Ficheros de Texto

Eva Lucrecia Gibaja Galindo
Dpto. Informática y Análisis Numérico

Introducción

- **Fichero.** Un fichero es un conjunto de información relacionada, grabada en un sistema de almacenamiento secundario y a la que se hace referencia mediante un nombre
- Los ficheros permiten conservar datos de forma ***permanente*** ➔ accesibles para diferentes programas y ejecuciones
  - **Fichero físico**: Bloque de información que queda almacenado en un dispositivo de almacenamiento masivo (discos duros, etc.)
  - **Fichero/flujo/*stream* en C**: Fuente o destino de datos asociado con un disco o periférico. Abstracción de un fichero físico, independiente del dispositivo

Introducción

- C ve a cada fichero como una **secuencia de *bytes***
  - **Ficheros de texto**: La unidad de almacenamiento es el carácter (*byte*)
  - **Ficheros binarios**: Otro tipo de estructura
- El lenguaje C proporciona una serie de *tipos* y *funciones*, definidas en *stdio.h,* que hacen de interfaz con el sistema operativo:
  - El programador sólo debe preocuparse de usarlas adecuadamente
  - Las operaciones a nivel físico sobre el fichero las realiza el sistema operativo
- Utilización de un fichero en un programa:

  1. Abrir fichero → 2. Procesar fichero → 3. Cerrar fichero
- Existen 3 ficheros predefinidos en C que se abren cuando comienza la ejecución de un programa:
  - *stdin*: entrada estándar → teclado
  - *stdout*: salida estándar → pantalla
  - *stderr*: salida de errores estándar → pantalla

Declaración de un fichero

- Cada fichero abierto tiene asociada una estructura de tipo `FILE` (**`stdio.h`**), interfaz entre el programa y el sistema operativo
  - La estructura `FILE` varía de un sistema operativo a otro. Esto es irrelevante para el programador, pues las operaciones a nivel físico sobre el fichero las realiza el sistema operativo
- Las funciones de procesamiento de ficheros utilizan un puntero esta estructura (**`FILE*`**)
- La estructura `FILE` almacena:
  - ***Buffer***. Almacena temporalmente la información intercambiada entre la memoria y el archivo
  - ***Cursor (file position indicator, file position pointer)***. Posición actual de L/E. Se mueve automáticamente cuando se lee o escribe
  - códigos de error
  - modo de apertura, etc.

```
typedef struct {
  int    _cnt;
  char *_ptr;
  char *_base;
  int    _bufsiz;
  int    _flag;
  int    _file;
  char *_name_to_remove;
  int    _fillsize;
} FILE;
```

Ficheros de texto

- ¿Cómo representar la información en un fichero de texto y binario?
  - Modelo: Atardecer
    - Precio: 5.7 euros
    - Unidades: 12
  - Modelo: Sol naciente
    - Precio: 8 euros
    - Unidades: 5
  - Modelo: Ibiza beach
    - Precio: 15 euros
    - Unidades: 3

(A) Exceso de información, postprocesado

```
Modelo: Atardecer
Precio: 5.7
Unidades: 12
Modelo: Sol naciente
Precio: 8
Unidades: 5
Modelo: Ibiza beach
Precio: 15
Unidades: 3
```

(D)

```
Atardecer
5.7
12
Sol naciente
8
5
Ibiza beach
15
3
```

- ¿Cuánto ocupa cada producto?
  - **Fichero de texto**: Cada producto ocupará un número de *bytes* (caracteres) diferente
  - **Fichero binario**: Todos los registros ocupan **sizeof(struct producto)**

Binario

```
struct producto
{   char modelo[100];
    float precio;
    int unidades;
};
```

(B)Postprocesado

```
Atardecer 5.7 12
Sol_naciente 8 5
Ibiza_beach 15 3
```

(C)Postprocesado

```
Atardecer*5.7*12
Sol naciente*8*5
Ibiza beach*15*3
```

(E)

```
Atardecer
5.7 12
Sol naciente
8 5
Ibiza beach
15 3
```

Ficheros de texto

- Los datos se almacenan de manera legible para las personas (se pueden inspeccionar con ayuda de un editor de texto)
- Son necesarias marcas de separación entre los diferentes elementos
- Las **marcas de separación** son caracteres que decide el programador
  - Normalmente suelen ser espacios en blanco, tabuladores o saltos de línea
  - Cuando los separadores son espacios en blanco, se permite libertad en cuanto a su número
  - Los caracteres separadores **aumentan el tamaño de los ficheros**

Apertura de ficheros de texto

```
FILE* fopen (const char* nombre, const char* modo);
```

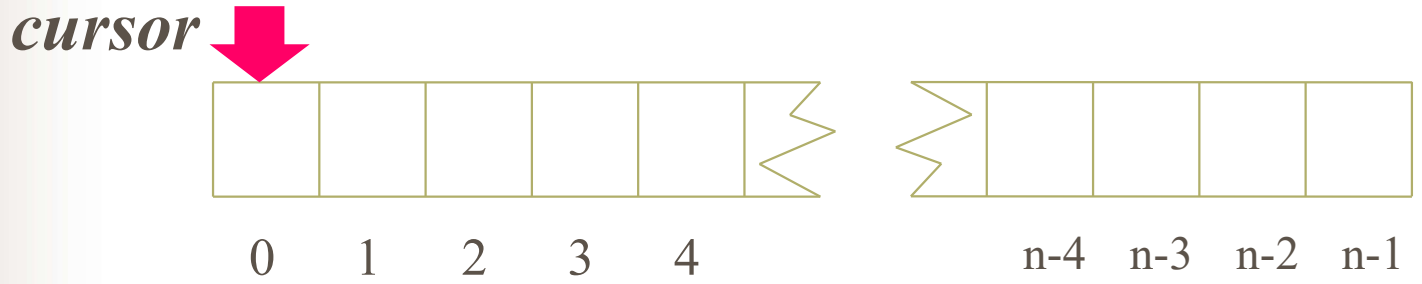
- Abre el fichero físico cuyo nombre está referenciado por `nombre`. Asocia un fichero a un archivo físico
- Devuelve:
  - Un puntero a `FILE`
  - `NULL` si hay algún problema durante su apertura
    - Abrir en modo "*r*" y el fichero no existe
    - Abrir en modo "*w*" y el disco está protegido contra escritura

