75.40 Algoritmos y Programación I Curso 4
Archivos

Dr. Mariano Méndez[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

23 de agosto de 2022

1. Introducción

Hasta este momento los arreglos o vectores de n dimensiones han permitido solucionar cualquier problema computacional que se ha planteado. Lamentablemente dichos problemas quedan resueltos mientras la computadora recibe alimentación eléctrica, y mientras los datos, que son utilizados para resolver el problema, se encuentran almacenados en la memoria principal de la misma. Es decir que los datos del programa no persisten en el tiempo. A continuación se describirá detalladamente un nuevo tipo de dato que permitirá persistir en el tiempo la información de nuestros programas. Haciendo una revisión de cuales serian las desventajas de los arreglos se puede determinar la siguiente lista:

- Cardinalidad Finita: los arreglos pueden almacenar una cantidad finita de elementos y esa cantidad no puede variar a lo largo del algoritmo. La cantidad está limitada por el tamaño de la memoria.
- Persistencia Temporalidad: Los datos que almacenan los arreglos o vectores, sean de la dimensión que sean, no persisten en el tiempo, solo viven el tiempo que el algoritmo se encuentra ejecutándose en la memoria principal de la computadora.

Puede definirse al tipo de dato FILE como un conjunto ordenado de datos homogéneos almacenado en un determinado dispositivo. Entre sus características más destacables están:

- Cardinalidad infinita: su tamaño puede variar en tiempo de ejecución del algoritmo, esto virtualmente pueda contener cualquier cantidad de elementos. La limitación está dada por el tamaño del dispositivo, que generalmente es 1000 veces mayor que el tamaño total de la memoria principal de la computadora.
- Persistencia Total: la información almacenada dentro de un archivo persiste en el tiempo, es decir que queda almacenada una vez que el algoritmo termino da hacer uso de la misma. La única forma de perderlos es debido a un error del algoritmo o de un problema del dispositivo de almacenamiento.

2. Dispositivos de Entrada y Salida

Como se ha visto anteriormente una computadora está formada por la unidad central de procesamiento o *cpu*, la memoria principal y los dispositivos (o también conocidos como **periféricos**) conectados a la misma. Estos dispositivos se dividen en dos principales clases. Los dispositivos de entrada, aquellos con los que el mundo externo se relaciona para proporcionarle datos a nuestros programas. Los dispositivos de salida, aquellos que se relacionan con el mundo que rodea a la computadora para proporcionar información procesada por la misma. Las operaciones de entrada y salida de datos desde y hacia una computadora representan la relación de la misma con el mundo que la rodea a través de la utilización de un dispositivo determinado. Se define una operación de salida de datos a el flujo de información entre una computadora y un dispositivo que permite mostrar datos ya sea en forma auditiva, táctil o visual. Se define como operación de entrada de datos al flujo de información que es ingresado en un dispositivo y es utilizado por una computadora para realizar algún tipo de procesamiento. Dentro de los dispositivos exclusivamente de entrada de datos se pueden encontrar:

- Teclado
- Ratón óptico
- Escáner

- Micrófono
- Joystick
- Tarjeta Perforada

Por otro lado dentro de la lista de dispositivos exclusivamente de salida se pueden encontrar:

- Auditivos
  - Altavoz
  - Auriculares
  - Tarjeta de sonido
- Táctiles
  - Impresora braille
  - Impresora 3D
- Visuales
  - Led
  - Proyector de vídeo
  - Monitor
    - De tubo de rayos catódicos (CRT)
    - De pantalla de plasma (PDP)
    - De pantalla de cristal líquido (LCD)
    - De paneles de diodos orgánicos de emisión de luz (OLED)
  - Impresora
    - Impresora de impacto
    - Impresora matricial
    - Impresora de margarita
    - Impresora de línea
    - Impresora de sublimación
    - Impresora de inyección
    - Impresora térmica
    - Impresora láser

Existen por otra parte dispositivos en los que se tiene la posibilidad de realizar ambos tipos de operaciones de entrada y de salida, estos se denominan dispositivos de entrada/salida:

- Pantalla Táctil
- Multitáctil
- Casco Virtual
- Impresora Multifunción
- Discos Rígidos
- Unidad de estado sólido
- Memorias Flash
- Discos Floppy
- Unidades de Cinta Magnética.

En esta sección se hará determinado hincapié en los dispositivos de entrada y salida de datos como los discos rígidos, discos floppy y unidades de cinta magnética.

2.1. Dispositivos

A continuación se realizará una breve descripción de la estructura alguno de estos dispositivos.

