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

Archivos binarios.

* En los archivos de texto teníamos registros formados por una cierta cantidad de caracteres alfanuméricos y finalizados en el par CR-LF (o sea **Retorno de Carro** y **Salto de Línea)** brindando el lenguaje un conjunto de funciones para el manejo cómodo de dichos archivos.
* En los archivos binarios el concepto de dato se pierde, pues ahora sólo tenemos un flujo de bytes hacia y desde un dispositivo de almacenamiento.
* Habitualmente se piensa en los archivos binarios como una secuencia de bytes.
* Los archivos binarios suelen ser interpretados como alguna cosa que no sean caracteres de texto. Un ejemplo típico son los programas de ordenador compilados; de hecho, las aplicaciones o programas compilados son conocidos como binarios, especialmente entre los programadores. Pero un archivo binario puede almacenar imágenes, sonido, versiones comprimidas de otros archivos, etc. en pocas palabras, cualquier tipo de información.

**Algunos archivos binarios tienen una cabecera**.

Esta cabecera es un **bloque de metadatos** que un programa informático usará para interpretar correctamente la información contenida. Por ejemplo, un archivo GIF puede consistir en múltiples imágenes y la cabecera se usa para identificar y describir cada bloque de datos de cada imagen.

Si el archivo binario no tiene cabecera se dice que **es un archivo binario plano**

Para abrir un archivo en forma binaria y para lectura, haremos lo siguiente.

if((Archi=fopen(NombArchi,"rb")==NULL) {

cprintf("No pudo abrir el archivo fuente");

getch( ); return;

}

y para abrirlo en modo escritura:

if((Archi=fopen(NombArchi,"wb")==NULL) {

cprintf("No pudo abrir el archivo fuente");

getch( ); return;

}

Siempre es conveniente chequear si la operación fue exitosa.

También es válido aquí todo lo dicho en el modo texto en cuanto a la actualización de archivos, salvo que ahora va la letra 'b' y no la 't'. Por ejemplo:

ab

rb+

wb+

etc.

Existen dos funciones claves para el manejo de archivos binarios:

**int fread(void \*Donde\_Guarda, int Tam\_Bloque, int NroDeBloques, FILE \*stream)**

**int fwrite(void \*Desde\_Saca, int Tam\_Bloque, int NroDeBloques, FILE \*stream)**

Si estamos leyendo - **fread( )** - el primer parámetro indica la dirección a partir de la cual se almacenarán los bytes extraídos del archivo. En cambio si estamos escribiendo -**fwrite( )** - indicará la dirección desde dónde se extraerán los datos a grabar en el archivo.

El elemento de programa que recibe o emite datos puede ser desde una variable simple hasta arreglos y estructuras complejas de datos.

Estas dos instrucciones reciben o emiten paquetes de bytes de la siguiente manera:

tamaño de cada bloque de una cantidad de bloques

El tamaño de bloque es el sizeof() de la estructura, y la cantidad de bloques es la cantidad de estructuras a ir guardando

Un ejemplo aclarará bien esta idea:

const DIM = .....

typedef struct TDatos { ............. };

TDato Datos[DIM];

.

.

.

for(i=0;i<DIM;i++)

fread(&Datos[i],sizeof(TDatos),1,Fuente);

O sea vamos extrayendo **sizeof(TDatos)** bytes - tamaño del registro - y lo vamos almacenando en cada domicilio del vector de estructuras.

En este caso particular **TODOS** los registros lógicos poseen la misma extensión en bytes. Pero no siempre es así. Una gran ventaja de tener todos los registros de igual tamaño, es que podemos acceder aleatoriamente a cualquiera de ellos ya que en el disco se comportan como un arreglo, disponiendo C de instrucciones precisas para acceder a cualquier posición remota. Veremos esto más adelante con un ejemplo.

