

MANEJO DE ARCHIVOS EN EL LENGUAJE C++

Elaborado por: Juan Miguel Guanira Erazo

PUCP

Archivos

DEFINICIÓN:

Un archivo es una colección de datos que se encuentran almacenados de manera permanente en algún dispositivo del computador, como el disco duro, dispositivo o memoria USB, etc.

Todo lo que se almacena en el computador se considera un archivo: programas, textos, gráficos, fotografías, videos, música, etc.

Formas de almacenar la información en un archivo

Todos los archivos son lo mismo para el computador.

Lo que hace la diferencia es la forma cómo se almacena la información.

Es decir, cómo se codifica la información para almacenarla y luego poder recuperarla.

Existe dos formas básicas de codificar la información para guardarlas en archivos: “*el formato de texto*” y “*el formato binario*”.

Todo archivo almacenado en el computador toma uno de estos dos formatos o una combinación de los ambos.

En programación se adoptan estos formatos para generar lo que se conoce como “*archivos de texto*” y “*archivos binarios*”.

DIFERENCIAS ENTRE

ARCHIVOS DE TEXTO

Y

ARCHIVOS BINARIOS

1.- Forma cómo se almacena un dato en el archivo.

Si se tiene en un programa:

short a = 573;

En la memoria principal se almacena como:

0011 1101	0000 0010
-----------	-----------

En un “*archivo de textos*” el dato se transforma y guarda en el archivo como una secuencia de caracteres:

‘5’	‘7’	‘3’
-----	-----	-----

Es decir que en el archivo se guardarán tantos bytes como cifras tenga el dato, en este caso se almacenarán 3 bytes para el número 573.

En un “*archivo binario*” el dato *NO* se transforma, se guarda tal y como está en la memoria principal:

0011 1101	0000 0010
-----------	-----------

Es decir que en el archivo se guardarán siempre 2 bytes para datos del tipo short, tengan el número de cifras que tengan, ya sean 15238, 9, 15, 222, etc.

2.- Separación entre datos.

Si se tiene en un programa los valores de:

short a = 37, b = 19, c = 715;

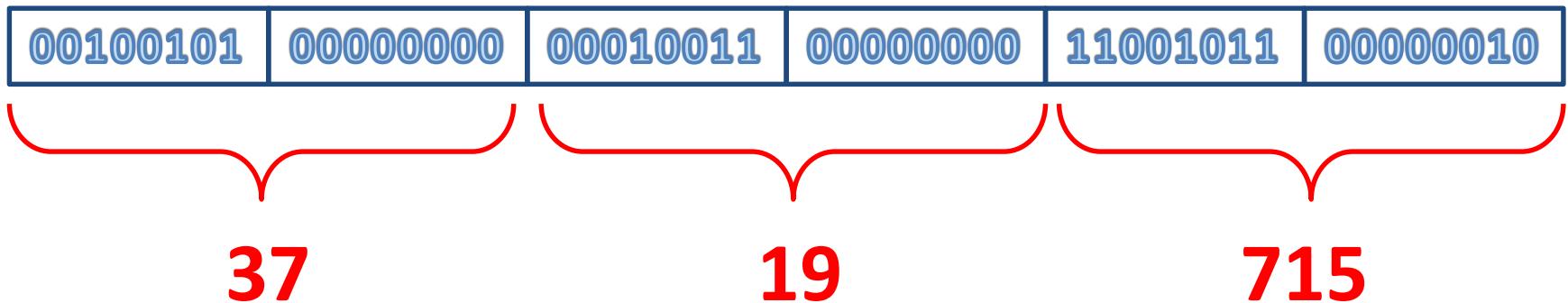
En un “*archivo de textos*”, como los datos se transforma una secuencia de caracteres, luego de almacenarlos no se podrían recuperar a menos que se les separe por un espacio en blanco, tabulador o cambio de línea.

‘3’	‘7’	‘ ’	‘1’	‘9’	‘ ’	‘7’	‘1’	‘5’
-----	-----	-----	-----	-----	-----	-----	-----	-----

De lo contrario no se podría determinar dónde empiezan y dónde terminan.

