D Manejo de archivos

Los archivos son manejados por el sistema operativo. Para trabajar con un archivo, primero se debe conectar el programa con el archivo, a través de la operación de apertura (open), y al finalizar, desconectarlo a través de la operación de cerrado (close). Además, por razones técnicas, se debe definir un bloque de memoria en el que se almacenará toda la información asociada con el archivo. Esta estructura ya está definida en el archivo de encabezamiento `stdio.h` y recibe el nombre de `FILE`.

Sin embargo, sólo se declara un puntero a esta estructura:

```
FILE *fp;
```

Aquí `fp` recibe el nombre de puntero a archivo. Para poder asociar este puntero a un archivo, usamos la función `fopen`:

- `fp = fopen(nombre,codigo)` abre el archivo cuyo nombre es especificado por el string `nombre`, de acuerdo a las especificaciones dadas por el segundo string `codigo`.

Algunos valores posibles para el string `codigo` son [9]:

código	descripción
`"r"`	Lectura. Sólo para archivos existentes.
`"w"`	Escritura. Si ya existe, sus contenidos se pierden, sino es creado.
`"a"`	Append. Si el archivo no existe, es creado.
`"r+"`	Igual que `"r"`, pero permitiendo además escritura.
`"w+"`	Igual que `"w"`, pero permitiendo además lectura.
`"a+"`	Igual que `"a"`, pero permitiendo además lectura.

Si un archivo no se puede abrir, la función `fopen` retorna `NULL`. La detección del error se puede hacer como sigue:

```
if ((fp = fopen("ejemplo.dat","r")) == NULL) {
  printf("Error: el archivo ejemplo.dat no se puede abrir\n");
  exit(1);
}
```

La lectura y la escritura se pueden realizar con un gran número de funciones. Las más simples de utilizar son `fscanf` y `fprintf`. Se comportan en forma similar a las tradicionales `scanf` y `printf`, pero realizando la lectura y la escritura en un archivo. Ambas esperan un puntero a archivos como su primer argumento:

- `fscanf(fp,str,...)` lee del archivo apuntado por `fp`, de acuerdo a las especificaciones de formato dadas por el string `str`, y almacena los valores leídos en las variables que son pasadas como argumento luego del string de formato.

[9] Sólo en el sistema operativo MS-DOS es necesario distinguir entre archivos de texto y archivos binarios. Para abrir un archivo binario agregar la letra b en la segunda posición del string `codigo`.

- `fprintf(fp,str,...)` escribe en el archivo apuntado por `fp`, de acuerdo a las especificaciones de formato dadas por el string `str`, los valores de las variables que son pasadas como argumento luego del string de formato.

Por ejemplo, el siguiente programa lee enteros del archivo `num.dat` y escribe sus valores absolutos en el archivo `abs.dat`:

```
#include <stdio.h>

int main() {
  FILE *in,*out;
  int i;

  if ((in = fopen("num.dat","r")) == NULL) {
    printf("Error: el archivo num.dat no se puede abrir\n");
    exit(1);
  }

  if ((out = fopen("abs.dat","w")) == NULL) {
    printf("Error: el archivo abs.dat no se puede abrir\n");
    exit(1);
  }

  while (fscanf(in,"%d",&i) > 0)
    fprintf(out,"%d",abs(i));

  fclose(in);
  fclose(out);
}
```

Las dos últimas sentencias corresponden a la invocación de la función `fclose`, la cual cierra la conexión entre el programa y el archivo cuyo puntero es pasado como argumento.

Con estas funciones, la información siempre se escribe en ASCII en el archivo. La ventaja es que estas funciones son simples de usar y que los archivos pueden ser leídos, creados o modificados directamente con un editor de textos. La desventaja es que los archivos son usualmente más grandes. Por ejemplo, en una computadora en la que el tamaño de una palabra es 16 bits, el entero 25000 en representación ASCII ocupa 5 bytes, más los espacios que se necesitan para separarlo de los otros datos. En su representación binaria ocupa sólo dos bytes y no necesita separadores.

Existen además funciones para la detección de errores:

- `feof(fp)` retorna un valor distinto de 0 si se alcanza el fin de archivo, o 0 en otro caso.
- `ferror(fp)` retorna un valor distinto de 0 si ha ocurrido un error durante la lectura o escritura en archivo, o 0 en otro caso.

También existen funciones que permiten trabajar caracter a caracter:

- `getc(fp)` retorna el próximo caracter en el archivo, o `EOF` si no hay más.

- `putc(c,fp)` escribe el caracter `c` en el archivo.

