Apunte para manejo de Archivos en C++

1. Introducción

1.1. Manipulación de Archivos

En C++, se utilizan streams (flujos) para gestionar la lectura y escritura de datos.
Existen cuatro flujos predefinidos.

- cin: Entrada Estandar (normalmente teclado).
- cout: Salida Estandar (normalmente Monitor).
- cerr: Salida de Error Estandar no almacenados en Buffer (normalmente Monitor).
- clog: Salida de Error Estandar almacenados en Buffer.

Desde el punto de vista informático, un fichero es una colección de información que almacenamos en un soporte magnético u optico para poder manipularla en cualquier momento.

Los Flujos o Streams se pueden también asociar a Nombres de ficheros y con ello podemos acceder a manipular un fichero.

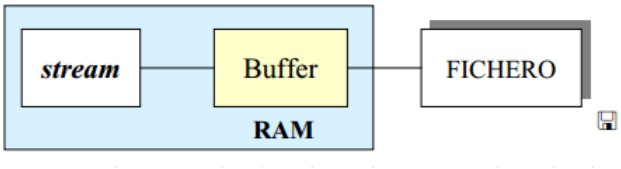

Figura 1: Relación Stream Archivo

Técnicamente, un Stream o Flujo es el enlace lógico utilizado por el programador en C o C++ para leer o escribir datos desde y hacia los dispositivos estándar conectados a la PC. A modo de recordatorio ya trabajamos con flujos de entrada y salida ellos son: cin y cout.

Ver que existe entre el Archivo y el Stream un buffer de memoria RAM.

Las operaciones de E/S implementadas así son más eficientes ya que el acceso a la memoria RAM consume *menos tiempo* que el acceso a un dispositivo físico. El buffer hace que el número de accesos al fichero físico sea menor. El uso del buffer permite realizar operaciones de entrada salida de forma más eficiente aún.

```
Importante:VER!!!!
Al igual que los flujos cin y cout, los flujos de E/S solo pueden
    transferir datos en una dirección, esto significa que se tienen que
    definir flujos diferentes para lectura y escritura de datos.
```

Normalmente, el dispositivo estándar para manipular entradas es el teclado y este, en C++, está asociado al objeto cin; el dispositivo estándar de salida está asociado (generalmente) con la pantalla o monitor de despliegue de la PC y, el mismo, en C++ se puede acceder por medio del objeto cout.

En definitiva, abrir un fichero significa definir un stream. Dicho stream permite la transferencia de datos entre el *programa y el fichero en disco*, o sea que *no trabajamos sobre el archivo*, si no mas bien sobre el stream, y los datos del stream son volcados luego al Fichero, es decir que llegamos al fichero de una *manera indirecta*.

Podemos pensar que los son como dispositivos lógicos streams generan o consumen información.

Desde el comienzo de C++, en Taller de Informática usamos cin y cout, en realidad como veremos a continuación, hay una cierta relación con otras funciones, y todas derivan de una clase, la :*ios*. En esta página podemos ver información al respecto:

Referencia Web:

http://www.cplusplus.com/reference/iostream/

`http://www.cplusplus.com/doc/tutorial/files/`

En ella podemos ver los métodos públicos usados en los ejemplos y una breve explicación de cada clase

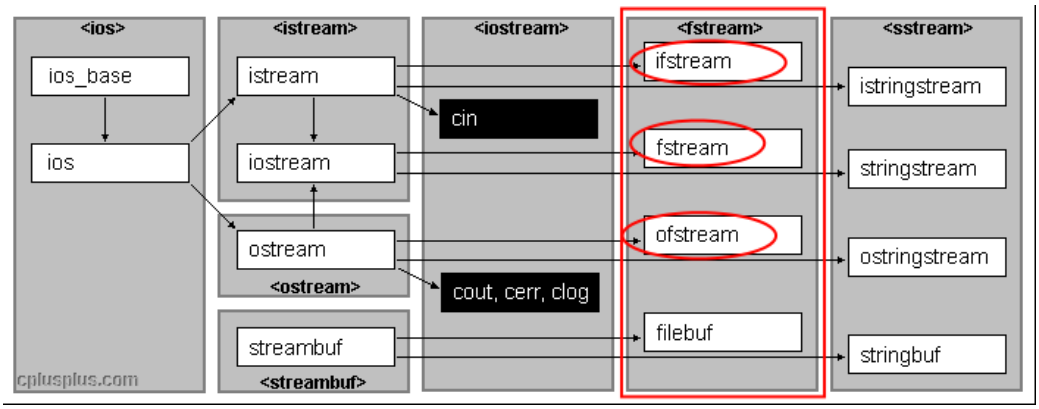

Figura 2: Jerarquía de la librería de entrada/salida

Como podemos ver en la figura, la librería <*ios*>, tiene una clase heradada <*istream*> , a su ves esta tiene un conjunto de clases que aparecen en el recuadro rojo de la figura, es el grupo de <*fstream*>.

Incluir esta librería fstream, en un programa, permite el uso de recursos para trabajar con ficheros.

Como ya dijimos , la *ios* es la librería estándar en C++ para poder tener acceso a los dispositivos estándar de entrada y/o salida, por lo tanto las clases derivada de esta librería también tendrán esas características pero cada una tiene algún campo en el que es mas util o mejor dicho para el que fué escrita, la relación entre cada una de ellas escapa a este apunte como así también la utilidad de alguna de ellas.

Tal sería el caso de <*streambuf*> a modo de comentaria, esta clase proporciona acceso a operaciones básicas de entrada y salida de datos para a un nivel inferior, y reiteramos , este tipo de clases no se tratará en esta apunte.

En este apunte nos centraremos en las siguientes clases dedivadas de <*fstream*>:

- *ifstream*: si desea abrir un archivo solo para lectura , es para leer del fichero (i de input, f de file y stream).
- *ofstream:* para escritura use objetos de la clase ofstream, es para escribir (o de output, f de file y stream).
- *fstream*.usada para abrir un archivo específico en modo de lectura y escritura use un objeto de esta clase.

En sus programas, si usted desea hacer uso de los objetos *cin*, *cout*, tendrá que incluir la biblioteca *iostream* ,por medio de la directiva *#include*, de igual manera para usar las clases *ifstream, ofstream y fstream* vamos a incluir *#include* <*fstream*>.

Observación: En general usaremos el nombre de Flujo para referirnos al archivo, si bien esto no es totalmente cierto, ya que sabemos que el Flujo o Stream está asociado al Archivo, pero NO ES EL ARCHIVO.

1.2. Tipos de Acceso.

Los archivos se pueden acceder de distintas maneras y en consecuencia se pueden clasificar según el tipo de acceso:

1.2.1. *Acceso Secuencial.*

Los datos se almacenan de forma consecutiva y no es posible leer un registro directamente, es decir para eer el registro n hay que leer los n-1 registros anteriores. Este tipo de archivos no son prácticos si queremos insertar algún contenido en el medio del mismo.

