Archivos

Del Grecco Daiana Betania

UTN Facultad Regional Resistencia

Laboratorio de Computación I

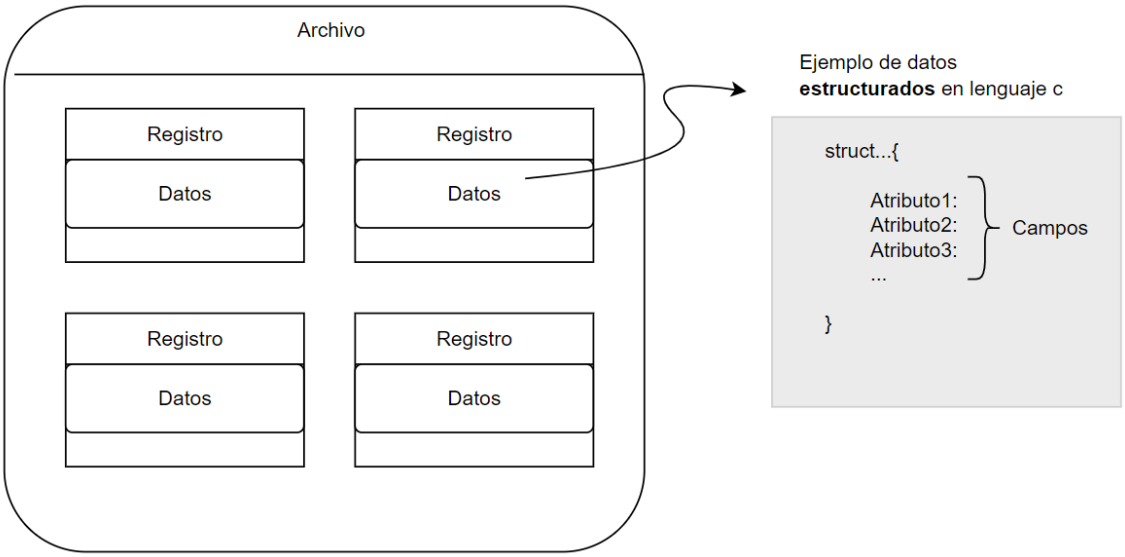
Ing. Carolina Vargas

14 Junio 2023

Índice

Archivos

En general, cuando hablamos de archivos en el contexto de la informática, nos referimos a los elementos que utilizamos para manipular documentos de texto, abrir y cerrar archivos con un simple clic, y llevar a cabo diversas acciones de manera dinámica y manual en una computadora. Sin embargo, si profundizamos en el concepto, un archivo de datos se refiere a un conjunto organizado de registros relacionados entre sí. Los registros contienen información (datos) que se guardan de manera estructurada, lo que significa que comparten una estructura de datos, un formato en común para luego almacenarse en un dispositivo externo, como un disco externo o una unidad de almacenamiento. Cada archivo tiene un nombre único y una extensión que indica su tipo de contenido.



Para la manipulación de archivos es necesario traer otro concepto, la declaración de Punteros.

Según Joyanes y Martínez (Año), "las variables punteros contienen valores que son direcciones de memoria donde se almacenan datos. En resumen, un puntero es una variable que contiene una dirección de memoria, y utilizando punteros su programa puede realizar muchas tareas que no sería posible utilizando tipos de datos estándar" (p. 323).

A la hora de realizar operaciones con archivos, se necesita principalmente los punteros para poder acceder a ellos y manipularlos. Éstos mismos habilitan a que en el lenguaje c, se puede utilizar todas las funciones de la librería <stdio.h> de manera eficiente. Principalmente ***fopen()*** para abrir un archivo.

El puntero se utiliza para almacenar la dirección de memoria donde se encuentra la estructura de datos que representa el archivo abierto, declararlo antes de abrir el archivo permite almacenar esa

dirección de memoria devuelta por la función ***fopen()*** y acceder a la información del archivo a través de ese puntero. Sin un puntero para almacenar esa referencia, no se podría interactuar con el archivo una vez que esté abierto.