Notas:

- Los nombres de archivo están limitados a `FILENAME_MAX` caracteres
- No pueden ser abiertos más de `FOPEN_MAX` archivos a la vez

Apertura de ficheros de texto

Modo	Acciones	Cursor	Si existe	Si no existe
r	Lectura	Inicio	Abre	Código error
w	Escritura	Inicio	Borra contenido	Crea
a	Adición	Final	Abre. Agrega al final	Crea
r+	L/E	Inicio	Abre. Agrega al inicio, sobreescribiendo	Código error
w+	L/E	Inicio	Borra contenido	Crea
a+	L/Adición	Final	Abre. Agrega al final	Crea



Cierre de ficheros

No olvidar que hay que cerrar siempre los ficheros

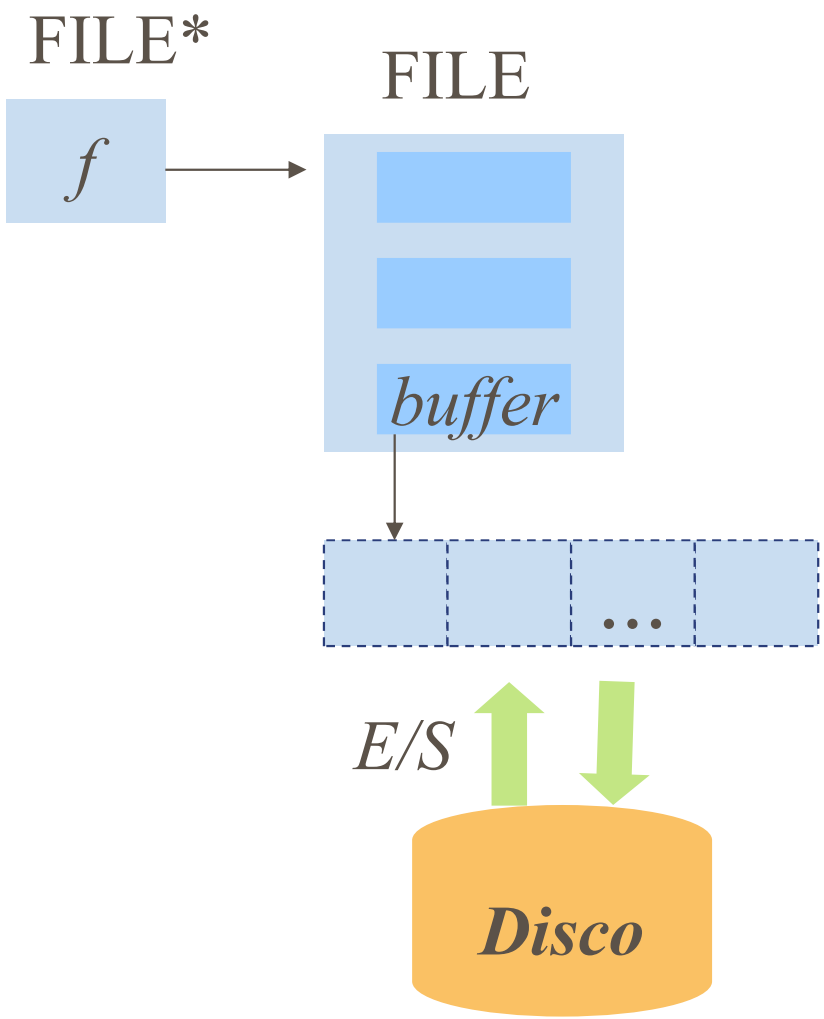
```
int fclose (FILE* f);
```

- Interrumpe la conexión establecida por `fopen()` entre el puntero de archivo (`FILE *`) y el nombre del fichero
  - Vacía el *buffer* (si el fichero es de salida o escritura)
  - Devuelve 0 si todo va bien o `EOF` (-1) si hay algún problema
  - Actualiza algunos datos de la cabecera del archivo que mantiene el sistema operativo
- Observaciones importantes:
  - Un fichero no puede abrirse más de una vez sin antes haberlo cerrado
  - ¿Qué ocurre si termina una aplicación sin cerrar el fichero?
    - El estándar de C no lo define, por lo que puede pasar cualquier cosa, el archivo puede quedar incoherente o se pueden perder datos del *buffer*
    - Puede que el fichero no pueda ser abierto por otra aplicación

    → Es **obligatorio** cerrar el fichero cuando se haya terminado su uso dentro del programa

