FACULTAD INGENIERÍA & ARQUITECTURA

Lógica y Algoritmos

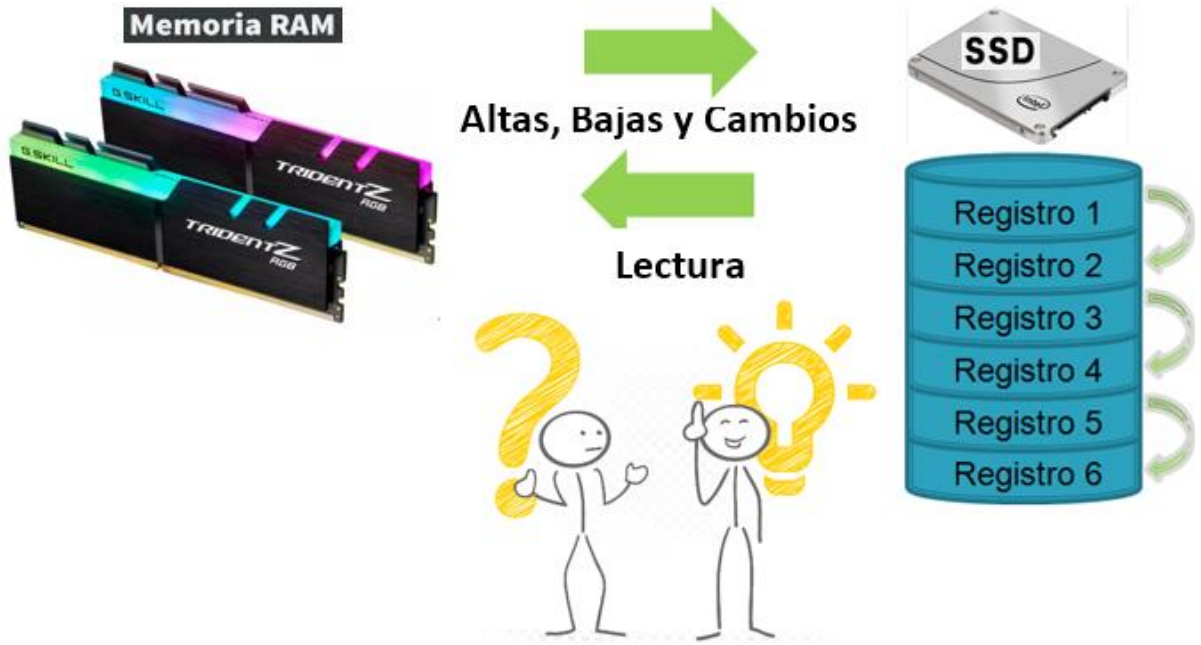
Unidad IV: Control de Flujo

Tema: Procesamiento de Archivos

Colectivo de asignatura 1S2024 | Mayo, 2024

Contenido

Procesamiento de Archivos	
►Punteros	►Archivos secuenciales y direccionables
►Flujo (*stream*)	►Operaciones con archivos (crear, guardar, abrir y leer)
►Modos de apertura de un archivo	►Ejemplos de programas con Archivos de Texto



Punteros (Apuntadores)

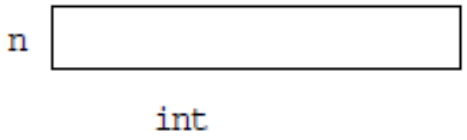
DIRECCIONES EN MEMORIA

Cuando una variable se declara, se asocian tres atributos fundamentales con la misma: su *nombre*, su *tipo* y su *dirección* en memoria.

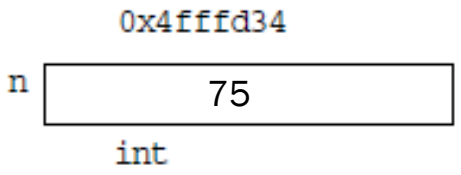
Ejemplo:

Asocia al nombre `n`, el tipo `int` y la dirección de alguna posición de memoria donde se almacena el valor de `n`.

```
int n;
```



La caja del ejemplo representa la posición de almacenamiento en memoria. El nombre de la variable está a la izquierda de la caja, la dirección de variable está encima de la caja y el tipo de variable está debajo en la caja. Si el valor de la variable se conoce, se representa en el interior de la caja:



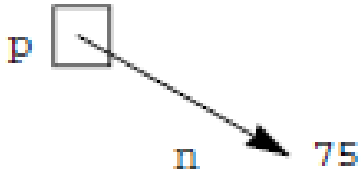
Punteros (Apuntadores)

Cada vez que se declara una variable, el compilador establece un área de memoria para almacenar el contenido de la variable. Cuando se declara una variable int (entera), el compilador asigna dos bytes del espacio para esa variable, y se sitúa en una posición específica de la memoria conocida como dirección de memoria. Cuando se referencia (se hace uso) al valor de la variable, el compilador de C accede automáticamente a la dirección de memoria donde se almacena el entero. Se puede ganar en eficacia en el acceso a esta dirección de memoria utilizando un puntero.