1.2.2. *Acceso Aleatorio.*

Se puede acceder a un registro concreto sin necesidad de leer todos los anteriores.

Para ver esto, pensemos por ejemplo en acceder a un tema puntual de un CD de música (modo Aleatorio) o en caso contrario de un video en Cassette VHS (acceso Secuencial). Para poder usar este tipo de archivos se debe crear en el mismo una estructura, donde cada estructura tiene no que se suele conocer como “record” y dentro de este cada dato tiene un tamaño específico o campo de tamaño determinado. Veremos mas adelante el manejo de este tipo de archivos.

Observación:
El final de archivo es marcado por distintos caracteres dependiendo el entorno en el cual no encontremos.
Linux –> ***Control + D***
Windows –> ***Control +Z***
Macintosh –> ***Control + D***

1.3. Tipos de Archivos.

El contenido de los archivos puede ser de dos tipos, texto o binario.

1.3.1. Archivos Texto.

Los datos que se almacenan en el archivo son código ASCII y por tanto, pueden ser procesados por cualquier editor de texto.

1.3.2. Archivos Binarios.

Los datos se almacenan en binario, por lo tanto NO esta asociado al código ASCII.

Cuando un archivo se abre en modo texto, se deben realizar varias conversiones de caracteres como la conversión de las secuencias de retorno de carro/salto de línea en nuevas líneas. Sin embargo cuando un archivo se abre en modo binario no se realiza ninguna de estas conversiones.

Cualquier archivo que contenga texto con fromato en modo datos binarios, puede abrise en modo binario o en modo texto. La única diferencia es la conversión de caracteres.

1.4. Operaciones sobre Archivos.

Por otro lado, sobre los archivos se pueden realizar varias operaciones , estas serían:

1. Abrir archivo.
   a) Para Entrada o lectura.
   b) Para Salida o escritura.
      1) En modo Truncado. Borro el contenido que tenía y escribo desde cero.
      2) En modo Añadir. Agrego al contenido al existente.
2. Cerrar Archivo.
3. Escribir Archivo.
4. Leer Archivo.
5. Funciones de Control.

Luego veremos en detalle cada una de ellas.

1.5. Sintaxis: Objetos y Métodos.

Antes de comenzar a ver como se utilizan estas clases y métodos de estas clases. Tienen argumentos similares. Vamos a comentar primero que básicamente existen sintaxis diferentes para declarar objetos, estas formas o sintaxis son iguales para las clases de en estudio: *ifstream, ofstream e iofstream.*

Vamos a ver un poco las Sintaxis.

- *fstreams (para lectura y escritura)*

```
Sintáxis 1)fstream nombre_flujo();
Sintáxis 2)fstream nombre_flujo(const char*, int, int = filebuf::openprot);
Sintáxis 3)fstream nombre_flujo(int);
Sintáxis 4)fstream nombre_flujo(int _f, char*, int);
```

- *ifstreams (solo para lectura)*

```
Sintáxis 1)ifstream nombre_flujo();
Sintáxis 2)ifstream nombre_flujo(const char*, int, int = filebuf::openprot)
    ;
Sintáxis 3)ifstream nombre_flujo(int);
Sintáxis 4)ifstream nombre_flujo(int _f, char*, int);
```

- *ofstreams (solo para escritura)*

```
Sintáxis 1)ofstream nombre_flujo();
Sintáxis 2)ofstream nombre_flujo(const char*, int, int = filebuf::openprot)
    ;
Sintáxis 3)ofstream nombre_flujo(int);
Sintáxis 4)ofstream nombre_flujo(int _f, char*, int);
```

Se puede notar que el uso de fstream , puede reemplazar a ifstream y ofstream, pero los ejercicios de este apunte usaremos basicamente ifstream y ofstream ya que la idea es trasmitir conceptos, este es el camino mas largo pero conceptualmente queda mas claro. De todas maneras una vez que se haya entendido el concepto es facil extrapolar al uso de fstream , solo hay que tener cuidado con los Modos por Omisión o por Defecto de cada método.

1.5.1. Argumento “nombre”.

Vamos a Comentar sobre las formas de sintaxis , pero *SOLO* refiriéndonos *AL PRIMER ARGUMENTO* , por el momento.

- Sintaxis 1) En este tipo de sintaxis , el método constructor crea un objeto que no está (aún) asociado a un archivo en disco (ver que no hay nombre de archivo figura solo paréntesis (), sin argumentos), en estos casos, se tendrá que otro método para establecer la conexión entre el stream y el archivo en disco. Vemos que se usa el método *stream.open("nombre de archivo")*;
  - ***ofstream carga();****// defino un flujo de salida llamado carga, ver que no hay nombre de archivo asociado al flujo.*
  - ***ofstream carga;*** *// defino un flujo de salida llamado carga, ver que no hay nombre de archivo asociado al flujo.*
  - ***fstream prueba;*** *//defino un flujos de entrada salida llamado prueba, ver que no hay nombre de archivo asociado al flujo.*
  - ***ifstream muestra;*** *//defino un flujo de entrada llamado muestra, ver que no hay nombre de archivo asociado al flujo.*
- Sintaxis 2) En el segundo tipo de Sintaxis , el método constructor crea un objeto asociado a un archivo en disco, en estos casos, el archivo en el disco queda abierto y asociado al stream. En este, el parámetro char * apunta a una cadena de caracteres con el nombre del archivo. Respecto del contenido del fichero, si existe, se conserva; los nuevos datos escritos se añaden al final. Por defecto, el fichero no se crea si no existe. Recordar que el nombre de un arreglo es el puntero al primer elemento del arreglo, esto es similar, el nombre del archivo es el puntero al archivo.
  - ***ofstream salida("prueba.txt", ios::out);****//creo flujo de salida y asocio a archivo prueba.txt.*
  - ***fstream salida("/home/usuario/prueba.txt", ios::out);****//creo flujo de entrada/salida y asocio a archivo prueba.txt ubicado específicamente en un lugar.*
  - ***fstream mostrar("/home/usuario/prueba.txt", ios::in);****//creo flujo de entrada y asocio a archivo prueba.txt ubicado específicamente en un lugar.*
  - ***fstream salida("c:/prueba.txt", ios::in | ios::out);****//creo flujo de entrada/salida y asocio a archivo prueba.txt ubicado específicamente en un lugar, sería para el caso de SO Windows.*
- Sintaxis 3)En la tercer sintaxis , el método crea un stream a raíz de un identificador de archivo y lo conecta al nombre de un archivo abierto previamente.
- Sintaxis 4)En el cuarto tipo de sintaxis, el método crea un stream conectado a un archivo.

Veamos los métodos mas usados con algunos ejemplos.