Cierre de ficheros. El buffer

- Las llamadas al núcleo del sistema (*system calls*) para operaciones de E/S ocupan una gran cantidad de tiempo, comparadas con una llamada a función
- Los accesos a disco consumen mucho más tiempo que los accesos a memoria
- Todo fichero tiene asociado un *buffer* (zona de memoria) para realizar las operaciones de E/S de forma más rápida
  - La escritura no se realiza directamente sobre el disco, sino que se escribe en el *buffer*. Cuando está lleno, la información pasa al disco
  - Al cerrar el fichero, se vuelca el *buffer* en el disco



abreLectura.c

fich_datos.txt

Ejemplos. Apertura en modo lectura

```
#include <stdio.h>
int main()
{
  FILE* f; //Declaración de un fichero
  //Apertura del fichero para lectura
  if((f=fopen("fich_datos.txt", "r"))==NULL)
  {
    printf("\nError al abrir fichero <%s>", "fich_datos.txt");
    exit(-1);
  }
  //Instrucciones de procesamiento del fichero
  //..............

  //Cierre del fichero
  fclose(f);
  return(0);
}
```

abreEscritura.c

fich_datos.txt

Ejemplos. Apertura en modo escritura

Modo "w" si el fichero existe se pierde su contenido!!

```
#include <stdio.h>
int main()
{
  FILE* f; //Fichero
  //Apertura del fichero para escritura
  if((f=fopen("..\\fich_datos.txt", "w"))==NULL)
  {
    printf("\nError, no se pudo crear fichero <%s>",
    "fich_datos.txt");
    exit(-1);
  }
  //Instrucciones de procesamiento del fichero
  //..............
  //Cierre del fichero
  fclose(f);
  return(0);
}
```

Crea el fichero en el directorio padre

Es fundamental que nos acordemos de cerrar el fichero para que la información pase del buffer al disco

existe.c

fichero.txt

Ejemplos. Comprobar existencia

```
int existeFichero(char *fichero)
{
  FILE *pFichero;

  pFichero = fopen(fichero, "r"); /* abre fichero para lectura */
  if (pFichero == NULL) /* el fichero no existe, devolver 0 */
  {
    return 0;
  }
  else /* el fichero existe, devolver 1 */
  {
    fclose (pFichero);
    return 1;
  }
}
```

EOF y constante EOF

- EOF: Constante definida en C (*#define EOF -1*) de tipo entero y es devuelta por algunas funciones
- EOF (*end-of-file*): indicador de que no hay más información que recuperar en el archivo
  - En UNIX se puede generar un EOF desde la consola tecleando *Ctrl+D* para indicar el EOF de datos introducidos por teclado
  - En DOS y Windows se genera mediante la combinación *Ctrl+Z*
  - Esto no quiere decir que un fichero termine almacene el EOF

El cursor se incrementa automáticamente cada vez que se procesa un registro (se lee o se escribe)

E/S con Formato

`int fprintf (FILE* f, const char* formato,...);`

- Escribe en el fichero referenciado por `f` el resultado de la conversión impuesto por el formato expresado en `formato`.
- Devuelve el número de caracteres escritos, o un valor negativo si se produce algún error
- **`int printf (const char* formato,...);`**
  - Equivale a `fprintf(stdout, formato, ...);`

`int fscanf (FILE *f, const char *formato, ...);`

- Entrada de datos con la conversión establecida en `formato` leyendo del fichero referenciado por `f`
- Devuelve EOF si se alcanza el fin del fichero o se produce un error durante la conversión. En otro caso, devuelve el número de objetos convertidos y asignados
- **`int scanf(const char*formato,...);`**
  - Equivale a `fscanf(stdin, formato, ...);`

Ejemplos

- Ante un fichero con la siguiente estructura:

```
Juan Lopez Lopez          100   4.15   100000
Antonio Ruperez Ruperez 77     5.67   40000
Luis Gomez Gomez          44 8.99   66000
```

- Se desea una presentación como la siguiente:

```
Cliente Num. 100 : Lopez Lopez, Juan
    Deuda total =   100000. Interes = 4.15
Cliente Num. 77  : Ruperez Ruperez, Antonio
    Deuda total =   40000.  Interes = 5.67
Cliente Num. 44  : Gomez Gomez, Luis
    Deuda total =   66000.  Interes = 8.99
```

leeClientes.c

clientes.txt

Ejemplos

```
#define MAX 256
#include <stdio.h>
int main()
{
  FILE *f; //referencia al fichero de clientes
  char nombreF[]="clientes.txt"; //nombre del fichero de clientes
  char nombre[50], apell1[50], apell2[50]; //nombre y apellidos
  int num; //codigo de cliente
  float interes; //tipo de interes
  int total; //deuda total

  //Apertura del fichero
  if((f=fopen(nombreF, "r"))==NULL)
  { fprintf(stderr, "\nError: no pudo abrirse fichero <%s>",
    nombreF);
    exit(-1);
  }
```

leeClientes.c

clientes.txt

Ejemplos

También serviría un **while(fscanf(. . .)!=EOF)**

```
//Procesamiento del fichero
while(fscanf(f, "%s%s%s%d%f%d", nombre, apell1, apell2,&num,
  &interes, &total)==6)
//También serviría while (fscanf(…)!=EOF)
{
  //Presentamos los datos obtenidos
  printf("\nCliente Num. %3d: %s %s, %s", num, apell1, apell2,
  nombre);
  printf("\n\tDeuda total= %8d. Interes=%4.2f\n", total,
  interes);
}

//Cierre del fichero
fclose(f);
return(0);
}
```

E/S de Líneas

```
char* fgets (char* s, int n, FILE* f);
```

- Lee como mucho *n-1* caracteres del fichero referenciado por *f* y los copia en la cadena *s*, parando cuando encuentra *'\n'*
- Devuelve *s* o NULL si encuentra EOF o hay error

```
int fputs (const char* s, FILE* f);
```

- Escribe la cadena s en el fichero referenciado por *f*. Devuelve un valor no negativo si tiene éxito o EOF si ocurre un error de escritura

