

ESTRUCTURA DE DATOS: ARCHIVOS



Repaso: Archivos. Objetivos. Concepto General. Operaciones. Organización. Acceso.

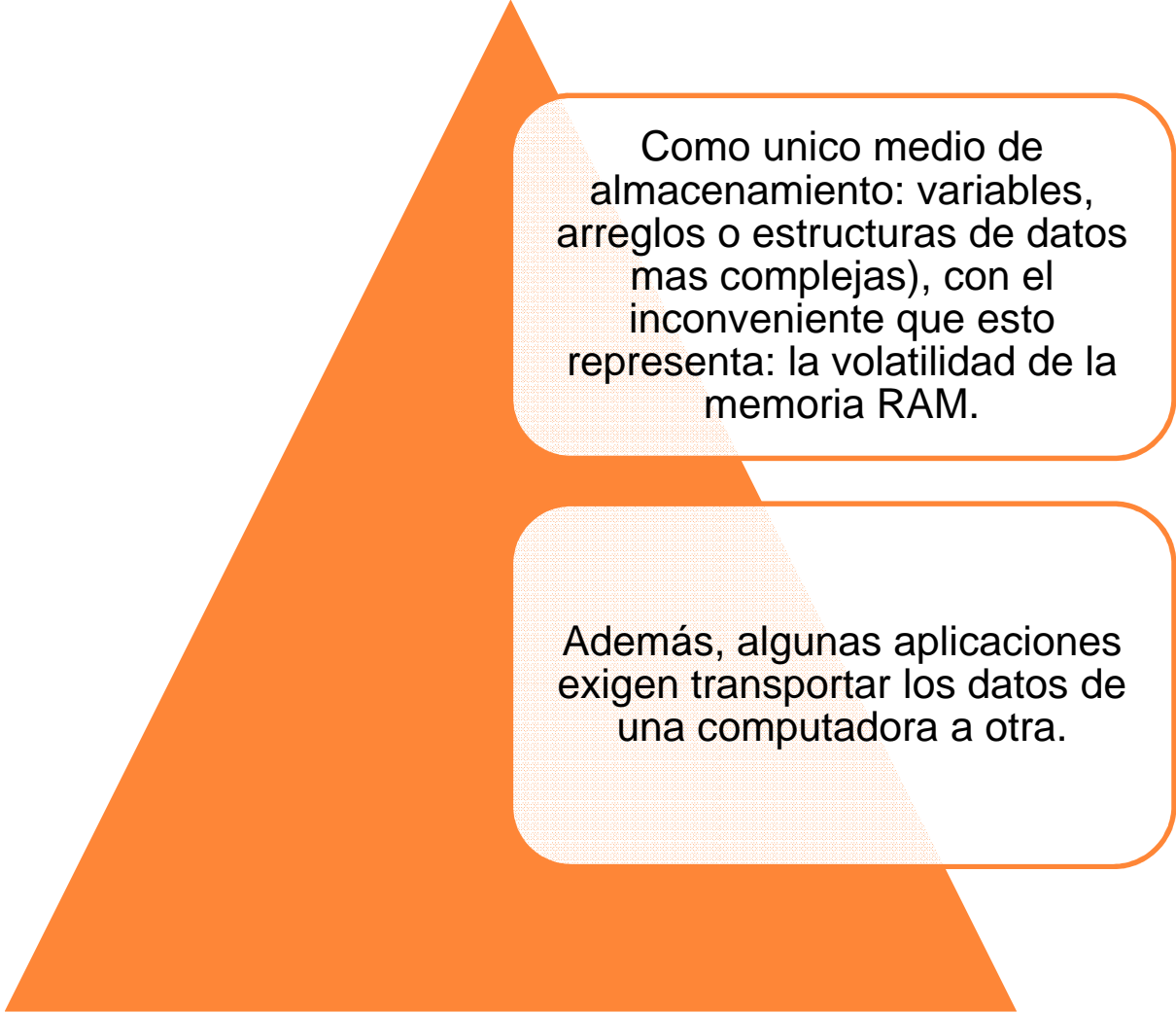
- **Tema IV: Archivos. Operaciones más complejas** Gestión de Archivos: ABM y Consultas en archivos de organización secuencial y directa. Corte de control. Unión de archivos.



PRIMERA PARTE




¿CÓMO SURGE LA NECESIDAD DE UTILIZAR ARCHIVOS?



Como unico medio de almacenamiento: variables, arreglos o estructuras de datos mas complejas), con el inconveniente que esto representa: la volatilidad de la memoria RAM.

Además, algunas aplicaciones exigen transportar los datos de una computadora a otra.



RELACIÓN ENTRE LA MEMORIA PRINCIPAL, EL MICROPROCESADOR Y DISPOSITIVOS DE ALMACENAMIENTO SECUNDARIO



Existe una estrecha relación entre la memoria principal, el microprocesador y los dispositivos de almacenamiento secundario ya que el procesamiento que realiza una computadora es tarea absoluta del microprocesador en conjunción con la memoria principal.



Los dispositivos de almacenamiento secundario **no procesan datos, sólo los almacenan.**



En estos dispositivos sólo se reflejan los datos previamente procesados y funcionan exclusivamente como una bodega.



Esto repercute de manera significativa al momento de programar archivos, ya que para hacerle modificaciones a los datos de un registro previamente almacenado es necesario primero **“cargarlo” en la memoria principal, es decir, localizar el registro en el archivo** y leerlo para colocar sus datos en la memoria RAM, ahí modificarlo y posteriormente **grabarlo en la misma posición en la que se encontraba, sin embargo** estas operaciones no se realizan directamente, sino a través de la unidad aritmética-lógica, la unidad de control y los registros del microprocesador.

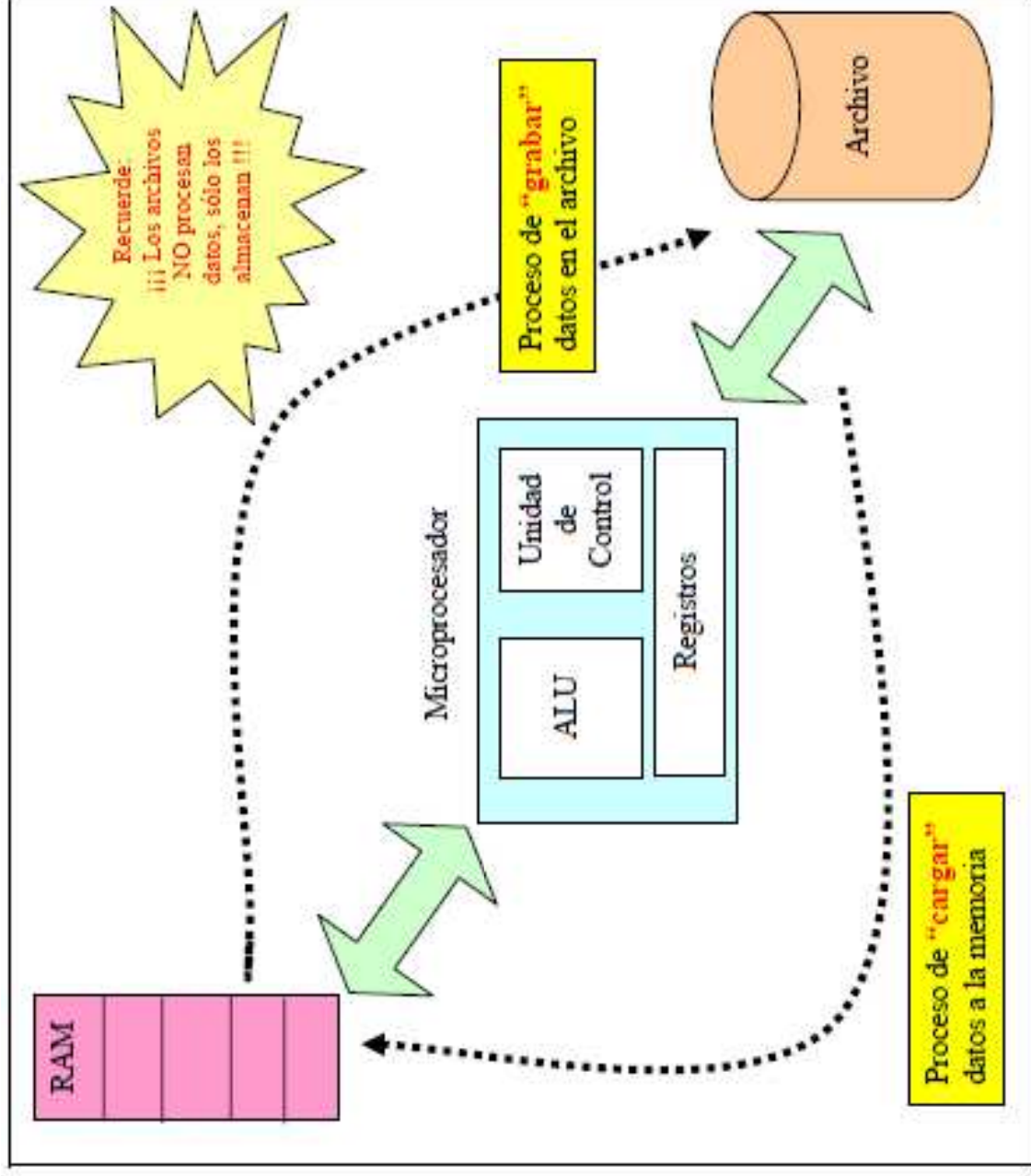


Fig. 1. Interacción entre la memoria principal, el microprocesador y los archivos

DEFINICIONES DE DATOS, REGISTROS Y ARCHIVOS

Datos: Básicamente se refieren a los testimonios individuales relacionados con hechos, ya sean características de ciertos objetos de estudio o condiciones particulares de situaciones dadas. Los elementos individuales de los archivos se llaman datos o campos.

Registro: Es el conjunto completo de datos relacionados pertenecientes a una entrada. P. ejem. Un almacén puede retener los datos de sus productos en registros.

No_prod	Descrip	Cantidad	Precio	Garantia
Entero	Cadena [30]	Entero	Real	Caracter

Fig. 2. Formato del registro de Productos

```
Type Tipo_registro = Record
  no_prod : integer;
  descrip : string[30];
  cantidad: integer;
  precio  : real;
  garantia: char;
End;

Var Registro : Tipo_registro;
```

Declaración del tipo de dato (en este caso del tipo de registro)

Declaración de la variable "Registro" de tipo "Tipo_registro"

Fig. 3. Declaración del registro de Productos

ARCHIVO

En términos computacionales es una colección de datos que tiene un nombre y se guardan en dispositivos de almacenamiento secundario, los cuales pueden ser magnéticos, ópticos, electrónicos, etc. (Ej.: Diskettes, discos duros, CD´s, ZIP drives, flash drives, memory sticks, etc.)

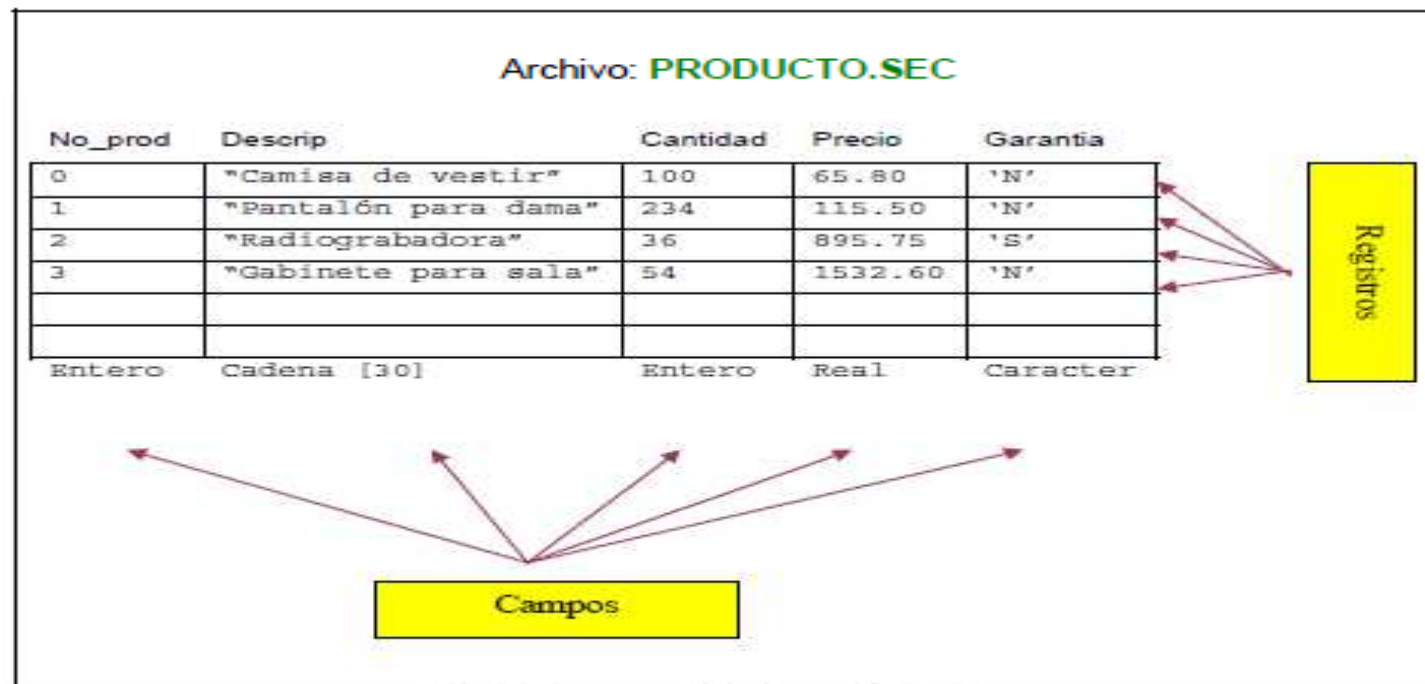



Fig. 4. Formato del registro de Productos

MANEJO DE BUFFERS

Memoria intermedia entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados definitivamente en memoria secundaria o donde los datos residen una vez recuperados de dicha memoria secundaria. Ocupan memoria RAM.



El SO es el encargado de manipular los buffers.



Diferentes tiempos de acceso en la memoria principal y memoria secundaria.

TIPOS DE REGISTROS

Registro Físico: Cantidad de datos que puede transferirse en una operación de I / O a través del buffer.