Ejempo #1

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

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void main ()

{

FILE \*pf;

int i;

int leidos;

int v1[6]={1,3,5,7,9,11};

int v2[6];

pf = fopen ("Datos.dat", "wb+");//abro el archivo de lectura y escritura

fwrite(v1,sizeof(int),6,pf);//grabo los 6 nros en el archivo

rewind(pf);

leidos=fread(v2,sizeof(int),6,pf);

cprintf("Los datos leidos en disco son:%d\r\n",leidos);

for(i=0; i<6; i++)

cprintf("%d ",v2[i]);

fclose(pf);

}

Ejempo #2

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

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

typedef struct persona

{

char Nombre[30];

char SegundoNombre[30];

char Apellido[30];

int Edad;

}TPersona;

void main ()

{

FILE \*pf;

int i;

int leidos;

TPersona per2;

TPersona per1={"Sergio","Antonio","Guardia",37};

cprintf("%s\r\n",per1.Nombre);

cprintf("%s\r\n",per1.SegundoNombre);

cprintf("%s\r\n",per1.Apellido);

cprintf("%d\r\n",per1.Edad);

pf = fopen ("DatosPersonas.dat", "wb+");//abro el archivo de lectura y escritura

fwrite(&per1,sizeof(TPersona),1,pf);

per1.Edad=31;

fwrite(&per1,sizeof(TPersona),1,pf);

per1.Edad=33;

fwrite(&per1,sizeof(TPersona),1,pf);

rewind(pf);

leidos=fread(&per2,sizeof(TPersona),1,pf);

while(!feof(pf))

{

cprintf("Los datos leidos en disco son:%d\r\n",leidos);

cprintf("%d\r\n",per2.Edad);

leidos=fread(&per2,sizeof(TPersona),1,pf);

getch();

}

fclose(pf);

}

En archivos binarios existen funciones adicionales que pueden ser muy útiles:

|  |  |
| --- | --- |
| Nombre | Función |
| fseek() | Salta al byte especificado en un archivo |
| ftell() | Retorna a cuántos bytes del origen se encuentra el puntero del archivo |
| putw() | Escribe un nro. entero en un archivo |
| getw() | Lee un nro. entero de un archivo |

**Función** **fseek():**

Esta función sirve para situar el cursor del fichero para leer o escribir en el lugar deseado.

El valor de retorno es cero si la función tuvo éxito, y un valor distinto de cero si hubo algún error.

**Sintaxis**

**int fseek(FILE \*pArchivo, long int desplazamiento, int origen);**

Los parámetros de entrada son: un puntero a una estructura FILE derchivo en el que queremos cambiar el cursor de lectura/escritura, el valor del desplazamiento y el punto de origen desde el que se calculará el desplazamiento.

El parámetro origen puede tener tres posibles valores:

1. **SEEK\_SET** el desplazamiento se cuenta desde el principio del fichero. El primer byte del fichero tiene un desplazamiento cero.

2. **SEEK\_CUR** el desplazamiento se cuenta desde la posición actual del cursor.

3. **SEEK\_END** el desplazamiento se cuenta desde el final del fichero.

**Función ftell():**

La función ftell sirve para averiguar la posición actual del cursor de lectura/excritura de un fichero.

**Sintaxis:**

**long int ftell(FILE \*pArchivo);**

El valor de retorno será esa posición, o -1 si hay algún error.

El parámetro de entrada es un puntero a una estructura FILE del Archivo del que queremos leer la posición del cursor de lectura/escritura.

**Función** **putw():**

Escribe un nro. entero en un archivo.

Esta función no espera ni genera ninguna alineación especial en el archivo.

**Sintaxis:**

**int putw ( int w, FILE \*pArchivo );**

**Exito --> retorna el entero w.**

**Fallo --> retorna EOF (valor -1)**

**Función** **getw():**

Lee un nro. entero en un archivo generado con **putw**

**Sintaxis:**

**int getw( FILE \*stream);**

**Exito --> retorna el entero w.**

**Fallo --> retorna EOF (valor -1)**

Ejemplo #4

Este programa genera una lista de enteros en un archivo y luego los lee utilizando las fns getw() y putw():

Para detectar el final de archivo convienen utilizar la función

**feof(FILE \*pArchivo)**

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

#include <conio.h>

#include <stdlib.h>

#include <stdio.h>

void main( )

{

const TOT\_NUM = 128;

FILE \*Archivo;

FILE \*Fuente;

char NombArchivo[128] = "putw\_getw.dat";

int NumGen;

int i;

clrscr(); randomize();

if((Archivo=fopen(NombArchivo,"wb+"))==NULL) {

cprintf("NO PUDO GENERAR EL ARCHIVO");

getch( ); return;

}

for(i=0;i<TOT\_NUM;i++)

{

NumGen=100+random(900);

putw(NumGen,Archivo);

}

rewind(Archivo);

while(!feof(Archivo))

{

NumGen = getw(Archivo);

if(NumGen>0)

{

cprintf("%4d",NumGen);

}

}

fclose(Archivo);

}

Ejemplo #5

**Cargar una matriz con una tabla creada con putw**

Este programa genera una lista de enteros en formato binario utilizando la función putw( ), con los cuales se piensa posteriormente alimentar una matriz 2D de 12 x 10. Esto quiere decir que la lista deberá contener 12x10=120 elementos enteros. Una vez creada la lista, se vuelve al inicio del archivo y se lee los datos. La carga de la matriz se realiza leyendo cada elemento con **getw( )** y asignándolo a cada elemento M[ i ][ j ]. Finalmente se muestra por pantalla.

