2011

UNAN – LEÓN
Departamento de Computación

Asignatura:
ALGORITMO Y ESTRUCTURA DE DATOS

MANEJO DE ARCHIVOS (FICHEROS) EN C

TEMA 1: MANEJO DE ARCHIVOS EN C

1.1 GENERALIDADES:

Muy a menudo necesitamos almacenar cierta cantidad de datos de forma más o menos permanente. La memoria del ordenador es volátil. De modo que cuando tenemos que guardar nuestros datos durante cierto tiempo tenemos que recurrir a sistemas de almacenamiento más económicos, aunque sea a costa de que sean más lentos. Durante la historia de los ordenadores se han usado varios métodos distintos para el almacenamiento de datos.

Los datos que hemos tratado hasta el momento han residido en la memoria principal. Sin embargo, las grandes cantidades de datos se almacenan normalmente en un dispositivo de memoria secundaria. Los **ficheros son estructuras de datos almacenadas en memoria secundaria.**

Hay dos tipos de archivos, **archivos de texto y archivos binarios**. Un **archivo de texto** es una secuencia de caracteres organizadas en líneas terminadas por un carácter de nueva línea. En estos archivos se pueden almacenar canciones, archivos fuentes de programas, base de datos, etc. Los archivos de texto se caracterizan por ser planos, es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño.

Un **archivo binario** es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo. Así que no tendrá lugar ninguna traducción de caracteres. Además, el número de bytes escritos (leídos) será el mismo que los encontrados en el dispositivo externo. Ejemplos de estos archivos son: Fotografías, imágenes, texto con formatos, archivos ejecutables (aplicaciones), etc.

En C, **un archivo es un concepto lógico** que puede aplicarse a muchas cosas desde archivos de disco hasta terminales o una impresora. Se asocia una secuencia con un archivo específico realizando una operación de apertura. Una vez que el archivo está abierto, la información puede ser intercambiada entre este y el programa. Se puede conseguir la entrada y la salida de datos a un archivo a través del uso de la biblioteca de funciones; C no tiene palabras claves que realicen las operaciones de E/S.

El estándar de C contiene 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.

Nombre	Función
fopen()	Abre un archivo.
fclose()	Cierra un archivo.
fgets()	Lee una cadena de un archivo.
fputs()	Escribe una cadena en un archivo
fseek()	Busca un byte especifico de un archivo.
fprintf()	Escribe una salida con formato en el archivo.
fscanf()	Lee una entrada con formato desde el archivo.
feof()	Devuelve cierto si se llega al final del archivo.
ferror()	Devuelve cierto si se produce un error.
rewind()	Coloca el localizador de posición del archivo al principio del mismo.
remove()	Borra un archivo.
fflush()	Vacía un archivo.

La secuencia que usaremos para realizar operaciones será la siguiente:

1. Crear un apuntador del tipo **FILE ***.
2. Abrir el archivo utilizando la función **fopen** y asignándole el resultado de la llamada a nuestro apuntador.
3. Hacer las diversas operaciones (lectura, escritura, etc).
4. Cerrar el archivo utilizando la función **fclose**.

1.2 EL PUNTERO A UN ARCHIVO:

El puntero a un archivo es el hilo común que unifica el sistema de E/S con buffer. Un puntero a un archivo es un puntero a una información que define varias cosas sobre él, incluyendo el nombre, el estado y la posición actual del archivo. En esencial identificar un archivo especifico y utilizar la secuencia asociada para dirigir el funcionamiento de las funciones de E/S con buffer.

Un puntero a un archivo es una **variable de tipo puntero a la estructura FILE** que se define en **"stdio.h"** para el manejo de ficheros. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos. La definición de la estructura **FILE** depende del compilador, pero en general mantienen un campo con la posición actual de lectura/escritura, un buffer para mejorar las prestaciones de acceso al fichero y algunos campos para uso interno, al programador le interesa su uso como estructura FILE.

Para obtener una variable de este tipo se utiliza una secuencia como esta: **FILE *F;**

En realidad, una variable de tipo FILE * representa un flujo de datos que se asocia con un dispositivo físico de entrada/salida (el fichero "real" estará almacenado en disco).

Existen flujos de datos estándar predefinidos asociados a otros dispositivos de entrada/salida. Algunos de ellos son:

- **stdin**: representa la entrada estándar del sistema (teclado).
- **stdout:** representa la salida estándar del sistema (pantalla).
- **stderr:** representa la salida de error estándar (pantalla).