Relación con funciones de cadenas:

- `char* gets (char* s);`
  - Lee una línea de *stdin* en *s*
  - Devuelve *s*, o NULL si encuentra EOF o hay error
    - gets() reemplaza el carácter *'\n'* por *'\0'* mientras que fgets lo mantiene
    - gets() lee de *stdin* hasta encontrar el carácter *'\n'* mientras que fgets(s, n, stdin) lee hasta encontrar el carácter *'\n'* o hasta que se han leído *n* caracteres
- `int puts (const char* s);`
  - Escribe la cadena *s* y un salto de línea en *stdout*.
  - Devuelve un valor no negativo si tiene éxito o EOF si hay error de escritura

fgetsfeof.c
personas.txt

Ejemplos

```
void leeVersion1(char* nombreFichero)
{
   FILE* fich;
   char cadena[30];

   fich=fopen(nombreFichero, "r");
   while(fgets(cadena, 30, fich)!=NULL)
   {
      if(cadena[strlen(cadena)-1]=='\n')
       cadena[strlen(cadena)-1]='\0';
     printf("<%s>\n", cadena);
   }
   fclose(fich);

}
```

Claudia
Gonzalez Ruiz
25
Alfonso
Ramirez Luque
19

Lectura

fgetsbasico.c
edadSaldo.txt

Ejemplos

```
void leeFichero(char* nombreFichero)
{  FILE* fich;
   char cadena[30];
   int edad;
   float saldo;
   fich=fopen(nombreFichero, "r");
   while(fgets(cadena, 30, fich)!=NULL)
   { if(cadena[strlen(cadena)-1]=='\n')
       cadena[strlen(cadena)-1]='\0';
     printf("\nNombre y apellidos: <%s>\n", cadena);

     fgets(cadena, 30, fich);
     sscanf(cadena, "%d", &edad);
     printf("\n\tEdad: <%d>\n", edad);

     fgets(cadena, 30, fich);
     sscanf(cadena, "%f", &saldo);
     printf("\n\tSaldo: <%f>\n", saldo);
   }
   fclose(fich);
}
```

```
Juan Jose Lopez
100
4.15
Antonio Ruperez Ruperez
77
5.67
Luis Gomez Gomez
44
8.99
```

E/S sobre ficheros de texto: funciones simples

***Ojo!! Devuelven un* int *no un* char**

Lectura de un carácter

```
int fgetc (FILE *f); int getc (FILE *f);
```

- Devuelven el siguiente carácter del fichero referenciado por *f*
- Avanza el cursor una posición hasta el siguiente carácter
- `EOF` si encuentra el fin del fichero o hay error
- `int getchar (void);`
  - Equivalente a `getc(stdin);`

Escritura de un carácter

```
int fputc (int c, FILE *f); int putc (int c, FILE *f);
```

- Escribe el carácter en el fichero referenciado por *f* y lo devuelve
- `EOF` si hay error
- `int putchar (int c);`
  - Equivalente a `putc(c, stdout);`

E/S sobre ficheros de texto: funciones simples

- La rutina `getc` es funcionalmente idéntica a `fgetc`, pero está implementada como una **macro** ⇒
  - Se ejecuta más rápido que `fgetc`
  - Necesita más espacio en memoria para su invocación
  - Su nombre no se puede pasar como argumento en una llamada a función
  - No funciona correctamente con un parámetro que tenga efectos colaterales (`getc(*f++)`)
- Es más habitual `fgetc`

escribe_A.c

Ejemplos. fputc(). *Escribir la letra 'A' en un fichero*

```
#include <stdio.h>
int main()
{ FILE* f;
  //Apertura del fichero en modo escritura: creacion del fichero
  // "letra_a.txt" si el fichero existe lo sobreescribe
  if((f=fopen("letra_a.txt", "w"))==NULL)
  { printf("Error: No se pudo crear el fichero <letra_A.txt>");
    exit(-1);}
  fputc('A', f); //Escribe el caracter 'A'
  fputc('\n', f); //escribe el '\n'
  fclose(f);
  return(0);
}
```

muestra1.c

Lectura carácter a carácter

Ejemplos. fgetc(), fputc()

```
#include <stdio.h>
int main()
{ FILE* f;
  int c;
  char nombreFich[] = "fichtexto.txt";
  if((f=fopen(nombreFich, "r"))==NULL)
  { printf("Error: No se pudo abrir el fichero <%s>", nombreFich);
    exit(-1); }
  while((c=fgetc(f))!=EOF)
  {fputc(c, stdout);}
  fclose(f);
  return(0);
}
```

Lectura

muestra2.c

Copia carácter a carácter el contenido de un fichero

Ejemplos

```
void fileCopy(char* nombreOrigen, char* nombreDestino)
{
  FILE* fOrigen, * fDestino;
  int c;

  fOrigen=fopen(nombreOrigen, "r");
  fDestino=fopen(nombreDestino, "w");

  while((c=fgetc(fOrigen))!=EOF)
  {
    fputc(c, fDestino);
  }

  fclose(fOrigen);
  fclose(fDestino);
}
```

muestra2.c

Ejemplos

```
int main()
{
  char nombreOrigen[]="fichtexto.txt";
  char nombreDestino[]="copiaFichero.txt";

  if(existeFichero(nombreOrigen))
  {
    fileCopy(nombreOrigen, nombreDestino);
  }
  else
  {
    printf("\nEl fichero <%s> no existe", nombreOrigen);
    return(1);
  }
  return(0);
}
```

dobleEspaciado.c

Ejemplos

Formatea a doble espaciado un fichero

```
int doblaEspacios(char* nombreOrigen, char* nombreDestino)
{
FILE* fOrigen, * fDestino;
  int c;
  if((fOrigen=(FILE*)fopen(nombreOrigen, "r"))!=NULL)
  {        if((fDestino=(FILE*)fopen(nombreDestino, "w"))!=NULL)
    {
      while((c=fgetc(fOrigen))!=EOF)
      {
         fputc(c, fDestino);
         if(c=='\n')
         { //si es '\n' se escribe el '\n' otra vez
            fputc(c, fDestino);
         }
      }
      fclose(fOrigen);
      fclose(fDestino);
      return(1);
    }
  }
  else return 0;
}
```