Registro Lógico: Definido por el programador.

Factor de Bloqueo: Numero de Registros Lógicos que puede contener un Registro Físico.

Campo	Tipo de dato	Tamaño en bytes
no_prod	Integer	2
Descrip	Char [30]	30
Cantidad	Integer	2
Precio	Real	4
Garantia	Char	1
TOTAL		39

Fig. 10. Ejemplo de cálculo del espacio ocupado por un registro

2.1. Características de los archivos

Las principales características que diferencian esta estructura de datos de las restantes son las siguientes:

- **Residencia en soportes de información externos**, también denominados memorias secundarias, masivas o auxiliares, como son las cintas, discos, CD, DVD.
- **Independencia respecto de los programas**. Significa que la vida del archivo no está limitada por la vida del programa que lo creó, y también que el archivo puede ser utilizado por diferentes programas.
- **Permanencia de la información almacenada**. Es decir, toda la información almacenada en la memoria central desaparece cuando se termina la ejecución del programa que la maneja, pero para hacer desaparecer un archivo será necesario realizar explícitamente una operación de borrado.
- **Gran capacidad de almacenamiento**. Teóricamente esta capacidad es ilimitada, está en función del soporte de almacenamiento. Por el contrario, las estructuras de datos que residen en la memoria central tienen limitado su tamaño por la capacidad de ésta.



Clasificación de los archivos según su uso

Archivos permanentes: Contienen información que varía poco a lo largo del tiempo:

- Archivos constantes: Contiene datos fijos para la aplicación. Su información permanece prácticamente inamovible, utilizándose principalmente como archivos de consulta. Un ejemplo puede ser el de la red del metro de una ciudad, que contiene la descripción, número de estaciones, número de trenes, etc., de cada línea.
- Archivos de situación: También denominados archivos **maestros**, contienen la información que refleja el estado o situación de una empresa o entidad, o algún aspecto de ella en un determinado momento. Estos archivos se actualizan periódicamente para adaptarlos a cada nueva situación. Un ejemplo es el archivo de personal en una empresa, o también el archivo de existencias en almacén.
- Archivos históricos: Se obtienen de los anteriores cuando se dejan fuera de uso para futuros estudios estadísticos o consultas. Será un archivo histórico el que contiene la información de libros adquiridos por una biblioteca en la década de los ochenta o un archivo con los pagos realizados en los últimos 5 años ó el detalle de comprobantes considerados en un ejercicio contable ya cerrado.

Archivos de movimientos. En ellos se almacena la información que se utilizará para actualizar los archivos maestros. Sus registros, denominados movimientos o transacciones, pueden ser de tres tipos: *altas, bajas y modificaciones*.

Una vez realizado el proceso de actualización de un archivo maestro por medio de un archivo de movimientos, éste pierde su validez y no se necesita conservarlo.

Un archivo de este tipo para actualizar, por ejemplo, el archivo de personal de una empresa, es el que refleja las nuevas incorporaciones, finalizaciones de la relación laboral y modificaciones de los mismos producidas en la empresa durante el mes actual.

Archivos de trabajo. Tienen una vida limitada, normalmente igual a la duración de la ejecución de un programa, y se utilizan como auxiliar de los anteriores. Por ejemplo, si se desea una lista alfabética del personal contratado, se hará por medio de un archivo de trabajo en el que se almacene esta información a partir del archivo de personal. Este archivo es temporal, desaparecerá una vez que se tenga la lista impresa.

ESTRUCTURA DE DATOS: ARCHIVOS

Soporte: A) Secuenciales; B) Direccionables;

Un soporte secuencial → Org. secuencial

Un soporte direccionable → Distintos tipos de Org.

Organización de Archivos: La organización de un archivo define la forma en la que los registros se disponen sobre el soporte de almacenamiento, o también se define la organización como la forma en que se estructuran los datos en un archivo. En general, se consideran tres organizaciones fundamentales:

- **Organización secuencial**
- *Organización directa o aleatoria (random)*
- *Organización secuencial indexada*



ESTRUCTURA DE DATOS: ARCHIVOS

Organización secuencial

No es mas que una sucesión de registros almacenados en forma consecutiva sobre un soporte externo.

Los registros están ubicados físicamente en una secuencia usualmente fijada por uno o más campos de control contenidos dentro de cada registro, en forma ascendente o descendente.

Esta organización tiene el último registro en particular, contiene una marca (en general un asterisco) de fin de archivo, la cual se detecta con funciones tipo EOF (end of file) o FDA (Fin de Archivo).



ORGANIZACIÓN DIRECTA

Los datos se colocan y se acceden aleatoriamente mediante su posición, es decir, indicando el lugar relativo que ocupan dentro del conjunto de posiciones posibles.

En esta organización se pueden leer y escribir registros, en cualquier orden y en cualquier lugar.

Inconvenientes:

- a) Establecer la relación entre la posición que ocupa un registro y su contenido;
- b) Puede desaprovecharse parte del espacio destinado al archivo.

Ventaja:

- a) Rapidez de acceso a un registro cualquiera.



ORGANIZACIÓN INDEXADA

Un archivo con esta organización consta de tres áreas:

- Área de índices
- Área primaria
- Área de excedentes (*overflow*)

Ventaja:

- Rápido acceso, y, además, el sistema se encarga de relacionar la posición de cada registro con su contenido por medio del área de índices.
- Gestiona las áreas de índices y excedentes.

Desventajas:

- necesidad de espacio adicional para el área de índices.
- el desaprovechamiento de espacio que resulta al quedar huecos intermedios libres después de sucesivas actualizaciones.

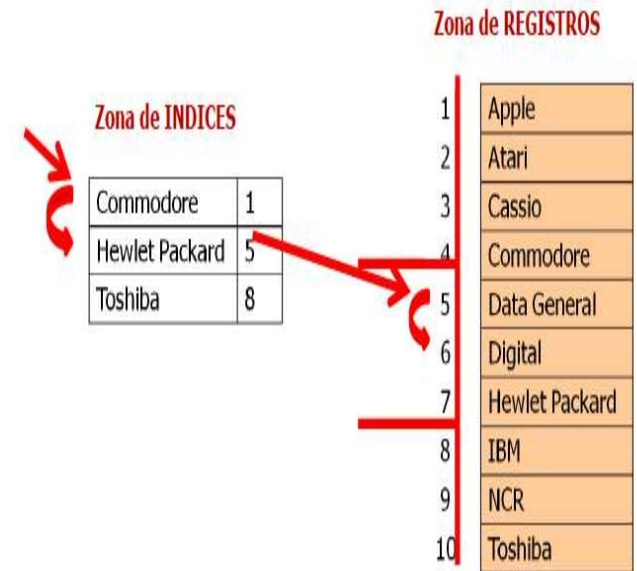


Fig. 7.8. Esquema de acceso secuencial-indexado

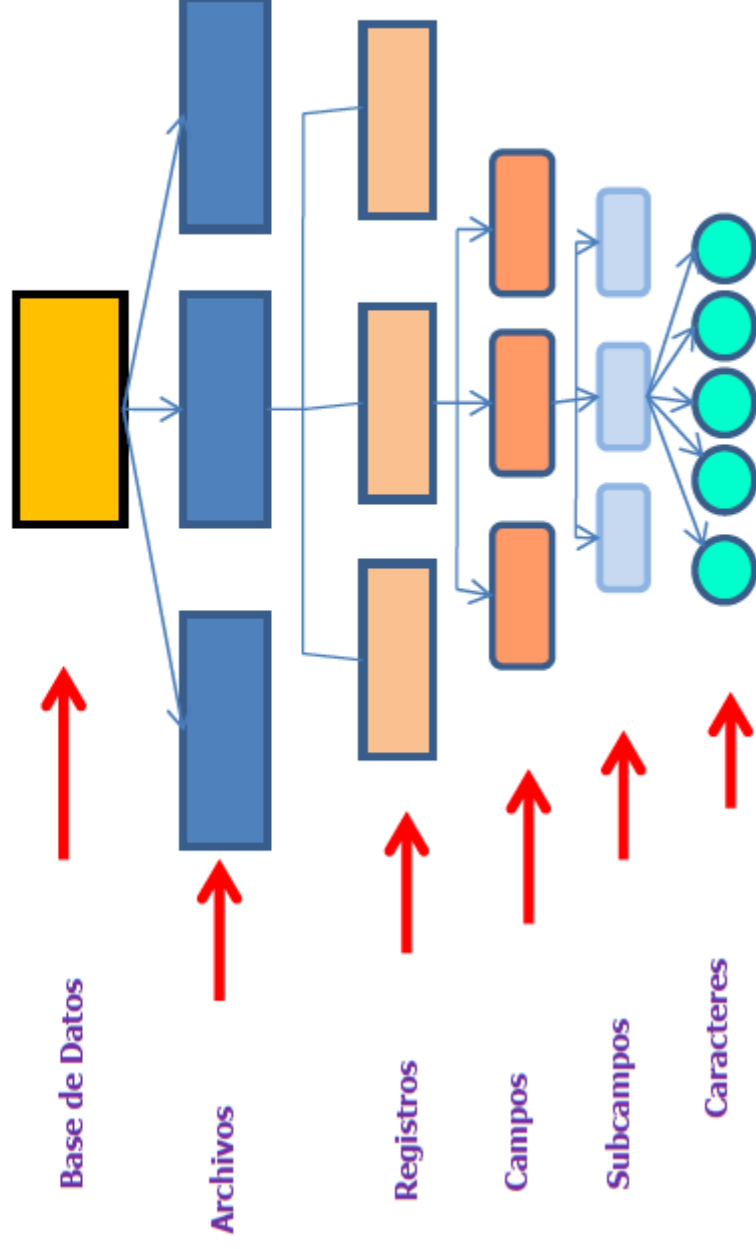
Modos de acceso

Se denomina **modo de acceso** a la forma en que el dispositivo que maneja el soporte se posiciona en un determinado lugar del mismo para realizar una operación de lectura o escritura de un registro. El modo de acceso lo decide el programador de la aplicación en función del soporte utilizado y del tipo de organización.

Hay dos modos básicos: **secuencial** y **directo**.



Bases de Datos



Tipos de archivos

- **De texto:** los datos están guardados en cadenas de texto, el acceso es secuencial
- **Binarios:** los datos están guardados en bits, el acceso es directo

20/02/12

Programación avanzada y métodos numéricos

8

Lenguaje C

- Abrir un archivo
- Cerrar un archivo
- Leer datos de un archivo
- Escribir datos en un archivo
- Añadir datos al final de un archivo

Un archivo de texto es una secuencia de caracteres organizados en líneas que terminan con un carácter de nueva línea (`/n`). Los archivos de texto se caracterizan por ser “planos”, es decir, contienen solo texto (caracteres) sin información sobre el tipo de letra, ni formato (negrita, subrayados) ni tamaño. Técnicamente cualquier archivo puede abrirse como texto plano desde un editor de texto, por ejemplo, el Bloc de notas de Windows. Por costumbre, los archivos de texto llevan la extensión `.txt`. Se utilizan también para almacenar código fuente en programación, archivos de configuración, etc.

Un archivo binario almacena la información tal cual la representa en forma interna, es una secuencia de bytes que tiene una correspondencia uno a uno con un dispositivo externo. La información contenida en el archivo no es visible directamente como los archivos `.txt`.

Un fichero en C es sencillamente una colección de bytes que lleva asociado un nombre. El sistema operativo de la computadora administra los archivos por su nombre. Las operaciones que se pueden realizar sobre un archivo son:

Nombre	Función
<code>fopen()</code>	Abre un archivo
<code>fclose()</code>	Cierra un archivo
<code>fgets()</code>	Lee una cadena de un archivo
<code>fputs()</code>	Escribe una cadena en un archivo
<code>fseek()</code>	Busca un byte específico de un archivo
<code>fprintf()</code>	Escribe una salida con formato en el archivo
<code>fscanf()</code>	Lee una entrada con formato desde el archivo
<code>fread()</code>	Lee un bloque de datos en archivos binarios
<code>fwrite()</code>	Escribe un bloque de datos en archivos binarios
<code>feof()</code>	Devuelve cierto si es final del archivo
<code>ferror()</code>	Devuelve cierto si se produce un error
<code>fflush()</code>	Vacía un archivo

El puntero a un archivo.

El puntero a un archivo es el hilo común que unifica el sistema de E/S con el buffer. Un puntero a un archivo es un puntero a una información que define varias cosas sobre él, incluyendo el nombre, el estado y la posición actual del archivo. En esencia identifica un archivo específico y utiliza la secuencia asociada para dirigir el funcionamiento de las funciones de E/S con buffer. Un puntero a un archivo es una variable de tipo puntero al tipo FILE que se define en stdio.h. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos. Para obtener una variable de este tipo se utiliza una secuencia como esta:

FILE *f;

Apertura de un archivo

La función **fopen()** abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su prototipo es:

```
FILE *fopen(constcharnombre_archivo, constcharmodo);
```

Donde *nombre_archivo* es un puntero a una cadena de caracteres que representa un nombre valido del archivo y puede incluir una especificación del directorio. La cadena a la que apunta *modo* determina como se abre el archivo. La siguiente tabla muestra los valores permitidos para *modo*.



Modo	Significado
r	Abre un archivo de texto para lectura
w	Crea un archivo de texto para escritura
a	Abre un archivo de texto para añadir
rb	Abre un archivo binario para lectura
wb	Crea un archivo binario para escritura
ab	Abre un archivo binario para añadir

La función **fopen()** devuelve un puntero a archivo. Si se produce un error cuando se está intentando abrir un archivo, **fopen()** devuelve un puntero nulo. La macro **NULL** está definida en **stdio.h**. Este método detecta cualquier error al abrir un archivo: como por ejemplo disco lleno o protegido contra escritura antes de comenzar a escribir en él. Ejemplo:

```
if (architext==NULL {  
    Printf("No existe el fichero");  
    Exit(EXIT-FAILURE);  
}
```

Si se usa **fopen()** para abrir un archivo para escritura, entonces cualquier archivo existente con el mismo nombre se borrará y se crea uno nuevo. Si no existe un archivo con el mismo nombre, entonces se creará. Si se quiere añadir al final del archivo entonces debe usar el modo "a". Si se usa "a" y no existe el archivo, se devolverá un error. La apertura de un archivo para las operaciones de lectura requiere que exista el archivo. Si no existe, **fopen()** devolverá un error.



