Tema 8 – Flujos de Entrada/Salida

a. Clases de entrada/salida

b. Procesamiento de ficheros

c. string Streams

Flujo = Stream

```
1   int a;
2   cout << a;
3   Jugador b;
4   cout << b;
```

```
1   ostream operator<<(ostream& os, Jugador& jugador)
```

- ¿Qué hace el operador << ?
- Si nos fijamos en una instrucción completa tendríamos **cout << “hola”;**
- Que se traduce a caracteres aislados: **cout << ‘h’ << ‘o’ << ‘l’ << ‘a’ ;**
- Hay un flujo de información desde el primer carácter hasta el último.
- **cout** es un objeto de la clase **ostream** que permite el flujo de datos “hacia afuera del programa”, por lo que fluyen los caracteres desde el programa, de alguna forma, hasta el monitor.
- **cout** y todos los otros objetos de flujo de salida siempre utilizan caracteres, por lo que un int a = 7, cuando se utilice el operador <<, se traduce a un string con el carácter 7.

>01 Formato de texto con manipuladores

- La librería <iomanip> permite manipular los flujos de entrada y salida.
- Por ejemplo, podríamos definir una cantidad mínima de caracteres a imprimir usando **setw(int)**. setw() funciona dentro de ostreams.

```
1  cout << "Con setw=30 : " << setw(30) << titulo << endl;
2  cout << "Con setw=10 : " << setw(10) << titulo << endl;
```

- También, podemos ajustar a la izquierda **left** o a la derecha **right**.

```
1  cout << "Justificado a la izquierda: " << left << setw(30) << titulo << endl;
2  cout << "Justificado a la derecha  : " << right << setw(30) << titulo << endl;
```

>02 Formato de texto con manipuladores

- Lo manipuladores de justificación (left, right) modifican todo el flujo, mientras que el de ancho (setw) modifica solo el siguiente dato a imprimir.

```
1  cout << right << setw(30) << titulo1 << setw(30) << titulo2 << " fin de linea" << endl;
2  cout << right << setw(30) << titulo1 << left << setw(15) << titulo2 << setw(30) << " fin de linea" <<endl;
```

- Por ende, debemos tener cuidado de donde utilizamos estos manipuladores.
- Por ejemplo, si queremos imprimir los 3 datos tipo string de Nombre, Apellido, DNI de una clase Cliente, podemos realizar:

```
1  cout << right << setw(12) << DNI << " "  << left << setw(20) <<  nombre << " " << setw(28) <<  apellido << endl;
```

>03 Flujo = Stream

- Para números podemos utilizar **setprecision(int)** para indicar la cantidad de dígitos que queremos ver de un número flotante o double.

```
1   cout << setprecision(5) << pi << endl;
```

- En el ejemplo, se busca imprimir en pantalla y total de 5 números: 3, 1, 4, 1 y 6.
- Si, además, utilizamos el manipulador **fixed** (**scientific**), permitimos que el formato de salida tenga un numero definido de decimales al usar setprecision(int)

```
1   cout << fixed;
2   cout << setprecision(5) << pi << endl;
3   cout << setprecision(9) << pi << endl;
```

https://www.h-schmidt.net/FloatConverter/IEEE754.html

Flujo de salida -> output stream -> ostream

- Los flujos de salida utilizan el operador << ya que el flujo se define desde el elemento hasta el stream.
- Por lo que las funciones amigas que utilizan ostream reciben, para un stream dado, un objeto de una clase cualquiera y devuelven siempre un stream.

```
1   ostream operator<<(ostream& os, Jugador& jugador)
```

- Entonces en lugar de crear un formato en la función main podemos crear un formato específico en la función amiga operator<<.
- Esto permite que cada clase tenga su propio formato definido.

