Tema VII. Archivos

Introducción al concepto de Archivos. Tipos de Archivos. Operaciones con Archivos. Corte de Control. Concepto corte de control. Definiciones. Requisitos. Proceso para el corte de control. Corte simple. Corte Compuesto. Corte por fin de datos. Operaciones de altas, bajas, modificaciones y consultas a Archivos Directos.

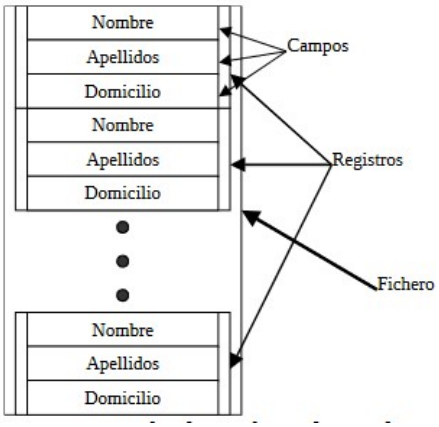
7.1 Introducción al concepto de Archivos.

¿Qué es un archivo?

En informática, entenderemos como archivo o fichero a un conjunto de informaciones sobre un mismo tema tratado como una unidad de almacenamiento y organizado de forma estructurada para la búsqueda de un dato individual. Por tanto, un archivo no es más que una agrupación de datos, cuya estructura interna es la que el usuario, programador o sistema operativo le haya conferido implícitamente, pues la organización del archivo no es intrínseca a la existencia del mismo.

Es necesario aclarar en este punto que los archivos son independientes de los programas y de hecho, un mismo archivo creado por un programa puede ser usado por otro. El único requisito para obtener información de un archivo cualquiera es conocer cómo se han organizado los datos en el mismo.

Un archivo, desde una concepción clásica, está formado por registros, que son las unidades elementales que componen el archivo. Un registro es una colección de información generalmente relativa a una entidad particular (un estudiante, un vehículo, etc.). Los registros pueden contener varios datos, siendo cada uno de estos datos los campos de los registros. Un campo puede tener una longitud fija o variable, debiendo existir en el segundo caso alguna manera de indicar su longitud. El término longitud del campo hace referencia al tamaño máximo de un campo. Los campos pueden ser de diferente tipo: numérico, cadena, fecha, etc. Como consecuencia de ello, un registro puede tener también una longitud variable, refiriendo el término longitud del registro al tamaño máximo de un registro.



Ejemplo de archivo formado por registros

¿Cómo surge la necesidad de utilizar archivos?

Como único medio de almacenamiento: variables, arreglos o estructuras de datos más 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.

7.1.1 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.

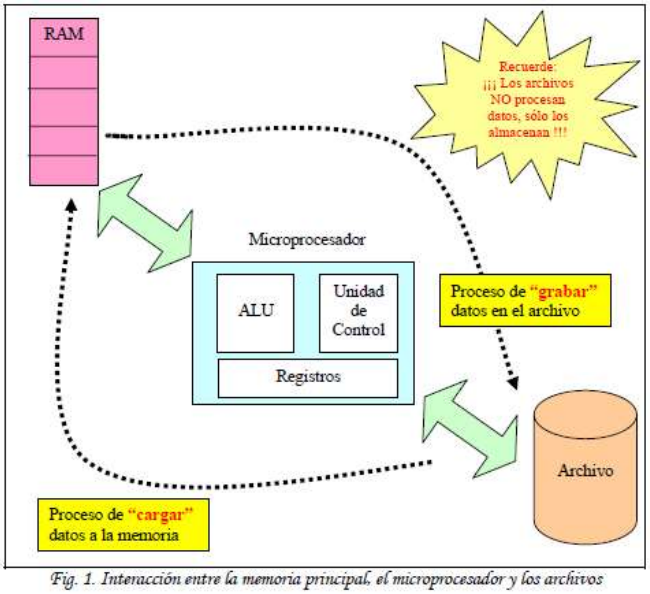


Fig. 1. Interacción entre la memoria principal, el microprocesador y los archivos

7.1.2 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 de acuerdo al formato mostrado en la Fig. 2.

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. P. ejem. Diskettes, discos duros, CD´s, ZIP drives, flash drives, memory sticks, etc.

Fig. 4. Formato del registro de Productos

7.1.3 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.

7.1.4 Tipos de Registros

Registro Físico: Cantidad de datos que puede transferirse en una operación de I / O a través del buffer. En el disco la información está escrita físicamente, y la cantidad más pequeña de datos que puede transferirse en una operación de lectura/escritura entre la memoria y el dispositivo periférico se denomina registro físico.

Registro Lógico: Definido por el programador. Es un conjunto de información relacionado lógicamente, perteneciente a una entidad y que puede ser tratado como una unidad por un programa. Los campos o elementos de un registro lógico pueden ser variables simples o estructuradas.

Factor de Bloqueo: Número 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

7.1.5 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 creo, 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 limitada, 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 esta.