1.5.2. Argumento “modo”.

El argumento modo, es el segundo argumento utilizado en el tipo de Sintaxis 2, y define como se abrirá el archivo. Los valores posibles para este modo son:

ios::app En este modo todas las operaciones de salida se llevan a cabo en el final del archivo. Dado que todas las escrituras son precedidos implícitamente por busca, no hay manera de escribir en otro lugar. Ejemplo:

ofstream flujo_uno ("a:\misdatos.txt", ios::app);// flujo de salida, agrego al final del archivo.

ios::ate Cuando se establece este modo, la posición inicial será el final del archivo (**at the end**), pero se puede libremente buscar en cualquier lugar.

ios::binary En este modo se abre el archivo en forma binaria.

ios::in Se usa este modo para indicar que un archivo admite entrada o es solo de entrada. En los casos en que se use ifstream, no será neceario su uso, ya que ifstream es para definir flujos de entrada solamente. Tampoco será necesario especificarlo si usamos ofstream ya que el flujo definido para esta clase es solo de salida. En ambos casos no es necesario usar estos especificadores.

ios::out Este modo es lo contrario a in, y se usa para indicar que el archivo admite solo salida. Por lo tanto no se usará en ifstream y no sería necesario incluirlo al usar ofstream.

ios::trunc El uso de este modo hará que el contenido de un archivo , si es que ya existe se destruya, es decir dejará el archivo vacío,con longitud cero.

Estas banderas pueden combinarse usando entre ellas el operador OR: |.

1.5.3. Argumento "acceso".

El valor de acceso determina como se accederá al archivo. Su valor implícito o por defecto es : *filebuf::openprot*, que especifica un archivo normal. (filebuf: es una clase derivada de streambuf).

Normalmente dejaremos el valor implícito, pero dependiendo el entorno de programación, podría ser necesario consultar en el manual del compilador. Este argumento puede ser necesario de especificar un valor que no se el implícito, por ejemplo para el caso de compartir archivos en una red, este caso no será estudiado en este apunte.

1.6. Manejo de archivos Secuenciales.

Para comenzar a ver los métodos que permiten trabajar con archivos, vamos a usar mas que nada los archivos del tipo secuencial,

1.6.1. Método Close.

Manos a la obra. Ahora comenzaremos a ver realmente código que nos permita utilizar los conceptos citados anteriormente.

```
1 #include <fstream>
2 using namespace std;
3 int main(){
4 ofstream flujo("/home/daniel/prueba.txt", ios::app);//creo flujo de salida
      y asocio a archivo prueba.txt
5 flujo<< "Hola"<<endl;//mando una línea
6 flujo<<" Mundo"<<endl; //mando otra línea, debería agregarse al final,
      luego de "Hola"
7 flujo.close();//Cierro el flujo
8 return 0; }
```

Podemos observar en el código anterior lo siguiente:

- Es un flujo de salida, si cambiamos ofstream por ifstream al compilar ,tiraría un error, ya que al ser un flujo de entrada no podemos usar el operador "<<" .
- Ver que al ser ofstream, no es necesario indicar el modo como : ios::out, ya que se establece por defecto.
- Al especificar ios::app, lo que se agregue al archivo; se ubicará al final.
- Ver que no se incluye la biblioteca iostream, no es necesaria en este ejercicio.
- Cuando el programa ha terminado de manipular el fichero, éste debe cerrarse. Para cerrar un archivo, basta con ejecutar la función close sobre el flujo asociado al fichero.Vemos el uso de el método *".close"*, este método se aplica al stream llamado "flujo" de manera de cerrar el mismo, esto hace que se vuelque el contenido del flujo al archivo y se cierre el archivo, puede que si omitimos esta línea al finalizar el programa el archivo es cerrado automáticamente y el flujo descartado, de todas maneras ponerlo de manera explícita hace al búen programador.

También podríamos escribir el programa anterior de la siguiente manera:

```
1 #include <fstream>
2 using namespace std;
3 int main(){
4 fstream flujo("/home/daniel/prueba.txt", ios::in | ios::out | ios::app);//
      creo flujo de E/S asoc. prueba.txt
5 flujo<< "Hola"<<endl;//mando una línea
6 flujo<<" Mundo"<<endl;//mando otra línea, debería agregarse al final, luego
      de "Hola"
7 flujo.close();//Cierro el flujo
8 return 0; }
```

- En este caso usamos la clase fstream, e indicamos que el modo será E/S (**ios::in | ios::out**) y que se agregue al final con ios::app.
- Como fstream básicamente trabaja con E/S, los argumentos de modo ios::in | ios::out , se pueden omitir y funcionaría igual el programa.

```
1 #include <fstream>
2 using namespace std;
3 int main(){
4 fstream flujo("/home/daniel/prueba.txt", ios::in | ios::out | ios::app);
5 flujo<< "Hola 1"<<endl;
6 flujo<<" Mundo 1"<<endl;
7 flujo.close();
8 return 0; }
```

- Este programa agrega al archivo anterior "prueba.txt"que tenía "Hola Mundo", el texto : Hola1 Mundo1.
- Si se omite en el programa ios::in | ios::out, y solamente dejamos ios::app ,esto no sucedería.
- Se debe prestar atención a los argumentos de modo, ya que cuando no indicamos en forma explícita el argumento de modo se suelen tomar argumentos por defecto y estos ocacionan lo indicado en el punto anterior.

Si ahora ejecutamos este código:

```
1 #include <fstream>
2 using namespace std;
```

```
3  int main(){
4  fstream flujo("/home/daniel/prueba.txt", ios::in | ios::out | ios::ate |
       ios::trunc);
5  flujo<< "Hola 1"<<endl;
6  flujo<<" Mundo 1"<<endl;
7  flujo.close();
8   return 0; }
```

- Este código tiene el argumento de modo **ios::trunc**, por lo tanto el contenido del archivo "prueba.txt" será eliminado y solo quedará "Hola 1" y "Mundo 1", eliminandose el "Hola Mundo" anterior.

1.6.2. Método open

Todas estas clases (**fstream,ifstream, ofstream**) disponen además del método "open", para abrir el fichero a lo largo de la ejecución del programa.

En los códigos escritos anteriormente, el **método open es implícito** y se abre al crear la relación entre el stream y el archivo, pero para el caso en que la asociaión se realice en *otro momento* se usa el método **open**.

El método *open* es un método público. El prototipo del método sería:

```
void open(const char *name, int mode, int prot=filebuf::openprot);
```

Donde:

name Es el nombre del fichero.

mode Es alguno de los modos citados anteriormente : in, out, ate, app, binary y trunc.

