CAPÍTULO 7

FICHEROS

7.1 Flujos

Un flujo (s**tream**) es una abstracción que se refiere a un flujo o corriente de datos que fluyen desde una fuente a un destino. Entre la fuente y el destino tiene que existir una conexión o canal (**pipe**) para que circulen los datos. La apertura de un fichero supone establecer una conexión entre el programa y el dispositivo que contiene el archivo.

Hay tres flujos y canales que se abren automáticamente al ejecutar un programa:

```
extern FILE *stdin;
```

asocia la entrada estándar (teclado) con el programa.

```
extern FILE *stdout;
```

asocia la salida estándar (pantalla) con el programa.

```
extern FILE *stderr;
```

asocia la salida de mensajes de error (pantalla) con el programa.

7.2 El puntero FILE

Los archivos se ubican en dispositivos externos, tales como los discos, CD-ROM, etc. y tienen un nombre y unas características. En un programa C el archivo tiene un nombre interno **que es un puntero** a una estructura predefinida (puntero a archivo).

Esta estructura contiene información sobre el archivo, tal como, la dirección del búfer que utiliza, el modo de apertura del archivo, el último carácter leído del búfer y otros detalles que el usuario no necesita saber.

El identificador del tipo de la estructura es FILE y está declarado en el archivo stdio.h:

```
typedef struct{
short level;
unsigned flags; /* Estado del archivo: binario,..*/
char fd;
unsigned char hold;
short bsize;
unsigned char *buffer; *curp;
unsigned istemp;
short token;
} FILE;
```

El archivo se declara por medio de un puntero a FILE de la siguiente forma:

FILE * nombre del puntero

Ejemplo:

```
FILE* pf;
```

En definitiva, un puntero a fichero es una variable de tipo puntero a FILE que apunta al fichero y, por tanto, permite acceder a él. Se definirá un puntero a FILE por cada fichero utilizado en el programa.

7.3 Apertura de un fichero

La primera operación a realizar con un archivo en un programa C es la de apertura. Antes de utilizar un fichero, éste debe abrirse con la función **fopen** según la siguiente sintaxis:

fopen (nombre_archivo,modo)

Donde:

nombre_archivo: es una cadena que contiene el identificador externo del archivo.

modo: es una cadena que indica el modo en que se va a tratar el archivo. Los modos de apertura de un fichero son:

"r" Abre un archivo para lectura.
"w" Abre para crear un nuevo archivo (si ya existe se pierden sus datos).
"a" Abre para añadir al final.
"r+" Abre un archivo existente para modificar (Lectura/Escritura).
"w+" Crea un archivo para lectura/escritura. Si ya existe se pierden los datos.
"a+" Abre el archivo para modificar (lectura/escritura) al final. Si el fichero no existe es como "w+".

Por defecto, el archivo se suele abrir en forma de texto. En caso de duda se pueden utilizar los siguientes modos:

"rt" ,"wt", "at", "r+t", "w+t","a+t".

En el caso de que se quiera abrir un fichero binario se utilizan los modos:

"rb" ,"wb", "ab", "r+b", "w+b","a+b".

La función fopen devuelve un puntero al fichero indicado en su primer argumento. El puntero a FILE definido antes recibe la asignación de la función. Si el fichero no puede abrirse la función devuelve el valor NULL.

Ejemplo:

```
pf=fopen("fichero1","r");
```

7.4 Cierre de ficheros

Cuando se ha finalizado la utilización del fichero se puede realizar la operación de cierre, con ello, se sacan los datos del búfer intermedio, para lo cual, se utiliza la función **fclose**:

fclose (puntero de fichero)

Ejemplo: fclose(fp);

7.5 Entradas / Salidas con formato

Se analizan a continuación algunas funciones para leer/escribir datos de/en un fichero de texto.

