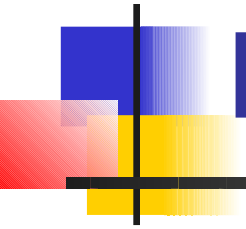
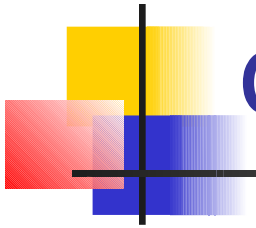


Entrada y salida en C

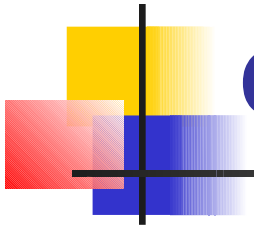
Informática II - 2012

Argumentos en la línea de comando



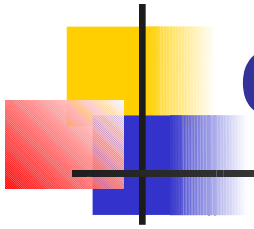
- Dentro de un entorno que maneje C hay forma de pasar argumentos en la línea de comando o de parámetros cuando se ejecuta un programa
- Cuando se invoca a main, se lo hace con 2 argumentos:
- El 1º (llamado por convención argc, por argument count) es el número de argumentos en la línea de comandos con los que se invocó el programa
- El 2º (argv, por argument vector) es un puntero a un arreglo de cadena de caracteres que contiene los argumentos, uno por cadena

Argumentos en la línea de comando



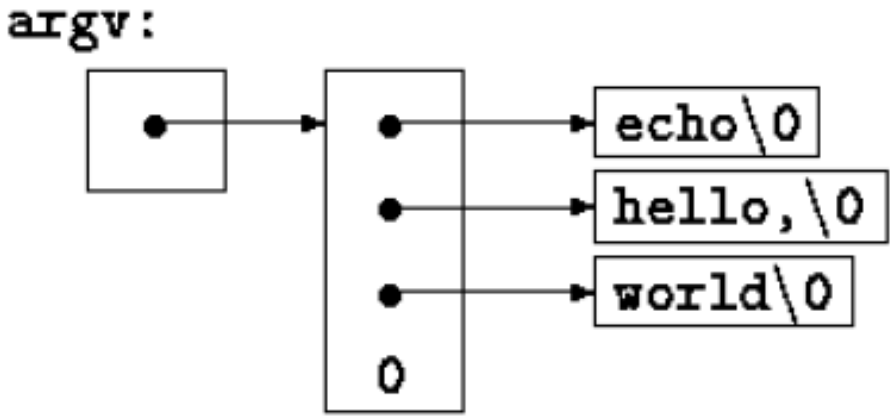
- El ejemplo más sencillo es el programa echo que despliega sus argumentos de la línea de comando en una línea, separados por blancos.
- Por convención argv[0] es el nombre con el que se invocó al programa (nombre del archivo donde está main), por lo que argc es por lo menos 1
- Si argc es 1, entonces no hay argumentos en la línea de comando después del nombre del programa

Argumentos en la línea de comando



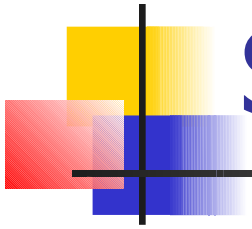
- En el ejemplo: echo hola, mundo
- argc=3, argv[0]="echo", argv[1]="hola," y argv[1]="mundo"
- El primer argumento optativo es argv[1] y el último es argv[argc-1]
- Además el estándar requiere que argv[argc] sea un puntero nulo

Argumentos en la línea de comando



Argumentos en la línea de comando

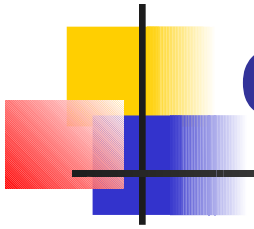
```
#include <stdio.h>

int main(int argc, char *argv[])
{
int i;
for (i = 1; i < argc; i++)
printf("%s%s", argv[i], (i < argc-1) ? " " :"");
printf("\n");
return 0;
}
```



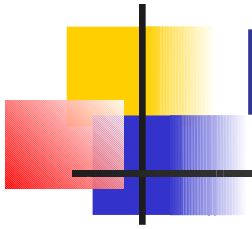
Salida por pantalla

```
hola, mundo
Presione una tecla para continuar . . .
```

Argumentos en la línea de comando



- En Visual C++ se añaden los comandos en Proyecto->Propiedades->Depuración->argumentos de comandos (añadirlos antes de ejecutar)
- Para convertir cadenas en enteros o flotantes uso atoi o atof, funciones de stdlib



