

Tema: TDA Archivos

Apellido y Nombre: ..... Fecha:...../...../.....

### TEORÍA

En términos computacionales, un **archivo** es una colección de datos (conjunto de bytes) que se identifican mediante un nombre y que se guardan en dispositivos de almacenamiento secundario tales como soportes magnéticos, ópticos, electrónicos, etc. Por ejemplo: discos duros, CD's, ZIP drives, flash drives, etc.

#### ***Clasificación de archivos por contenido***

Existen muchas clasificaciones de archivos de acuerdo a diferentes criterios o aplicaciones. Aquí se presenta una muy importante: de acuerdo al contenido, se pueden clasificar como **archivos de texto** o **binarios**.

##### **Archivos de texto**

Los datos en un archivo de texto se almacenan usando el código ASCII, en el cual cada carácter es representado por un simple byte. Debido a que los archivos de texto utilizan el código ASCII, se pueden desplegar o imprimir. En este tipo de archivos, todos sus datos se almacenan como cadenas de caracteres.

##### **Archivos binarios**

Este tipo de archivos almacenan los datos numéricos con su representación binaria. Pueden ser archivos que contienen instrucciones en lenguaje máquina listas para ser ejecutadas. En este tipo de archivos también se pueden almacenar diferentes tipos de datos incluyendo datos numéricos; sin embargo, cabe destacar que los datos numéricos se graban con su representación binaria (no con su representación ASCII).

#### ***Clasificación de archivos por tipo de acceso***

De acuerdo a la forma de acceder los datos de los archivos, éstos se clasifican en secuenciales o directos (también conocidos como de acceso directo, relativos o aleatorios).

##### **Archivos secuenciales**

Como su nombre lo indica, en este tipo de archivos los registros se graban en secuencia o consecutivamente y deben accederse de ese mismo modo, es decir, conforme se van insertando nuevos registros, éstos se almacenan al final del último registro almacenado; por lo tanto, cuando se desea consultar un registro almacenado es necesario recorrer completamente el archivo leyendo cada registro y comparándolo con el que se busca. En este tipo de archivo se utiliza una marca invisible que el sistema operativo coloca al final de los archivos: EOF (End of File), la cual sirve para identificar dónde termina el archivo

##### **Archivos directos (relativos, de accesos directos o aleatorios)**

A diferencia de los archivos secuenciales, en los archivos directos no es necesario realizar un recorrido para acceder un registro en particular, sino que se puede colocar el apuntador interno del archivo directamente en el registro deseado, permitiendo con esto mayor rapidez de acceso.

#### ***Implementación de archivos en C/C++***

##### **Declaración de archivos**

*Declaración del puntero al archivo:* Para realizar programas de manejo de archivos en Lenguaje C++ se requiere el encabezado "Standard I/O" y se necesita incluirlo de la siguiente forma:

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

Además es necesario declarar un puntero de tipo *FILE* que opere como el apuntador a la estructura del archivo, esto se realiza de la siguiente forma:

```
typedef FILE *parchivo;
...
main()
{ parchivo datos;
...
}
```

### Apertura de Archivos

La función *fopen()* y modos de apertura de archivos

Se usa la función *fopen* para abrir un archivo, determinar el modo de apertura y establecer la vía de comunicación mediante su puntero correspondiente. Además determinar el tipo de contenido del archivo (texto o binario). Esta función tiene dos argumentos: el nombre del archivo y su modo. Los modos de apertura de archivos de texto y binarios, se detallan en la siguiente tabla.

Modo de apertura (archivos de texto)	Modo de apertura (archivos binarios)	Operación
"r"	"rb"	Apertura en modo de sólo lectura. El archivo debe existir.
"w"	"wb"	Apertura en modo de sólo escritura. Si el archivo existe, se reescribirá (pierde el contenido anterior). Si el archivo no existe, lo crea.
"a"	"ab"	Apertura en modo de agregar. Si el archivo existe, los datos se agregan al final del archivo, en caso contrario, el archivo se crea.
"r+"	"rb+"	Apertura en modo de lectura/escritura. El archivo debe existir.
"w+"	"wb+"	Apertura en modo de lectura/escritura. Si el archivo existe, se reescribirá (pierde el contenido anterior).
"a+"	"ab+"	Apertura en modo de lectura/agregar. Si el archivo no existe lo crea.