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

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

const DIM1 = 12;

const DIM2 = 10;

typedef int TMat[DIM1][DIM2];

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

void main()

{

FILE \*pArchivo;

char NombArchivo[128] = "Lista\_Num.dat";

TMat M;

int NumGen;

int i, j;

clrscr();

if((pArchivo=fopen(NombArchivo,"wb+"))==NULL) {

cprintf("NO PUDO GENERAR ARCHIVO DESTINO");

getch( ); return;

}

for(i=0;i<DIM1\*DIM2;i++) {

NumGen=100+random(900);

putw(NumGen,pArchivo);

}

rewind(pArchivo);

for(i=0;i<DIM1;i++) {

for(j=0;j<DIM2;j++) {

M[ i ][ j ] = getw(pArchivo);

cprintf("%4d",M[ i ][ j ]);

}

cprintf("\r\n");

}

fclose(pArchivo);

getch();

}

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

En realidad obtenemos un archivo secuencial donde todos los valores numéricos se hallan uno a la par del otro. La lectura también es secuencial, pero no la asignación de estos valores puesto que los lazos for se encargan de distribuir correctamente las magnitudes leídas.

Ejemplo #6

Convertir un archivo de texto con formato en un archivo binario

Este programa lee un archivo de texto con formato utilizando la función **fscanf( )** y lo convierte en un archivo binario mediante la función **fwrite( )**. Como elemento intermediario emplea una estructura (que será el registro lógico) y la irá almacenando para cada registro leído. Es decir, ahora tendremos un nuevo registro lógico (siempre de la misma medida), que será la estructura.

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

Este programa trabaja sobre un archivo ya conocido por nosotros:

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

1026 Pje Lavalle 2447 250.00 Modificaciones

1027 Av. Ernesto Padilla 568 315.00 Excelente

1028 Italia 2200 272.00 Regular

1029 Pje Virrey Vertiz 1235 285.00 Excelente

1030 Asuncion 1240 180.00 Excelente

1031 Bolivia 3518 125.00 Reparaciones

1032 Lamadrid 2560 100.00 Habitable

1033 Constitucion 450 480.00 Excelente

1034 Baltazar Aguirre 385 295.00 Reparaciones

1035 Rondeau 628 150.00 Habitable

1036 Alsina 2320 280.00 Excelente

1037 Pueyrredon 520 110.00 Habitable

1038 Colombia 3485 158.00 Reparaciones

1039 Guatemala 618 260.00 Regular

1040 Pje Zantillan 3125 290.00 Excelente

1041 Mejico 350 168.00 Regular

1042 Chacabuco 325 200.00 Reparaciones

1043 Ayacucho 156 350.00 Regular

1044 Bolivar 963 215.00 Regular

1045 Jujuy 420 380.00 Excelente

1046 Las Piedras 932 60.00 Excelente

1047 Lincoln 530 90.00 Reparaciones

1048 Gorriti 180 240.00 Regular

1049 Agustin Masa 340 218.00 Excelente

1050 Viamonte 968 114.00 Reparaciones

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

#include <conio.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <alloc.h>

typedef struct { int Codigo;

char Domicilio[30];

float SupCub;

char Estado[16];

} TDatos;

void \*ReservarMemoria(int TotBytes);

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

void main()

{

TDatos \*Datos;

FILE \*Fuente;

FILE \*Destino;

char NombFuente [128] = "listado de propiedades.txt";

char NombDestino[128] = "listado de propiedades.bin";

clrscr();

Datos=(TDatos \*)ReservarMemoria(sizeof(TDatos));

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

cprintf("NO PUDO ABRIR ARCHIVO FUENTE");

getch(); return;

}

if((Destino=fopen(NombDestino,"wb"))==NULL) {

cprintf("NO PUDO CREAR ARCHIVO DESTINO");

fclose(Fuente);

getch(); return;