7.2 Tipos de Archivos

<table>
<tr><th colspan="3">Según su uso</th></tr>
<tr><td rowspan="3">Archivos permanentes</td><td>Archivo maestro</td><td>Un archivo maestro contiene el estado actual de los datos susceptibles de ser modificados en la aplicación. Es el núcleo central de la aplicación y todos los procesos están, en general, orientados a actualizar el archivo maestro o a obtener resultados de él. Un ejemplo de este tipo de archivo es el archivo de clientes de un banco, en el que los registros contienen información de identificación de clientes, su saldo en cuenta, etcétera.</td></tr>
<tr><td>Archivo constante</td><td>Un archivo constante es aquel que contiene datos fijos para la aplicación. En él las modificaciones no son frecuentes, normalmente se accede a él sólo para consultar. Serán archivos constantes los que contengan los intereses para distintos tipos de cuentas bancarias, la ubicación de estantes en una biblioteca, la capacidad de las aulas de un centro, una tabla de números primos, etcétera.</td></tr>
<tr><td>Archivo histórico</td><td>Un archivo histórico es aquel que contiene datos que fueron actuales en tiempos anteriores. Se conservan para poder reconstruir la situación actual o situaciones anteriores. En algunos casos puede estar formado simplemente por los registros borrados del archivo maestro. Un archivo histórico puede contener, por ejemplo, los clientes que se han dado de baja en una entidad bancaria.</td></tr>
<tr><td>Archivo de movimientos</td><td colspan="2">En ellos se almacena la información que se utilizara 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, este 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.</td></tr>
<tr><td>Archivos de trabajo</td><td colspan="2">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.</td></tr>
</table>

Desde el punto de vista hardware, para acceder a un archivo sólo existen direcciones físicas. Por ejemplo, en un disco la información se lee o escribe en forma de bloques (sectores) referenciados por unidad/superficie/pista/sector. El sistema operativo se encarga de transformar la dirección lógica de un dato en el archivo (posición del dato dentro del archivo) a la dirección física de forma transparente para el usuario.
Desde un punto de vista físico, el acceso a los datos puede realizarse de dos formas:

Según el acceso	
Acceso secuencial	Se caracteriza porque el acceso a la información de un archivo se produce en un orden determinado y fijo. El acceso a los bloques de datos se hace de forma sucesiva, uno tras otro, tal como se reproducen las canciones grabadas en una cinta.
Acceso directo	Se caracteriza porque el acceso a determinada información contenida en el archivo puede hacerse sin acceder a las informaciones anteriores o posteriores, como ocurre al acceder a una canción grabada en un CD-ROM /DVD

Existen diferentes formas de estructurar u organizar los registros que componen un archivo sobre un soporte de información. La eficiencia en la utilización del archivo depende de la organización del mismo; por ello se debe optar por una u otra organización atendiendo a la forma en que se va a usar el archivo. Las principales organizaciones de archivos son:

Según su organización	
Secuencial	No es más 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).
Secuencial indexada	Un archivo con esta organización consta de tres áreas: · Área de índices · Área primaria · Área de excedentes (*overflow*) Ventaja: a) Rápido acceso, y, además, el sistema se encarga de relacionar la posición de cada registro con su contenido por medio del área de índices. b) Gestiona las áreas de índices y excedentes. Desventajas: a) necesidad de espacio adicional para el área de índices. b) el desaprovechamiento de espacio que resulta al quedar huecos intermedios libres después de sucesivas actualizaciones.

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: Rapidez de acceso a un registro cualquiera.

7.2.1 Archivos en C

Manejo de archivos en lenguaje C

El manejo de archivos es una habilidad esencial para los desarrolladores de software, ya que les permite interactuar con datos externos al programa. Al utilizar el manejo de archivos en lenguaje C, puedes leer datos de entrada desde un archivo, procesarlos y generar resultados en otro archivo. Esto es especialmente útil en aplicaciones que manejan grandes volúmenes de información o necesitan almacenar datos para su uso posterior.

El manejo de archivos en lenguaje C también es crucial para el desarrollo de sistemas de bases de datos, compiladores y cualquier programa que requiera almacenamiento de información de forma persistente.

El manejo de archivos en lenguaje C ofrece varias ventajas importantes:

1. **Persistencia de datos:** Los archivos permiten que los datos persistan más allá de la duración de un programa, lo que significa que se pueden acceder en futuras ejecuciones.
2. **Almacenamiento de grandes volúmenes de datos:** El manejo de archivos es especialmente útil cuando se trabaja con grandes volúmenes de información que no caben en la memoria principal.
3. **Compartir información entre programas:** El almacenamiento de datos en archivos facilita el intercambio de información entre diferentes programas, incluso escritos en diferentes lenguajes.
4. **Organización de datos:** Los archivos permiten organizar los datos en estructuras específicas, lo que facilita su lectura y escritura.
5. **Recuperación de datos:** En caso de fallos o interrupciones en el programa, los datos almacenados en archivos pueden ser recuperados y restaurados.

Tipos de archivos

Hay dos tipos de archivos, archivos de texto y archivos binarios.

Un **archivo de texto** es una secuencia de caracteres organizadas en líneas terminadas por un carácter de nueva línea. En estos archivos se pueden almacenar canciones, fuentes de programas, base de datos simples, etc. Los archivos de texto se caracterizan por ser planos, es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho.

Un **archivo binario** es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo. Así que no tendrá lugar ninguna traducción de caracteres. Además, el número de bytes escritos (leídos) será el mismo que los encontrados en el dispositivo externo. Ejemplos de estos archivos son Fotografías, imágenes, texto con formatos, archivos ejecutables (aplicaciones), etc.-

Manejo de archivos en C

En c, un archivo es un concepto lógico que puede aplicarse a muchas cosas desde archivos de disco hasta terminales o una impresora. Se asocia una secuencia con un archivo específico realizando una operación de apertura. Una vez que el archivo está abierto, la información puede ser intercambiada entre este y el programa.

Se puede conseguir la entrada y la salida de datos a un archivo a través del uso de la biblioteca de funciones; C no tiene palabras claves que realicen las operaciones de E/S. La siguiente tabla da un breve resumen de las funciones que se pueden utilizar. Se debe incluir la librería STDIO.H. Observe que la mayoría de las funciones comienzan con la letra “F”, esto es un vestigio del estándar C de Unix.

7.3 Operación con archivos

Un programa no puede trabajar directamente con la información contenida en un fichero. Las operaciones sobre ficheros se hacen en base a llamadas a funciones del sistema operativo, que es quien controla directamente el acceso al sistema de ficheros y al hardware de almacenamiento secundario.