Mezcla de archivos

Es el resultado de combinar archivos de datos (archivos de entrada) en un único archivo nuevo, denominado archivo de salida. Este proceso de combinación puede ser de dos maneras:

La *mezcla directa* se utiliza cuando todos los archivos de entrada tienen el mismo formato de registro y el número de registros en el archivo de salida es la suma de los registros de los archivos de entrada. Los registros se combinan directamente uno tras otro. Contiene un *ciclo incluyente* ya que se unen todos los registros de los archivos.

```
/*Ejemplo Mezcla directa
#include <stdio.h>+
int main() {
   FILE *archivo1, *archivo2, *resultado;
   int num1, num2;

   archivo1 = fopen("archivo1.txt", "r");
   archivo2 = fopen("archivo2.txt", "r");
   resultado = fopen("resultado.txt", "w");

   // Verificar si los archivos se abrieron correctamente
   if (archivo1 == NULL || archivo2 == NULL || resultado == NULL) {
      printf("Error al abrir los archivos.\n");
      return 1;
   }

   // Leer el primer número de cada archivo
   fscanf(archivo1, "%d", &num1);
   fscanf(archivo2, "%d", &num2);

   // Mezcla directa de archivos
   while (!feof(archivo1) && !feof(archivo2)) {
      if (num1 < num2) {
         fprintf(resultado, "%d\n", num1);
         fscanf(archivo1, "%d", &num1);
      } else {
         fprintf(resultado, "%d\n", num2);
         fscanf(archivo2, "%d", &num2);
      }
   }

   // Agregar los números restantes del archivo 1 (por si quedan números sin procesar en alguno de los archivos si uno de los archivos tiene más números que el otro).
   while (!feof(archivo1)) {
```

```
        fprintf(resultado, "%d\n", num1);
        fscanf(archivo1, "%d", &num1);
    }

    // Agregar los números restantes del archivo 2 (por si quedan números sin procesar en alguno de los archivos si uno de los archivos tiene más números que el otro).

    while (!feof(archivo2)) {
        fprintf(resultado, "%d\n", num2);
        fscanf(archivo2, "%d", &num2);
    }

    // Cerrar los archivos
    fclose(archivo1);
    fclose(archivo2);
    fclose(resultado);

    printf("Mezcla directa de archivos completada con éxito.\n");

    return 0;
}
*/
```

La *mezcla indirecta* se utiliza cuando hay varios archivos de entrada con diferentes formatos de registro. En este caso, uno de los archivos se considera de mayor prioridad y se utiliza para guiar el proceso de combinación. El archivo de salida adopta el formato del archivo de mayor prioridad o una mezcla de los formatos de los archivos de entrada, sin que se pueda establecer una relación numérica precisa entre los elementos de los registros. Contiene un *ciclo excluyente* ya que se unen algunos los registros de los archivos.

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

        int main() {
            FILE *archivo1, *archivo2, *resultado;
            int num1, num2;

            archivo1 = fopen("archivo1.txt", "r");
            archivo2 = fopen("archivo2.txt", "r");
            resultado = fopen("resultado.txt", "w");

            // Verificar si los archivos se abrieron correctamente
            if (archivo1 == NULL || archivo2 == NULL || resultado == NULL) {
                printf("Error al abrir los archivos.\n");
                return 1;
            }
        // Leer el primer número de cada archivo
            fscanf(archivo1, "%d", &num1);
            fscanf(archivo2, "%d", &num2);
```

```
    // Mezcla indirecta de archivos excluyendo números pares
    while (!feof(archivo1) && !feof(archivo2)) {
        if (num1 % 2 == 0) {
            fscanf(archivo1, "%d", &num1);
            continue;  // Salta al siguiente ciclo sin escribir el número en el archivo de salida
        }
        if (num2 % 2 == 0) {
            fscanf(archivo2, "%d", &num2);
            continue;  // Salta al siguiente ciclo sin escribir el número en el archivo de salida
        }

        if (num1 < num2) {
            fprintf(resultado, "%d\n", num1);
            fscanf(archivo1, "%d", &num1);
        } else {
            fprintf(resultado, "%d\n", num2);
            fscanf(archivo2, "%d", &num2);
        }
    }

    // Agregar los números restantes del archivo 1 (por si quedan números sin procesar en alguno de los archivos si uno de los archivos tiene más números que el otro).

    while (!feof(archivo1)) {
        if (num1 % 2 != 0) {
            fprintf(resultado, "%d\n", num1);
        }
        fscanf(archivo1, "%d", &num1);
    }

