Elementos de Programación

UNIDAD 10. MANEJO DE ARCHIVOS

INDICE

UNIDAD 10 - MANEJO DE ARCHIVOS

OBJETIVOS: *Realizar programas que puedan almacenar datos y resultados en una memoria no volátil a fin de respaldar la información generada permitiendo construir programas que no pierdan los datos cargados al finalizar su ejecución.*

1. Que es un archivo binario.

Los datos con los que se han trabajado hasta el momento han residido en la memoria principal. Sin embargo, las grandes cantidades de datos se almacenan normalmente en un dispositivo de memoria secundaria. Estas colecciones de datos se conocen como archivos (en ingles **FILE**).

Un **archivo** es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas **estructura de datos** (registros) que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajos denominadas **miembros** (campos).

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.

Ejemplo de una Estructura de Datos:

Estructura **PERSONA**					
DNI	Apellido y Nombre	Sexo	Dia Nacim.	Mes Nacim	Año Nacim

Ejemplo de un Archivo “datos.dat” que utiliza la Estructura de Datos antes descripta:

Estructura **PERSONA**	Estructura **PERSONA**	Estructura **PERSONA**	**NULL**

Ejemplo de un Archivo “datos.dat” que utiliza los miembros de la Estructura de Datos antes descripta:

DNI Apellido y Nombre Sexo Dia Nacim. Mes Nacim Año Nacim	**DNI Apellido y Nombre Sexo Dia Nacim. Mes Nacim Año Nacim**	**NULL**

En el **Lenguaje 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 sentencia 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; el **Lenguaje 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 biblioteca **<stdio.h >**. Observe que la mayoría de las funciones comienzan con la letra "F".

fopen ()	Abre un archivo.
fclose ()	Cierra un archivo.
fread ()	Lee un registro del archivo.
fwrite ()	Guarda un registro en el archivo.
feof()	Devuelve un número distinto de 0 si se llega al final del archivo.

2. Variable puntera 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 sentencia 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 de dato **FILE** que se define en **<stdio.h>**. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos. Para declarar una variable de este tipo se utiliza la siguiente sentencia:

FILE * identificador;

Por ejemplo, si se desea declarar un puntero llamado pf, se escribe:

FILE * pf;

Si es necesario declarar más de un puntero a archivo a cada uno se le debe anteponer el asterisco. El siguiente ejemplo define dos punteros pf y pf2.

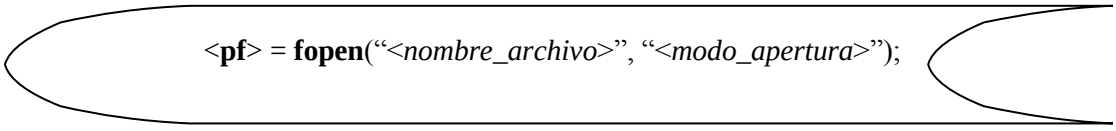
FILE *pf, *pf2;

3. Como Abrir un Archivo. Apertura de un archivo (función fopen ())

Antes de utilizar un Archivo en un programa, ya sea para escribir datos en él y/o para leer datos de él, el Archivo debe ser abierto, con lo que se establece un canal de comunicación entre el programa y el Archivo. Y la función para abrir un Archivo se denomina **fopen ()** y su sintaxis es:

<**pf**> = **fopen**("<nombre_archivo>", "<modo_apertura>");

En diagrama de lógica:



Como puede verse, esta función devuelve un puntero a Archivo, <**pf**>, que será el puntero que quede asociado al Archivo abierto hasta que sea cerrado. Todas las operaciones sobre el Archivo se

realizarán a través de ese puntero <**pf**>. Para indicar el Archivo que se desea abrir se usa la cadena de caracteres < **nombre_archivo** >, la cual puede incluir, además del nombre del Archivo, la ruta completa donde está ubicado, es decir la unidad de almacenamiento y la carpeta. No olvidar que la barra invertida (\) de la ruta debe escribirse dos veces (\\), porque si se escribe sólo una vez es interpretada como secuencia de escape. Sino se coloca la ruta el archivo se busca en la misma carpeta donde se encuentra el ejecutable. Para que el programa pueda ser ejecutado en distintas computadoras sin errores se recomienda no poner una ruta fija.