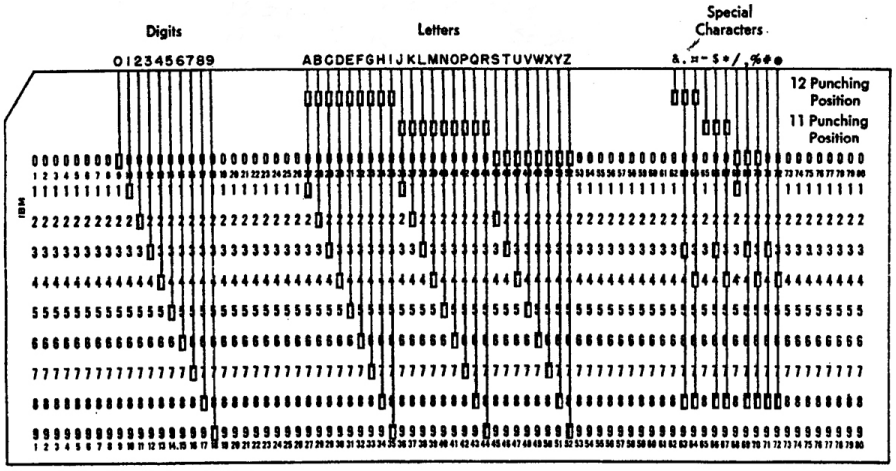


Figure 2. Punching Positions in Card

Figura 1: Esquema de la codificación de una tarjeta perforada

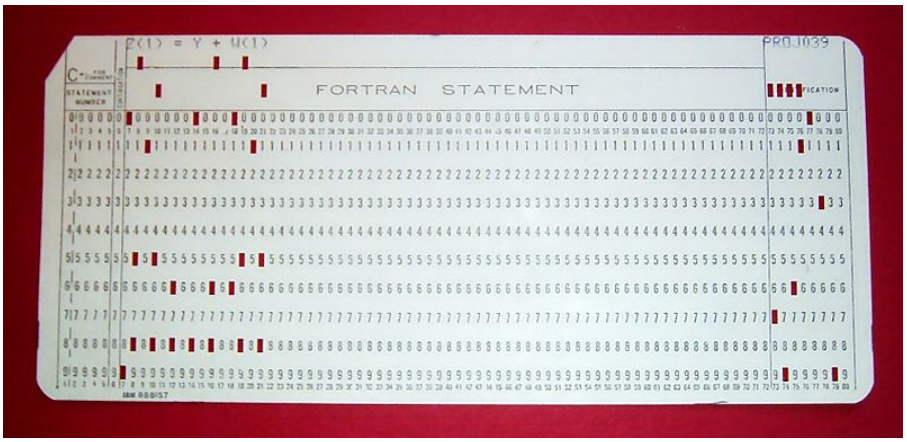


Figura 2: Esquema de la codificación de una tarjeta perforada

2.2. Lectoras y Grabadoras de Tarjetas Perforadas

Este tipo de dispositivo se encuentra en desuso hace ya varios años, pero debido a que ha sido utilizado por tiempo y ha dejado herencia técnica es interesante recordar su funcionamiento. Las tarjetas perforadas estaban hechas de cartulina que almacenaban información en forma de perforaciones rectangulares. Estas tarjetas tenían una codificación especifica que puede verse en la Figura **??**. Cabe destacar que cada una de estas tarjetas equivalía a una linea de información, si se estaba hablando de un programa equivalían a una linea del mismos, como se puede ver el la Figura **??** equivalente a la linea $Z(1) = Y + W(1)$ en FORTRAN.

El legado dejado por las tarjetas perforadas fue el formato de acceso a la información en ciertos dispositivos de almacenamientos de datos, el llamado acceso secuencial a la información, entre otras cosas

2.3. Cintas

Una cinta magnética de almacenamiento de información era similar a las cintas utilizadas para almacenar música. Conceptual-mente el principio era el mismo, la cinta era de un material magnetizable y el dispositivo de lectura/escritura poseía unos cabezales capaces de leer o escribir el información, ver Figura **??**. En el caso de Cintas magnéticas para almacenar datos estas tenían características especiales como por ejemplo que su ancho estándar era de media pulgada. El primer dispositivo capaz de leer o escribir en una cinta magnetita fue en 1951, de *marca Remington Rand modelo:UNISERVO* con una capacidad de 224 kB. El sistema de escritura y lectura en una cinta magnética es conocido como escritura o lectura secuencial, pues los bit son escritos uno a continuación de otro a medida que la cinta pasa por debajo del cabezal. Se debe tener en cuenta que aunque parezca trivial la cinta sólo podía leerse o escribirse, no era posible realizar ambas operaciones simultáneamente. Los cabezales solían escribir simultáneamente en lo que se llamaban tracks o trazas, por lo general el numero de tracks oscilaba entre 7 y 9 ver Figura **??**. Usualmente un operador era el encargado de cambiar y disponer la cinta necesaria en la unidad , ver Figura **??**.

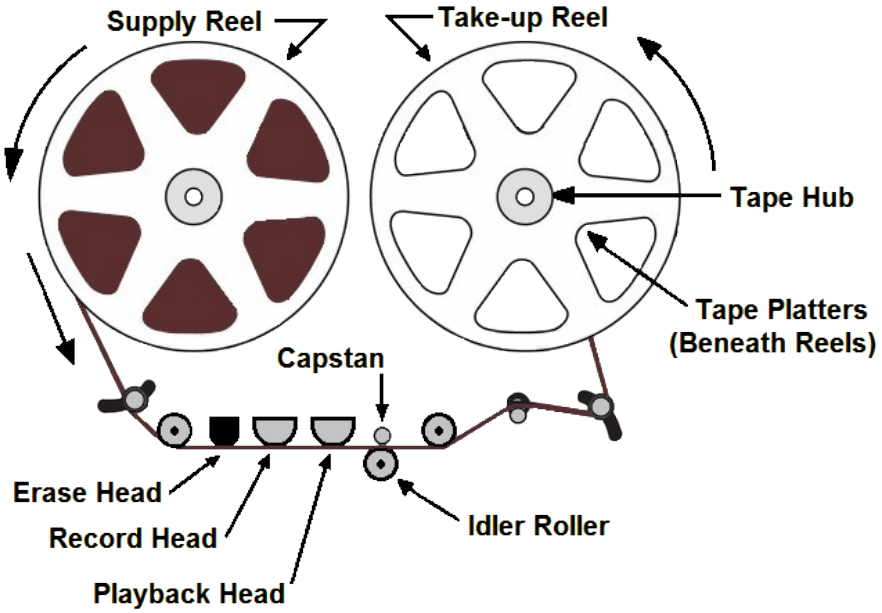


Figura 3: Esquema interno de un disco

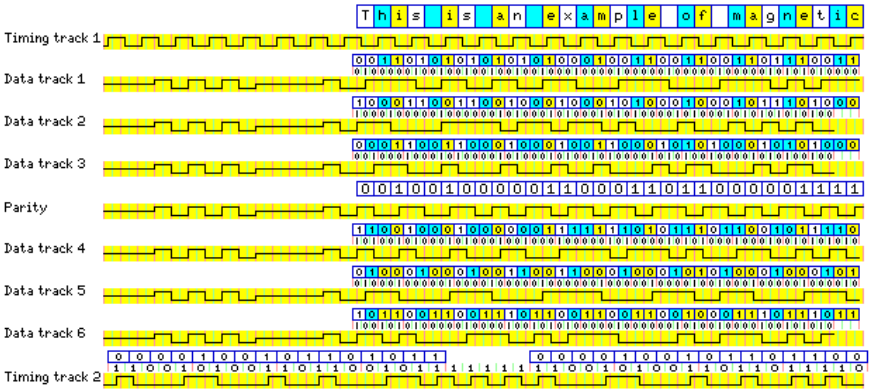


Figura 4: Esquema interno de un disco

Figura 5: Operador Cambiando una cinta

2.4. Discos

Un disco esta compuesto por uno o más platos (platter) circulares que giran sobre un eje. Un brazo mecánico movido por un motor desplaza a una serie de cabezales (heads) sobre la superficie de estos platos, hechos de un material sensible al magnetismo. Estos cabezales son capaces de transformar el estado del material en impulsos eléctricos (ceros o unos), ver Figura **??**.

El funcionamiento de todos estos dispositivos mencionados anteriormente dependía exclusivamente del fabricante del mismo. Con el surgimientos de los lenguajes de programación de alto nivel se inicio una tendencia a independizar el funcionamiento del dispositivo del lenguaje de programación.

