Archivos

Algoritmos y Estructuras de datos

Ing. Pablo Méndez

Consideraciones

Estamos utilizando, en este curso, un subconjunto de instrucciones de C++ para facilitar ciertas características del lenguaje C (como la posibilidad de utilizar el tipo de dato string en vez de char[] cuando es posible, o el uso de cout y cin en vez de printf y scanf).

Con archivos vamos a utilizar, en mayor medida, la forma de utilizarlos en C, con el tipo de dato (más genérico) File*.

En el caso de C++ el uso de archivos se realiza a través de las clases ifstream y ofstream, para flujos de entrada y flujos de salida respectivamente. Si bien veeremos estas posibilidades, haremos más uso de los File*.

Flujos (Streams)

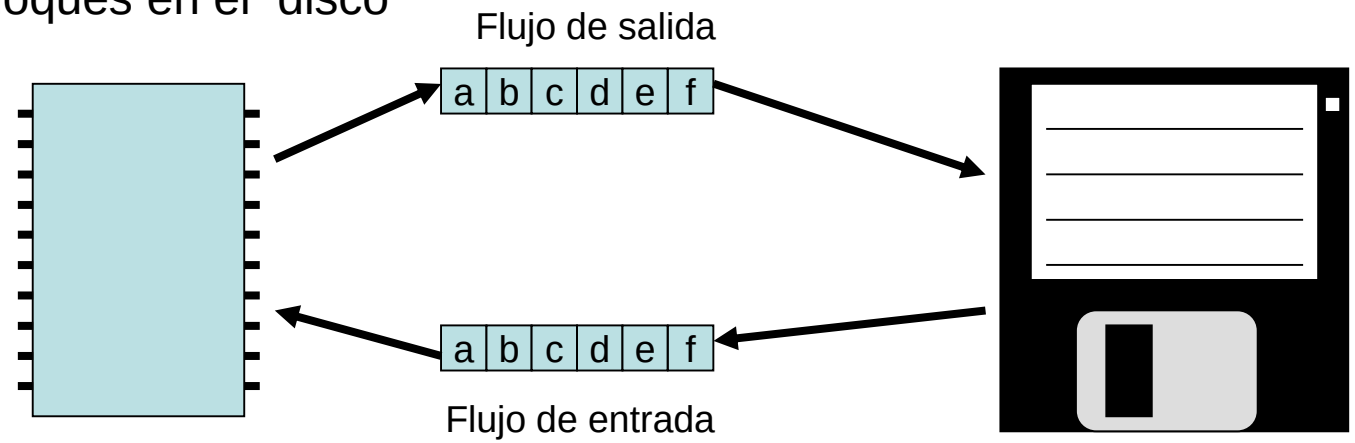
- Un flujo es un buffer (zona de memoria que tiene como objetivo adaptar velocidades distintas de dispositivos, en este caso, disco y procesador).
- Antes de leer o escribir se debe asociar un flujo (stream) a los archivos. El flujo es una apuntador a una estructura FILE*
- Todo programa en C abre tres flujos: stdin, stdout, stderr
  - stdin: entrada estándar esta relacionada con el teclado
  - stdout: salida estándar esta relacionada con la pantalla
  - stderr: salida estándar de errores esta relacionada con la pantalla

Que es un Flujo?

•Un flujo crea una región de memoria (buffer) entre el programa y el archivo en disco

•Esto adapta distintas velocidades (disco lento, procesador rápido) y reduce los accesos a disco

• Por ejemplo, al leer texto, los caracteres se leen en bloques en el buffer, o se escriben en bloques en el disco



Uso de archivos básicamente en 4 pasos

1. Crear una variable para manejar el stream (de tipo FILE *).
2. Abrir el archivo utilizando la función **fopen** y asignándole el resultado de la llamada a la variable creada en el punto 1.
3. Hacer las diversas operaciones (lectura, escritura, etc).
4. Cerrar el archivo utilizando la función **fclose**.