Con la cadena <**modo_apertura**> se indicará si se va a usar el archivo en modo texto o en modo binario, además servirá para especificar si se va a leer del archivo, se va a escribir en él o ambas cosas. En esta materia utilizaremos archivos binarios y sobre estos archivos se pueden emplear los siguientes modos de apertura:

Modo de apertura	
Modo	***Significado***
rb	Abre un archivo para lectura (read)
wb	Crea un archivo para escritura (write)
ab	Abre un archivo para añadir (add)
r+b	Abre un archivo para lectura/escritura
w+b	Crea un archivo para lectura/escritura
a+b	Abre o crea un archivo para lectura/escritura

Como puede verse en la tabla, el <modo_apertura> indicará qué debe hacer el sistema con el archivo, dependiendo de si existe o no. Por ejemplo, abrir para lectura un archivo que no exista producirá un error. Además, el <modo_apertura> también establece donde apuntará el puntero al abrir el fichero, al principio o al final de este. Por ejemplo, si deseamos añadir datos al archivo, el puntero deberá colocarse al final, mientras que, si queremos leer todos los datos, deberá colocarse al principio. Por otro lado, un archivo abierto solo para lectura no permitirá realizar escrituras en él, a no ser que se cierre y se vuelve a abrir para escritura. En la siguiente tabla se concretan todos estos casos.

Modo	Acción	Archivo ya existe	Archivo no existe	Lectura	Escritura	Posición Puntero
rb	Lectura	Correcto	*** Error ***	Correcto	*** Error ***	Principio
wb	Escritura	Borra contenido	Se crea	*** Error ***	Correcto	Principio
ab	Añadir	Correcto	Se crea	*** Error ***	Correcto	Final
r+b	Lect/Esc	Correcto	*** Error ***	Correcto	Correcto	Principio
w+b	Lect/Esc	Borra contenido	Se crea	Correcto	Correcto	Principio
a+b	Lect/Añadir	Correcto	Se crea	Correcto	Correcto	Principio

En la tabla se observa que la apertura de un Archivo con el modo “wb” puede ser peligrosa, ya que si el Archivo existe se perderán todos sus datos. Aunque hay situaciones en las que es esa acción precisamente la que se quiere realizar, destruir todos los datos previos del Archivo. Por otra parte, debe tenerse en cuenta que si se utiliza el modo de apertura “ab” o “a+b”, los datos nuevos que se escriban en el Archivo nunca sobrescriben otros datos, sino que siempre se añaden al final, aunque el puntero esté en otra posición. El puntero apuntará al final después de añadir los nuevos datos. Además, al abrir un Archivo con el modo de apertura “ab”, el puntero inicialmente se coloca al principio, aunque después de añadir algún dato el puntero automáticamente pasa al final.

Nota: En esta materia se utilizan solo los tres primeros modos de apertura (rb, wb y ab) el resto de los modos se verán en materias siguientes y requieren del uso de la función fseek para posicionar el puntero en el lugar deseado del archivo para leer o escribir.

Si la función fopen () tiene éxito, es decir abre el Archivo sin ningún problema, devolverá el puntero al Archivo. Por el contrario, si se ha producido algún error al intentar la apertura devuelve el valor NULL. Por tanto, después de ejecutar la función fopen (), el programa deberá comprobar siempre si el puntero devuelto vale NULL.

Ejemplos:

```
FILE *fp;
//Especificando unidad y nombre de Archivo
if((fp = fopen("C:\\cursos.dat", "r+b")) == NULL)
{
      printf ( "ERROR" );
      getch();
      exit(1);
}
```

```
//Especificando unidad, carpeta y nombre de Archivo
if ((fp =fopen("C:\\BC\\cursos.dat","a+b")) == NULL)
{
      printf ( "ERROR" );
      getch();
      exit(1);
}
```

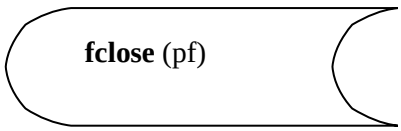
4. Como Cerrar un Archivo. Cierre de un archivo (función fclose ())

Cuando un programa deja de necesitar un Archivo, éste debe cerrarse, cortándose por tanto el canal de comunicación entre el programa y el Archivo. Para cerrar un Archivo se utiliza la función **fclose ()**.

