ARCHIVOS
(File Management)

Archivos

- Un archivo es una colección de bytes almacenados en un dispositivo.
  Un archivo contiene datos, el sistema que lo alberga contiene metadatos del mismo

- En un programa en C es muy importante utilizar archivos para almacenar información de manera persistente.

- Las operaciones básicas sobre archivos son
  - Nombrar
  - Abrir
  - Leer
  - Escribir
  - Cerrar

- El nombre de un archivo está compuesto por una cadena de caracteres, típicamente nombre.extension

Estructura FILE

- FILE es un tipo de dato definido en <stdio.h>

```
typedef struct _IO_FILE FILE;
```

- Es una estructura de datos que contiene toda la información sobre un archivo
  Esta estructura es dependiente de la implementación, es decir puede diferir entre sistema y sistema

- Hace las veces de “interfaz” con el archivo real almacenado en el sistema

- Se crea un puntero a FILE (FILE*) para realizar las operaciones sobre el archivo

- **FILE (/usr/include/libio.h)**

```
struct _IO_FILE {
  int _flags;           /* High-order word is _IO_MAGIC; rest is flags. */

  /* The following pointers correspond to the C++ streambuf protocol. */
  /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
  char* _IO_read_ptr;    /* Current read pointer */
  char* _IO_read_end;    /* End of get area. */
  char* _IO_read_base;   /* Start of putback+get area. */
  char* _IO_write_base;  /* Start of put area. */
  char* _IO_write_ptr;   /* Current put pointer. */
  char* _IO_write_end;   /* End of put area. */
  char* _IO_buf_base;    /* Start of reserve area. */
  char* _IO_buf_end;     /* End of reserve area. */

  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base;   /* Pointer to start of non-current get area. */
  char *_IO_backup_base; /* Pointer to first valid character of backup area */
  char *_IO_save_end;    /* Pointer to end of non-current get area. */
  struct _IO_marker *_markers;
  struct _IO_FILE *_chain;
  int _fileno;
  . . .
};
```

- **Apertura de archivo**

  - Mediante **fopen** de **<stdio.h>**

    `FILE* fopen(const char* nombre_archivo, const char* modo)`

    Argumentos:

    `nombre_archivo`: cadena que contiene el nombfre del archivo

    `modo`: cadena que especifica modo de apertura

    Retorno: puntero a FILE señalando al archivo accedido o bien
    NULL si el archivo no puede accederse

Modo	Significado
r	Abre un fichero de texto para lectura. El fichero debe existir.
w	Crea un fichero de texto para escritura. Si el fichero existe, la información en él contenida se destruye.
a	Abre un fichero de texto para añadir nuevos datos. Si el fichero no existe, se crea.
rb	Abre un fichero binario para lectura. El fichero debe existir.
wb	Crea un fichero binario para escritura. Si el fichero existe, la información en él contenida se destruye.
ab	Abre un fichero binario para añadir nuevos datos. Si el fichero no existe, se crea.
r+	Abre un fichero de texto para lectura/escritura
w+	Crea un fichero de texto para escritura/lectura. Si el fichero existe se destruye.
a+	Abre un fichero de texto para lectura/escritura. Si el fichero no existe, se crea.
rb+	Abre un fichero binario para lectura/escritura.
rw+	Crea un fichero binario para lectura/escritura.
ra+	Abre un fichero binario para lectura/escritura.

Cierre de archivo

- Mediante **fclose** de **<stdio.h>**

```
int fclose(FILE *nombre_archivo)
```

Argumentos:

nombre_archivo: puntero a FILE correspondiente al archivo a cerrar

Retorno: 0 en caso correcto, o bien
EOF (End Of File) en presencia de errores

- EOF está definido en stdio.h y vale -1. EOF es un carácter especial de la tabla ASCII (valor 26) y sigue al último registro almacenado en el archivo.
- Para archivos binarios EOF no es útil para detectar el final del archivo, pues puede aparecer este valor como parte del contenido del archivo

- La siguiente función de stdio.h

```
int feof(FILE *nombre_archivo)
```

Permite detectar el final de un archivo de texto o binario, retornando =! 0 si se alcanzó el final del archivo y 0 en caso contrario

Acceso secuencial a nivel caracter

- `int fputc(int c, FILE *nombre_archivo)`

  Argumentos:

  `c`: caracter a escribir en el archivo

  `nombre_archivo`: puntero a FILE correspondiente al archivo a editar

  Retorno: `c` en caso de éxito

  EOF (End Of File) en caso contrario

- `int fgetc(FILE *nombre_archivo)`

  Argumentos:

  `nombre_archivo`: puntero a FILE* del archivo donde leer los caracteres

  Retorno: EOF (End Of File) al alcanzar el fin del archivo

Existen dos funciones equivalentes a `fputc` y `fgetc`, llamadas `putc` y `getc`, las cuales pueden ser implementadas como macros y pueden evaluar el `nombre_archivo` más de una vez.