Archivos de texto vs archivos binarios – La función fopen

La función fopen abre el archivo, ya sea para lectura, para escritura o ambos. Pero también, al utilizarla debemos saber el tipo de archivo con el que vamos a trabajar. Además de la ruta del archivo, la función fopen recibe el “modo de apertura”, las posibilidades son las siguientes:

- "r" : abrir un archivo de texto para lectura, el fichero debe existir.
- "w" : abrir un archivo de texto para escritura, se crea si no existe o se sobreescribe si existe.
- "a" : abrir un archivo para escritura, ubicándose al final del mismo, si no existe se crea.
- "rt" : abrir un archivo para lectura, el fichero debe existir.
- "wt" : crear un archivo para lectura y escritura, se crea si no existe o se sobreescribe si existe.
- "r+b ó rb+" : Abre un archivo en modo binario para actualización (lectura y escritura).
- "rb" : Abre un archivo en modo binario para lectura.
- " wb" : Abre un archivo en modo binario para escritura.

fopen y fclose

Se pueden abrir la cantidad de archivos que se desee al mismo tiempo, pero deben ser manejados con distintos punteros.
Luego del fclose, la variable FILE* puede ser reutilizada para abrir cualquier otro archivo.

Ejemplo:

```
FILE *p1, *p2;
p1 = fopen(“leo.lalala”, “rb”);
p2 =fopen(“escribo.sarasa”, “wb”);
……..
……..
fclose(p1);
fclose(p2);
```

Qué es un archivo de texto?

- Es un archivo doc o docx?
- Es un archivo .txt? Si es así, es requisito que sea .txt para que sea “de texto”?
- Conocen las extensiones html, xml, csv, etc?
- Cuántos tipos de archivo de texto hay o pueden haber?

Archivos de texto – Lectura de una palabra con fscanf

La función fscanf sirve para leer un texto con formato desde un archivo abierto. Por ejemplo:

```
int main()
{
    FILE *variable_archivo;
    char str[100];
    char ruta_del_archivo[]= "Pruebadedatos.txt";
    variable_archivo = fopen(ruta_del_archivo, "r");
    fscanf(variable_archivo,"%s", str);
    cout << str << endl;
    fclose(variable_archivo);
    return 0;
}
```

- Qué es leer con formato?
- Qué hace específicamente el programa anterior?
- Qué pasa si el archivo que abre para lectura no existe?
- Si se hiciera otro fscanf antes de cerrar el archivo, qué leería?

Lectura con formato a través del fscanf

int fscanf(FILE *stream, const char *format, dir de identificador1[, … dir de identificador n])

Stream es la variable FILE * con la que se está manejando el archivo, por supuesto debe estar previamente abierto.

Format es un string que especifica el formato a leer. Los formatos posibles se muestran en la tabla siguiente. La cantidad de formatos de la tabla que se indiquen en el string format, debe coincidir con la cantidad de punteros a variables pasados a continuación del parámetro format.

Formato	Descripción	Tipo de variable
%c	Lee un carácter simple	char *
%d	Lee un integer	int *
%e,%E, %f, %g, %G	Un float, el más usado es %f	float *
%o	Integer en octal	int *
%s	Lee un string desde donde se está posicionado en el archivo hasta el primer espacio, salto de línea o fin de archivo (EOF), lo que encuentre primero.	char *, o char[]
%u	Lee un entero sin signo	unsigned int *
%x, %X	Un entero en hexadecima	int *

Lectura con formato a través del fscanf hasta el EOF

En el caso que estamos analizando el fscanf tiene la siguiente forma:
fscanf(variable_archivo,"%s", str);
Con %s estamos indicando que queremos leer un string (desde donde se está ubicado en el archivo hasta el primer espacio, salto de línea o fin de archivo). El string leído será almacenado en la variable str.
Por lo tanto, este programa sólo mostrará la primer palabra del archivo de texto por pantalla.
Y si quisiéramos leer todas las palabras del archivo? De qué manera podríamos repetir esta operación? Hasta cuándo debemos repetirla?
Para ello nos vamos a valer de la marca de fin de archivo **EOF (END OF FILE)**.

Lectura palabra por palabra hasta el EOF con fscanf

```
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
    FILE *variable_archivo;
    char str[100];
    char ruta_del_archivo[]= "Pruebadedatos.txt";
    variable_archivo = fopen(ruta_del_archivo, "r");
    while (fscanf(variable_archivo,"%s", str) != EOF)
          cout << str << endl;
    fclose(variable_archivo);
    return 0;
}
```

Atando cabos, y si no existía el archivo? – Conociendo más a fopen()