Ejemplo

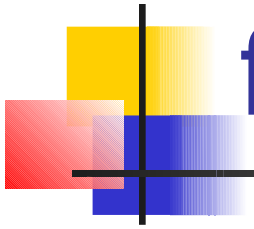
```
#include <stdio.h>
#include <stdlib.h> // para atoi, atof
int main(int argc, char *argv[])
// para argumentos de linea de comando
{ double base;
  int exponent;
  double answer = 1.0;
  int i;
  // chequeo de argumentos de linea de comando
  if (argc != 3){
      printf("Uso: al ejecutar añadir base exponente\n");
      return 1;
    }
```

Ejemplo

```
// convertir argumentos
  base = atof(argv[1]); // use atof para double
  exponent = atoi(argv[2]); // use atoi para enteros
  //chequee si el exponente es negativo
  if (exponent < 0){
      printf("No se aceptan exponentes negativos\n");
      return 2;
    }
  // compute la respuesta
  for (i=0; i<exponent; i++){
      answer *= base;}
   // imprime respuesta
  printf("La respuesta es %lf\n", answer);
  return 0;}
```

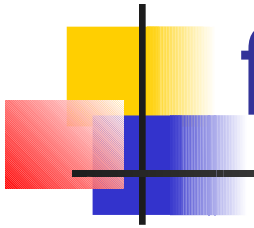
Acceso a archivos

- http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_d
- El estándar de C contiene varias funciones para la edición de ficheros, estas están definidas en la cabecera *stdio.h* y por lo general empiezan con la letra f, haciendo referencia a file. Adicionalmente se agrega un tipo **FILE**, el cual se usará como *apuntador a la información del fichero*. La secuencia que usaremos para realizar operaciones será la siguiente:
- Crear un apuntador del tipo **FILE ***
- Abrir el archivo utilizando la función **fopen** y asignándole el resultado de la llamada a nuestro apuntador.
- Hacer las diversas operaciones (lectura, escritura, etc).
- Cerrar el archivo utilizando la función **fclose**.



fopen

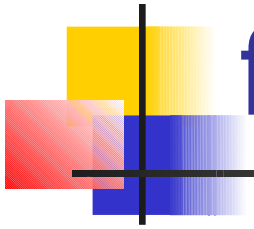
- Esta función sirve para abrir y crear ficheros en disco.
- El prototipo correspondiente de **fopen** es:
  - FILE * fopen (const char *filename, const char *opentype);
- Los parámetros de entrada de **fopen** son:
  - filename: una cadena que contiene un nombre de fichero válido.
  - opentype: especifica el tipo de fichero que se abrirá o se creará.



fopen

- Una lista de parámetros opentype para la función fopen son:

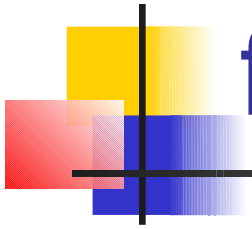
- "r" : abrir un archivo para lectura, el fichero debe existir.
- "w" : abrir un archivo para escritura, se crea si no existe o se sobreescribe si existe.
- "a" : abrir un archivo para escritura al final del contenido, si no existe se crea.
- "r+" : abrir un archivo para lectura y escritura, el fichero debe existir.
- "w+" : crear un archivo para lectura y escritura, se crea si no existe o se sobreescribe si existe.
- "r+b ó rb+" : Abre un archivo en modo binario para actualización (lectura y escritura).
- "rb" : Abre un archivo en modo binario para lectura.



fopen

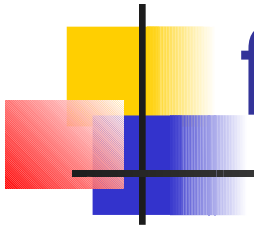
- Si fopen pudo abrir el archivo con éxito devuelve la referencia al archivo

  (FILE *), de lo contrario devuelve NULL y en este caso se deberá revisar la dirección del archivo o los permisos del mismo.



fclose

- Esta función sirve para poder cerrar un fichero que se ha abierto.
- El prototipo correspondiente de fclose es:
  - int fclose (FILE *stream);
- Un valor de retorno cero indica que el fichero ha sido correctamente cerrado, si ha habido algún error, el valor de retorno es la constante EOF.

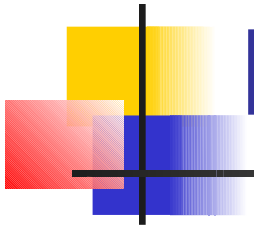


feof

- Esta función sirve para determinar si el cursor dentro del archivo encontró el final (end of file). Existe otra forma de verificar el final del archivo que es comparar el caracter que trae fgetc del archivo con el macro EOF declarado dentro de stdio.h, pero este método no ofrece la misma seguridad (en especial al tratar con los archivos "binarios"). La función feof siempre devolverá cero (Falso) si no es encontrado EOF en el archivo, de lo contrario regresará un valor distinto de cero (Verdadero).

