UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Objetivo: Comprender los algoritmos y las operaciones relacionadas con el uso de archivos de texto y binarios.

## Introducción

El propósito de las siguientes secciones de centrará en la operación y explotación de datos almacenados en archivos de texto y binarios. Suponemos que quien lea este apunte, ha tenido contacto con la Parte I, ya que los conceptos ahí expresados serán utilizados a lo largo de todo este apunte.

## Funciones de biblioteca útiles para el tratamiento de archivos - Parte I

Las siguientes funciones resultan útiles para la generación de cadenas de texto a partir de variables y viceversa. Esas funciones son **`sprintf`** y **`sscanf`**. Las Figuras 1 y 2 ejemplifican su uso.

```
3   void probar_sprintf(void)
4   {
5       char cadena[80];
6       tEmpleado emp;
7
8       emp.dni = 12345678;
9       strcpy(emp.apyn, "PEREZ, Juan");
10      emp.categoria = 'A';
11      emp.fecIngreso.dia = 1;
12      emp.fecIngreso.mes = 2;
13      emp.fecIngreso.anio = 1999;
14      emp.sueldo = 65535.65;
15
16      sprintf(cadena,"%08ld %-15s %c %02d/%02d/%04d %9.2f",
17                  emp.dni,
18                  emp.apyn,
19                  emp.categoria,
20                  emp.fecIngreso.dia,
21                  emp.fecIngreso.mes,
22                  emp.fecIngreso.anio,
23                  emp.sueldo);
24
25      printf("Cadena generada por sprintf: %s\n", cadena);
26  }
```

*Figura 1. Ejemplo de uso de la función de biblioteca **`sprintf`**.*

En el ejemplo de la Figura 2 explique qué es lo que sucede cuando se utiliza la cadena de texto de la línea 31 en lugar de la cadena de la línea 30.

```
28  void probar_sscanf(void)
29  {
30      char cadena[] = {"12345678 PEREZ,Juan A 01/02/1999 65535.65"};
31      /**char cadena[] = {"12345678 PEREZ, Juan A 01/02/1999 65535.65"};*/
32
33      tEmpleado emp;
34      sscanf(cadena,"%ld %15s %c %d/%d/%d %f",
35                    &emp.dni,
36                    emp.apyn,
37                    &emp.categoria,
38                    &emp.fecIngreso.dia,
39                    &emp.fecIngreso.mes,
40                    &emp.fecIngreso.anio,
41                    &emp.sueldo);
42
43      mostrarEmpleado(NULL);
44      mostrarEmpleado(&emp);
45  }
```

*Figura 2. Ejemplo de uso de la función de biblioteca **sscanf**.*

En el apunte anterior hicimos uso de la función **fprintf** para hacer escrituras formateadas sobre archivos de texto. La función **fscanf** permite realizar la operación contraria, es decir, realizar la lectura formateada desde un archivo de texto. Si bien mencionamos su existencia, no la utilizaremos para la recuperación de registros desde archivos de texto.

**Ejercicio propuesto:** A partir de la documentación de C analice el prototipo y el funcionamiento de las funciones **sprintf**, **sscanf** complementando los ejemplos de las Figuras 1 y 2.

### Apertura de archivos

Ya hemos visto como realizar la apertura de archivos binarios y de texto. El procedimiento se realizaba con a función de **fopen**, a la que se le indicaba el nombre del archivo y su modo de apertura. Luego era necesario verificar que el archivo se había logrado abrir, verificando que el puntero a **FILE** retornado por **fopen** no haya sido **NULL**. Vamos a crear una función que resuelva ésta operación. La función tendrá el siguiente prototipo:

```
int abrirArchivo(FILE **fp,
                 const char *nombreArchivo,
                 const char *modoApertura,
                 int mostrarError);
```

En donde:

**`fp:`** puntero a puntero de tipo FILE.
**`nombreArchivo:`** nombre del archivo que se desea abrir.
**`modoApertura:`** modo de apertura del archivo.
**`mostrarError:`** flag para indicarle a la función si debe o no mostrar por la salida estándar de error que ha ocurrido un error.