Veamos un ejemplo

Código 1: Ejemplo de uso de metodo open en fstream

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main () {
5    ofstream outfile; //creo el flujo
6    outfile.open ("test.txt"); //<--Método Open,asocio flujo a fichero, solo
         paso un argumento!!
7    // >> Las operaciones de entrada salida aquí!! <<
8    outfile.close();
9    return 0;
10 }
```

1.6.3. Modos por Omisión o por Defecto.

Todas las funciones miembro open de las clases **ofstream, ifstream y fstream** incluyen un modo por omision o por defecto como ya comentamos, cuando abren archivos , veamos cuales son:

clase	modo por omision
ofstream	ios::out | ios::trunc
ifstream	ios::in
fstream	ios::in | ios::out

El valor por omision solamente es aplicado si la funcion es llamada ***sin especificar un parametro modo***. Si la funcion es llamada con algún argumento de modo , el parametro por omision es saltado, no combinado.

Observación:

Ya que la primera tarea que es desepeñada sobre un objeto de las clases ofstream, ifstream y fstream es frecuentemente abrir un archivo, estas tres clases incluyen un constructor que directamente llama a la funcion miembro ***open*** y tiene los mismos parametros que este.

1.6.4. Método is_open.

Se puede chequear archivo ha sido abierto correctamente mediante una llamada a la funcion miembro *is_open()*:

bool is_open();

Este método retorna un valor de tipo bool indicando *true* en caso que en realidad el objeto haya sido correctamente asociado con un archivo abierto o *false* en otro caso.

Veamos un ejemplo:

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main(){ double n=0;
5  fstream flujo("/home/daniel/prueba.txt", ios::in | ios::out | ios::ate );//
       Añade al final
6  // El objeto flujo es abierto implicitamente.
7  flujo<<"ABCDEFGHIJKLMNOPQRSTUVWXYZ"<<endl;//Envio texto
8  flujo<<"abcdefghijklmnopqrstuvwxyz"<<endl; //Envio mas texto
9  if(flujo.is_open())cout<<"Flujo Abierto"<<endl;
10 cout<<"cierro el flujo"<<endl;
11 flujo.close();
12 if(!flujo.is_open())cout<<"Flujo cerrado"<<endl;
13 return 0; }
```

Veamos un ejemplo en el que crearemos un archivo , vamos a introducir datos en el y luego ver los datos que introducimos, con la idea de aplicar los conceptos hasta aquí vistos.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main(){ cout<<"Este programa permite crea un archivo con lecturas de
       temperatura "<<endl;
5  cout<<"humedad para distintos días de la semana"<<endl;
6  cout<<"Todo se guarda en un archivo /home/daniel/prueba.txt"<<endl;
7  cout<<"ctrl + D finaliza el programa."<<endl;
8  int i=1; char dia[11];float temp,humedad;//Variables usadas para pasar
       datos al flujo
9  ofstream flujo("/home/daniel/prueba.txt", ios::out | ios::ate );//Añade al
       final
10 // El objeto flujo es abierto implicitamente.
11 if(!flujo) //Controlo que se pueda crear el flujo y asociar al archivo
12         {cout<<"Error no se puede abrir archivo. Finalizando";cin.get();
               return -1;}
13 cout<<"Ingrese: Dia y luego Temperatura y Humedad dejando un espacio entre
       cada campo"<<endl; while(cin>>dia>>temp>>humedad){flujo<<i<<" "<<dia<<"
         "<<temp<<"  "<<humedad<<endl;i++;}
14 flujo.close();
15 if(!flujo.is_open())cout<<"Flujo cerrado"<<endl;
16 cout<<"Finalizando el programa!!"; return 0; }
```

Observaciones.

- Este programa solo abre un flujo de Salida (Out).
- Si al ingresar los números flotantes no equivocamos el punto "." por la coma "," el programa finalizará ya que el caracter que separa los decimales no es el esperado para un flotante.

Vamos a agregar ahora mas líneas al programa con el objeto de poder leer el contenido del archivo.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main(){ cout<<"Este programa permite crea un archivo con lecturas de
       temperatura "<<endl;
5  cout<<"humedad para distintos días de la semana"<<endl;
6  cout<<"Todo se guarda en un archivo /home/daniel/prueba.txt"<<endl;
7  cout<<"ctrl + D finaliza el programa."<<endl;
8  int i=0;char cr; char dia[11];float temp,humedad;//Variables usadas para
       pasar datos al flujo
9  ofstream flujo_salida("/home/daniel/prueba.txt", ios::out | ios::ate );//
       Añade al final
10 // El objeto flujo es abierto implicitamente.
11 if(!flujo_salida) //Controlo que se pueda crear el flujo y asociar al
       archivo
12 {cout<<"Error no se puede abrir archivo. Finalizando";     cin.get();
       return -1;     }
13 cout<<endl<<"Flujo de salida abierto correctamente!!"<<endl;
14 cout<<"Ingrese: Dia y luego Temperatura y Humedad dejando un espacio entre
       cada campo"<<endl; while(cin>>dia>>temp>>humedad){flujo_salida<<i<<" "
       <<dia<<"  "<<temp<<"  "<<humedad<<endl;i++;}
15 flujo_salida.close();
16 if(!flujo_salida.is_open())cout<<"Flujo de Salida :cerrado"<<endl;
17 ifstream flujo_entrada("/home/daniel/prueba.txt");//ios::in implicito!!
18 if(!flujo_entrada) //Controlo que se pueda crear el flujo y asociar al
       archivo
19 {cout<<"Error no se puede abrir archivo. Finalizando"; cin.get(); return
       -1;}
20 cout<<endl<<"Flujo  de entrada abierto correctamente!.!."<<endl;
21 cout<<"Nro."<<'\t'<<"Dia"<<'\t'<<"Temp."<<'\t'<<"Humedad"<<endl; while(!
       flujo_entrada.eof()){flujo_entrada>>i>>dia>>temp>>humedad; cout<<i<<'\t
       '<<dia<<'\t'<<temp<<'\t'<<humedad<<endl;cin.get(cr);}
22 flujo_entrada.close();
23 if(!flujo_entrada.is_open())cout<<"Flujo de Entrada:cerrado"<<endl;
24 cout<<"Finalizando el programa!!";
25 return 0; }
```

Observaciones.

- Este programa abre un flujo (archivo) escribe y luego lee.
- Podemos ver que existen varias líneas que controlan que los flujos (archivos) sean abiertos y cerrados como corresponde.
- Se sugiere al lector que pruebe el mismo, verá que existe un pequeño error en la última del archivo de texto que la misma se repite, se deja como desafío para el lector ver por que sucede y como solucionar.

Tips: Es de buen programador, abrir archivos solo en modo input only (ios::in) , de esta manera se evitan escrituras o alteraciones del contenido del archivo.

Hasta ahora estuvimos trabajando con archivos del tipo secuencial, en estos ejemplo nosotros no controlamos la ubicación de DONDE vamos a escribir, basicamente lo que agregabamos lo hacíamos al final (usando el modo **ios::ate**), o al inicio destruyendo el contenido del

archivo en caso de que tuviera algo (usando el modo **ios::trunc**).

Veremos en la sección siguiente como posicionarnos dentro del archivo, para poder controlar el lugar donde insertemos algo en el fichero.

Ambas clases ofstream e ifstream poseen métodos para obtener o posicionarse dentro del fichero.

