***MANEJO DE ARCHIVOS EN C.***

Un archivo puede definirse en forma lógica diciendo que:

Es un grupo de datos estructurados que son

almacenados en algún medio y pueden ser

utilizados por las aplicaciones.

Las aplicaciones conocidas como Word, PowerPoint, Excel, etc. generan sus propios archivos a los cuales le anexan una extensión que los caracteriza. Por ejemplo:

.docx Archivos de Word.

.xlsx Archivos de Excel.

.pptx Presentación Power.

.txt Cuaderno de notas.

.psd Photoshop.

.mp3 Reproductor de Windows / AIMP / Winamp / etc.

Estos archivos deben ser utilizados exclusivamente por la aplicación que los generó.

Sin embargo la mayoría de las aplicaciones pueden manejar archivos de imágenes:

.jpg

.bmp

.tif

etc.

Un documento de Word puede insertar una imagen .jpg.

Una planilla en Excel puede importar un archivo de texto con formato.

etc.

Desde un punto de vista más básico un archivo es:

Un flujo o secuencia de bytes hacia un dispositivo de almacenamiento, en donde el concepto de dato desparece.

Esto sería mirando desde el hardware: el SO recibe un pedido de guardar o leer una cierta cantidad de bytes, los cuales carecen de todo significado para él. El encargado de verter esos bytes en el molde adecuado a fin de que tenga algún sentido para el usuario, es la Aplicación que lo solicitó.

Ocurre algo similar a cuando generábamos una reserva de memoria (una cierta cantidad de bytes) y almacenábamos algo allí. Si accedíamos a esa reserva con un puntero "tipificado" como una estructura, los bytes adquirían automáticamente un sentido perfectamente lógico al ubicarse en cada miembro de la estructura.

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 archivos, 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 puntero a la información del archivo.

Las funciones más comunes para trabajar con archivos son:

|  |  |
| --- | --- |
| **Nombre** | **Función** |
| fopen() | Abre un archivo |
| fclose() | Cierra un archivo |
| fgets() | Lee una cadena de caracteres de un archivo |
| fputs() | Escribe una cadena de caracteres en un archivo |
| putc() | Escribe un caractér en un archivo |
| getc() | Lee un caractér desde un archivo |
| fseek() | Salta al byte especificado en un archivo |
| fprintf() | Escribe en un archivo |
| fscanf() | Lee de un archivo |
| feof() | Devuelve verdadero si se alcanza la marca EOF (fin del archivo) |
| ferror() | Devuelve verdadero si a ocurrido un error |
| rewind() | Pone el localizador de posición de archivo al comienzo |
| fflush() | Vacía un archivo |
| remove() | Borra 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()**.

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 archivos. 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 archivo y algunos campos para uso interno.

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 archivo "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).

**APERTURA DE UN ARCHIVO:**

Para poder operar con un archivo, 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:

* **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 archivo.
* Si la apertura del archivo falla, fopen devuelve un puntero nulo.
* El argumento **filename** (una cadena de caracteres) es el nombre "real" del archivo que va a abrirse mediante fopen (es decir, es el nombre con el que el archivo aparece en el disco).
* El argumento **mode** (una cadena de caracteres): indica qué tipo de operaciones se realizarán sobre el archivo abierto y el tipo de datos que puede contener, de texto o binarios:
* Los posibles valores de mode son:

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

**CIERRE DE UN ARCHIVO**

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

Un valor de retorno cero indica que el archivo 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 archivo que queremos cerrar.

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

**Ejemplo #1: Programa en C para abrir un archivo.**

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

**void main() {**

**FILE \*fp;**

**fp=fopen("archivo.txt","r");**

**if(fp==NULL){  
printf("Error al abrir el archivo para leer");  
exit(1);   
}**

**fclose(fp);**

**}**

**Archivos de texto.**

En el caso particular de los archivos de texto, que es lo que nos interesa en este momento, nos centraremos exclusivamente en los textos puros, generados por ejemplo por el Cuaderno de Notas o los Editores de algún lenguaje de programación.

Las características fundamentales de estos archivos son:

* Sólo pueden almacenar caracteres.
* Están formados por registros que pueden tener cualquier longitud.
* Cada registro finaliza en uno o dos caracteres de control: **CR (Retorno de Carro)** y **LF (Salto de Linea)**.
* No se pueden acceder aleatoriamente.