Son muchas las operaciones que podemos realizar sobre un fichero, todas se basan en las siguientes, llamadas operaciones básicas.

A) Operaciones básicas

- **Apertura**: es necesario abrir un fichero para poder trabajar con él, de esta forma se permite el acceso a los datos y se pueden realizar las operaciones de lectura y escritura necesarias. Es conveniente que un fichero sólo permanezca abierto durante el tiempo estrictamente necesario, este es, el justo para realizar las operaciones anteriormente indicadas.
- **Cierre**: Una vez finalizadas las operaciones efectuadas sobre un fichero, éste debe permanecer cerrado para limitar el acceso a los datos y evitar así un posible deterioro o pérdida de información.
- **Creación**: crear un fichero es generarlo, esto es, almacenar sobre el soporte seleccionado la información requerida para su posterior tratamiento.
- **Consulta**: a través de las consultas es posible acceder, si existen, a los registros de un fichero y conocer el contenido de sus campos.

- **Actualización**: con esta operación realizamos la inserción, modificación o eliminación de registros, de esta forma, cambiamos algunos registros con información antigua por otros con nueva información.
- **Borrado**: es la operación inversa a la creación, por tanto, eliminamos físicamente el fichero, perdiendo toda la información almacenada en el mismo y liberando el espacio de memoria que ocupaba.

B) Operaciones compuestas

Las siguientes operaciones son el resultado de componer varias de las operaciones básicas anteriormente descritas que podemos realizar sobre un fichero:

- **Copia**: también llamada *duplicado*, es el resultado de crear un fichero con los mismos datos que otro. Podemos descomponerla en una creación de un nuevo archivo, una consulta del archivo que contiene los datos y una actualización de inserción del primero.
- **Concatenación**: Partiendo de la existencia de dos ficheros con la misma estructura, se genera uno nuevo que contiene los datos de los otros dos, como resultado de colocar uno detrás del otro. De esta forma la concatenación se puede descomponer en la creación de un fichero nuevo, la consulta de otros dos ficheros que ya existían y la actualización por inserción del fichero nuevo.
- **Partición**: también denominada *rotura*, permite obtener varios ficheros a partir de uno inicial en función de alguna de las características internas de sus campos. La partición se puede descomponer en la creación de varios ficheros nuevos, la consulta de uno ya existente y la actualización por inserción en los ficheros nuevos.
- **Fusión**: también conocida como *mezcla*, es similar a la concatenación. A partir de dos o más ficheros existentes creamos uno nuevo que contiene los datos de los anteriores, pero en este caso, en vez de colocar los registros de forma secuencial, los ordenamos siguiendo un criterio. La fusión puede descomponerse en una creación de un nuevo fichero, la consulta de varios ficheros ya existentes y, a continuación, la actualización por inserción del fichero nuevo.
- **Clasificación**: u *ordenación*, permite establecer un orden entre los registros almacenados dentro del fichero. Dicha ordenación puede ser ascendente o descendente. La descomposición parte, en primer lugar, de la creación de un nuevo fichero, seguidamente, realizamos consultas en el fichero original para ir actualizando por inserción en el fichero nuevo; por último, borramos el fichero original.
- **Reorganización**: también llamada *compactación* o *empaquetamiento*, se usa en un fichero que se va usando con el tiempo, en el cual se van eliminando registros, añadiendo nuevos, etc. A largo plazo, esto va creando huecos en el fichero, y por tanto, habrá registros vacíos. Con la reorganización se intenta reubicar los registros de un fichero aprovechando estos huecos generados por el uso. De esta forma, la descomposición cosiste en realizar sucesivas consultas y actualizaciones, tanto de borrado como de inserción.

7.3.1 Operaciones en C

La tabla siguiente da un breve resumen de las funciones que se pueden utilizar.

Observe que la mayoría de las funciones comienzan con la letra **"f"** (*file*), esto es un vestigio del estándar C de Unix.

Nombre Función

`fopen()`	Abre un archivo.
`fclose()`	Cierra un archivo.
`fgets()`	Lee una cadena de un archivo.
`fputs()`	Escribe una cadena en un archivo

`fseek()`	Busca un byte específico de un archivo.
`fprintf()`	Escribe una salida con formato en el archivo.
`fscanf()`	Lee una entrada con formato desde el archivo.
`feof()`	Devuelve cierto si se llega al final del archivo.
`ferror()`	Devuelve cierto si se produce un error.
`rewind()`	Coloca el localizador de posición del archivo al principio del mismo.
`remove()`	Borra un archivo.
`fflush()`	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 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;

7.3.a. 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(`*const char nombre_archivo, cost charmodo*`);`

Donde *nombre_archivo* es un puntero a una cadena de caracteres que representan 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.
r+	Abre un archivo de texto para lectura / escritura.
w+	Crea un archivo de texto para lectura / escritura.
a+	Añade o crea un archivo de texto para lectura / escritura.

r+b	Abre un archivo binario para lectura / escritura.
w+b	Crea un archivo binario para lectura / escritura.
a+b	Añade o crea un archivo binario para lectura / escritura

La función `fopen`() devuelve un puntero a archivo. Un programa nunca debe alterar el valor de ese puntero. Si se produce un error cuando se está intentando abrir un archivo, `fopen`() devuelve un puntero nulo.

Se puede abrir un archivo bien en modo texto o binario. En la mayoría de las implementaciones, en modo texto, la secuencias de retorno de carro / salto de línea se convierten a caracteres de salto de línea en lectura. En la escritura, ocurre lo contrario: los caracteres de salto de línea se convierten en salto de línea. Estas conversiones no ocurren en archivos binarios.

7.3.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 encuentre en el buffer en el 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.

