va, vn, vm1, vm2, fa, fn, fm1, fm2) es
Min: numérico;
va, vn: T; {para leer de los ficheros fa, fn}
vm1, vm2: TM; {para leer de los ficheros fm1, fm2}
fa, fn: fichero de T; {ficheros con datos antiguos y nuevos}
fm1, fm2: ficheros de TM; {ficheros que contienen los movimientos}
escritura: lógico; {existe información para escribir en Vn}
Inicio
{inicialmente no hay datos que escribir}
    escritura := falso;
    {va.clave >= min. Es igual cuando va es el mínimo}
    si min = va.clave entonces {existe un registro que se va a modificar}
        escritura := cierto; {ya se puede escribir vn}
        copiar (vn, va);
        {pido elemento sgte, ya que sólo puede haber una clave va.clave = min}
        leer_ord (fa, va);
    finsi
    {va.clave > min}
```

```
    {vm1 >= min porque están ordenados de forma creciente;
    vm1 = mínimo cuando vm1 sea la clave a tratar}
    mientras vm1.info.clave = min hacer
        {vm1 modifica a vn}
        {tratar elemento}
        tratar_una_clave (vn, vm1, escritura);
        {obtener elemento siguiente}
        leer_ord (fm1, vm1);
    finmientras;
    {vm1 > min}

    {vm2 >= min porque están ordenados de forma creciente;
    vm2 = mínimo cuando vm2 sea la clave a tratar}
    mientras vm2.info.clave = min hacer {vm2 modifica a vn}
        tratar_una_clave (vn, vm2, escritura); {tratar elemento}
        leer_ord (fm2, vm2); {obtener elemento siguiente}
    finmientras;
    {vm2 > min}

    {Esto podría repetirse para cuantos ficheros de movimientos sean necesarios}

    si escritura = cierto entonces
        escribir_f (fn, vn);
    finsi
    {va.clave > min, vm1.info.clave > min, vm2.info.clave > min}
Fin
```

```
Algoritmo tratar_una_clave (vn, vm, escritura) es
Vn: T; {parámetro dato-resultado}
Vm: TM; {parámetro dato}
Escritura: lógico;
Inicio
{para modificar el registro vn hay que mirar el valor de vm.mvto y el valor de escritura}
    según vm.mvto hacer
        'A': {Alta → válida si no existía}
        si escritura = cierto entonces
            Error ("Ya existía y se da de alta");
        sino
            copia (vn, vm.info);
            escritura := cierto;
        fin si;
        'M': {modificación → posible si ya existe}
        si escritura = falso entonces
            Error ("No existía y se modifica");
        sino
            modifica (vn, vm.info);
        fin si;
        'B': {baja → posible si ya existía}
        si escritura = falso entonces
            Error ("No existía y se quiere borrar");
        sino
            borrar (vn, vm.info);
            escritura := falso;
        fin si;
    fin según
Fin
```