- El prototipo correspondiente de feof es:

  - int feof(FILE *fichero);

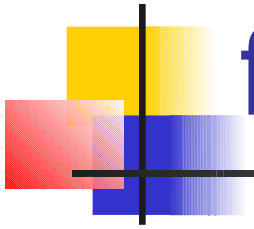


Lectura

- Un archivo generalmente debe verse como un string (una cadena de caracteres) que esta guardado en el disco duro. Para trabajar con los archivos existen diferentes formas y diferentes funciones. Las funciones que podríamos usar para leer un archivo son:

- char fgetc(FILE *archivo)
- char *fgets(char *buffer, int tamano, FILE *archivo)
- size_t fread(void *puntero, size_t tamano, size_t cantidad, FILE *archivo);
- int fscanf(FILE *fichero, const char *formato, argumento, ...);

- Las primeras dos de estas funciones son muy parecidas entre si.



fgetc

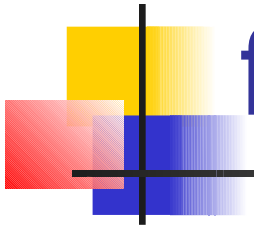
- Esta función lee un caracter a la vez, del archivo que esta siendo señalado con el puntero *archivo. En caso de que la lectura sea exitosa devuelve el caracter leído y en caso de que no lo sea o de encontrar el final del archivo devuelve EOF.
- El prototipo correspondiente de fgetc es:
  - char fgetc(FILE *archivo);
- Esta función se usa generalmente para recorrer archivos de texto.

Ejemplo

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

int main()
{
        FILE *archivo;
        char caracter;

        archivo = fopen("prueba.txt","r");

        if (archivo == NULL){

          printf("\nError de apertura del archivo. \n\n");
        }
```

Ejemplo

```
else{
      printf("\nEl contenido del archivo de prueba es \n\n");
      while (feof(archivo) == 0){
                caracter = fgetc(archivo);
                printf("%c",caracter);
           }}
        fclose(archivo);
        return 0;}
```

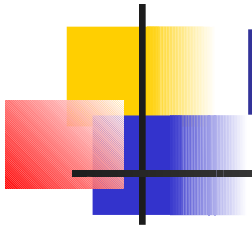


fscanf

- La función fscanf funciona igual que scanf en cuanto a parámetros, pero la entrada se toma de un fichero en lugar del teclado.

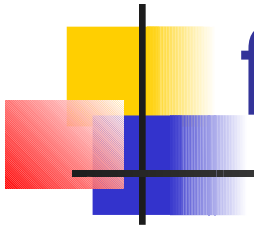
- El prototipo correspondiente de fscanf es:

  - int fscanf(FILE *fichero, const char *formato, argumento, ...);



Ejemplo

```
#include <stdio.h>
int main ( int argc, char **argv ){
        FILE *fp;
        char buffer[100];
        fp = fopen ( "prueba.txt", "r" );
        fscanf(fp, "%s" ,buffer);//lee hasta el blanco
        printf("%s",buffer);
        fclose ( fp );
        return 0;}
```



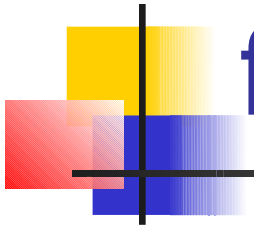
fgets

- Esta función está diseñada para leer cadenas de caracteres. Leerá hasta n-1 caracteres o hasta que lea un retorno de línea. En este último caso, el carácter de retorno de línea también es leído.

- El prototipo correspondiente de fgets es:

  - char *fgets(char *buffer, int tamaño, FILE *archivo);

- El primer parámetro buffer lo hemos llamado así porque es un puntero a un espacio de memoria del tipo char (podríamos usar un arreglo de char). El segundo parámetro es tamaño que es el limite en cantidad de caracteres a leer para la funcion fgets. Y por ultimo el puntero del archivo por supuesto que es la forma en que fgets sabrá a que archivo debe leer.



fgets

- La función fgets se comporta de la siguiente manera, leerá del archivo apuntado por archivo los caracteres que encuentre y a ponerlos en buffer hasta que lea un caracter menos que la cantidad de caracteres especificada en tamaño o hasta que encuentre el final de una linea (\n) o hasta que encuentre el final del archivo (EOF).
- El beneficio de esta función es que se puede obtener una linea completa a la vez. Y resulta muy útil para algunos fines como la construcción de un parser de algún tipo de archivo de texto.

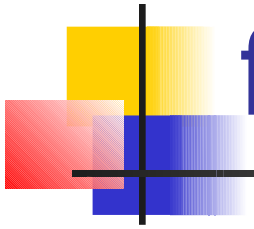
Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        FILE *archivo;
        char caracteres[100];
        archivo = fopen("prueba.txt","r");
        if (archivo == NULL)
                exit(1);
        printf("\nEl contenido del archivo de prueba es
\n\n");
        while (feof(archivo) == 0){
                fgets(caracteres,100,archivo);
                printf("%s",caracteres);}
        fclose(archivo);
        return 0;}
```

