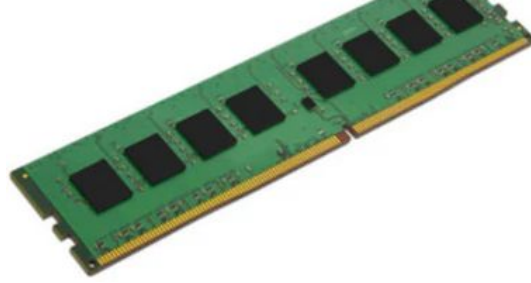


ARCHIVOS DE TEXTO EN C

HASTA AHORA...

Las variables simples, estructuras, vectores o arreglos de N dimensiones nos ayudaron a plantear y solucionar cualquier problema computacional que hemos visto.

NUEVO PROBLEMA...



Lamentablemente dichos problemas quedan resueltos mientras nuestro programa esté en ejecución y mientras los datos, que son utilizados para resolver el problema, se encuentran almacenados en la memoria principal de la misma. Es decir, que los datos del programa no persisten luego de finalizado nuestro programa.

EN RESUMEN:

Cardinalidad Finita: Los arreglos pueden almacenar una cantidad finita de elementos y esa cantidad no puede variar a lo largo del algoritmo. La cantidad está limitada por el tamaño de la memoria.

Persistencia Temporalidad: Los datos de que almacenan los arreglos o vectores, sean de la dimensión que sean, no persisten en el tiempo, solo viven el tiempo que el algoritmo se encuentra ejecutándose en la memoria principal de la computadora.



ARCHIVOS



Nuevo concepto que permitirá persistir
en el tiempo la información de nuestros
programas.

CARACTERÍSTICAS

Cardinalidad infinita: Su tamaño puede variar en tiempo de ejecución del algoritmo, esto virtualmente pueda contener cualquier cantidad de elementos. La limitación está dada por el tamaño del dispositivo, que generalmente es entre 200 y 1000 veces mayor que el tamaño total de la memoria principal de la computadora.

Persistencia Total: La información almacenada dentro de un archivo persiste en el tiempo, es decir que queda almacenada una vez que el algoritmo terminó de hacer uso de la misma. La única forma de perderlos es debido a un error del algoritmo o de un problema del dispositivo de almacenamiento.



Muy lindo
todo, pero
¿cómo uso
archivos en
C?

LA BIBLIOTECA ESTÁNDAR DE C NOS PROVEE DE FUNCIONES PARA MANIPULAR ARCHIVOS



FILE

FILE es un tipo de dato que en lenguaje de programación C representa un archivo abierto.

FOPEN()

```
FILE* fopen (const char* filename, const char* mode);
```

Abre un archivo indicado por el nombre de archivo y devuelve un puntero a FILE (o NULL en caso de error).
mode se utiliza para determinar el modo de acceso a archivos.

Modo	Descripción	Acción si existe	Acción si no existe
r"	read	Open a file for reading	read from start failure to open
"w"	write	Create a file for writing destroy contents	create new
.a"	append	Append to a file write to end	create new
r+"	read extended	Open a file for read/write	read from start error
"w+"	write extended	Create a file for read/write	destroy contents create new
.a+"	append extended	Open a file for read/write	write to end create new

FCLOSE()

```
int fclose (FILE* file);
```

Cierra el archivo dado. Si el puntero a FILE es utilizado una vez que fclose() es ejecutado, el comportamiento en ese caso no está definido.

Devuelve 0 si la operación se realizó con éxito.
EOF (end-of-file) en cualquier otro caso.

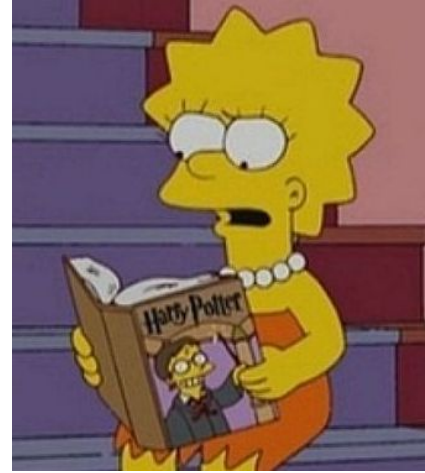
FSCANF()

```
int fscanf (FILE* file, const char* format, ...);
```

Permite leer datos de un archivo.

En caso de éxito, esta función devuelve la cantidad de elementos de entrada que fueron correctamente leídos y asignados; esta cantidad puede ser menor a la esperada, o incluso cero, en caso de que ocurra un fallo temprano en la coincidencia.

Devuelve EOF (end-of-file) en cualquier otro caso.



FORMATOS

Conocidos:

`%i`

`%d`

`%c`

`%f`

`%s`

`%lu`

Personalizados:

`%[^FIN]` => leer hasta un FIN

`%[^\\n]` => leer hasta un '\\n'

`%[^;]` => leer hasta un ';'

`%[^\\0]` => leer hasta un '\\0'

FPRINTF()

```
int fprintf (FILE* file, const char* format, ...);
```

Permite volcar/imprimir/escribir datos en un archivo.

En caso de retorno exitoso, esta función devuelve la cantidad de caracteres impresos (excluyendo el byte nulo utilizado para terminar la salida en cadenas).

Si se encuentra un error de salida, se devuelve un valor negativo.



RENAME()

```
int rename (const char* old_filename , const char*  
new_filename);
```

El archivo cuyo nombre está dado por el valor de `old_name` se renombra con el nombre dado por `new_filename`. Si el valor `new_filename` ya existe el comportamiento depende de la implementación.

Devuelve 0 si la operación se realizó con éxito.

Un valor distinto de cero en cualquier otro caso.