1.6.5. Métodos seekp y tellp de Clase ofstream

Estos dos métodos seekp y tellp que terminan con la letra p , de "put"y son métodos de ofstream que permiten trabajar con la posición del cursor dentro del archivo. Veamos en detalle cada uno de ellos.

1.6.5.1. Método seekp: (Set position in output sequence) Este método pone, determina la posición de la secuencia donde el próximo carácter será insertado en el flujo de salida o output. Existe la posibilidad de pasar arguementos para hacer referencias relativas. Veamos ejemplos.

```
1 ofstream flujo_output;
2 flujo_output.seekg(n);//posiciona en el byte enesimo
3 flujo_output.seekg(n, ios::cur);//posiciona a n bytes desde la posicion
      actual o corriente.
4 flujo_output.seekg(n, ios::end);//desde el final del archivo n bytes hacia
      atras
5 flujo_output.seekg(0, ios::end);//se posiciona al final del archivo
6 flujo_output.seekg(0);//se posiciona al inicio del archivo.
```

Es argumento que se pasa para indicar la posición , es del tipo "long int".

1.6.5.2. Método tellp: (Tell position in output sequence) Este método obtiene la posición actual del flujo de salida, no modifica nada solo obtiene la información de la posición actual. Retorna un long int .

Veamos un ejemplo de uso de ambos.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main()
5  {
6  long pos;
7  ofstream flujo_out("/home/daniel/prueba.txt");
8  flujo_out<< "123456789"<<endl;//mando una línea
9  pos=flujo_out.tellp();//obtengo la posicion esto me daría 11
10 cout<<"Ahora esta en:"<<pos<<endl;
11 flujo_out.seekp(pos-4);
12 flujo_out<<" abc"<<endl; //mando otra línea en 11-7
13 flujo_out.close();//Cierro el flujo return 0;
14 }
```

La salida de este programa será:
10
Y el archivo prueba.txt tendría:
123456abc.

Observaciones.
Vemos que el cursor queda en la posición 10, luego de escribir 1 en la posición 0, 2 en la posición 1, 3 en la posición 2..etc 9 en la posición 10.
Como pos=10 y se le resta 4, 6 será la nueva posición (0,1,2,3,4,5,6) seis es la septima posición y a partir de allí comienza a agregar abc.

1.6.6. Métodos seekg y tellg de Clase ifstream

Los métodos seekg y tellg son de ifstream, y sirven para posicionar el cursor en un archivo secuencial de entrada o input.

1.6.6.1. Método seekg: (Set position in input sequence) Este método cambia o determina la posición del cursor en la secuencia o flujo de entrada, por eso es parte de ifstream.

Tiene posibilidad de usar una referencia relativa y los argumentos para este caso son los mismos que para seekp de ofstream, así que no los repetiremos.

```
1  // read a file into memory
2  #include <iostream>     // std::cout
3  #include <fstream>      // std::ifstream
4
5  int main () {
6    std::ifstream is ("test.txt", std::ifstream::binary);
7    if (is) {
8      // get length of file:
9      is.seekg (0, is.end);
10     int length = is.tellg();
11     is.seekg (0, is.beg);
12
13     // allocate memory:
14     char * buffer = new char [length];
15
16     // read data as a block:
17     is.read (buffer,length);
18
19     is.close();
20
21     // print content:
22     std::cout.write (buffer,length);
23
24     delete[] buffer;
25   }
26
27   return 0;
28 }
```

1.6.6.2. Método tellg: (Get position in input sequence). Este método permite obtener la posición actual dentro del flujo de entrada de asociado a un archivo.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main () {
5  ifstream flujo_input ("/home/daniel/prueba.txt");
6  if (flujo_input) {
7  // Me ubico al final del archivo
8   flujo_input.seekg (0, flujo_input.end);
9  //Obtengo la cantidad de bytes del archivo
10 int longitud = flujo_input.tellg();
11 cout<<"El archivo tiene :"<<longitud<<" bytes"<<endl;}
12 else {cout<<"No se pudo asociar el flujo al arcivo"; return -1;}
13 cout<<"Finalizando el prorama"<<endl;
14 return 0;}
```

Observaciones:
Ver que longitud es int, esto por que el archivo es chico, en caso contrario debería ser long int.

1.7. Manejo de archivos Aleatorios.

Hasta ahora , hemos visto cómo trabajar con archivos secuenciales, crearlos, escribir , leer y buscar.

Primero diremos que se conoce con el nombre de "record" o "registro" un conjunto de datos que se encuentran en una línea , si hacemos una analogía con una planilla de cálculo sería algo así como una fila, y se llama "campo" o "field", a cada elemento del arreglo, siguiendo la analogía con la planilla de cálculo, sería como una celda dentro de una fila. Aclarado esto, usaremos estos terminos en esta sección.

Los archivos secuenciales son inapropiados para algunos tipos de aplicaciones, primero por que para acceder a un campo hay que recorrer TODOS los campos y segundo por que para insertar algún dato corremos el riesgo de sobre escribir algún otro. Veamos esto con algún ejemplo.

Supongamos que tenemos un archivo del tipo secuencial que contiene los siguiente registros o records, donde se quiere guardar los siguientes datos.

1 Juan 424555
2 Jorge 423555
3 Pedro 402555
4 Anibal 421555

Esto como es un archivo secuencial en realidad se encuentra dispuesto fisicamente de la siguente manera:

1 Juan 424555 2 Jorge 423555 3 Pedro 402555 4 Anibal 421555

Bien imaginemos que queremos editar este archivo y modificar el campo teléfono del segundo registro :**423555** (teléfono de Jorge), el nuevo dato que nos da Jorge que es un número de teléfono celular: **15123456**. El programa , busca la ocurrencia de ese *campo en el archivo*, busca en TODOS los campos!! , cuando encuentra esa cadena de caracteres escribe el nuevo dato, esto es lo que quedaría de la siguiente manera:

1 Juan 424555 2 Jorge 15123456 Pedro 402555 4 Anibal 421555

Observaciónes:

1. Podemos ver que se sobreescribió cierta información que es lo mismo que decir que se ***perdió***.
2. Otro punto a tener en cuenta es que por ejemplo el nro. de orden de esta simple agenda que está representado con ***un dígito*** , es un sencillo número del tipo ***int***, que por ejemplo generalmente usa 4 bytes, pero cuando superemos los nueve registros, el nro. será de ***dos dígitos*** , pero aún así seguirá siendo un entero de 4 bytes.

Podemos ver con punto 2 de la observación ,que el hecho de que trabajemos con caracteres en un archivo (entiendase con archivos de texto), no ayuda a resolver nuestros problemas.

Bueno el caso expuesto es un caso sencillo, y expone dos características importantes, una sobre escribir en los archivos de acceso secuencial y otra en particular sobre los archivos de texto de acceso aleatorio.

Dado que C++, no tiene definido nada para manejar el formato de los registros o records y en particular para los campos, es el programa el que tiene que realizar estos controles.