Escritura

- Así como podemos leer datos desde un fichero, también se pueden crear y escribir ficheros con la información que deseamos almacenar. Para trabajar con los archivos existen diferentes formas y diferentes funciones. Las funciones que podríamos usar para escribir dentro de un archivo son:

- int fputc(int caracter, FILE *archivo)
- int fputs(const char *buffer, FILE *archivo)
- size_t fwrite(void *puntero, size_t tamano, size_t cantidad, FILE *archivo);
- int fprintf(FILE *archivo, const char *formato, argumento, ...);

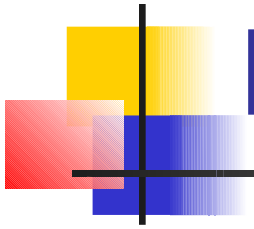


fputc

- Esta función escribe un carácter a la vez del archivo que esta siendo señalado con el puntero *archivo. El valor de retorno es el carácter escrito, si la operación fue completada con éxito, en caso contrario será EOF.

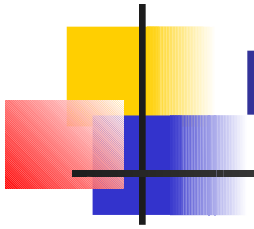
- El prototipo correspondiente de fputc es:

- int fputc(int carácter, FILE *archivo);



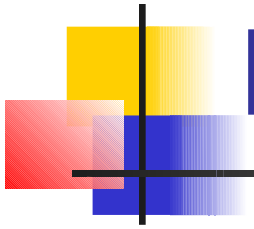
Ejemplo

```
#include <stdio.h>
int main ( int argc, char **argv ){
        FILE *fp;
        char caracter;
        fp = fopen ( "prueba.txt", "r+" );//lectura/escritura
        printf("\nIntroduce un texto al fichero:");
        //hasta que pulse ENTER, sobreescribe
         while((caracter = getchar()) != '\n'){
                printf("%c", fputc(caracter, fp));}
        printf("\n");
        fclose ( fp );
        return 0;}
```



rewind

- Literalmente significa "rebobinar", sitúa el cursor de lectura/escritura al principio del archivo.
- El prototipo correspondiente de rewind es:
  - void rewind(FILE *fichero);

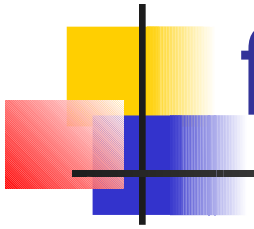


Dispositivos de I/O estándar

- stdin corresponde a la entrada estándar (por defecto, el teclado).
- stdout corresponde a la salida estándar (por defecto, la pantalla).
- stderr corresponde a la salida de error estándar (por defecto, la pantalla).

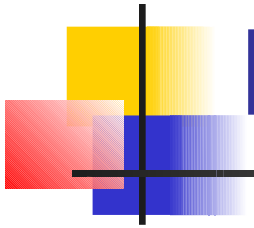
Ejemplo

```
// ejemplo1.cpp: Muestra este fichero dos veces.
#include <stdio.h>
#include <stdlib.h>
int main(){
FILE *fichero;
fichero = fopen("ejemplo1.cpp", "r");
while(!feof(fichero)) fputc(fgetc(fichero), stdout);
rewind(fichero);
while(!feof(fichero)) fputc(fgetc(fichero), stdout);
fclose(fichero);
return
0;}//www.lcc.uma.es/~afdez/apuntes/metodologia/ficheros/ficheros.PD
F
```



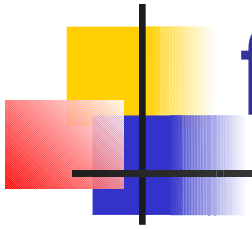
fprintf

- fprintf
- La función fprintf funciona igual que printf en cuanto a parámetros, pero la salida se dirige a un archivo en lugar de a la pantalla.
- El prototipo correspondiente de fprintf es:
  - int fprintf(FILE *archivo, const char *formato, argumento, ...);



Ejemplo

```
#include <stdio.h>
int main ( int argc, char **argv ){
        FILE *fp;
        char buffer[100] = "Esto es un texto dentro del
fichero.";
        fp = fopen ( "prueba.txt", "r+" );
        fprintf(fp, "%s", buffer);//Sobreescribe el archivo
        fprintf(fp, "%s", "\nEsto es otro texto dentro del
fichero.");//ídem
        fclose ( fp );
        return 0;}
```

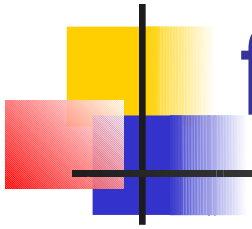


fputs

- La función fputs escribe una cadena en un fichero. No se añade el carácter de retorno de línea ni el carácter nulo final. El valor de retorno es un número no negativo o EOF en caso de error. Los parámetros de entrada son la cadena a escribir y un puntero a la estructura FILE del fichero donde se realizará la escritura.