El siguiente programa permite copiar un archivo en otro. Asume que los nombres de los archivos han sido pasados como argumentos en la línea de comandos:

```
#include <stdio.h>

int main(int argc,char *argv[]) {
  FILE *in,*out;
  int c;

  if ((in = fopen(argv[1],"r")) == NULL) {
    printf("Error: el archivo de entrada no se puede abrir\n");
    exit(1);
  }

  if ((out = fopen(argv[2],"w")) == NULL) {
    printf("Error: el archivo de salida no se puede abrir\n");
    exit(1);
  }

  while (c = getc(in),c != EOF)
    putc(c,out);

  fclose(in);
  fclose(out);
}
```

C también provee funciones que permiten la lectura y escritura no formateada. Algunas de ellas son:

- `fread(buffer,tam,n,fp)` lee del archivo apuntado por `fp`, `n` componentes de tamaño `tam`, y las almacena a partir de la dirección apuntada por `buffer`.

- `fwrite(buffer,tam,n,fp)` escribe en el archivo apuntado por `fp`, `n` componentes de tamaño `tam`, que son tomadas a partir de la dirección apuntada por `buffer`.

Ambas funciones retornan un entero que representa el número de elementos leídos o grabados, por lo que es simple chequear si se han producido errores.

El siguiente programa muestra como es posible grabar un arreglo de registros en forma completa:

```
#include <stdio.h>

struct punto {
  int x;
  int y;
} p[2] = { { 2 , 3 } , { 1 , 8 } };
```

```
int main() {
  FILE *fp;

  if ((fp = fopen("arch.dat","w")) == NULL) {
    printf("Error: el archivo arch.dat no se puede abrir\n");
    exit(1);
  }

  if (fwrite(p,sizeof(struct punto),2,fp) != 2) {
    printf("Error: no se pudieron grabar las dos componentes\n");
    exit(1);
  }

  fclose(fp);
}
```

El siguiente programa muestra como se pueden leer los registros almacenados en el archivo generado con el programa anterior:

```
#include <stdio.h>

struct punto {
  int x;
  int y;
} p;

int main() {
  FILE *fp;

  if ((fp = fopen("arch.dat","r")) == NULL) {
    printf("Error: el archivo arch.dat no se puede abrir\n");
    exit(1);
  }

  while (fread(&p,sizeof(struct punto),1,fp) == 1) {
    printf("x = %d y = %d\n",p.x,p.y);
  }

  fclose(fp);
}
```

Notar que aunque las estructuras fueron almacenadas juntas, se pueden leer una por una. Notar además que el valor retornado por la función `fread` se utiliza para detectar el fin de archivo. Si la función `fread` retorna un valor distinto al número de componentes que se solicitó que leyera (1 en este caso), implica que no pudo realizar una lectura satisfactoria.

C provee también funciones que trabajan con la posición física en el archivo:

- `fseek(fp,desplazamiento,origen)` permite modificar la posición de lectura y escritura en el archivo. Sus argumentos son el puntero al archivo `fp`,

un desplazamiento en bytes desplazamiento y un valor origen que especifica el origen del desplazamiento. La función retorna 0 si la operación es exitosa y un valor distinto de 0 en otro caso.

- ftell(fp) retorna la posición corriente en el archivo apuntado por fp.

Los valores posibles para origen son:

origen	descripción
SEEK_SET	el desplazamiento es considerado desde el comienzo.
SEEK_CUR	el desplazamiento es considerado desde la posición corriente.
SEEK_END	el desplazamiento es considerado desde el final.

El siguiente programa muestra como se puede escribir al final del archivo que fue generado anteriormente, y luego leer todas las componentes:

```
#include <stdio.h>

struct punto {
  int x;
  int y;
} p,q = { 4 , 5 };

int main() {
  FILE *fp;

  if ((fp = fopen("arch.dat","r+")) == NULL) {
    printf("Error: el archivo arch.dat no se puede abrir\n");
    exit(1);
  }

  if (fseek(fp,0,SEEK_END) != 0) {
    printf("Error: no se pudo ir al final del archivo\n");
    exit(1);
  }

  if (fwrite(&q,sizeof(struct punto),1,fp) != 1) {
    printf("Error: no se pudo grabar la estructura\n");
    exit(1);
  }

  if (fseek(fp,0,SEEK_SET) != 0) {
    printf("Error: no se pudo ir al comienzo del archivo\n");
    exit(1);
  }

  while (fread(&p,sizeof(struct punto),1,fp) == 1) {
    printf("x = %d y = %d\n",p.x,p.y);
  }

  fclose(fp);
}
```