La función retorna 1 si la apertura ha sido exitosa. En caso contrario retorna 0. La Figura 3 muestra el código de la función. La Figura 4 muestra cómo se realiza su invocación.

```
4   int abrirArchivo(FILE ** fp,
5                    const char *nombreArchivo,
6                    const char *modoApertura,
7                    int mostrarError)
8   {
9       *fp = fopen(nombreArchivo, modoApertura);
10
11      if(*fp == NULL)
12      {
13          if(mostrarError == 1)
14          {
15              fprintf(stderr,
16                      "Error abriendo \"%s\" en modo \"%s\".",
17                      nombreArchivo,
18                      modoApertura);
19          }
20          return 0;
21      }
22      return 1;
23  }
```

*Figura 3. Función que realiza la apertura de un archivo.*

```
1   #include "main.h"
2
3   int main()
4   {
5       FILE *fp;
6
7       if(!abrirArchivo(&fp, "prueba.txt", "wt", 1))
8       {
9           exit(1);
10      }
11
12      /**
13          Hacer otras cosas.
14      */
15
16      fclose(fp);
17      return 0;
18  }
```

*Figura 4. Invocación a la función que realiza la apertura de un archivo.*

Note que la función **`abrirArchivo`** recibe uno de sus parámetros como puntero a puntero del tipo **`FILE`**.

**Ejercicio propuesto:** Explique el motivo del uso del doble puntero a **`FILE`** utilizado en la función **`abrirArchivo`**.

**Tratamiento de archivos de texto**

En el primer apunte vimos como generar registros (filas) en archivos de texto a partir de estructuras de datos binarias. Puntualmente, en los ejemplos utilizamos el tipo de datos **`tEmpleado`**.

Lo que deseamos ahora es efectuar el proceso inverso que en este caso será transformar una línea de texto proveniente de un archivo de texto para transformarla en una estructura binaria que permita ser tratada en un programa. A esta acción la denominaremos trozado de campos. En nuestro caso debemos abordar los dos tipos de archivos de texto que hemos estudiado: archivos con campos de longitud fija y archivos con campos de longitud variable. De estos dos escenarios se derivan dos estrategias:

- Trozado de líneas de registros de longitud variable.
- Trozado de líneas de registros de longitud fija.

En los siguientes apartados analizaremos la estrategia para cada uno de los casos.

**Trozado de campos de longitud variable**

Recordemos que en este formato para organizar datos en un archivo de texto utilizábamos un carácter de separación entre campos destinado a esa función que se había seleccionado en tiempo de diseño y además, la longitud de los campos pueden variar. Veamos nuevamente el ejemplo que utilizamos en la parte 1 del apunte:

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
44444444|Persona Cuatro|A|1/4/2004|44000.44
22222222|Persona Dos|B|1/2/2002|22000.25
33333333|Persona Tres|B|1/3/2003|33000.32
55555555|Persona Cinco|A|1/5/2005|55000.50
1111111|Persona Uno|C|1/1/2001|111000.10
```

En este caso, el carácter de separación entre campos el pipe ( `'|'` ). Se debe tener en cuenta que cada línea del archivo finaliza con el carácter de fin de línea ( `'\n'` ). Sin embargo, cuando la línea sea leída por la función **`fgets`**, ésta le agregará el carácter de fin de cadena ( `'\0'` ).

Recordemos que el objetivo es llevar cada campo contenido en una línea del archivo a una estructura binaria. La estrategia que utilizaremos para tratar la línea de texto leída desde el archivo se muestra en la Figura 5.

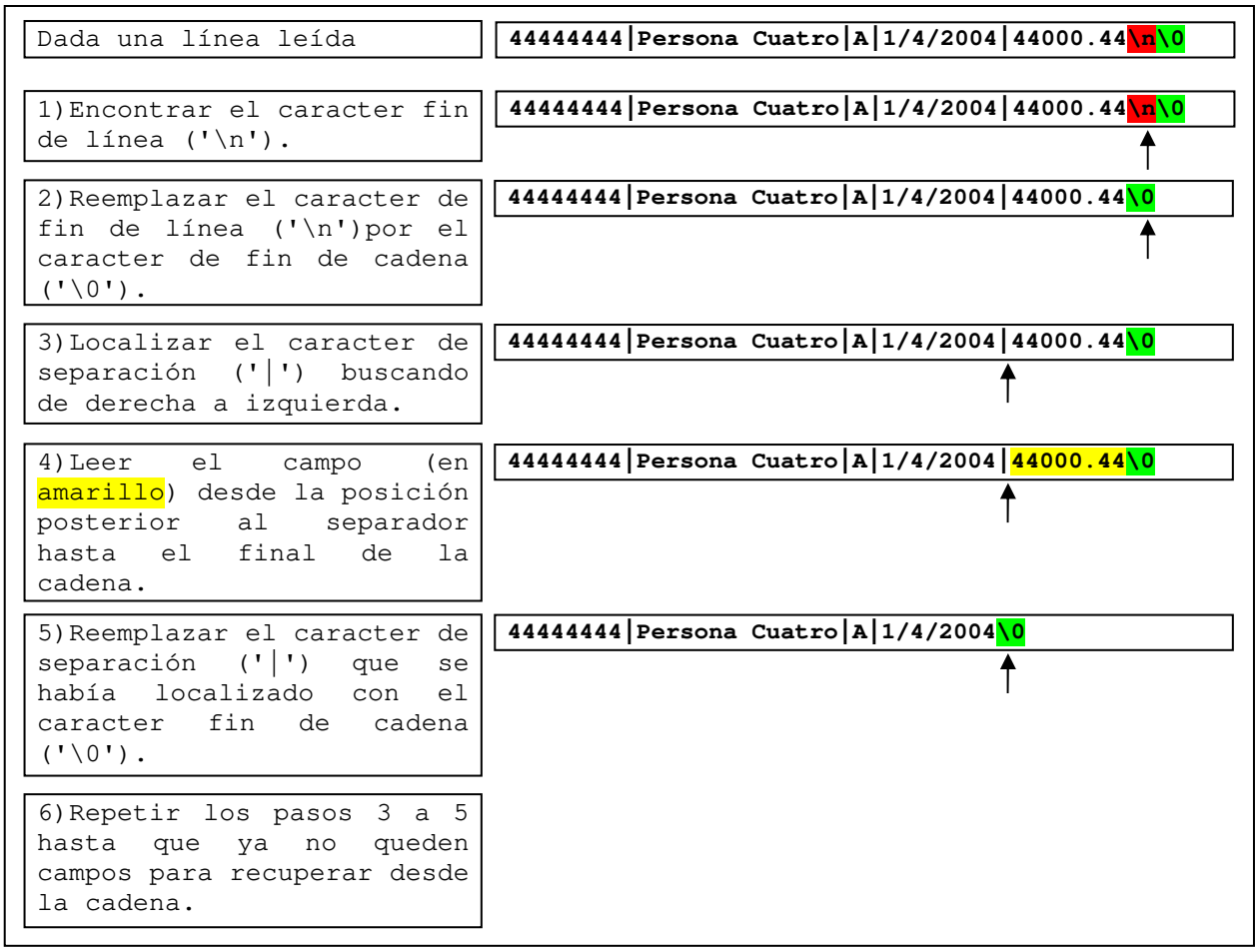



*Figura 5. Algoritmo para el trozado de campos de una línea de texto proveniente de un archivo de texto con campos de longitud variable.*

Como se puede ver, la cadena es tratada desde el final hacia el inicio.

La Figura 6 muestra la implementación de la función. Es importante destacar el uso de la función **`strrchr`**. Otro punto a destacar es que se debe "esquivar" el carácter de separación de campos. Para esta tarea se pueden elegir dos estrategias. La primera es incluir en la máscara de formato el carácter de separación. La segunda consiste en leer a partir de la posición posterior a la que corresponde al carácter de separación. Las líneas 10 y 12 del código de la Figura 6 ejemplifican estas dos situaciones.

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```
1  void trozarCamposLongitudVariable(tEmpleado *d, char *s)
2  {
3      char *aux = strchr(s, '\n');
4      *aux = '\0';
5
6      /** Sueldo*/
7      aux = strrchr(s, ';');
8      /**
9          Utilizar una de las dos opciones
10         sscanf(aux, "%|f", &d->sueldo);
11     */
12     sscanf(aux + 1, "%f", &d->sueldo);
13     *aux = '\0';
14
15     /** Fecha de Ingreso */
16     aux = strrchr(s, ';');
17     sscanf(aux + 1,
18            "%d/%d/%d",
19            &d->fecIngreso.dia,
20            &d->fecIngreso.mes,
21            &d->fecIngreso.anio);
22     *aux = '\0';
23
24     /** Categoria */
25     aux = strrchr(s, ';');
26     sscanf(aux + 1, "%c", &d->categoria);
27     *aux = '\0';
28
29     /** Apellido y nombre */
30     aux = strrchr(s, ';');
31     strcpy(d->apyn, aux + 1);
32     *aux = '\0';
33
34     /** DNI */
35     sscanf(s, "%ld", &d->dni);
36 }
```

*Figura 6. Implementación de una función para el trozado de campos de una línea de texto proveniente de un archivo de texto con campos de longitud variable.*

**Ejercicio propuesto:** Investigue el funcionamiento de la función **`strrchr`** de la biblioteca **`<string.h>`**.

### Trozado de campos de longitud fija

Repasemos el formato utilizado para campos de longitud fija. Para ello veamos el ejemplo de la parte 1 del apunte:

```
44444444Persona Cuatro                    A01042004 44000.44
22222222Persona Dos                       B01022002 22000.25
33333333Persona Tres                      B01032003 33000.32
55555555Persona Cinco                     A01052005 55000.50
01111111Persona Uno                       C01012001111000.10
```

Recordemos que cada campo del registro tendrá un tamaño específico y se debe respetar siempre, por ese motivo en caso de que el dato quepa con espacio de sobra en el campo se debe rellenar con caracteres específicos para cada caso. Por ejemplo, el DNI 1111111 se completa con un carácter 0 al inicio para que el campo tenga 8 caracteres.