La función **fclose ()** cierra un Archivo que fue abierto 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 un Archivo puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa, cuya sintaxis es:

```
fclose (pf);
```

En Diagrama de Lógica:



Donde "pf" es el puntero al archivo devuelto por la llamada a **fopen ()**. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

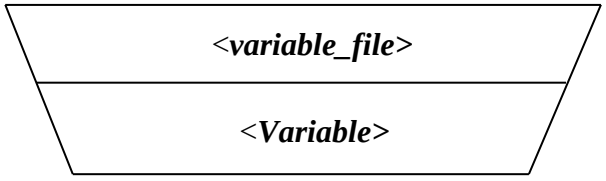
5. Como leer un registro de un archivo. Función de Entrada (función fread ())

La función **fread ()** trabaja con registros de longitud constante. Es capaz de leer desde un Archivo, uno o varios registros de la misma longitud y a partir de una dirección de memoria determinada. Se debe asegurar de que haya espacio suficiente para contener la información leída. La sintaxis de esta función es:

En lenguaje C:

```
fread( <Dir_Variable> , < n°_bytes>,  <n°_cuenta>,  <variable_file>);
```

En diagrama de lógica:



La función **fread ()** significa que se leerá del Archivo que apunta la <variable_file> el número de bytes <nº_bytes> tantas veces como indique <nº_cuenta>, dejando dichos bytes grabados en

memoria a partir de la dirección <Dir_Variable>. Esta dirección será la de una variable con el tamaño suficiente para guardar los datos leídos. El número de bytes que se leen será por tanto el resultado del producto: <nº_bytes> * <nº_cuenta>.

Ejemplos:

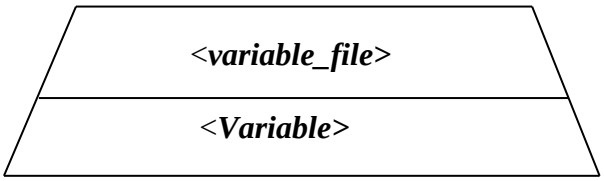
```
int num;
fread(&num, sizeof(int), 1, fp);
//Se leen 4*1=4 bytes del archivo, que caben en num.
float Notas[5];
fread(Notas, sizeof(float), 5, fp);
//Se leen 4*5=20 bytes del fichero, que caben en Notas
```

En el Ejemplo N°1, en el segundo caso se leerán 20 bytes, ya que se leerán 5 veces el número de bytes indicado por sizeof(float), o sea 4*5 = 20. Por tanto, se leen 5 números float desde el archivo y se guardan en el array Notas, cuyo tamaño es de 20 bytes.

6. Como escribir un registro de un archivo. Función de Salida *(función fwrite())*

La función **fwrite()** permite escribir datos en un archivo como una sucesión de bytes. La sintaxis de esta función es:

```
fwrite( <Dir_Variable> , < n°_bytes>,  <n°_cuenta>,  <variable_file>);
```



A partir de la dirección de memoria <Dir_Variable> se leerán tantos bytes como se indique en <nº_bytes> tantas veces como se especifique en < nº_cuenta> y se escribirán en el Archivo que apunta la < variable_file >. El número de bytes escritos será por tanto el resultado del producto: <nº_bytes> * < nº_cuenta>.

Ejemplos:

```
int num=1067;
fwrite(&num, sizeof(int), 1, fp);
//Graba 4 bytes en el archivo, desde la dirección num, por tanto el
valor 1067 queda escrito en el archivo.
float Notas[5]={5.5, 7.25, 8.5, 4.75, 9.5};
fwrite(Notas, sizeof(float), 5, fp);
//Se escriben 4*5=20 bytes en el archivo, desde la dirección Notas,
quedando las 5 notas grabadas en el archivo.
```

La función **fwrite()** devuelve el número de veces que ha escrito el <nº_bytes>, que no siempre coincide con <cuenta>, ya que puede producirse algún error. Por tanto, si el valor que devuelve **fwrite()** no coincide con <cuenta> es que ha ocurrido un error.

Ejemplo 1. Escribir un número real en un Archivo. Leerlo en otra variable real.

```
FILE *fp;
float var;
float num = 12.23;
if ((fp = fopen("prueba.dat", "wb")) == NULL )
```

```
      {
         printf("No se puede abrir.\n");
         getch(); exit(1);
      }
      fwrite( &num, sizeof(float), 1, fp);
      fclose(fp);
      if ((fp = fopen("prueba.dat", "rb")) == NULL )
      {
         printf("No se puede abrir.\n");
         getch(); exit(1);
      }
      fread (&var, sizeof(float), 1, fp );
      fclose(fp);
```

Uno de los usos más comunes de **fread()** y **fwrite()** es el manejo de Archivo en forma de conjunto de registros, donde cada registro está dividido en campos. Para ello en el programa debe definirse un tipo de estructura de datos con el mismo formato que el registro del Archivo, con el objeto de leer y escribir registros completos con las funciones **fread()** y **fwrite()** respectivamente, en lugar de leer y escribir campos concretos.