}

while(fscanf(Fuente,"%d%\*[^A-Za-z]%30c%f%\*[^A-Za-z]%s\n",&Datos->Codigo,Datos->Domicilio,&Datos->SupCub,Datos->Estado)!=EOF)

{

cprintf("Codigo : %d\r\n", Datos->Codigo);

cprintf("Domicilio : %s\r\n", Datos->Domicilio);

cprintf("SupCubierta : %6.2f\r\n", Datos->SupCub);

cprintf("Estado : %s\r\n", Datos->Estado);

fwrite(Datos,sizeof(TDatos),1,Destino);

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

getch();

}

fclose(Fuente);

fclose(Destino);

getch();

}

void \*ReservarMemoria(int TotBytes)

{

void \*pAux;

if((pAux=malloc(TotBytes))==NULL) {

cprintf("No pudo reservar memoria dinámica");

getch( ); exit(1);

}

return(pAux);

}

El tramo de código:

while(fscanf(Fuente,"%d%\*[^A-Za-z]%30c%f%\*[^A-Za-z]%s\n",&Datos->Codigo,Datos->Domicilio,&Datos->SupCub,Datos->Estado)!=EOF)

utiliza los trucos ya explicados en la clase pasada para poder leer correctamente archivos de texto con formatos y extrae, o asigna, a variables individuales del tipo adecuado. Estas variables son, obviamente, cada uno de los miembros de la estructura que constituirá cada registro lógico del archivo binario (todos de la misma longitud).

La instrucción:

**fwrite(Datos,sizeof(TDatos),1,Destino);**

es la encargada de guardar cada registro en el archivo binario.

Aquí se ve mucho mejor lo que dijimos en un comienzo: tomar como tamaño de bloques **sizeof(TDatos)** bytes y guardar de a 1 registro.

Ejemplo #7

Leer **aleatoriamente un archivo binario**, o sea saltando a registros remotos:

Este programa lee de manera **aleatoria** 4 registros de un archivo binario. Para ello utiliza

**La fn fseek()**

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

#include <conio.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <alloc.h>

typedef struct TDatos { int Codigo;

char Domicilio[30];

float SupCub;

char Estado[16];

} TDatos;

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

void main( )

{

TDatos \*Datos;

FILE \*Fuente;

char NombFuente[128] = "listado de propiedades.bin";

int NroReg[4] = { 3,10,28,0 }; // ... registros a leer aleatoriamente.

int i;

clrscr( );

if((Datos=(TDatos \*)malloc(sizeof(TDatos)))==NULL){

cprintf("NO PUDO RESERVAR MEMORIA DINAMICA");

getch(); return;

}

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

cprintf("NO PUDO ABRIR ARCHIVO FUENTE");

getch(); return;

}

for(i=0;i<4;i++) {

fseek(Fuente,NroReg[i]\*sizeof(TDatos),SEEK\_SET);

fread(Datos,sizeof(TDatos),1,Fuente);

cprintf("Codigo : %d\r\n", Datos->Codigo);

cprintf("Domicilio : %s\r\n", Datos->Domicilio);

cprintf("SupCubierta : %6.2f\r\n", Datos->SupCub);

cprintf("Estado : %s\r\n", Datos->Estado);

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

}

fclose(Fuente); getch();

}

La instrucción:

**fseek(Fuente,NroReg[i]\*sizeof(TDatos),SEEK\_SET);**

Es la encargada de posicionar el puntero de archivo en una posición dada, realiza saltos en **cantidades de bytes**. Es por eso que si se desea ir al registro 4, en realidad tendríamos que hacer **4.sizeof(TDatos)** de lo contrario sólo aterrizaríamos en el byte nro 4.

En cuanto a la lectura de cada registro:

**fread(Datos,sizeof(TDatos),1,Fuente);**

Debe extraerse el paquete completo equivalente a la estructura: **sizeof(TDatos)**.

Otro ejemplo de aplicación de archivos binarios.

/\* ---

Archivo binario de números primos.cpp

Este programa genera una tabla de números primos entre 2 y 32000

y los va almacenando en un archivo binario con la función putw( ).