Para poder tratar de manera correcta un registro, es necesario conocer la longitud cada uno de los campos que lo componen. Recordemos la definición del tipo de dato **`tEmpleado`** y la máscara de formato de la función **`fprintf`** de la función **`crearArchivoTextoLF`** de la parte 1 del apunte:

```c
typedef struct
{
    int dia,
        mes,
        anio;
}tFecha;

typedef struct
{
    long dni;
    char apyn[36];
    char categoria;
    tFecha fecIngreso;
    float sueldo;
}tEmpleadd;
```

```c
fprintf(fpTxtLF,
        "%08ld%-*.*s%c%02d%02d%04d%9.2f\n",
        emp.dni,
        sizeof(emp.apyn)-1,
        sizeof(emp.apyn)-1,
        emp.apyn,
        emp.categoria,
        emp.fecIngreso.dia,
        emp.fecIngreso.mes,
        emp.fecIngreso.anio,
        emp.sueldo);
```

La cantidad de caracteres de cada campo será:

**`dni: 8, apyn: 35, categoria: 1, fecIngreso: 8, sueldo: 9`**. El total es 61.

Con esta información podemos ver en la Figura 7 la estrategia para este escenario.

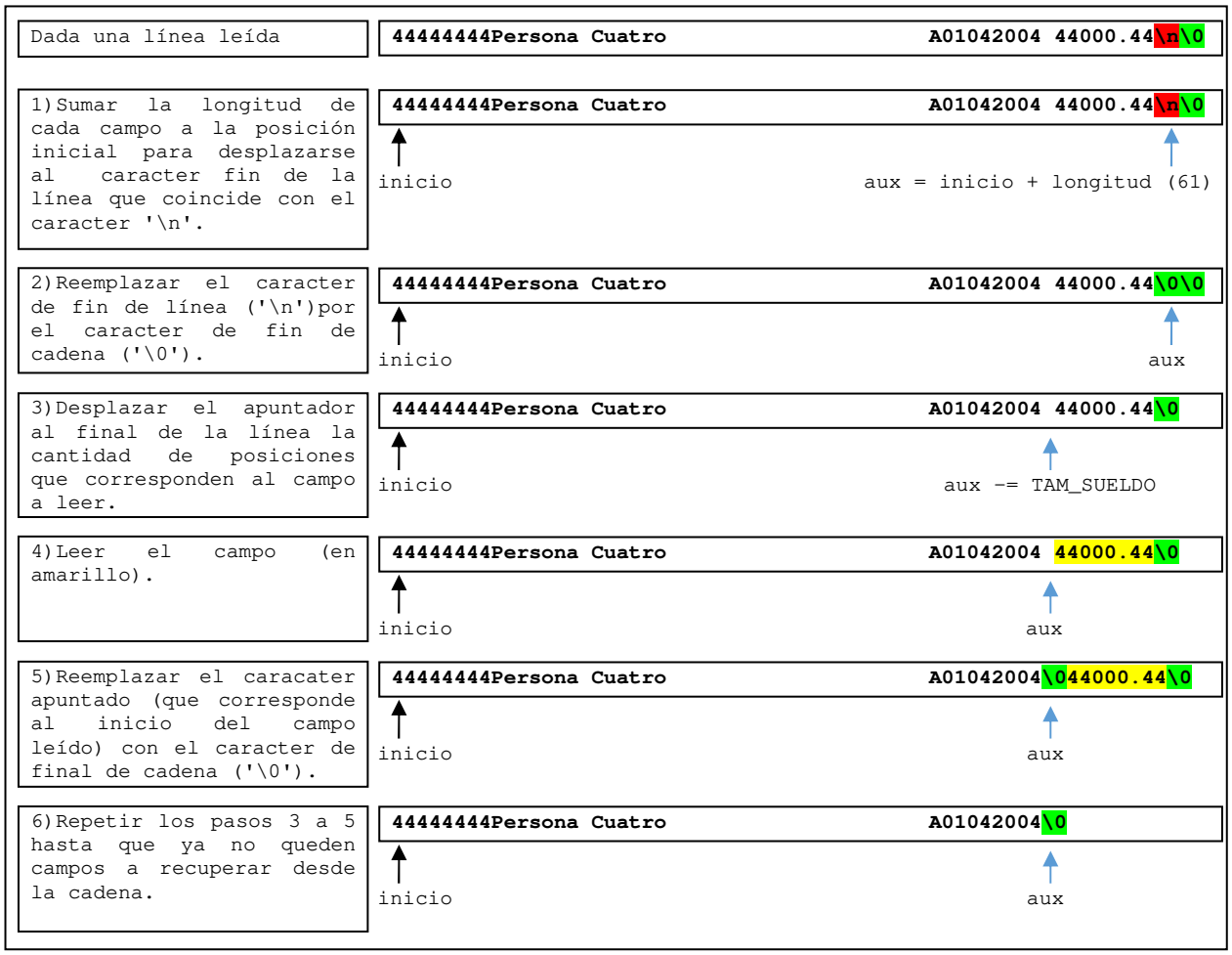



*Figura 7. Algoritmo para el trozado de campos de longitud fija.*

Nuevamente se puede ver que la cadena es tratada desde el final hacia el inicio de la línea. La Figura 8 muestra la implementación de la función.

```
1  void trozarCamposLongitudFija(tEmpleado *d, char *s)
2  {
3      char *aux = s + TAM_LINEA;
4      *aux = '\0';
5
6      /** Sueldo*/
7      aux -= TAM_SUE;
8      sscanf(aux, "%f", &d->sueldo);
9      *aux = '\0';
10
11     /** Fecha de Ingreso */
12     aux -= TAM_FEC_INGRESO;
13     sscanf(aux,
14            "%2d%2d%4d",
15            &d->fecIngreso.dia,
16            &d->fecIngreso.mes,
17            &d->fecIngreso.anio);
18     *aux = '\0';
19
20     /** Categoria */
21     aux -= TAM_CAT;
22     sscanf(aux, "%c", &d->categoria);
23     *aux = '\0';
24
25     /** Apellido y nombre */
26     aux -= TAM_APYN;
27     strcpy(d->apyn, aux);
28     *aux = '\0';