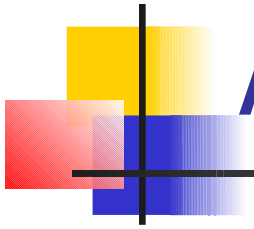
- El prototipo correspondiente de fputs es:

  - int fputs(const char *buffer, FILE *archivo)



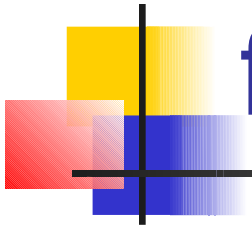
fputs

```
#include <stdio.h>
int main ( int argc, char **argv ){
        FILE *fp;
        char cadena[] = "Mostrando el uso de fputs en un fichero.\n";
        fp = fopen ( "prueba.txt", "r+" );
        fputs( cadena, fp );
        fclose ( fp );
        return 0;}
```



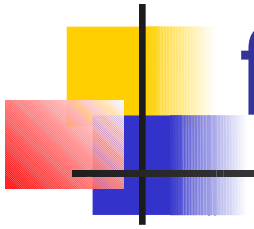
Archivos binarios

- Para generar archivos binarios podemos usar fwrite y para leer los mismos fread; estas funciones también están diseñadas para guardar de forma binaria tipos de datos complejos como estructuras (serialización de datos)
- Es aconsejable utilizarlas en pareja; es decir, si se escribe con fwrite se debe leer con fread



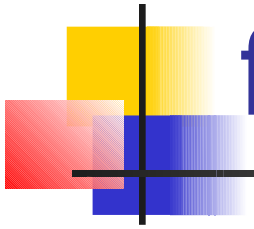
fwrite

- Esta función está pensada para trabajar con registros de longitud constante y forma pareja con fread. Es capaz de escribir hacia un fichero uno o varios registros de la misma longitud almacenados a partir de una dirección de memoria determinada.
- El valor de retorno es el número de registros escritos, no el número de bytes.



fwrite

- Los parámetros son: un puntero a la zona de memoria de donde se obtendrán los datos a escribir, el tamaño de cada registro, el número de registros a escribir y un puntero a la estructura FILE del fichero al que se hará la escritura.



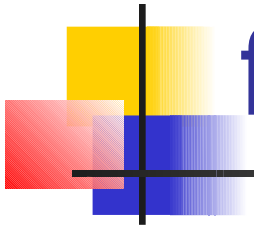
fwrite

- El prototipo correspondiente de fwrite es:

  - size_t fwrite(void *puntero, size_t tamano, size_t cantidad, FILE *archivo);

fwrite

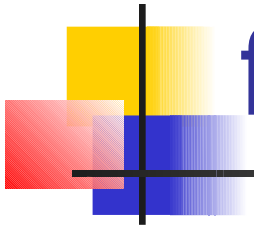
```
FlLE *f;//http://mimosa.pntic.mec.es/~flarrosa/fichc.pdf

int v[6], elem_escritos, num;

f = fopen (“datos.dat", "wb ");

/* Para escribir los elementos 2, 3 y 4 de v */

elem-escritos = fwrite (&v[2], sizeof(int), 3, f );

/* Para escribir el primer elemento de v, valen las 2
instrucciones siguientes */

fwrite (v, sizeof (int), 1, f );

fwrite (&v[0], sizeof(int), 1, f );

/* Para escribir un entero valen las dos siguientes */

fwrite (&num, sizeof(int), 1, f);

fwrite (&num, sizeof(num), 1, f);
```



fread

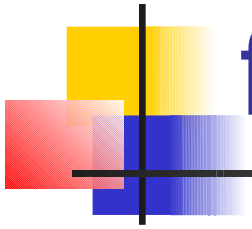
- size_t fread (void * ptr, size_t size, size_t count, FILE * stream);

- Esta función lee un bloque de una "stream" de datos. Efectúa la lectura de un arreglo de elementos "count", cada uno de los cuales tiene un tamaño definido por "size". Luego los guarda en el bloque de memoria especificado por "ptr". El indicador de posición de la cadena de caracteres avanza hasta leer la totalidad de bytes. Si esto es exitoso la cantidad de bytes leídos es (size*count).



fread

- Parámetros:
- ptr : Puntero a un bloque de memoria con un tamaño mínimo de (size*count) bytes.
- size : Tamaño en bytes de cada elemento (de los que voy a leer).
- count : Número de elementos, los cuales tienen un tamaño "size".
- stream: Puntero a objetos FILE, que especifica la cadena de entrada.