Las formas de leer un archivo son varias:

* Se pude leer línea a línea sin formato.
* Línea a línea con formato.
* Caracter a caracter.

**IMPORTANTE**:

Hasta este momento hemos estado acostumbrados a que una cadena de caracteres siempre finaliza con '\0'. Pues en un archivo de texto esta situación no se da, ya que hemos dicho que finalizan en el par CR+LF.

**OPERACIONES DE LECTURA Y ESCRITURA EN ARCHIVOS**

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

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

**Las funciones empleadas con archivos en formato texto son:**

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

**Las funciones empleadas con archivos en formato binario son:**

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

**Lectura y escritura con formato de un archivo de texto.**

La función **fscanf ( )** permite la lectura con formatos, y la función **fprintf( )** permite la escritura con formatos.

Su sintaxis puede llegar a ser bien complicada.

El prototipo de estas funciones es la siguiente;

**int fscanf (FILE \*pf, const \*Formatos, DireccDeVariables);**

**int fprintf (FILE \*pf, const \*Formatos, Variables);**

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

**#include <stdio.h>**

**#include <stdlib.h>**

**void main() {**

**FILE \*fpl,\*fp2;**

**char c1,c2;**

**fpl=fopen("archivo.txt","r");**

**if(fpl==NULL)**

**{**

**printf("Error al abrir el archivo para leer");**

**exit(1);**

**}**

**fp2=fopen("copia.txt","w");**

**if(fp2==NULL)**

**{**

**printf("Error al abrir el archivo copia.txt");**

**exit(1);**

**}**

**fscanf (fpl,"%c%c",&c1,&c2);**

**fprintf(fp2,"%c%c",c1,c2);**

**fclose(fpl);**

**fclose(fp2);**

**}**

El formato de dichas funciones se puede complicar bastante, por ejempo, considere la siguiente instrucción:

fscanf(pArch,”%2d%5c%4f”,&entero,cadena,&real);

La misma acepta solamente un entero de 2 dígitos, una cadena de 5 caracteres y un número real que ocupa como máximo cuatro espacios (por ejemplo 2,97; 12,5).

Ahora intente imaginar qué hace la siguiente instrucción:

fscanf(fp1,"%\*[\t\n] %[A-Za-z] %\*[^A-Za-z]%[^\"]",s1,s2);

* %\*[\t\n] 🡪 Almacena los espacios en blancos, como está el símbolo \* indica a la fn que no se guarda en una variable dichos datos
* %[A-Za-z] 🡪Almacena todos los caracteres que sean letras entre “A y Z” y entre “a y z”
* %\*[^A-Za-z] 🡪 Almacena todos los caracteres distintos a letras entre “A y Z” y entre “a y z”, como está el símbolo \* indica a la fn que no se guarda en una variable dichos datos
* %[^\"] 🡪almacena todos los caracteres hasta encontrar una “ (no la incluye a la misma)

**Ejemplo #3: Programa en C que lee con formato**

**un archivo y los copia en otro.**

Seguramente sería más ilustrativo un archivo de texto con algún formato más complicado, por ejemplo como el siguiente:

/\* --- TALLER DE LENGUAJES I 2014

Leer con fscanf archivo de texto conformato.cpp

Este programa lee un archivo escrito con el Cuaderno de Notas

de la siguiente manera:

ENRIQUE SUAREZ 18 72.50 CASADO

INES MARIA RODRIGUEZ 24 51.50 SOLTERA

SOLEDAD VILLAFAÑEZ 37 72.00 CASADA

MARIO ALBERTO GUTIERREZ 22 68.25 CASADA

ROQUE SALOMON KRIEG 48 75.00 SOLTERO

ROMELIA DEL VALLE PEREZ 28 49.50 SOLTERA

en el cual el primer campo ocupa 30 caracteres, el segundo 10,

el tercero 7 y el cuarto hasta el final de la linea.

Notese que en la linea:

fscanf(Fuente,"%30c%10d%7f%s\n",Nombre,&Edad,&Peso,EC);

se han tomado en cuenta estas especificaciones.

En la impresion de salida se ha agregado el simbolo '!'

a fin de visualizar los blancos aidicionales que pudera

tener un campo de texto.

