2. Archivos lógicos y físicos

- El significado más genérico de archivo o fichero es de flujo o sucesión de elementos que entran o salen del ordenador.

- Normalmente hablaremos de ficheros cuando el caudal de datos viaje o proceda desde el disco o soporte de almacenamiento estable (fichero en disco o archivo).

- Un fichero podría definirse como un elemento de almacenamiento de datos sobre un medio permanente con las siguientes características:

  - Es una estructura de datos dinámica.
  - Permite el almacenamiento permanente de la información.
  - No tiene un tamaño fijo preestablecido ni un máximo.
  - Tener un acceso lento a la información, si lo comparamos con los accesos a memoria principal.

- **Archivo lógico** es una abstracción que nos ofrece el lenguaje de programación, de forma que nos permita manejar ficheros independientemente de su representación, almacenamiento, etc.
- **Archivo físico** es una colección de datos real que ocupa un espacio exacto y concreto en el disco físico y que tiene asociado un nombre, un tamaño y otras informaciones adicionales.
- **Asignación** es el momento en el que se asocia un fichero lógico con su correspondiente fichero físico.

3. Tipos de archivos

- **En cuanto al método de acceso.**

El modo de acceso a los archivos depende principalmente del soporte empleado para los mismos y del modo físico en el que se ha organizado su información.

  - **Acceso secuencial**: se accederá a cada elemento del archivo uno tras otro en el mismo orden en el que se situaron.

  - **Acceso directo**: permite acceder a un elemento determinado sin tener que acceder previamente a otros precedentes.

- **En cuanto a su contenido.**

  - **Ficheros de texto**.

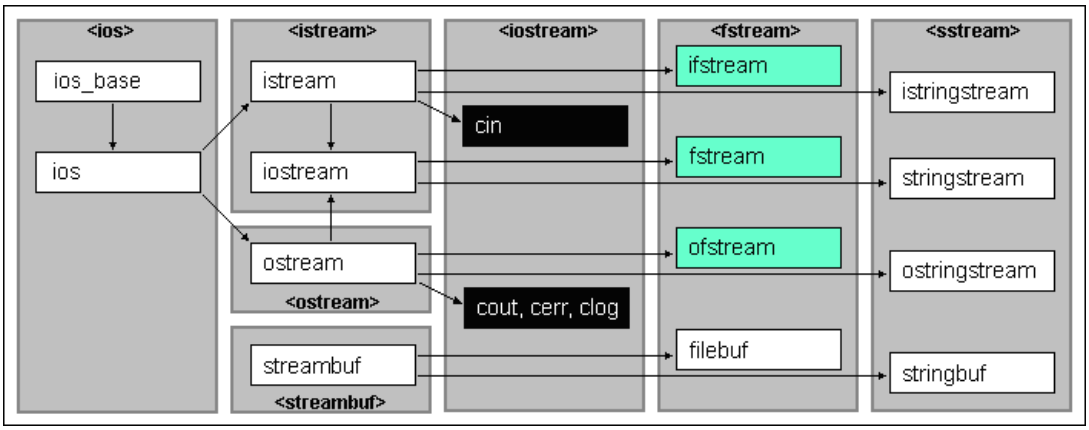
  - **Ficheros binarios**.

4. Operaciones con archivos

- **Declaración y apertura de ficheros.**

Las funciones específicas en C++ para el manejo de archivos parten de vincular, en su apertura, dicho archivo a un flujo (*stream*). Hay tres tipos de flujos:

  - Entrada → clase *ifstream*, para leer de un fichero
  - Salida → clase *ofstream*, para escribir en un fichero
  - Entrada/salida → clase *fstream*, para leer y/o escribir



La asociación de un archivo lógico con un archivo físico se realiza con el método **open**:

```
void open( char* nombreFichero, ios_base::openmode modo )
```

Donde *nombreFichero* es la ruta completa de localización del archivo, pudiendo incluir un especificador de camino. Y *modo* determina cómo se abre el archivo, debiendo ser uno (o varios, uniéndolos con |) de los valores:

ios::in	Apertura para lectura
ios::out	Apertura para escritura
ios::binary	Apertura en modo binario (no en modo texto)
ios::app	Para añadir únicamente por el final del fichero
ios::trunc	Borra previamente el contenido del fichero

Si el tipo de fichero es *ifstream* u *ofstream*, existen valores implícitos para el modo, sin necesidad de especificarlos:

- Para *ifstream* el modo implícito es *ios::in*
- Para *ofstream* el modo implícito es *ios::out*

Se puede hacer uso de los constructores de las clases *ifstream*, *ofstream* y *fstream*. Estos constructores tienen los mismos parámetros que la función *open*, simplificando de esta forma las operaciones de apertura. Así, son equivalentes:

```
ofstream ficheroSalida("datos", ios::app | ios::binary);
```

```
ofstream ficheroSalida;
ficheroSalida.open("datos", ios::app | ios::binary);
```

Si la apertura de un fichero, a través de open, registrase algún tipo de error, la variable de flujo asociada recibiría un valor de 0.