Modos de abrir un archivo

"r"	Abre en modo lectura un archivo
"w"	Crea archivo para escribir o lo sobrescribe si ya existe
"a"	Abre para añadir datos al final del archivo
"r+"	Abre para leer y escribir un archivo, manteniendo su contenido
"w+"	Crea archivo para escribir y leer o lo sobrescribe
"a+"	Abre para escribir y leer al final del archivo
"b"	Al agregar b, se abre el archivo en modo binario ("rb", "ab", "w+b", "a+b")

Cierre de un archivo.

La función **fclose()** cierra una secuencia que fue abierta mediante una llamada a **fopen()**. Escribe toda la información que todavía se encuentra en el buffer del disco y realiza un cierre formal del archivo a nivel del sistema operativo.

Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa.

El prototipo de esta función es:

```
int fclose(FILE *F) ;
```

Donde F es el puntero al archivo devuelto por la llamada a **fopen()**. Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.



Leer y grabar datos en un archivo

Para leer o grabar datos de un archivo tenemos las siguientes cuatro funciones:

fgets() y **fputs()**

Estas funciones pueden leer y escribir cadenas a o desde los archivos.

Los prototipos de estas funciones son:

```
char *fputs(char *str, FILE *F);  
char *fgets(char *str, int long, FILE *F);
```

La función **fputs()** escribe la cadena a un archivo específico. La función **fgets()** lee una cadena desde el archivo especificado hasta que lee un carácter de nueva línea o longitud-1 caracteres. Si se produce un EOF (End of File) la función gets retorna un NULL.

fscanf() y **fprintf()**

Estas funciones realizan la lectura y escritura de variables escalares (es decir tipos de datos que no son compuestos) y se comportan exactamente como **printf()** y **scanf()** usadas anteriormente, excepto que operan sobre archivo.

Sus prototipos son:

```
int fprintf (FILE *F, constchar *cadena_de_control, .....);  
int fscanf (FILE *F, constchar *cadena_de_control, .....);
```

Donde F es un puntero al archivo devuelto por una llamada a **fopen()**. **fprintf()** y **fscanf()** dirigen sus operaciones de E/S al archivo al que apunta F.

Función feof()

La función **feof()** se utiliza para detectar cuando no quedan más elementos en un archivo, devolviendo un valor distinto de cero en este caso. El prototipo de la función es:

int feof (FILE *F);

Devuelve verdadero si detecta fin de archivo, en cualquier otro caso, devuelve 0. Se aplica de la misma forma a archivos de texto y binarios. Tener en cuenta que esta función no indica fin de fichero hasta que no se intenta volver a leer después de haber leído el último elemento del archivo. Por lo tanto, en el programa debe hacerse la lectura siguiendo estos pasos:

```
Leer un dato del fichero
mientras (!feof(archivo))
    Procesar el dato leído
    Leer un dato del archivo
fin-mientras
```



ferror()

La función **ferror()** determina si se ha producido un error en una operación sobre un archivo. Su prototipo es:

```
int ferror(FILE *F);
```

donde F es un puntero a un archivo válido. Devuelve cierto si se ha producido un error durante la última operación sobre el archivo. En caso contrario, devuelve falso. Debido a que cada operación sobre el archivo actualiza la condición de error, se debe llamar a **ferror()** inmediatamente después de la operación de este tipo; si no se hace así, el error puede perderse.

fflush()

La función escribe todos los datos almacenados en el buffer sobre el archivo asociado con un apuntador. Su prototipo es:

```
int fflush(FILE *F);
```

Si se llama esta función con un puntero nulo se vacían los buffers de todos los archivos abiertos. Esta función devuelve cero si tiene éxito, en otro caso, devuelve EOF.



Tipo de dato registro: struct

Tal como se mencionó en el tema 4, los datos se clasifican en simples y compuestos. Dentro de estos últimos se encuentran los tipos de datos cadenas, registros y arreglos. Los registros están vinculados a los archivos binarios, dado que permiten el procesamiento de datos estructurados.

Una estructura que permite almacenar diferentes tipos de datos bajo una misma variable se denomina **registro**. Un registro es un "contenedor" de datos de diferentes tipos. Un registro se declara con la palabra reservada **struct**, cuyo formato es el siguiente:

```
struct datos {  
    tipo1 dato1;  
    tipo2 dato2;  
    /* ...otros campos del registro */  
} variable1, variable2;
```

Ejemplo: Supongamos que se desea almacenar en una estructura los datos de mediciones de temperaturas, por ejemplo, hora, minuto y valor de la medición en grados. Para ello se define la siguiente estructura:

```
struct medida {  
    char hora;  
    char minuto;  
    float temperatura;  
} registro
```

La declaración typedef

En C la declaración **typedef** permite crear nuevos nombres de tipos de datos. Por ejemplo:

```
typedef struct reg {  
    char letra;  
    int numero;  
} REGISTRO;  
  
REGISTRO nuevo, actual [10];
```

En este caso, REGISTRO no es un nombre de variable de tipo reg sino, es un nuevo nombre de tipo de dato que puede utilizarse para definir nuevas variables. En el ejemplo una variable *nuevo* y un vector *actual* de 10 elementos.



Ejemplo de programa que lee un archivo de texto

```
#include<stdio.h>
main() {
    FILE *filetext;    /* Define la variable "filetext" como puntero a FILE */
    char lineaTexto;    /* Define la línea del archivo de tipo char */

    filetext = fopen("poema20.txt","r"); /*Abre el archivo en modo lectura */

    printf("Contenidos del archivo:\n" );
    printf("-----:\n" );
    lineaTexto = fgetc(filetext);    /*Lee una línea del archivo de texto */
    while (feof(filetext) == 0) {    /* mientras no sea fin de archivo*/
        printf( "%c",lineaTexto);    /* muestra en pantalla la línea */
        lineaTexto = fgetc(filetext); /* lee otra línea del archivo */
    }
    fclose(filetext); /*Cierra el archivo de texto */
    return 0;
}
```



Ejemplo de programa que graba un archivo de texto

```
#include <stdio.h>
int main() {
    FILE* fichero;
    fichero = fopen("prueba.txt", "w");
    fputs("Esta es una línea\n", fichero);
    fputs("Esta es otra línea", fichero);
    fputs("y esta es continuación de la anterior\n", fichero);
    fclose(fichero);
    return 0;
}
```



Ejemplo de uso de registro:

```
#include<stdio.h>
main(){
    struct alumno {      /* Define el registro alumnos */
        int dni;
        float parcial1;
        float parcial2;
        float promedio;
    } regalumno;
    /* completa datos del registro */
    printf("dni alumno:\n");
    scanf("%i", &(regalumno.dni));
    printf("nota parcial 1:\n");
    scanf("%f", &(regalumno.parcial1));
    printf("nota parcial 2:\n");
    scanf("%f", &(regalumno.parcial2));
    printf("d:\n");
    /* Calcula el promedio de notas */
    regalumno.promedio=(regalumno.parcial1+regalumno.parcial2)/2;
    printf("\n Promedio %i: es %f", regalumno.dni,
    regalumno.promedio);
    return 0;
}
```