- **E/S con buffer**

  - **`int fputs(const char* cadena, FILE *nombre_archivo)`**

    Argumentos:

    `cadena`: cadena a escribir en el archivo

    `nombre_archivo`: puntero a FILE correspondiente al archivo a editar

    Retorno: último carácter escrito en caso de éxito
    EOF (End Of File) en caso contrario

En la man page dice:

`fputs() writes the string s to stream, without its trailing '\0'`

Lo que significa es que el carácter de fin de cadena '\0' se convierte al carácter LF (salto de línea) al ser escrita la cadena en el archivo.
Esta conversión sucede si el modo de apertura es w (texto) y no wb (binario).

- **E/S con buffer**

  - `char* fgets(char* cadena, int longitud, FILE *nombre_archivo)`

    Argumentos:

    `cadena`: cadena que hace las veces de buffer

    `longitud`: indica la cantidad de caracteres a ser leidos

    `nombre_archivo`: puntero a FILE correspondiente al archivo a leer

    Retorno: en caso de éxito retorna un puntero a **`cadena`**

    en caso de error o alcanzar EOF, retorna NULL

Lee una cadena de caracteres hasta leer LF(salto línea) o hasta leer **`longitud`**-1 caracteres del archivo, almacenándolos en **`cadena`**, y agregando el '\0'.

- **E/S de bloques de datos**

```
size_t fread(void* buffer, int nbytes, int n, FILE *archivo)
size_t fwrite(const void* buffer, int nbytes, int n, FILE *archivo)
```

Argumentos:

buffer: puntero a un región de memoria para la lectura/escritura de datos
nbytes: cantidad de bytes a leer/escribir (tamaño del bloque)
n: indica cuantos bloques de datos se van a leer/escribir
archivo: puntero a FILE correspondiente al archivo
Retorno: en caso de éxito retorna la cantidad de bloques leidos/escritos
En caso de error o alcanzarse EOF retorna 0

Estas funciones se emplean típicamente para leer/escribir datos estructurados como ser estructuras de datos o arrays.

fread no distingue entre EOF y error, el usuario debe utilizar **feof** o **ferror** para determinar lo ocurrido. Vea ferror(FILE*) y clearerr(FILE*)

- **Acceso directo**

  - `int fseek(FILE *archivo, long offset, int origen)`

Argumentos:

`archivo`: puntero a FILE correspondiente al archivo

`offset`: cantidad de bytes a desplazarse a partir de **`origen`** en **`archivo`**

`origen`: macro que indica desde donde se hará el desplazamiento (offset)

Valores posibles:

`SEEK_SET` (valor 0): Principio de archivo

`SEEK_CUR` (valor 1): Posición actual

`SEEK_END` (valor 2): Final del archivo

Retorno: en caso de éxito retorna 0

en caso de error != 0

Esta función debe utilizarse con archivos binarios, pues con archivos de texto puede generar problemas debido a conversiones de caracteres.

Acceso directo

- `int ftell(FILE *archivo)`

  Argumentos:

  archivo: puntero a FILE correspondiente al archivo

  Retorno: retorna el valor actual del indicador de posición en archivo siendo el valor (en bytes) desde el comienzo del archivo a la posición actual

  En caso de error, retorna -1y se setea errno

Existe una función `void rewind(FILE *archivo)` que sirve para establecer el indicador de posición del archivo al principio y la cual está relacionada con **fseek** de la siguiente forma

```
(void) fseek(archivo, 0L, SEEK_SET)
```

También existen las funciones

```
int fgetpos(FILE *archivo, fpos_t *pos);
int fsetpos(FILE *archivo, fpos_t *pos);
```

que son front-ends de **ftell** y **fseek** para establecer el origen con **pos**.