3. Entrada y salida de datos en C

El lenguaje de programación C no está provisto de instrucciones específicas de entrada y salida de datos [**?**]. Para tal fin se utilizan ciertas funciones de entrada y salida definidas en la biblioteca estándar de C. Dicho conjunto de funciones se encuentra agrupado en en encabezado o header llamado <stdio.h>[**?**]. En los albores de los lenguajes de programación hacia los fines de los años 50 las operaciones de entrada y salida de datos dependían exclusivamente del hardware, por ejemplo, en el compilador de FORTRAN IV para la IBM 704 las instrucciones de entrada y salida eran dependientes del hardware en el cual se ejecutara.

Una de las características del Lenguaje de programación C es su independencia de la máquina en que se esté ejecutando [**?**]. Para ello se utilizó una abstracción con el fin de lograr que las operaciones de entrada y salida, de datos, sean independientes de la plataforma en el que se está ejecutando. Esta abstracción se denominó *stream* o flujo. Un stream es una fuente o destino de datos que puede estar asociado a cualquier dispositivo. En el caso de C existen dos tipos de streams: *text stream*s o *Binary streams*. "Un stream de texto es una secuencia de líneas; cada línea tiene cero o más caracteres y está terminada por \n". Un entorno puede necesitar convertir un stream de texto a alguna representación, o de alguna otra representación (tal como la asociación de '\n' a retorno de carro y avance de línea).Un stream binario es una secuencia de bytes no procesados que representan datos internos, con la propiedad de que si es escrito y después leído de nuevo en el mismo sistema, será comparado como igual. Un stream se conecta a un archivo o dispositivo al abrirlo-, la conexión se rompe cerrando el stream. El abrir un archivo regresa un puntero a un objeto de tipo FILE, que registra cualquier información necesaria para controlar el stream. Usaremos puntero a stream y stream indistintamente cuando no haya ambigüedad. Cuando un programa inicia su ejecución, los stream **stdin**, **stdout**, y **stderr** ya están abiertos" [**?**]

Como ya se ha dicho para trabajar con streams o archivos en C hay que incluir el header <stdio.h >. Esta biblioteca proporciona el tipo de dato FILE. Este tipo de dato es el que permite obtener una abstracción del o hacia

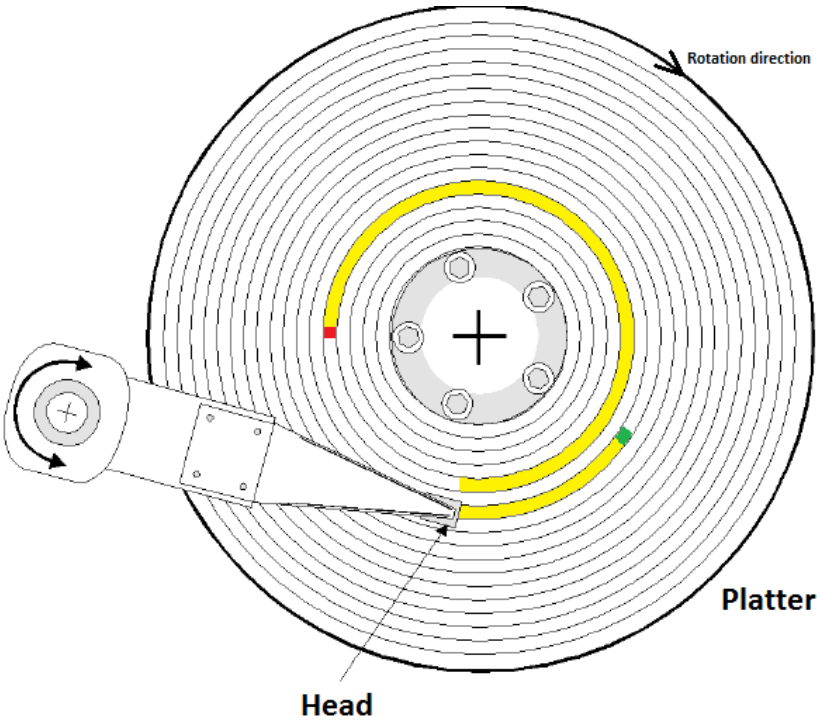


Figura 6: Esquema interno de un disco

el dispositivo con el que se está trabajando y permite manejarlo como un stream o chorro de bytes. Dado que un Stream no es mas que un chorro o tira de bytes (conceptualmente) es necesario que cada algoritmo defina una unidad mínima de almacenamiento. Esta unidad mínima de almacenamiento es la utilizada por el algoritmo para estructurar los datos almacenados, y se denomina registro lógico (no confundir con el tipo de dato registro o struct de C). El concepto de registro lógico de equivalente a la cantidad necesaria de bytes que se deben tomar para que la información almacenada en el stream tenga sentido alguno para el programa.

3.1. Funciones para el Manejo de Streams

Estas funciones de la biblioteca estándar de C sirven para el manejo de cualquier tipo de Stream, estas funciones se encuentran en **stdio.h** (STanDar Input Output).

3.1.1. fopen

```
1 FILE *fopen( const char *filename, const char *mode );
```

Abre un archivo indicado por el nombre de archivo y devuelve un puntero al stream asociado a dicho archivo. El modo se utiliza para determinar el tipo de acceso a archivos.

Parámetros	
filename	Nombre del archivo a ser asociado al stream
mode	String de un carácter que determina el modo de acceso, ver tabla
resultado	
streamptr	Un puntero a un puntero en la cual la funcion almacena el resultado o NULL en caso contrario

Modo de Acceso a un Stream:

Modo	Descripción	Acción si existe	Acción si no existe
“r”	Leer	Abrir un archivo para lectura	Falla al abrir
”w”	Escribir	Destruye lo existente y abre un archivo nuevo en blanco	Crea uno nuevo en blanco
“a”	Adjuntar	Adjunta al final del archivo	Crea uno nuevo
“r+”	Lectura extendida	Abre un archivo para lectura/escritura	Falla al abrir
”w+”	Escritura extendida	Crea un archivo para lectura/escritura	Destruye el contenido y crea uno nuevo
“a+”	Adjunto extendido	Abre un archivo para lectura/escritura	Escribe al final o crea uno nuevo