ALTAS SECUENCIALES

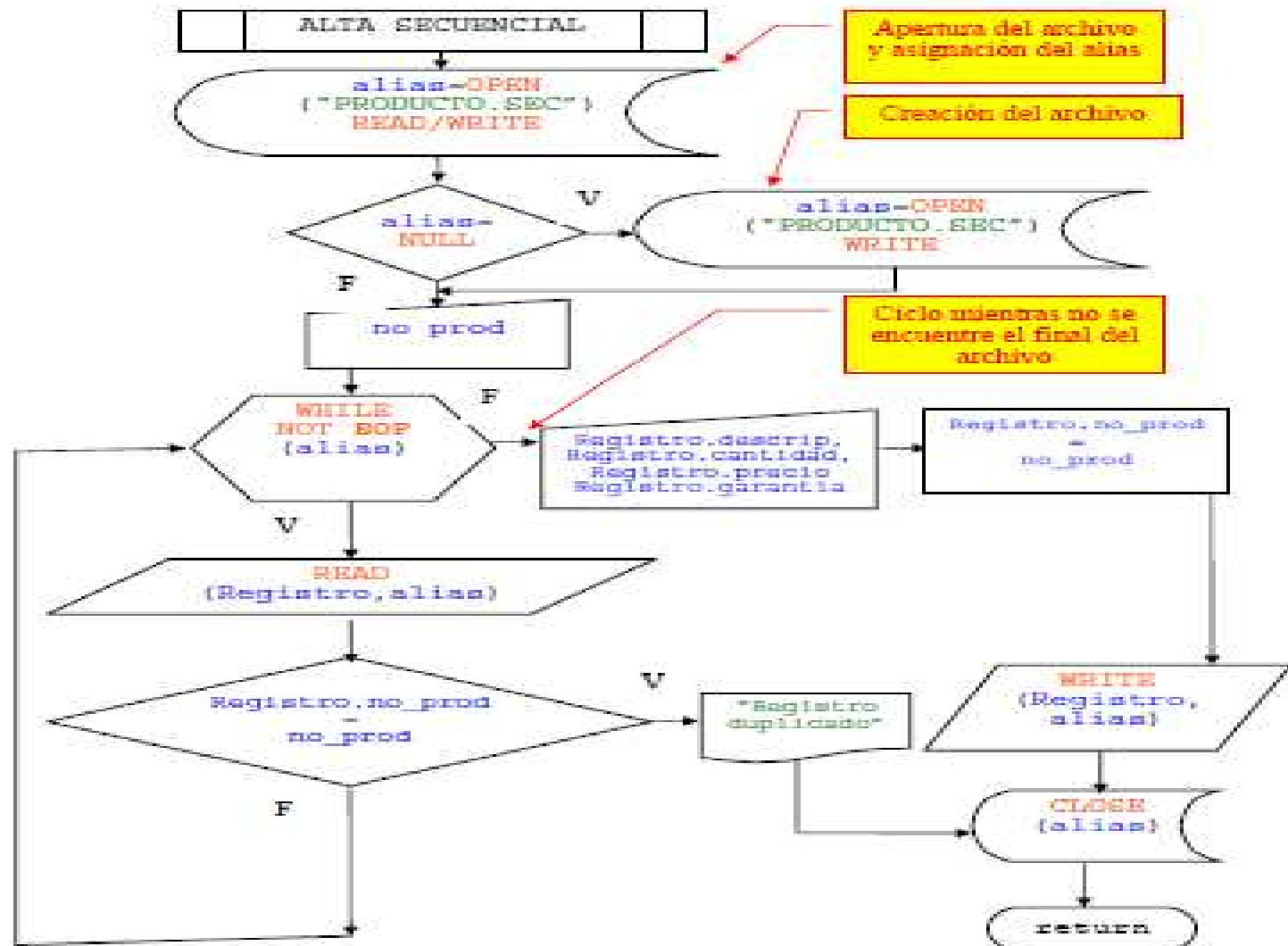


Fig. 13. Diagrama de flujo de rutina de alta secuencial

LISTADO SECUENCIAL

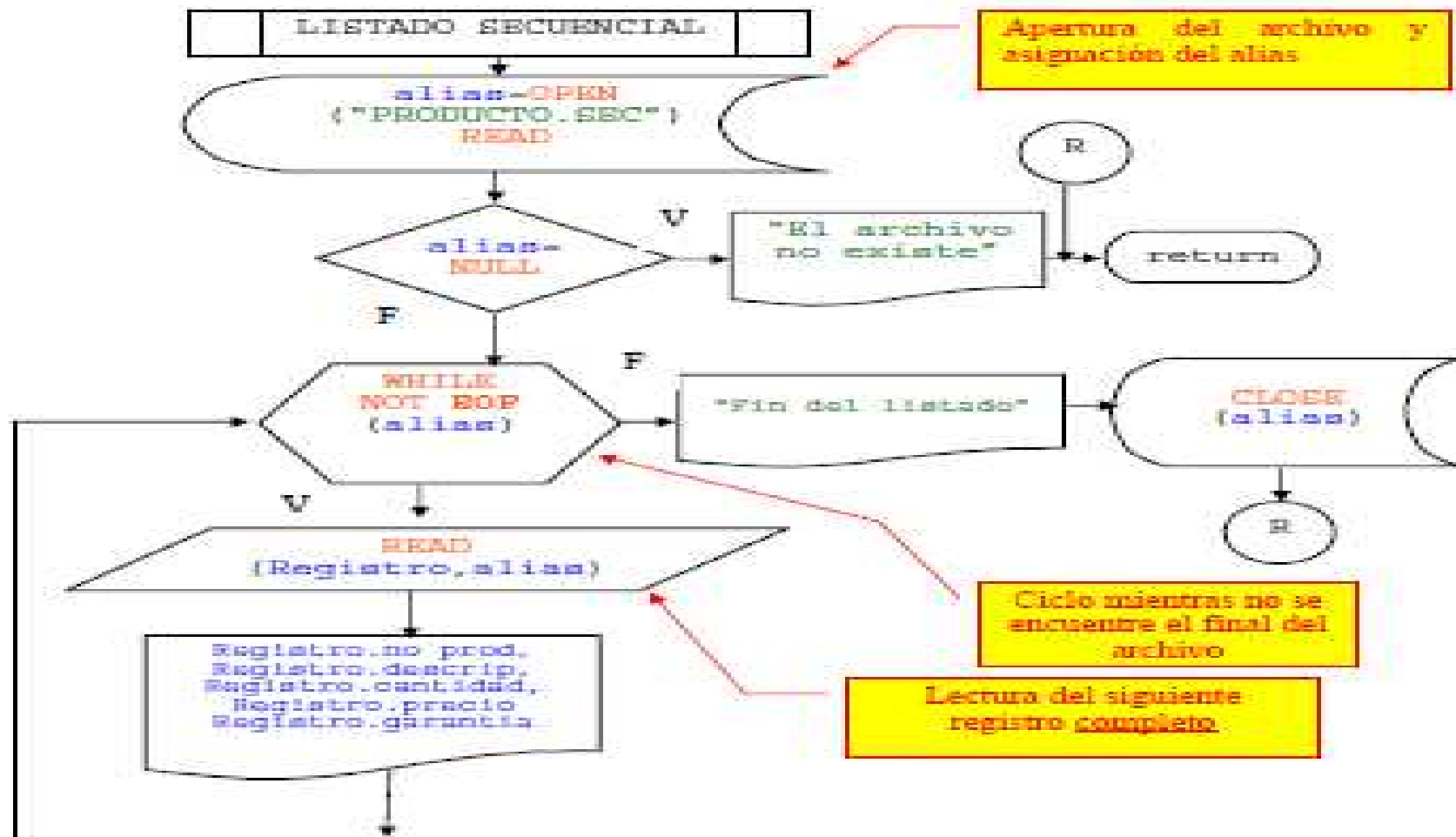
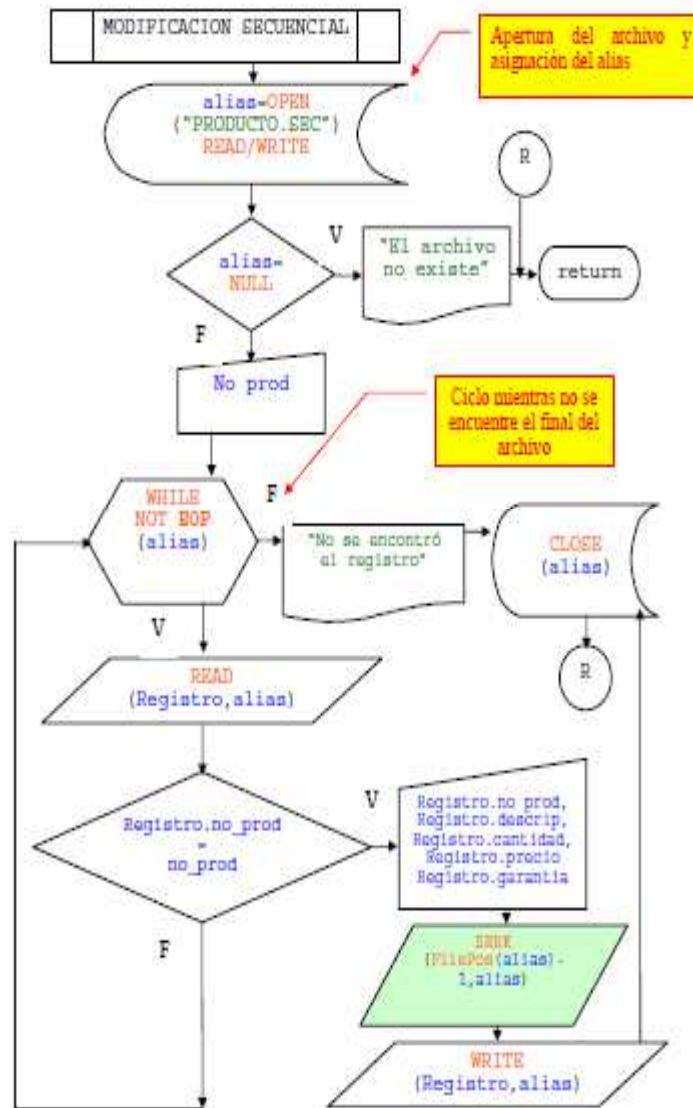
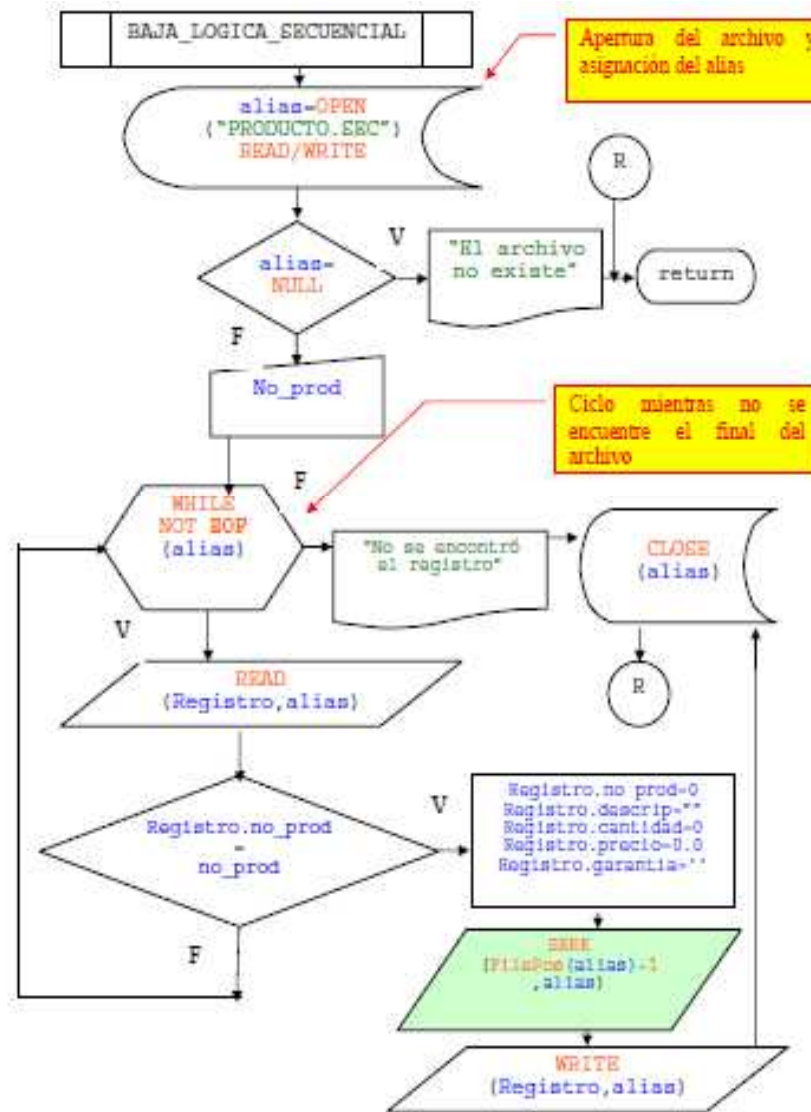


Fig. 26. Diagrama de flujo de rutina de listado secuencial

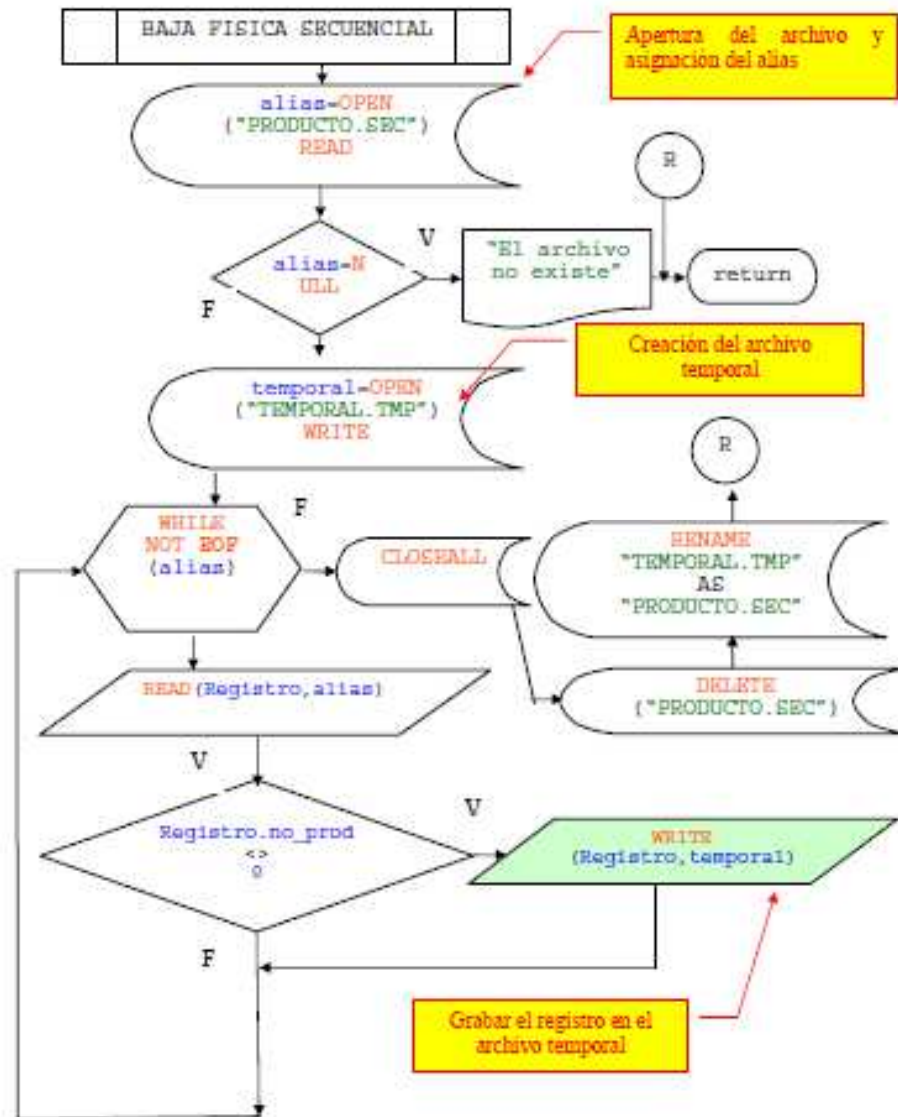
MODIFICACIONES SECUENCIALES



BAJAS EN ARCHIVOS SECUENCIALES



BAJAS FISICAS



ARCHIVOS DIRECTOS EN ...

A diferencia del archivo secuencial en el que se hace un recorrido secuencial para localizar la dirección del registro solicitado, en el archivo de acceso directo, se posiciona el apuntador del archivo directamente en el registro solicitado usando la función Seek. Sin embargo **es muy importante destacar que esta función permite posicionar el apuntador interno del archivo en una dirección lógica válida; esto es, que marcará error al intentar posicionarse fuera del archivo mediante una dirección lógica que exceda el tamaño del mismo.**



POSICIONAMIENTO EN EL FICHERO: FSEEK Y FTELL

Esta función permite situarnos en la posición deseada de un fichero abierto. Cuando leemos un fichero existe un puntero o cursor que indica en qué lugar del fichero nos encontramos. Cada vez que leemos datos del fichero este puntero se desplaza. Con la función `fseek` podemos situar este puntero en el lugar que se precise.

El formato de `fseek` es el siguiente:

```
int fseek(FILE *p_fichero, long desplazamiento, int modo);
```

Como siempre, `p_fichero` es un puntero de tipo `FILE` que apunta al fichero con el que queremos trabajar.

`desplazamiento` es el número de posiciones (o bytes) que queremos desplazar el puntero.



- Este desplazamiento puede ser de tres tipos, dependiendo del valor de modo:

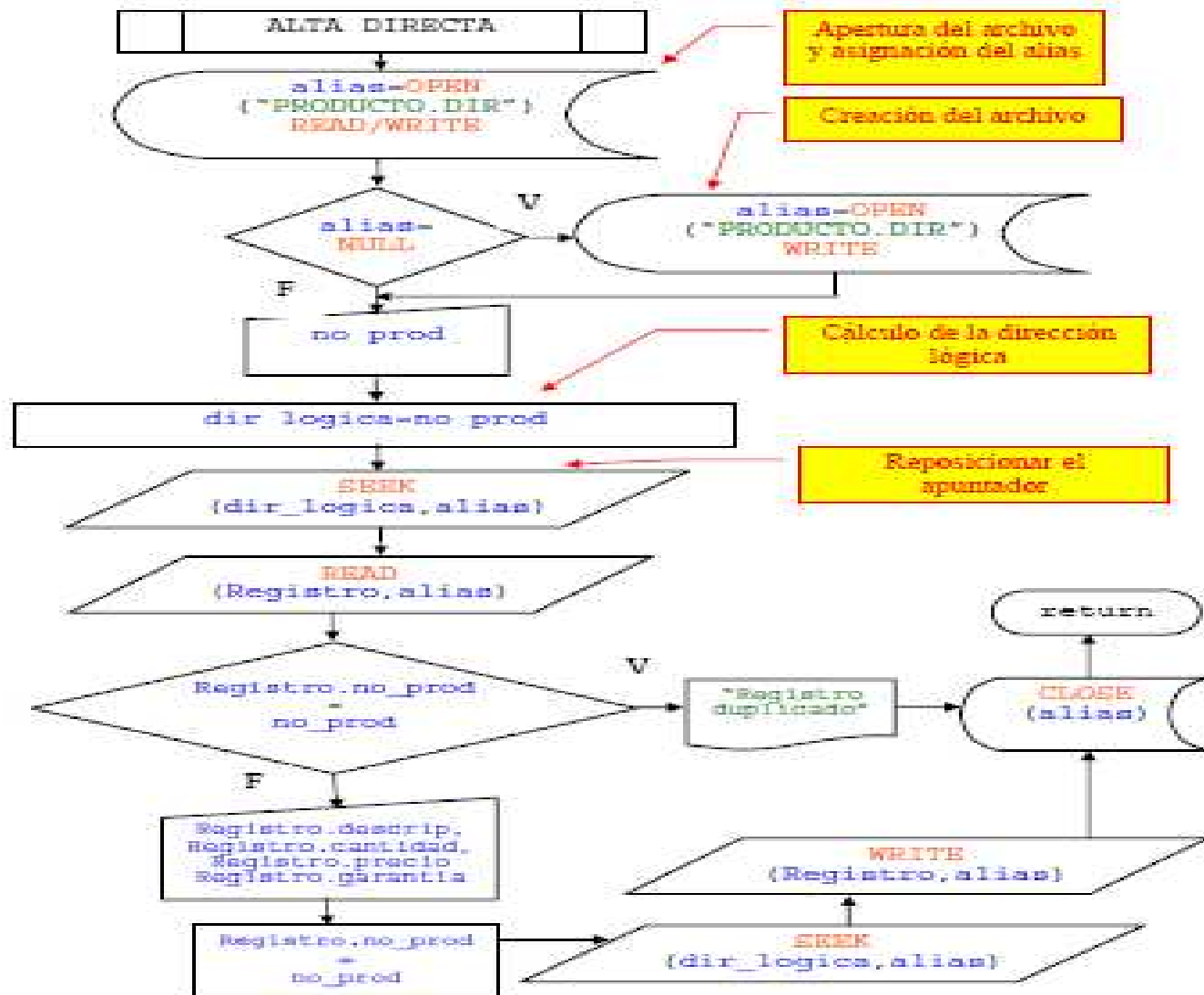
SEEK_SET	El puntero se desplaza desde el principio del fichero.
SEEK_CUR	El puntero se desplaza desde la posición actual del fichero.
SEEK_END	El puntero se desplaza desde el final del fichero.

```
#define SEEK_SET      0
#define SEEK_CUR      1
#define SEEK_END      2
```