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

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void main()

{

int Total = 32000;

int i,j;

int EsPrimo;

FILE \*Destino;

char NombDestino[128] = "D:\\DISCO F\\Leng214\\Tabla de Primos.bin";

clrscr();

if((Destino=fopen(NombDestino,"wb"))==NULL) {

cprintf("NO PUDO CREAR ARCHIVO DESTINO");

getch(); return;

}

for(i=2;i<Total;i++) {

EsPrimo=1;

for(j=2;j<=sqrt(i);j++)

if(i%j==0) {

EsPrimo=0;

break;

}

if(EsPrimo) {

cprintf("%20d\r\n",i);

fwrite(&i,1,sizeof(int),Destino);

}

}

fclose(Destino);

cprintf(" ARCHIVO DE PRIMOS GENERADO EXITOSAMENTE...");

getch();

}

Dos modos de determinar la longitud de un archivo.

Vamos a tratar binariamente un archivo del tipo que sea y generado por la aplicación que sea, a fin de determinar su tamaño en bytes. Existen dos formas: una un tanto artesanal en la cual se realiza un proceso de lectura con **fread( ),** que, como sabemos nos devuelve en cada ciclo el número de bytes extraídos, y simplemente llevando una cuenta de estos caracteres. La otra: utilizando funciones más específicas como **fseek( )** y **ftell( )** que nos ahorra varios pasos. Veamos el programa:

Ejemplo #8

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

Este programa determina la longitud en bytes de un archivo de cualquier tipo (binario o de texto). Lo resuelve de dos maneras distintas: una un tanto artesanal recorriendo el archivo hasta el final y la otra utilizando instrucciones más específicas.

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

#include <conio.h>

#include <stdio.h>

int FileSize ( FILE \* );

long TamanioArchivo( FILE\* );

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

void main( )

{

char NombArchi[128] = "listado de propiedades.bin";

FILE \*pArchivo;

clrscr();

if((pArchivo=fopen(NombArchi,"rb"))==NULL) {

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

getch();

}

cprintf("Tamanio de %s : %d\r\n",NombArchi,FileSize(pArchivo));

cprintf("Tamanio de %s : %d\r\n",NombArchi,TamanioArchivo(pArchivo));

getch();

}

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

int FileSize(FILE \*NombArchi)

{

char Buff[1024];

int Cuenta = 0;

int Leidos;

//while((Leidos=fread(Buff,sizeof(Buff),1,NombArchi))!=0)

while((Leidos=fread(Buff,1,sizeof(Buff),NombArchi))!=0)

Cuenta+=(Leidos);

return(Cuenta);

}

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

long TamanioArchivo(FILE \*P\_File)

{

long Pos, TamArch;

Pos = ftell(P\_File); // Guardar el posicion actual del puntero

fseek(P\_File,0, SEEK\_END); // Ir al final del Archivo

TamArch = ftell(P\_File); // Almacenar el Tamaño del Archivo

fseek(P\_File,Pos, SEEK\_SET); // Restablecer puntero

return(TamArch);

}

En la primera función:

**while((Leidos=fread(Buff,1,sizeof(Buff),Aux))!=0) Cuenta+=Leidos**;

los bytes leídos son acumulados en Cuenta, la cual, al final, contendrá un equivalente del total de bytes del archivo.

La segunda función de usuario hace lo siguiente:

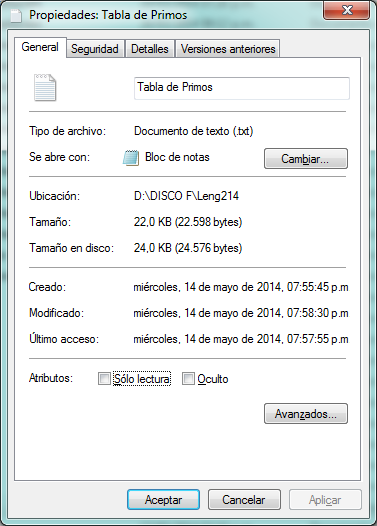
**fseek(Archi,0,SEEK\_END);**

**Tamanio=ftell(Archi);**

envía el puntero de archivo al final del mismo mediante **seek()** y luego solicita que indique la posición de dicho puntero: **ftell().**

fseek( ) dice: vaya al final del archivo (**SEEK\_END**) y tome un desplazamiento 0 mientras que **ftell( )** retorna a cuántos bytes del origen se halla este apuntador.