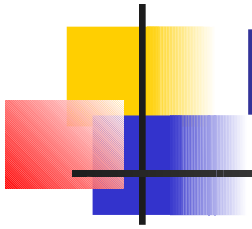


fread

```
f = fopen (“datos.dat ", "rb ");

int elem-escritos = fread (&v[2], sizeof(int),
3,f);

fread (v, sizeof(int), 1, f);

fread (&v[0], sizeof(int), 1, f);

fread (&num, sizeof(int), 1, f);

fread (&num, sizeof(num), 1, f);
```

Ejemplo de wiki

```
#include <stdio.h>

void menu();
void CrearFichero(FILE *Fichero);
void InsertarDatos(FILE *Fichero);
void VerDatos(FILE *Fichero);

struct sRegistro {
   char Nombre[25];
   int Edad;
   float Sueldo;
} registro;
```

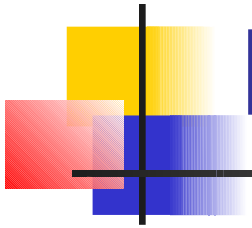


Ejemplo

```
int main(){
        int opcion;
        int exit = 0;
        FILE *fichero=NULL;
         while (!exit){
                menu();
                printf("\nOpcion: ");
                scanf("%d", &opcion);
                fflush(stdin);
```

Ejemplo

```
switch(opcion){        case 1:
                               CrearFichero(fichero);        break;
                       case 2:
                               InsertarDatos(fichero);       break;
                       case 3:
                               VerDatos(fichero);            break;
                       case 4:
                               exit = 1;                     break;
                       default:
                               printf("\nopcion no valida");
              }}
       return 0;}
```



Ejemplo

```
void menu()
{
        printf("\nMenu:");
        printf("\n\t1. Crear fichero");
        printf("\n\t2. Insertar datos");
        printf("\n\t3. Ver datos");
        printf("\n\t4. Salir");
}
```

Ejemplo

```
void CrearFichero(FILE *Fichero){
        Fichero = fopen("fichero", "r");
         if(!Fichero){
                Fichero = fopen("fichero", "w");
                printf("\nArchivo creado!");}
        else{
                printf("\nEl fichero ya existe!");}
         fclose (Fichero);
         return;
}
```

Ejemplo

```
void InsertarDatos(FILE *Fichero){
        Fichero = fopen("fichero", "a+");
        if(Fichero == NULL){
                printf("\nFichero no existe! \nPor favor creelo");
                return;}
        printf("\nDigita el nombre: ");
        gets(registro.Nombre);
        printf("\nDigita la edad: ");
        scanf("%d", &registro.Edad);
        printf("\nDigita el sueldo: ");
        scanf("%f", &registro.Sueldo);
         fwrite(&registro, sizeof(struct sRegistro), 1, Fichero);
         fclose(Fichero);
         return;}
```

Ejemplo

```
void VerDatos(FILE *Fichero){
    int numero = 1;
    Fichero = fopen("fichero", "r");
    if(Fichero == NULL){
            printf("\nFichero no existe! \nPor favor creelo");
            return;}
     fread(&registro, sizeof(struct sRegistro), 1, Fichero);
     printf("\nNumero \tNombre \tEdad \tSueldo");
     while(!feof(Fichero)){
            printf("\n%d \t%s \t%d \t%.2f", numero, registro.Nombre,
            registro.Edad, registro.Sueldo);
            fread(&registro, sizeof(struct sRegistro), 1, Fichero);
            numero++;}
     fclose(Fichero);          return;}
```