    // Agregar los números restantes del archivo 2  (por si quedan números sin procesar en alguno de los archivos si uno de los archivos tiene más números que el otro).

    while (!feof(archivo2)) {
        if (num2 % 2 != 0) {
            fprintf(resultado, "%d\n", num2);
        }
        fscanf(archivo2, "%d", &num2);
    }

    // Cerrar los archivos
    fclose(archivo1);
    fclose(archivo2);
    fclose(resultado);

    printf("Mezcla indirecta de archivos completada con éxito (excluyendo números pares).\n");
    return 0;
}
*/
```

Apareo de archivos

Es un proceso en el cual se combinan dos o más archivos relacionados, usando un campo en común o clave de vinculación. Y se genera un nuevo archivo de datos combinados. Se utiliza en situaciones donde se tienen conjuntos de datos separados con información relacionada y se desea agruparlos a partir de ese mismo valor o clave.

Ahora, ¿cuál es la diferencia entre mezcla y apareo? La principal diferencia radica en que el apareo de archivos implica la combinación de registros según un filtro o condición definida, lo que a menudo resulta en un ciclo excluyente. Esto significa que solo se requiere la unión de algunos registros específicos, en contraste con la mezcla que combina todos los registros sin excepción.

```
/*Ejemplo de apareo
#include <stdio.h>
int main() {
   FILE *archivo1, *archivo2, *resultado;
   int num1, num2;

   archivo1 = fopen("archivo1.txt", "r");
   archivo2 = fopen("archivo2.txt", "r");
   resultado = fopen("resultado.txt", "w");

   // Verificar si los archivos se abrieron correctamente
   if (archivo1 == NULL || archivo2 == NULL || resultado == NULL) {
      printf("Error al abrir los archivos.\n");
      return 1;
   }

   // Leer el primer número de cada archivo
   fscanf(archivo1, "%d", &num1);
   fscanf(archivo2, "%d", &num2);

   // Apareo de archivos
   while (!feof(archivo1) && !feof(archivo2)) {
      if (num1 < num2) {
         fprintf(resultado, "%d\n", num1);
         fscanf(archivo1, "%d", &num1);
      } else {
         fprintf(resultado, "%d\n", num2);
         fscanf(archivo2, "%d", &num2);
      }
   }

   // Agregar los números restantes del archivo 1 (si hay)
   while (!feof(archivo1)) {
      fprintf(resultado, "%d\n", num1);
      fscanf(archivo1, "%d", &num1);
   }
```

```
    // Agregar los números restantes del archivo 2 (si hay)
    while (!feof(archivo2)) {
        fprintf(resultado, "%d\n", num2);
        fscanf(archivo2, "%d", &num2);
    }

    // Cerrar los archivos
    fclose(archivo1);
    fclose(archivo2);
    fclose(resultado);

    printf("Apareo de archivos completado con éxito.\n");

    return 0;
}
*/
```

Corte de control

Es un proceso que se realiza como parte del control de archivos y sus datos dentro. Es una práctica que ayuda a garantizar la integridad y la fiabilidad de los datos almacenados, permitiendo identificar y corregir posibles errores o inconsistencias que podrían afectar la precisión y la confiabilidad de la información. Utilizando registros que representan entidades, se compara los datos esperados con el resultado real para analizar mejorías en esas estadísticas.

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

    int main() {
        FILE *archivo;
        int valorEsperado = 1000;
        int total = 0;
        int valorObtenido;

        archivo = fopen("datos.txt", "r");
        if (archivo == NULL) {
            printf("Error al abrir el archivo.\n");
            return 1;
        }
        while (fscanf(archivo, "%d", &valorObtenido) == 1) {
            total += valorObtenido;
        }
        fclose(archivo);

        if (total == valorEsperado) {
            printf("Corte de control exitoso. Los datos son consistentes.\n");
        } else {
            printf("Error en el corte de control. Los datos no son consistentes.\n");
        }

        return 0;}*/
```

En resumen, la mezcla, apareo y corte de control en archivos puede ser realizada manualmente, pero generalmente se lleva a cabo utilizando software o herramientas especializadas que facilitan el proceso y automatizan las tareas relacionadas.

