Versión 3.0

800x600 mínimo

HECHO EN MEXICO

En esta lección:

Otras secciones:

Contácteme:

Escritura de un archivo

A lo largo de ésta lección veremos la mecánica necesaria para escribir y leer datos a un archivo, empezaremos con la escritura. Como siempre, los códigos especifican en primer lugar algunas sentencias **#include**, y en el caso concreto del primer código de ejemplo se ha declarado un nuevo tipo de variable. Estudie el siguiente código:

```
#include <stdio.h>
#include <string.h>

int main()
{
    FILE *fp;

    fp = fopen("prueba.htm", "w");   /* Abrir archivo para escritura */
    fprintf(fp, "<HTML> \n");
    fprintf(fp, "<BODY> \n");
    fprintf(fp, "Esta es la primera linea de texto. \n");
    fprintf(fp, "<CENTER>Esta es la segunda"
      "linea</CENTER> \n");
    fprintf(fp, "Y esta es la <B>tercera linea"
      "de texto.</B> \n");

    fclose(fp);    /* Cerrar el archivo antes de terminar el programa */
    printf("Se ha creado el archivo: prueba.htm \n");

    return 0;
}
```

El tipo **FILE** es una estructura (misma que estudiaremos en la siguiente lección) que está definida en el archivo de cabecera **stdio.h**, se usa para definir un puntero que se utilizará en operaciones con archivos. Por definición, C requiere para accesar a un archivo de un puntero de tipo **FILE**, como es normal, se puede utilizar cualquier nombre para representar dicho puntero, es común utilizar **fp**, así que éste nombre utilizamos en el primer código.

Cómo abrir un archivo

Antes de poder escribir datos en un archivo, debemos abrirlo, esto significa que debemos decirle al sistema que deseamos escribir en un archivo especificando el nombre del mismo, para esto utilizamos la función **fopen ( )**, especificada en la línea 8 del código. El puntero de archivo, **fp** en éste caso, señala a la estructura para el archivo siendo necesarios dos argumentos para ésta función, el nombre del archivo en primer lugar, y el atributo del archivo. El nombre del archivo es cualquier nombre válido para su sistema operativo y puede ser expresado sea en minúsculas ó mayúsculas, incluso si así lo desea, como una combinación de ámbas, el nombre se encierra entre comillas. En el ejemplo escogí el nombre **prueba.htm**. Es importante que en el directorio donde trabaje éstos ejemplos no exista un archivo con éste nombre pues al ejecutar el programa se sustituirán los datos del mismo, en caso de no existir un archivo con el nombre especificado, el programa lo creará.

Lectura ("r")

El segundo parámetro es el atributo del archivo y puede ser cualquiera de éstas tres letras, "r", "w", ó "a", y deben estar en letra minúscula. Existen atributos adicionales en C que permiten operaciones de Entrada/Salida (E/S) más flexibles por lo que es recomendable la consulta de la documentación del compilador. Cuando se utiliza "r" el archivo se abre para operaciones de lectura, para operaciones de escritura utilizamos "w" y cuando se especifica "a" es porque deseamos agregar datos adicionales a los ya existentes en el archivo, o sea concatenar datos. Abrir un archivo para lectura implica la existencia del mismo, si ésta condición no es válida el puntero de archivo será igual a NULL y ésto puede ser verificado utilizando el siguiente código:

```
if (fp==NULL)
{
    printf("Error al abrir el archivo \n");
    exit (1);
}
```

Es una buena práctica de programación checar todos los punteros de archivo en una forma similar al código de arriba, el valor de 1 utilizado como parámetro de **exit ( )** será explicado más adelante.

Escritura ("w")

Cuando un archivo se abre para operaciones de escritura, si éste no existe entonces será creado, y si existe será reescrito dando como resultado la pérdida de los datos existentes en el archivo previo. Si ocurre por alguna razón un error al abrir el archivo, el puntero de archivo retorna un valor de NULL que puede ser checado como se especificó arriba.

Concatenar ("a")

Cuando un archivo se abre para concatenar datos, si no existe será creado inicialmente vacío. Si el archivo existe, el punto de entrada de datos se situa al final de los datos existentes en el archivo, de ésta manera es como se agregan nuevos datos al archivo. El puntero de archivo se puede verificar como yá se explicó.

Salida al archivo

La salida de datos hacia un archivo es prácticamente idéntica a la forma en que desplegamos datos en el dispositivo estándar de salida, las únicas diferencias reales son el nombre de una nueva función y la adición del puntero de archivo como uno de los argumentos de la función. En el código de ejemplo, la función **fprintf ( )** reemplaza a la familiar **printf ( )** y el puntero de archivo vá como argumento dentro del paréntesis de la función, como se aprecia en las líneas 9 a la 13 del código de ejemplo.