Ejemplo: Uso de la función open (archivos binarios)

```
#include <stdio.h>
typedef FILE *parchivo;
...
main()
{ parchivo arch1, arch2, arch3;
  arch1=fopen("EJEMPLO.DAT","rb"); // Abre el archivo binario EJEMPLO.DAT en modo de sólo
  lectura y lo asigna al apuntador.
  arch2=fopen("ARCHIVO.TXT","ab"); // Nótese que se necesitan dos '\\' ya que el backslash
  indica el inicio de una secuencia de escape en C++
  arch3=fopen("c:\\tarea\\PRODUCTO.005","w"); // Crea el archivo de texto PRODUCTO.005 en
  modo de
  //sólo escritura y lo asigna al apuntador alias3. El archivo lo crea en el subdirectorío
  c:\tarea
...
}
```

### Validar la apertura de un archivo

Algunas funciones requieren la existencia del archivo para realizar operaciones, por ello es necesario verificar que cuando se intenta abrir un archivo haya tenido éxito la operación. Si un archivo no se puede abrir, la función *fopen* devuelve el valor de 0 (cero), definido como NULL en *stdio.h*.

*Ejemplo:* El siguiente código verifica la apertura de un archivo.

```
#include <stdio.h>
#include <iostream>
using namespace std;
typedef FILE *parchivo;
main()
{
    parchivo arch1
    arch1=fopen("EJEMPLO.DAT", "rb");
    if (arch1==NULL)
        cout << "No se puede abrir el archivo !!!";
    system("pause");
}
```

### Cierre de Archivos

Al finalizar el trabajo con un archivo es necesario cerrarlo. Esto se logra mediante las funciones ***fclose***. Si se usa ***fclose*** es necesario indicarle el alias del archivo que se desea cerrar.

### Lectura y Escritura de Archivos

La función ***fread*** permite "cargar" todos los campos de un registro en un archivo, es decir, lee un registro y lo copia en la memoria RAM. La función ***fread*** tiene la siguiente sintaxis:

```
fread(&registro_entrada, sizeof(registro_entrada), 1, puntero_archivo);
```

Donde ***registro\_entrada*** se refiere a la variable tipo registro que almacenará los datos leídos desde archivo, ***sizeof(registro\_entrada)*** indica el tamaño o cantidad de información leída desde el archivo, el número **1** especifica que se leerá sólo un registro del archivo y ***puntero\_archivo*** especifica el nombre del puntero que hace referencia al archivo.

La función ***fwrite*** proporciona el mecanismo para almacenar todos los campos de un registro en un archivo. Cabe destacar que al utilizar esta función, se almacena una variable (de tipo struct) que representa un bloque de datos o campos; es decir, no se almacena campo por campo. La función ***fwrite*** tiene la siguiente sintaxis:

```
fwrite(&registro_salida, sizeof(registro_salida), 1, puntero_archivo);
```

Los argumentos de la función son: el nombre del registro que se grabará, tamaño en bytes del registro, cantidad de variables (registros) a grabar y el alias del archivo donde se desea almacenar.

Ejemplo:

```
#include <stdio.h>

FILE *parchivo; //Declaración del puntero a archivo

typedef char tcad[30];
typedef struct tproducto
    {
        int no_prod;
        tcad descrip;
        int cantidad;
        float precio;
    };

main()
{
    parchivo producto;
    tproducto prod; //Declaración de la variable "prod" de tipo "tproducto"
    fwrite(&prod, sizeof(prod), 1, producto); // La función fwrite graba el registro prod en
    en el archivo producto.
```

### Vaciando los buffers con fflush()

Un buffer es un área de almacenamiento temporal en memoria para el conjunto de datos leídos o escritos en el archivo. Estos buffers retienen datos en tránsito desde y hacia al archivo y tienen la finalidad de hacer más eficiente las operaciones de entrada/salida en los archivos de disco, provocando menor cantidad de accesos, los cuales son más lentos que la memoria. Por ejemplo, si se requiere consultar constantemente un dato del archivo, no es necesario calcular su dirección física, reposicionar el apuntador del archivo, "cargar" el dato en memoria mediante una operación de lectura cada vez que se necesita, sino que el sistema operativo controla y mantiene este dato en los buffers de memoria y de ahí lo toma cuando lo requiera. Sólo hay una consideración importante al utilizar los buffers, los datos escritos en ellos no se reflejan exactamente en los archivos de disco en forma inmediata, sino hasta que se "vacía" el buffer. Para ello se utiliza la función *fflush* y basta enviarle el puntero del archivo como argumento. Los buffers también se vacían cuando se cierra el archivo.