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

int main(void)
{
        FILE* fp = fopen("test.txt", "r");
        if(!fp) {
                perror("File opening failed");
                return EXIT_FAILURE;
        }

        int c; // nota: int, no char, es requerido para manejar el EOF
        while ((c = fgetc(fp)) != EOF) { // forma standard de un loop de lectura en  C  de I/O
                putchar(c);
        }

        if (ferror(fp))
                puts("I/O error when reading");
        else if (feof(fp))
                puts("End of file reached successfully");

        fclose(fp);
}
```

3.1.2. fclose

```c
int fclose( FILE *stream );
```

Cierra el stream dado. Todos aquellos datos que se encuentren en el buffer del sistema operativo serán guardados inmediatamente en el dispositivo. Ademas el stream se queda des-asociado a la estructura FILE, y los buffers creados para las operaciones de entrada o salida son destruidos por el sistema operativo.Es de vital importancia para evitar la corrupción de los archivos cerrarlos antes de finalizar la ejecución del programa. Si el puntero a FILE es utilizado una vez que fclose() es ejecutado, el comportamiento en ese caso no está definido. Dicha operación devolverá **True** (0) si la operación fue exitosa o cualquier otro valor si no.

3.1.3. feof

```c
int feof( FILE *stream );
```

Verifica si un determinado stream ha llegado al fin del archivo. Esta funcion devolvera **True (0)** mientras no sea el fin del archivo o a cualquier otro valor en caso de que lo sea. Un ejemplo tipico de uso es el siguiente:

```
        ...
        while( !feof(mi_archivo)){
         \*Declaracion y procedimientos*\
        }
        ...
```

3.1.4. Diferentes Formas de Controlar el Fin de Archivo

El control de cuando un archivo alcanzó su fin o eof (end-of-file) depende generalmente del lenguaje de programación que se esté utilizando. Normalmente se encuentran dos posible formas de detectar el fin de archivo:

- Pascal : "Eof returns True if the file-pointer has reached the end of the file, or if the file is empty."
- C: "nonzero value if the end of the stream has been reached."

A continuación se muestran estas diferencias.

3.1.5. Pascal

```pascal
Program HelloWorld;

procedure CrearArchivo ();
var
        archivo: file of integer;
        number:  integer;
begin
        assign(archivo,'datos.dat');
```

```
        rewrite(archivo);
        for number:= 1 to 20 do
                write(archivo,number);
        close(archivo);
end;

procedure MostrarArchivo ();
var
        archivo: file of integer;
        number:  integer;
begin
        assign(archivo,'datos.dat');
        reset(archivo);
        read(archivo,number);
        while(not EOF(archivo)) do
        begin
                write(number);
                read(archivo,number);
        end     ;
        close(archivo);
end;


begin {Programa ppal}
        CrearArchivo();
        MostrarArchivo ();
end.
```

La salida de la ejecución de este programa es:

12345678910111213141516171819

Por ende falta la lectura del registro numero 20. La solución para este problema se denomina "'Lectura Anticipada", y consiste en implementar un procedimiento por cada archivo que capte el fin de archivo.

```
procedure Leer (var arch:file of integer, var registro:integer, var fin:boolean );
begin
        if (eof(arch)) then fin:=true
                       else
                           begin
                                   read(arch,registro);
                                   fin:=falso;
                           end
end;
```

3.1.6. fflush

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

Si el stream apunta a un archivo de salida o de actualización cuya operación más reciente no era de entrada, la función fflush envía cualquier dato aún sin escribir al entorno local o a ser escrito en el archivo; si no, entonces el comportamiento no está definido. Si stream es un puntero nulo, la función fflush realiza el despeje para todos los streams cuyo comportamiento está descrito anteriormente.

Parámetros	
stream	el stream con el cual operar
Resultado	
Un valor no nulo	si la operación ha sido exitosa
EOF	en cualquier otro caso devuelve EOF y enciende el indicador de error del stream

3.1.7. Streams de caracteres

Un stream de caracteres puede ser interpretado como un flujo continuo de caracteres, Existen, ademas de las funciones anteriores, principalmente dos funciones que se utilizan etc. (para el manejo de este tipo de streams). Ha que destacar que un stream de caracteres tiene una naturaleza **exclusivamente secuencial**.

En todos los programas escritos en C siempre existen tres stream de caracteres que se abren, cuando el programa inicia a ejecutarse. Y se cierran cuando el programa concluye su ejecución. Estos archivos responden al nombre de :

- Standard input: Corresponde a la entrada estándar del programa, es decir al stream por el cual ingresan datos al programa, el puntero a este archivo se llama **stdin** (FILE * stdin) y por defecto es asociado al teclado de la computadora.
- Standard output: Corresponde a la salida estándar del programa, es decir al stream al que se mandan los datos para ser mostrados. El puntero a este archivo se llama **stdout** (FILE * stdout) y por defecto es asociado al monitor o consola.
- Standard error: Corresponde al stream de errores estándar, es decir al stream al cual se mandan los errores que pudieran ocurrir durante la ejecución del mismo. El puntero a este archivo se denomina **stderr** (FILE * stderr) y es asociado al monitor o consola.

3.1.8. fgetc()

La función fgetc() es utilizada exclusivamente cuando el stream es abierto en modo de solo lectura ("r"). La idea de fgetc() es la de extraer el próximo carácter del flujo de caracteres:

```
int fgetc(FILE *stream);
```

fgetc() lee el próximo carácter del flujo de caracteres o **stream** y lo devuelve como un carácter sin signo casteado a un entero, o devuelve EOF cuando llega al final del stream o un código de error.

3.1.9. fput()

La función fputc() es utilizada exclusivamente cuando el stream es abierto en modo de solo escritura ("w"). La idea de fputc() es la de escribir un carácter en un flujo de caracteres:

```
int fputc(int c, FILE *stream);
```

fputc() escribe el carácter **c**, casteado (moldeado como) un unsigned char hacia el **stream**.

3.1.10. Ejemplo

En el sistema operativo unix existe un comando llamado **cat**. Este comando recibe un archivo como parámetro e imprime su contenido en el standard output (stdout). A continuación se muestra una implementación posible de ese comando.

La idea del funcionamiento del comando es la siguiente:

1. Se chequea que los parametros recibidos sean exactamente 2.
2. Se chequea que el parametro que corresponde al stream del cual se mostrará el contenido exista, para ello se utilizar la función **fopen()**.
3. Si existe el archivo se leerá un carácter del stream correspondiente utilizando **fgetc()** y se lo escribirá en el stream que corresponde al estandar output del programa.
4. Por ultimo se cierra el stream abierto.