Ejemplo 2. Teniendo un vector de estructuras ya cargado con datos, deberá grabarse en un Archivo y después dejarlo en otro vector de estructuras.

```
struct datos
{
   char nombre[20];
   char apellido[40];
};

int main ()
{
      FILE *fp;
      int i;
      struct datos lista1[30], lista2[30];
      //Suponemos que lista1 ya contiene datos.

      if ( (fp = fopen("fich.dat", "wb")) == NULL )
      {
                  printf("No se puede abrir.\n");
                  getch(); exit(1);
      }
      //Escribe 30 registros desde el array lista1.
      for ( i = 0; i < 30; i++)
                  fwrite(&lista1[i],sizeof(struct datos),1,fp)
      fclose(fp);
      if ( (fp = fopen("fich.dat", "rb")) == NULL )
      {
                  printf("No se puede abrir.\n");
                  getch(); exit(1);
      }

      //Lee 30 registros, dejándolos en el array lista2.
      for ( i = 0; i < 30; i++)
                  fread(&lista2[i], sizeof(struct datos),1,fp)
      fclose(fp);
      return 0;
}
```

7. Como leer un archivo sin conocer la cantidad de registros. Función feof().

Cuando se abre un archivo para lectura binaria, se puede leer un valor entero igual al de la marca de fin de archivo (EOF). Esto podría hacer que la rutina de lectura indicase una condición de fin de archivo aún cuando el fin físico del mismo no se haya alcanzado. Para resolver este problema, **lenguaje 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 *);
```

Su prototipo se encuentra en <stdio.h>. Devuelve un número distinto de 0 si se ha alcanzado el final del archivo, si aún hay datos devuelve un 0.

Cuando no se conoce la cantidad de registros que posee un Archivo, se puede utilizar la función **feof()**, la cual permitirá leer la información hasta el fin de Archivo (ver **Ejemplo 3**).

Esta función debe utilizarse **luego** de hacer una lectura sobre el archivo con fread.

Ejemplo 3:

```
struct datos
{
   char nombre[20];
   char apellido[40];
};

int main ( )
{
      FILE *fp;
      int i;
      struct datos lista[30];

      // Se supone que el archivo no va a contener más de 30 registros.