29
30     /** DNI */
31     aux -= TAM_DNI;
32     sscanf(aux, "%ld", &d->dni);
33     *aux = '\0';
34 }
```

*Figura 8. Implementación de una función para el trozado de campos de una línea de texto proveniente de un archivo de texto con campos de longitud fija.*

### Ejemplo de lectura de un archivo de texto con campos de longitud fija

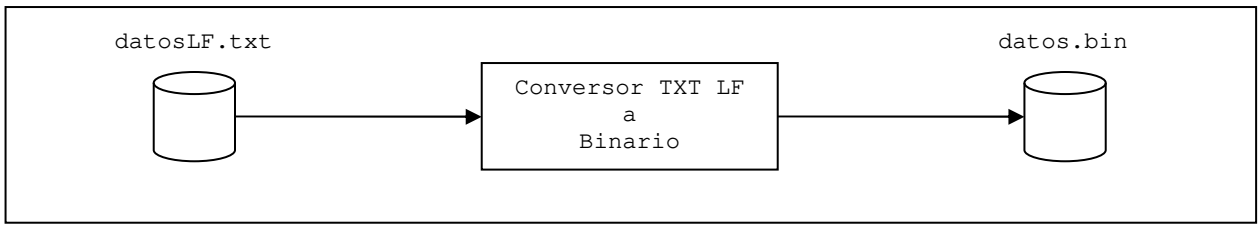
La función indicada en la Figura 9 muestra el código obtener los registros contenidos en un archivo de campos de longitud fija. En el ejemplo se hace uso de las funciones **`abrirArchivo`** y **`trozarCamposLongitudFija`**.

```
194  int leerYmostrarArchivoDeTextoLF(char *nombreArchivo)
195  {
196      char cad[100];
197      tEmpleado emp;
198      FILE *fp;
199
200      if(!abrirArchivo(&fp, nombreArchivo, "rt", 1))
201      {
202          return -1;
203      }
204
205      mostrarEmpleado(NULL);
206      while(fgets(cad, sizeof(cad), fp))
207      {
208          trozarCamposLongitudFija(&emp, cad);
209          mostrarEmpleado(&emp);
210      }
211
212      fclose(fp);
213      return 0;
214  }
```

*Figura 9. Función para mostrar los registros de un archivo de texto de longitud fija.*

**Ejercicio propuesto:** Modifique el programa de la Figura 9 para que permita mostrar los registros contenidos en un archivo con campos de longitud variable.

**Ejercicio propuesto:** Diseñe y codifique un programa que a partir de un archivo de texto con campos de longitud fija genere un archivo binario según el esquema siguiente:



### Funciones de biblioteca útiles para el tratamiento de archivos - Parte II

En esta sección revisaremos las funciones de biblioteca **`ftell`**, **`rewind`** y **`fseek`**. Estas funciones operan sobre la estructura apuntada por un puntero de tipo **`FILE`** de un flujo correctamente asociado a un archivo. Permiten modificar y obtener información acerca de la posición que se está apuntando sobre el archivo.