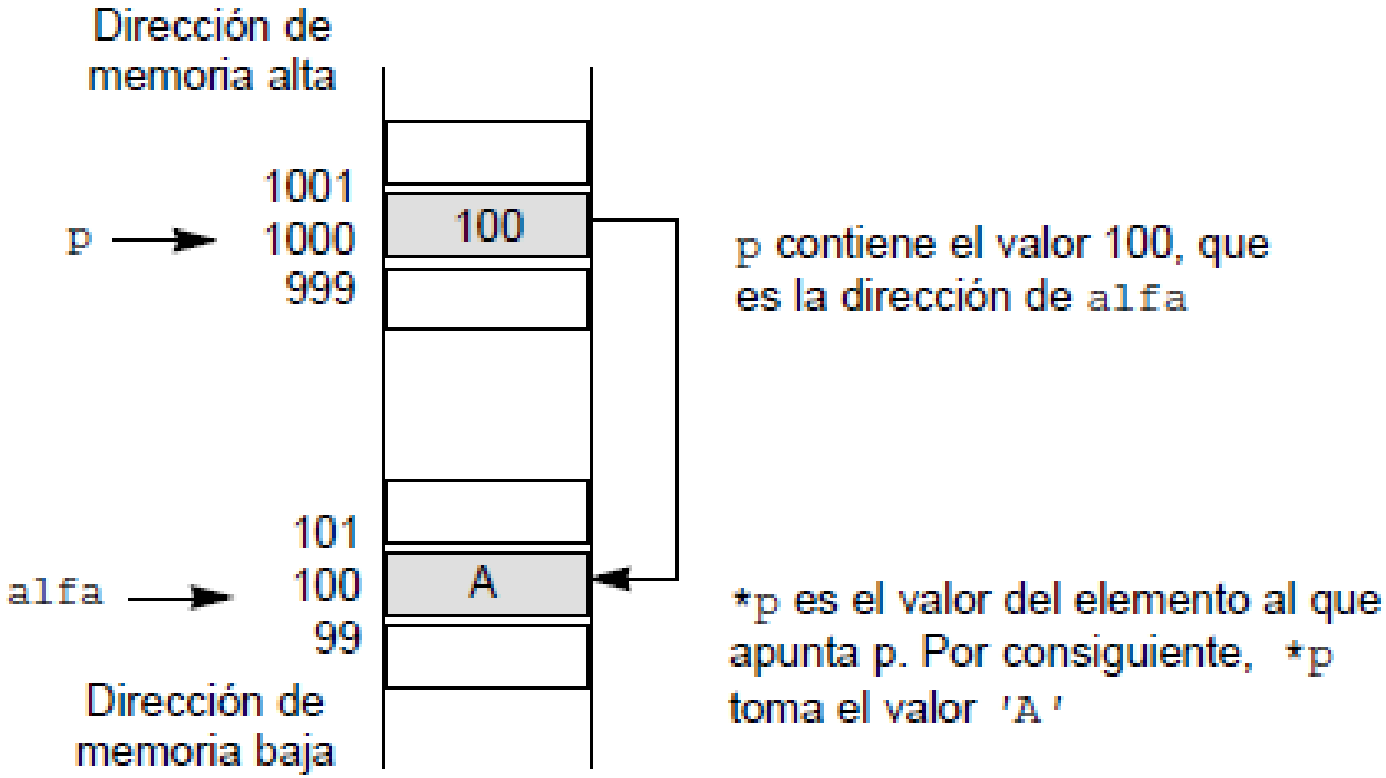
Una variable que se declara en C, tiene una dirección asociada con ella. Un puntero, es una dirección de memoria. El concepto de punteros tiene correspondencia en la vida diaria. Cuando se envía una carta por correo, su información se entrega basada en un puntero que es la dirección de esa carta. Cuando se llama por teléfono a una persona, se utiliza un puntero (el número de teléfono que se marca). Un puntero en C, también indica dónde encontrar los datos que están asociados con una variable. Los punteros se rigen por estas reglas básicas:

- Un puntero es una variable como cualquier otra.
- Una variable puntero, contiene una dirección que apunta a otra posición en memoria.
- En esa posición de memoria, se almacenan los datos a los que apunta el puntero.
- Un puntero, apunta a una variable de memoria.