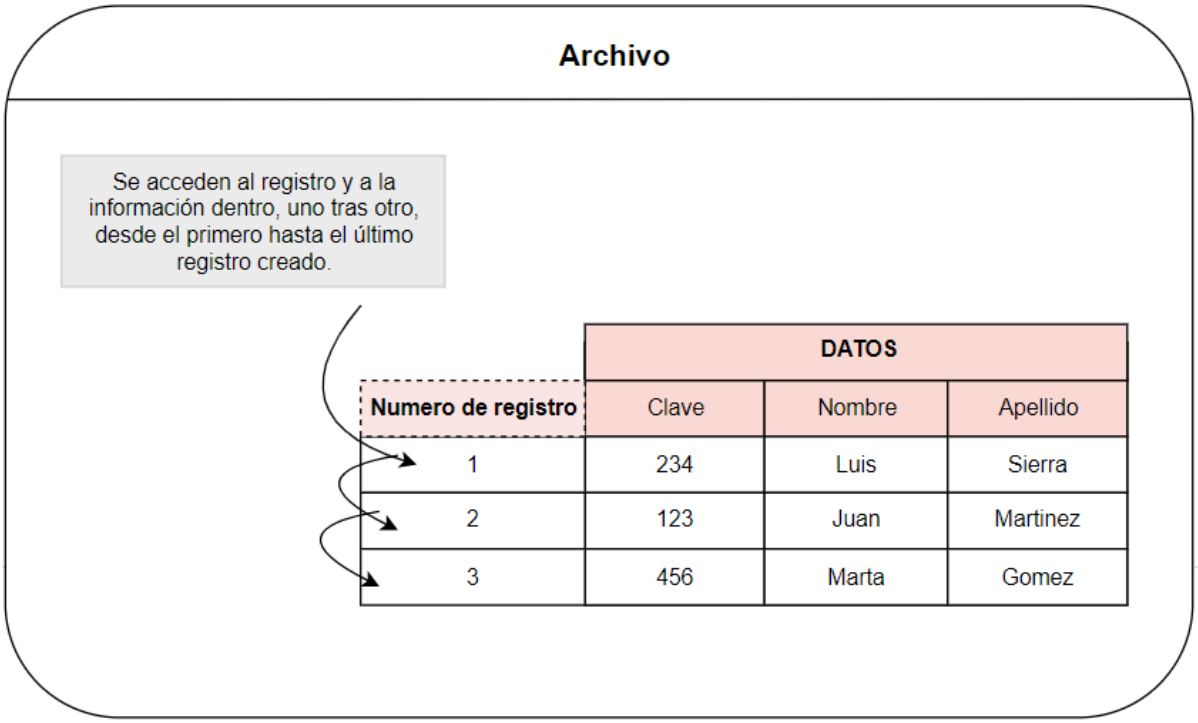
Diferencia entre arreglo y archivo

Aunque los arreglos y los archivos comparten algunas similitudes, como la capacidad de almacenar y manipular datos, es importante destacar sus diferencias fundamentales.

Conceptos	Arreglo	Archivo
Definición	estructura de datos que almacena una colección de elementos del mismo tipo	un conjunto de registros organizados y relacionados entre sí, que contiene datos estructurados
Elementos que contiene	números, caracteres, booleanos, etc.	texto, imágenes, sonido y otros tipos de formatos.
Almacenamiento	almacenan datos en memoria de forma contigua	en un dispositivo de almacenamiento secundario, como un disco duro o una memoria flash.
Método de acceso	se acceden mediante un índice numérico que indica la posición del elemento dentro del arreglo.	mediante operaciones de lectura y escritura en disco (Acceso Secuencial o directo)
Funcionalidad	se utilizan ampliamente en programación para implementar estructuras de datos como listas, pilas, colas, matrices multidimensionales, entre otros.	se utilizan en aplicaciones que requieren almacenamiento duradero de datos, como bases de datos, sistemas de archivos, registros de eventos, archivos de configuración, entre otros.

Métodos de acceso de Archivos

Uno de los métodos es el *acceso secuencial*, los datos se leen o escriben en orden secuencial, desde el principio hasta el final del archivo. Para llegar a un registro, se recorren sus precedentes en el mismo orden en el que situaron.



El acceso secuencial es útil cuando no conoces de antemano la ubicación del registro que necesitas, ya que recorres los registros en orden hasta encontrarlo. Sin embargo, si el archivo es grande y el registro deseado se encuentra hacia el final, el acceso secuencial puede ser menos eficiente en términos de tiempo de acceso.