### Cálculo de Direcciones Físicas

Para reposicionar el apuntador de un archivo en un registro específico es necesario calcular su dirección física correspondiente de acuerdo al espacio ocupado por sus registros predecesores.

La función *sizeof* calcula el tamaño en bytes de una variable o estructura.

```
typedef struct tproducto
{
    int no_prod;
    char descrip[30];
    int cantidad;
    float precio;
};

...
main()
{ tproducto prod;
  ...
```

#### Archivo: PRODUCTO.SEC

Dir.	Dir.	No_prod	Descrip	Cantidad	Precio	Garantia
0	0	0	"Camisa de vestir"	100	65.80	'N'
1	41	1	"Pantalón para dama"	234	115.50	'N'
2	82	2	"Radiograbadora"	36	895.75	'S'
3	123	3	"Gabinete para sala"	54	1532.60	'N'
4	164					
5	205					
		Entero	Cadena [30]	Entero	Real	Caracter

Para la estructura de un registro anterior se realiza el cálculo de la dirección física a partir de la dirección lógica:

```
long int dir_fisica, dir_logica;
dir_fisica= dir_logica*sizeof(prod);
```

Por ejemplo, en el caso del registro "Radiograbadora" cuya dirección lógica es 2 (código del producto), la dirección física (82) se obtiene calculando cuántos bytes ocupan los registros precedentes (41 bytes cada uno).

### Reposicionando el apuntador mediante fseek()

Cuando se abre un archivo en modo de sólo lectura, sólo escritura o lectura/escritura, el apuntador del archivo se posiciona al inicio del mismo y cuando un archivo se abre en modo agregar se posiciona al final, sin embargo, se puede reposicionar este apuntador mediante la función *fseek*. También es importante mencionar que cada vez que se realiza una operación de lectura o de escritura de cualquier tipo de datos (carácter, cadena, estructura, etc.), el apuntador del archivo se mueve al final de dicho

dato, de tal forma que está posicionado al comienzo del siguiente. Por ello es importante asegurarse que el apuntador se encuentre en la posición deseada antes de realizar cualquier operación.

Los archivos en lenguaje C++ son controlados por direcciones físicas (no lógicas) indicadas en bytes y la función *fseek* permite reposicionar el apuntador del archivo en la dirección física deseada mediante tres argumentos: puntero al archivo, desplazamiento (en bytes) y el punto de referencia. Para reposicionar el apuntador del archivo es necesario que éste se encuentre abierto. El conteo de bytes indicado por el desplazamiento se realiza a partir del principio del archivo, la posición actual del apuntador o el final.

El último argumento de la función *fseek* es conocido como el punto de referencia y se refiere a dónde se iniciará el conteo de bytes determinado por el desplazamiento.

Modo	Nombre	Operación
0	SEEK_SET	Desde el principio del archivo
1	SEEK_CUR	Desde la posición actual del apuntador del archivo
2	SEEK_END	Desde el final del archivo

Ejemplo:

```
fseek(puntero_archivo, desplazamiento, punto_referencia);
```

Argumentos de la función *fseek*:

- puntero\_archivo: puntero del archivo que se desea reposicionar.
- desplazamiento: cantidad de bytes que se desplazará el puntero (hacia adelante o hacia atrás) respecto al punto de referencia.
- punto\_referencia: posición a partir de la que se considera el desplazamiento para posicionar el puntero (0: inicio del archivo, 1: posición actual del puntero, 2: final del archivo).

### Función ftell para conocer la posición del apuntador del archivo

Se usa la función *ftell()* para conocer la posición actual del apuntador de un archivo abierto. La posición se expresa en bytes (dirección física) contados desde el principio del archivo.