Punteros (Apuntadores)



El valor de un puntero es una dirección. La dirección depende del estado de la computadora en la cual se ejecuta el programa.



Relaciones entre `*p` y el valor de `p` (dirección de `alfa`).

Flujos (Stream)

Flujos

Un flujo (*stream*), es una abstracción que se refiere a un *flujo o corriente* de datos que fluye entre un origen o fuente (*productor*) y un destino o sumidero (*consumidor*). Entre el origen y el destino, debe existir una conexión o canal (*"pipe"*) por la que circulen los datos. La apertura de un archivo, supone establecer la conexión del programa con el dispositivo que contiene al archivo; es decir, por el canal que comunica el archivo con el programa, van a fluir las secuencias de datos. Hay tres flujos o canales abiertos automáticamente:

```
extern FILE *stdin;
extern FTLE *stdout;
extern FILE *stderr;
```

Estas tres variables, se inicializan al comenzar la ejecución del programa, y permiten admitir secuencias de caracteres en modo texto. Tienen el siguiente cometido:

stdin : Asocia la entrada estándar (teclado) con el programa.
stdout : Asocia la salida estándar (pantalla) con el programa.
stderr : Asocia la salida de mensajes de error (pantalla) con el programa.

Flujos (Stream)

Puntero FILE

Los archivos se almacenan en dispositivos externos como discos duros, discos sólidos, USB, etc.; tienen un nombre y unas características. En el programa, el archivo tiene un nombre interno que es un puntero a una estructura predefinida (*puntero a archivo*). Esta estructura, contiene información sobre el archivo, tal como la dirección del *buffer* que utiliza, el modo de apertura del archivo, el último carácter leído del *buffer,* y otros detalles que generalmente el usuario no necesita saber. El identificador del tipo de la estructura es FILE y está declarada en el archivo de cabecera stdio.h

Apertura de un archivo

Para procesar un archivo en C (y en todos los lenguajes de programación), la primera operación que hay que realizar es abrir el archivo. La apertura del archivo supone conectar el archivo externo con el programa, e indicar cómo va a ser tratado el archivo: binario, de caracteres, etc. El programa accede a los archivos a través de un puntero a la estructura `FILE`, la función de apertura devuelve dicho puntero. La función para abrir un archivo es `fopen()`, el formato de llamada es:

```
fopen(nombre_archivo, modo);
```

nombre	≡ cadena	*Contiene el identificador externo del archivo.*
modo	≡ cadena	*Contiene el modo en que se va a tratar el archivo.*

La función devuelve un puntero a `FILE`, a través de dicho puntero el programa hace referencia al archivo. La llamada a `fopen()` se debe de hacer de tal forma, que el valor que devuelve se asigne a una variable puntero a `FILE`, para así después referirse a dicha variable.

Esta función puede detectar un error al abrir el archivo, por ejemplo, que el archivo no exista y se quiera leer, entonces devuelve `NULL`.

Flujos (Stream)

NULL y EOF (End Of File)

Las funciones de biblioteca que devuelven un puntero (`strcpy()`, `fopen()...`), especifican que si no puede realizar la operación (generalmente si hay un error) devuelven `NULL`. Ésta es una macro definida en varios archivos de cabecera, entre los que se encuentran `stdio.h` y `stdlib.h` .

Las funciones de biblioteca de E/S de archivos, generalmente empiezan por `f` de `file`, tienen especificado que son de tipo entero,de tal forma que si la operación falla,devuelven `EOF`, también devuelven `EOF` para indicar que se ha leído el fin de archivo. Esta macro está definida en `stdio.h` .

Cierre de archivos

Los archivos en C,trabajan con una memoria intermedia, *buffer*. La entrada y salida de datos se almacena en ese *buffer*, volcándose cuando está lleno. Al terminar la ejecución del programa,podrá ocurrir que haya datos en el *buffer*, si no se volcasen en el archivo,éste quedaría sin las últimas actualizaciones. Siempre que se termina de procesar un archivo, siempre que se termine la ejecución del programa;los archivos abiertos hay que cerrarlos para que, entre otras acciones, se vuelque el *buffer*.