Cuando se trabaja un archivo (tanto de texto como binario) en modo lectura, debe existir el archivo que se intenta abrir. La manera de controlar esta situación es verificar si la operación de apertura es o no exitosa.
Para ello hay que analizar lo devuelto por fopen, que es asignado a la variable FILE *.

```
int main()
{
    FILE *variable_archivo;
    char str[100];
    char ruta_del_archivo[]= "Pruebadedatos.txt";
    variable_archivo = fopen(ruta_del_archivo, "r");
    if(variable_archivo == NULL) {
        cout << "Error al intentar abrir el archivo" << endl;
        return 1;
    }
    while (fscanf(variable_archivo,"%s", str) != EOF)
          cout << str << endl;
    fclose(variable_archivo);
    return 0;
}
```

Formatos más avanzados con fscanf

Supongamos que tenemos ahora, un archivo de texto con la siguiente forma:

ESTHER WALLACE 55
GABRIEL REY 30
ALEJANDRO RODRIGUEZ 37
SILVINA PEIRANO 32
VICTORIA PAZ 20

Se desea leer por cada scanf, el nombre, el apellido y la edad de la persona; por lo tanto debemos utilizar un formato adaptado a esta necesidad. Además se requiere listar el promedio de edad de las personas al final, con lo cual, conviene leer las edades como un valor numérico entero.
Cuál podría ser el formato adecuado?

Solución

```
int main()
{
    FILE *variable_archivo;
    char nombre[50];
    char apellido[50];
    int edad, sum=0, cont=0;
    char ruta_del_archivo[]= "Pruebadedatos.txt";
    variable_archivo = fopen(ruta_del_archivo, "r");
    if(variable_archivo == NULL) {
        cout << "Error al intentar abrir el archivo" << endl;
        return 1;
    }
    while (fscanf(variable_archivo,"%s %s %d", nombre, apellido, &edad) != EOF)
    {
        cout << nombre << " | " << apellido << " | " << edad << endl;
        sum += edad;
        cont++;
    }
    cout << "El promedio de edad es: " << sum/cont << " años." << endl;
    fclose(variable_archivo);
    return 0;
}
```

Otros tipos de lectura

Hay otras maneras de leer archivos de texto sin considerar el formato:

- Leer de a un caracter por vez, utilizando la función getc.
- Leer de a una línea, sin considerar formato, con la función fgets, almacenando todos los caracteres de la línea leída en un array de char.

```
FILE *fpuntero;
char c, *pc;
char buffer[100];


/************ LEO CARACTER POR CARACTER *********/
fpuntero = fopen("prueba.txt", "r");
if(fpuntero == NULL) {
    cout<< "Error al intentar abrir el archivo" << endl;
    return 1;
}
do
{
    c = getc(fpuntero); /* Obtiene un caracter del archivo */
    cout << c << endl;
}
while (c != EOF); /* hasta encontrar EOF (el final del archivo)*/
fclose(fpuntero);
```

```
/************ LEO LINEA POR LINEA *********/
fpuntero = fopen("prueba.txt", "r");
if(fpuntero == NULL)
{
    cout << "Error al intentar abrir el archivo" << endl;
    return 1;
}
do
{
    pc = fgets(buffer, 100, fpuntero);
    if (pc != NULL)
        cout << buffer << endl;
}
while (pc != NULL);
fclose(fpuntero);
```

Escritura con formato en un archivo de texto

Para la escritura de archivos de texto utilizamos la función fprintf a la que se le debe especificar el formato de la escritura de la misma manera que a fscanf. Deberíamos tener en cuenta que no hay pasaje de parámetros por referencia, ya que la función no se diseñó con intención de modificar los valores de los identificadores que son parámetro.

```
int main(void)
{
    FILE *arch1;
    int dia,mes,agno;
    string smes;
    arch1 = fopen("holamundo.txt", "w");
    if(arch1 == NULL) {
        cout << "Error al intentar abrir el archivo" << endl;
        return 1;
    }
    fprintf(arch1, "Hola mundo!\n");
    /* TAMBIEN PODEMOS ESCRIBIR CON FORMATOS MÁS COMPLEJOS: */
    cout << "Ingrese día mes y año:" << endl;
    cin >> dia >> mes >> agno;
    switch (mes)
    {
        case 1: smes= "enero"; break;
        case 2: smes= "febrero"; break;
        case 3: smes= "marzo"; break;
        case 4: smes= "abril"; break;
        case 5: smes= "mayo"; break;
        case 6: smes= "junio"; break;
        case 7: smes= "julio"; break;
        case 8: smes= "agosto"; break;
        case 9 : smes= "septiembre"; break;
        case 10: smes= "octubre"; break;
        case 11: smes= "noviembre"; break;
        case 12: smes= "diciembre"; break;
    }
    fprintf(arch1, "Hoy es el %d de %s de %d.\n", dia, smes.c_str(), agno);
    fclose(arch1);
    cout << "Archivo escrito. Pulse una tecla para salir del programa" << endl;
    getch();
    return 0;
}
```

Archivos binarios o de tipo aleatorio

Los archivos de acceso aleatorio permiten escribir registros arbitrarios sin necesidad de ser reescrito el archivo por completo.