Ejemplo:

```
long int dir_fisica;
dir_fisica =ftell(puntero_archivo);
```

### Función rewind para colocar el apuntador al archivo al principio

Se usa la función *rewind* para colocar el apuntador del archivo al principio de un archivo abierto sin necesidad de usar la función *fseek*. Basta con indicar el puntero del archivo como argumento.

```
rewind(puntero_archivo);
```

### Función feof para detectar el final del archivo

Se usa *feof()* (definido en *stdio.h*) para determinar si se ha encontrado el final de un archivo. Si el puntero del archivo alcanza el final de éste se devuelve un valor diferente de cero, caso contrario, retorna cero. Para invocarlo es necesario indicar el puntero del archivo abierto como argumento. Esta función se utiliza al recorrer un archivo de forma secuencial.

### **Aplicaciones de Archivos en C++**

A continuación se presentan algunas de las aplicaciones prácticas más comunes con archivos. Se analizarán los procedimientos de inserción, consulta y eliminación de registros así como el listado de contenido.

**Alta y Listado**

El siguiente programa presenta un menú de opciones que permiten: a) agregar registros al archivo PRODUCTO.SEC y b) listar el contenido del archivo PRODUCTO.SEC. La operación *alta\_producto*, que abre el archivo en modo lectura/escritura (ab+), permite agregar registros al final del archivo (y leer los registros anteriores si fuera necesario). Esta operación se vale de la operación *carga\_producto* que asigna valores ingresados por el usuario a una variable tipo *tproducto*.

La operación *listar\_producto*, que abre el archivo en modo lectura (rb), recorre registro a registro el archivo asignando cada registro leído a la variable *p* tipo *tproducto*. Esta operación se vale de la operación *mostrar\_producto* que muestra en pantalla el contenido de los campos de una variable tipo *tproducto*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

typedef char tcad[30];
typedef struct tproducto
    {   int no_prod;
        tcad descrip;
        int cantidad;
        float precio;
    };
typedef FILE *parchivo;

void carga_producto(tproducto &p);
void alta_producto(parchivo prod);
void mostrar_producto(tproducto p);
void listar_producto(parchivo prod);

main()
{   parchivo productos;
    int op;
    do
    {   system("cls");
        cout << " **** MENU **** " << endl;
        cout << "1- Alta Producto" << endl;
        cout << "2- Listar Productos " << endl;
        cout << "3- Salir" << endl;
        cout << "ELIJA UNA OPCION: ";
        cin >> op;
        switch (op)
        {   case 1: cout << "ALTA DE PRODUCTOS" << endl;
            alta_producto(productos);
                break;
            case 2: cout << "LISTADO DE PRODUCTOS" << endl;
            listar_producto(productos);
                break;
            case 3: cout << "FIN DE PROGRAMA" << endl;
                break;
            default: cout << "ERROR DE OPCIÓN" << endl;
        }
        cout << "<<< Oprima cualquier tecla para continuar >>>"<<endl;
        system ("pause");
    } while(op!=3);
}

void carga_producto(tproducto &p)
{   cout << "Codigo producto: ";
    cin >> p.no_prod;
    cout << "Descripcion: ";
    fflush(stdin)
    gets(p.descrip);
    cout << "Cantidad : ";
    cin >> p.cantidad;
    cout << "Precio : ";
    cin >> p.precio;
```

```

}

void alta_producto(parchivo prod)
{
    tproducto p;
    char rta;
    prod=fopen("PRODUCTO.SEC","ab"); // Abre el archivo en modo de lectura/escritura
    do
    {
        carga_producto(p);
        fwrite(&p,sizeof(p),1,prod); // Grabar el Registro
        cout << "Mas datos S/N: ";
        cin >> rta;
    } while(rta!='n' && rta!='N');
    fclose(prod);
}

void mostrar_producto(tproducto p)
{
    cout << "Codigo producto: " << p.no_prod << endl;
    cout << "Descripcion: " << p.descripcion << endl;
    cout << "Cantidad : " << p.cantidad << endl;
    cout << "Precio : " << p.precio << endl;
}