De forma análoga al uso de open, las funciones constructoras asignarán un valor de 0 a las variables de flujo asociadas, en caso de producirse error.

```
ofstream  salida("fich");
if (!salida)
 {
   cout << "Se ha producido error al abrir el archivo";
  /*Tratamiento del error*/
 }
```

- **Comprobación del estado de un fichero.**

En C++, estos métodos pueden usarse para comprobar el estado actual de un fichero. Devuelven un valor *bool*:

`fail()`	Devuelve `verdadero` si hubo error lógico o de lectura/escritura en la última operación con el fichero
`eof()`	Devuelve `verdadero` si se ha alcanzado el final de un fichero abierto para lectura

El método **`clear()`** permite borrar la información de error asociada a un fichero.

- **Cierre de ficheros.**

Los ficheros deben cerrarse una vez que dejen de usarse. Para ello se hace uso del método **close**:

```
   ficheroSalida.close();
```

- **Ficheros de texto.**

La lectura y escritura en un fichero de texto es bastante simple, pues puede hacerse uso de los operadores **<<** y **>>** . Pueden llevarse a cabo ciertas traducciones de caracteres, que pueden ser evitadas tratando el fichero en modo binario.

Como ejemplo, el siguiente código va leyendo líneas de texto (hasta leer la cadena "salir") y las escribe en un fichero. Luego, recorre dicho fichero para mostrar su contenido por pantalla.

```
#include <iostream>
#include <fstream>
#include "string.h"
using namespace std;
int main()
 {
  char cadena[100];

  ofstream salida("prueba.txt");
  if (!salida.fail())
  {
    do
    {
      cin >> cadena;
      salida << cadena << endl;
    } while (strcmp(cadena, "salir") !=0);
    salida.close();

    cout << "\nAhora se muestra el contenido del fichero\n";
    ifstream ent("prueba.txt");
    ent>>cadena;
    while (!ent.eof())
    {
     cout <<cadena<<endl;
     ent>>cadena;
    }
   ent.close();
   }
  return 0;
}
```

En este otro ejemplo, se almacenan los datos de la edad y el sueldo en un fichero, y luego se lee dicha información para mostrarla por pantalla.

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

int main() {
  int edad;
  float sueldo;

  ofstream salida("datos");
  cout << "Edad: ";
  cin >> edad;
  cout << "Sueldo: ";
  cin >> sueldo;
  salida << edad << " " << sueldo;
  salida.close();

  ifstream entrada("datos");
  entrada >> edad >> sueldo;
  cout << edad << endl << sueldo;
  entrada.close();

  return 0;
}
```

- **Ficheros en modo binario.**

Para los ficheros binarios, se dispone de otros métodos más apropiados para trabajar con ellos:

```
 read( char* destino, int num )
```

Para la lectura desde el fichero, donde *destino* es un puntero a la zona de memoria donde escribir los datos leídos del flujo, y *num* qué cantidad de bytes se desea leer.

En el caso de que se alcance el final del archivo antes de haber leído *num* bytes, simplemente se obtienen los bytes disponibles.

```
int gcount()
```

Mediante el método *gcount* se puede saber cuántos bytes se han leído realmente en la última operación binaria de entrada.

```
write(char* origen, int num)
```

Para la escritura en un fichero, donde *origen* es un puntero a la zona de memoria donde residen los datos y *num* indica el número de bytes a escribir.

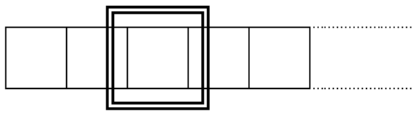
En este ejemplo se escriben en un fichero una serie de fichas de alumnos (nombre y nota) y luego se muestra el contenido grabado:

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

struct ficha {
  char nombre[50];
  float nota;
};

int main() {
  ficha fichas[3] = { {"Ana", 7.2} ,{"Juan", 6.3}, {"Rodolfo",8} };
  int i;

  ofstream salida("fichas", ios::binary);
  if(salida.fail()) {
    cout << "No se puede abrir el archivo fichas";
  }
  else
  {
    salida.write((char*) fichas, sizeof(fichas));
    salida.close() ;

    for(i=0; i<3; i++) {
      fichas[i].nota = 0;
    }
```

```
    ifstream entrada("fichas", ios::binary);
    entrada.read((char *) fichas, sizeof(fichas));
    for(i=0; i<3; i++)
      cout << fichas[i].nombre << " " << fichas[i].nota << endl;
    entrada.close() ;
  }

  return 0;
}
```

- **Acceso secuencial y directo.**

Lo visto ahora permite un acceso secuencial a un fichero. Pero en C++ los ficheros disponen de un indicador de la posición donde se realizará la siguiente operación de entrada/salida sobre el mismo.



Este indicador de posición avanza automáticamente tras efectuar cada operación. Y para cambiar dicha posición se disponen de los métodos *seekg* y *seekp*, según se pretenda realizar una lectura o escritura:

```
seekg( streampos pos )
```

```
seekp( streampos pos )
```

Siendo *streampos* el tipo empleado para definir posiciones en un *stream*.