REMOVE()

```
int remove (const char* pathname);
```

Elimina el archivo cuyo nombre se encuentra en pathname.

Devuelve 0 si la operación se realizó con éxito.

-1 en cualquier otro caso.

EJEMPLO CHARLADO:

Tengo la siguiente lista del super en un archivo y quiero saber si está el “Tomate” en ella.

Lechuga

Queso

Tomate

Jabón

Fideos

ACCESO SECUENCIAL

Las funciones para trabajar con archivos en C son de acceso secuencial.

Este tipo de acceso a datos implica que para acceder al n -esimo elemento deseado, debemos pasar por los $(n-1)$ datos anteriores.

ACCESO SECUENCIAL + ¡COSTO!

Acceder a datos que se encuentran en archivos en disco es muuuuuuuucho más costoso (en cálculos y tiempo de procesador) que acceder a datos de memoria ram (stack o heap).

Por lo que SIEMPRE que trabajemos con archivos vamos a tratar de recorrerlos lo menos posible. Es más, vamos a recorrer los archivos una única vez (salvo excepciones).

EJEMPLO CHARLADO:

Tengo la siguiente lista del super en un archivo y quiero saber si está el “Tomate” en ella.

Lechuga

Queso

Tomate

Jabón

Fideos

EJEMPLO CHARLADO:

Tengo la siguiente lista del super en un archivo y quiero saber si está el “Tomate” en ella.

Lechuga

Queso

Tomate

Jabón

Fideos

RESOLUCIÓN:

1) Abro el archivo con ***fopen*** en modo lectura.

2) Leo renglón por renglón con ***fscanf*** con formato ***“%[^\n]\n”*** y me fijo si alguno es “Tomate”.

3) Hago lo que tenga que hacer si hay o no tomate y cierro el archivo con ***fclose***.

EJEMPLO CHARLADO 2:

Tengo la siguiente lista del super en un archivo y quiero eliminar el “Tomate” de ella.

Lechuga

Queso

Tomate

Jabón

Fideos

EJEMPLO CHARLADO 2: RESOLUCIÓN

- 1) Abro el archivo con ***fopen*** en modo **lectura**.
- 2) Abro un archivo auxiliar con ***fopen*** en modo **escritura**.
- 3) Leo renglón por renglón el archivo de lista del super con ***fscanf*** con formato ***“%[^\n]\n”*** y si NO es “Tomate” lo escribo en el archivo auxiliar con ***fprintf*** con formato ***“%s\n”***.
- 4) Cierro ambos archivos con ***fclose***.
- 5) Borro el archivo de lista del super con ***remove***.
- 6) Le cambio el nombre del archivo auxiliar por el de la lista del super con ***rename***.

EJEMPLO CHARLADO 3:

Tengo la siguiente lista del super en un archivo y quiero agregar “Agua” al principio de ella.

Lechuga

Queso

Tomate

Jabón

Fideos

EJEMPLO CHARLADO 3: RESOLUCIÓN

- 1) Abro el archivo con **fopen** en modo **lectura**.
- 2) Abro un archivo auxiliar con **fopen** en modo **escritura**.
- 3) Escribo en el archivo auxiliar “Agua” con **fprintf** con formato **“%s\n”**.
- 4) Leo renglón por renglón el archivo de lista del super con **fscanf** con formato **“%[^n]\n”** y escribo en el archivo auxiliar con **fprintf** con formato **“%s\n”**.
- 5) Cierro ambos archivos con **fclose**.
- 6) Borro el archivo de lista del super con **remove**.
- 7) Le cambio el nombre del archivo auxiliar por el de la lista del super con **rename**.

EJEMPLO CHARLADO 4:

Tengo la siguiente lista del super en un archivo y quiero agregar “Mermelada” al final de ella.

Lechuga

Queso

Tomate

Jabón

Fideos

EJEMPLO CHARLADO 4: RESOLUCIÓN

- 1) Abro el archivo con ***fopen*** en modo **append**.
- 2) Escribo en el archivo auxiliar “Mermelada” con ***fprintf*** con formato ***“%s\n”***.
- 3) Cierro el archivo con ***fclose***.

EJEMPLOS EN CÓDIGO



Coma Separated Values /
Valores Separados por Comas

ARCHIVOS .CSV

Son archivos de texto donde cada línea representa un registro con datos los cuales están separados por un delimitador (‘.’, ‘,’, ‘;’, etc) a elección del creador o la coma (‘,’) en caso de no especificar.

Estos, deben respetar un formato para su correcto manejo.

EJEMPLO:

Tenemos un archivo csv con datos de los alumnos con el siguiente formato:

```
id;apellidos_nombres;padron;corrector
```

Archivo:

```
55;LOPEZ APOLO, PATRICIO XAVIER;115353;Agus B.
```

```
56;LUGO, SANTIAGO IGNACIO;114808;Danny
```

```
58;MAMANI TORREZ, RONNY;114779;Agus S.
```

```
59;MARENZI, SIRO;115626;Tute
```

```
60;MARTINEZ, NICOLÁS;115882;Ani
```

```
61;MASEDA, AGUSTÍN;115214;Caro
```



Fácil, seguro
que stdio de C
también nos
provee de
funciones para
csv

¡NOP!

LA BIBLIOTECA ESTÁNDAR DE C NO NOS PROVEE DE
FUNCIONES EXCLUSIVAS PARA MANIPULAR ARCHIVOS CSV



iNOP!

FUN



CSV

PERO...

Estos tipos de archivos siguen siendo de texto por lo que podemos trabajar con ellos usando las funciones que antes vimos gracias a que podemos determinar el formato de lectura y/o escritura que usamos en las funciones **fscanf** y **fprintf**.

EJEMPLOS EN CÓDIGO