Lectura

Escritura

Ejemplos

```
int main()
{
  FILE* fOrigen, *fDestino;
  char nombreOrigen[]="fichtexto.txt";
  char nombreDestino[]="destino.txt";

  if (!doblaEspacios(nombreOrigen, nombreDestino))
  {
      printf("\nNo se ha podido realizar la operacion");
  }

  return(0);
}
```

Otras funciones para ficheros de texto y binarios

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

- Borra el fichero llamado nombre
  - Devuelve un valor distinto de cero en caso de error

```
int rename (const char* viejo, const char* nuevo);
```

- Cambia el nombre del fichero llamado `viejo` por el de `nuevo`
  - Devuelve un valor distinto de cero en caso de error

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

- Fuerza a que el fichero `f` se libere: se vacía el *buffer* asociado al fichero de salida. El efecto está indefinido para ficheros de entrada. Devuelve `EOF` si hay algún error de escritura y cero en otro caso
  - `fflush(NULL)` libera todos los ficheros de salida
  - `fflush(stdout)` libera el *buffer* del dispositivo de salida
  - **`fflush(stdin)` su efecto está indefinido → NO UTILIZARLO**

renameRemove.c

Ejemplo remove y rename

```
int main()
{
  char nombreFichero[30], nombreNuevo[30];
  int opcion;
  printf("\nQue desea hacer?:");
  printf("\n1 Borrar fichero");
  printf("\n2 Renombrar fichero");
  scanf("%d", &opcion);
  switch(opcion)
  { case 1:
          printf("\nIntroduzca el nombre del fichero a borrar: ");
          scanf("%s", nombreFichero);
          if(remove(nombreFichero)!=0)
             printf("\n\tNO se ha podido realizar la operacion");
          else
             printf("\nOperacion realizada con exito");
          break;
     case 2:
          printf("\nIntroduzca el nombre del fichero a renombrar: ");
          scanf("%s", nombreFichero);
          printf("\nIntroduzca el nombre del fichero a renombrar: ");
          scanf("%s", nombreNuevo);
          if(rename(nombreFichero, nombreNuevo)!=0)
             printf("\n\tNO se ha podido realizar la operacion");
          else
             printf("\nOperacion realizada con exito");
          break;
  }
  return(1);
}
```

Ejemplo de tmpnam

`char* tmpnam(char* s)`

- Genera una cadena de caracteres, *s*, que es un nombre válido de fichero y no coincide con el nombre de un fichero existente. La cadena *s* debe estar previamente reservada. Genera una cadena diferente cada vez que se llama

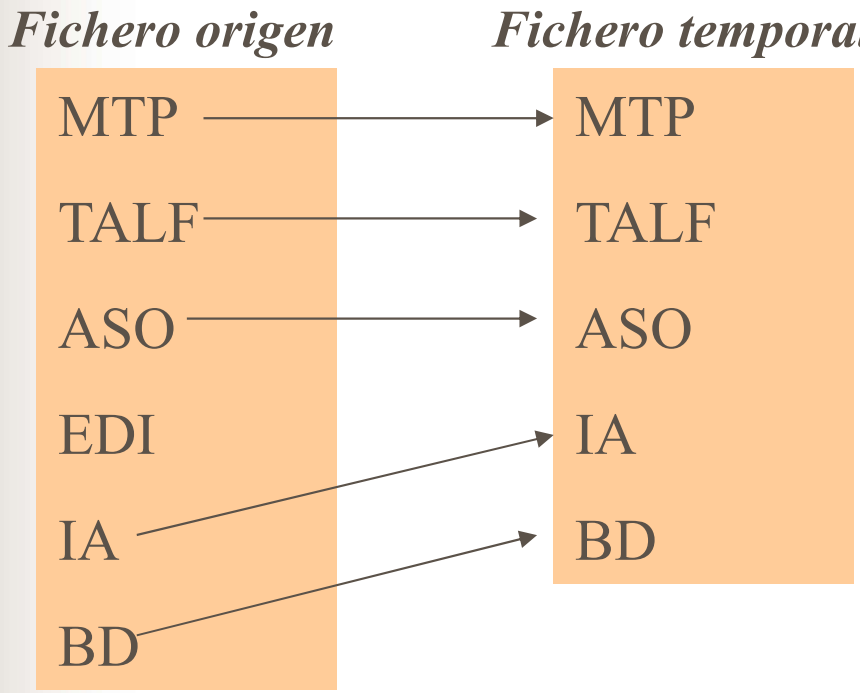
```
#include <stdio.h>
int main()
{
   //char  name[100];
   //mejor usar L_tmpnam para evitar salirnos
   //de la memoria reservada
   char  name[L_tmpnam];

   tmpnam(name);
   printf("Nombre generado: <%s>\n", name);
   return 0;
}
```