```
#include<stdio.h>


int main( int argc , const char * argv[]){
     FILE * stream;
     int    registro;

        if (argc!=2) {
                perror("catA: error en la cantidad de parametros que recibe el comando");
                return -1;
        }

        stream=fopen(argv[1],"r");
        if (stream==NULL){
                perror("catA: no existe el archivo");
                return -2;
        }

        while ( (registro=fgetc(stream) )!=EOF  )
                fputc(registro,stdout);

        return 0;
}
```

el programa ejemplo se ejecuta por la linead de comando de la siguiente forma:

```
./catA catA.c
```

lo que debe observarse como resultado de la ejecución es el contenido del archivo.

3.2. Stream de Datos Binarios

Si bien los streams de caracteres han sido introducidos primero, cabe destacar que son un caso especial de streams, es decir son un subconjunto de uno aun mayor: Los streams de datos binarios.

Estos pueden imaginarse como un torrente o flujo de bits que va desde un dispositivo, por ejemplo un disco, hacia el programa en ejecución.

3.2.1. fread

```
size_t fread( void *buffer, size_t size, size_t count, FILE *stream );
```

Lee de un stream llamado **stream**, una **cantidad (count)** de objetos binarios que ocupan una cantidad de memoria en bytes determinada por **size** solicitada dejándola en un **buffer** que representa un arreglo de longitud **count** de objetos de tamaño **size**.

El indicador de posición del stream avanza en **size * count** de bytes que representan además la cantidad de bytes leídos.

Parámetros que recibe fread:

- buffer: representa un puntero a un arreglo de objetos binarios (pensarlo como un tipo de datos en C) a ser leídos almacenados en la memoria.
- size: El tamaño de cada objeto binario en bytes.
- count: El numero de objetos binario a ser leídos.
- stream: El stream del cual serán leídos los datos.

fread() devuelve el número de objetos que se han leído exitosamente, fread() no reconoce entre un error o end-of-file, por ende el programador debe controlar esas condiciones con feof() o ferror para determinar cual ha ocurrido.

3.2.2. fwrite

```
size_t fwrite( const void *buffer, size_t size, size_t count,
FILE *stream );
```

Escribe en un **stream** una cantidad (**count**) de objetos binarios, de un determinado tamaño (**size**), que se encuentran almacenados en un arreglo en memoria (**buffer**). Los objetos binarios son escritos como si cada objeto fuera reinterpretado como un arreglo de unsigned char llamando a fputc() **size** veces para escribirlos en el stream, en el orden en que están en la memoria.

Parámetros que recibe fread:

- buffer: representa un puntero a un arreglo de objetos binarios (pensarlo como un tipo de datos en C) a ser escrito almacenados en la memoria.
- size: El tamaño de cada objeto binario en bytes.
- count: El numero de objetos binario a ser leídos.
- stream: El stream del cual serán leídos los datos.

Valor de retorno: Devuelve el número de objetos que se escribieron exitosamente, que puede ser menor al valor count si ocurre algún error. Si size o count es cero, fwrite devuelve cero y no realiza ninguna otra accion

```
#include <stdio.h>
#define  SIZE  5 };

int main(void){
        double a[SIZE] = {1.,2.,3.,4.,5.};

        FILE *fp = fopen("test.bin", "wb");      // must use binary mode
        fwrite(a, sizeof *a, SIZE, fp);                  // writes an array of doubles
        fclose(fp);
```

```
        double b[SIZE];
        fp = fopen("test.bin","rb");
        size_t ret_code = fread(b, sizeof *b, SIZE, fp); // reads an array of doubles
        if(ret_code == SIZE) {
        puts("Array read successfully, contents: ");
        for(int n = 0; n < SIZE; ++n) printf("%f ", b[n]);
        putchar('\n');
        } else { // error handling
                if (feof(fp))
                        printf("Error reading test.bin: unexpected end of file\n");
                else if (ferror(fp)) {
                perror("Error reading test.bin");
                }
        }
        fclose(fp);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
enum { SIZE = 5 };
int main(void)
{
        double a[SIZE] = {1, 2, 3, 4, 5};
        FILE *f1 = fopen("file.bin", "wb");
        assert(f1);
        int r1 = fwrite(a, sizeof a[0], SIZE, f1);
        printf("wrote %d elements out of %d requested\n", r1,  SIZE);
        fclose(f1);

        double b[SIZE];
        FILE *f2 = fopen("file.bin", "rb");
        int r2 = fread(b, sizeof b[0], SIZE, f2);
        fclose(f2);
        printf("read back: ");
        for(int i = 0; i < r2; i++)
        printf("%f ", b[i]);
}
```

3.3. Operaciones sobre archivos

3.3.1. remove

```
int remove( const char *fname );
```

Elimina el archivo cuyo nombre se encuentra en la constante fname. Devuelve cero si la operación fue exitosa o devuelve un valor distinto de cero si hubo un error

3.3.2. rename

```
int rename( const char *old_filename, const char *new_filename );
```

Cambia el nombre de un archivo. El archivo cuyo nombre esta dado por el valor de old_filename se renombre con el nombre dado por new_filename. si el valor new_filename ya existe el comportamiento depende de la implementacion.

4. Tipos de acceso secuencial a los datos

El tipo de dispositivo sobre o desde el cual se esté trabajando definirá el tipo de acceso a los datos. Si se está utilizando una cinta magnética únicamente podrán leerse o escribirse datos una vez que la cinta pase por debajo del cabezal de lectura o escritura. Este tipo de acceso a datos se define como *Acceso Secuencial*, ya que para acceder al n-esimo elemento deseado, los (n-1) datos anteriores deben pasar por debajo del cabezal de lectura o escritura según corresponda. Por ellos este tipo de dispositivo solo permite un estado de funcionamiento a la vez. O es abierto para escribir datos en el medio o es abierto par leer datos del medio, *nunca para ambas cosas*.

4.1. Ejemplos Básicos

A continuación se describirán algunos ejemplos básicos de utilización de archivos Binarios acceso secuencial.

4.1.1. Creación

Problema:Crear un archivo de los 10 primeros números enteros, cuyo nombre es "enteros.dat". Solución: Se debe generar un stream utilizando como registro lógico una variable entera que contenga los valores de los 10 primeros números pares.

```c
#include<stdio.h>

int main(void){
        FILE*   archivo;
        int     registro;
        int     contador;
        int     escritos;

        archivo=fopen("enteros.dat","w");

        if(!archivo){
                perror("Error de Apertura de archivo");
                return -1;
        }

        registro =2;
        contador =1;

        while (contador <=10) {
                escritos=fwrite(&registro,sizeof(registro),1,archivo);
                if (!escritos){
                        perror("Error de Escritura en el archivo");
                        return -1;
                }
                contador++;
                registro +=2;
        }
        fclose(archivo);
        return 0;
}
```

4.1.2. Lectura

Problema: Dado un archivo cuyos registros lógicos son números enteros leer su contenido completo, su nombre es "enteros.dat".
Solución: Se debe abrir un stream para lectura y utilizando como registro lógico una variable entera que contenga los valores de cada uno de los registros del stream o archivo.