Acceso directo a archivos

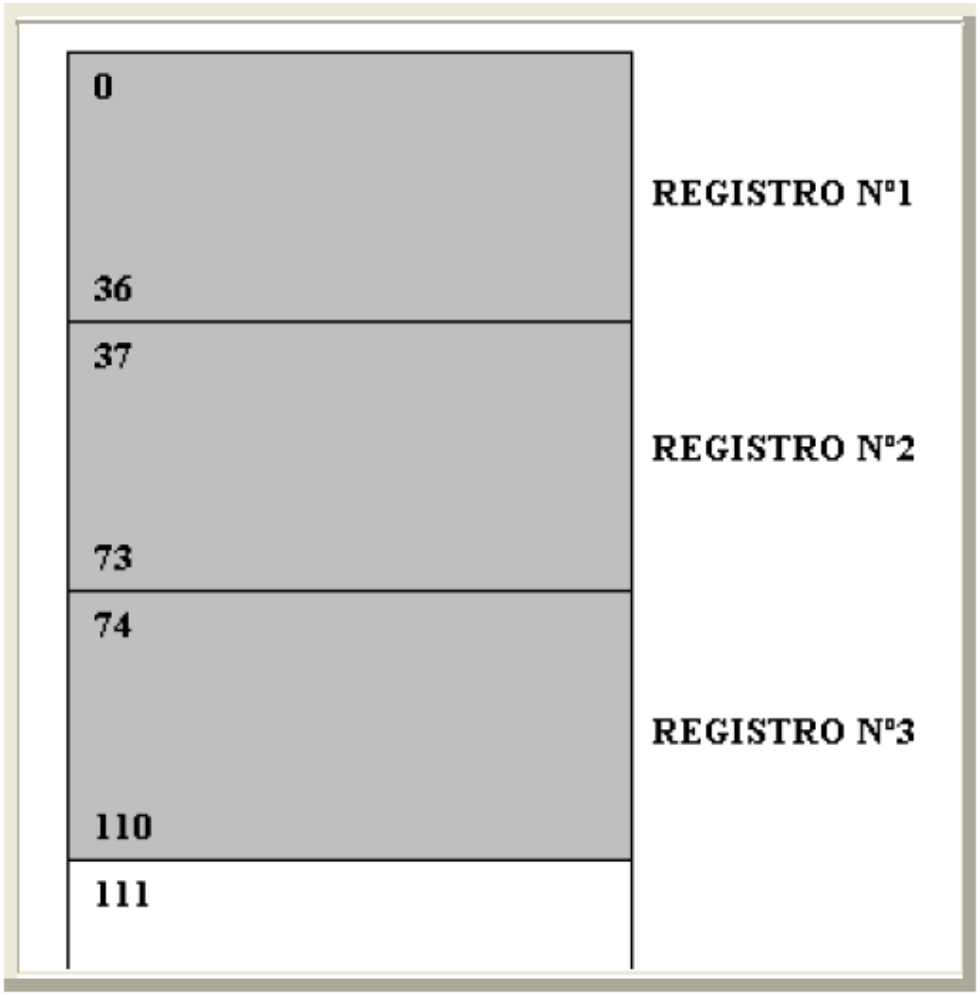
- Supongamos que tenemos definido un archivo con la siguiente estructura de registro:
- struct{

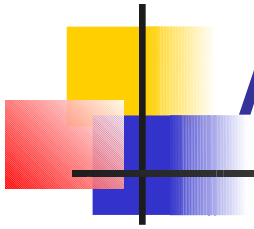
```
int codigo;
char nomart[31];
float precio;
}articulo;
```

Acceso directo a archivos

- La longitud de cada registro es de 37 bytes (2+31+4 bytes).
- De esta forma, la disposición de los registros dentro del archivo en disco se realiza en las siguientes posiciones:

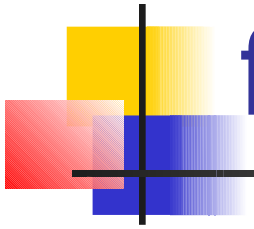
Acceso directo a archivos





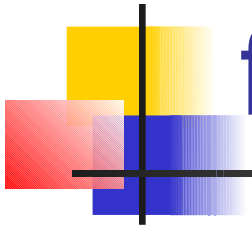
Acceso directo a archivos

- El acceso directo consiste en indicar la posición a la que queremos acceder en bytes.
- Por ejemplo, para acceder directamente al registro 2, indicaremos que queremos ir al byte 37, contando desde el principio del registro. Las órdenes que posibilita este acceso son fseek y ftell



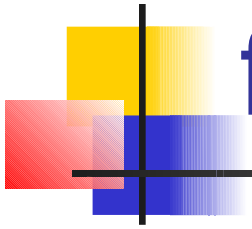
fseek

- <valor> = fseek (<var_fich>, <dirección>, <desde>);
- valor nos dirá si ha ocurrido un error al desplazarnos por el fichero. 0 si todo ha ido bien.
- var_fich es el puntero al archivo que estamos utilizando.
- dirección es el número de bytes que vamos a desplazarnos (desplazamiento), es un long int
- desde: es el punto de partida del desplazamiento. Este punto admite tres posibles valores:



fseek

Referencia	Valor	Desde
SEEK_SET	0	Principio de fichero
SEEK_CUR	1	Posición actual del fichero
SEEK_END	2	Final del fichero



fseek

- fseek(f1, 100L, SEEK_SET) nos situaremos en el byte 100 del fichero, empezando a contar desde el principio.
- fseek(f1, -100L, SEEK_END) nos situaremos 100 bytes antes del último byte del fichero.
- Si acabamos de abrir el fichero y leemos cien bytes y luego hacemos fseek(f1, 100, SEEK_CUR), nos situaremos 100 bytes después del último byte leido, que era el 100, así que nos situaremos en el 200.