- Lectura de datos de un fichero con **fscanf**

fscanf es una función que se encuentra en la librería stdio.h y tiene por sintaxis:

int **fscanf** (FILE *pf, cte char* formato, argumentos)

La función fscanf lee sus variables o argumentos del fichero apuntado por pf con el formato especificado. La descripción del formato es la misma que la

utilizada para scanf. El tipo de cada variable debe corresponderse con la especificación de formato indicada para cada una de ellas.

fscanf devuelve el número de argumentos que han sido leídos. Si el valor devuelto es 0 significa que no se han asignado valores. Si es EOF significa que se ha detectado el final del fichero.

Ejemplo:

```
FILE *pt;
int ent;
float real;
pt=fopen("fichero","r");
fscanf (pt, "%d %f", &ent , &real);
```

NOTA: Si el *stream* especificado por fscanf es stdin, el resultado es el mismo que si se hubiera invocado la función scanf. Por ejemplo, las siguientes sentencias, son equivalentes:

```
scanf("%d",&n);

fscanf(stdin,"%d",&n);
```

- Escritura de datos en un fichero con **fprintf**

fprintf es una función que se encuentra en la librería stdio.h y tiene por sintaxis:

```
int fprintf (FILE *pf,  cte char* formato, argumentos)
```

La función fprintf escribe sus argumentos en el fichero apuntado por pf, con el formato especificado. La descripción de formato es la misma que se utilizó con printf.

fprintf devuelve el número de caracteres escritos o un valor negativo si hay error.

Ejemplo:

```
FILE *pt;
int ent;
float real;
     pt=fopen("fichero","w");
fprintf (pt, "%7d %9.2f  \n", ent,  real);
```

NOTA: Si el *stream* especificado por fscanf es stdout, el resultado es el mismo que si se hubiera invocado la función printf. Por ejemplo, las siguientes sentencias son equivalentes:

```
printf("n=%d",n);

fprintf(stdout,"n=%d",n) ;
```

- Lectura de cadenas de un fichero con **fgets**

Para leer cadenas de caracteres se utiliza la función **fgets** que está incluida en la librería stdio.h. Su sintaxis es la siguiente:

```
char * fgets (char * cadena, int n ,FILE *pf)
```

La función fgets lee una cadena de caracteres del fichero apuntado por pf y la almacena en cadena.

Se leen n-1 caracteres desde la posición actual en el fichero (la indicada por el puntero lectura\escritura (L\E)), pero la lectura se interrumpe si se encuentra con el carácter de nueva línea: ('\n') o el final del fichero. En el caso en el que se haya encontrado con el carácter de nueva línea en la lectura, éste también se lee y almacena y, por tanto, al escribir cadena, se imprime también el salto de línea.

La terminación '\0' es añadida automáticamente a la cadena leída.

Ejemplo:

```
char cadena [75];
FILE *pf;
pf=fopen("fichero","r");
fgets(cadena, 25, pf);
```

En un fichero de datos, por lo general, suele haber ciertos registros que contienen información que no interesa al programa. Utilizando la función fgets de forma adecuada se pueden "saltar" las líneas del archivo que no interesen. Veamos:

Si el valor del segundo argumento de la función fgets es suficientemente grande, es seguro que la lectura llegará al carácter salto de línea y, por tanto, la siguiente lectura comenzará en el registro posterior. Esta técnica se emplea en el siguiente ejemplo:

Sea el fichero notas.txt con el contenido:

```
Notas del curso 2003-2004
Primer parcial
Ej1   Ej2    Ej3
7     5      8
3     2      6
5     1      3
10    5      4
.....................
```

Mediante un programa C se desea leer y almacenar las notas de cada alumno; por ello, los primeros datos importantes son los contenidos en el registro 4. Se van a leer las tres primeras líneas guardando la información en una variable auxiliar (que sólo usamos para este fin); a continuación, se puede producir la lectura almacenando la información que nos interesa:

```
#include <stdio.h>
void main( )
{
      FILE *p;
      int i,n1,n2,n3;
      char aux[30];
      p=fopen("notas.txt","r");
for(i=1;i<=3;i++)
          fgets(aux, 30, p);
fscanf(p,"%d %d %d",&n1, &n2, &n3);
printf("%d %d %d",n1,n2,n3);
}
```

- Escritura de cadenas en un fichero con **fputs**

Para escribir cadenas de caracteres en un fichero se utiliza la función **fputs**, que está incluida en la biblioteca stdio.h. Su sintaxis es:

int **fputs** (cte char* cadena, FILE *pf)

La función fputs copia la cadena de caracteres almacenada en cadena en el fichero apuntado por pf. La terminación en '\0' con que finaliza toda cadena no se copia. fputs devuelve el valor 0 si se ejecuta satisfactoriamente y en caso contrario un valor distinto de cero.

