UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ASIGNATURA: PROGRAMACIÓN DE SISTEMAS

PRÁCTICA LABORATORIO N°06

OBJETIVO

- La presente práctica de laboratorio tiene como objetivo trabajar con ficheros en C++.

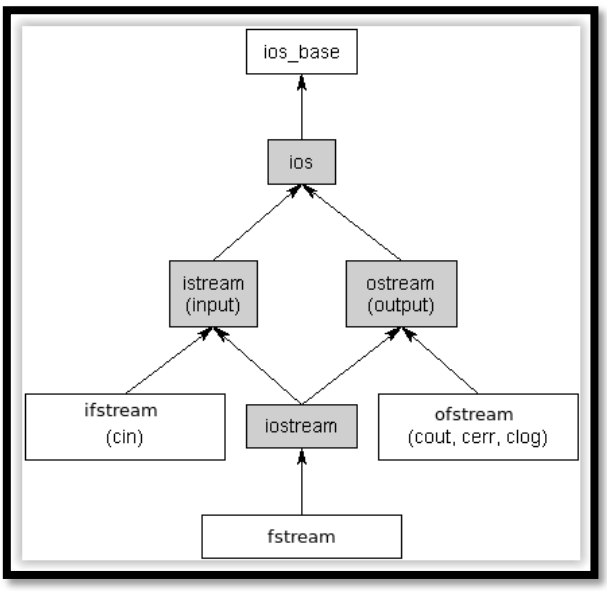
TEMA

- Trabajar con Ficheros.
- Crear un Fichero de Salida.
- Ficheros Binarios.
- Ficheros de Acceso Aleatorio.
- Ficheros de Entrada y Salida.

MARCO TEÓRICO

TRABAJAR CON FICHEROS:

Los archivos o ficheros son la forma en la que C++ permite el acceso al disco. Todos los procesos tienen abiertos, por defecto, los archivos 0(entrada), 1(salida) y 2(salida de errores), de manera que en C++ se corresponden con los objetos cin, cout y cerr. De estos últimos, el primero pertenece a la clase ifstream, que a su vez desciende de istream (flujo de entrada). Los dos últimos pertenecen a la clase ofstream, que desciende de la clase ostream (flujo de salida). Una jerarquía aproximada puede verse a continuación.



Archivo del sistema operativo	Clase	Objeto
0	ifstream	cin
1	ofstream	cout
2	ofstream	cerr

Usar streams facilita mucho el acceso a ficheros en disco, veremos que una vez que creemos un stream para un fichero, podremos trabajar con él igual que lo hacemos con cin o cout.

Mediante las clases ofstream, ifstream y fstream tendremos acceso a todas las funciones de las clases base de las que se derivan estas: ios, istream, ostream, fstreambase, y como también contienen un objeto filebuf, podremos acceder a las funciones de filebuf y streambuf.

❖ CREAR UN FICHERO DE SALIDA:

Vamos a crear un fichero mediante un objeto de la clase ofstream, y posteriormente lo leeremos mediante un objeto de la clase ifstream:

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

int main() {
  char cadena[128];
  // Crea un fichero de salida
  ofstream fs("nombre.txt");

  // Enviamos una cadena al fichero de salida:
  fs << "Hola, mundo" << endl;
  // Cerrar el fichero,
  // para luego poder abrirlo para lectura:
  fs.close();

  // Abre un fichero de entrada
  ifstream fe("nombre.txt");

  // Leeremos mediante getline, si lo hiciéramos
  // mediante el operador << sólo leeríamos
  // parte de la cadena:
  fe.getline(cadena, 128);

  cout << cadena << endl;

  return 0;
}
```

Veamos otro ejemplo sencillo, para ilustrar algunas limitaciones del operador >> para hacer lecturas, cuando no queremos perder caracteres.

Supongamos que llamamos a este programa "streams.cpp", y que pretendemos que se autoimprima en pantalla:

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

int main() {
  char cadena[128];
  ifstream fe("streams.cpp");

  while(!fe.eof()) {
    fe >> cadena;
    cout << cadena << endl;
  }
  fe.close();

  return 0;
}
```

El resultado quizá no sea el esperado. El motivo es que el operador >> interpreta los espacios, tabuladores y retornos de línea como separadores, y los elimina de la cadena de entrada.

❖ FICHEROS BINARIOS:

Los ficheros binarios son más útiles para guardar información cuyos valores no estén limitados. Por ejemplo, para almacenar imágenes, o bases de datos. Un fichero binario permite almacenar estructuras completas, en las que se mezclen datos de cadenas con datos numéricos.