>04

Flujo de salida -> output stream -> ostream

Tema8 - utils.h

```
1  ostream& operator<<(ostream& os, Estudiante& estudiante){
2      os << setw(12) << right << estudiante.DNI << " ";
3      os << setw(20) << left << estudiante.nombre << " ";
4      os << setprecision(1) << fixed << setw(10) << right << estudiante.puntaje_acumulado;
5      return os;
6  }
```

- Si la función amiga esta correctamente definida podemos usar el objeto ostream e ir creando un flujo según un formato requerido.
- En este ejemplo el formato requerido es:
  - Cantidad mínima de caracteres: 12 para DNI, 20 para nombre y 10 para puntaje_acumulado
  - Justificación: DNI y puntaje_acumulado a la derecha, nombre a la izquierda.
  - Formato numérico: precisión de 1 dígito y nomenclatura fija para puntaje_acumulado.

>05 Input stream

- Así como << es para crear un flujo de salida, se utiliza >> para un flujo de entrada.
- Podríamos pedir al usuario que ingrese de forma manual los datos de inicialización de un objeto cualquiera, por ejemplo, de la clase Estudiante.
- En este ejemplo pedimos los valores de los atributos uno a uno.
- Esta forma es bastante informativa y fácil de seguir para el usuario.
- Pero también podemos crear otras funciones que sean más complejas pero más eficientes para los usuarios avanzados o acostumbrados a los objetos.

Tema8 - Clase8.cpp

```
1  Estudiante est;
2  long int dni;
3  string nombre;
4  float puntaje;
5
6  cout << "Ingrese el nombre del estudiante: ";
7  cin >> nombre;
8  cout << "Ingrese el DNI del estudiante: ";
9  cin >> dni;
10 cout << "Ingrese el puntaje del estudiante: ";
11 cin >> puntaje;
12
13 est.setNombre(nombre);
14 est.setDNI(dni);
15 est.setPuntajeAcumulado(puntaje);
```

>06 Input stream

- Como sabemos, podemos recibir múltiples parámetros de entrada separados por un espacio si utilizamos varias variables en un mismo cin.

Tema8 - Clase8.cpp

```cpp
long int dni;
string nombre;
float puntaje;

cout << "Ingrese los datos del estudiante: ";
cout << "DNI Nombre Puntaje" << endl;
cin >> dni >> nombre >> puntaje;

Estudiante est = Estudiante(nombre, dni, puntaje);
```

- Con esto disminuimos drásticamente la cantidad de líneas de código y también hacemos que la ejecución sea más eficiente.

>07 Input stream

- Con esta forma, el flujo de entrada espera exactamente 3 grupos de caracteres separados por un espacio.
- Pero existen casos en que uno de los datos de entrada es un string y los strings pueden tener 1 o más caracteres de espacio: “Maria del Mar” lleva 2 espacios, por lo que este nombre compuesto no podrá ser leído con nuestro código.
- Para solucionar este problema la solución consiste en utilizar la función **quoted( )**, que es un modificador parte de iomanip y lo que hace es esperar que el dato quede delimitado no por espacios sino por doble comillas.
- Para utilizar:

Tema8 - Clase8.cpp

```
1  cin >> dni >> quoted(nombre) >> puntaje;
```

- Siempre hay que usar:

```
123456 "Nombre1 apellido1" 9.2
```

>08 Input stream

- En lugar de crear este flujo de entrada exclusivamente en el programa principal para el flujo cin, también es posible crear la función amiga que se encarge de rellenar datos de un objeto de entrada.
- La forma de la función es análoga a la definición de un ostream:

Tema8 - Estudiante.h

```
1  friend istream& operator>>(istream& is, Estudiante& estudiante);
```

- Por lo que su uso es igual al de un cin, pero los datos de destino pueden ser directamente los atributos de la clase:

Tema8 - utils.h

```
1  istream& operator>>(istream& is, Estudiante& estudiante){
2      is >> estudiante.DNI >> quoted(estudiante.nombre)>> estudiante.puntaje_acumulado;
3      return is;
4  }
```

>09 Input stream

- Es posible combinar múltiples datos de entrada como diferentes objetos de diferentes clases.
- De hecho, es una práctica común, ya que es posible realizar:

Tema8 - Clase8.cpp

```
1  float f;
2  int i;
3  cin >> f >> i;
```

- También es posible realizar:

Tema8 - Clase8.cpp

```
1  cout << "Ingrese un estudiante, un string entre comillas y otro estudiante: ";
2  Estudiante est3;
3  string s;
4  Estudiante est4;
5  cin >> est3 >> quoted(s) >> est4;
```

Streams

- Cree una lista de tamaño indefinido de objetos de la clase Complejo a partir de cin. A partir de la lista, ordénelos según el módulo e imprima en pantalla los elementos del contenedor.