Ejemplo de Acceso Secuencial en lenguaje C :

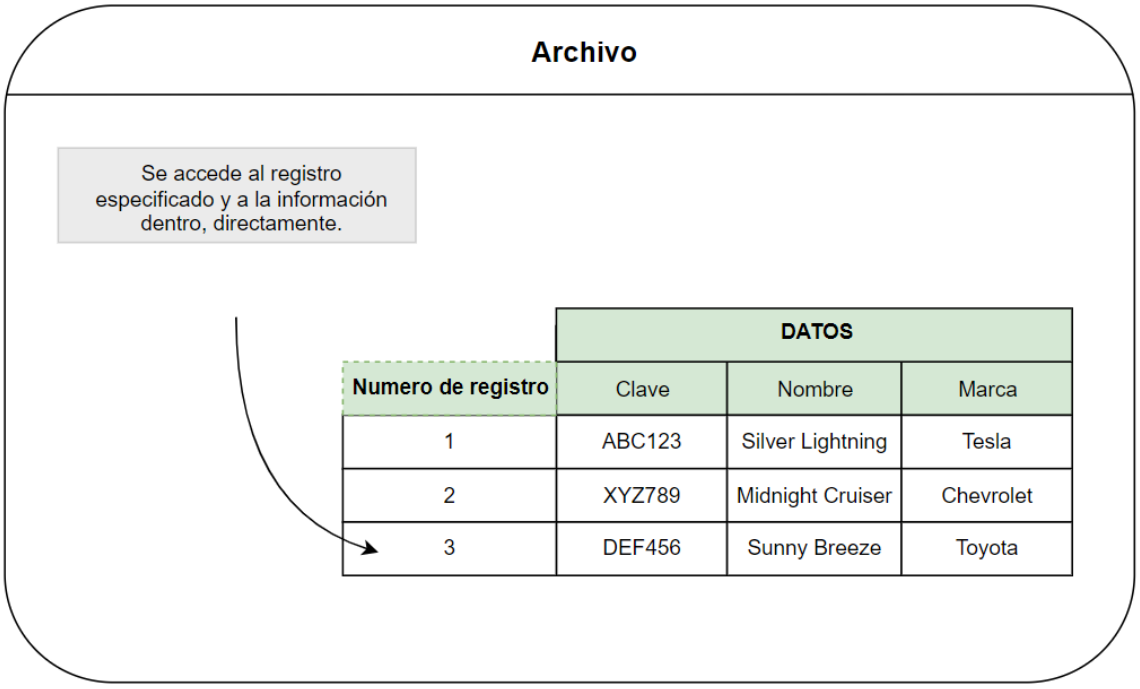
```
/*#include <stdio.h>

int main() {
   FILE *archivo;
   char linea[100];
   archivo = fopen("datos.txt", "r");
   if (archivo == NULL) {
      printf("Error al abrir el archivo.\n");
      return 1;
   }
   while (fgets(linea, sizeof(linea), archivo) != NULL) {
```

```
        printf("%s", linea);
    }
    fclose(archivo);

    return 0;
}*/
```

Por otro lado, el *acceso directo* consiste en acceder a un registro en particular, se utiliza una técnica de indexación o una estructura adicional que mapea la posición lógica del registro a su ubicación física en el archivo. Este tipo de acceso es útil cuando se necesita acceder rápidamente a registros específicos sin tener que leer o procesar todos los registros anteriores.



Ejemplo de Acceso Directo en lenguaje C :

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

typedef struct {
   int id;
   char nombre[50];
   float salario;
} Empleado;

int main() {
   FILE *archivo;
   Empleado empleado;

   archivo = fopen("empleados.dat", "rb");
   if (archivo == NULL) {
      printf("Error al abrir el archivo.\n");
      return 1;
   }


   int registro = 2;  // Acceder al tercer registro (índice 2)
   fseek(archivo, registro * sizeof(Empleado), SEEK_SET);
   fread(&empleado, sizeof(Empleado), 1, archivo);

   printf("ID: %d\n", empleado.id);
   printf("Nombre: %s\n", empleado.nombre);
   printf("Salario: %.2f\n", empleado.salario);

   fclose(archivo);