En realidad, no hay nada que nos impida almacenar cualquier valor en un fichero de texto, el problema surge cuando se almacena el valor que el sistema operativo usa para marcar el fin de fichero en un archivo de texto. En MS-DOS ese valor es 0x1A. Si abrimos un fichero en modo de texto que contenga un dato con ese valor, no nos será posible leer ningún dato a partir de esa posición. Si lo abrimos en modo binario, ese problema no existirá.

Los ficheros que hemos usado en los ejemplos anteriores son en modo texto, veremos ahora un ejemplo en modo binario:

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

struct tipoRegistro {
  char nombre[32];
  int edad;
  float altura;
};

int main() {
  tipoRegistro pepe;
  tipoRegistro pepe2;
  ofstream fsalida("prueba.dat",
    ios::out | ios::binary);

  strcpy(pepe.nombre, "Jose Luis");
  pepe.edad = 32;
  pepe.altura = 1.78;

  fsalida.write(reinterpret_cast<char *>(&pepe),
    sizeof(tipoRegistro));
  fsalida.close();

  ifstream fentrada("prueba.dat",
    ios::in | ios::binary);

  fentrada.read(reinterpret_cast<char *>(&pepe2),
    sizeof(tipoRegistro));
  cout << pepe2.nombre << endl;
  cout << pepe2.edad << endl;
  cout << pepe2.altura << endl;
  fentrada.close();

  return 0;
}
```

Al declarar streams de las clases ofstream o ifstream y abrirlos en modo binario, tenemos que añadir el valor ios::out e ios::in, respectivamente, al valor ios::binary. Esto es necesario porque los valores por defecto para el modo son ios::out e ios:in, también respectivamente, pero al añadir el flag ios::binary, el valor por defecto no se tiene en cuenta.

❖ FICHEROS DE ACCESO ALEATORIO:

Otra característica importante de los ficheros es la posibilidad de trabajar con ellos haciendo acceso aleatorio, es decir, poder hacer lecturas o escrituras en cualquier punto del fichero. Para eso disponemos de las funciones seekp y seekg, que permiten cambiar la posición del fichero en la que se hará la siguiente escritura o lectura. La 'p' es de put y la 'g' de get, es decir escritura y lectura, respectivamente.

Otro par de funciones relacionadas con el acceso aleatorio son tellp y tellg, que sirven para saber en qué posición del fichero nos encontramos.

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

int main() {
  int i;
  char mes[][20] = {"Enero", "Febrero", "Marzo",
    "Abril", "Mayo", "Junio", "Julio", "Agosto",
    "Septiembre", "Octubre", "Noviembre",
    "Diciembre"};
  char cad[20];

  ofstream fsalida("meses.dat",
    ios::out | ios::binary);

  // Crear fichero con los nombres de los meses:
  cout << "Crear archivo de nombres de meses:" << endl;
  for(i = 0; i < 12; i++)
    fsalida.write(mes[i], 20);
  fsalida.close();

  ifstream fentrada("meses.dat", ios::in | ios::binary);

  // Acceso secuencial:
  cout << "\nAcceso secuencial:" << endl;
  fentrada.read(cad, 20);
  do {
    cout << cad << endl;
    fentrada.read(cad, 20);
  } while(!fentrada.eof());

  fentrada.clear();
  // Acceso aleatorio:
  cout << "\nAcceso aleatorio:" << endl;
  for(i = 11; i >= 0; i--) {
    fentrada.seekg(20*i, ios::beg);
    fentrada.read(cad, 20);
    cout << cad << endl;
  }
```

```
  // Calcular el número de elementos
  // almacenados en un fichero:
  // ir al final del fichero
  fentrada.seekg(0, ios::end);
  // leer la posición actual
  pos = fentrada.tellg();
  // El número de registros es el tamaño en
  // bytes dividido entre el tamaño del registro:
  cout << "\nNúmero de registros: " << pos/20 << endl;
  fentrada.close();

  return 0;
}
```

La función seekg nos permite acceder a cualquier punto del fichero, no tiene por qué ser exactamente al principio de un registro, la resolución de las funciones seek es de un byte.

Cuando trabajemos con nuestros propios streams para nuestras clases, derivándolas de ifstream, ofstream o fstream, es posible que nos convenga sobrecargar las funciones seek y tell para que trabajen a nivel de registro, en lugar de hacerlo a nivel de byte.

La función seekp nos permite sobrescribir o modificar registros en un fichero de acceso aleatorio de salida. La función tellp es análoga a tellg, pero para ficheros de salida.