Ejemplo

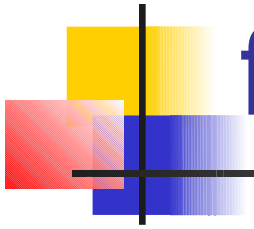
```
#include <stdio.h>
typedef struct {
   char nombre[20];
   int edad;
} persona;
int main(void){
        FILE *f1;
        persona dato;
        int i;
f1 = fopen ("persona.dat", "wb");
if (f1 == NULL){
   perror("No se puede abrir persona.dat");
   return -1;}
```

Ejemplo

```
/* Escribimos 10 datos, que serán Juan0, Juan1, Juan2,
Juan3...con edad 0,1,2,3....*/
for (i=0; i<10; i++){
   sprintf (dato.nombre, "Juan%d", i);
//sprintf: salida formateada a un string
   dato.edad=i;
   /* El tamaño es sizeof(dato) y escribimos un registro */
   fwrite (&dato, sizeof(dato), 1, f1);}
/* Cerramos el fichero */
fclose(f1);
return 0;}
```

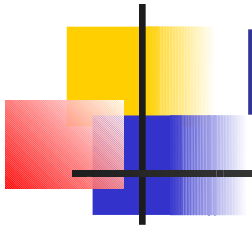
Ejemplo

```
int main(void){
        FILE *f1;
        persona dato;
f1 = fopen ("persona.dat", "rb+");
if (f1 == NULL){
   perror("No se puede abrir persona.dat");
   return -1;}
/* Vamos al cuarto registro, indice 3 Juan 3 de edad 3*/
fseek (f1, 3*sizeof(dato), SEEK_SET);
/* Leemos y sacamos el resultado por pantalla */
fread (&dato, sizeof(dato), 1, f1);
printf ("nombre = %s\n", dato.nombre);
printf ("edad = %d\n", dato.edad);
fclose(f1); return 0;}
```



ftell

- <posición> = ftell (<var_fich>);
- posición es la variable que contendrá la posición en bytes en la que nos encontramos en ese momento en el fichero.
- var_fich es la variable declarada como FILE.

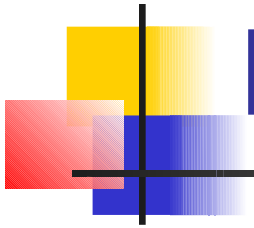


Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
FILE *ifp;
long pos;
int c;
/* Ejecutar con 2 argumentos, el 2º es el nombre del archivo a leer */
if (argc != 2) {
fprintf(stderr, "Error\n"); exit(1);}
if ((ifp = fopen(argv[1], "r")) == NULL) {
fprintf(stderr, "Could not open file: %s\n", argv[1]); exit(1);}
```

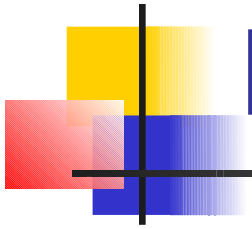
Ejemplo

```
c = fgetc(ifp);
printf("%c\n", c); /* imprime el 1º char, offset=0 */
fseek(ifp, 5L, SEEK_SET);
c = fgetc(ifp);
printf("%c\n", c); /* imprime el caracter con offset=5 desde el principio */
pos = ftell(ifp);
printf("%ld\n", pos); /* imprime el offset,queda en 6 */
fseek(ifp, 3L, SEEK_CUR); /* va a 3 más del offset actual */
c = fgetc(ifp); /* alli lee un char y lo imprime */
printf("%c\n", c);
pos = ftell(ifp); printf("%ld\n", pos); /* imprime offset, queda en 10. */
```



Ejemplo

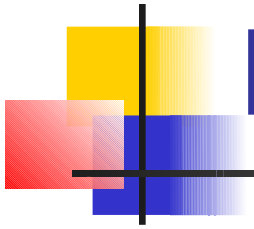
```
fseek(ifp, -4L, SEEK_CUR); /* va al caracter cuyo offset es 4 menos del actual */
c = fgetc(ifp);
printf("%c\n", c);
pos = ftell(ifp); printf("%ld\n", pos);//offset 7
fseek(ifp, -5L, SEEK_END); /* va al caracter que esta 5 menos del final */
c = fgetc(ifp);
printf("%c\n", c);
pos = ftell(ifp); printf("%ld\n", pos);//12
rewind(ifp);//va al principio
```



Ejemplo

```
c = fgetc(ifp);
printf("%c\n", c);
pos = ftell(ifp);
printf("%ld\n", pos);//1
if (fclose(ifp) == EOF) {
fprintf(stderr, "No puede cerrar el archivo:
%s\n", argv[1]);}
return 0;}
```

Ejemplo



- Archivo: abcdefghijklmno\n
- Salida por pantalla:

a
f
6
j
10
g
7
l
12
a
1