Ejemplo #1: Programa en C para abrir un fichero.

```
//AperturaArchivo.c
#include <stdio.h>
#include <stdlib.h>
int main()
{
      FILE *fp;
      fp=fopen("archivo.txt","r");
      if(fp==NULL)
      {
            printf("Error al abrir el archivo para leer");
            exit(1);
      }
      fclose(fp);
      return 0;
}
```

7.3.c. Leer y grabar datos en un archivo (modo texto)

Para introducir u obtener datos de un archivo tenemos las siguientes cuatro funciones:

`fprintf`() y `fscanf`()

Estas funciones se comportan exactamente como `prinft`() y `scanf`() discutidas anteriormente, excepto que operan sobre archivo. Sus prototipos son:

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

Las funciones `fgets`() y `fputs`() 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 `puts`() escribe la cadena a un archivo especifico. 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.

Ejemplo #2: Programa en C que lee dos caracteres de un archivo y los copia en otro.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      FILE *fp1,*fp2;
      char c1,c2;
      fp1=fopen("archivo.txt","r");
      if(fp1==NULL)
      {
            printf("Error al abrir el archivo para leer");
            system("PAUSE");
            exit(1);
      }
      fp2=fopen("copia.txt","w");
      if(fp2==NULL)
      {
            printf("Error al abrir el archivo copia.txt");
            system("PAUSE");
            exit(1);
      }
      fscanf(fp1,"%c%c",&c1,&c2);
      fprintf(fp2,"%c%c",c1,c2);
      fclose(fp1);
      fclose(fp2);
      return 0;
}
```

7.3.d. Funcion feof()

Cuando se abre un archivo para entrada binaria, se puede leer un valor entero igual de la marca EOF. Esto podría hacer que la rutina de lectura indicase una condición de fin de archivo aun cuando el fin físico del mismo no se haya alcanzado. Para resolver este problema, C incluye la función `feof`(), que determina cuando se ha alcanzado el fin del archivo leyendo datos binarios. La función tiene el siguiente prototipo:

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

Su prototipo se encuentra en STDIO.H. Devuelve cierto si se ha alcanzado el final del archivo, en cualquier otro caso, 0. Por supuesto, se puede aplicar este método a archivos de texto también.

7.3.e. Función ferror()

La función `ferror`() determina si se ha producido en 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. Esta función se encuentra en STDIO.H

7.3.f. Función remove()

La función `remove`() borra el archivo especificado. Su prototipo es el siguiente:

```
int remove(char *nombre_archivo);
```

Devuelve cero si tiene éxito. Si no un valor distinto de cero.

7.3.g. Función fflush()

La función `fflush`() 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.

7.3.h Manejo del desplazamiento sobre el archivo: fseek()

Esta función está sitúa el puntero (cursor) de un archivo en una posición deseada, trabaja para modo binario y texto.

```
int fseek(FILE *archivo, long offset, int origen);
```

`archivo` es un puntero a la estructura del archivo offset es un entero largo que especifica el número de bytes de origen, donde se colocará el cursor.

`origen` es un número entero que especifica la posición de origen.

Puede ser:

- SEEK_SET: El origen es el comienzo del archivo
- SEEK_CUR: El origen es la posición actual
- SEEK_END: El origen es el final del archivo

Ejemplo #3: Programa para desplazamiento sobre el archivo