```c
#include<stdio.h>

int main(void){
        FILE*   archivo;
        int     registro;
        int     leidos;
        int     contador;

        archivo=fopen("enteros.dat","r");

        if(!archivo){
                perror("Error de Apertura de archivo");
                return -1;
        }

        contador=1;
        leidos=fread(&registro,sizeof(registro),1,archivo);
        while ( !feof(archivo)  ) {
                printf("registro lógico %i: %i\n",contador,registro);
                leidos=fread(&registro,sizeof(registro),1,archivo);
                contador++;
        }
        fclose(archivo);
        return 0;
}
```

4.1.3. Eliminación de registros

Tal vez sea este el ejemplo más importante de cómo deben manejarse los archivos de acceso secuencial. Problema: Dado un archivo cuyos registros lógicos son números enteros eliminar los números menores a 10, su nombre es

"enteros.dat".
Solución: Para poder solucionar dicho problema se debe tener en cuenta que un archivo de acceso secuencial o stream de acceso secuencial no puede abrirse simultáneamente para lectura y para escritura, esto es debido justamente a las restricciones del acceso secuencial. Para poder solucionar el problema se debe utilizar el archivo original el cual será leído y otro archivo en el que se almacenara? los registros que corresponda segun el procesamiento del problema en este caso solo los números mayores que 10.

```c
#include<stdio.h>
int main(void){
        FILE*   arch_entrada;
        FILE*   arch_salida;
        int     registro;
        int     leidos;
        int     escritos;

        arch_entrada=fopen("enteros.dat","r");
        arch_salida=fopen("enteros_nuevo.dat","w");

        if(!(arch_entrada) ){
                perror("Error de Apertura de archivo de entrada");
                return -1;
        }

        if(!(arch_salida) ){
                perror("Error de Apertura de archivo de salida");
                return -1;
        }

        leidos=fread(&registro,sizeof(registro),1,arch_entrada);
        while ( !feof(arch_entrada) ) {
                if(registro>10) {
                escritos=fwrite(&registro,sizeof(registro),1,arch_salida);
                if (!escritos){
                        perror("Error de Escritura en el archivo de salida");
                        return -1;
                        }
                }
        leidos=fread(&registro,sizeof(registro),1,arch_entrada);
        }
        fclose(arch_entrada);
        fclose(arch_salida);
        return 0;
}
```

4.1.4. Eliminación de registros V 2.0

Esta versión del problema agrega la eliminación y el renombrado del archivo a ser borrado. Problema: Dado un archivo cuyos registros lógicos son números enteros eliminar los números menores a 10, su nombre es "enteros.dat".
Solución: Para poder solucionar dicho problema se debe tener en cuenta que un archivo de acceso secuencial o stream de acceso secuencial no puede abrirse simultáneamente para lectura y para escritura, esto es debido justamente a las restricciones del acceso secuencial. Para poder solucionar el problema se debe utilizar el archivo original el cual será leído y otro archivo en el que se almacenara los registros que corresponda según el procesamiento del problema en este caso solo los números mayores que 10.

```c
#include<stdio.h>
int main(void){
        FILE*   arch_entrada;
        FILE*   arch_salida;
        int     registro;
        int     leidos;
        int     escritos;

        arch_entrada=fopen("enteros.dat","r");
        arch_salida=fopen("enteros_nuevo.dat","w");

        if(!(arch_entrada) ){
                perror("Error de Apertura de archivo de entrada");
                return -1;
        }

        if(!(arch_salida) ){
                perror("Error de Apertura de archivo de salida");
                return -1;
```