Complejo
- real: double - imaginario: double
+ Complejo() + Complejo(real: double, imaginario: double) + getters + setters + operator==(const Complejo& otro) const: bool + operator<(const Complejo& otro) const: bool + operator<<(ostream & os, Complejo& complejo): friend ostream & + operator>>(istream & is, Complejo& complejo): friend istream &

File stream

- El stream de datos no se limita solamente a flujos de datos entre un programa y la línea de comandos.
- También es posible crear un flujo de datos desde un programa a otro o desde un programa a un fichero.
- Entonces, es posible crear ficheros de cualquier tipo, por ejemplo, .txt desde un programa en C++.
- Para poder crear o editar un fichero habrá que utilizar varias librerías: string, iostream, **fstream** e iomanip.

File stream

- La forma básica de escribir en un archivo es la siguiente:

1. Abrir un fichero en donde queramos escribir información.
2. Asegurarnos de que el fichero está disponible.
3. Crear el flujo de caracteres saliente (del programa al fichero) usando el operador <<.
4. Cerrar el fichero.

- Cuando abrimos una ventana de comandos cmd o Terminal, C++ está ejecutando los pasos 1 y 2 de forma interna para crear el cout y luego elimina la referencia a la terminal cuando el programa termina.

File stream

1. Abrir un fichero en donde queramos escribir información: Significa crear un objeto de la clase **ofstream**
   Estas clases definen constructores con 2 atributos de entrada **ruta: string** y **tipo** de operación.
   Si la ruta es local, el fichero debe estar en la misma carpeta que el main.
   El tipo de operación debe ser de escritura **ios::out.**
2. Asegurarnos de que el fichero está disponible:
   Una vez creado el objeto, habrá que verificar si el fichero se puede utilizar mediante el `operator bool()`: Aunque sea difícil de comprender, se puede verificar si objeto es true/false mediante **if(objeto)**
3. Crear un flujo de caracteres saliente (del programa al fichero) usando el operador <<: En lugar de usar cout <<, habrá que usar **objeto <<**/
4. Cerrar el fichero:
   objeto.close() cierra el objeto.

File stream

- El ejemplo más sencillo es escribir un string cualquiera dentro de un fichero:

Tema8 - Clase8.cpp

```
1  ofstream fichero = ofstream("datos.txt", ios::out); // 1
2  if (fichero){ // 2
3      string dato = "Hola mundo";
4      fichero << dato << endl; // 3
5      fichero.close(); // 4
6  }
```

- En este ejemplo, todos los datos son parte de la memoria de apilamiento, pero normalmente era posible utilizar cout en cualquier parte del programa.
- Por ello, es preferible que los ficheros (como objetos) se generen siempre dentro de la memoria estática, es decir fuera del main.

File stream

- Por supuesto, si tenemos múltiples datos que queremos exportar, podemos encolumnar los datos para exportar de forma ordenada.
- Así podemos exportar los datos de los objetos de la clase Estudiante almacenados en un vector a un fichero utilizando:

Tema8 - Clase8.cpp

```
1  for(int i = 0; i < estudiantes.size(); i++){
2      datos_estudiantes << *it << endl;
3      it++;
4  }
```

- Con esto el ostream recibido en el operator<<() no es el cout sino que el fichero almacenado como objeto en la variable datos_estudiantes, permitiendo el uso de la función amiga para dar formato de texto a los atributos de una clase.