Ejemplo:

```
fputs(cadena, pf);          /*escribe en el fichero apuntado por pf*/
fputs(cadena, stdout);      /*escribe en pantalla*/
```

- Lectura de un carácter de un fichero con **fgetc**

Para la lectura de un carácter de un fichero, se utiliza la función **fgetc** que está almacenada en la biblioteca stdio.h. Su sintaxis es la siguiente:

```
int fgetc( FILE *pf )
```

fgetc lee un carácter de la posición indicada por el puntero de lectura/escritura (L/E) del fichero apuntado por pf y avanza la posición del puntero de L/E al siguiente carácter. fgetc devuelve el carácter leído o EOF si se llega al fin del fichero u ocurre un error.

Ejemplo:

```
FILE* pf;
char c;
pf=fopen("fichero","r");
c=fgetc(pf);
```

- Escritura de un carácter en un fichero con **fputc**

Para la escritura de un carácter en un fichero se utiliza la función **fputc** que se encuentra en la biblioteca stdio.h. Su sintaxis es:

```
int fputc (int car, FILE *pf )
```

fputc escribe un carácter (car) en la posición indicada por el puntero de L/E del fichero apuntado por pf. fputc devuelve el carácter escrito o EOF si ocurre un error o se llega al final del fichero.

Ejemplo:

```
FILE* pf;
pf=fopen("fichero","r");
fputc('z',pf);
```

7.6 Otras funciones auxiliares para el tratamiento de ficheros

Para gestionar la terminación de un fichero, se utiliza la función **feof** que se encuentra en la biblioteca stdio.h. Su sintaxis es la siguiente:

```
int feof( FILE *pf )
```

La función feof devuelve un valor distinto de 0 cuando se intenta leer un elemento y se encuentra con el final del fichero. En caso contrario devuelve cero. feof se emplea, por tanto, para detectar cuándo se ha llegado al final del fichero. La utilización de feof consiste en permitir leer del fichero mientras la función devuelva el valor 0.

Ejemplo de utilización de feof:

```
while(! feof(pf))
{
        leer el siguiente registro del fichero
}
fclose(pf);
```

Para regresar al comienzo de un fichero se utiliza la función **rewind** que está incluida en la librería stdio.h. Su sintaxis es la siguiente:

```
void rewind( FILE *pf )
```

Actúa desplazando el puntero de L/E al comienzo del fichero.

Ejemplo:

```
FILE* pf;
pf=fopen("fichero","r");
...../*lectura del fichero*/
rewind(pf); /*rebobinado*/
..... /*lectura desde el comienzo del fichero
```

7.7 Enunciados de las prácticas

NOTA PRELIMINAR

La creación de ficheros de texto en los que se pueden almacenar datos para ser leídos por programas, se puede hacer dentro del entorno del Developer Studio de Microsoft, siguiendo los pasos que se detallan a continuación:

- Del menú **File** escoger el ítem **New**.
- En el cuadro que aparece, hacer clic en la pestaña **Files** y dentro de ella seleccionar *Text File*.
- Desmarcar la opción *Add to project*.
- Dar nombre al Fichero, por ejemplo, **datose.txt** e incluirlo en el directorio del proyecto al que pertenece el programa que va a utilizar el fichero.
- Escribir los datos dentro del Fichero.
- Guardar el Fichero escogiendo el ítem *Save* del menú **File**.

Ejercicio 1.