**Función `ftell`:** retorna el valor del indicador de posición del archivo para el flujo pasado como parámetro. Su prototipo es:

```
long int ftell(FILE *fp);
```

En donde:

**`fp`:** puntero al flujo.

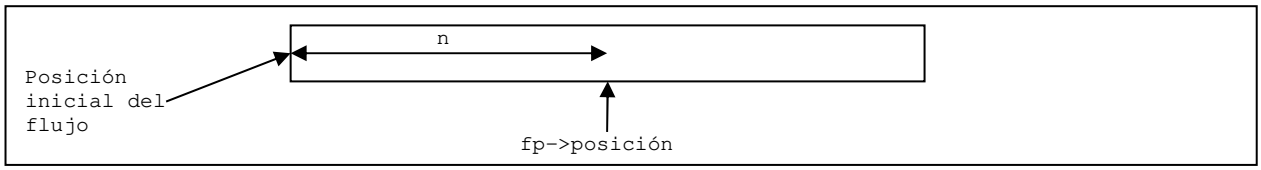



*Figura 10. Funcionamiento de la función **`ftell`**. Retorna la cantidad de bytes (n) desde el inicio a la posición actual.*

En la Figura 10 se muestra el valor retornado por la función (n en la figura).

**Función `rewind`:** lleva al inicio el indicador de posición del archivo para el flujo pasado como parámetro. Su prototipo es:

```
void rewind(FILE *fp);
```

En donde:

**`fp`:** puntero al flujo.

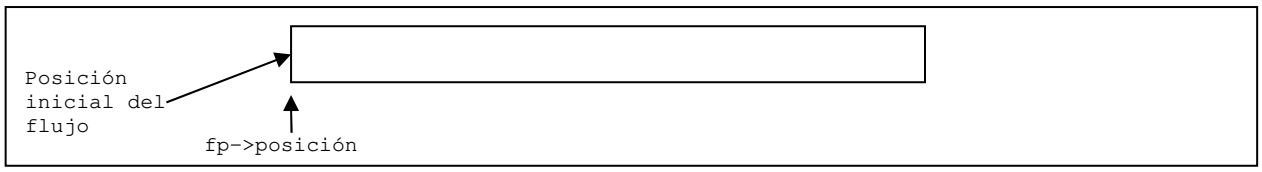



*Figura 11. Funcionamiento de la función **`rewind`**. Sitúa el indicador de posición al inicio del flujo.*

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

**Función `fseek`:** esta función permite el desplazamiento sobre el archivo. A través de sus argumentos, la función modifica el indicador de posición del puntero al flujo. La consecuencia de ello es que permite localizarse en cualquier posición dentro del archivo. Esto se denomina **acceso directo o aleatorio** y es sumamente útil al momento de realizar modificaciones sobre archivos binarios (posteriormente trataremos este tema). El prototipo de la función es el siguiente:

```
int fseek(FILE *fp, long int desplazamiento, int origen);
```

En donde:

**`fp`:** puntero al flujo.
**`desplazamiento`:** cantidad de bytes que se desea desplazar el indicador de posición.
**`origen`:** lugar de referencia desde donde se va a realizar el desplazamiento, inicio, posición actual o final del archivo. Los casos para este argumento de la función están descriptos en los macro reemplazos **`SEEK_SET`**, **`SEEK_CUR`**, **`SEEK_END`**.

La función retorna un valor distinto de 0 si no ha podido cumplir con la acción solicitada.

**Actualización de registros en archivos binarios**

Trataremos la actualización de registros en archivos binarios. El objetivo será modificar los datos almacenados en el archivo teniendo en cuenta algún criterio definido mediante alguna regla. Para realizar esta tarea necesitaremos hacer uso de la función fseek, pero es muy importante notar que para las operaciones de actualización el archivo debe ser abierto de la manera adecuada. En la Tabla 2 de la Parte 1 del apunte se listaban los modos para poder realizar el acceso directo.

Vamos a plantear el esquema que comprende la actualización. Imaginemos que se desea modificar de manera masiva un determinado campo (por ejemplo, para el tipo de dato tEmpleado el sueldo) de los registros almacenados. Al decir en forma masiva, entendemos que se modificarán todos los registros del archivo. Esto no necesariamente debe ser siempre así, ya que en otros casos podría ser necesario modificar los registros que cumplan con una determinada condición (por ejemplo, en


tEmpleado solo los que posean categoría 'D'). Para ello, debemos recorrer todo el archivo. Se leerán los registros uno a uno mientras no se alcance el final del archivo, se harán la modificaciones correspondientes sobre el registro, y se escribirá nuevamente el registro modificado. Estos pasos se repetirán hasta que se encuentre el final del archivo.