❖ FICHEROS DE ENTRADA Y SALIDA:

Para eso usaremos la clase fstream, que al ser derivada de ifstream y ofstream, dispone de todas las funciones necesarias para realizar cualquier operación de entrada o salida.

Hay que tener la precaución de usar la opción ios::trunc de modo que el fichero sea creado si no existe previamente.

Este programa crea un fichero con una palabra, a continuación, lee todo el fichero y cambia todos los caracteres 'a' por 'e'. Finalmente muestra el resultado. Básicamente muestra cómo trabajar con ficheros simultáneamente en entrada y salida.

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

int main() {
  char l;
  long i, lon;
  fstream fich("prueba.dat", ios::in |
    ios::out | ios::trunc | ios::binary);
```

```
  fich << "abracadabra" << flush;

  fich.seekg(0L, ios::end);
  lon = fich.tellg();
  for(i = 0L; i < lon; i++) {
    fich.seekg(i, ios::beg);
    fich.get(l);
    if(l == 'a') {
      fich.seekp(i, ios::beg);
      fich << 'e';
    }
  }
  cout << "Salida:" << endl;
  fich.seekg(0L, ios::beg);
  for(i = 0L; i < lon; i++) {
    fich.get(l);
    cout << l;
  }
  cout << endl;
  fich.close();

  return 0;
}
```

❖ ACTIVIDADES

- Consideraciones: Puede utilizar como IDE: Zinjal, Visual Studio 2019, entre otros. Ello debe ser especificado en su informe.

Nota: Para los siguientes ejercicios, programar de forma ordenada de tal manera que las clases y/o estructuras queden definidas en archivos cabecera (.h), los métodos u operaciones en un archivo .cpp y finalmente el programa principal en otro archivo .cpp. Asimismo, debe utilizar la experiencia de la práctica anterior y debe estar visible en el código.

- **EJERCICIOS PROPUESTOS:**

1. Crear el proyecto respectivo, explicar y documentar el archivo **Código Fuente - Ficheros** que está en Aula Virtual del curso. Realizar el trace respectivo.
2. Dadas las siguientes instrucciones:

```
struct Tdato
{
        int b;
        char s[100];
};
int x, n, a[10]={1,2,3,4,5,6,7,8,9,0};
```

```
double f;
char nombre[]="Ejercicios ficheros binarios";
Tdato p;
ifstream f1;
ofstream f2;
f1.open("entrada.dat", ios::binary);
f2.open("salida.dat", ios::binary);
```

Escribe las instrucciones para realizar cada una de las siguientes operaciones:
a. Escribir el dato entero x en el fichero.
b. Escribir el dato double f en el fichero.
c. Escribir los 5 primeros elementos del array a en el fichero.
d. Escribir los 10 primeros caracteres de la cadena nombre en el fichero.
e. Escribir el dato de tipo Tdato p en el fichero.
f. Leer un dato entero del fichero y almacenarlo en la variable x.
g. Leer un dato double del fichero y almacenarlo en la variable f.
h. Leer 5 enteros y almacenarlos en el array a.
i. Leer 10 caracteres y almacenarlos en la cadena nombre.
j. Leer un dato de tipo Tdato y almacenarlo en la variable p.

3. Crear un programa que pida al usuario que teclee frases, y las almacene en el fichero "frases.txt". Acabará cuando el usuario pulse Intro sin teclear nada. Después deberá mostrar el contenido del fichero.

4. Un programa que pregunte un nombre de fichero y muestre en pantalla el contenido de ese fichero, haciendo una pausa después de cada 25 líneas, para que dé tiempo a leerlo. Cuando el usuario pulse intro, se mostrarán las siguientes 25 líneas, y así hasta que termine el fichero.

5. Crear una agenda que maneje los siguientes datos: nombre, dirección, tlf móvil, email, y día, mes y año de nacimiento (estos tres últimos datos deberán ser números enteros cortos). Deberá tener capacidad para 100 fichas. Se deberá poder añadir un dato nuevo, visualizar los nombres de las fichas existentes, o mostrar todos los datos de una persona (se preguntará al usuario cual es el nombre de esa persona que quiere visualizar). Al empezar el programa, leerá los datos de un fichero llamado "agenda.dat" (si existe). Al terminar, guardará todos los datos en ese fichero.

❖ BIBLIOGRAFIA

- Deitel, Paul J., Deitel, Harvey M., "Cómo Programar en C++", 6ta Edición, Ed. Pearson Educación, México 2009.
- Stroustrup, Bjarne, "El Lenguaje de Programación C++", 3ra Edición, Adisson Pearson Educación S.A., Madrid 2002.
- Eckel, Bruce, "Thinking in C++", 2da Edición, Prentice Hall, 2000.