--------------------------------------------------------------- \*/

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

void main()

{

FILE \*Fuente;

char NombFuente[128]="D:\\DISCO F\\Leng214\\Datos en lista.txt";

char Linea[128];

char Nombre[30];

int Edad;

float Peso;

char EC[16];

clrscr();

if((Fuente=fopen(NombFuente,"rt"))==NULL){

cprintf("NO PUDO ABRIR ARCHIVO FUENTE");

getch(); return;

}

while(!feof(Fuente)){

**fscanf(Fuente,"%30c%10d%7f%s\n",Nombre,&Edad,&Peso,EC);**

Nombre[30] = '\0';

cprintf("NOMBRE : %s%c\r\n", Nombre,'!' );

cprintf("EDAD : %d\r\n", Edad );

cprintf("PESO : %2.2f\r\n", Peso );

cprintf("EC : %s%c\r\n", EC,'!' );

cprintf("---------------------------------------\r\n" );

}

fclose(Fuente); getch();

}

**0 29 30 39 40 46 47**

**EC**

**Peso**

**Edad**

**Nombre**

En la instrucción **fscanf( ):**

%30c significa que lee 30 caracteres de corrido y los asigna a Nombre.

%10d lee una magnitud entera con un formato de 10 espacios.

%7f lee una magnitud float con un campo de 7 espacios.

%s una cadena de caracteres hasta el final de la línea.

A través de este programa, los datos han sido exitosamente extraídos y mostrados por pantalla.

Sin embargo si tuviésemos otra disposición de campos, como por ejemplo:

**10**

**19**

**30**

**Superf**

**Cód**

**Dirección**

**Estado**

1021 Av. Alem 2145 258.50 Habitable

1022 San Lorenzo 3520 185.00 Reparaciones

1023 Crisostomo Alvarez 1948 325.50 Excelente

1024 Lamadrid 4200 225.00 Habitable

1025 Amador Lucero 75 450.00 Excelente

Se presenta el inconveniente que al extraer el segundo campo (Dirección), el compilador asume que el primer blanco a partir de Cód pertenece ya a Dirección y ésta queda conformada por un conjunto de blancos anteriores que resulta necesario eliminar.

/\* --- TALLER DE LENGUAJES I 2014

"Leer con fscanf archivo de texto conformato - 02.cpp"

Este programa lee un archivo escrito con el Cuaderno de Notas

de la siguiente manera:

1021 Av. Alem 2145 258.50 Habitable

1022 San Lorenzo 3520 185.00 Reparaciones

1023 Crisostomo Alvarez 1948 325.50 Excelente

1024 Lamadrid 4200 225.00 Habitable

1025 Amador Lucero 75 450.00 Excelente

en el cual el primer campo ocupa 10 caracteres, el segundo 30,

el tercero 12 y el cuarto hasta el final de la línea.

Nótese que en la línea:

**fscanf(Fuente,"%10d%30c%19f%s\n",Nombre,&Edad,&Peso,EC);**

se han tomado en cuenta estas especificaciones.

En la impresión de salida se ha agregado el simbolo '!'

a fin de visualizar los blancos adicionales que pudiera

tener un campo de texto.

IMPORTANTE:

------------

El segundo campo: Direccion[ ] es de tipo char, precedido por

un campo numérico que finaliza varios caracteres ANTES de co-

menzar este. El inconveniente que crea esta situación es que

al buscar el segundo campo, la búsqueda comienza en la inme-

diata terminación del anterior, apareciendo un conjunto de

blancos previos a la dirección en sí.

Para paliar esta situación se ha implementado la función

**EliminarBlancos( )** que recibe la cadena Direccion y la con-

vierte en una cadena normalizada que empiece en **[0]** y posea

un **'\0'** al final de la misma.