La función `fclose(puntero_file)`, cierra el archivo asociado al `puntero_file`, devuelve `EOF` si ha habido un error al cerrar. El prototipo de la función se encuentra en `stdio.h` y es:

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

Flujos (Stream)

Funciones `fprintf()` y `fscanf()` (File Print Format y File Scan Format)

Las funciones `printf()` y `scanf()`, permiten escribir o leer variables de cualquier tipo de dato estándar, los códigos de formato (`%d`, `%f`...) indican a C la transformación que debe de realizar con la secuencia de caracteres (conversión a entero…). La misma funcionalidad tienen `fprintf()` y `fscanf()` con los flujos (archivos asociados) a que se aplican. Estas dos funciones tienen como primer argumento, el puntero a `FILE` asociado al archivo de texto.

Función `feof()` (Flag End Of File)

Diversas funciones de lectura de caracteres, devuelven `EOF` cuando leen el carácter de fin de archivo. Con dicho valor, que es una macro definida en `stdio.h`, ha sido posible formar bucles para leer un archivo completo. La función `feof()` realiza el cometido anterior, devuelve un valor distinto de 0 (`true`) cuando se lee el carácter de fin de archivo, en caso contrario devuelve 0 (`false`).

Modos de Apertura de un Archivo

Modos de apertura de un archivo

Al abrir el archivo, `fopen()` espera como segundo argumento el modo de tratar el archivo. Fundamentalmente se establece si el archivo es de lectura, escritura o añadido y si es de texto o binario.

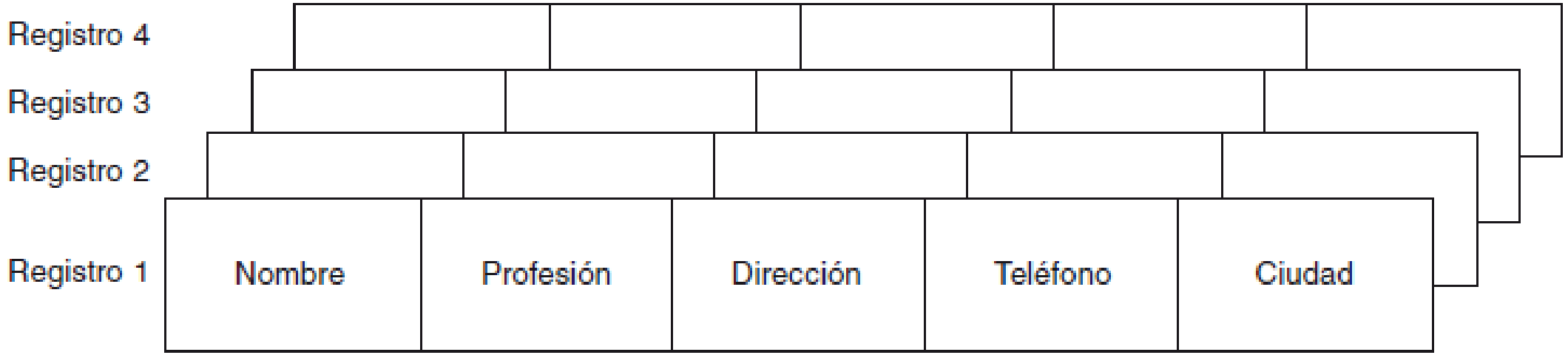
Aparentemente en estos modos, no se ha establecido el tipo del archivo, de texto o binario. Siempre por defecto, el archivo es de texto. Para no depender del entorno, es mejor indicar si es de texto o binario. Se utiliza la letra *b* para modo binario como último carácter de la cadena `modo` (también se puede escribir como carácter intermedio). Algunos compiladores admiten la letra *t* para indicar archivo de texto. Por consiguiente, los modos de abrir un archivo de texto:

Modo	**Significado** (read, write, append)
`"r"`	Abre para lectura.
`"w"`	Abre para crear nuevo archivo (si ya existe se pierden sus datos).
`"a"`	Abre para añadir al final.
`"r+"`	Abre archivo ya existente para modificar (leer/escribir).
`"w+"`	Crea un archivo para escribir/leer (si ya existe se pierden los datos).
`"a+"`	Abre el archivo para modificar (escribir/leer) al final. Si no existe es como w+.