1.3 APERTURA DE UN FICHERO:

Para poder operar con un fichero, exista previamente o no, es necesario "abrirlo" mediante la función **fopen**.

El prototipo de dicha función es: **FILE *fopen (const char *filename, const char *mode);**

Respecto a este prototipo:

- Nótese que fopen devuelve un valor de tipo **FILE *** (o sea, un puntero a FILE). Por supuesto, se supone que el valor devuelto será asignado a una variable de tipo FILE *, que se usará en otras funciones que manipulen dicho fichero.
- Si la apertura del fichero falla, fopen devuelve un puntero nulo.
- El argumento **filename** (una cadena de caracteres) es el nombre "real" del fichero que va a abrirse mediante fopen (es decir, es el nombre con el que el fichero aparece en el disco).
- El argumento **mode** (una cadena de caracteres): indica qué tipo de operaciones se realizarán sobre el fichero abierto y el tipo de datos que puede contener, de texto o binarios:.

Los posibles valores de mode son:

- **r:** El fichero se abre para lectura. Si el fichero no existe, se devuelve un puntero nulo.
- **w:** Se crea el fichero para escritura. Si ya existe un fichero con ese nombre, el fichero antiguo será eliminado.
- **a:** Si ya existe un fichero con ese nombre, se abre para escritura (al final del fichero). Si no existe, se crea.
- **r+**: Si el fichero existe, se abre para lectura y escritura (al principio del fichero).
- **w+:** Se crea el fichero para lectura y escritura. Si ya existe un fichero con ese nombre, el fichero antiguo será eliminado.
- **a+:** Si el fichero existe, se abre para lectura y escritura (al final del fichero). Si el fichero no existe, se crea.

En estos modos no se ha establecido el tipo de archivo, para ello se utilizará t para especificar un archivo de texto o b para binario.

- **t: tipo texto**, si no se especifica "t" ni "b", se asume por defecto que es **"t"**.
- **b: tipo binario**.

Es decir: **"rt", "wt", "at", "r+t", "w+t", "a+t"** o bien **"rb", "wb", "ab", "r+b", "w+b", "a+b"**

Nota: Sea cual sea el valor elegido para mode, debe aparecer entre dobles comillas en la llamada a fopen.

1.4 CIERRE DE UN FICHERO:

Es importante cerrar los ficheros abiertos antes de abandonar la aplicación. Esta función sirve para eso. Cerrar un fichero almacena los datos que aún están en el buffer de memoria, y actualiza algunos datos de la cabecera del fichero que mantiene el sistema operativo. Además permite que otros programas puedan abrir el fichero para su uso. Muy a menudo, los ficheros no pueden ser compartidos por varios programas.

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**. El parámetro es un puntero a la estructura FILE del fichero que queremos cerrar.

Para cerrar un fichero, se usa la función **fclose**. Su prototipo es: **int fclose(FILE *fp);**

Ejemplo #1: Programa en C para abrir un fichero.

```
//AperturaArchivo.c
#include <stdio.h>
#include <stdlib.h>
int main()
{
        FILE *fp;
        fp=fopen("archivo.txt","r");
        if(fp==NULL)
        {
                printf("Error al abrir el archivo para leer");
                exit(1);
        }
        fclose(fp);
         return 0;
}
```

1.5 OPERACIONES DE LECTURA Y ESCRITURA EN FICHEROS:

Las funciones empleadas para realizar operaciones de lectura y escritura en ficheros varían dependiendo del tipo de fichero manipulado:

- Si se asume que la información contenida en el fichero debe interpretarse exclusivamente como caracteres ASCII, se dice que el fichero tiene formato de texto.
- Si se asume que el formato de la información contenida en el fichero no está restringido a uno concreto, se dice que el fichero tiene formato binario.

- **Las funciones empleadas con ficheros en formato texto son:**

1. Para lectura y escritura con formato: **fscanf y fprintf.**
2. Para leer y escribir un carácter: **fgetc y fputc.**
3. Para leer y escribir una cadena: **fgets y fputs.**

- **Las funciones empleadas con ficheros en formato binario son:**

Se lee y escribe directamente un bloque de bytes mediante las funciones **fread y fwrite**.

1.5.1 Lectura y Escritura con formato

Para leer y escribir cualquier dato con formato en un fichero de texto se emplean las funciones fscanf y fprintf. Los prototipos de dichas funciones son:

- **int fscanf(FILE *fp, const char *fmt,…);**
- **int fprintf(FILE *fp, const char *fmt,…);**

El valor entero devuelto por ambas funciones indica si la operación se ha llevado a cabo con éxito.