void listar_producto(parchivo prod)
{
    tproducto p;
    char rta;
    prod=fopen("PRODUCTO.SEC","rb");
    if(prod==NULL)
        cout << "El archivo no existe" << endl;
    else
        while(!feof(prod))
        {
            fread(&p,sizeof(p),1,prod); // Leer un registro
            if (!feof(prod))
                mostrar_producto(p);
        }
    fclose(prod);
}

```

### Consulta

La operación *consulta\_producto*, presentada a continuación, permite buscar y mostrar un registro específico del archivo PRODUCTO.SEC. Para ello, el archivo se abre el modo lectura y se recorre registro a registro consultando por el valor buscado. Si alguno de los registros coincide con el criterio de búsqueda entonces se muestra su contenido, caso contrario, se muestra el mensaje "Producto inexistente". Este módulo puede integrarse fácilmente al programa anterior.

```

void consulta_producto(parchivo prod,int codigo)
{
    tproducto p;
    bool existe=false;
    prod=fopen("PRODUCTO.SEC","rb");
    if(prod==NULL)
        cout << "El archivo no existe" << endl;
    else
        while(!feof(prod) && !existe)
        {
            fread(&p,sizeof(p),1,prod); // Leer un registro
            if (!feof(prod) && codigo==p.no_prod)
                {
                    mostrar_producto(p);
                    existe=true;
                }
        }
    if (!existe)
        cout << "Producto inexistente" << endl;
    fclose(prod);
}

```

### Modificación

La operación *modifica\_producto*, presentada a continuación, permite buscar un registro específico del archivo PRODUCTO.SEC y modificar su contenido. Para ello, el archivo se abre el modo lectura/escritura y se recorre registro a registro consultando por el valor buscado. Si alguno de los registros coincide con el criterio de búsqueda entonces se detiene el recorrido y se reposiciona el

puntero al comienzo de ese registro. Entonces se solicita al usuario los datos del registro y se sobrescribe. Este módulo puede integrarse fácilmente al programa ejemplo ya presentado.

```
void modifica_producto(parchivo prod,int codigo)
{ tproducto p;
  bool existe=false;
  prod=fopen("PRODUCTO.SEC","rb+");
  if(prod==NULL)
    cout << "El archivo no existe" << endl;
  else
    while(!feof(prod) && !existe)
      { fread(&p,sizeof(p),1,prod); // Leer un registro
        if (!feof(prod) && codigo==p.no_prod)
          existe=true;
        }
    if (existe==true)
      { carga_producto(p);
        fseek(prod,-sizeof(p),1);
        fwrite(&p,sizeof(p),1,prod);
        }
    else
      cout << "Producto inexistente" << endl;
  fclose(prod);
}
```

## PRÁCTICA

- 1) Codifique el siguiente programa, analice y explique los módulos *cargar* y *mostrar*. ¿Cuál es el propósito de la instrucción *if (!feof(f))* en el procedimiento *mostrar*? ¿qué ocurre si comenta esta condición?

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
using namespace std;
typedef FILE *parchivo;
void cargar(parchivo f);
void mostrar(parchivo f);

main()
{
  parchivo fp;
  srand(time(NULL));
  cargar(fp);
  mostrar(fp);
  system("pause");
}

void cargar(parchivo f)
{ char letra;
  int i;
  f=fopen("fichero.dat","wb+");
  for (i=0;i<15;i++)
  { letra=97+rand()%26;
    fwrite(&letra, sizeof(letra),1,f);
  }
  fclose(f);
}

void mostrar(parchivo f)
{ char letra;
  f=fopen("fichero.dat","rb");
  while (!feof(f))
  { fread(&letra, sizeof(letra),1,f);
    if (!feof(f))
      cout << letra << " ";
    }
  cout << endl;
  fclose(f);
}
```



- 2) Modifique el programa anterior agregando un módulo que cuentes las vocales almacenadas en el archivo. ¿En qué modo puede abrir el archivo en el módulo cargar para no perder el contenido?
- 3) Codifique el siguiente programa, analice el número de byte que está siendo apuntado por el identificador de posición del archivo. ¿Qué es lo que obtiene la función *ftell*? ¿cuál es el tamaño (en bytes) del tipo de dato almacenado? ¿qué control falta en el fragmento de programa que muestra la ubicación del puntero?