```
        }

        leidos=fread(&registro,sizeof(registro),1,arch_entrada);
        while ( !feof(arch_entrada) ) {
                if(registro>10) {
                        escritos=fwrite(&registro,sizeof(registro),1,arch_salida);
                if (!escritos){
                        perror("Error de Escritura en el archivo de salida");
                        return -1;
                        }
                }
                leidos=fread(&registro,sizeof(registro),1,arch_entrada);
        }
        fclose(arch_entrada);
        fclose(arch_salida);
        remove("enteros.dat");
        rename("enteros_nuevo.dat", "enteros.dat");
        return 0;
}
```

El programa anterior demuestra cual es la forma exacta de utilizar u operar con archivos secuenciales. Siempre para modificar un archivo de acceso secuencial es necesario crear uno nuevo con las modificaciones y si se requiere que el nuevo se llame exactamente igual al anterior, el primero debe borrarse y este último debe re nombrarse.

4.2. Operaciones con Archivos de acceso secuencial

Una vez que las operaciones básicas sobre archivos secuenciales están planteadas otras más complejas se derivan de las primeras. Estas operaciones son, al igual que con los vectores:

- Mezcla
- Unión
- Intersección
- Diferencia
- Actualización

Por una cuestión de facilidad en todos los ejemplos que se muestran a continuación el registro lógico de los archivos son un numero entero.

4.2.1. Mezcla

La mezcla entre dos o mas archivos binarios de acceso secuencial consiste en generar un único archivo binario de acceso secuencial ordenado a partir de dos o mas archivos binarios secuenciales ordenados por el mismo criterio, ver Figura **??**. Consideraciones:

- Los archivos a ser mezclados deben estar ordenados todos por el mismo criterio de ordenamiento.
- El archivo resultante debe estar ordenado por el mismo criterio de los archivos que han sido mezclados.
- Los registros de cada archivo únicamente pueden *ser leí dos una sola vez en todo el proceso*.
- De existir registros repetidos en más de un archivo éstos deben ser escritos y aparecer tantas veces se repitan en el archivo resultante.

4.2.2. Union

La unión entre dos o mas archivos binarios de acceso secuencial consiste en generar un único archivo binario de acceso secuencial ordenado a partir de dos o mas archivos ordenados por el mismo criterio, utilizando el mismo criterio que la unión de conjuntos, ver Figura **??**. Consideraciones:

- Los archivos a ser unidos deben estar ordenados todos por el mismo criterio de ordenamiento.
- El archivo resultante debe estar ordenado por el mismo criterio de los archivos que participen del proceso de unión.
- Los registros de cada archivo únicamente pueden *ser leídos una sola vez en todo el proceso*.
- De existir registros repetidos en mas de un archivo únicamente pueden aparecer una solo vez en el archivo resultante.

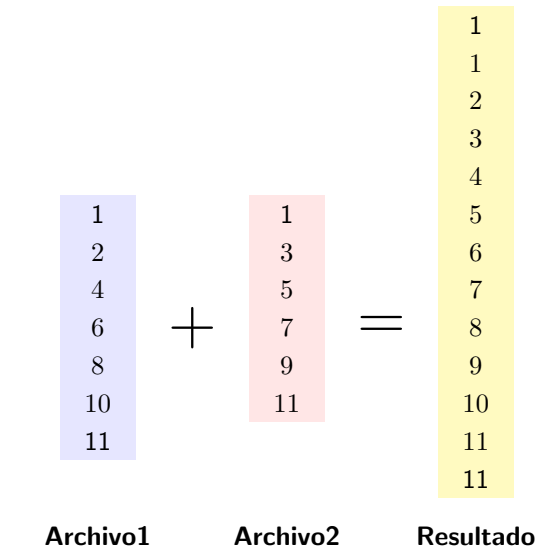


Figura 7: Mezcla

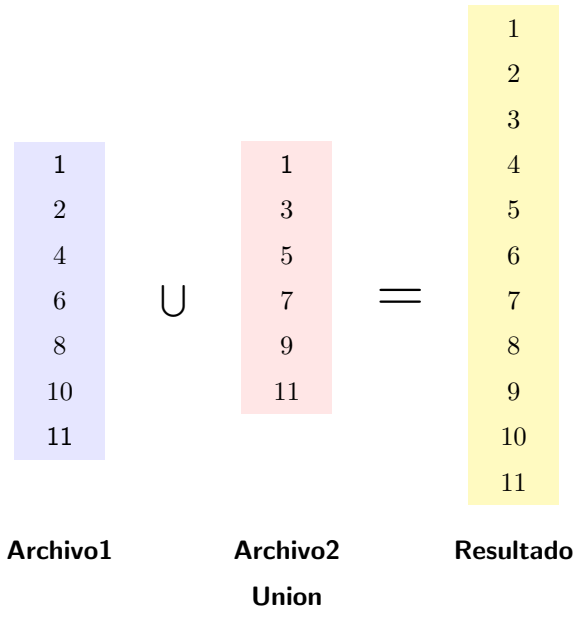


Figura 8: Unión

4.2.3. Intersección

La intersección entre dos o mas archivos binarios de acceso secuencial consiste en generar un único archivo binario de acceso secuencial ordenado a partir archivos anteriores, cuyo contenido sean los registros en común en todos los archivos, ver Figura **??**. Consideraciones:

- Los archivos en los que se busca la intersección deben estar ordenados todos por el mismo criterio de ordenamiento.
- El archivo de salida debe estar ordenado por el mismo criterio de los archivos que han participado de la intersección.
- Los registros de cada archivo únicamente pueden *ser leídos una sola vez en todo el proceso*.
- Unicamente los registros repetidos deben aparecer una sola vez en el archivo resultante.

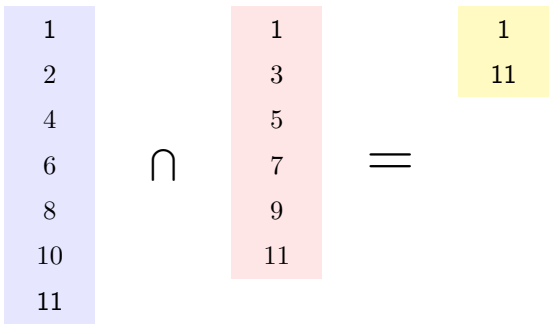


Figura 9: Intersección

4.2.4. Diferencia

La diferencia entre dos archivos binarios de acceso secuencial consiste en generar un único archivo binario de acceso secuencial ordenado a partir de archivos, cuyo contenido sean los registros en un archivo y no en el otro, ver Figura **??**. Consideraciones:

- Los archivos de los cuales se busca la diferencia deben estar ordenados todos por el mismo criterio de ordenamiento.
- El archivo de salida debe estar ordenado por el mismo criterio de los archivos de entrada.
- Los registros de cada archivo únicamente pueden *ser leídos una sola vez en todo el proceso*.
- Únicamente los registros que cumplan (Archivo1 - Archivo2) o viceversa podrán estar en el archivo de salida.

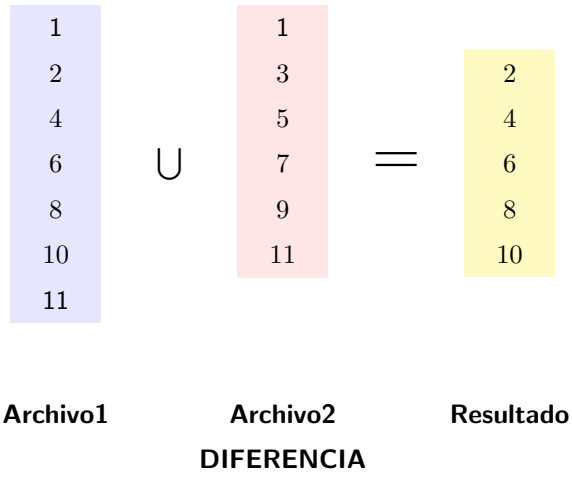
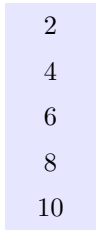


Figura 10: Diferencia (Archivo1-Archivo2)

4.2.5. Resolución

Para encarar la resolución de las operaciones sobre archivos presentadas anteriormente se debe tener en cuenta y considerar que a priori no se conoce la longitud de cada archivo. Por ende el algoritmo que se plantee debe servir para cualquier situación, ver Figura **??**, Figura **??**, Figura **??**. El algoritmo que se plantee para solucionar la mezcla, intersección, unión y diferencia debe contemplar todos los casos posibles planteados anteriormente.

Ejemplo Complejo a continuación se solucionará la unión de dos archivos binarios de acceso secuencial. El primer archivo es llamado "inginfo.dat" y posee los siguientes datos de los alumnos de la carrera de Ingenieria en informatica de la Facultad, que posee los siguientes datos:

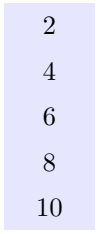


Archivo1 **Archivo2**

Figura 11: Ambos archivos tienen la misma cantidad de registros

Archivo1 **Archivo2**

Figura 12: El Archivo 1 tiene más cantidad de registros que el Archivo 2



Archivo1 **Archivo2**

Figura 13: El Archivo 1 tiene menos cantidad de registros que el Archivo 2

- padron (entero largo)
- nombre (string 20)
- apellido (string 20)
- dni (string 8)

Y otro archivo llamado "licsist.dat" con la misma estructura, en el cual se almacenan los datos de los alumnos de la facultad que siguen la carrera de Licenciado en Sistemas. Ambos archivos se encuentran ordenados ascendentemente por número de padron. Se necesita un programa que obtenga un archivo con la misma estructura de todos aquellos alumnos que cursan una carrera de informatica, los que hagan la simultaneidad deben estar solo una vez.

Solución Para obtener la solución del problema en primer lugar se genera un set de pruebas:

```
73519;Mariano Mendez;23511444
74234;Leslie Winkle;34567821
89083;Melina Valeria Diaz;12345678
89121;Rajesh Koothrappali;87654321
89122;Sheldom Lee Cooper;56784321
89231;Leonar Leaky Hofstadter;43215678
89232;Howard Joel Wolowitz;12563487
90001;Wil Wheaton;32124565

89082;Bernadette Marian Rostenkowski;52368412
89121;Rajesh, Koothrappali;87654321
89122;Sheldom Lee Cooper;56784321
89123;Amy Farrah Fowler;14698752
89124;Barry Kripke;78965321
```

Con estos datos se procederá aprobar el algoritmo, de debe generar los dos archivos en formato binario. Posteriormente se debe tener en cuenta que los archivos (A y B) pueden tener cualquier configuración respecto de la cantidad de registros, por ende la solución debe contemplar:

1. ambos archivos podrían tener la misma cantidad de registros.
2. El archivo A podría tener mas registros que el archivo A
3. El archivo B podría tener más registros que el archivo A.

Se ha de tener en cuenta que uno y solo UNO de estos escenarios es posibles en cada caso en particular pero el algoritmo debe contemplarlos a los tres.

```
typedef struct alumno{
        long    padrón;
        char    nombre[50];
        long    dni;
}alumno_t;
```

```c
#include<stdio.h>
#include<string.h>
#include"alumnos.h"


int  fileunion( const char *name_one_file, const char * name_second_file, const char *
    name_result_file ){

        FILE * one_file;
        FILE * second_file;
        FILE * result_file;
        alumno_t reg_one_file;
        alumno_t reg_second_file;

        /*
        *
        *  apertura de los archivos y validación
        *
        */
        one_file=fopen(name_one_file,"r");
        if (!one_file){
                perror("Error apertura primer archivo");
                return -1;
        }

        second_file=fopen(name_second_file,"r");
        if (!second_file){
                perror("Error apertura segundo archivo");
                return -1;
        }

        result_file=fopen(name_result_file,"w");
        if (!result_file){
                perror("Error apertura archivo de salida");
                return -1;
        }

        /*
        *
        *  proceso de los archivos
        *
        */

        fread(&reg_one_file,sizeof reg_one_file,1,oneFile);
        fread(&reg_second_file,sizeof reg_second_file,1,secondFile);

        while (!feof(oneFile) && !feof(secondFile)  ) {
                if(reg_one_file.padron==reg_second_file.padron){
                        fwrite(&reg_one_file,sizeof reg_one_file,1,result_file);
                        fread(&reg_one_file,sizeof reg_one_file,1,oneFile);
                        fread(&reg_second_file,sizeof reg_second_file,1,secondFile);
                } else if (reg_one_file.padron<reg_second_file.padron){
                                        fwrite(&reg_one_file,sizeof reg_one_file,1,result_file
    );
                                        fread(&reg_one_file,sizeof reg_one_file,1,oneFile);
                                } else {
                                        fwrite(&reg_second_file,sizeof reg_second_file,1,
    result_file);
                                        fread(&reg_second_file,sizeof reg_second_file,1,
    secondFile);
                                }
                }

        while(!feof(secondFile)){
                fwrite(&reg_second_file,sizeof reg_second_file,1,result_file);
                fread(&reg_second_file,sizeof reg_second_file,1,secondFile);
        }

        while(!feof(oneFile)){
                fwrite(&reg_one_file,sizeof reg_one_file,1,result_file);
                fread(&reg_one_file,sizeof reg_one_file,1,oneFile);
        }

        /*
        *
        *  cierre de los archivos
```

```
73          *
74          */
75          fclose(oneFile);
76          fclose(secondFile);
77          fclose(result_file);
78          return 1;
79  }
80
81  void mostrarArchivo(const char *nameFile){
82          FILE *          unStream;
83          Talumno         unRegistro;
84
85          unStream=fopen(nameFile,"r");
86
87          printf("Registros del archivo: %s \n",nameFile);
88          fread(&unRegistro,sizeof unRegistro,1,unStream);
89          while ( !feof(unStream) ){
90                  printf("%li\t%s\t%li\n", unRegistro.padron,unRegistro.nombre,unRegistro.dni);
91                  fread(&unRegistro,sizeof unRegistro,1,unStream);
92          }
93          printf("\n");
94          fclose(unStream);
95  }
96
97  int main(){
98          mostrarArchivo("licsist.dat");
99          mostrarArchivo("inginfo.dat");
100         if (fileunion("licsist.dat","inginfo.dat","union.dat")>0 )      mostrarArchivo("union.
    dat");
101         return 0;
102 }
```

El resultado obtenido tras la ejecución del programa es el siguiente:

```
Registros del archivo: licsist.dat
89082 Bernadette Marian Rostenkowski 52368412
89121 Rajesh, Koothrappali 87654321
89122 Sheldom Lee Cooper 56784321
89123 Amy Farrah Fowler 14698752
89124 Barry Kripke 78965321

Registros del archivo: inginfo.dat
73519 Mariano Mendez 23511444
74234 Leslie Winkle 34567821
89083 Melina Valeria Diaz 12345678
89121 Rajesh Koothrappali 87654321
89122 Sheldom Lee Cooper 56784321
89231 Leonar Leaky Hofstadter 43215678
89232 Howard Joel Wolowitz 12563487
90001 Wil Wheaton 32124565

Registros del archivo: union.dat
73519 Mariano Mendez 23511444
74234 Leslie Winkle 34567821
89082 Bernadette Marian Rostenkowski 52368412
89083 Melina Valeria Diaz 12345678
89121 Rajesh, Koothrappali 87654321
89122 Sheldom Lee Cooper 56784321
89123 Amy Farrah Fowler 14698752
89124 Barry Kripke 78965321
89231 Leonar Leaky Hofstadter 43215678
89232 Howard Joel Wolowitz 12563487
90001 Wil Wheaton 32124565
```