Ampliación. Ejemplo de fflush()

```
#include <stdio.h>

int main()
{
  char acumulador[BUFSIZ]; //Definido en stdio.h
  setbuf( stdout, acumulador );

  printf( "Esto es una prueba\n" );
  printf( "Este mensaje se mostrara a la vez\n" );
  printf( "setbuf, acumula los datos en un puntero\n" );
  printf( "hasta que se llene completamente\n" );
  getchar();
  fflush( stdout );

  return 0;
}
```

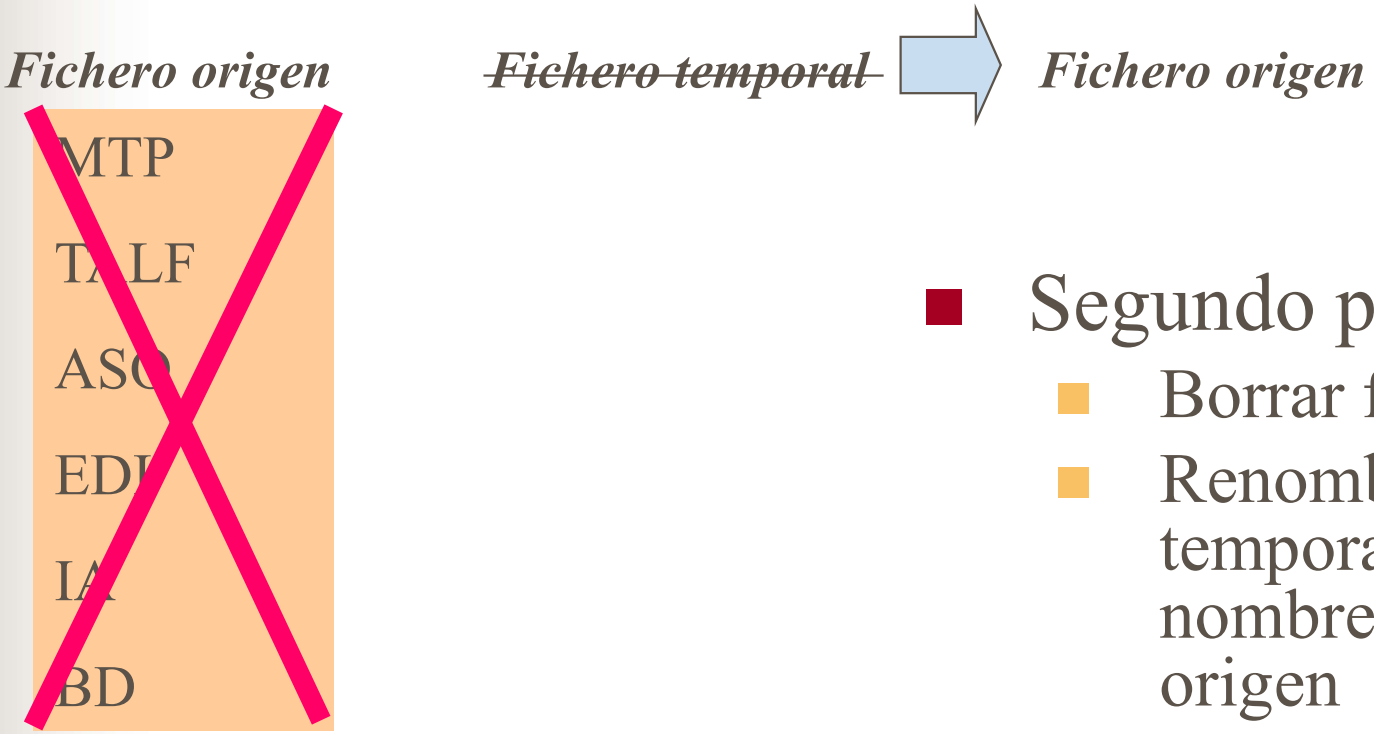
Borrado en ficheros de texto

- Son necesarios dos ficheros.
  - Ejemplo, borrar EDI



- Primer paso.
  - Copiar en el fichero temporal aquello que NO se va a borrar
  - Cerrar los dos ficheros

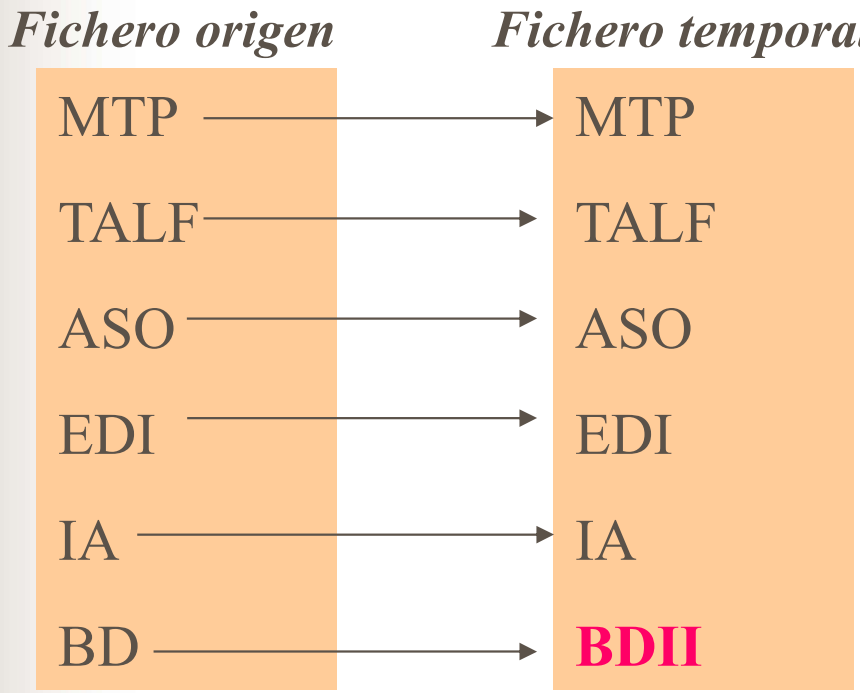
Borrado en ficheros de texto



- Segundo paso:
  - Borrar fichero origen
  - Renombrar fichero temporal con el nombre del fichero origen