      if ( (fp = fopen("fich.dat", "rb")) == NULL )
      {
         printf("No se puede abrir.\n");
         getch(); exit(1);
      }
      //Lee los registros del archivo hasta el final del mismo.
      i = 0;
      fread(&lista[i], sizeof(struct datos),1,fp)
      while (  !feof (fp)  )
      {
         i++;
         fread(&lista[i], sizeof(struct datos),1,fp)
      }
      fclose(fp);
      return 0;
}
```

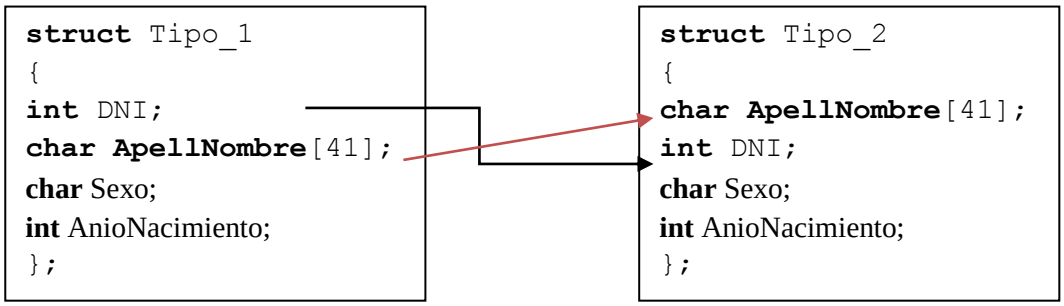
8. Ejemplos del uso de archivos con estructuras de datos

El uso de estructuras de datos en el manejo de “archivos binarios”, resulta imprescindible pues la esencia de estos archivos binarios es la grabación y recuperación de registros es decir estructuras de datos (**struct**).

Tanto para la recuperación (lectura) o grabación (escritura) de un registro del archivo el mismo se realiza a través de una variable tipo estructura (**struct**) que obedezca al diseño del registro del archivo.

Es importante tener en cuenta que para poder recuperar (lectura) o grabar (escritura) la información que se encuentra en un archivo se debe conocer exactamente como está formada la estructura, porque será la única manera que se pueda trabajar con archivos. En el ejemplo que se desarrolla a continuación se corresponden a dos estructuras distintas, aunque ambas posean los mismos datos.

La estructura "**struct** Tipo_1" y la estructura "**struct** Tipo_2" contienen los mismos datos y la misma cantidad de bytes (50 bytes), pero son al mismo momento diferentes porque la información dentro de ambas se encuentra en órdenes diferentes. Para que dos estructuras sean exactamente iguales deben coincidir totalmente y no es el caso anterior por lo cual son distintas e incompatibles en su tratamiento.