Algo que si nos ayuda de C++, es que dispone de algunas funciones que permiten trabajar sobre una cantidad determinadad de bytes, esto implicitamente quiere decir que al leer no importa el tipo de dato (char ,1 byte, int 4 bytes) , lo que definitivamente quiere decir que los archivos serán tratados como ***binarios!!***. Esta condición nos ayudaría con el punto 2 de la observación realizada.

1.7.1. Introducción a Métodos de lectura y escritura en Archivos de acceso aleatorio

Antes de avanzar con las funciones que permiten escribir y leer vamos a recordar algo de punteros.

1.7.1.1. Puntero a Char del tipo constante y puntero a char Los punteros son variables que guardan una dirección de memoria. Los punteros son variables especiales por no tener sólo un valor (dirección de memoria), pero también tienen asociado el valor a que apuntan (tipo de la variable).

Como cualquier variable su valor puede ser constante o no.

La palabra clave const declara que un valor no es modificable por el programa. Puesto que no puede hacerse ninguna asignación posterior a un identificador especificado como const

El tipo "const char" es una constante y el valor **no puede ser cambiado** por lo tanto SIEMPRE debe tener un valor al inicializar, ya que luego no se puede modificar, mientras que una variable , si puede ser cambiado, veamos unos ejemplos:

```
const char X = 'A'; //X es un Valor constante...no puede ser modificado
char Y = 'B'; // Y es Variable, si puede ser modificado.
Y='C'; //ok.
X='D';// El compilador acusaría error:assignment of read-only varaible 'X'
const int peso = 45; // ok, una constante int
const int altura ;//Error , no se inicializa al declarar la constante
const float talla = 40.2; // ok, una constante float
const float talla; // Error, no se inicializa al declarar la constante
```

De la misma manera que a las variables, a los punteros se les puede anteponer la palabra const, veamos como sería:

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {char msg[]="Sol";
5  char *const pt1 = msg;// pt1 puntero constante arreglo de char
6  char const *pt2 = "Luna";// pt2 puntero a arreglo de char constante
7  //const char *pt2 = "Luna";  // es lo mismo que la línea anterior.
8  pt1++; //Esta linea tiraría error, ya que modifico un puntero constante.
9  *(pt1+1)='a';//Esta linea remplaza al 'o' de Sol por una 'a'y no tira error
       por que el arreglo no es constante, si es constante el puntero al
       arreglo de char
10 cout<<pt1<<endl;
11 *(pt2+1)='o';//Esta línea tiraría error por que modifica un arreglo
       constante
12 pt2++;//Esta línea no tira erro por que modifica el puntero pt2 que NO es
       constante. do{cout<<*pt2;pt2++;}while(*(pt2)!=0);
13 }
```

Puntero a un caracter constante.

Este tipo de definición se usa para definir tipicamente cadenas de caracteres. Con un puntero a constante caracter puedo hacer que apunte a varios elementos del arreglo o strings, pero no puede modificar los strings.

```
1  #include <iostream>
2  using namespace std;
3  void imprimefrace (const char *);//prototipo, recibe un puntero a un
       arreglo constante
4  int main() {
5  char frace[]="Hola mundo cruel"; //ver que frace el nombre del arreglo es
       un puntero.
6  cout <<"La frace es: \n";
7  imprimefrace(frace); //paso puntero a un string constante.
```

```
8  return 0; }
9  void imprimefrace (const char *prt)
10 {
11 *(prt+1)='B';//Esta línea me tiraría error! ya que intento modificar una
       constante.
12 for(;*prt!='\0';prt++) //Muestro c/ caracter del string hasta que venga el
       nulo.
13 cout<<*prt; }
```

En la función ***prt** , no puede modificar el caracter al cual apunta.. pero puede apuntar a varios.. es mas a cada uno de los caracteres del string!!.

Un puntero no constante a los datos constantes es un puntero que puede ser modificado para apuntar a cualquier elemento de datos del tipo adecuado, pero los datos a las que apunta no se puede modificar a través de ese puntero.

1.7.2. Método Write y read.

El método write (pertence a ofstream) ,espera recibir un **const char * (puntero a una arreglo constante)** y en número de n caracteres a insertar. En este caso entiendase que con caracteres se está pensando en que un caracter es un byte, por lo que podemos pensar que escribe una cantidad de bytes.

El método read (pertence a ifstream) realiza la operación de lectura y espera recibir un **char * (puntero a un arreglo)** y un número n de bytes y vuelca el contenido de n bytes al puntero pasado como argumento.

En un archivo de texto, facilmente podemos usar <<, para sacar una variable, pero recordemos que un entero puede tener 1, o mas caracteres, pero desde el punto de vista de bytes, serán normalmente 4 bytes.

Veamos un ejemplo de uso de read y write que el contenido de un archivo y lo copia a otro, en este caso solo los toma como bytes.

```
1  // Copia en contenido de un archivo a otro
2  #include <fstream>
3  using namespace std;
4  int main ()
5  {
6  ifstream infile ("/home/daniel/componentes.txt", ios::binary);//Ver que el
       archiv es BINARIO
7  ofstream outfile ("/home/daniel/componentes1.txt",ios::binary);//Ver que el
        archiv es BINARIO
8  // Obtengo el tamanio del archivo
9  infile.seekg (0,infile.end);//Me ubico al final
10 long size = infile.tellg(); //obtengo la cantidad de bytes del archivo.
11 infile.seekg (0); //me ubica en el comienzo del archivo.
12 // Reservo espacio para almacenar el archivo
13 char* buffer = new char[size];//creo un espacio con del tamanio del archivo
       .
14 // Leo el contenido del archivo
15 infile.read (buffer,size); //paso puntero y cantidad.
16 // Escribo el contenido del Archivo
17 outfile.write (buffer,size);paso puntero y cantidad.
18 // Libero la memoria reservada dinamicamente
19 delete[] buffer;
20 outfile.close();//cierro los flujos
21 infile.close();
22 return 0;
23 }
```

Ahora, vamos a mostrar dos archivos que escriben lo mismo: 123456789 en un archivo, pero en un caso vamos a ingresar el número como “ENTERO” y en otro caso como un “ARREGLO DE CHAR” y veremos como cambia el tamaño de los datos a guardar.

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
int  cant;
const char arreglo[]= "123456789";//Ver que es un arreglo de char como es
    const es solo de lectura
char arreglo1[10]; // como arreglo es const. defino otro arreglo para
    escribir
cout<<"Vamos a guardar en el Archivo binario, el nro. :" <<arreglo<<endl;
cant=sizeof(arreglo);
cout<<"Que tiene: "<<cant<<" bytes."<<endl;
ofstream out ("/home/daniel/data.bin", ios::binary);
if (out.is_open())
        {out.seekp (0);
        out.write (arreglo, cant);
        out.close();
        }
else cout << "No es posible abrir el archivo" << endl;
cout<<"Vamos a leer y mostrar el contenido del archivo"<<endl;
ifstream in ("/home/daniel/data.bin",ios::binary);
if (in.is_open())
        {cout << "Archivo abierto para lectura correctamente." << endl;
    in.seekg (0, ios::beg);
        in.read (arreglo1, cant);
        in.close();
    cout<<"El contenido es: "<<arreglo1;
    }
else cout << "No es posible abrir el archivo" << endl;
return 0;
}
```

La salida de este archivo sería:

```
Vamos a guardar en el Archivo binario, el nro. :123456789
Que tiene: 10 bytes.
 Vamos a leer y mostrar el contenido del archivo
Archivo abierto para lectura correctamente.
El contenido es: 123456789

------------------
(program exited with code: 0)
Press return to continue
```

Observaciones:

- El archivo tiene un total de 10bytes.
- El archivo es del tipo binario.