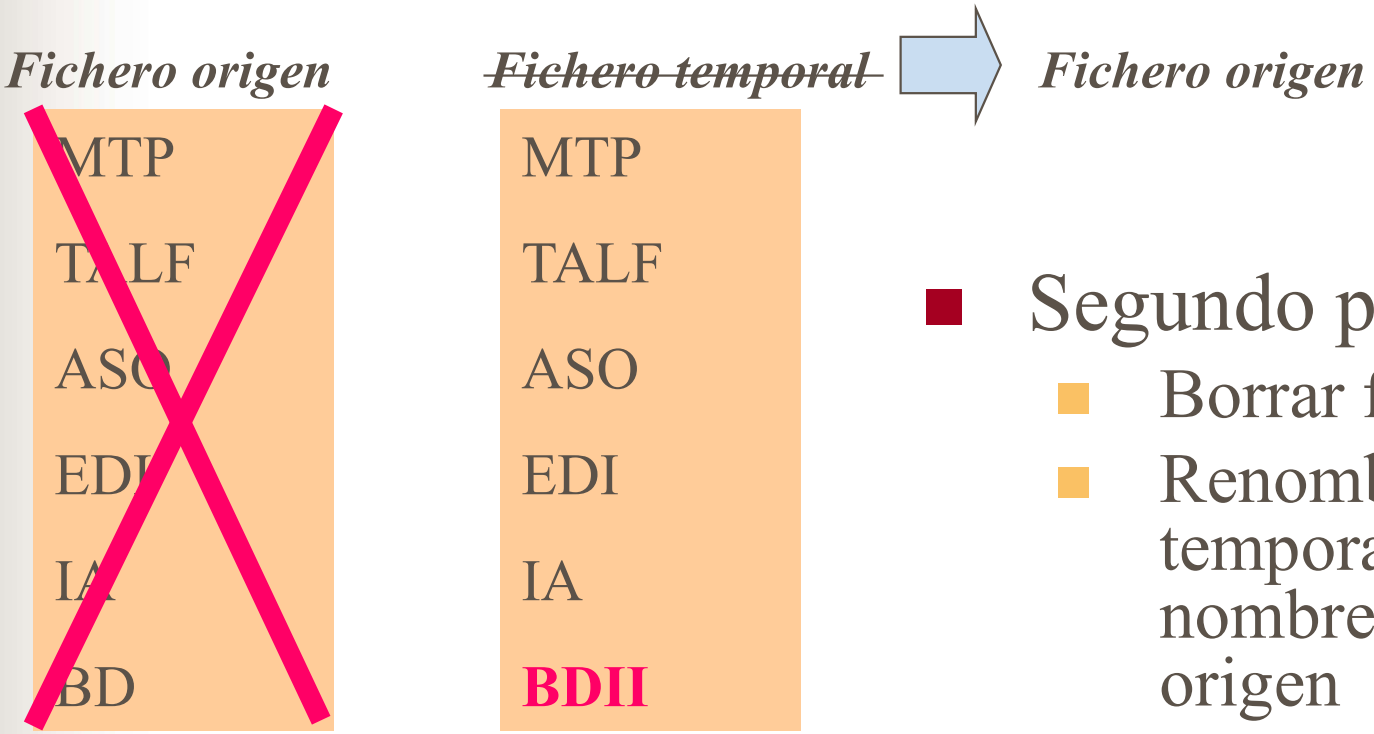
Actualización en ficheros de texto

- Son necesarios dos ficheros.
  - Ejemplo, actualizar BD a BDII



- Primer paso.
  - Copiar en el fichero temporal la información actualizada
  - Cerrar los dos ficheros

Actualización en ficheros de texto



- Segundo paso:
  - Borrar fichero origen
  - Renombrar fichero temporal con el nombre del fichero origen

Resumen

Ficheros de texto y binarios	Apertura	fopen
	Cierre	fclose
	Otras	remove, rename, ~~fflush~~, ~~feof~~, tmpnam
Ficheros de texto	L/E de caracteres	fgetc, ~~getc~~, fputc, ~~putc~~
	L/E Líneas	fgets, fputs
	L/E Formato	fscanf, fprintf
Ficheros binarios	L/E	fread, fwrite

Resumen

- Lectura de un fichero. Bucle while(<condición>) utilizando como condición los códigos de error devueltos por las funciones de lectura

```
//Carácter a carácter
int c;
while((c=fgetc(...))!=EOF)
{
    ...
}
```

```
//Línea a línea
while(fgets(...)!=NULL)
{
    ...
}
```