Ejemplo 4:

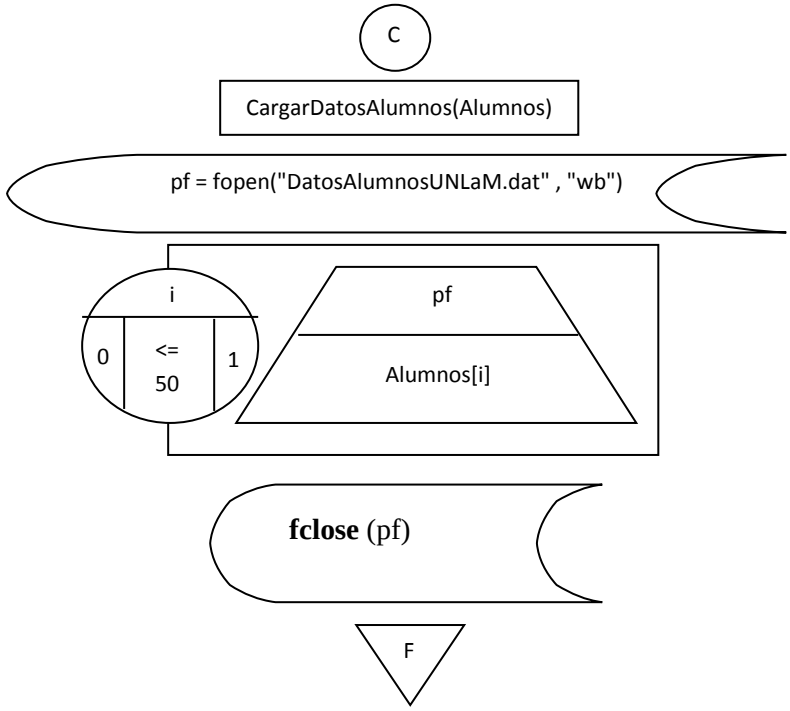
Es un programa que permite grabar los datos de 10 alumnos en un archivo llamado "AlumnosUNLaM.dat".

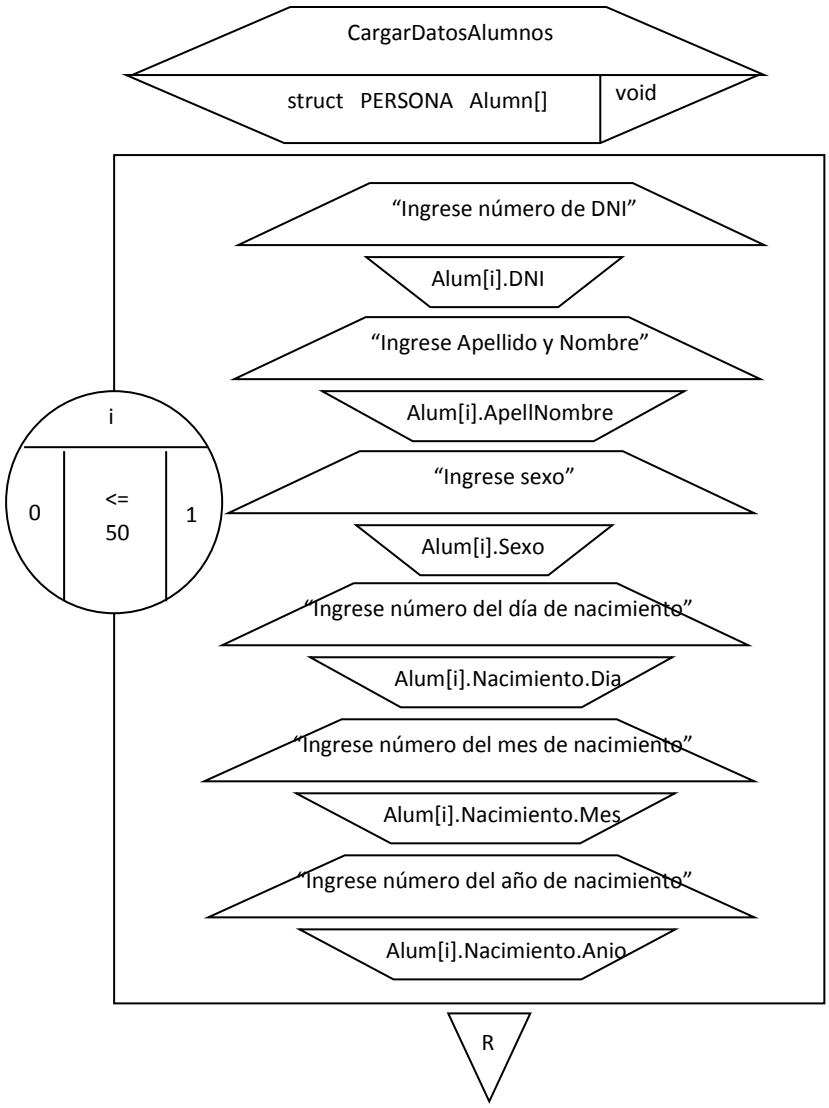
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};
struct PERSONA
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   struct FECHA Nacimiento;
};
int main()
{
   struct PERSONA alumno;
   int i;
   FILE  *pf;
   pf  = fopen("AlumnosUNLaM.dat" , "wb");
   if (pf==NULL)
   {
      printf("No se pudo crear el archivo.");
      getch();
      exit (1);
   }
   for(i=0 ; i <= 10; i++)
   {
      printf("Ingrese Numero de DNI");
      scanf("%d", &alumno.DNI);
```

```
      printf("Ingrese Numero de Apellido y nombre");
      fflush(stdin);
      gets(alumno.ApellNombre);
      printf("Ingrese Sexo");
      fflush(stdin);
      scanf("%c", &alumno.Sexo);
      printf("Ingrese Numero del Dia de Nacimiento");
      scanf("%d", &alumno.Nacimiento.Dia);
      printf("Ingrese Numero del Mes de Nacimiento");
      scanf("%d", &alumno.Nacimiento.Mes);
      printf("Ingrese Numero del Año de Nacimiento");
      scanf("%d",  &alumno.Nacimiento.Anio);
      fwrite(&alumno, sizeof(struct PERSONA), 1, pf);
      /*Aquí la función que permite graba un registro en un archivo binario. Se
      indica el nombre de la variable estructura  " alumno " (registro) que se va
      a grabar pf que es el puntero del archivo a grabar. La funcion sizeof
      determina el largo del registro por eso esta invocada la estructura PERSONA
      (58 bytes). */
   }
   return 0;
}
```