Estos métodos admiten también su empleo con un desplazamiento (bytes) desde una posición dada:

```
seekg( streamoff desplazamiento, seekdir pos )
```

```
seekp( streamoff desplazamiento, seekdir pos )
```

Siendo *streamoff* el tipo para representar un desplazamiento y *seekdir* el

tipo para representar la posición origen:

ios::beg	Principio del archivo
ios::cur	Posición actual
ios::end	Final del archivo

Para consultar la posición actual en un fichero se usan los métodos *tellg* y *tellp*:

```
streampos tellg()
```

```
streampos tellp()
```

En el ejemplo siguiente se muestra cómo saber el número de bytes de un fichero. El razonamiento seguido es abrir el fichero en modo lectura, situar el puntero de entrada en la última posición del fichero (desplazamiento 0 desde el final) y mostrar el valor proporcionado por *tellg*:

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

int main() {
  ifstream fich("fichas");

  if (!fich.fail()) {
    fich.seekg(0, ios::end);
    cout << fich.tellg();
    fich.close();
  }
  else
    cout << "No se encuentra el fichero";

  return 0;
}
```

- **Lectura en Acceso directo:**

Al igual que en el acceso secuencial, las operaciones de lectura en acceso directo consisten en copiar la información contenida en un elemento del fichero sobre una variable del programa localizada en memoria principal, pero en este caso indicaremos necesariamente la posición del elemento del fichero desde la que deseamos realizar la lectura. Para ello actuamos del siguiente modo:

- Uso de seekg para posicionarnos.
- Uso de read para leer.

- **Escritura en Acceso directo:**

De igual modo que en el acceso secuencial, se trata de copiar la información contenida en una variable del programa sobre un elemento del fichero, pero en este caso debemos indicar la posición del fichero en la que deseamos situar el elemento.

Para ello actuamos del siguiente modo:

- Uso de seekp para posicionarnos.
- Uso de write para escribir.

Es importante destacar que la escritura en una posición del fichero no inserta sino sobrescribe el contenido del elemento de esa posición.

Veamos el siguiente ejemplo

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include "string.h"
#include "stdlib.h"
using namespace std;
#define MAX_STR 30
```

```
typedef char cadena[MAX_STR]; //define tipo string

//-----Definición de la Clase coche-------------
class Coche
{
  long NumKm;
  cadena Matricula;
 public:
  void setMat(cadena mat){strcpy(Matricula,mat);};
  void setNumKm(long num){NumKm=num;};
  const char *getMat(){return Matricula;};
  long getNumKm(){return NumKm;};
};

int main()
 {
  Coche C;

  char op;
  int indice;
  cadena mat;
  long Km;
  fstream f;
  f.open("datos1.dat",ios::in|ios::out|ios::binary);//vemos     si     datos1.dat
//existía previamente
  if (f.fail())
   //el fichero no existe, debo crearlo
   {
    f.close();
    f.clear();
    f.open("datos1.dat",ios::out|ios::binary); //se crea el fichero
    f.close();
    f.clear();
    f.open("datos1.dat",ios::in|ios::out|ios::binary); //ya abrimos de nuevo el
//fichero tras crearlo
   }
  if (f)
  {
   do
   {
   system ("cls");
   cout <<"Elija operación\n";
   cout <<"1-Ver Matrícula y Km de un Coche\n";
   cout <<"2-Asignar Matrícula y Km a un Coche\n";
   cout <<"3-  Salir\n";
   op=getch();
```

```
switch(op)
{
case '1':
 cout <<"Introduzca N° vehículo a consultar\n";
 cin >>indice;
 if (indice>0)
  {
    f.seekg(sizeof(Coche)*(indice-1),ios::beg);
    if (((int)f.tellg()!=(sizeof(Coche)*(indice-1)))||(f.fail()))
    {
     cout<<"Se produce Error\n";
     f.clear();
    }
   else
    {
     f.read((char *) &C,sizeof(Coche));
     if (!f.fail())
      cout << "Matrícula: " << C.getMat()<< " Km: " << C.getNumKm();
     else
      {
        cout<<"Se produce error\n";
        f.clear();
       }
    }
   getch();
  }
 break;
case '2':
 cout <<"Introduzca matrícula\n";
 cin >>mat;
 C.setMat(mat);
 cout <<"Introduzca sus Km\n";
 cin >>Km;
 C.setNumKm(Km);
 cout <<"Introduzca el Nro de Vehículo\n";
 cin>>indice;
 if (indice>0)
  {
    f.seekp(sizeof(Coche)*(indice-1),ios::beg);
    if ((int)f.tellp()!=sizeof(Coche)*(indice-1))
    {
     cout<< "Se produce Error\n";
    }
   else
    {
      f.write((char *)&C,sizeof(Coche));
```

```
       }
     }
    break;
   case '3': break;
   default : cout <<"Opción Incorrecta";
   }
   }while (op!='3');
  f.close();
 }
 else
  cout<<"fichero no abierto";
 system("pause");
 return 0;
}
```