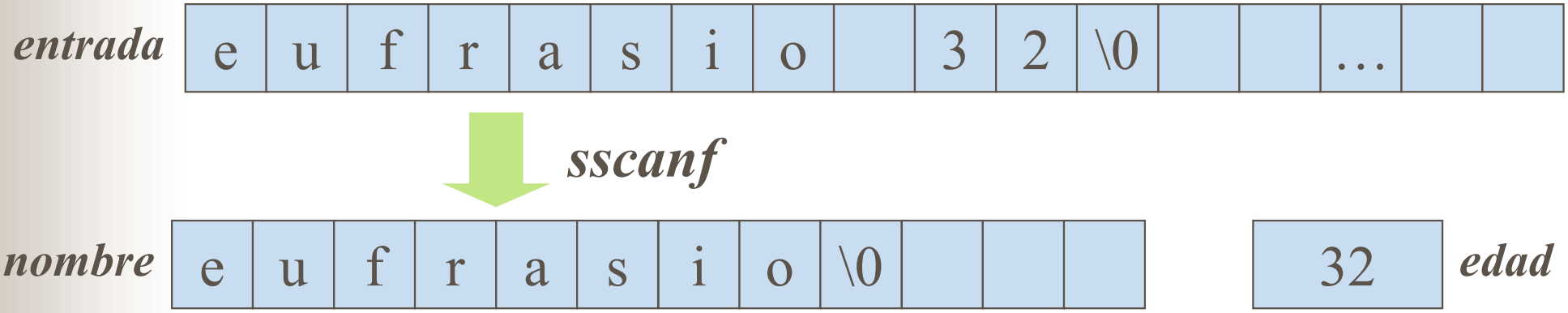
```
//Con formato
while(fscanf(...)==4)
//while(fscanf(...)!=EOF)
{
    ...
}
```

Conversión número ↔ cadena

- **Número → cadena**
  - `int sprintf(char *cadena, const char *formato, ...);`
    - Equivalente a *printf*, excepto que el argumento cadena especifica un *array* en el cual la salida generada es para ser escrita
    - Retorna el número de caracteres escritos al array, sin contar el carácter nulo al final
    - Escribe un carácter nulo al final de los caracteres escritos; no es contado como parte de la suma retornada
- **Cadena → número**
  - `int sscanf(const char *cadena, const char *formato,...);`
    - Equivalente a *scanf* excepto que el argumento cadena especifica un *array* desde el cual la entrada es obtenida
    - Devuelve el número de datos de entrada asignados

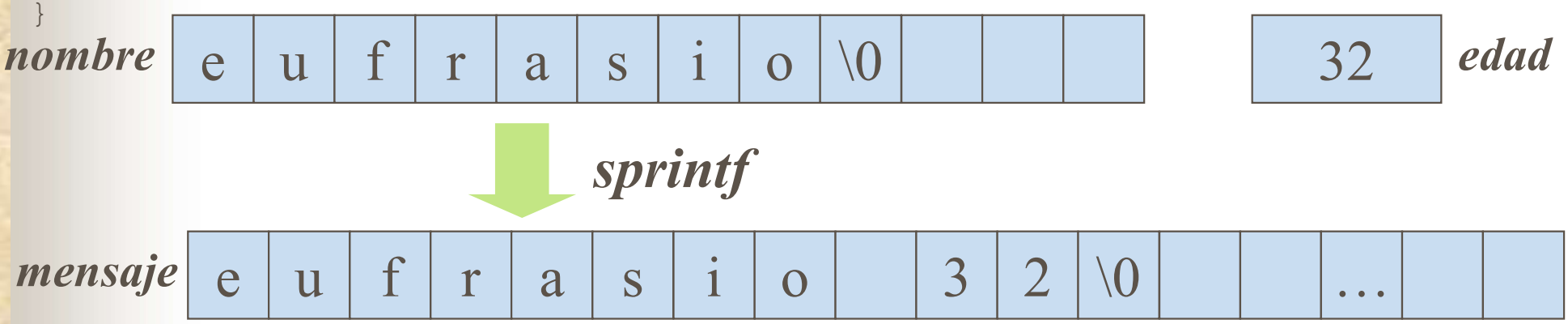
Conversión número ↔ cadena

```
#include <stdio.h>
int main()
{   char nombre[20]="", entrada[81]="";
    int edad=0;

    printf( "Escriba su nombre y edad, separados por un espacio:\n" );
    gets( entrada );
    sscanf( entrada, "%s %u", nombre, &edad );
    printf( "Has escrito: <%s>\n", entrada );
    printf( "Nombre: %s. Edad: %d\n", nombre, edad );
    return 0;
}
```



Conversión número ↔ cadena

```
#include <stdio.h>
int main()
{
   char nombre[20], mensaje[81];
   int edad=0;
   printf( "Escriba su nombre: " );
   scanf( "%s", nombre );
   printf( "Escriba su edad: " );
   scanf( "%d", &edad );
   sprintf( mensaje, "\nHola <%s>. Tu edad es <%d>.\n", nombre, edad );
   puts( mensaje );
   return 0;
}
```



Conversión número ↔ cadena

- **Cadena → número**
  - `int atoi(const char *numPtr);`
    - Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **int**. Devuelve el valor convertido.
    - Equivale a `sscanf(numPtr, "%d", & numero);`
  - `long int atol(const char *numPtr);`
    - Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **long**. Devuelve el valor convertido.
    - Equivale a `sscanf(numPtr, "%ld", & numero);`
  - `double atof(const char *numPtr);`
    - Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **double**. Devuelve el valor convertido.
    - Equivale a `sscanf(numPtr, "%lf", & numero);`

Conversión número ↔ cadena

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

int main()
{
   char flotante[11] = "123.456789";
   char entero[5] = "1234";
   char enteroLargo[11] = "1234567890";

   printf( "Atof: Convirtiendo la cadena <%s> en un numero: <%lf>\n", flotante,
     atof(flotante) );
   printf( "Atol: Convirtiendo la cadena <%s> en un numero: <%u>\n",
     enteroLargo, atol(enteroLargo) );
   printf( "Atoi: Convirtiendo la cadena <%s> en un numero: <%d>\n", entero,
     atoi(entero) );

   return 0;
}
```