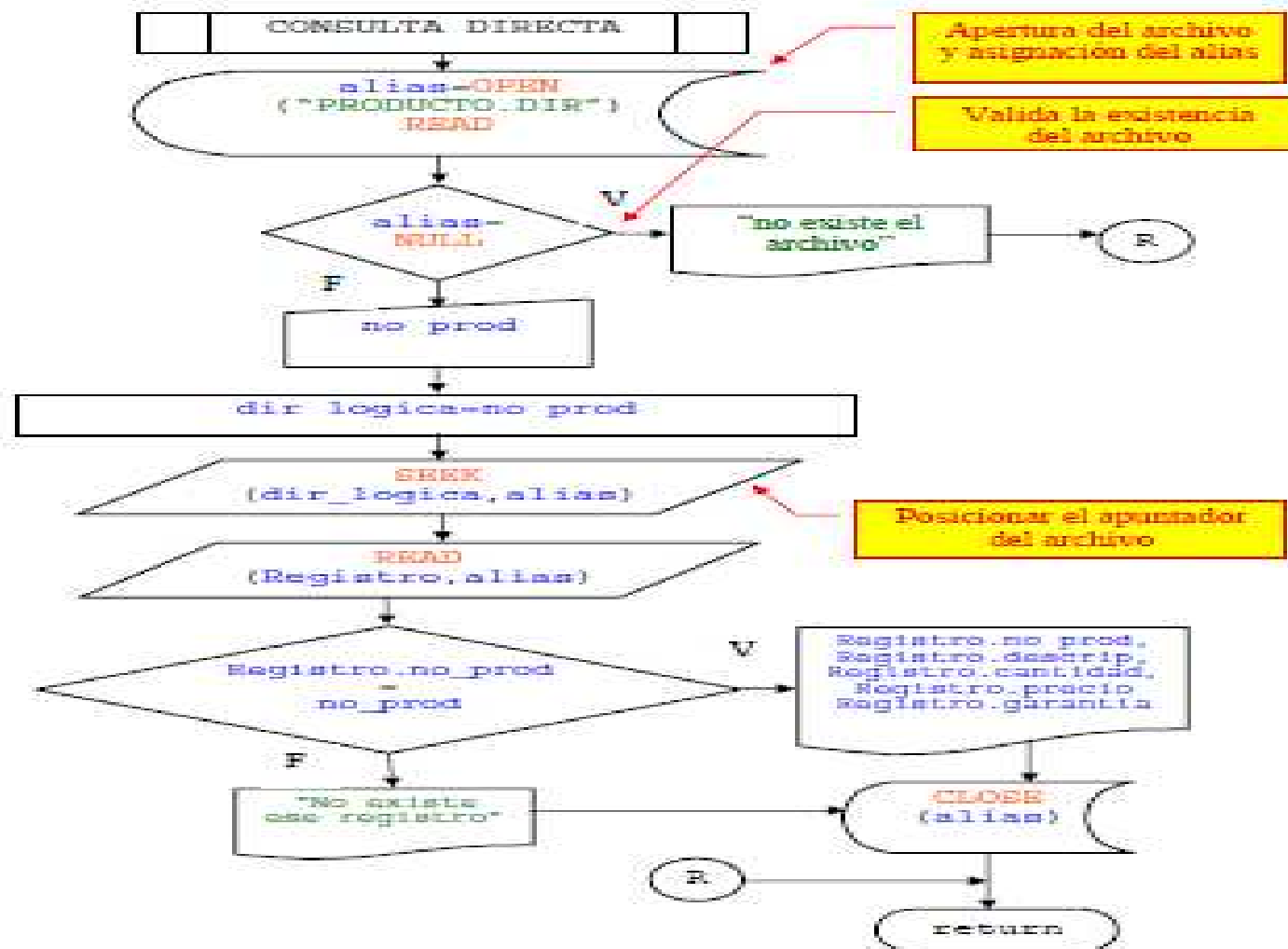



```
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *fichero;
    long posicion;
    int resultado;
    if ((fichero = fopen( "origen.txt", "r" )) == NULL) {
        printf( "No se puede abrir el fichero.\n" );
        exit( 1 );
    }
    printf( "¿Qué posición quieres leer? " );
    scanf( "%li", &posicion );
    resultado = fseek( fichero, posicion, SEEK_SET );
    if (!resultado)
        printf( "En la posición %li está la letra %c.\n", posicion, getc(fichero) );
    else
        printf( "Problemas al posicionar el cursor.\n" );
    fclose( fichero );
}
```

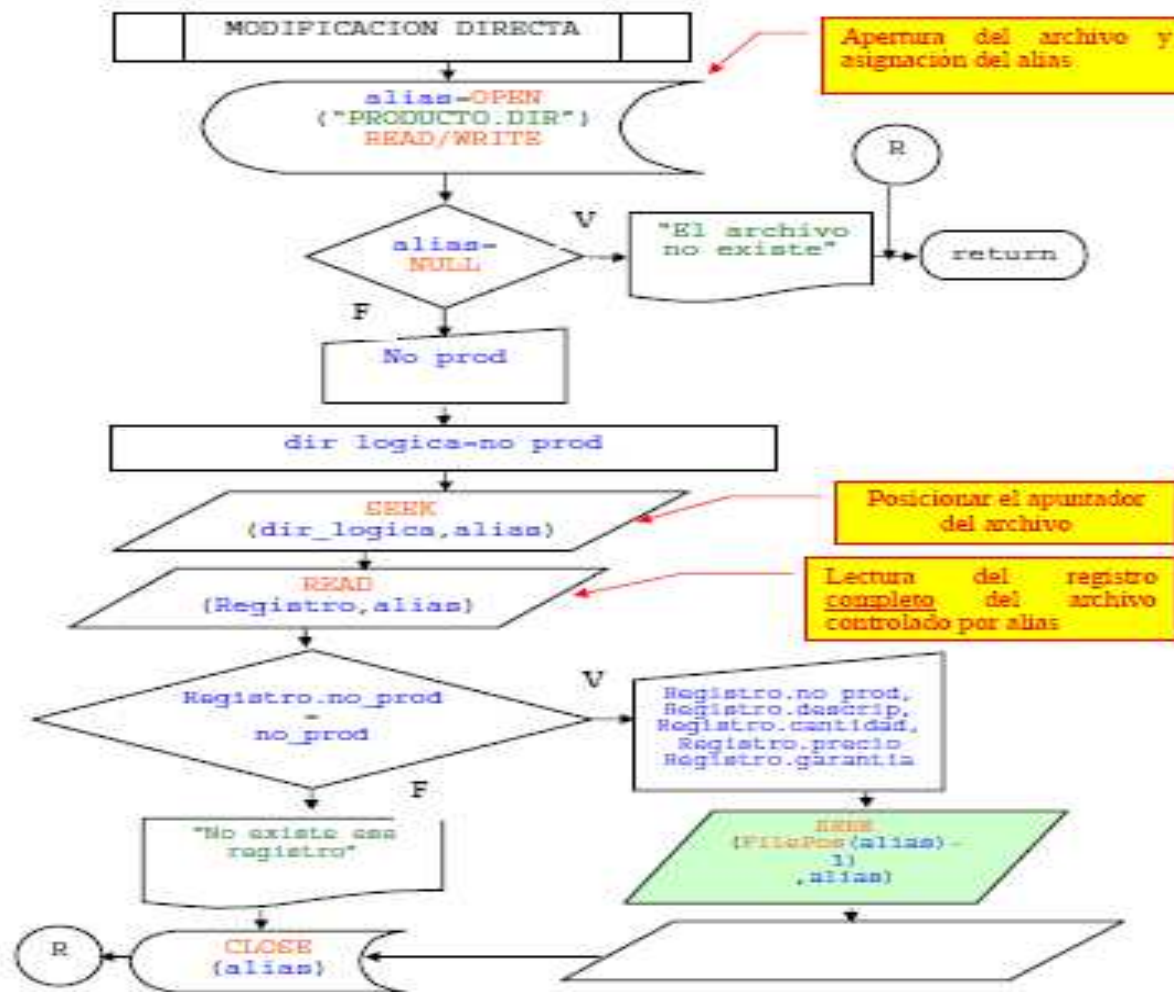
ALTAS DIRECTAS



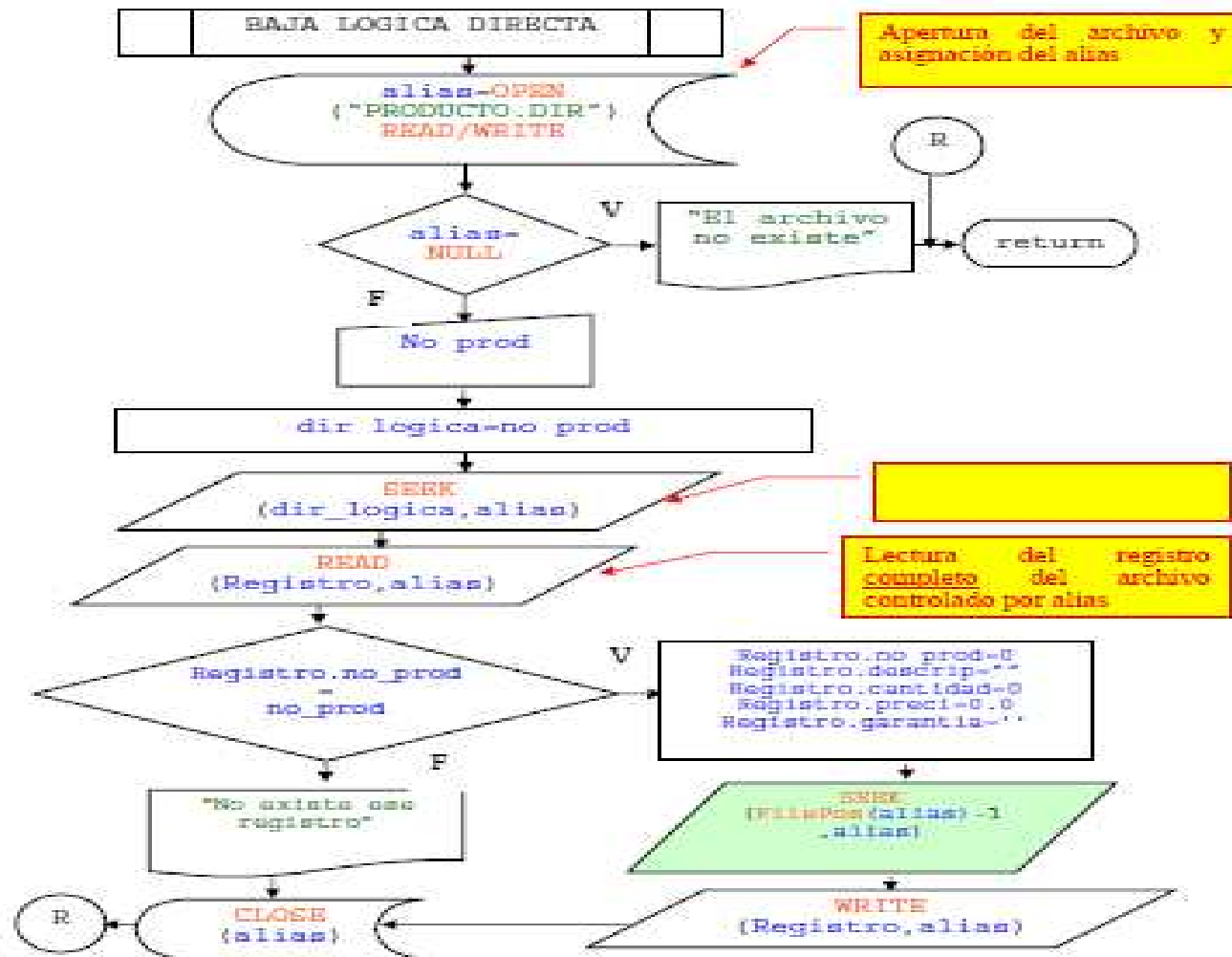
CONSULTAS



MODIFICACIONES



BAJAS



SEGUNDA PARTE



CORTE DE CONTROL EN ARCHIVOS SECUENCIALES

Los registros en los archivos secuenciales están grabados en posiciones físicamente contiguas.

Resulta en algunos casos interesante o necesario, ordenar un archivo secuencial por un determinado campo antes de procesar los registros.

Esta clasificación u ordenación resulta necesaria cuando existen en el archivo varios campos que repiten su contenido.

El problema consiste en determinar el momento preciso en que finalizan los elementos de entrada de un grupo para comenzar con otro.



DEFINICIONES QUE AYUDARAN A COMPRENDER EL CONCEPTO DE CORTE DE CONTROL

Control: significa *mando, gobierno, dirección, dominio*.

Control de Programa: Es el mecanismo para dirigir, gobernar la ejecución de las instrucciones respetando la secuencia lógica establecida en el diagrama.

Corte de Control: *cuando se interrumpe el circuito de instrucciones que se estaban ejecutando.*

Caso típico de corte de control: Procesos donde se solicita determinados procedimientos para grupos de entes que mantienen cierta homogeneidad.

Es necesario determinar el momento preciso en que finalizan los elementos de entrada de un grupo para comenzar con otro.

En síntesis: Detectar el momento en que cambia el valor (contenido) de la variable **campo de control**.



Campo de Control: Es el campo que identifica a cada subconjunto o grupo de elementos de entrada (registros) de un conjunto mayor de datos. No hay que confundir con clave de control.

Ej.

Tenemos un archivo que contiene todos los estudiantes de una facultad, los cuales están identificados por su código de carrera y número de libreta, pero se desea realizar un listado que contenga todos los alumnos ordenado por carrera.



REQUISITOS

Ordenamiento de los datos de entrada

Que existan varios subconjuntos para que tenga sentido el corte de control.

Que cada subconjunto tenga varios elementos o registros.

(El corte se produce en un archivo).



METODOLOGÍA

Se debe tener en cuenta:

Cuando se lee un registro de un archivo, su contenido se guarda en memoria en variables asociadas a dicho archivo. Luego, cuando se lee el siguiente registro, su contenido se almacena en las mismas variables destruyendo la información almacenada del registro anterior.