Ejemplo 5:

Es un programa que permite grabar los datos que se encuentra en un arreglo (vector) de estructura de datos (los cuales fueron ingresados por teclado dentro de una función llamada “CargaDatosAlumnos”), en un archivo llamado “DatosAlumnosUNLaM.dat”



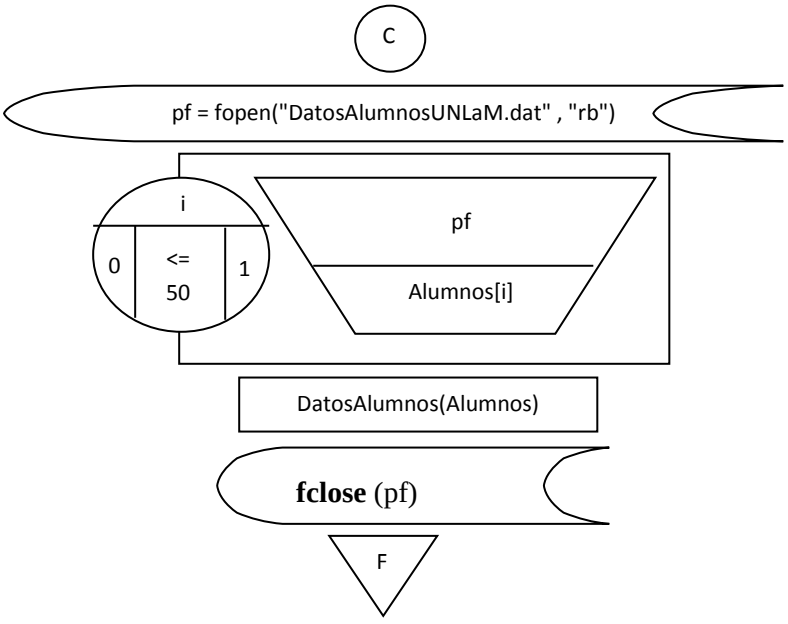


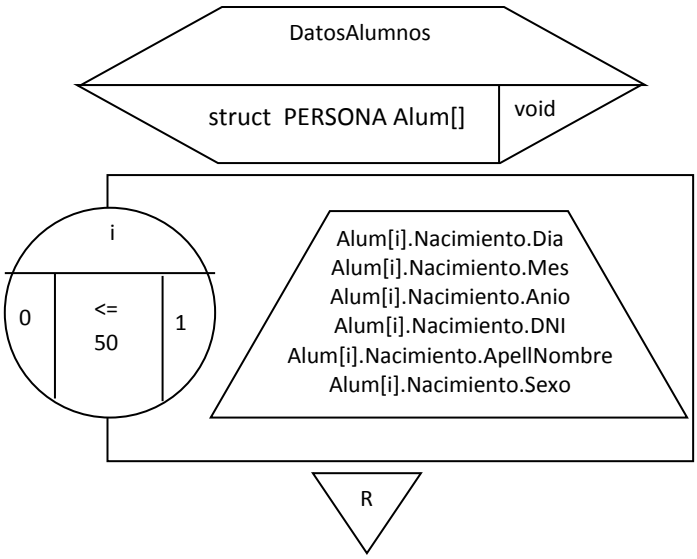

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};
struct PERSONA
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   struct FECHA Nacimiento;
} ;
void CargaDatosAlumnos (struct PERSONA []); //Prototipo de la función
int main()
{
   struct PERSONA Alumnos[50];
   int i;
   FILE *pf;

   CargaDatosAlumnos( Alumnos );// Llamada de la función
   pf = fopen("DatosAlumnosUNLaM.dat" , "wb");
   if (pf==NULL)
   {
      printf("No se pudo crear el archivo.");
      getch();
```

```
      exit (1);
   }
   for(i=0 ;  i <= 50; i++)
   {
      fwrite(&Alumnos[i], sizeof(struct PERSONA), 1, pf);
      //Aquí la función que permite graba un registro a la vez en un archivo
      binario.
   }
   fclose(pf);
   return 0;
}
void CargaDatosAlumnos (struct PERSONA Alum[]) //Declaración de la función
{
   int i;
   for(i=0 ;i <= 50; i++)
   {
      printf("Ingrese Numero de DNI");
      scanf("%d", & Alum[i].DNI);
      printf("Ingrese Apellido y nombre");
      fflush(stdin);
      gets(Alum[i].ApellNombre);
      printf("Ingrese Sexo");
      fflush(stdin);
      scanf("%c", & Alum[i].Sexo);
      printf("Ingrese Numero del Dia de Nacimiento");
      scanf("%d", & Alum[i].Nacimiento.Dia);
      printf("Ingrese Numero del Mes de Nacimiento");
      scanf("%d", & Alum[i].Nacimiento.Mes);
      printf("Ingrese Numero del Año de Nacimiento");
      scanf("%d",  & Alum[i].Nacimiento.Anio);
   }
}
```

Ejemplo 6:

Es un programa que recupera (lee) los datos que se encuentran en un archivo llamado “DatosAlumnosUNLaM.dat”, y luego guardarlos en un arreglo (vector) de estructura de datos, los cuales se muestran por pantalla por medio de una función llamada “DatosAlumnos”. Para realizar este ejemplo se debe compilar siempre y cuando exista el archivo llamado “DatosAlumnosUNLaM.dat” y que contenga 50 registros.






```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};
struct PERSONA
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   struct FECHA  Nacimiento;
};
void DatosAlumnos (struct PERSONA []);//Prototipo de la función
int main()
{
   struct PERSONA Alumnos[50];
   int i;
   FILE *pf;
   pf = fopen("DatosAlumnosUNLaM.dat", "rb");
   if (pf==NULL)
   {
      printf("No se pudo abrir el archivo.");
      getch();
      exit (1);
   }
   for(i=0 ;  i <= 50; i++)
   {
      fread(&Alumnos[i], sizeof(struct PERSONA), 1, pf);
      //Aquí la función que permite recuperar (leer) un registro a la vez en un
      archivo binario.
   }
   DatosAlumnos ( Alumnos );// Llamada de la función
   fclose(pf);
   return 0;
}
```