   return 0;
}
*/
```

Tipo de Contenido

Archivo de texto

cuando hablamos de archivos de texto, nos referimos a los archivos que contienen datos almacenados en formato de texto legible por humanos. Estos archivos suelen estar compuestos por caracteres alfanuméricos y símbolos comunes. Por ejemplo, los archivos .csv (valores separados por comas), .xml (lenguaje de marcado extensible), .json (notación de objetos de JavaScript), .html (lenguaje de marcado de hipertexto), entre otros, pueden contener datos en formato de texto.

Archivo binario

los archivos binarios almacenan información en una representación codificada en forma de secuencias de bits. lo que es útil para trabajar con datos estructurados y no legibles directamente para los humanos.

Algunos ejemplos comunes de archivos binarios son los archivos de imágenes (como .jpg, .png, .gif), archivos de audio (como .mp3, .wav), archivos de video (como .mp4, .avi), archivos de bases de datos (como .db, .mdb), archivos ejecutables (como .exe, .bin), entre otros.

¿Qué diferencia hay entre un archivo de texto y un archivo binario?

Archivo	Texto	Binario
Representación	los datos se almacenan en forma de caracteres legibles. Texto plano.	los datos se almacenan en su representación binaria, lo cual no resulta ser de fácil legibilidad
Tamaño	ocupan más espacio, ya que caracteres de control y saltos de línea.	ocupan menos espacio, los datos se guardan como secuencias de bits, almacenan los datos en un formato más directo, sin la necesidad de codificar los caracteres en una representación textual
Interpretación	los archivos de texto se pueden abrir y leer con un editor de texto simple, lo que permite una fácil interpretación de los datos.	los archivos binarios generalmente requieren programas específicos o algoritmos personalizados para interpretar y procesar los datos contenidos en ellos.

Apertura y Cierre de Archivos

¿Por qué no se puede abrir un archivo?

Hay varios elementos a tener en cuenta a la hora de abrir un archivo y a su vez para verificación que se pueda trabajar correctamente con el mismo.

1. Utilizar la librería que contiene las funciones para trabajar con el archivo deseado.
2. Declarar un putero de Archivo, para poder acceder a él y abrirlo.
3. Principalmente utilizar la función fopen().

fopen(ruta/nombre de Archivo, método de apertura)

¿Por qué es necesario cerrar los archivos al final del proceso?

Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa.

```
/*
Ejemplo
Int main (){
        FILE *parch;
        if((parch=fopen("c:\\banco.dat","rb"))==NULL)//Se abre en modo lectura
        printf("\nEl archivo no puede ser abierto");
         if((fclose(parch))= = -1) //Se cierra el archivo
                printf("\nNo se pudo cerrar el archivo");
        else
                printf("\nEl archivo se cerro exitosamente");
return 0;
*/
```

Parte práctica: Ejercicios de Archivos

Ejercicio N°3: Indique en C las instrucciones para realizar las siguientes operaciones

1. **¿Qué significa este tipo de apertura?**

- r: ……
- w: ……
- a: ……
- r+:……
- w+: ……
- a+: ……
- rb: ……
- wb: ……
- ab: ……
- r+b: ……
- w+b: ……
- a+b: ……

r	Lectura
w	Escritura
a	Adjuntar , agrega
r+	Lectura y escritura(no sobrescribe contenido)
w+	Lectura y escritura(sobrescribe contenido)
a+	Adjuntar y lectura
rb	Lectua binaria
wb	Escritura binaria
ab	Adjuntar binario
r+b	Lectura y escritura binaria
w+b	Lectura y escritura binaria
a+b	Adjuntar y lectura bianria

2. **¿Qué significa lo siguiente?**
   a. t: modo texto. Normalmente es el modo por defecto. Se suele omitir.
   b. b: modo binario.

Respuesta:

Son especificaciones para el tipo de acceso del archivo.

a. Se utiliza para indicar que el archivo se abre en modo texto, lo cual significa que se interpretan como un archivo de texto legible para los humanos.
b. el archivo se abre en modo binario, lo cual significa que se acceden a los datos en su representación binaria, sin ninguna interpretación específica de caracteres. En este modo, se pueden leer y escribir datos en su forma binaria.

3. **Indicar para que se usan las siguientes funciones**

Función	Significado
fopen()	Abre un archivo
fclose()	Cierra un archivo
fgets()	Guarda en un array de caracteres
fputs()	Escribe cadena de caracteres en archivo
fseek()	Mueve el puntero a una posición especifica dentro del archivo
fprintf()	Escribe datos en el archivo
fscanf()	Lee y guarda datos del archivo
feof()	Devuelve un valor distinto de cero si se alcanza el final del archivo
ferror()	Devuelve un valor distinto de cero si se ha producido un error durante una operación de archivo
rewind()	Mueve el puntero de archivo al principio del archivo
remove()	Elimina un archivo del sistema de archivos. El archivo especificado se borra permanentemente y no se puede recuperar
fflush()	Limpia el búfer del archivo y asegura que todos los datos pendientes de escritura se escriban en el archivo.

4. **¿Qué significa "FILE *nombrearchivo"?**

   Respuesta: El asterisco (*) indica que se trata de un puntero, una variable que almacena la dirección de memoria de otro objeto. En este caso, el puntero "nombrearchivo" apuntará a la estructura FILE que representa el archivo.

5. **¿Qué significa el siguiente trozo de programa? ¿Qué tipo de archivo se está abriendo?**