En un “*archivo binario*”, no se requieren separadores porque los datos se almacenan según el tipo y por lo tanto ocupan el mismo espacio cada uno.

short a = 37, b = 19, c = 715;



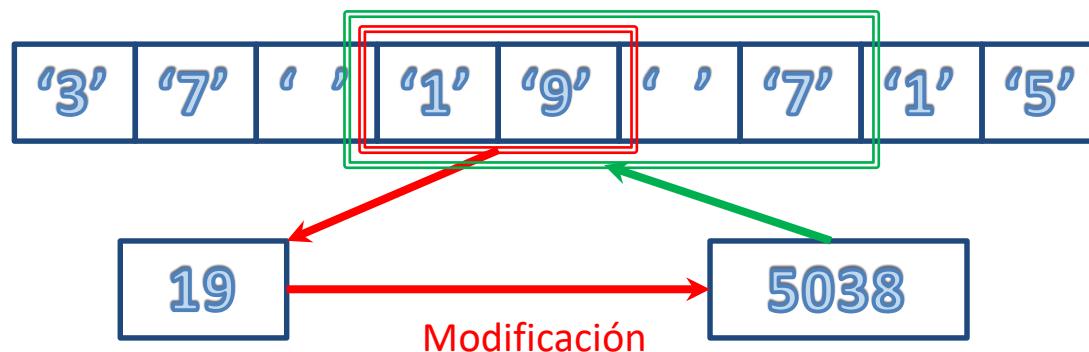
3.- Acceso a los datos

En un “*archivo de textos*”, por el tamaño no uniforme de los datos, no se puede determinar la posición de uno de ellos, por lo que habrá que leerlos uno a continuación del otro. Esto se denomina “*acceso secuencial*”.

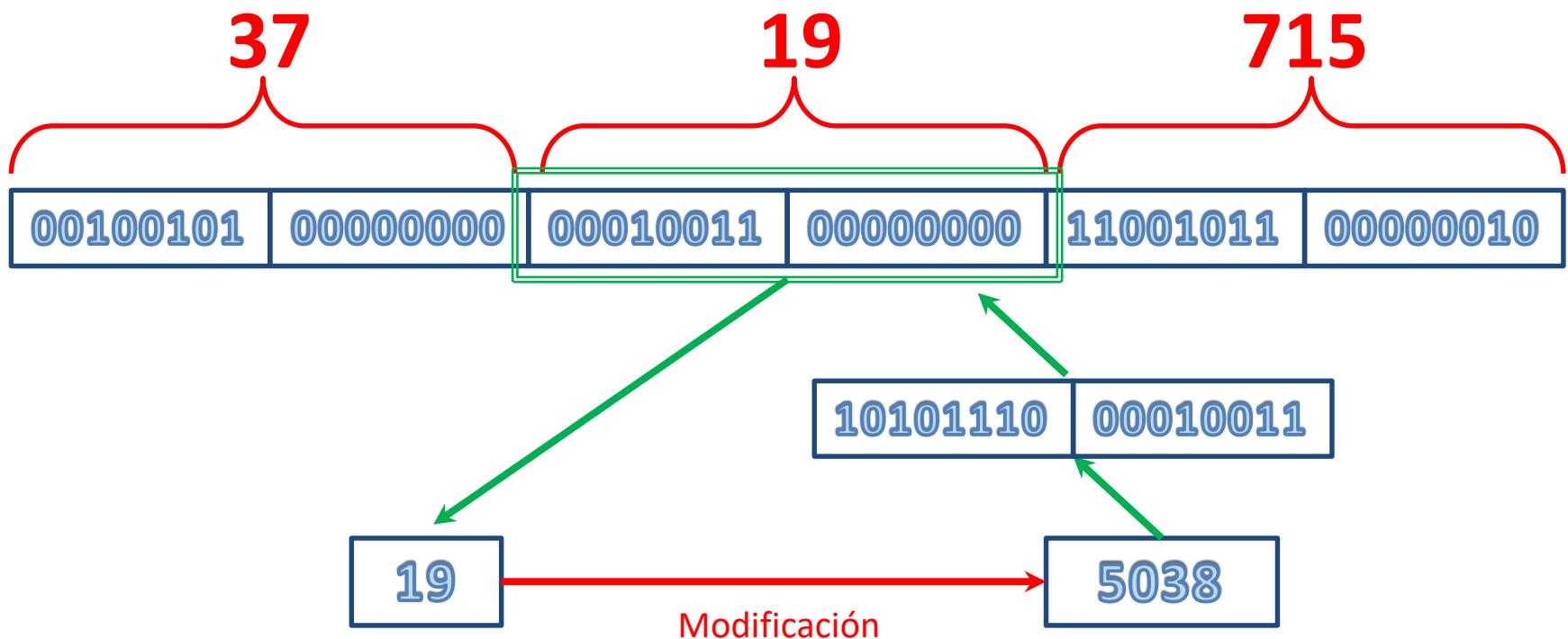
En un “*archivo binario*”, por el contrario, debido a la uniformidad del tamaño de los datos, se puede determinar la posición de uno de ellos, por lo que se podrá acceder a uno de ellos sin tocar el resto, a esto se le denomina “*acceso directo o aleatorio*”.