Archivos Secuenciales y Direccionables

Archivos

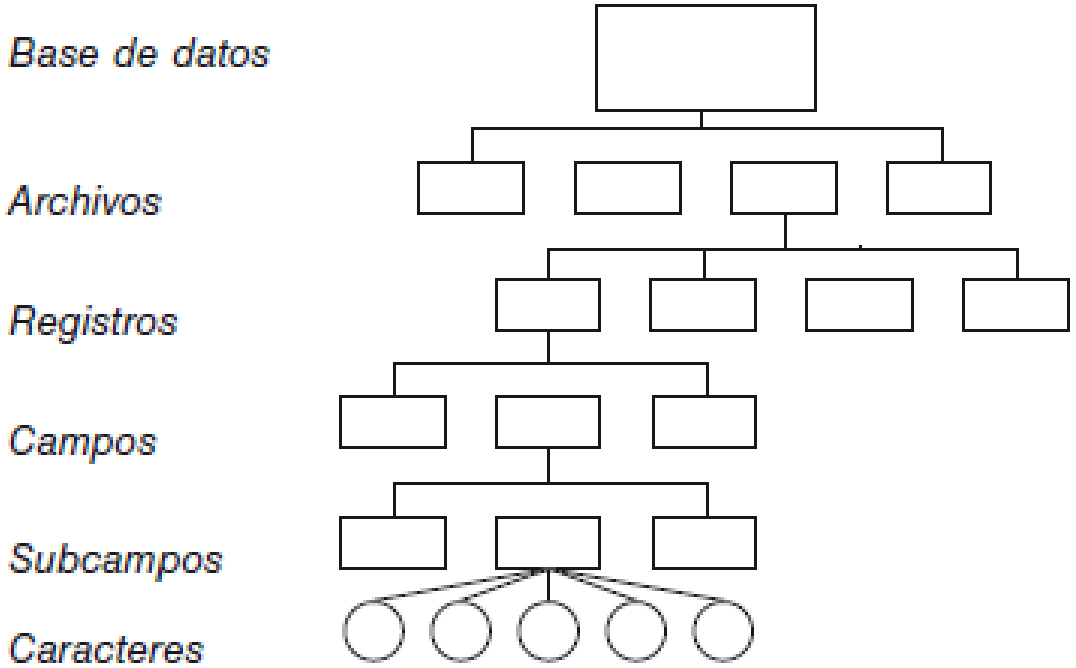
Un *fichero* (*archivo*) de datos –o simplemente un **archivo**–, es una colección de registros relacionados entre sí con aspectos en común, y organizados para un propósito específico. Por ejemplo, un fichero de una clase escolar, contiene un conjunto de registros de los estudiantes de esa clase. Otros ejemplos pueden ser: la nómina de una empresa, inventario, stock de productos, etc.



Estructura de un archivo de “Suscriptores de una revista informática”.

Archivos Secuenciales y Direccionables

Entonces, un archivo es un grupo de registros relacionados. Así, una universidad puede tener muchos estudiantes y profesores; y un archivo de estudiantes, contiene un registro para cada estudiante. Un archivo de una universidad, puede contener miles de registros, o incluso, miles de millones de caracteres de información.



Estructuras jerárquicas de datos.

Archivos Secuenciales y Direccionables

El soporte, es el medio físico donde se almacenan los datos. Los tipos de soporte utilizados en la gestión de archivos, son: *Soportes secuenciales y Soportes direccionables.*

Organización de los archivos

Según las características del soporte empleado y el modo en que se han organizado los registros, se consideran dos tipos de acceso a los registros de un archivo: *Acceso secuencial y Acceso directo.*

El ***acceso secuencial,*** implica el acceso a un archivo según el orden de almacenamiento de sus registros, uno tras otro.

El ***acceso directo,*** implica el acceso a un registro determinado, sin que ello implique la consulta de los registros precedentes. Este tipo de acceso sólo es posible con soportes direccionables.

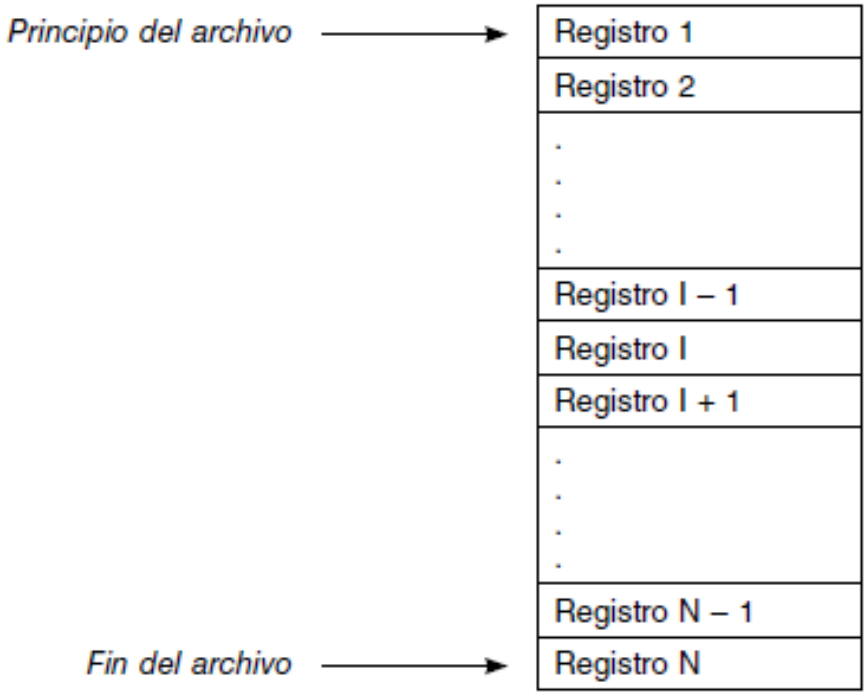
En general, se consideran tres organizaciones fundamentales en los archivos:

- *Organización secuencial.*
- *Organización directa o aleatoria ("random").*
- *Organización secuencial indexada ("indexed").*

Archivos Secuenciales y Direccionables

Organización secuencial

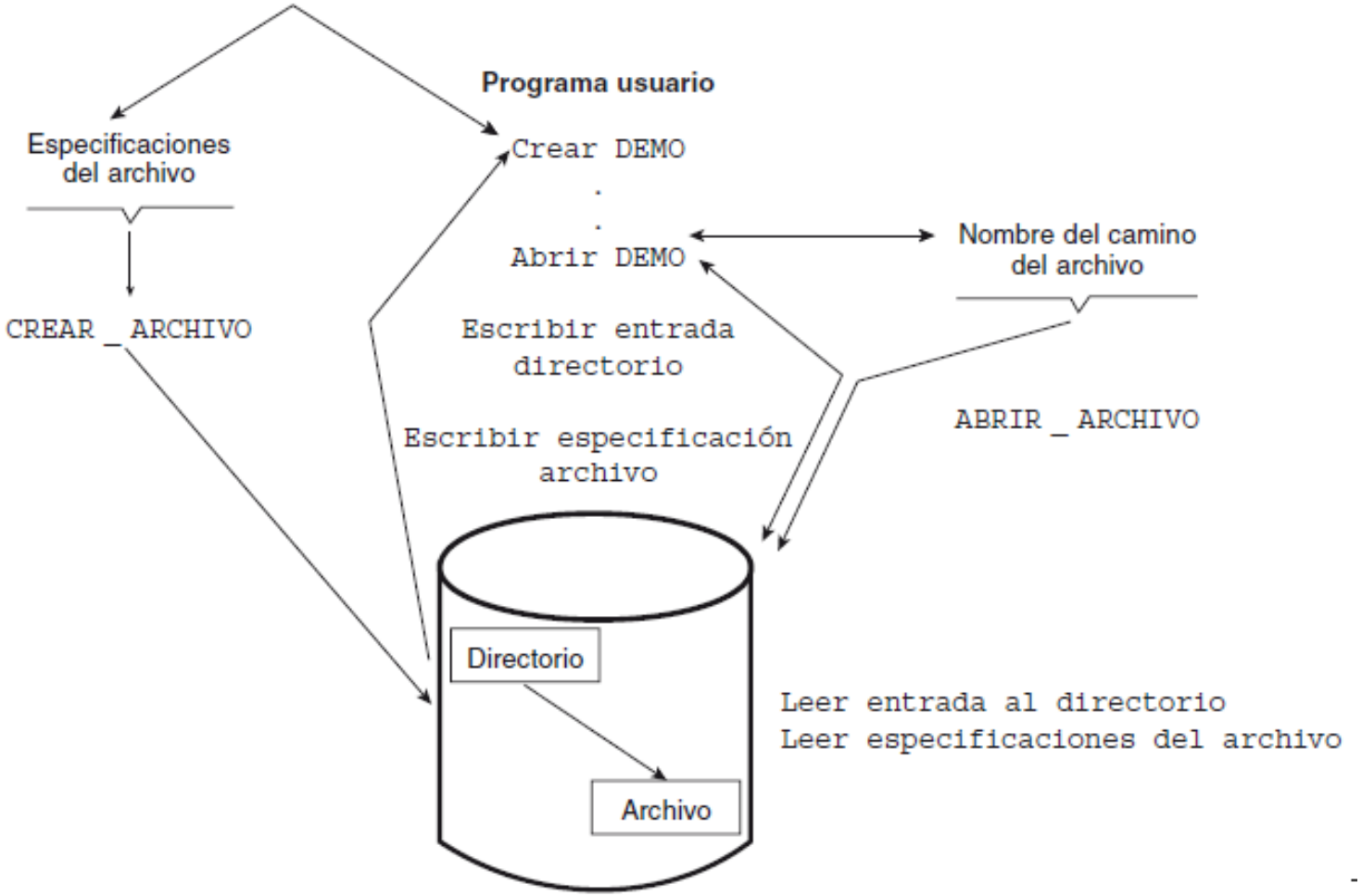
Un archivo con organización secuencial, es una sucesión de registros almacenados consecutivamente sobre el soporte externo, de tal modo que para acceder a un registro *n* dado, es obligatorio pasar por todos los *n – 1* artículos que le preceden.
Los registros se graban consecutivamente cuando el archivo se crea, y se debe acceder consecutivamente cuando se leen dichos registros.