```
void DatosAlumnos (struct PERSONA Alum[])//Declaración de la función. Permite
recibir como parámetro un arreglo (vector) de estructura de datos y luego
mostrarlos por patalla.
{
   int i;
   for(i=0 ;  i <= 50; i++)
   {
      printf("\nFecha de Nacimiento: %d-%d-%d Nro DNI: %d nombre: %s  sexo %c",
      Alum[i].Nacimiento.Dia,   Alum[i].Nacimiento.Mes,   Alum[i].Nacimiento.Anio,
      Alum[i].DNI, Alum[i].ApellNombre, Alum[i].Sexo);
   }
}
```

En el Anexo de esta unidad encontrará ejercicios resueltos con su correspondiente diagrama y código en lenguaje C abarcando las distintas alternativas que se pueden dar al trabar con archivos binarios.

9. Exportar datos para otros programas

Los archivos binarios son dependientes del tamaño del tipo de dato del lenguaje y de la arquitectura del procesador con el cual fueron creados. Por ejemplo, en los entornos Windows o Linux actuales una variable int ocupa 4 bytes mientras que en sistemas operativos anteriores como D.O.S. un entero ocupaba 2 bytes en memoria. Adicionalmente no se puede leer un archivo binario sino se conoce exactamente como es el formato del registro de datos con el cual fue generado. Es por ello que cuando se desea intercambiar datos entre distintos sistemas en general se utilizan archivos de texto plano ya que pueden ser interpretados por cualquier lenguaje y bajo cualquier arquitectura sin problemas.

El uso de archivos de texto para exportar datos se puede realizar de dos formas distintas:

- Utilizar algún formato estandarizado que permita grabar los datos de una forma estructurada para que puedan ser interpretados. Algunos ejemplos son los formatos XML o JSON este último creado para representar objetos en lenguajes de programación no estructurados.

- Grabar los registros en el archivo de texto con algún separador de campo, por ejemplo, grabando cada campo de la estructura separado por coma o por punto y coma y haciendo que cada registro se grabe en una línea distinta del archivo de texto.

De las dos estrategias utilizaremos la segunda por su sencillez y además permitirá rápidamente poder consultar los datos exportados mediante programas de hojas de cálculo como por ejemplo Excel.

El primer paso es crear un archivo de texto. Para ello se debe modificar el modo de apertura del archivo cambiando la letra b en los modos indicados previamente por la letra t que representa un archivo de texto. Para exportar los datos utilizaremos dos modos de apertura de archivos de texto:

- wt: cuando se quiera crear un archivo de texto únicamente para escritura tomando en consideración de que si ya existía los datos se van a sobrescribir

- at: cuando se quieran exportar datos en un archivo de texto, agregando registros si ya existe o creando un nuevo archivo si el mismo no existía.

Para escribir los datos en el archivo de texto utilizaremos una nueva función **fprintf**. Esta función es similar al printf para mostrar los datos por pantalla con formato solo que en lugar de por pantalla se le indica el archivo en el cual se guardaran los datos. El formato de la función fprintf es el siguiente:

```
fprintf(puntero FILE, texto con formato, lista de variables separadas por coma);
```

A continuación, se muestra un programa que permite ingresar datos de alumnos por teclado y generar un archivo de texto en formato csv (del inglés comma separated values, valores separados por coma) con los datos ingresados. Como separador de campo en lugar de coma se recomienda utilizar punto y coma ya que de esta forma programas como el Excel al abrirlo automáticamente reconocen los distintos campos y ya nos muestran los datos en distintas columnas.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct
{
   char nombre[40];
   int dni;
   float promedio;
}snombres;
int main()
{
   snombres reg;
   FILE * fp;
   //se usa el modo de apertur wt que crea o reemplaza el archivo de texto
   fp=fopen ("alumnos.csv", "wt");
   if (fp==NULL)
   {
      printf("Error al abrir el archivo.");
      exit(1);
   }
   /*Graba una fila de encabezados para identificar los campos, el \n al final hace que se grabe una lína de texto completa y luego baje  a la fila siguiente   */
   fprintf(fp, "Nombre;DNI;Promedio\n");

   printf ("ingrese dni 0 para terminar:");
   scanf("%d", &reg.dni);
   while (reg.dni>0)
   {
      printf ("ingrese el nombre:");
      getchar();
      gets(reg.nombre);
      printf ("Ingrese el promedio:");
      scanf("%f", &reg.promedio);
      //El formato es identico al printf solo se agrega el ;como separadador entre dato y dato
      fprintf(fp, "%s;%d;%.2f\n", reg.nombre, reg.dni, reg.promedio);

      printf ("ingrese dni 0 para terminar:");
      scanf("%d", &reg.dni);
   }
   fclose(fp);
   return 0;
}
```