- **Ejemplos: files1.c**

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

int main () {
  FILE *f;
  char nombre_archivo[20];

  printf("Ingrese el nombre del archivo: ");
  scanf("%s", nombre_archivo);

  if ((f=fopen(nombre_archivo, "r")) == NULL) {
    printf("errno = %d\n", errno);
    perror("main");
  } else {
    printf("El archivo existe\n");
    fclose(f);
  }

  return 0;
}
```

- **Ejemplos: files2.c**

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

int main () {
  FILE *f;
  char c;

  printf("Ingrese un texto. Ctrl+d para terminar\n");

  if ((f=fopen("salida.out", "w")) != NULL) {
    while ((c=getchar()) != EOF)
      fputc(c, f);
    fclose(f);
  } else {
    printf("errno = %d\n", errno);
    perror("main");
  }

  return 0;
}
```

- **Ejemplos: files3.c**

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

int main () {
  FILE *f;
  char c, nombre_archivo[20];

  printf("Ingrese el nombre del archivo: ");
  scanf("%s", nombre_archivo);

  if ((f=fopen(nombre_archivo, "r")) == NULL) {
    printf("errno = %d\n", errno);
    perror("main");
  } else {
    while ((c = fgetc(f)) != EOF)
       fputc(c, stdout);

    printf("Fin del archivo %s.\n", nombre_archivo);
    fclose(f);
  }
  return 0;
}
```

- **Ejemplos: files4.c**

```
FILE *f1, *f2;

if ((f1=fopen("origen", "r")) == NULL) {
  printf("errno = %d\n", errno);
  perror("main");
  return -1;
}

if ((f2=fopen("destino", "w")) == NULL) {
  printf("errno = %d\n", errno);
  perror("main");
  return -1;
}

while ((c = fgetc(f1)) != EOF)
     fputc(c, f2);

fclose(f1);
fclose(f2);
```

- **Ejemplos: files5.c**

```
char nombre[25];
FILE *f1, *f2;

printf("Ingrese el nombre de un archivo de texto: ");
scanf("%s", nombre);
if ((f1=fopen(nombre, "a")) == NULL) {
  printf("errno = %d\n", errno); perror("main");
  return -1;
}

printf("Ingrese el nombre de otro archivo de texto: ");
scanf("%s", nombre);
if ((f2=fopen(nombre, "r")) == NULL) {
  printf("errno = %d\n", errno); perror("main");
  return -1;
}

while ((c = fgetc(f2)) != EOF)
     fputc(c, f1);

fclose(f1); fclose(f2);
```

- **Ejemplos: files6.c**

```
int main () {
 FILE *f;
 int entero, entero2 = 100;
 void *buffer=(int*)malloc(sizeof(int));

 f = fopen("salida.out", "r+b");  //apertura en modo r/w binario
 printf("Ingrese un entero (distinto de 100): ");
 scanf("%d", &entero);

 buffer = &entero; //se guarda en buffer lo que ingresa el usuario
 fwrite(buffer, sizeof(int), 1, f);

 buffer = &entero2; //se "ensucia" la variable deliberadamente

 rewind(f); //se rebobina para leer desde el ppio del archivo
 fread(buffer, sizeof(int), 1, f);
 printf("El entero guardado fue %d\n", *(int*)buffer);

 fclose(f);
```

- Ejemplos: files7.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define N 5

int main () {
  FILE *f;
  unsigned short int i;
  void *buffer = (int*)malloc(N * sizeof(int));

  if ((f=fopen("enteros.out", "w+b")) != NULL) {  //modo r/w binario
    for (i=0; i<N; i++) {
      printf("Ingrese entero en posicion %u: ", i);
      scanf("%d", &((int*)buffer)[i]);
    }
    fwrite(buffer, sizeof(int), N, f); //se escriben los N enteros

    do {
      printf("Ingrese posicion en el archivo a recuperar: ");
      scanf("%hu", &i);
    } while (i < 0 || i >= N);

    fseek(f, i*sizeof(int), SEEK_SET);
    fread(buffer, sizeof(int), 1, f);
    printf("El entero en la posicion %hu es %d\n", i, *(int*)buffer);
    fclose(f);
```

- **Quiz**

1. Qué objeto utilizamos para representar un archivo en C?

A. FILE*
B. fopen
C. printf
D. fprintf

2. Antes de leer/escribir un archivo en C, qué debe hacerse?

A. Invocar a fopen sobre el archivo
B. Crear el archivo
C. Asignar memoria dinámicamente a FILE*
D. Usar fprintf

3. Cómo se puede escribir una string en un archivo de texto?

A. Abrir el archivo y utilizar la función fprintf
B. Abrir el archivo y usar printf, la salida se direcciona al archivo en lugar de stdout
C. Abrir el archivo y usar fputc de manera reiterada
D. Abrir el archivo y utilizar la funcion fputs

- **Quiz**

4. Qué flag se utiliza con fopen para agregar contenido a un archivo en lugar de sobrescribirlo?

A. a
B. w++
C. w
D. W+

5. Cómo se abre un archivo binario?

A. Usar "b" como flag con fopen
B. Los archivos se abren en modo binario por defecto
C. El archivo se tener la extensión .bin
D. Se utiliza el tipo BINARYFILE*