Veamos ahora como sería el caso de guardar el mismo nro. pero como entero.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
int  x=123456789;//Ver que es un Entero
cout<<"Vamos a guardar en el Archivo binario, el nro. :" <<x<<endl;
cout<<"Que tiene : "<<sizeof(int)<< " bytes."<<endl;
ofstream out("/home/daniel/data.bin", ios::out|ios::binary|ios::ate);
if (out.is_open())
        {out.seekp (0);
        out.write ((const char*)&x, sizeof (int));
        out.close();
        }
```

```
15 else cout << "No es posible abrir el archivo" << endl;
16 cout<<" Vamos a leer y mostrar el contenido del archivo"<<endl;
17 ifstream in ("/home/daniel/data.bin",ios::binary);
18 if (in.is_open())
19         {cout << "Archivo abierto para lectura correctamente." << endl;
20         in.seekg (0, ios::beg);
21         in.read ((char*)&x, sizeof (int));
22         in.close();
23         cout<<"El contenido es: "<<x;
24         }
25 else cout << "No es posible arbrir el archivo" << endl;
26 return 0;
27 }
```

La salida de este archivo sería:

```
Vamos a guardar en el Archivo binario, el nro. :123456789
Que tiene : 4 bytes.
 Vamos a leer y mostrar el contenido del archivo
Archivo abierto para lectura correctamente.
El contenido es: 123456789

-----------------
(program exited with code: 0)
Press return to continue
```

Observaciones:
El archivo tiene un total de 4 bytes. Seis bytes menos que el anterior y estamos guardando en mismo nro., esto es un 60 % menos de capacitad para guardar el mismo dato.
El archivo es del tipo binario.

Hay un par de cosas que tenemos que aclara, sobre el primer argumento del método write y read.

Estos métodos esperan recibir un puntero a un arreglo char constante para el caso de write y un puntero a un arreglo de char para el caso de read, por lo que debemos acondicionar lo que tenemos como arguemento.

Primer caso:
const char arreglo[]= "123456789";
char arreglo1[10];
out.write (arreglo, cant);
in.read (arreglo1, cant);
Segundo caso:
int x=123456789;
out.write ((const char*)&x, sizeof (int));
in.read ((char*)&x, sizeof (int));

Vemos que en el primer caso el nombre del arreglo ya es un puntero a un arreglo del tipo char, en este caso write y read no necesitan mayores modificaciones del argumento

Para el segundo caso, como el nro. que se tiene es un entero, para el caso de :

write: debemos pasarle un puntero a un arreglo constante, pero tenemos un entero, por lo que obtenemos la direción del entero y le hacemos un casting explicito a const char.

```
out.write ((const char*)&x, sizeof (int));
```

read: debemos pasar un puntero a un arreglo de char , hacemos entonces un cast explicito.

```
in.read ((char*)&x, sizeof (int));
```

Mostraremos ahora dos programas que hacen lo mismo escribir, 10000,20000,30000,40000,50000, en un archivo y luego compararemos nuevamente el tamaño. No aportará nada nuevo pero podremos repasar la forma de uso de read y write para el caso de char e int.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main ()
5  {
6  int cant,t=0;
7  ofstream out("/home/daniel/file.dat", ios::binary);
8  int arreglo [5]= {10000,20000,30000,40000,50000};
9  cout<<"Valores a Guardar en el Archivo: ";
10 for(int i=0;i<5;i++){cout<<arreglo[i];cout<<'\t';}
11 cant=sizeof(arreglo);
12 cout<<endl<<"Cantidad de bytes: "<<cant<<endl;
13 out.write((char*)&arreglo,cant);
14 out.close();
15 //Vamos a leer y mostrar por pantalla lo que contiene archivo binario
16 //creo un buffer para asociar al archivo.
17 ifstream in("/home/daniel/file.dat", ios::binary);
18 in.seekg(0,ios::beg);
19 int  *buffer= new int[cant];
20 in.read(reinterpret_cast< char * >(buffer),cant);
21 cout<<"El contenido es: ";
22 do{ cout<<*(buffer+t)<<'\t';t++;}while(t<5);
23 return 0;
24 }
```

La salida de este programa sería:

```
Valores a Guardar en el Archivo: 10000  20000   30000   40000   50000
Cantidad de bytes: 20
El contenido es: 10000  20000   30000   40000   50000

------------------
(program exited with code: 0)
Press return to continue
```

Observaciones:
Se recomienda al lector comentar la línea out.close() y ver la salida del programa.

Ahora para el caso de hacer lo mismo pero con Arreglo de Char.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main ()
5  { int
6  cant,t=0;
7  ofstream out("/home/daniel/file.dat", ios::binary);
8  char arreglo [5][6]= {"10000","20000","30000","40000","50000"};
9  cout<<"Valores a Guardar en el Archivo: "; for(int i=0;i<5;i++){for(int j
       =0;j<6;j++)cout<<arreglo[i][j];
10 cout<<'\t';} cant=sizeof(arreglo);
11 cout<<endl<<"Cantidad de bytes: "<<cant<<endl;
12 out.write((char*)&arreglo,cant);
13 out.close();
14 ifstream in("/home/daniel/file.dat", ios::binary);
15 in.seekg(0,ios::beg);
16 char *buffer= new char [cant];
17 in.read((char *)buffer,cant);
18 cout<<"El contenido es: "; for (int i=0;i<5;i++)
19 {
20 do{ cout<<*buffer;buffer++;
21         }while(*(buffer)!=0);
```

```
22 cout<<'\t'; }
23 }
```

La salida de este programa sería:

```
Valores a Guardar en el Archivo: 10000  20000   30000   40000   50000
Cantidad de bytes: 30
El contenido es: 10000  20000   30000   40000   50000