Por lo tanto para saber si el campo de control del registro recién leído tiene el mismo contenido que el registro anterior, *será necesario haber almacenado en una variable auxiliar el contenido del campo de control del primer registro del grupo o bloque*, para poder compararlo con él, y de esta manera determinar en forma precisa cuando se produce la ruptura del proceso *corte de control*.

Cuando esto se produce, estaremos seguros que el bloque o conjunto de datos que estábamos procesando ha finalizado. Es este el momento de realizar las operaciones relacionadas con la finalización del conjunto (impresión de contadores, impresión de acumuladores, acumular para totales generales, etc.-)

Luego de esto se debe volver a iniciar la variable auxiliar con el contenido del nuevo campo de control del conjunto nuevo a procesar a fin de poder usarlo como referencia de este nuevo bloque.

Se aplica la estructura **mientras** para repetir el conjunto de operaciones relativas a los registros del mismo conjunto. La condición de salida es que el campo auxiliar sea distinto al campo recién leído.



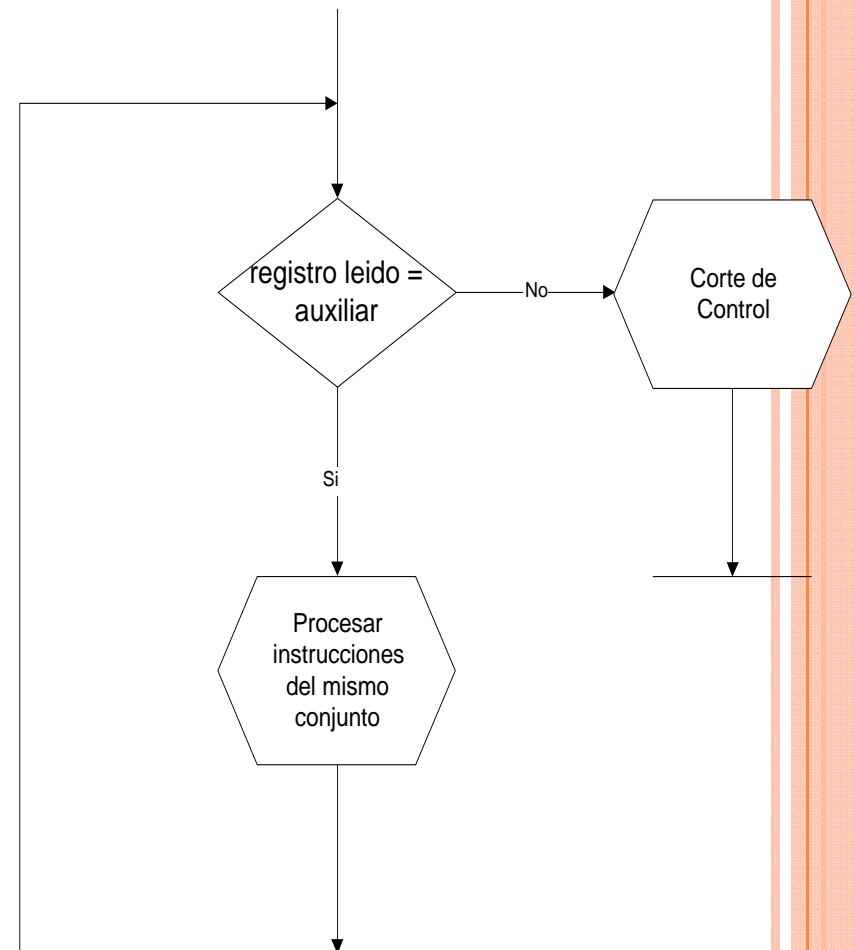
❑ *Habr  una estructura repetitiva general cuya condici n de salida ser  la finalizaci n del archivo (EOF). Debe realizarse una primer lectura antes de entrar a la estructura.*

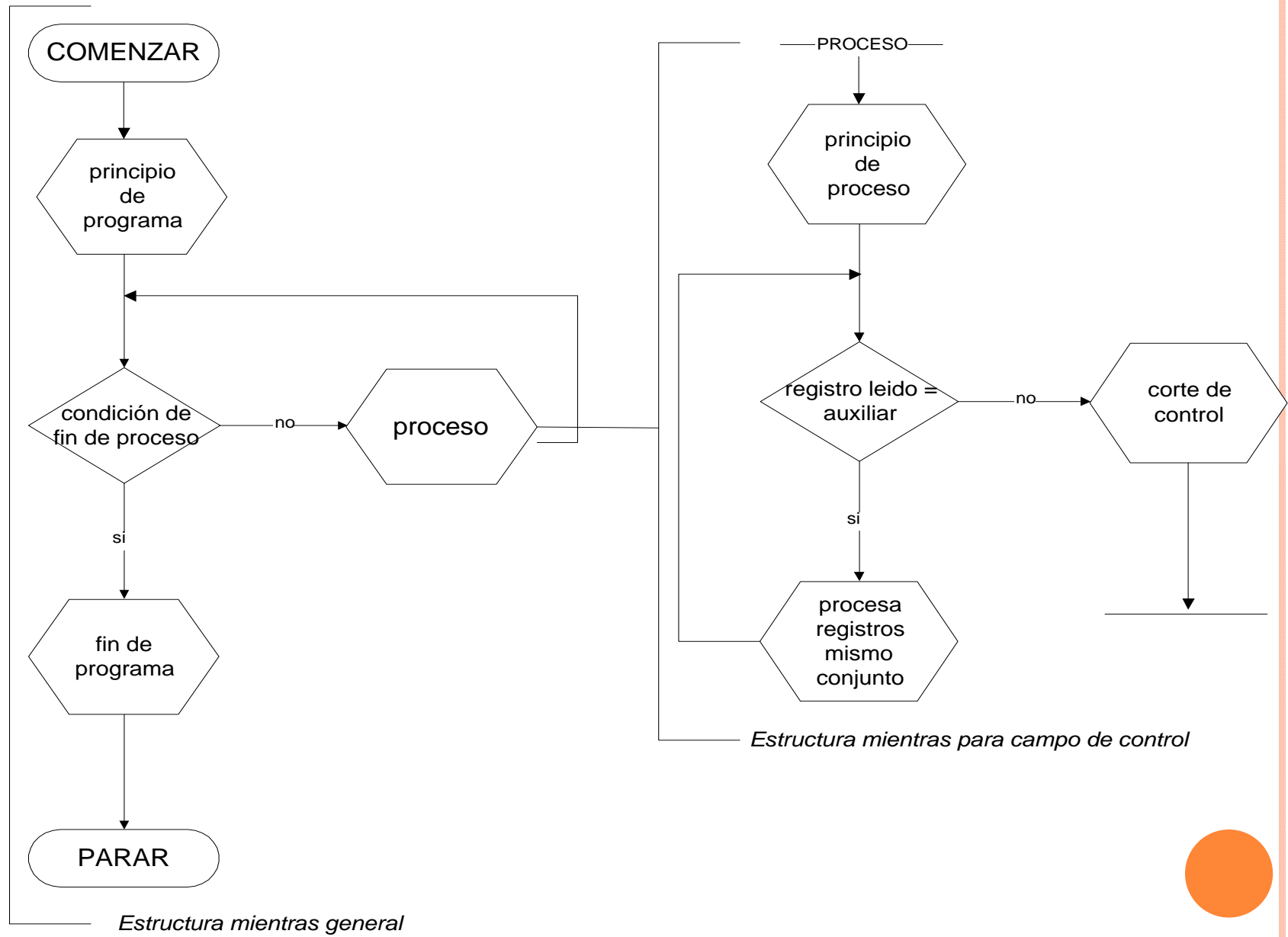
❑ *Antes de entrar a la estructura repetitiva general, se inicializar n los acumuladores y contadores.*

❑ *Se deber  inicializar variables relacionadas con el comienzo de cada bloque o conjunto de control.*

❑ *Las operaciones necesarias relativas a la finalizaci n de un bloque de control se realizar n inmediatamente despu s de salir de la estructura asociada.*

❑ *Se debe realizar lecturas como  ltima instrucci n de la estructura repetitiva interna.*





Corte por Fin de Archivo (EOF o FDA)


En todo programa que lee un archivo secuencial existe una estructura repetitiva general cuya condición de salida será la finalización del archivo (EOF). Por lo tanto tendrá que realizarse una primer lectura antes de entrar a la estructura de proceso.

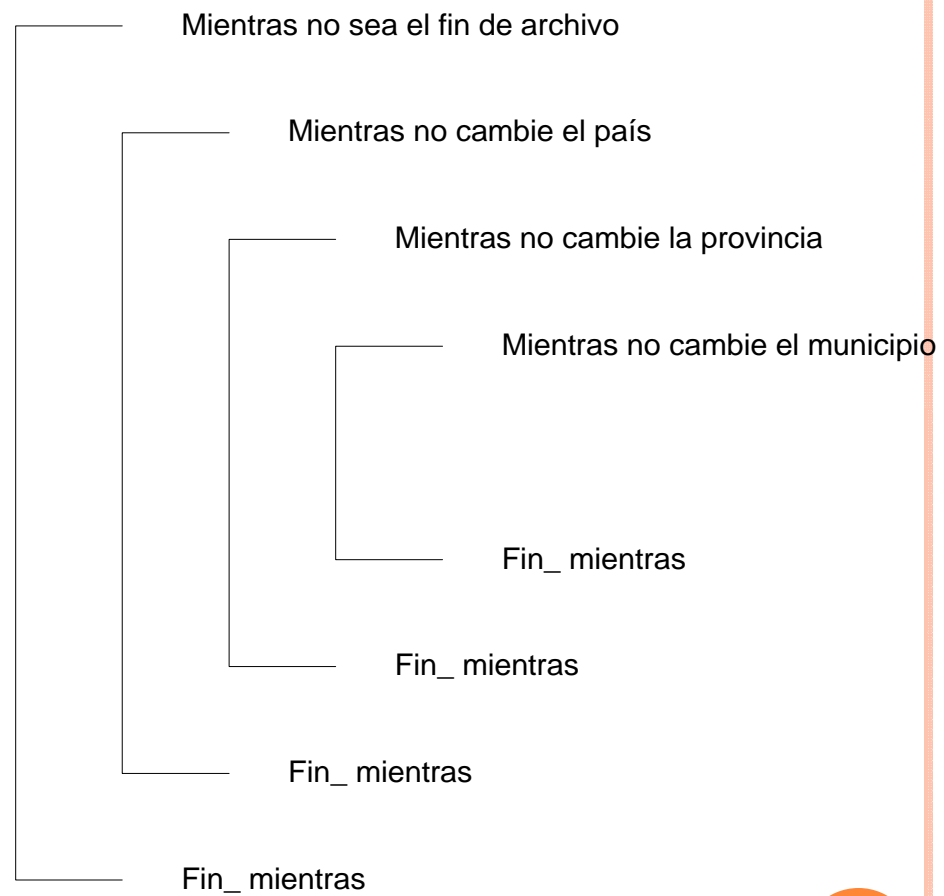
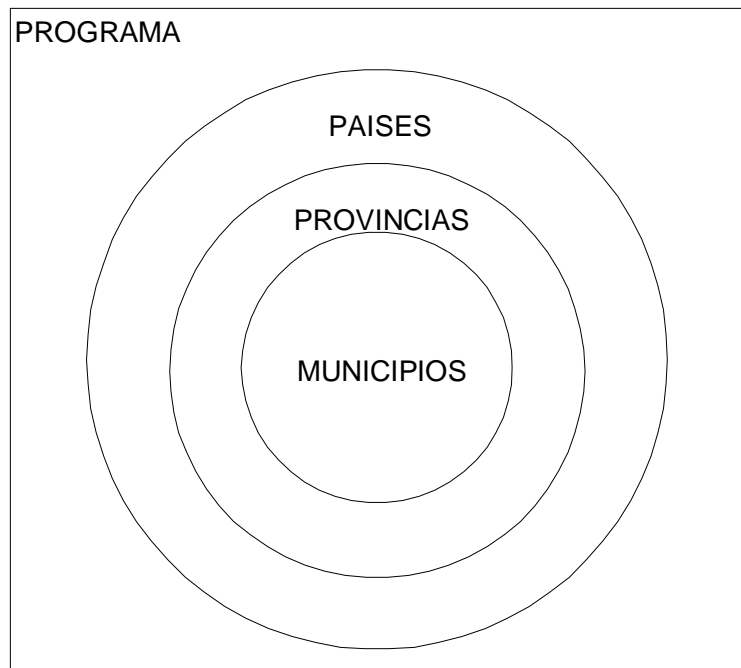
Corte Simple

Se da cuando existe un solo campo de control y por lo tanto un solo corte de control.

Corte Compuesto

Cuando existen varios campos de control de los cuales queremos obtener información. En otras palabras, existirán tantas estructuras repetitivas anidadas dentro de la estructura general como cortes de control haya, entonces la estructura más externa será aquella que contenga al resto y en consecuencia la mas interna la que representa a la entidad contenida en las demás.





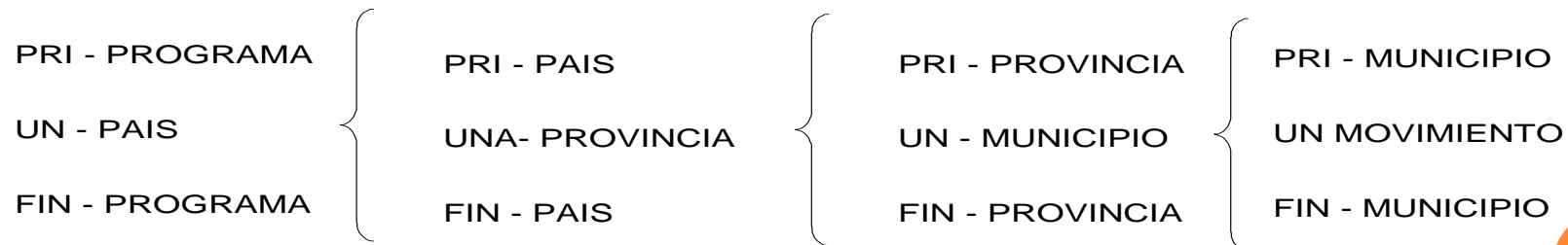
Jerarquía de comparación

Depende del enunciado del problema. O sea cual es la división más importante y consecuentemente el diseño de salida de los resultados buscados.