Ejemplo #2: Programa en C que lee dos caracteres de un archivo y los copia en otro.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	FILE *fp1,*fp2;
	char c1,c2;
	fp1=fopen("archivo.txt","r");
	if(fp1==NULL)
	{
		printf("Error al abrir el archivo para leer");
		system("PAUSE");
		exit(1);
	}
	fp2=fopen("copia.txt","w");
	if(fp2==NULL)
	{
		printf("Error al abrir el archivo copia.txt");
		system("PAUSE");
		exit(1);
	}
	fscanf(fp1,"%c%c",&c1,&c2);
	fprintf(fp2,"%c%c",c1,c2);
```

```
        fclose(fp1);
        fclose(fp2);
        return 0;
}
```

1.5.2 Lectura y Escritura de un carácter:

Para leer y escribir un carácter en un fichero de texto se emplean las funciones **fgetc y fputc**. Los prototipos de dichas funciones son:

- **int fgetc(FILE *fp);**
- **int fputc(int c,FILE *fp);**

Respecto a estos prototipos:

- El valor entero que devuelve fgetc es el valor ASCII del carácter leído.
- fputc admite como primer argumento un carácter ó un entero.
- Si se da un carácter, se escribirá en el fichero dicho carácter.
- Si se da un entero, se escribirá el carácter correspondiente a dicho entero según el código ASCII.
- En cualquier caso, fputc devuelve el valor ASCII del carácter escrito si la operación se lleva a cabo con éxito.

Ejemplo #3: Programa en C que escribe un carácter leído del teclado en un fichero.

//Ej3_Funciones.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        FILE *fp;
        char caracter;
        printf("\nIntroduce caracter:\n");
        scanf("%c",&caracter);
        fp=fopen("caracter.txt","w");
        fputc(caracter,fp);
        fclose(fp);
        printf("\nCaracter guardado en el fichero!!\n");
        system("PAUSE");
        return 0;
}
```

Ejemplo #4: Programa en C que imprime un carácter leído de un fichero.

```
//Ej4_Funciones.c
#include <stdio.h>
int main()
{
        FILE *fp;
        char leido;
        fp=fopen("archivo.txt","r");
        leido=fgetc(fp);
        fclose(fp);
        printf("\nEl Caracter Leido es: %c\n",leido);
        system("PAUSE");
        return 0;
}
```

1.5.3 Lectura y Escritura de un cadena:

Para leer y escribir una cadena de caracteres en un fichero se emplean las funciones fgets **y fputs**. Los prototipos de dichas funciones son:

- **char *fgets(char *s,int n,FILE *fp);**
- **int fputs(char *s,FILE *fp);**

Respecto a estos prototipos:

- **fgets** lee una serie de caracteres de un fichero y los asigna a una cadena.
- El nombre de la cadena debe indicarse como primer argumento, y el tamaño de la misma como segundo argumento.
- Si se indica un tamaño de cadena **n** se leerán n-1 caracteres del fichero, salvo que se llegue a un carácter \n, en cuyo caso se detiene la lectura.
- El valor devuelto por **fgets** es el primer argumento (el puntero a la cadena) si la operación se lleva a cabo con éxito, y un puntero nulo si la operación falla.
- **fputs** copia los caracteres que forman la cadena (salvo el carácter de terminación) indicada como primer argumento en el fichero.
- **fputs** devuelve el valor ASCII del último carácter escrito si la operación se lleva a cabo con éxito.

Ejemplo #5: Programa en C que muestra la lectura y escritura de una cadena de N caracteres en un fichero.

//Ej5_Funciones.c

```
#include <stdio.h>
int main()
{
	FILE *fp;
	char cadena[]="Algoritmo y Estructura de Datos",leida[20];
	fp=fopen("ejemplo5.txt","w");
	fputs(cadena,fp);
	fclose(fp);
	fp=fopen("ejemplo5.txt","r");
	fgets(leida,sizeof(leida)/sizeof(char),fp);
	fclose(fp);
	printf("\nCadena leida: %s\n",leida);
	system("PAUSE");
	return 0;
}
```

Ojo: Sólo se leerán 19 caracteres del fichero. El lugar 20 se reserva para el '\0'.

BIBLIOGRAFÍA BÁSICA:

- Ceballos, Francisco Javier: **C/C++ Curso de Programación**, 2da Edición. Editorial RA-MA, 2002.
- Joyanes Aguilar, Luis; Zahonero Martínez Ignacio: **Programación en C.** McGraw Hill, 2001.
- Gottfried, Byron S: **Programación en C**. McGraw Hill, 1991.