Determinar la salida a pantalla del siguiente programa C:

```
#include<stdio.h>

void main( )
{
    FILE *pf1;
    char nombre[60];
    int i2,i3,i4;
    float i1;

    printf("\n Nombre del fichero: ");
    scanf("%s",nombre);

    pf1=fopen(nombre,"r");

    fscanf(pf1,"%2f %d %d",&i1,&i2,&i3);
    printf("\n i1=%2f i2=%d i3=%d",i1,i2,i3);

    fscanf(pf1,"%3d %4d %2d",&i2,&i3,&i4);
    printf("\n i2=%d i3=%d i4=%d",i2,i3,i4);

    fclose(pf1);

}
```

si lee un fichero que contiene los siguientes datos:

```
1.245
5    98
12345678901234
```

Nota: en este programa se pregunta al usuario el nombre del fichero que contiene los datos y se guarda este nombre en la variable de caracteres *nombre*, que es utilizada como primer argumento de la función fopen.

Ejercicio 2.

Dado el siguiente programa C :

```
#include<stdio.h>

void main(  )
{
     FILE *pf1;
     char nombre[60];
     char car;
     int nc=0;
     printf("\n Nombre del fichero");
     scanf("%s",nombre);

     pf1=fopen(nombre,"r");
     while((car=fgetc(pf1))!=EOF)
          if (car!=' '&&car!='\n')
          {
          ++nc;
          printf("\n caracter numero %d =%c",nc,car);
          }
     printf("\n nc= %d",nc);
     fclose(pf1);

}
```

Determinar la salida a pantalla, si el usuario introduce como nombre del fichero "e.txt", y ese fichero contiene la siguiente información:

```
12   5     898
metodosmatema 34 ticos
12345678901234
```

Explicar cuál es el significado de la variable nc.

Indicación: Las líneas de un fichero están separadas por el carácter no visible: \n

Ejercicio 3.

Determinar la salida a pantalla del siguiente programa C:

```
#include<stdio.h>

void main(  )
{
     FILE *pf1;
     char nombre[60],palabra[100],c;
     printf("\n Nombre del fichero");
     scanf("%s",nombre);

     pf1=fopen(nombre,"r");
     do
     {
          c=fscanf(pf1,"%s",palabra);
          if(c!=EOF)printf("\n el valor leido es %s",palabra);
      } while(c!=EOF);
      fclose(pf1);

}
```

si el usuario introduce como nombre del fichero "e.txt", y ese fichero contiene la siguiente información:

```
12   5     898
metodosmatema 34 ticos
12345678901234
```

Ejercicio 4.

Escribir un programa C que lea un fichero (**datose.txt**) con un número indeterminado de registros y que genere un nuevo fichero (**datosa.txt**) con los mismos registros pero numerados desde 1 al número total de registros que existan en el fichero. Por ejemplo:

datose.txt	**datosa.txt**
primero	1 primero
segundo	2 segundo
tercero	3 tercero
cuarto	4 cuarto
quinto	5 quinto
.....	.....

Ejercicio 5.

Escribir un programa C que lea un fichero (**datose.txt**) con un número indeterminado de registros y que genere un nuevo fichero (**datosa.txt**) con los mismos registros pero ordenados alfabéticamente. Por ejemplo:

datose.txt	**datosa.txt**
primero	cuarto
segundo	primero
tercero	quinto
cuarto	segundo
quinto	tercero
.....	.....

Ejercicio 6. (Preparación de un fichero de datos).

Un plano π en R^3 viene dado por el vector de posición **a** de uno cualquiera de sus puntos y un vector normal al plano **n.** Se llama SEMIESPACIO generado por el plano π al conjunto de puntos que están situados en el lado de π que se encuentra en dirección contraria a la normal. Dados k planos, se llama POLIEDRO generado por ellos, a la intersección de los k semiespacios generados por los k planos.