```
#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 );
}
```

7.4 Corte de Control

La idea básica de este algoritmo es poder analizar información, generalmente provista mediante *registros*, agrupándolos según diversos criterios. Como precondición se incluye que la información debe estar ordenada según los mismos criterios por los que se la quiera agrupar. De modo que si varios registros tienen el mismo valor en uno de sus *campos*, se encuentren juntos, formando un grupo.

Se lo utiliza principalmente para realizar reportes que requieren subtotales, cantidades o promedios parciales u otros valores similares.

El algoritmo consiste en ir recorriendo la información, de modo que cada vez que se produzca un cambio en alguno de los campos correspondiente a uno de los criterios, se ejecutan los pasos correspondientes a la finalización de un criterio y el comienzo del siguiente.

7.4.1 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.

Ejemplo:

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.

7.4.2 Requisitos para usar corte de control

1. Ordenamiento de los datos de entrada
2. Que existan varios subconjuntos para que tenga sentido el corte de control.
3. Que cada subconjunto tenga varios elementos o registros.

(El corte se produce en un archivo).

7.4.3 Proceso para el corte de control

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.

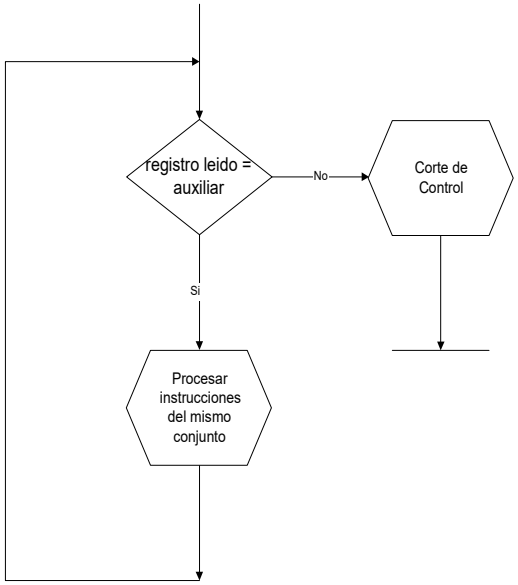
Por lo tanto para saber si el campo de control del registro recién leído tiene el mismo contenido que el registro anterior, ***será necesario haber almacenado en una variable auxiliar el contenido del campo de control del primer registro del grupo o bloque***, para poder compararlo con él, y de esta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

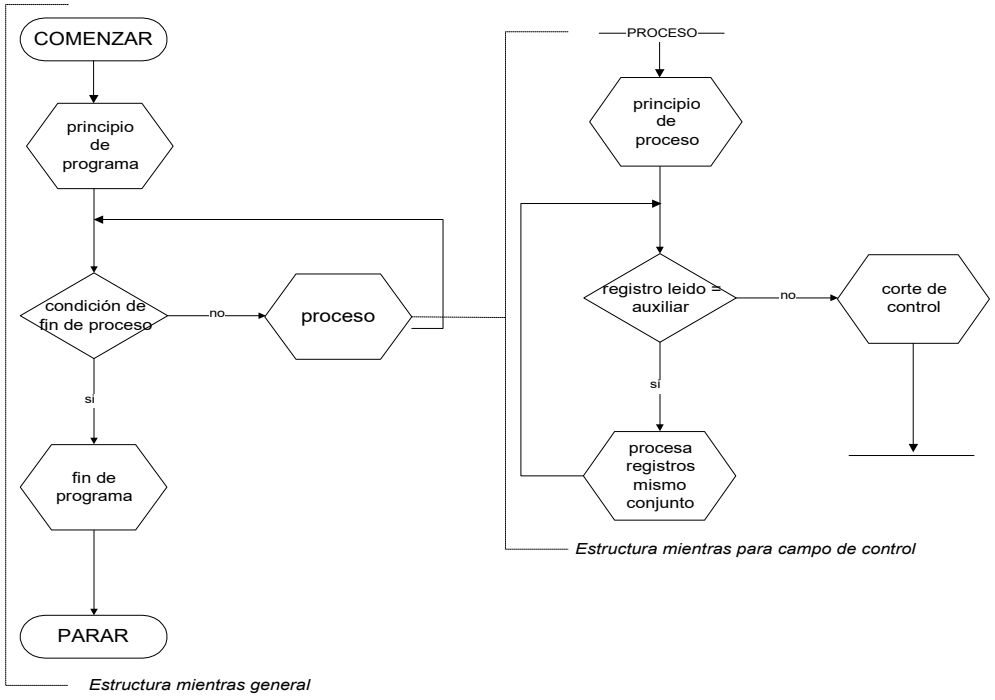
Cuando esto se produce, estaremos seguros que el bloque o conjunto de datos que estábamos procesando han finalizado. Es este el momento de realizar las operaciones relacionadas con la finalización del conjunto (impresión de contadores, impresión de acumuladores, acumular para totales generales, etc.-)

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.

a) Habrá una estructura repetitiva general cuya condición de salida será la finalización del archivo (EOF). Debe realizarse una primera lectura antes de entrar a la estructura.
b) Antes de entrar a la estructura repetitiva general, se inicializarán los acumuladores y contadores.
c) Se deberá inicializar variables relacionadas con el comienzo de cada bloque o conjunto de control.
d) Las operaciones necesarias relativas a la finalización de un bloque de control se realizarán inmediatamente después de salir de la estructura asociada.
e) Se debe realizar lecturas como última instrucción de la estructura repetitiva interna.



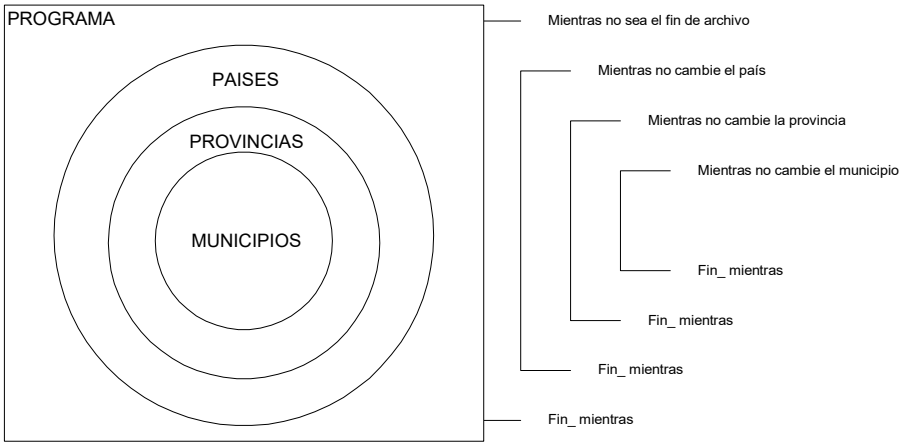


Jerarquía de comparación

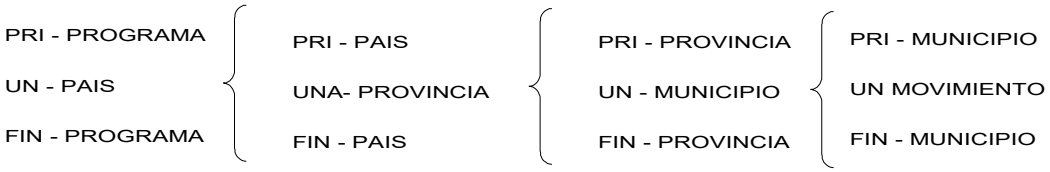
Depende del enunciado del problema. O sea cual es la división más importante y consecuentemente el diseño de salida de los resultados buscados.

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



7.4.4 Tipos de cortes

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 primera 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 más interna la que representa a la entidad contenida en las demás.

7.5 Operaciones de altas, bajas, modificaciones y consultas a Archivos Directos.

7.5.1 Apareo de archivos

Así como el corte de control nos sirve para generar un reporte, el apareo nos sirve para asociar/relacionar datos que se encuentran en distintos archivos.

La idea básica es: a partir de dos archivos (uno principal y otro relacionado) que tienen alguna información que los enlace, generar un tercero (o una salida por pantalla), como una mezcla de los dos.

Para hacer esto es conveniente que ambos archivos estén ordenados por el valor que los relaciona.

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).

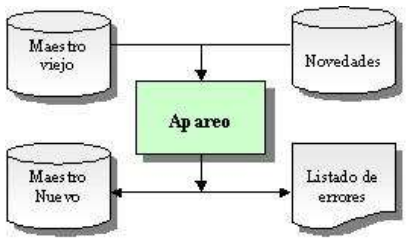
Todos los archivos deberán tener un campo común que es el nexo para realizar el proceso.

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.

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.



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.

Requisitos

1. **Ordenamiento**. Todos los archivos deben estar ordenados por el mismo campo clave de control (nexo común) de la misma forma.
2. **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.
3. **Homogeneidad**. Todos los archivos deben ser de entes de la misma naturaleza.

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 más 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.

7.5.2 Casos de apareo

CASO 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;
```