Los registros en los archivos aleatorios son todos del mismo tamaño.

Se utiliza las siguietes funciones

`fwrite(arreglo, tamaño, #de registros, archivo)` – escribe un arreglo de registros en un archivo.

`fread(arreglo, tamaño, #de registros, archivo)` – lee un arreglo de registros de un archivo.

Ventajas de los archivos binarios

- Se puede acceder directamente a un registro, es decir, no es necesario una lectura secuencial como en archivos de texto (random access).
- Generalmente son más rápidos que los archivos de texto ya que no se requieren conversiones.
- C toma el disco como un bloque de bytes, y su file pointer puede apuntar a cualquier byte en cualquier momento.
- Seguridad de la información.

Lectura y escritura secuencial de registros

```
struct rec
{
        int x,y,z;
};
void main()
{
    int i,j;
    FILE *f;
    rec r;
    f=fopen(“PRUEBA","wb");
    for (i=1;i<=10; i++)
    {
        r.x=i;
        fwrite(&r,sizeof(rec),1,f);
    }
    fclose(f);
```

```
    f=fopen(“PRUEBA","rb");
    for (i=1;i<=10; i++)
    {
        fread(&r,sizeof(rec),1,f);
        cout << r.x << endl;
    }
    fclose(f);
}
```

fread y fwrite

int fread (puntero destino, sizeof (tipo_dato), cantidad a leer de tipo dato, puntero al archivo)

La función fread toma:

- Un puntero donde guardar lainfo.
- La cantidad de bytes a leer por bloque
- La cantidad de bloques a leer.
- El puntero al archivo.

La línea **fread(&r,sizeof(rec),1,f)** implica leer 12 bytes del archivo f desde la posición actual, y guardar los datos en el puntero &r.

La función fwrite funciona de manera análoga.

int fwrite (puntero origen, sizeof (tipo_dato), cantidad a escribi de tipo dato, puntero al archivo)

La función fseek

- int fseek(**FILE** **stream_pointer*, **long** *offset*, **int** *origen*);

Offset es la cantidad de bytes que se desea mover el puntero.

Origen es desde donde empieza a contar

  - SEEK_SET: cuenta desde el comienzo
  - SEEK_CUR: cuenta desde la posición actual
  - SEEK_END: cuenta desde el final del archivo (así que el offset debe ser un valor negativo).

Valores que devuelve:

- *0* (cero) : Se pudo hacer el fseek.
- Valor distinto de cero : Hubo un error

Fseek – Ejemplo

```
/* Vamos a cambiar el 5to registro. El modo “rb+” en fopen es lectura escritura*/
f=fopen(“PRUEBA","rb+");
fseek(f,sizeof(rec)*4,SEEK_SET);
fread(&r,sizeof(rec),1,f);
r.x+=100;
fseek(f,sizeof(rec)*4,SEEK_SET);
//fseek(f,-sizeof(rec),SEEK_CUR);
fwrite(&r,sizeof(rec),1,f);
fclose(f);
printf("\n");
```

Ftell - Posición en el archivo

Posicion_bytes = ftell(FILE *f);

Esta función nos devuelve la posición actual del file pointer en bytes. Es decir, si hemos leído 2 struct rec (que cada una tiene un tamaño de 12 bytes) ftell nos devolvería 24 (avanzó 24 bytes desde el comienzo).

Ftell – tamaño del archivo

La función ftell nos puede decir el tamaño del archivo para no pasarnos con el fseek.

Si nos posicionamos al final del archivo y luego hacemos un ftell:

```
fseek( f, 0, SEEK_END );
file_length = ftell(f);
```

Lo que tenemos es la cantidad de bytes hasta el final del archivo, lo que es equivalente al tamaño del archivo en bytes.

Por lo tanto la cantidad de registros que tiene un archivo:

```
nRegistros = file_lenght/sizeof(tRegistro);
```