Un punto, de vector de posición **p,** será interior al poliedro si:

$$\mathbf{n_i}.(\mathbf{p}-\mathbf{a_i}) <= 0 \quad \text{para} \quad i=1,\ldots\ldots,k$$

Si p es un punto exterior al poliedro, se dice que la cara j es VISIBLE desde **p** si

$$\mathbf{n_j}.(\mathbf{p}-\mathbf{a_j}) > 0.$$

Se pide:

a) Elaborar un programa C que tenga como entrada los k vectores de posición y las k normales que definen el poliedro (leídas de un fichero en forma secuencial, al que se le llamará *dataico.txt*) y las coordenadas del punto **p** (leído del teclado). Como salida, el programa debe decir si el punto **p** es interior o exterior al poliedro y en caso de que sea exterior, debe indicar las caras que son visibles desde **p**.

b) Aplicar dicho programa al caso de un icosaedro, cuyos datos son los siguientes:

Cara 1:	a=(0,x,1)	n=(y,z,0)
Cara 2:	a=(0,x,1)	n=(-y,z,0)
Cara 3:	a=(0,x,1)	n=(-v,v,v)
Cara 4:	a=(0,x,1)	n=(0,y,z)
Cara 5:	a=(0,x,1)	n=(v,v,v)
Cara 6:	a=(1,0,x)	n=(z,0,y)
Cara 7:	a=(1,0,-x)	n=(z,0,-y)
Cara 8:	a=(0,x,-1)	n=(v,v,-v)
Cara 9:	a=(0,x,-1)	n=(0,y,-z)
Cara 10:	a=(0,x,-1)	n=(-v,v,-v)

Cara 11:	a=(0,-x,1)	n=(-v,-v,v)
Cara 12:	a=(0,-x,-1)	n=(-v,-v,-v)
Cara 13:	a=(0,-x,-1)	n=(y,-z,0)
Cara 14:	a=(0,-x,1)	n=(0,-y,z)
Cara 15:	a=(0,-x,1)	n=(v,-v,v)
Cara 16:	a=(-1,0,x)	n=(-z,0,y)
Cara 17:	a=(-1,0,-x)	n=(-z,0,-y)
Cara 18:	a=(0,-x,-1)	n=(0,-y,-z)
Cara 19:	a=(0,-x,-1)	n=(v,-v,-v)
Cara 20:	a=(0,-x,-1)	n=(-y,-z,0)

Donde se ha utilizado la notación:

x=1.618034
y=0.356822
z=0.934172
v=0.577350

El fichero *dataico.txt* SE DEBE ENCABEZAR por el número 20 (k) y los datos reseñados arriba, que pueden teclearse como letras y luego ser pasados a números por medio de las utilidades del editor de ficheros de Visual C.

El fichero incluiría la siguiente información:

```
20
0       x       1       y       z       0
0       x       1       -y      z       0
0       x       1       -v      v       v
.....................
.........
```

Del menú **Edit** elegir Replace. En **Find What** escribir *x* y en **Replace with** escribir *1.618034* . Hacer los cambios mediante un clic en **Replace All**. Repetir esta operación para los otros tres valores.

7.8 Soluciones a las prácticas del capítulo 7

Ejercicio 1.

La salida a pantalla es la siguiente:

```
i1=1.000000    i2=245     i3=5
i2=98    i3=1234    i4=56
```

El formato de lectura de la función **fscanf** actúa del siguiente modo:

"......%d %d"	si en los formatos no se indica anchura de campo, se leen como datos distintos los separados por blancos o saltos de línea.
"%2f..... " "%3d %4d %2d"	si en los formatos aparece anchura de campo, se lee un dato de esa anchura como máximo cortando antes si se encuentra un blanco o salto de línea.

A continuación se recuadran los datos del fichero tal como se leen con fscanf.

```
1. 245
5   98
1234 5678901234
```

Ejercicio 2.

A continuación se escribe la salida a pantalla:

```
caracter numero 1 =1
caracter numero 2 =2
caracter numero 3 =5
caracter numero 4 =8
caracter numero 5 =9
caracter numero 6 =8
caracter numero 7 =m

        ..
        ..          así seguiría hasta finalizar todos los
        ..          caracteres no blancos ni fin de línea

caracter número 40 =4
 nc=40
```

La variable nc representa el número de caracteres distintos de blanco que contiene el fichero.

Ejercicio 3.