Depende mucho de la organización jerárquica de los datos (Archivo), es decir de su ordenamiento.

Si el archivo no está ordenado NO SE PUEDE aplicar corte de control. (Solución: Arreglos).

Corte por arrastre



Corte de Control (Acceso secuencial)

Ejercicio 1: Se tiene registrada la información sobre las notas de los exámenes finales de los alumnos de la facultad durante el presente año, en un archivo ordenado en forma ascendente por número de libreta. De cada alumno se conoce el: Número de libreta, código de materia y nota obtenida.

Se deberá:

1. Listar un renglón por alumno con el promedio de notas obtenida.
2. Informar el total de alumnos.

Listado de promedios

Número de Libreta	Nota promedio
X000x	X0,X0x
X000x	X0,X0x
.....
X000x	X0,X0x
Total de alumnos: xxx	

TERCERA PARTE



APAREO DE ARCHIVOS

El apareo de archivos es un proceso que dependiendo del tipo de organización que tengan estos archivos, el algoritmo adoptará una estructura particular.

Fusionar varios archivos (al menos dos) secuenciales homogéneos, mediante el uso de un campo común a todos ellos llamado clave de control.

Encontrar parejas de registros pertenecientes al mismo ente, en el cual cada uno de ellos contiene parte de la información requerida para el proceso siguiente (ej. Armar un nuevo registro con los datos de ambos archivos).

Todos los archivos deberán tener un campo común que es el nexo para realizar el proceso.

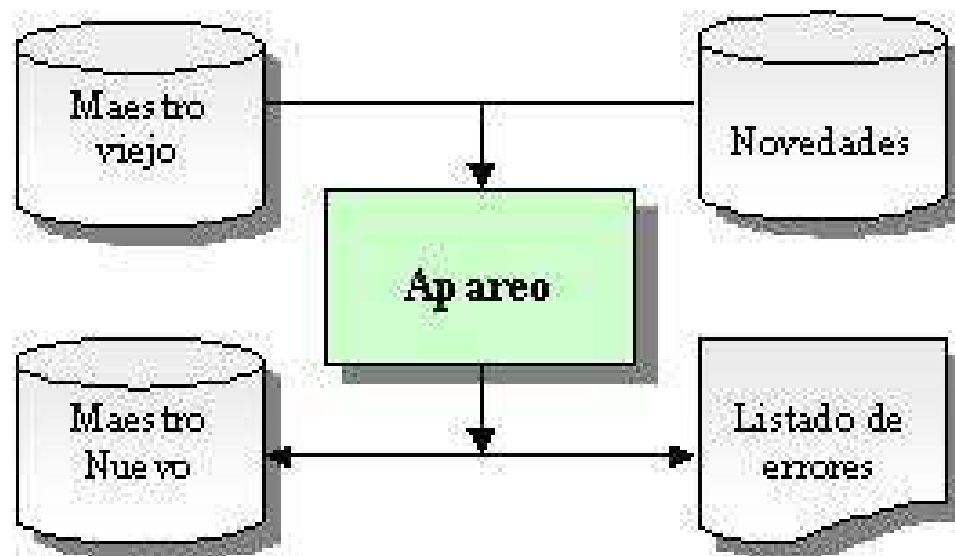
Ej. De uso de la técnica: Actualizar saldos de Cuentas corrientes de una Entidad Financiera; actualización de movimientos de artículos de un almacén; liquidación de haberes de los empleados; etc.

APAREO DE ARCHIVOS

El procesamiento de la técnica se hace en lotes de registros (batch).

Contraposición de la técnica: procesos en línea usando métodos de acceso directo.

Este proceso es muy empleado para la actualización del maestro a través del archivo de novedades. Las novedades tendrán que ver con altas, bajas o modificaciones o algunas de ellas solamente.




APAREO DE ARCHIVOS

Generalmente se presenta con dos tipos de archivos muy diferenciados por su grado de volatilidad (mayor o menor duración de los datos).

Archivo Maestro: Contiene información permanente o semi permanente. Tiene poca o ninguna volatilidad.

Archivo Detalle (Movimiento o Novedad): Contiene información transitoria, con la que se actualiza o modifica los datos del archivo maestro. Tiene alta volatilidad. Normalmente, luego del proceso, son desechados.



APAREO DE ARCHIVOS

Requisitos

Ordenamiento. Todos los archivos deben estar ordenados por el mismo campo clave de control (nexo común) de la misma forma.

Nexo Común. Todos los archivos (que participan del proceso de apareo) deben tener un mismo campo común que servirá de nexo entre ellos. Normalmente es la clave de control.

Homogeneidad. Todos los archivos deben ser de entes de la misma naturaleza.

APAREO DE ARCHIVOS

Mantenimiento de Archivos

Consiste en la realización de todas las intervenciones que soportan los mismos (archivos) durante su vida útil.

Actualizaciones. Consiste en modificar (total o parcialmente), eliminar o agregar registros de/en un archivo (generalmente maestro), a partir de los registros informados en el archivo de novedad.

Consiste en cambiar el contenido de uno o mas campos de un registro.

Casos:

- a) Mediante el reemplazo directo de dato que contiene el movimiento. (ej. Domicilio)
- b) Resultado de cálculos que se realizan.

APAREO DE ARCHIVOS

Casos

1.- Uno o ningún registro de movimiento por cada registro del maestro.

```
IF CM = CD THEN  
    CON_MOV  
    LEER_MOV  
    LEER_MAE  
ELSE  
    SIN_MOV  
    LEER_MAE;
```



APAREO DE ARCHIVOS

Casos

2.- Uno o ningún registro de movimiento por cada registro del maestro, con error.

```
IF CM = CD THEN  
    CON_MOV  
    LEER_MOV  
    LEER_MAE  
ELSE  
    IF CM < CD THEN  
        SIN_MODIF  
        LEER_MAE  
    ELSE  
        ERRORES  
        LEER_MOV;
```



APAREO DE ARCHIVOS

Casos

3.- Uno, varios o ningún registro de movimiento por cada registro del maestro.

```
PRI_ENTE;  
DO WHILE CM = CD  
    UN_MOV  
    LEER_MOV;  
FIN_ENTE;
```

```
PROCEDURE PRI_ENTE;
```

** Conjunto de instrucciones para iniciar el proceso de pareo (puede estar vacio);

```
PROCEDURE FIN_ENTE; (* puede ser necesario preguntar por una bandera*)
```

** Conjunto de instrucciones para procesar Maestro sin movimiento

```
LEER_MAE;
```

```
PROCEDURE UN_MOV;
```

** Conjunto de instrucciones para procesar Maestro con uno o varios movimiento/s

```
LEER_NOV;
```



APAREO DE ARCHIVOS

Casos

4.- Uno, varios o ningún registro de movimiento por cada registro del maestro, con error.

```
PRI_ENTE;  
DO WHILE CM = CD  
    UN_MOV  
    LEER_MOV;  
FIN_ENTE;
```

```
PROCEDURE PRI_ENTE;  
** Conjunto de instrucciones para iniciar el proceso de  
pareo (puede estar vacio);
```

```
PROCEDURE FIN_ENTE; (* puede ser necesario  
preguntar por una bandera*)  
IF CM < CD THEN  
    SIN_MODIF (*Maestro sin movimiento, se graba  
tal cual se lee)  
    LEER_MAE  
ELSE  
    ERROR (* movimiento sin maestro, es un error. Se  
lo trata como error)  
    LEER_MOV;
```

```
PROCEDURE UN_MOV;  
** Conjunto de instrucciones para procesar Maestro  
con uno o varios movimiento/s
```

APAREO DE ARCHIVOS: CASO: BAJA

```
PRI_ENTE;  
IF CM = CD THEN  
    BAJA  
    LEER_MOV  
    LEER_MAE;  
FIN_ENTE;
```

```
PROCEDURE PRI_ENTE;  
** Conjunto de instrucciones para iniciar el  
proceso de BAJA (puede estar vacio);
```

```
PROCEDURE FIN_ENTE; (* puede ser  
necesario preguntar por una bandera*)  
IF CM < CD THEN  
    SIN_MODIF (*Maestro sin movimiento  
(baja), se graba tal cual se lee)  
    LEER_MAE  
    ELSE  
    ERROR (* baja (movimiento) sin maestro, es  
un error)  
    LEER_MOV;
```

```
PROCEDURE BAJA;  
** Conjunto de instrucciones para procesar la  
baja del Maestro;
```


APAREO DE ARCHIVOS: ALTAS



```
PRI_ENTE;  
DO WHILE CM = CD  
    ERROR  
    LEER_MOV;  
FIN_ENTE;
```

```
PROCEDURE PRI_ENTE;
```

** Conjunto de instrucciones para iniciar el proceso de ALTAS (puede estar vacio);

```
PROCEDURE FIN_ENTE; (* puede ser necesario preguntar  
por una bandera*)
```

```
IF CM < CD THEN
```

```
    SIN_MODIF (*Maestro sin movimiento (alta), se graba tal  
cual se lee)
```

```
    LEER_MAE
```

```
    ELSE
```

```
    ALTA (* es un registro que no esta en el maestro, hay que  
incorporarlo (alta).
```

```
    LEER_MOV;
```

```
PROCEDURE ERROR;
```

** es un registro (alta) que ya esta en el maestro. ERROR;

```
PROCEDURE ALTA;
```

** conjunto de instrucciones para dar de alta el registro;



APAREO DE ARCHIVOS

Casos

6.- Altas, bajas y modificaciones en forma conjunta.





PROBLEMAS A TRATAR

Enunciado del problema

Especificar el tipo de caso de apareo

Diagrama de flujo con la solución

Codificación en Pascal

Fin



ENUNCIADO DEL PROBLEMA

A partir de los datos de los alumnos contenidos en un archivo MAESTRO y los datos de NOVEDADES con la materia aprobada por cada alumno que ha aprobado, cuyos diseños son los siguientes:

Diseño Archivo Maestro. Un registro por cada alumno.

--	--	--	--

Diseño Archivo Novedad. Un o ningún registro por alumno.

--	--	--

Se desea:

- A) Actualizar la fecha de aprobación de la ultima asignatura
- B) Actualizar la cantidad de asignaturas aprobadas
- C) Listar los datos y la asignatura aprobada



ESPECIFICAR EL CASO DE APAREO

En el Maestro están todos ?

SI

Puede faltar registros en el Maestro?

NO

En la Novedad, viene un registro por cada Maestro?

NO

En la Novedad, viene uno, varios o ningún registro por cada Maestro?

NO

En la Novedad, viene uno o ningún registro por cada Maestro?

SI

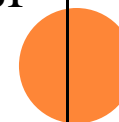
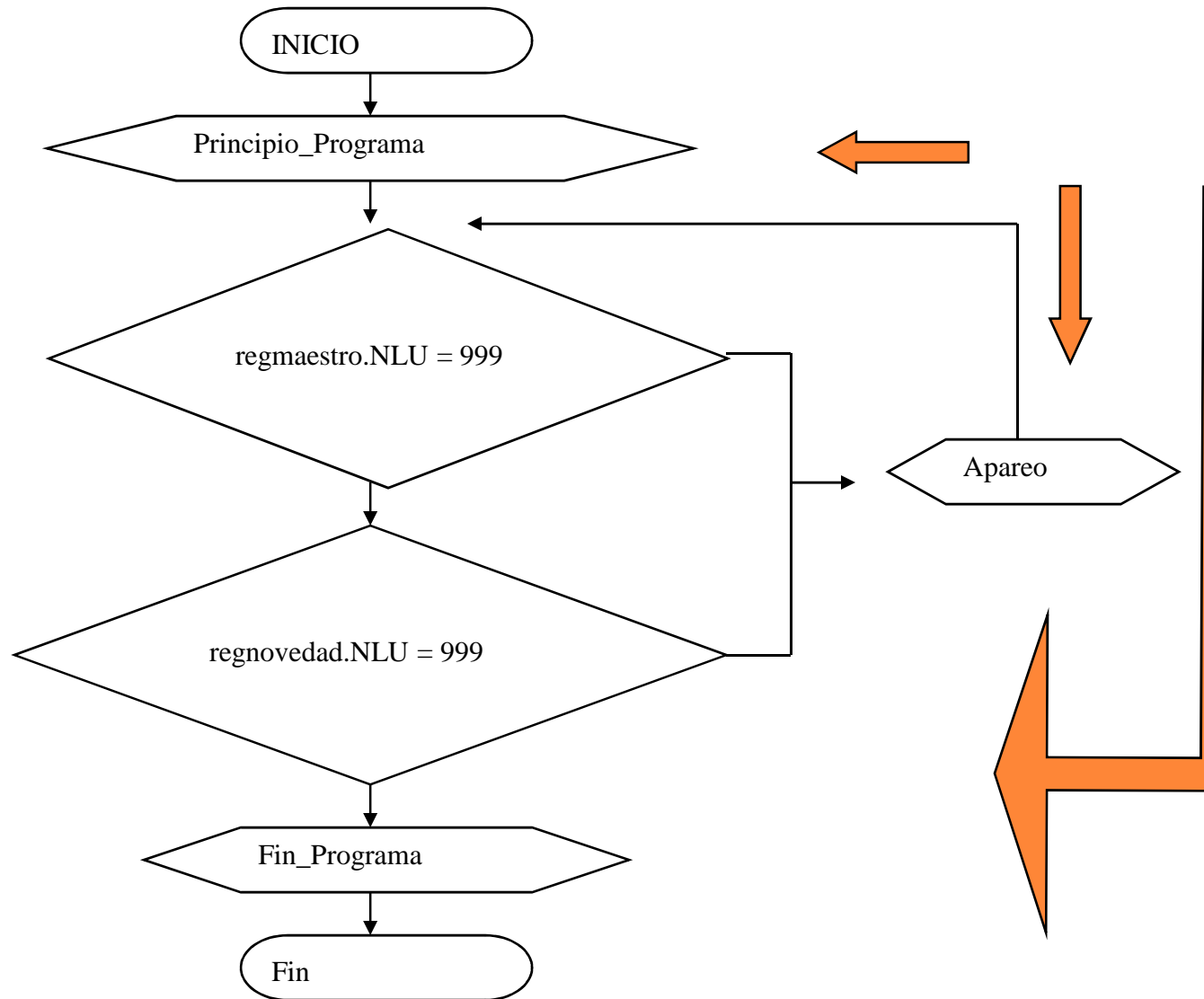
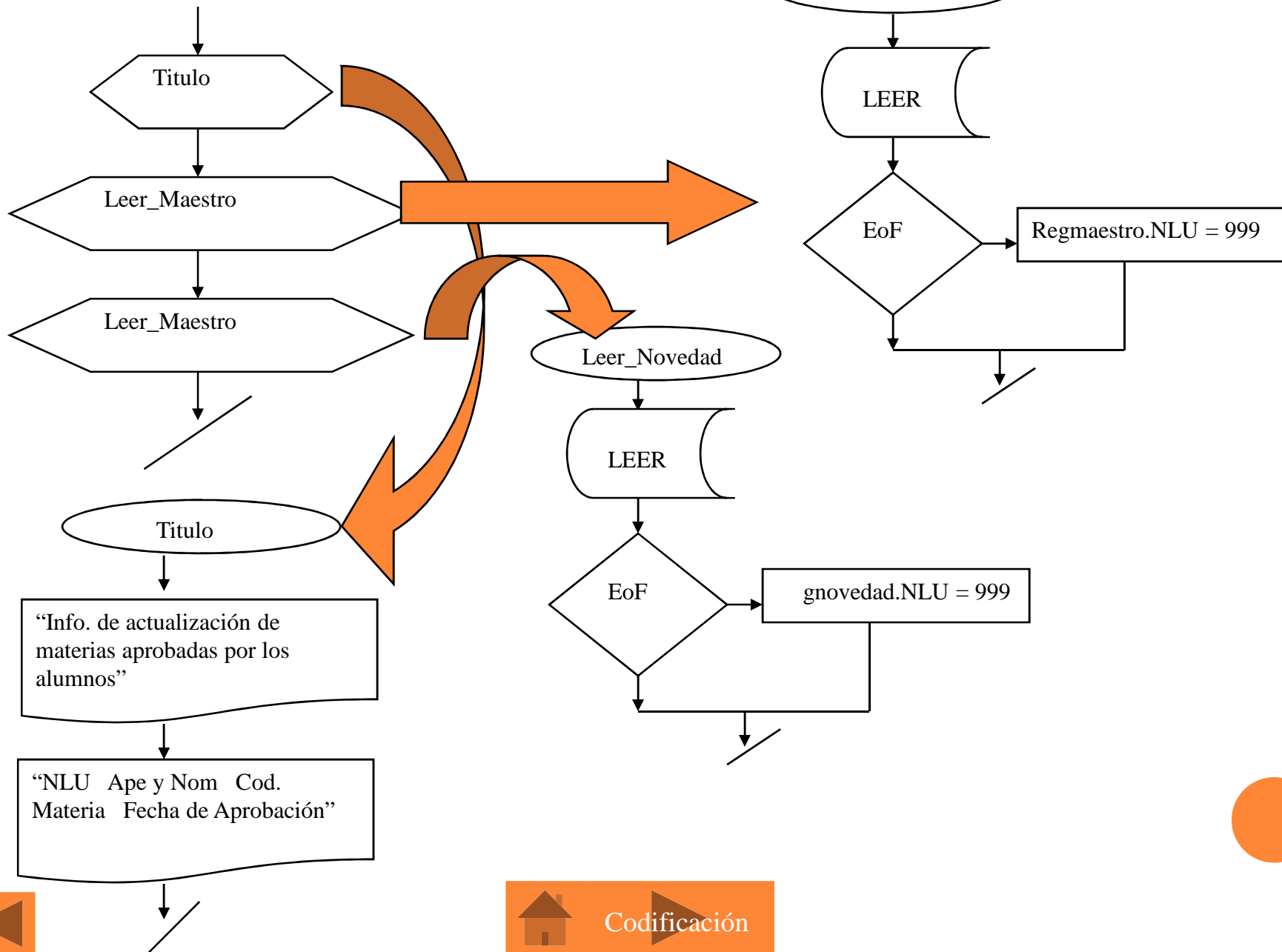