-----------------
(program exited with code: 0)
Press return to continue
```

1.7.3. Métodos read y write con Estructuras

Bueno, ahora que vimos las manera de escribir y leer de un archivo de acceso aleatorio y del tipo binario, veamos como darle alguna estructura al mismo usando Estructuras. Comencemos por algo sencillo.

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  struct datos{float temp};//Escturctura sencilla
5  int main()
6  {
7  datos muestra[3];
8  float z;
9  ofstream out("/home/daniel/muestras.dat", ios::binary);
10 for(int i=0; i<3;i++)
11         {
12         cout<<"Ingrese la temperatura: "<<endl;
13         cin>>muestra[i].temp;
14         out.write((const char*)&(muestra[i].temp),sizeof(float));
15         }
16         out.close();
17 ifstream in("/home/daniel/muestras.dat", ios::binary);
18 for(int i=0; i<3;i++)
19         {
20         in.read((char*)&z,sizeof(float));
21         cout<<"Temperatura: "<<z<<'\t';
22         }
23 in.close();
24 return 0;
25 }
```

Ahora escribiremos una estructura mas real, es decir que no sea solo una esctructura con un float, esto sería mas parecido a una Estructura un poco mas compleja. Veamos como quedaría el código:

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  struct datos{char dia[9];float temp;};
5  int main()
6  {
7  datos *muestra;
8  char dia_semana[9];
9  int cantidad;
10 cout<<"Ingrese cantidad de instancias: ";
11 do{ cin>>cantidad; }while(cantidad<=0);
12 muestra=new datos[cantidad];
13 if(!muestra){cout<<"Saliendo, no hay espacio en Memoria";return -1;}
14 ofstream out("/home/daniel/muestras.dat", ios::binary);
15 for(int i=0; i<cantidad;i++)
```

```
16          {
17          cout<<"Ingrese día (Lu , Ma,..): ";
18          cin>>(muestra+i)->dia;
19          out.write((muestra+i)->dia,(9*(sizeof(char))));
20          cout<<"Ingrese la temperatura: ";
21          cin>>(muestra+i)->temp;
22          out.write((char*)&((muestra+i)->temp),sizeof(float));
23          cout<<endl;
24          }
25  out.close();
26  ifstream in("/home/daniel/muestras.dat", ios::binary);
27  float z;
28  for(int i=0; i<cantidad;i++)
29          {
30          in.read(dia_semana,(9*(sizeof(char))));
31          cout<<"Dia: "<<dia_semana;
32          cout<<'\t';
33          in.read((char*)&z,sizeof(float));
34          cout<<"Temperatura: "<<z<<endl;
35          }
36     in.close();
37  return 0;
38  }
```

Veamos como sería la salida cuando corremos este programa.

```
Ingrese cantidad de instancias: 4
Ingrese día (Lu , Ma,..): Lunes
Ingrese la temperatura: 1.2

Ingrese día (Lu , Ma,..): Martes
Ingrese la temperatura: 2.3

Ingrese día (Lu , Ma,..): Miercoles
Ingrese la temperatura: 3.4

Ingrese día (Lu , Ma,..): Jueves
Ingrese la temperatura: 4.5

Dia: Lunes      Temperatura: 1.2
Dia: Martes     Temperatura: 2.3
Dia: Miercoles  Temperatura: 3.4
Dia: Jueves     Temperatura: 4.5


------------------
(program exited with code: 0)
Press return to continue
```

Observaciones.

- Ver que si bien se usa una esctructura, esta sola es para definir el formato y tipo de datos a cargar, en si las instancias de la estructura no se utilizan luego, por lo que se podría haber creado una sola instancia e ir volcando el contenido de esta a el archivo.
- Se deja al lector ver que sucede si el nombre supera a 9 caracteres.

1.8. Pasando argumentos a main.

Muy a menudo necesitamos especificar valores u opciones a nuestros programas cuando los ejecutamos desde la línea de comandos. Por ejemplo, si queremos que un programa cree un archivo pasado como argumento, necesitaremos especificar el nombre del archivo. Hasta

ahora siempre hemos usado la función main sin parámetros, sin embargo, como veremos ahora, se pueden pasar argumentos a nuestros programas a través de los parámetros de la función main. Para tener acceso a los argumentos de la línea de comandos hay que declararlos en la función main, la manera de hacerlo puede ser una de las siguientes:

*int main(int argc, char *argv[]);*

El primer parámetro, "argc" (**arg**ument **c**ounter), es el *número de argumentos* que se han especificado en la línea de comandos.

El segundo, "argv", (**arg**ument **v**alues) es un array de cadenas, que contiene en cada uno de los elementos del arreglo, los argumentos especificados en la línea de comandos.

Por ejemplo, si nuestro programa se llama "crear_archivo", y lo ejecutamos con la siguiente línea de comandos:

crear_archivo arg1 arg2 arg3 arg4

Argc valdrá 5, que es la cantidad de argumentos, ya que el nombre del programa incluido el camino también se cuenta como un argumento y argv[] contendrá la siguiente lista de los argumentos.

Veamos un ejemplo:

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[]){
cout<<"Cantidad de Argumentos:"<<argc<<endl;
for(int i=0;i<argc; i++)cout<<"Argumento # "<<i<<" es: "<<argv[i]<<endl;
return 0; }
```

Observaciones.
Si este programa se llamara crea_archivos y desde la línea de comando escribimos: crea_archivo alfa beta gama y presionamos Enter, por pantalla se verá:
./crear_archivo alfa beta gama
Cantidad de Argumentos: 4
Argumento # 0 es: /home/usuario/crear_archivo
Argumento # 1 es: alfa
Argumento # 2 es: beta
Argumento # 3 es: gama

Veamo

como podemos usar esto para escribir un código que nos permita controlar si los argumentos pasados son correctos.

El código de este programa, recibe al llamar al ejecutable un nombre que se utiliza como nombre de archivo, luego se escriben hasta 256 caracteres en el archivo o hasta que se presione ESC. Veamos el código:

```
#include <iostream>
#include <fstream>
# define ESC 27 //Defino la tecla ESC.
using namespace std;
int main(int argc, char *argv[])
{int i=0;char ingreso[256];
//Controlo que se haya pasado el nombre del archivo como argumento.
if(argc!=2){cout<<"Faltan parámetros al llamar al archivo"; return 1;}
fstream flujo(argv[1],ios::in | ios::out |ios::app);//Creo el flujo y
    asocio al archivo.
//Controlo si el archivo se puede abir
if (!argv[1]) {cout<<"No se puede abrir el archivo";return 1;}
cout<<"Ingrese el Texto, con ESC sale, Nombre del archivo:" <<argv[0]<<endl
    ;
cin.getline(ingreso,256,ESC);
flujo<<ingreso;//vuelco el contenido del String al Flujo
flujo.close(); }
```

Referencias

[1] http://c.conclase.net/curso/index.php?cap=042

[2] http://www.chuidiang.com/clinux/ficheros/fichero-texto-cpp.php

[3] http://www.cplusplus.com/doc/tutorial/typecasting/

[4] http://msdn.microsoft.com/en-us/library/e0w9f63b(v=vs.80).aspx

[5] http://c.conclase.net/ficheros/?cap=002b#FIC_FUNCCPP

[6] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Estructuras_II

[7] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Streams

Índice