Lenguaje de Programación: C++ ARCHIVOS I/O

José Luis Alonzo Velázquez

Universidad de Guanajuato

Octubre 2010

Como 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(). El puntero de archivo 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.

IMPORTANTE!!!

Es importante que en el directorio donde trabaje éstos ejemplos no exista un archivo con éste nombre pues al ejecutar el programa podría sustituir los datos del mismo, en caso de no existir un archivo con el nombre especificado, el programa lo creará.

Abrir un archivo

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE *nombre;
    int c;
    nombre = fopen("Prueba.txt", "r");
    if (nombre == NULL){
        printf("El archivo no existe \n");
        exit (EXIT_FAILURE);
    }else{
        do{
            c = getc(nombre); /* Obtiene un carácter del archivo
            putchar(c); /* Lo despliega en pantalla y continua..
        }while (c != EOF); /* hasta encontrar EOF (el final del
    }
    fclose(nombre);
    return EXIT_SUCCESS;
}
```

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:

Código de verificación

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

IMPORTANTE!!!

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 ( ).

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.

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 sitúa 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 debe verificar como se explicó en diapositivas anteriores.

Salida de datos

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.

Salida de datos

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.

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.

Lectura por palabras

```
#include <stdio.h>
int main(){
    FILE *fp1;
    char palabra[100];
    int c;
    fp1 = fopen("Prueba.txt", "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;
}
```

Lectura por palabras

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;
}
```

Lectura por palabras

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 linea

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE *fp1;
    char palabra[100];
    char *c;
    fp1 = fopen("Prueba.txt", "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 carácter 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 carácter de nueva línea, ó lea el máximo número de caracteres menos uno, lo que ocurra primero. El espacio final se reserva para el carácter 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

Abriendo un archivo dado por el usuario

```
#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 a
        if (c != NULL)
        printf("%s", palabra); /* La despliega en pantalla */
    }while (c != NULL); /* Hasta encontrar NULL */
    fclose(fp1);
```

Creado una página web

```
#include <stdio.h>
#include <string.h>
int main(){
    FILE *fp;
    /* Abrir archivo para escritura */
    fp = fopen("prueba.html", "w");
    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");
    /* Cerrar el archivo antes de terminar el programa */
    fclose(fp);
    printf("Se ha creado el archivo: prueba.html \n");
    return 0;
}
```

Terminando página web: Agregando al final

```
#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;
}
```

La función putc()

La parte del programa que nos interesa es la función llamada putc() ejemplificada, ésta función extrae al archivo un carácter a la vez, el carácter 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 carácter determinado se utiliza la comilla sencilla, incluyendo el caso del carácter de retorno de carro '\n'(enter o salto de linea).

 Programming Principles and Practice Using C++, Bjarne Stroustrup.

 http://www.codeblocks.org

 http://www.wxwidgets.org

 (O'Reilly) Practical C Programming (3rd Edition)

 http://www.cplusplus.com