**NOTA**: Debemos hacer notar un detalle: si hacemos click derecho sobre el archivo evaluado:



se visualizarán dos tamaños:

**El tamaño real.**

**El tamaño en disco.**

Lo que nosotros hemos determinado con nuestro programa es el **tamaño real**. Como el sistema operativo posee un paquete mínimo de acceso en disco que es el "cluster", cuya extensión en bytes se establece en el momento de particionar y formatear el disco, 4096 en nuestro caso, al guardar un archivo de 5 bytes, en **Propiedades** leeríamos:

**Tamaño : 5**

**Tamaño en disco: 4096**

o sea que en archivos pequeños estamos desperdiciando cierta cantidad de bytes.

Implementando un replicador de archivos.

Otra aplicación útil y sencilla de implementar con archivos binarios es un replicador o duplicador de archivos. Simplemente se trata de un proceso sistemático de lectura/escritura hasta finalizar el archivo en cuestión.

/\* --- Replicador.cpp

Este programa replica (copia) un archivo en la misma o en alguna otra

carpeta y lo mismo con el nombre.

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

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

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

void main()

{

FILE \*Fuente;

FILE \*Destino;

char NombFuente [128] = "Travesura.mp3";

char NombDestino[128] = "Travesura2.mp3";

char BuffFuente [32768];

int Leidos;

int i;

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

cprintf("ARCHIVO FUENTE NO ENCONTRADO\r\n");

getch(); return;

}

if((Destino=fopen(NombDestino,"wb"))==NULL) {

cprintf("ARCHIVO DESTINO NO ENCONTRADO\r\n");

getch(); fclose(Fuente); return;

}

while((Leidos=fread(BuffFuente,1,32768,Fuente))!=0)

fwrite(BuffFuente,1,Leidos,Destino);

fclose(Fuente); fclose(Destino);

cprintf("ARCHIVO REPLICADO\r\n");

getch();

}

Las líneas:

**while((Leidos=fread(BuffFuente,1,32768,Fuente))!=0)**

**fwrite(BuffFuente,1,Leidos,Destino);**

Realizan una operación simétrica: la misma cantidad de bytes que lee fread( ) es la que copia o replica fwrite( ).

Comparador de archivos.

/\*

Este programa compara dos archivos de texto: "Texto para lectura.txt" y "Texto para lectura con diferencia.txt", prácticamente iguales, salvo una letra que hemos escrito mal a propósito: una c por una s.

Primero compara las longitudes: si son diferentes asume que los archivos también lo son. Si las longitudes son iguales comienza a leer bloques de tamaño idéntico y compara byte a byte. Mientras tanto va llevando una cuenta del número de caracteres comparados a fin de poder establecer el valor exacto a partir del cual encuentra la primera diferencia.

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

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

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

void main()

{

FILE \*Fuente1;

FILE \*Fuente2;

char NombFuente1[128] = " Texto para lectura.txt";

char NombFuente2[128] = "Texto para lectura con diferencia.txt";

char Buff1[128];

char Buff2[128];

int Tam1,Tam2;

int Leidos;

int ComDeDiff = 0;

int i;

if((Fuente1=fopen(NombFuente1,"rb"))==NULL) {

cprintf("ARCHIVO FUENTE 1 NO ENCONTRADO\r\n");

getch(); return;

}

if((Fuente2=fopen(NombFuente2,"rb"))==NULL) {

cprintf("ARCHIVO FUENTE 2 NO ENCONTRADO\r\n");

getch(); fclose(Fuente1); return;

}

fseek(Fuente1,0,SEEK\_END); Tam1=ftell(Fuente1);

fseek(Fuente2,0,SEEK\_END); Tam2=ftell(Fuente2);

fseek(Fuente1,0,SEEK\_SET); // ... reposiciona al comienzo.

fseek(Fuente2,0,SEEK\_SET); // ... reposiciona al comienzo.

if(Tam1!=Tam2) {

cprintf("POSEEN DISTINTOS TAMANIOS");

getch(); fclose(Fuente1); fclose(Fuente2);

return;

}

while((Leidos=fread(Buff1,1,128,Fuente1))!=0) {

fread(Buff2,1,128,Fuente2);

for(i=0;i<Leidos;i++) {

if(Buff1[i]!=Buff2[i]) {

ComDeDiff+=i;

cprintf("Los archivos son diferentes.\r\n");

cprintf("Las diff se dieron a partir del byte : %d\r\n",ComDeDiff);

cprintf("Caracteres diferentes: %c %c\r\n",Buff1[i],Buff2[i]);

getch ( ); fclose(Fuente1); fclose(Fuente2);

exit(1);

}

}

ComDeDiff+=Leidos;

}

getch();

}