--------------------------------------------------------------- \*/

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void EliminarBlancos ( char \* , int );

void main()

{

FILE \*Fuente;

char NombFuente[128] = "D:\\DISCO F\\Leng214\\Lista de propiedades.txt";

char Linea[128];

char Direccion[42];

int Cod;

float Superf;

char Estado[42];

clrscr();

if((Fuente=fopen(NombFuente,"rt"))==NULL){

cprintf("NO PUDO ABRIR ARCHIVO FUENTE");

getch(); return;

}

while(!feof(Fuente)){

fscanf(Fuente,"%10d%30c%19f%s\n",&Cod,Direccion,&Superf,Estado);

Direccion[30]='\0';

EliminarBlancos(Direccion,30);

cprintf("Codigo : %d\r\n", Cod );

cprintf("Direccion : %s%c\r\n", Direccion,'!' );

cprintf("Superficie : %2.2f\r\n", Superf );

cprintf("Estado : %s%c\r\n", Estado,'!' );

cprintf("-------------------------------------------\r\n" );

}

fclose(Fuente); getch();

}

// --------------------------------------------------------------

void EliminarBlancos(char \*Cad, int DIM)

{

int i;

int j=0;

// --- elimina blancos finales ----

for(i=DIM-1;Cad[i]==' ';i--);

Cad[i+1]='\0';

// --- elimina blancos iniciales ---

i=0;

while(Cad[i]==' ') i++;

while(Cad[i]) Cad[j++]=Cad[i++];

Cad[j]='\0';

}

Quizás le haya pasado desapercibido, pero el último campo, que también es una cadena de caracteres, no presenta el inconveniente del campo Direccion para ser extraído, o sea no toma los blancos iniciales del campo numérico anterior. Misterio.

La conclusión es que **fscanf( )** es bastante útil para extraer texto con formato y asignarlo a variables adecuadas, pero deben tomarse algunas precauciones para que todo salga bien. Por ejemplo en el caso de **Dirección**, a su variable receptora debemos darle un espacio mayor en bytes que lo que se supone que es el campo (debido a los caracteres previos que toma del campo anterior). Podríamos pensar lo mismo del campo Estado y así sucesivamente.

Sin embargo existe una forma mucho más confiable de leer archivos con formatos es lo que desarrollaremos a continuación.

/\* --- TALLER DE LENGUAJES I 2014

"Leer con formato y fgets.cpp"

Este programa rescata cada cadena del archivo de texto utilizando la

función **fgets(Linea,n,FILE \* )**

Luego rescata cada subcadena, o sea cada campo en variables cadenas de

longitud adecuada (los caracteres numéricos también se toman como sub-

cadenas).

Finalmente los campos que sean numéricos se convierten a su tipo original

mediante las funciones de conversión **atoi( )** y **atof( ).**

Para hacer este código más práctico y reutilizable, se ha sobrecargado

un poco con variables adicionales como el byte de comienzo de cada sub-

cadena dentro de la **Linea**, como así también la longitud de cada subcadena.

**NOTA:** Recordar siempre que los arreglos comienzan en el domicilio [0]

de manera que si un campo posee longitud 10, comienza en [0] y

finaliza en [9], y así sucesivamente.

---------------------------------------------------------------------------------------------------------------- \*/

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void EliminarBlancos ( char \* , int );

// -------------------------------------------------------------

void main()

{

FILE \*Fuente;

char NombFuente[128]="D:\\DISCOF\\Leng214\\Lista de propiedades.txt";

char Linea[128];

// -- cadenas para cada campo

char Direccion[42];

char StrCod[12];

char StrSuperf[19];

char Estado[42];

int Cod;

float Superf;

// -- comienzo de cada campo

int C1erCampo = 0;

int C2doCampo = 10;

int C3erCampo = 40;

int C4toCampo = 59;

// -- longitud de cada campo

int L1erCampo = 10;

int L2doCampo = 30;

int L3erCampo = 19;

int L4toCampo = 16;

clrscr();

if((Fuente=fopen(NombFuente,"rt"))==NULL){

cprintf("NO PUDO ABRIR ARCHIVO FUENTE");

getch(); return;

}

while(fgets(Linea,128,Fuente)!=0){

Linea[strlen(Linea)-1]='\0';

strncpy ( StrCod, &Linea[C1erCampo], L1erCampo );

strncpy ( Direccion, &Linea[C2doCampo], L2doCampo );

strncpy ( StrSuperf, &Linea[C3erCampo], L3erCampo );

strncpy ( Estado, &Linea[C4toCampo], L4toCampo );

StrCod [L1erCampo-1] = '\0';

Direccion[L2doCampo-1] = '\0';

StrSuperf[L3erCampo-1] = '\0';

EliminarBlancos ( StrCod, L1erCampo-1);