Organización secuencial.

Operaciones con Archivos

Operaciones con archivos (crear, abrir y leer)



Organización secuencial.

Ejemplos de Programas con Archivos de Texto

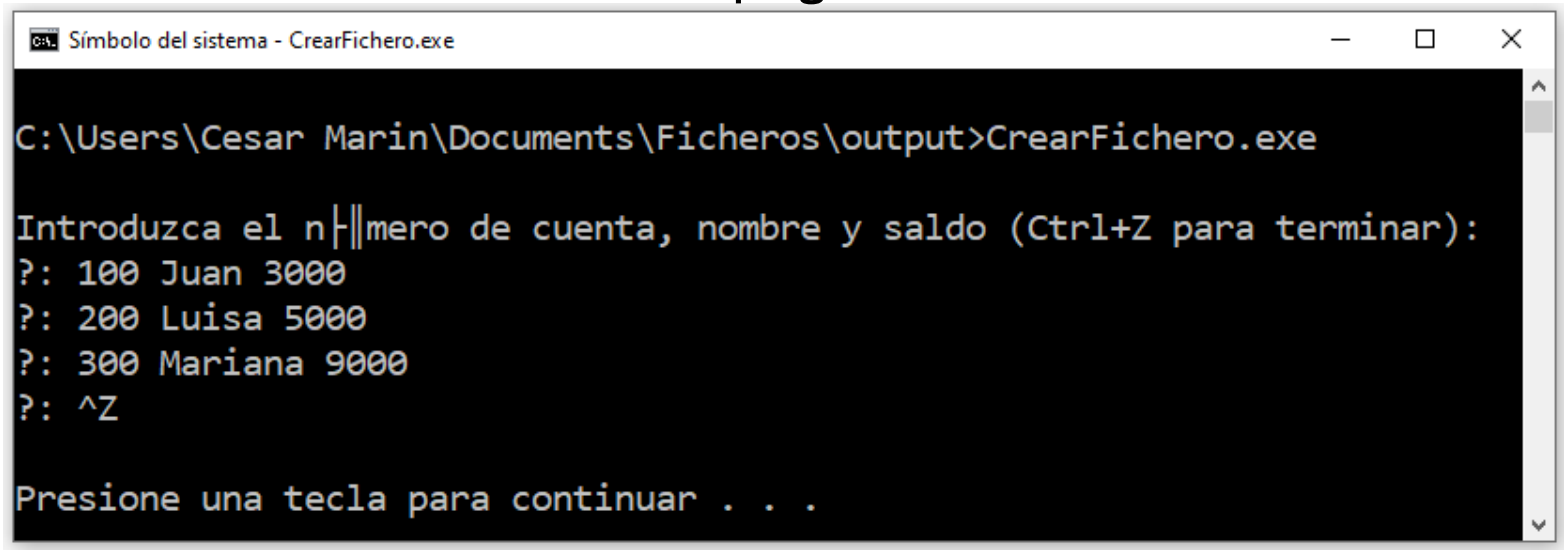
Programa en C: Creación de un archivo secuencial de texto (almacenamiento persistente: RAM → archivo → disco duro). Inserción de datos.