CASO 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;
```

CASO 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 vacío);
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_MOV;
```

CASO 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
LEER_MOV;
```

CASO BAJA: Eliminar un registro del maestro

```
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;
```

CASO ALTA: Incorporar un registro del maestro

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

PROCEDURE PRI_ENTE;
** Conjunto de instrucciones para iniciar el proceso de ALTAS (puede estar vacío);

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 está en el maestro, hay que incorporarlo (alta).
   LEER_MOV;

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

PROCEDURE ALTA;
** conjunto de instrucciones para dar de alta el registro;
```

Desarrollo de un caso

Enunciado

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:

Archivo Maestro. Un registro por cada alumno.

Archivo Novedad. Uno o ningún registro por alumno**.**

Se desea:

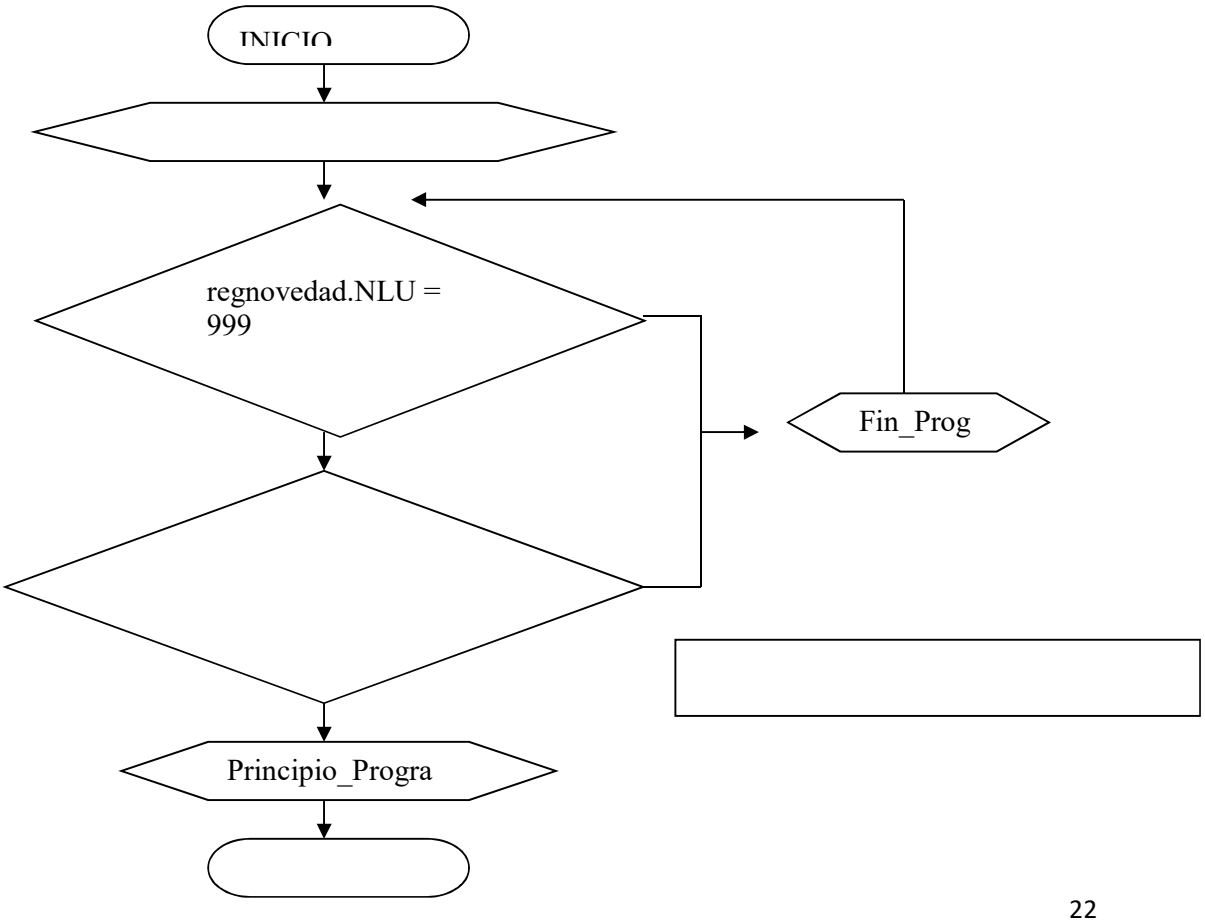
A) Actualizar la fecha de aprobación de la última asignatura
B) Actualizar la cantidad de asignaturas aprobadas
C) Listar los datos y la asignatura aprobada