Cerrando el archivo

Para cerrar un archivo se utiliza la función **fclose ( )** con el puntero de archivo dentro del paréntesis. En algunos programas sencillos no es necesario cerrar el archivo ya que el sistema operativo se encarga de cerrar los archivos que hayan quedado abiertos antes de retornar el control al usuario, sin embargo es buena práctica cerrar en código todo aquel archivo que se abra.
Compile y ejecute el programa, la única salida que verá en pantalla es la línea que indica la creación del archivo especificado, después de correr el programa verifique en su directorio de trabajo la existencia del archivo **prueba.htm**. Por la extensión utilizada es fácil suponer

que se trata de un pequeño archivo web, su navegador lo puede visualizar de la forma convencional, pero también puede abrir éste archivo con un editor de texto común (como Notepad), entonces se dará cuenta que el código HTML está inconcluso, este "problemita" lo resolveremos más adelante por lo que le recomiendo que conserve éste archivo pues se utilizará en las prácticas que siguen.

Volver al principio

Concatenar datos

Como vimos en el programa anterior, el archivo generado llamado **prueba.htm** está inconcluso así que es hora de corregir ésta situación, lo haremos utilizando el código que sigue el cual hace uso del atributo para concatenar datos y además utilizaremos una nueva función para escribir en el archivo un solo dato a la vez:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    FILE *final;

    final = fopen("Prueba.htm", "a"); /* Abrir archivo para concatenar */
    if (final == NULL)
    {
        printf("Falla al abrir el archivo \n");
        exit (EXIT_FAILURE);
    }

    putc('\n', final);
    putc('<', final);
    putc('/', final);
    putc('B', final);
    putc('O', final);
    putc('D', final);
    putc('Y', final);
    putc('>', final);
    putc('\n', final);
    putc('<', final);
    putc('/', final);
    putc('H', final);
    putc('T', final);
    putc('M', final);
    putc('L', final);
    putc('>', final);
    putc('\n', final);

    fclose(final);

    return EXIT_SUCCESS;
}
```

En primer lugar observe que en este programa se efectúa la verificación del éxito al abrir el archivo, la constante llamada **EXIT_FAILURE** está definida en el archivo de cabecera **stdlib.h** generalmente con el valor de 1. La constante llamda **EXIT_SUCESS** a su vez está definida generalmente con el valor de 0. El sistema operativo puede utilizar el valor retornado para determinar si el programa está operando normalmente ó si es necesario tomar alguna acción correctiva, por ejemplo, si un programa se ejecuta en dos partes y la primera de ellas retorna un valor de error, entonces no hay necesidad de ejecutar la

segunda parte del programa.

La función putc ( )

La parte del programa que nos interesa es la función llamada **putc ( )** ejemplificada de la línea 16 a la 32, ésta función extrae al archivo un caracter a la vez, el caracter en cuestión es el primer argumento de la función y el puntero de archivo el segundo y último argumento dentro del paréntesis. Observe que para especificar un caracter determinado se utiliza la comilla sencilla, incluyendo el caso del caracter de retorno de carro **'\n'**. Compile y ejecute el programa. Antes de correr el programa asegurese de la existencia del archivo **prueba.htm** en su directorio de trabajo, generado en el programa anterior, después de correr el programa, abra el archivo con un editor de texto y observe que ahora el documento web está completo.

Volver al principio

Lectura de un archivo

Como ya tenemos un archivo para leer podemos utilizar un nuevo programa, como en los programas anteriores, éste empieza con algunas declaraciones y abriendo el archivo **prueba.htm** especificando que deseamos efectuar operaciones de lectura mediante el atributo "r", el programa ejecuta un bucle **do while** para leer del archivo un sólo caracter a la vez y desplegarlo en pantalla hasta detectar un caracter EOF (End Of File, Fin de Archivo). Por último cerramos el archivo y el programa termina.

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

int main()
{
    FILE *nombre;
    int c;

    nombre = fopen("Prueba.htm", "r");

    if (nombre == NULL)
    {
        printf("El archivo no existe \n");
        exit (EXIT_FAILURE);
    }
    else
    {
        do
        {
            c = getc(nombre); /* Obtiene un caracter del archivo */
            putchar(c); /* Lo despliega en pantalla y continua... */
        }
        while (c != EOF); /* hasta encontrar EOF (el final del archivo)*/
    }
    fclose(nombre);

    return EXIT_SUCCESS;
}
```