Si recordamos el funcionamiento de las funciones **`fread`** y **`fwrite`**, cada vez que se ejecuta alguna de ellas, el indicador de posición en la estructura apuntada por el puntero de tipo **`FILE`** quedaba posicionado al inicio del siguiente registro (obviamente si no se ha alcanzado el final del archivo). Dijimos que debíamos leer un registro, modificarlo y luego volver a escribirlo, sin embargo, el indicador de posición de puntero ya se encuentra al inicio del siguiente registro, si se intenta escribir en realidad se estaría escribiendo el siguiente registro, lo que estaría MAL. Para resolver esta situación se recurre al acceso aleatorio utilizando la función **`fseek`**.

En la Figura 12 se plantea el esquema utilizado para la actualización. Se supone que es un archivo binario que contiene 5 registros de tipo tDato y tamaño **n** bytes y se quiere actualizar un determinado campo.

La Figura 13 muestra un ejemplo de una función de actualización de un archivo binario. La función recibe un puntero a **`FILE`** y se desea que al retornar de la función el indicador de posición se encuentre en el mismo lugar que se encontraba antes de la actualización. Para resolver este requerimiento en la línea 6 se obtiene la posición actual. Luego, antes de retornar, en la línea 21 la función **`fseek`** se encarga de restablecer el indicador a la posición inicial. Las líneas 16, 17 y 18 contienen las siguientes sentencias:

```
16 fwrite(&emp, sizeof(tEmpleado), 1, fp);
17 fseek(fp, 0L, SEEK_CUR);
18 fread(&emp, sizeof(tEmpleado), 1, fp);
```

El objetivo de la línea 17 es restablecer el sentido del flujo en modo lectura. Esto se debe a que cuando se realiza una operación de lectura seguida de una de escritura (no debemos olvidarnos de la lectura de la línea 10), la operación de escritura cambia el sentido del flujo, por lo cual si se volviera a realizar una operación de lectura no funcionaría de manera correcta. Por ese motivo se coloca el **`fseek`**. Nótese que el desplazamiento es cero y no altera la posición del indicador de posición de la estructura apuntada por el puntero de tipo **`FILE`**, pero si cambia el sentido del flujo y restablece la lectura de manera correcta.