Detectar el caso de apareo

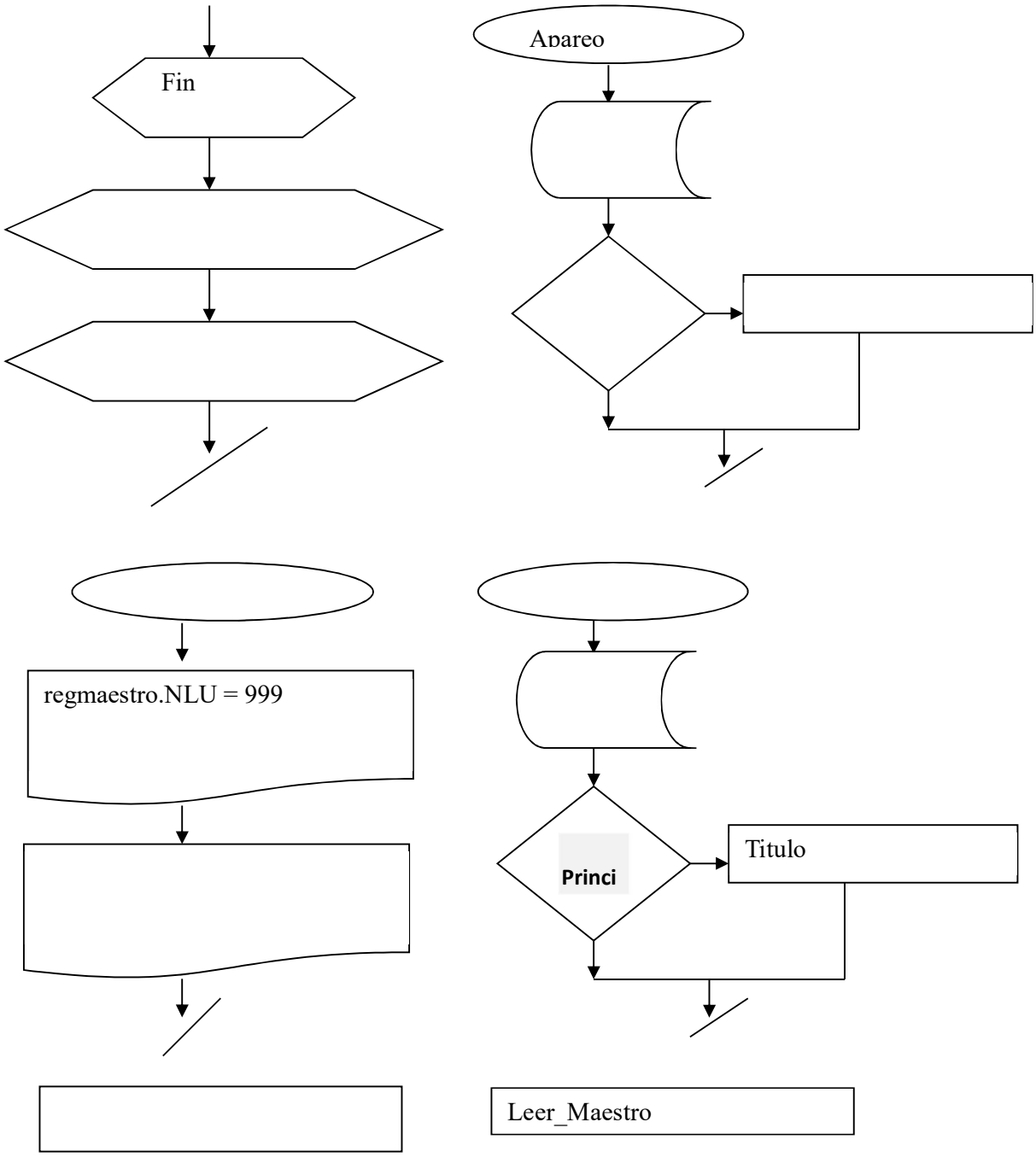
En el Maestro están todos?	SI
Puede faltar registros en el Maestro?	NO
En la Novedad, viene un registro por cada Maestro?	NO
En la Novedad, viene uno, varios o ningún registro por cada Maestro?	NO
En la Novedad, viene uno o ningún registro por cada Maestro?	SI