File stream

- Pero habrá veces en que los strings que generemos sean diferentes de acuerdo con los flujos en los que enviamos información: Cómo haríamos si necesitamos enviar solamente los datos de DNI y Puntuación al cout pero los datos de Nombre y DNI al fichero de salida?
- Como los objetos de cout y datos_estudiantes se generaron en la memoria estática, es posible verificar cual es la fuente del flujo y confirmar si una dirección es igual a otra:

Tema8 - Clase8.cpp

```cpp
ostream& operator<<(ostream& os, Estudiante& estudiante){
    if (&os == &cout){
        os << setw(12) << right << estudiante.DNI << " ";
        os << setprecision(1) << fixed << setw(10) << right << estudiante.puntaje_acumulado;
    }
    else if (&os == &datos_estudiantes){
        os << setw(20) << left << estudiante.nombre << " ";
        os << setw(12) << right << estudiante.DNI << " ";
    }
    return os;
}
```

File stream

- Si existen diferentes formatos definidos para diferentes ficheros de salida, habrá que comprobar si la dirección del flujo se corresponde con el fichero esperado.

Tema8 - Clase8.cpp

```
1  else if (&os == &datos_estudiantes)
```

Tema8 - Clase8.cpp

```
1  else if (&os == &dni_puntaje)
```

Tema8 - Clase8.cpp

```
1  else if (&os == &solo_nombres)
```

- También, por supuesto, se puede crear un flujo por defecto.
- La importancia de verificar la existencia del fichero y el posterior cierre de este se debe hacer siempre dentro de la función principal y no dentro de las funciones amigas, ya que eso implicaría abrir y cerrar el fichero tantas veces como datos existan.

File stream

- También, por supuesto, es posible obtener datos o ficheros que ya existen en el ordenador con dirección al programa >>
- La funcionalidad de los streams de ficheros es inequívocamente la más importante en el mundo de la informática.
- La comunicación entre dos ordenadores diferentes siempre se realizar a través de la lectura/escritura de un fichero.
- Por ejemplo, un fichero “Temas 8.pdf” puede ser creado en el programa Powerpoint y el profesor, que rellena el fichero mediante el flujo >>, y lo sube al Moodle (también mediante flujos).
- Luego, el estudiante descarga el fichero (transferencia de ficheros) y lo abre mediante una lectura de flujo de entrada en un programa como por ejemplo Acrobat Reader mediante el flujo <<.

File stream

- Para leer de un fichero debemos repetir los pasos de apertura del fichero, pero con la salvedad de que, en este caso, el fichero **ifstream** se abre en modo lectura **ios::in**.

```
1   ifstream fichero = ifstream("datos.txt", ios::in); // 1
```

- A continuación, existen varias formas de leer los datos, pero podemos concentrarnos en leer dato por dato, grupos de caracteres que están separados por espacios.
- Por lo que la implementación de lectura no varía excepto en el objeto que realiza el flujo de entrada, en lugar de: cin >> datos;

```
1   fichero >> dato; //3
```

>13 File stream

- Por supuesto que, si queremos leer un string con varios espacios en blanco, en primer lugar, el fichero debería almacenar entre comillas al string compuesto.
- Para ello, también podemos utilizar **quoted( )** durante la escritura, siempre que al leer el mismo dato recordemos de utilizar **quoted( )**.

```
1  fichero << quoted(dato) << endl;
```

```
1  fichero2 >> quoted(dato);
```

- En definitiva, la mejor práctica siempre consiste en envolver a los strings con quoted para evitar ambigüedades.

>14 File stream

- Adicionalmente, obtener elementos que corresponden a los atributos de una clase a partir de un fichero de entrada será posible siempre que:
  - Exista la definición utilizando istream del operador>> amiga de la clase.
  - El fichero de entrada tenga datos ordenados de la misma manera en que se esperan en la función amiga.

```
1  list<Estudiante> estudiantes;
2  Estudiante est;
3  while(datos_de_lectura >> est){
4      estudiantes.push_back(est);
5  }
```

File stream

- Extraordinaria 2023