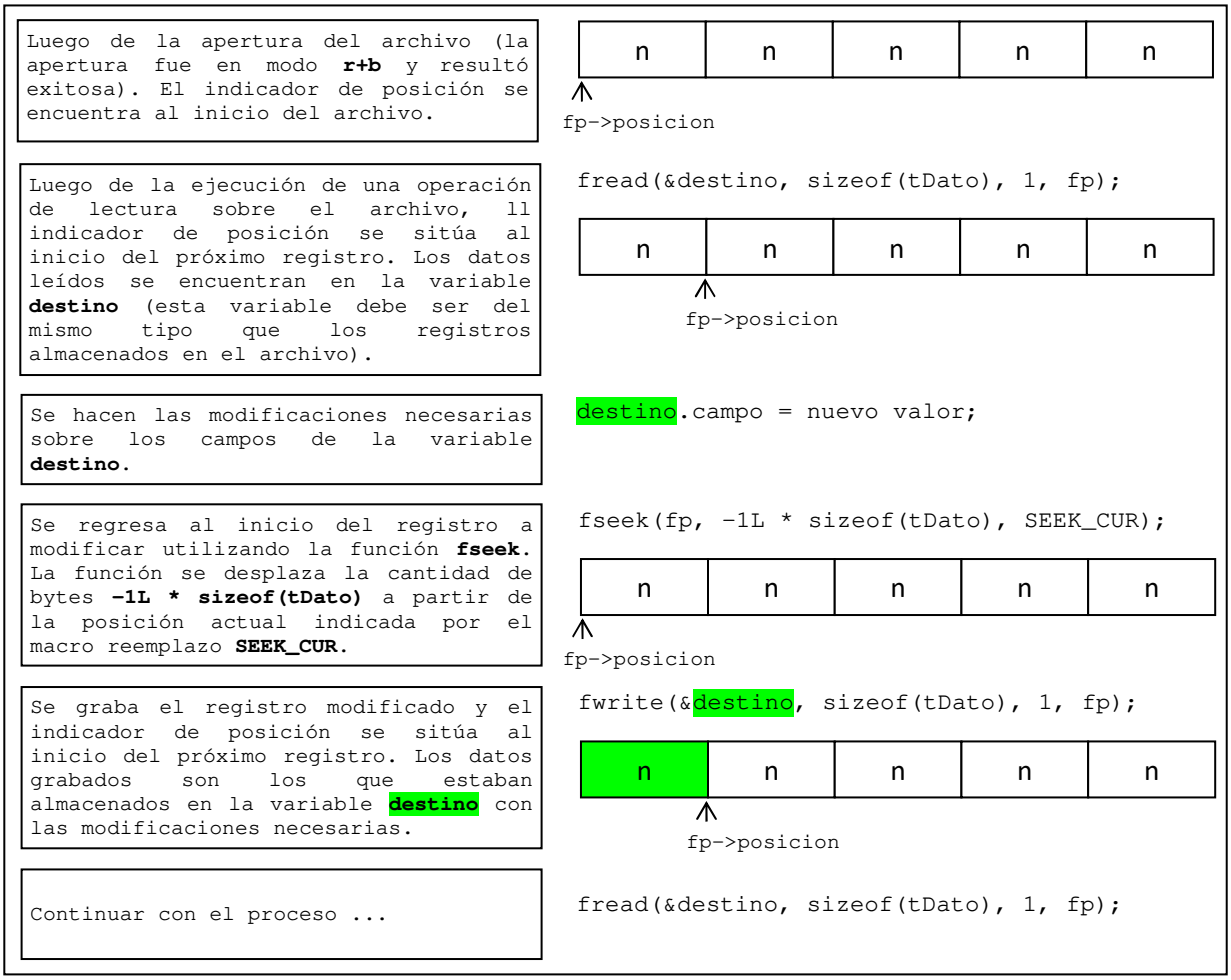



*Figura 12. Esquema de procedimiento para la actualización de un archivo binario.*

```c
#include "actualizacionBinarios.h"

void actualizarPorcentajeSueldo(FILE *fp, float porcentaje, int *nroRegistros)
{
    tEmpleado emp;
    long posInicial = ftell(fp);
    *nroRegistros = 0;
    rewind(fp);

    fread(&emp, sizeof(tEmpleado), 1, fp);
    while(!feof(fp))
    {
        (*nroRegistros)++;
        emp.sueldo *= porcentaje;
        fseek(fp, -1L * sizeof(tEmpleado), SEEK_CUR);
        fwrite(&emp, sizeof(tEmpleado), 1, fp);
        fseek(fp, 0L, SEEK_CUR);
        fread(&emp, sizeof(tEmpleado), 1, fp);
    }

    fseek(fp, posInicial, SEEK_SET);
}
```

*Figura 13. Función que actualiza un archivo binario.*

La Figura 14 muestra como se realiza el llamado a la funcion de actualizacion del archivo. La figura 15 muestra el estado del archivo antes y despues de la actualización.

```c
int main()
{
    FILE *fp;
    int nroRegistrosArchivo;
    leerYmostrarArchivo(NOMBRE_ARCHIVO_BIN);

    if(!abrirArchivo(&fp, NOMBRE_ARCHIVO_BIN, "r+b", 1))
    {
        exit(1);
    }
    actualizarPorcentajeSueldo(fp, 1.25, &nroRegistrosArchivo);
    fclose(fp);

    printf("\n\nCantidad de registros: %d\n\n", nroRegistrosArchivo);

    leerYmostrarArchivo(NOMBRE_ARCHIVO_BIN);
    return 0;
}
```

*Figura 14. Invocación a la función que actualiza un archivo binario.*

```
DNI      NOMBRE Y APELLIDO                    CAT FEC. ING   SUELDO
-------- ------------------------------------ --- ---------- ----------
44444444 Persona Cuatro                       A   01/04/2004   44000.44
22222222 Persona Dos                          B   01/02/2002   22000.25
33333333 Persona Tres                         B   01/03/2003   33000.32
55555555 Persona Cinco                        A   01/05/2005   55000.50
01111111 Persona Uno                          C   01/01/2001  111000.10


Cantidad de registros: 5

DNI      NOMBRE Y APELLIDO                    CAT FEC. ING   SUELDO
-------- ------------------------------------ --- ---------- ----------
44444444 Persona Cuatro                       A   01/04/2004   55000.55
22222222 Persona Dos                          B   01/02/2002   27500.31
33333333 Persona Tres                         B   01/03/2003   41250.40
55555555 Persona Cinco                        A   01/05/2005   68750.63
01111111 Persona Uno                          C   01/01/2001  138750.13

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

*Figura 15. Contenido del archivo binario antes y después de la actualización.*

**Ejercicio propuesto:** Modifique la función de la Figura 13 de manera tal que solo actualice a los empleados con categoría 'B' y que además de retornar la cantidad de registros que posee el archivo retorne la cantidad de registros modificados.

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

## Índice de figuras



## Bibliografía

[1] B. W. Kernighan y D. M. Ritchie, El lenguaje de programación C, Pearson Educación, 1991.

[2] H. M. Deitel y P. J. Deitel, Cómo programar en C/C+, Pearson Educación, 1995.

[3] Herbert Schildt, Turbo C/C++ 3.1 Manual de referencia, Osborne/McGraw-Hill, 1994.