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

typedef FILE *parchivo;

main()
{
    parchivo fp;
    int pos;
    float num;
    fp=fopen("fichero.dat","wb+");
    cout << "INGRESE VALORES - INGRESO FINALIZA CON CERO" << endl;
    do
    {
        cout << "Ingrese un numero real: ";
        cin >> num;
        if (num!=0)
            fwrite(&num, sizeof(num), 1, fp);
    }while (num!=0);

    rewind(fp); // reubica el puntero al inicio del archivo

    while (!feof(fp))
    {
        fread(&num,sizeof(num), 1, fp);
        pos=ftell(fp); // obtiene el número de byte en que se encuentra el puntero
        cout<<"la variable letra contiene: "<< num <<endl;
        cout<<"byte nro: "<<pos<<endl;
    }
    fclose (fp);
    system ("pause");
}
```

- 4) Modifique el programa del ítem 3 de modo que el archivo guarde valores de tipo *double*, ¿se produce algún cambio en la salida del programa? Además adicione un módulo que calcule la suma de los valores almacenados y otro que determine el mínimo valor del archivo.
- 5) Codifique el siguiente programa, analice y explique los módulos *cargar*, *mostrar\_cadena*, *mostrar\_palabra* y *mostrar\_letra*.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string.h>

using namespace std;

typedef FILE *parchivo;
typedef char tcad[30];

void cargar(parchivo f);
void mostrar_cadena(parchivo f);
void mostrar_palabra(parchivo f);
void mostrar_letra(parchivo f);

main()
{
    parchivo fp;
    cargar(fp);
    mostrar_cadena(fp);
    system ("pause");
    mostrar_palabra(fp);
    system ("pause");
    mostrar_letra(fp);
    system ("pause");
}
```

```

void cargar(parchivo f)
{
    tcad frase;
    f=fopen("fichero.dat","wb+");
    cout << "Ingrese cadena de caracteres:";
    gets(frase);
    fwrite(&frase, sizeof(frase),1,f);
    fclose (f);
}
void mostrar_cadena(parchivo f)
{
    tcad frase;
    f=fopen ("fichero.dat","rb");
    fread(&frase, sizeof(frase),1,f);
    cout << frase << endl;
    fclose(f);
}

void mostrar_palabra(parchivo f)
{
    tcad frase;
    f=fopen ("fichero.dat","rb");
    while (!feof(f))
    {
        fscanf(f,"%s",&frase);
        cout << frase << endl;
    }
    fclose(f);
}

void mostrar_letra(parchivo f)
{
    char letra=' ';
    f=fopen ("fichero.dat","rb");
    while (!feof(f) && letra!='\0')
    {
        fread(&letra,sizeof(letra),1,f);
        cout << letra << endl;
    }
    fclose(f);
}

```

6) Modifique el programa anterior de modo que incluya módulos para:

- contar la cantidad de palabras del archivo
- mostrar las palabras del archivo y su longitud
- contar los caracteres del archivo.

7) Considerando que el siguiente programa permite crear y mostrar el contenido del archivo alumnos.txt (archivo de texto), codifique lo y compruebe su funcionamiento.

```

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
typedef char tcad[50];
typedef struct talumno
    { int libreta;
      tcad nombre;
      tcad carrera;
    };
typedef FILE *parchivo;

void alta (parchivo alumnos);
void carga(talumno &a);
void mostrar(parchivo alumnos);

```

```

main()
{ parchivo estudiantes;
  alta(estudiantes);
  mostrar(estudiantes);
  system("pause");
}
void carga (talumno &a)
{ fflush(stdin);
  cout << "Ingrese LU: ";
  cin >> a.libreta;
  fflush(stdin);
  cout << "Ingrese nombre: ";
  gets(a.nombre);
  fflush(stdin);
  cout << "Ingrese carrera: ";
  gets(a.carrera);
  fflush(stdin);
}

void alta (parchivo alumnos)
{ talumno a;
  int cantidad;
  alumnos=fopen("alumnos.txt","ab");
  cout << "Cuántos registros desea guardar: ";
  cin >> cantidad;
  while (cantidad>0)
  { carga(a);
    fwrite(&a,sizeof(a),1,alumnos);
    cantidad--;
  }
  fclose(alumnos);
}

void mostrar(parchivo alumnos)
{ talumno a;
  alumnos=fopen("alumnos.txt","rb");
  if (alumnos==NULL)
    cout << "Archivo Inexistente" << endl;
  else
  { while (!feof(alumnos))
    { fread(&a,sizeof(a),1,alumnos);
      if (!feof(alumnos))
      { cout << a.libreta << endl;
        cout << a.nombre << endl;
        cout << a.carrera << endl; }
    }
  }
  fclose(alumnos);
}