4.- Actualización de datos

En un “*archivo de textos*”, no se pueden actualizar los datos, esto se debe a que el espacio que ocupa un dato depende del número de cifras. Si se lee un dato de un archivo y en la modificación cambia su número de cifras, no habrá lugar para volverlo a grabar.



En un “*archivo binario*”, esta tarea es perfectamente posible ya que a pesar que el valor cambie, el espacio que ocupa no.



ARCHIVOS BINARIOS EN C++

El manejo de los archivos binarios en C++ se realiza bajo el mismo principio que en lenguaje C, pero con instrucciones muy diferentes.

Lo que debe recordar es que, para obtener el máximo provecho del archivo, debe almacenar la información de manera homogénea, en bloque del mismo tamaño que llamaremos registros.

Un registro puede contener diferentes tipos de datos (campos), pero su tamaño total debe ser siempre el mismo.

Esto es fácil si los campos fueran todos valores numéricos. Entonces debe centrarse en las cadenas de caracteres, éstas deben definirse como arreglos del mismo tamaño, dependiendo del campo, y lo que se guarde en el archivo debe ser todo el arreglo, así el texto que contiene no lo llene.

DEFINICIÓN Y APERTURA DE UN ARCHIVO BINARIO

```
ofstream arch("arch.bin",
ios::out | ios::binary);
```

```
ifstream arch("arch.bin",
ios::in | ios::binary);
```

```
fstream arch("arch.bin",
ios::out | ios::in | ios::binary);
```

Este último sirve para poder leer y escribir indistintamente en el archivo (actualización).

OPERACIONES DE LECTURA Y ESCRITURA EN ARCHIVOS BINARIOS

El principio del manejo de archivos binarios es manejar la información tal como está almacenada en la memoria principal del computador.

Por lo tanto, en el caso del método de escritura, se debe proporcionar la dirección dónde se encuentre el dato a grabar y la cantidad de bytes que a partir de allí se van a enviar al archivo .

En el caso del método de lectura, le debe proporcionar la dirección dónde se encuentre la variable que recibirá el dato leído y la cantidad de bytes que se extraerán del archivo .

Este principio hace muy flexible las operaciones con los archivos, permitiendo inclusive leer o escribir el contenido de todo un arreglo en una sola operación.

ESCRITURA :

```
arch.write (   
    reinterpret_cast<const char *> (&v),  
    sizeof (v_o_t) );
```

Donde:

- **reinterpret_cast<const char *>**

Es un operador que busca poder manipular la información contenida en la variable **v**, byte por byte.

- **v** Variable donde se encuentra el dato.

- **t** Tipo de dato de la variable **v**.

ESCRITURA (arreglos):

```
arch.write (   
    reinterpret_cast<const char *> (a),  
    sizeof (t)*n );
```

Donde:

- a Indica un arreglo.
- t Indica el tipo de dato de los elementos del arreglo.
- n El número de elementos del arreglo que se quiere guardar.

LECTURA :

```
arch.read (   
    reinterpret_cast< char *> (&v),  
    sizeof (v) );
```

Donde:

- **reinterpret_cast< char *>**

Es un operador que busca poder manipular la información contenida en la variable **v**, byte por byte.

LECTURA (arreglos):

```
arch.read (   
    reinterpret_cast< char *> (a),  
    sizeof (t)*n);
```

Donde:

- **reinterpret_cast< char *>**

Es un operador que busca poder manipular, byte por byte, la variable **v** de modo que pueda colocar la información que viene del archivo.

ACCESO ALEATORIO O DIRECTO EN ARCHIVOS BINARIOS

Método para posicionar el indicador del archivo en algún byte del archivo:

arch.seekg (n, ios::beg);

arch.seekg (n, ios::cur);

arch.seekg (n, ios::end);

Mueve el indicador del archivo **n** bytes desde el inicio del archivo (**ios::beg**), desde la posición del indicador (**ios::cur**) o desde el final del archivo (**ios::end**).

Método para determinar la cantidad de bytes que hay desde el inicio del archivo hasta la posición del archivo:

arch.tellg ()

El método devuelva un valor entero (int) con la cantidad de bytes.

**PASEMOS A LA PARTE
PRÁCTICA**