EliminarBlancos ( Direccion, L2doCampo-1);

EliminarBlancos ( StrSuperf, L3erCampo-1);

// --- monitoreo para control ---

cprintf("%s%c\r\n",StrCod,'!');

cprintf("%s%c\r\n",Direccion,'!');

cprintf("%s%c\r\n",StrSuperf,'!');

cprintf("%s%c\r\n",Estado,'!');

cprintf("-----------------------------------\r\n");

// --- convierte cadenas a valores numericos

Cod=atoi(StrCod);

Superf=atof(StrSuperf);

textcolor(LIGHTRED);

cprintf("Codigo = %d\r\n",Cod);

cprintf("Superf = %4.2lf\r\n",Superf);

textcolor(WHITE);

cprintf("----------------------------------\r\n");

}

fclose(Fuente);

getch();

}

// -----------------------------------------------------------

void EliminarBlancos(char \*Cad,int DIM)

{

int i;

int j=0;

// --- elimina blancos finales ----

for(i=DIM-1;Cad[i]==' ';i--);

Cad[i+1]='\0';

// --- elimina blancos iniciales ---

i=0;

while(Cad[i]==' ') i++;

while(Cad[i]) Cad[j++]=Cad[i++];

Cad[j]='\0';

}

// -------------------------------------------------------------

Las funciones de conversión utilizadas son:

**int atoi(const char \*s)**  convierte la cadena "s" a un entero.

**double atof(const char \*s)** convierte la cadena "s" a un double.

En caso de éxito regresan el valor numérico correspondiente.

En caso de fallo regresan 0.

Para la extracción de las subcadenas se ha utilizado la función:

**char \*strncpy(char \*Destino, const char \*Fuente, nro\_de\_caracteres)**

donde Destino es la cadena char que recibirá el total de caracteres extraídos de la cadena Fuente (Linea en nuestro caso) a partir de la dirección &Linea[ ] y en un total de n caracteres (según la longitud del campo a extraer).

En conclusión, este es un método simple en su concepción aunque tal vez un poquito largo. En contrapartida es muy confiable y sistemático.

Otro detalle importante es que hemos utilizado la propia funcion **fgets( )** como detectora del final de archivo (en lugar de feof( ) ). Es más confiable y funciona muy bien.

**Lectura de un archivo caracter a caracter.**

Una lectura de este tipo algunas ventajas interesantes. Para ello utilizaremos la función:

int fgetc(FILE \*)

regresa el próximo caracter leído del archivo.

* En caso de éxito retorna el caracter leído.
* En caso de fallo retorna EOF.
* En caso de final de archivo regresa EOF.

Cuando leemos un archivo de texto línea a línea, suele ocurrir que los tabuladores (ASCII=9) entran a formar parte de la línea extraída. Mediante fgetc( ) pueden filtrarse caracteres no deseables (o convertirse a lo que en el texto representan).

El siguiente código fuente ilustra esta situación:

/\* --- TALLER DE LENGUAJES I 2014 -

Leer archivo de texto mejorado.cpp

Este programa lee un archivo de texto a sabiendas de que cada línea finaliza en el

par CR+LF. Por eso es que la funcion Leer Archivo( ) lee caracter a caracter y detecta

cuando dicho par se halla presente tomando los recaudos del caso.

--------------------------------------------------------------------------------------------------------------- \*/

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int LeerLinea ( FILE \*, char \* );

// ----------------------------------------------------------------------------------------------------------------

void main()

{

FILE \*Fuente;

char NombFuente[128] = "D:\\DISCO F\\Leng214\\Archi08.cpp";

char Linea[128];

clrscr();

if((Fuente=fopen(NombFuente,"rt"))==NULL){

cprintf("NO PUDO ABRIR EL ARCHIVO FUENTE");

getch(); return;

}

while(LeerLinea ( Fuente, Linea)){

cprintf("%s\r\n",Linea);

getch();

}

fclose(Fuente);

}

// --------------------------------------------------------------------------------------------------------------

int LeerLinea(FILE \*pF,char \*Linea)

{

const CR = 13;

const LF = 10;

const TAB = 9;

char c;

int i = 0;

int j;

while((c=fgetc(pF))!=EOF) {

switch(c) {

case CR : break;

case LF : Linea[i]='\0';

return(1);

case TAB : for(j=0;j<4;j++) Linea[i+j]=' ';

i=i+j;

break;

default : Linea[i]=c; i++;

}

}

return(0);

}

// ----------------------------------------------------------------------------------------------------------

La función **LeerLinea( )** detecta además cuando el archivo ha sido leído completamente y en ese caso retorna 0.