```

8) Modifique el programa del ítem anterior de modo que incluya siguiente rutina de búsqueda.

```

bool buscar_archivo(parchivo alumnos, int buscado)
{ bool existe=false;
  talumno a;
  alumnos=fopen("alumnos.txt","rb");
  if (alumnos!=NULL)
  while (!feof(alumnos)&& existe==false)
  { fread(&a,sizeof(a),1,alumnos);
    if (a.libreta==buscado)
      existe=true;
  }
  fclose(alumnos);
  return existe;
}

```

- 9) Modifique el programa del ítem 7 de modo que incluya la siguiente rutina de modificación de registros.

```
void modificar(parchivo alumnos, int buscado)
{ talumno a;
  bool band=false;
  alumnos=fopen("alumnos.txt","rb+");
  if (alumnos==NULL)
    cout << "Archivo Inexistente" << endl;
  else
  { while (!feof(alumnos)&& band==false)
    { fread(&a,sizeof(a),1,alumnos);
      if (a.libreta==buscado)
        band=true;
    }
    if (band==true)
    { carga(a);
      fseek(alumnos,-sizeof(a),1);
      fwrite(&a,sizeof(a),1,alumnos);
    }
    else
      cout << "REGISTRO NO ENCONTRADO" << endl;
  }
  fclose(alumnos);
}
```

- 10) Modifique el programa del ítem 7 de modo que incluya la siguiente rutina de eliminación de registros.

```
void borrar(parchivo alumnos, int buscado)
{ talumno a;
  parchivo aux;
  bool band=false;
  alumnos=fopen("alumnos.txt","rb");
  aux=fopen("temporal.txt","w");
  if (alumnos!=NULL)
  { while (!feof(alumnos))
    { fread(&a,sizeof(a),1,alumnos);
      if (a.libreta!=buscado && !feof(alumnos))
        fwrite(&a,sizeof(a),1,aux);
    }
    fclose(aux);
    fclose(alumnos);
    if (remove("alumnos.txt")==0)
    { cout << "ELIMINADO EXITOSAMENTE" << endl;
      if (rename("temporal.txt","alumnos.txt")==0)
        cout << "RENOMBRADO EXITOSAMENTE" << endl;
      else
        cout << "ERROR AL RENOMBRAR" << endl;
    }
    else
      cout << "ERROR AL ELIMINAR" << endl;
  }
}
```

- 11) Consigne la declaración de tipos y variables necesaria para almacenar información acerca de los pacientes de una clínica. Considere que por cada paciente se debe registrar: apellido, nombre, dni, número carnet, obra social, fecha de ingreso (dd/mm/aaaa) y médico de cabecera. Además codifique un programa que incluya las siguientes operaciones: Altas, Consultas, Listados, Búsqueda, Modificaciones, Bajas. También desarrolle una operación que permita listar los pacientes que pertenezcan a una obra social especificada por el usuario.
- 12) Consigne la declaración de tipos y variables necesaria para almacenar información acerca de usuarios de una red social. Considere que por cada usuario se debe registrar: id de usuario, apellido y nombre, fecha de nacimiento (dd/mm/aaaa), fecha de registro (dd/mm/aaaa), correo electrónico, cantidad de contactos, cantidad de publicaciones. Además codifique un programa que incluya las siguientes operaciones: Altas, Consultas, Listados, Búsqueda, Modificaciones, Bajas. También desarrolle una operación que permita listar los usuarios cuya cantidad de publicaciones sea mayor a un valor especificado, indicando cuántos se encontraron.