Desarrollo del algoritmo

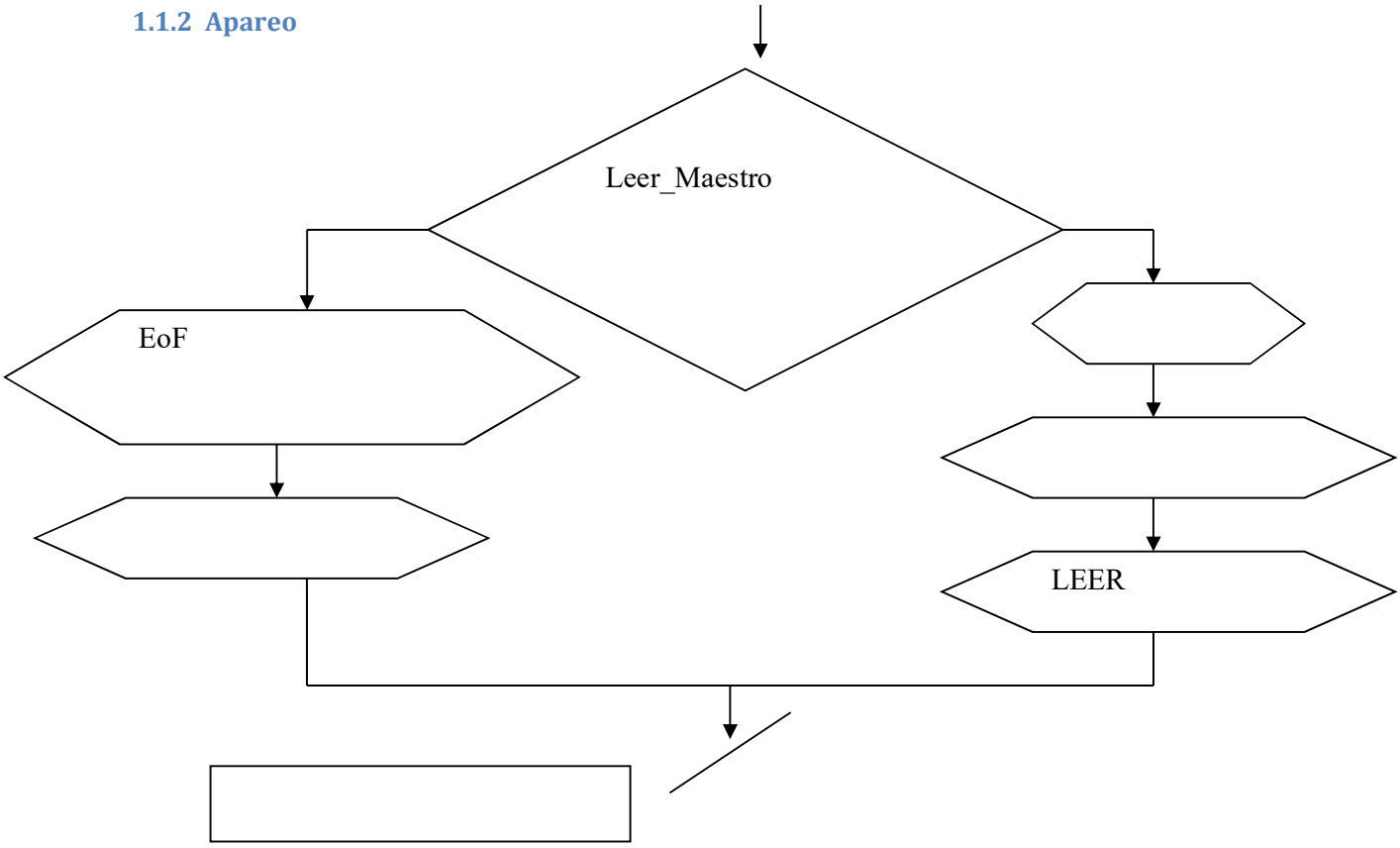
1.0 Esquema principal



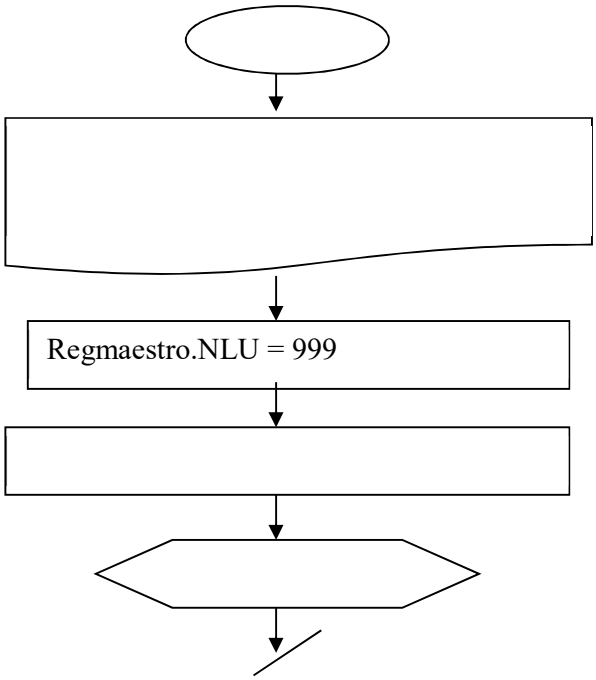
1.1.1 Principio_Programa



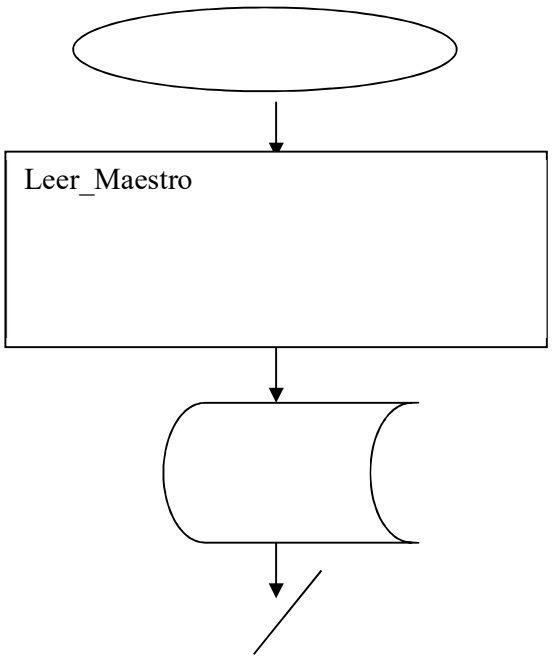
1.1.2 Apareo



1.1.2.1 MCD

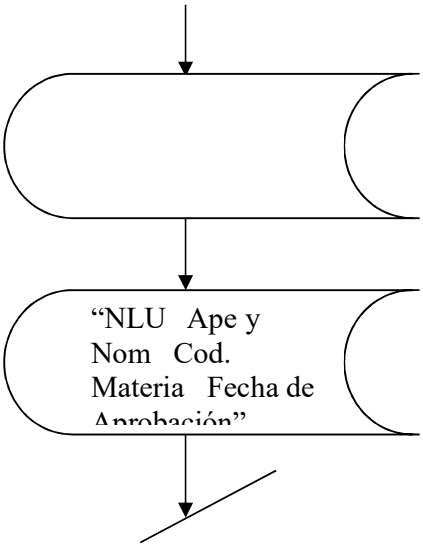


1.1.2.2 Grabar_MaeNue



"Info. de actualización de materias aprobadas por los alumnos"

1.1.3 Fin Programa



Bibliografia

- 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. Capítulo 12.
- PROGRAMACIÓN; Castor F. Herrmann, María E. Valesani.; 2001; Editorial: MOGLIA S.R.L..ISBN: 9874338326. Capítulo 2.