En este punto afrontamos un problema común en programación C. La variable regresada por la función **getc ( )** es un caracter, por lo que podemos utilizar para el propósito una variable de tipo **char**, el problema empieza si tratamos de utilizar una variable de tipo **unsigned char**, ya que C regresa -1 para EOF. Una variable de tipo **unsigned char** no

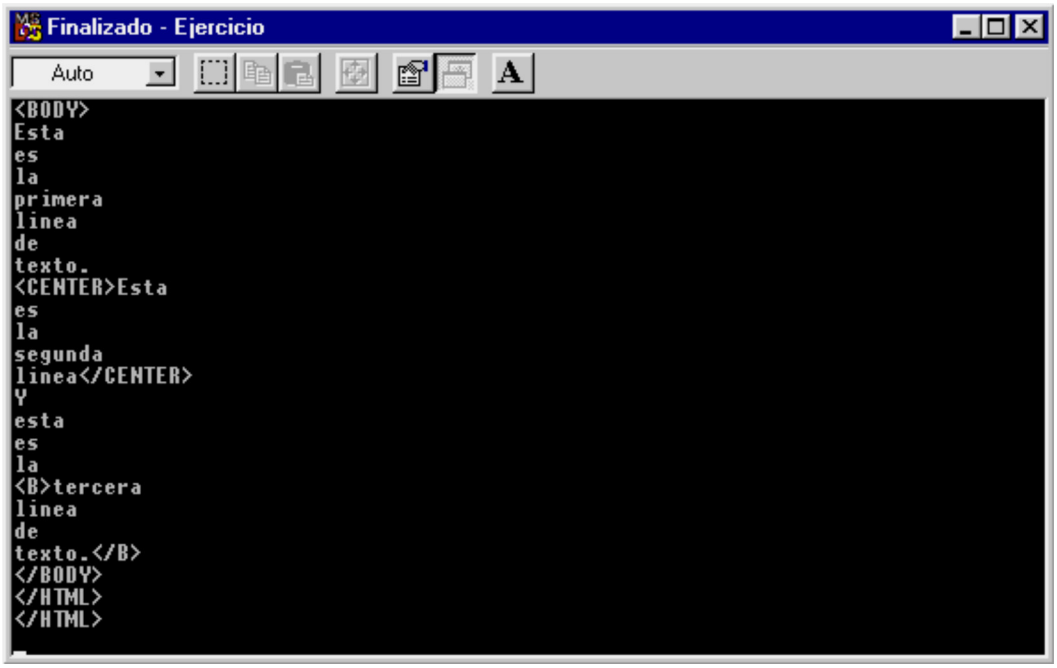
puede contener valores negativos ya que su rango está entre 0 y 255 por lo que retornará 255 para un valor negativo valor que no compara con EOF, en este caso el programa nunca terminará. Para prevenir esta situación, utilice una variable de tipo **int**, ya que este tipo de variable siempre incluye el signo. En este programa leimos del archivo un caracter a la vez, en el siguiente leeremos una palabra a la vez.

Lectura de una palabra

El siguiente programa es prácticamente igual que el anterior excepto que ésta vez se utiliza la función **fscanf ( )** para leer una cadena a la vez, como **fscanf ( )** detiene la lectura de caracteres al encontrar un caracter de espacio ó uno de nueva línea, lee por lo tanto una palabra a la vez desplegando los resultados en una palabra por línea, el nuevo código es:

```
#include <stdio.h>

int main()
{
    FILE *fp1;
    char palabra[100];
    int c;

    fp1 = fopen("Prueba.htm", "r");

    do
    {
        /* Obtiene una palabra del archivo */
        c = fscanf(fp1, "%s", palabra);
        printf("%s\n", palabra); /* la despliega en pantalla */
    }
    while (c != EOF);           /* Se repite hasta encontrar EOF */

    fclose(fp1);

    return 0;
}
```

Al ejecutar éste programa la salida es la siguiente:



El problema es que se imprime dos veces la última palabra, para resolver este detalle modificamos el anterior código así:

```
#include <stdio.h>

int main()
{
    FILE *fp1;
    char palabra[100];
    int c;

    fp1 = fopen("Prueba.htm", "r");

    do
    {
        /* Obtiene una palabra del archivo */
        c = fscanf(fp1, "%s", palabra);
        if (c != EOF)
        printf("%s\n", palabra); /* La despliega en pantalla */
    }
    while (c != EOF);            /* Se repite hasta encontrar EOF */

    fclose(fp1);

    return 0;
}
```