```
FILE *fich;
if ((fich = fopen("nomfich.dat", "r")) == NULL) printf ( " Error en la apertura. Es posible que el fichero no exista \n ");
```

Respuesta: Luego de abrir un archivo en modo lectura, se establece una condición para que se muestre un mensaje por pantalla que indique el archivo no pudo abrirse correctamente. ***(" Error en la apertura. Es posible que el fichero no exista \n ")***

6. **¿Qué tipo de apertura tienen los siguientes archivos?**

```
FILE * datosdatos = fopen ("nombres.dat","r");
datos = fopen ("nombres.dat", "w");
datos = fopen ("nombres.dat", "a");
datos = fopen ("nombres.dat", "ra");
```

   Respuesta:

   1. Lectura *("r");*
   2. Escritura *("w");*
   3. Adjuntar *("a");*
   4. Lectura y Adjuntar *("ra");*

7. **¿Qué representa el siguiente trozo de programa?**

```
FILE *parch;
if((parch=fopen("c:\\banco.dat","rb"))==NULL)//Se abre en modo lectura
        printf("\nEl archivo no puede ser abierto");
if((fclose(parch))= = -1) //Se cierra el archivo
        printf("\nNo se pudo cerrar el archivo");
        else
       printf("\nEl archivo se cerro exitosamente");
```

Respuesta:

1. Se declara un puntero a FILE llamado "parch" utilizando la sintaxis "FILE *parch;"
2. Se abre el archivo "c:\banco.dat" en modo de lectura binaria utilizando la función fopen()
3. "(parch=fopen("c:\banco.dat","rb"))==NULL" verifica si el archivo se abrió correctamente. Si el archivo no se puede abrir, se imprime el mensaje "El archivo no puede ser abierto".
4. Luego hay ua condición if que verifica si el archivo se cerró correctamente "(fclose(parch)) == -1" . Si no se pudo cerrar el archivo, se imprime el mensaje "No se pudo cerrar el archivo". De lo contrario, se imprime el mensaje "El archivo se cerró exitosamente".

Referencias

Joyanes Aguilar, L., & Martínez Torres, I. (2005). PROGRAMACIÓN EN C: Metodología, algoritmos y estructura de datos.

Barcelona Geeks. (s.f.). Métodos de acceso a archivos en el sistema operativo. Recuperado de https://barcelonageeks.com/metodos-de-acceso-a-archivos-en-el-sistema-operativo/

Haras Da Dincó. (s.f.). ¿Cuál es la diferencia entre acceso directo y secuencial? Recuperado de https://www.harasdadinco.cl/cual-es-la-diferencia-entre-acceso-directo-y-secuencial/

López, J. A. (s.f.). Métodos de acceso a archivos. Recuperado de https://webs.um.es/jal/43-edtextos.xml

Becerra, A. (s.f.). Archivos. Recuperado de https://w3.ual.es/~abecerra/ID/archivos

Canal de YouTube de OpenWebinars. (2017, 6 de octubre). Programación en C++ #4 - Archivos: Acceso secuencial y directo [Video]. Recuperado de https://youtu.be/4IabpaIObzMhttps://sqlesba.blogspot.com/2011/09/113-archivos.html