* Realiza lecturas línea a línea sabiendo que cada línea de texto finaliza en el par **CR+LF** obvia el **CR** y en lugar del **LF** coloca un finalizador **'\0'**.
* Si encuentra en caracter **TAB** (ASCII=9) agrega 4 espacios en la cadena.

**Escribir un archivo de texto con formato.**

Ahora vamos a realizar un proceso inverso: partiendo de los datos individuales grabar un archivo de texto con formato.

Para ello es conveniente que analicemos una función sumamente útil para este evento:

**int sprintf(char \*Buff, const char \*Formatos,ListaDeVariables)**

En caso de éxito retorna el número de bytes de la cadena.

En caso de error, retorna EOF.

Esta instrucción no incluye el finalizador '\0' al final de la cadena, por lo cual tendremos que agregarlo en la cadena de formatos.

Nada mejor que un buen ejemplo para aclarar dudas:

/\* --- TALLER DE LENGUAJES I 2014

Este programa genera una tabla con la función matemática:

F(x) = A.[Sen(x)/x]

y lo graba como un archivo de texto con formato utilizando la función:

**sprintf(Cad,"...formatos...",Variables).**

La idea es importar luego esta Tabla desde Excel y proceder a su graficación.

**IMPORTANTE:** Excel no acepta el punto como decimal, sino la coma.

Por lo tanto deben convertirse a dicho símbolo.

------------------------------------------------------------------------------------------------------- \*/

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

// -------------------------------------------------------------------------------------------------------------------

void main()

{

double A = 10;

char Linea[128];

FILE \*Tabla;

char NombTabla [128] = "D:\\DISCO F\\Leng214\\Tabla.dat";

double Krad = M\_PI/180;

double Fx;

int x;

int x\_ini = -360;

int x\_fin = 360;

int Paso\_x = 10;

int i;

clrscr();

if((Tabla=fopen(NombTabla,"wt"))==NULL){

cprintf("NO PUDO CREAR ARCHIVO DE DATOS");

getch(); return;

}

for(x=x\_ini;x<=x\_fin;x+=Paso\_x){

if(x==0) Fx=A;

else Fx=A\*(sin(x\*Krad))/(x\*Krad);

sprintf(Linea,"%10d%10.2lf\n",x,Fx); // crea el formato.

// --- reemplaza el (.) decimal por la (,) p/que Excel lo acepte.

for(i=0;i<strlen(Linea);i++)

if(Linea[i]=='.') Linea[i]=',';

fprintf(Tabla,"%s",Linea); // lo graba en disco.

cprintf("%s\r",Linea); // lo muestra por pantalla.

}

fclose(Tabla);

cprintf("TABLA FINALIZADA....");

getch();

}

Este programa genera una Tabla de valores para luego, desde Excel, importarla y graficarla con todas las facilidades que éste posee. La función matemática para la generación la célebre:

**F(x) = Seno(x) / x**

para un rango de valores comprendido entre -360º a +360º.

Obviamente las funciones trigonométricas requieren argumentos en radianes, por lo cual definimos Krad = M\_PI / 180.

En definitiva, nuestras variables son el ángulo **x** (int) en grados, y **Fx** de tipo double.

La línea:

**sprintf(Linea,"%10d%10.2lf\n",x,Fx);**

define perfectamente cómo quedará configurada cada línea a imprimir en el archivo:

* Un entero en un campo de 10 espacios.
* Un double en un campo de 10 espacios y con 2 decimales.
* Un finalizador de cadena '\0'.

Un detalle importante es que EXCEL no admite los (.) como indicador decimal, sino la coma (,). Ello lo solucionamos de manera muy sencilla:

**for(i=0;i<strlen(Linea);i++)**

**if(Linea[i]=='.') Linea[i]=',**';

Finalmente el volcado de Linea hacia el archivo destino viene dada por:

**fprintf(Tabla,"%s",Linea);**