Es bueno hacer notar que un programador experimentado no escribiría el código como lo hicimos en el ejemplo ya que compara **c** con EOF dos veces por cada ciclo del bucle y esto es ineficiente. Utilizamos código que trabaja y es fácil de leer pero conforme Usted gane experiencia en C, Usted utilizará métodos más eficientes de codificar, aunque sean más difíciles de leer, por ejemplo:

```
while((c = fscanf(fp1, "%s", palabra) != EOF)
{
    printf("%s\n", palabra);
}
```

Lectura de una línea

Siguiendo la misma secuencia de los ejemplos de éste capítulo, analizaremos la forma de leer una línea completa de texto, para esto tenemos el código que detallo enseguida:

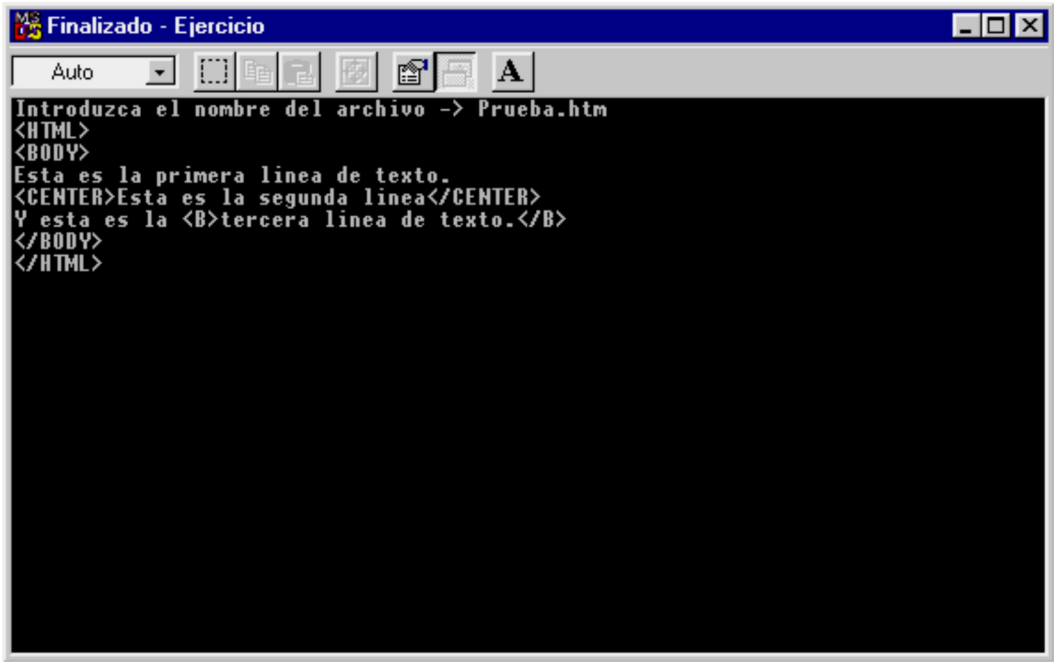
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *fp1;
    char palabra[100];
    char *c;

    fp1 = fopen("Prueba.htm", "r");
    if (fp1 == NULL)
    {
        printf("Error al abrir el archivo \n");
        exit (EXIT_FAILURE);
    }
```

```
    do
    {
        c = fgets(palabra, 100, fp1); /* Obtiene una linea del archivo */
        if (c != NULL)
        printf("%s", palabra);        /* La despliega en pantalla */
    }
    while (c != NULL);                /* Se repite hasta encontrar NULL */

    fclose(fp1);

    return EXIT_SUCCESS;
}
```

Ahora utilizamos la función **fgets ( )** la cual lee una línea completa, incluyendo el caracter de nueva línea y coloca los datos en un buffer (espacio de memoria RAM temporal). El buffer a ser leído es el primer argumento en la llamada a la función en tanto que el máximo número de caracteres a ser leídos es el segundo argumento, seguido por el puntero de archivo. Esta función leerá caracteres en el buffer hasta que encuentre el caracter de nueva línea, ó lea el máximo número de caracteres menos uno, lo que ocurra primero. El espacio final se reserva para el caracter nulo (NULL) del fin de la cadena. Además, si se encuentra un EOF, la función retorna NULL. NULL está definido a cero en el archivo stdio.h
Los ejemplos de éste capítulo realmente no requieren de mucha explicación, el código lo podemos modificar para introducir el nombre del archivo que deseamos abrir de ésta forma:

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

int main()
{
    FILE *fp1;
    char palabra[100], nombre[25];
    char *c;

    printf("Introduzca el nombre del archivo -> ");
    scanf("%s", nombre);        /* Lee el archivo deseado */
    fp1 = fopen(nombre, "r");
    if (fp1 == NULL)
    {
        printf("Error al abrir el archivo \n");
        exit (EXIT_FAILURE);
    }

    do
    {
        c = fgets(palabra, 100, fp1);   /* Obtiene una linea del archivo */
        if (c != NULL)
        printf("%s", palabra);          /* La despliega en pantalla */
    }
    while (c != NULL);                  /* Hasta encontrar NULL */

    fclose(fp1);

    return EXIT_SUCCESS;
}
```