CrearFichero.cpp

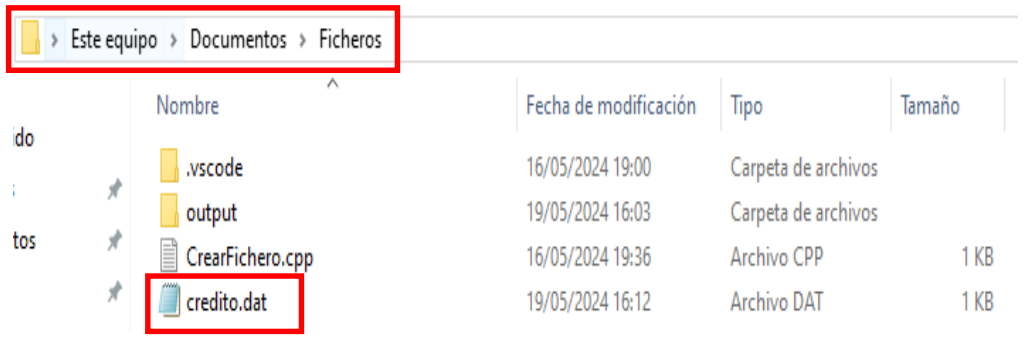
```c
#include <stdlib.h>
#include <stdio.h>

int main()
{
 int cuenta;
 char nombre[20];
 float saldo;
 FILE *ptrF;

 if((ptrF = fopen("C:\\Users\\Cesar Marin\\Documents\\Ficheros\\credito.dat", "w")) == NULL)
      printf("\nEl archivo no pudo ser abierto");
 else
  {
    printf("\nIntroduzca el número de cuenta, nombre y saldo (Ctrl+Z para terminar): ");
    printf("\n?: ");
    scanf("%d %s %f", &cuenta, nombre, &saldo);

    while(!feof(stdin))
    {
     fprintf(ptrF, "%d %s %.2f\n", cuenta, nombre, saldo);
     printf("?: ");
     scanf("%d %s %f", &cuenta, nombre, &saldo);
    }
    fclose(ptrF);
  }
 printf("\n");
 system("pause");
 printf("\n");
 return 0;
}
```

Ejemplos de Programas con Archivos de Texto

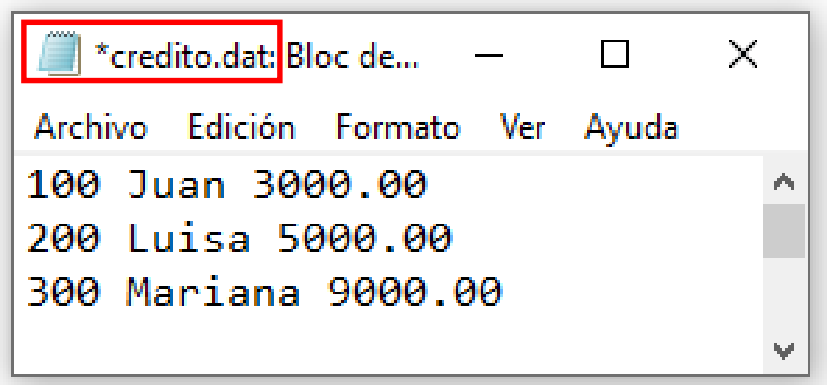
Salida del programa



Ruta de acceso al archivo



Contenido del archivo credito.dat



Ejemplos de Programas con Archivos de Texto

Programa en C: Lectura de un archivo secuencial de texto (almacenamiento persistente: disco duro → archivo → RAM). Recuperación de datos.

LeerFichero.cpp

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

int main()
{
 int cuenta;
 char nombre[20];
 float saldo;
 FILE *ptrF;

 if((ptrF = fopen("C:\\Users\\Cesar Marin\\Documents\\Ficheros\\credito.dat", "r")) == NULL)
     printf("\nEl archivo no pudo ser abierto");
 else
  {
    printf("%-10s %-13s %s\n", "Cuenta", "Nombre", "Saldo");
    fscanf(ptrF, "%d %s %f", &cuenta, nombre, &saldo);

    while(!feof(ptrF))
    {
     printf("%-10d %-13s %7.2f\n", cuenta, nombre, saldo);
     fscanf(ptrF, "%d %s %f", &cuenta, nombre, &saldo);
    }
    fclose(ptrF);
  }
 printf("\n");
 system("pause");
 printf("\n");
 return 0;
}
```

Ejemplos de Programas con Archivos de Texto

Salida del programa

```
C:\Users\Cesar Marin\Documents\LeerFichero>cd "c:\Users\Cesar Marin\Documents\LeerFichero\output"

c:\Users\Cesar Marin\Documents\LeerFichero\output>.\"LeerFichero.exe"
Cuenta     Nombre        Saldo
100        Juan          3000.00
200        Luisa         5000.00
300        Mariana       9000.00

Presione una tecla para continuar . . .
```

Bibliografía

- Joyanes, L. (4ta. Ed.). Fundamentos de Programación. Algoritmos, estructura de datos y objetos (pp. 127-148). Madrid: McGraw-Hill/Interamericana.

- Gottfried, B. (2da. Ed.) Programación en C (pp. 168-191) Madrid: McGraw-Hill/Interamericana.

UAM | FACULTAD INGENIERÍA & ARQUITECTURA