La salida a pantalla es la siguiente:

```
el valor leido es 12
el valor leido es 5
el valor leido es 898
el valor leido es metodosmatema
el valor leido es 34
el valor leido es ticos
el valor leido es 12345678901234
```

El programa podría modificarse muy fácilmente para contar el número de palabras; quedaría así:

```
int npal=0;
  ..                El resto queda igual
  ..
do
 {
 c=fscanf(pf1,"%s",palabra);
 if(c!=EOF)
 {
       printf("\n el valor leido es %s",palabra);
       npal++;
 }
while(c!=EOF);
```

Ejercicio 4.

Una solución válida es el siguiente:

```
#include <stdio.h>

void main(  )
{
    char cad[30];
    int i=0;
    FILE *pfe,*pfs;
    pfe=fopen("datose.txt","r");
    pfs=fopen("datosa.txt","w");

    while(!feof(pfe))
    {
        fgets(cad,30,pfe);
        fprintf(pfs, "%d %s",i+1,cad);
        i=i+1;
    }

    printf("Se han leido %d registros \n",i);
}
```

Ejercicio 5.

Un posible programa es el siguiente:

```
#include <stdio.h>
#define  MAX  100

void ordenar(int v[ ], int n);


void main(  )
{
    FILE *pfe,*pfs;
    int i=0,j,k,v[MAX],v0[MAX],s[MAX],o[MAX];
    char cad[MAX][30];
    pfe=fopen("datose.txt","r");
    pfs=fopen("datosa.txt","w");

/* Lectura de los registros. La inicial de
cada registro se sitúa en el vector de tipo entero v */

    while(!feof(pfe))
    {
        fgets(cad[i],30,pfe);
        v[i]=cad[i][0];
        v0[i]=v[i];
        i++;
    }

/* i es el número de componentes de v*/

/* ordenación de v */

    ordenar(v,i);


/* determinación del orden de escritura de los registros
en el fichero de salida, mediante el vector auxiliar  o[  ]  */



  for(j=0;j<i;j++)
    s[j]=0;
```

```
    for(j=0;j<i;j++)
    {
        for(k=0;k<i;k++)
       {
           if(v[j]==v0[k] && s[k]==0)
           {
            o[j]=k;
            s[k]=1;
            break;
           }
       }
   }
/* Escritura ordenada de los registros  */
    for(j=0; j<i; j++)
    {
        fputs(cad[o[j]],pfs);
    }
        fclose(pfs);
}

void ordenar(int v[ ], int n)
{
/* ordenación de los elementos de un vector por el método de selección */
    int i,j,k,m;
    for(i=0;i<n-1;i++)
    {
    m=v[i];
    k=i;

/* Búsqueda de la menor componente */
    for(j=i+1;j<n;j++)
    {
        if(v[j]<m)
        {
          m=v[j];
          k=j;
        }
    }

/* Intercambios */
    v[k]=v[i];
    v[i]=m;
    }
}
```

Ejercicio 6.

A continuación se ofrece una solución válida:

```
#include <stdio.h>

void main(  )
{
    int i,j,k,y,cv[100];
    double p[3], a[3][100],n[3][100], prod;
/* cv es un vector que va a contener las caras visibles del poliedro */
    FILE *pf;
    pf=fopen("datosico.txt","r");
    fscanf(pf, "%d",&k);
    for(j=0;j<k;j++)
    {
      for(i=0;i<3;i++)   fscanf(pf, "%lf",&a[i][j]);
      for(i=0;i<3;i++)   fscanf(pf, "%lf",&n[i][j]);
    }
    printf("Introduce las coordenadas del punto p \n");
    for(i=0;i<3;i++) scanf("%lf", &p[i]);
    i=0;
    for(j=0;j<k;j++)
    {
       prod=0;
       for(y=0;y<3;y++)   prod=prod+n[y][j]*(p[y]-a[y][j]);
       if(prod>0)
      {
         cv[i]=j+1;
         i++;
      }
    }
    if(i == 0) printf("punto p interior al poliedro \n");
    else
    {
        printf("punto p exterior al poliedro \n");
        printf("el nº de caras visibles es: %d \n ", i);
        printf("concretamente: \n");
        for(j=0; j<i; j++)
                    printf("%d \n",cv[j]);
    }
```