La salida del programa es la siguiente:



Volver al principio

Asignación especial

A lo largo de este capítulo hemos tratado el uso de diferentes funciones para operaciones de lectura y escritura, se trata de un tema particularmente útil en el desarrollo de un programa. Como seguramente habrá notado, utilizamos para los ejemplos un archivo llamado **Prueba.htm**, por la extensión utilizada y por la naturaleza del texto incluído en el mismo sabemos que se trata de un documento web que puede ser visualizado en su navegador. Para terminar este capítulo y a manera de resumen que a la vez nos sirva de introducción al siguiente capítulo, le presento el siguiente código que Yo espero despierte en Usted un poco (ó un mucho) de curiosidad, experimente con el programa y si Usted desea, mándeme su opinión por correo electrónico.

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

enum HTMLid
{
    HTML_NINGUNO,
    HTML_BODY,
    HTML_cBODY,
    HTML_B,
    HTML_cB,
    HTML_HTML,
    HTML_cHTML,
    HTML_CENTER,
    HTML_cCENTER
};

static struct
{
    char    *htmlcodigo;
    enum HTMLid id;
}
lista_de_codigos[]=
```

```
{
    {"<HTML>",      HTML_HTML},
    {"</HTML>",     HTML_cHTML},
    {"<BODY>",      HTML_BODY},
    {"</BODY>",     HTML_cBODY},
    {"<CENTER>",    HTML_CENTER},
    {"</CENTER>",   HTML_cCENTER},
    {"<B>",         HTML_B},
    {"</B>",        HTML_cB},
    {NULL,          HTML_NINGUNO}
};

char        texto[128];
int         itexto=0, c;
int main()
{
    int             i, ietiqueta=0;
    char            etiqueta[64];
    FILE            *archivo;
    enum HTMLid     codigo;

    archivo = fopen("Prueba.htm", "r"); /* Abre el archivo para lectura */

    if (archivo == NULL)
    {
        printf("El archivo no existe...\n");
        exit (EXIT_FAILURE);
    }
    else
    {
        do      /* Checa todos los caracteres del archivo */
        {
            c=getc(archivo);
            if (c=='<')     /* Lee la etiqueta html */
            {
                /* incluye el principio de la etiqueta */
                etiqueta[ietiqueta++]=c;
                do
                {
                    c=getc(archivo);
                    etiqueta[ietiqueta++]=c;
                }
                while(c!='>');
                etiqueta[ietiqueta]=0;
                codigo=HTML_NINGUNO;
                for(i=0; lista_de_codigos[i].htmlcodigo!=NULL; i++)
                {
                    if(stricmp(etiqueta,
                      lista_de_codigos[i].htmlcodigo)==0)
                    {
                        codigo=lista_de_codigos[i].id;
                        break;
                    }
                }
                switch (codigo)
                {
                    case HTML_NINGUNO:
                    break;
                    case HTML_HTML:
                    printf("Empieza el documento web \n");
                    break;
                    case HTML_cHTML:
                    printf("Fin del documento web \n");
                    break;
                    case HTML_B:
                    printf("Empieza la etiqueta B \n");
                    break;
                    case HTML_cB:
                    printf("Termina la etiqueta B \n");
```

```
                    break;
                    case HTML_BODY:
                    printf("Empieza la etiqueta BODY \n");
                    break;
                    case HTML_cBODY:
                    printf("Termina la etiqueta BODY \n");
                    break;
                    case HTML_CENTER:
                    printf("Empieza la etiqueta CENTER \n");
                    break;
                    case HTML_cCENTER:
                    printf("Termina la etiqueta CENTER \n");
                    break;
                }
                ietiqueta=0;
            }
            else
            rollo();
        }
        while(c!=EOF);
    }
    fclose(archivo);
    texto[itexto]=0;
    printf(texto);
    return EXIT_SUCCESS;
}

rollo()
{
    texto[itexto++]=c;
}
```

La salida del programa es la siguiente:

Volver al principio