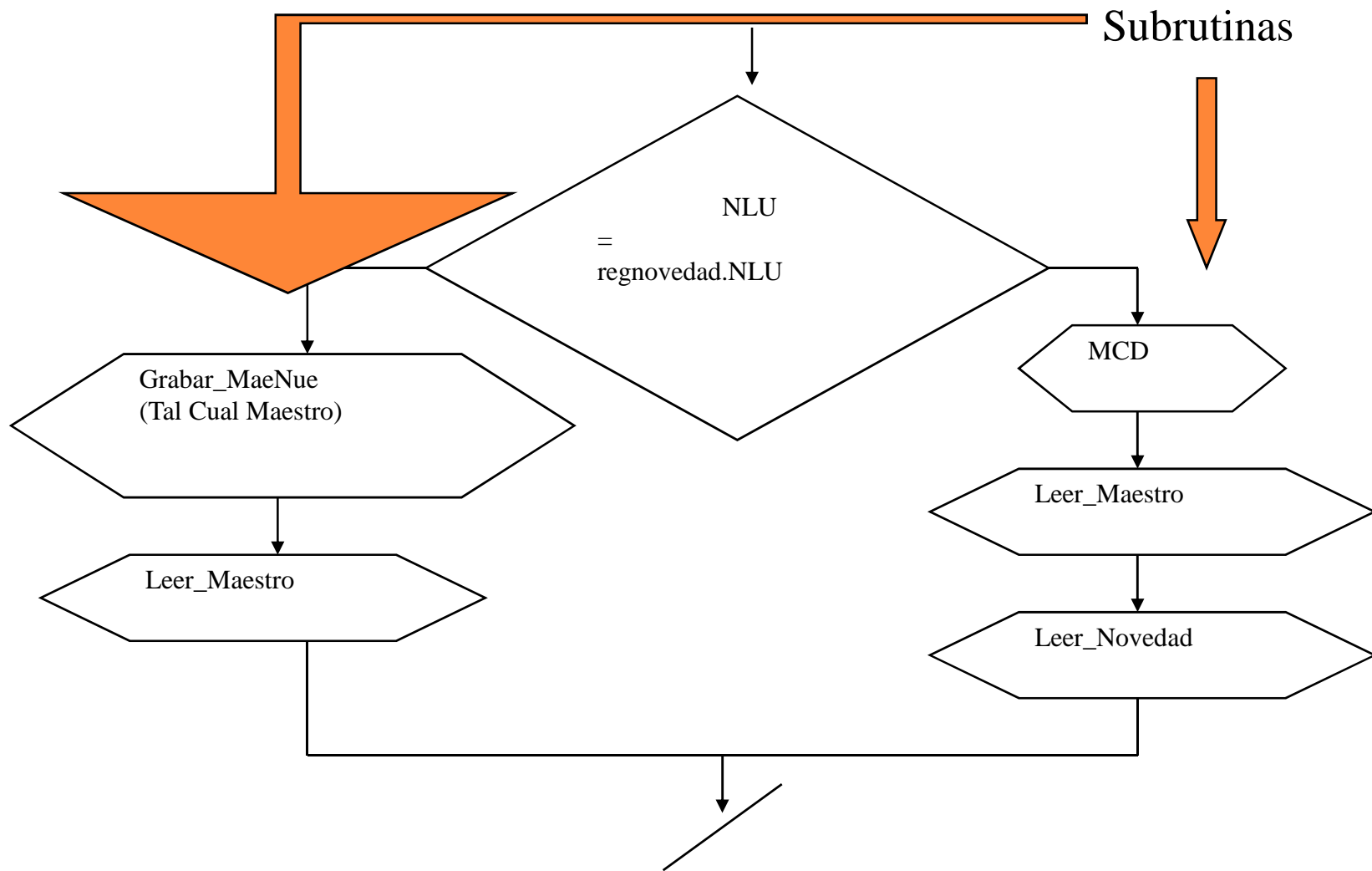
DIAGRAMA DE FLUJO



PRINCIPIO_PROGRAMA



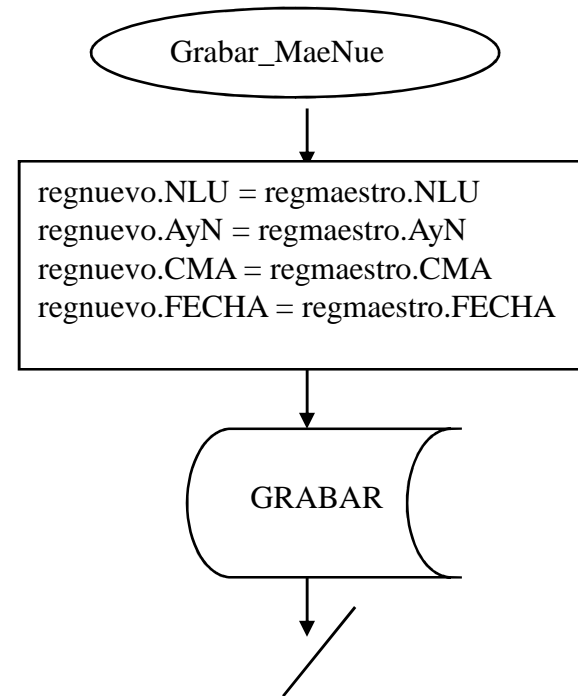
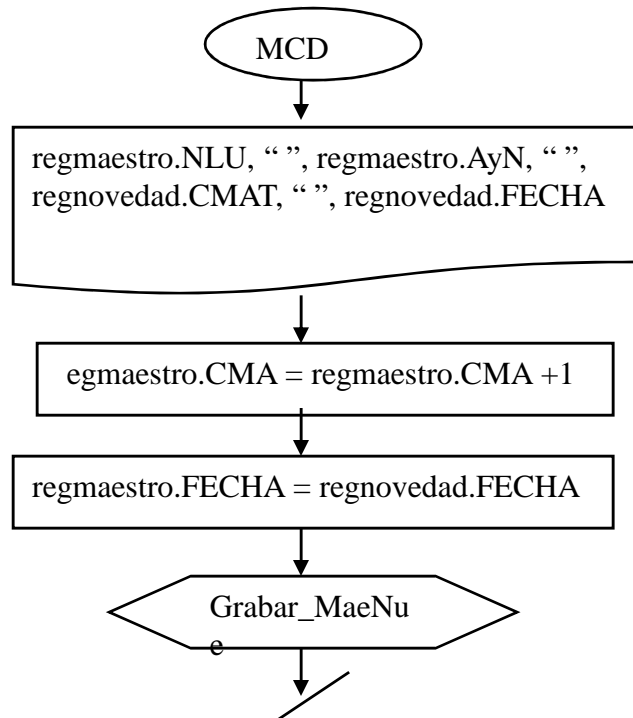
APAREO



Codificación



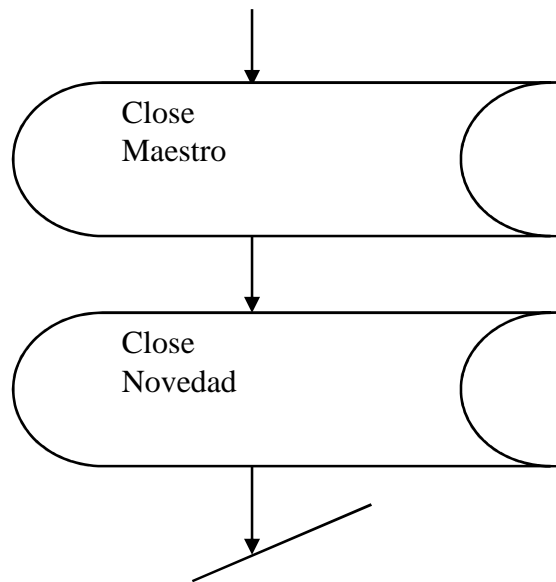
RUTINAS MCD Y GRABAR_MAE NUE



Codificación



FIN_PROGRAMA



Codificación



CODIFICACIÓN DE LA RUTINA PRINCIPIO_PROGRAMA (DF)

```
Principio_Programa (){  
    do while (regmaestro.nlu < > 999)  
        & (regnovedad.nlu < > 999)  
        Apareo ();  
    Fin_Programa ();  
};
```

CODIFICACIÓN DE LA RUTINA PRINCIPIO_PROGRAMA (DF) CONTINUA....

```
void Leer_Maestro {  
    if eof(maestro)  
        regmaestro.nlu:= 999  
    else  
        fread(maestro, regmaestro);  
};
```

```
void Leer_Novedad {  
    if eof(novedad)  
        regnovedad.nlu:= 999  
    else  
        fread (novedad, regnovedad);  
};
```

CODIFICACIÓN DE LA RUTINA PRINCIPIO_PROGRAMA (DF) ..CONTINUA

```
Procedure Apareo () {  
    if regmaestro.nlu = regnovedad.nlu {  
        MCD ();  
        Leer_Maestro ();  
        Leer_Novedad ();  
    }  
    else {  
        Grabar_MaeNue ();  
        Leer_Maestro ();  
    }  
}
```

CODIFICACIÓN DE LA RUTINA PRINCIPIO_PROGRAMA (DF) ..CONTINUA

```
void Procedure MCD (); {  
    fprintf(“%...”, regmaestro.NLU, regmaestro.AyN, regnovedad.CMAT,);  
    fprintf(“%...”, regnovedad.FECHA);  
    regmaestro.CMA := regmaestro.CMA + 1;  
    regmaestro.FECHA := regnovedad.FECHA;  
    Grabar_MaeNue ();  
};
```

```
void Grabar_MaeNue () {  
    regnuevo.NLU = regmaestro.NLU;  
    regnuevo.AyN = regmaestro.AyN;  
    regnuevo.CMA = regmaestro.CMA;  
    regnuevo.FECHA = regmaestro.FECHA;  
    fwrite (nuevo, regnuevo);  
};
```

CODIFICACIÓN DE LA RUTINA PRINCIPIO_PROGRAMA (DF) ..CONTINUA

```
void Fin_Programa () {  
    fclose(maestro);  
    fclose(novedad);  
    fclose(nuevo);  
};
```

BIBLIOGRAFÍA

- ALGORITMOS, DATOS Y PROGRAMAS con aplicaciones en Pascal, Delphi y Visual Da Vinci. De Guisti. Armando. 2001. editorial: Prentice Hall. ISBN: 987-9460-64-2. Capitulo 12.
- PROGRAMACIÓN; Castor F. Herrmann, María E. Valesani.; 2001; Editorial: MOGLIA S.R.L..ISBN: 9874338326. Capitulo 2.

