

Programación Orientada a Objetos

Unidad 5: Flujos de I/O - texto

Parte 1

Streams

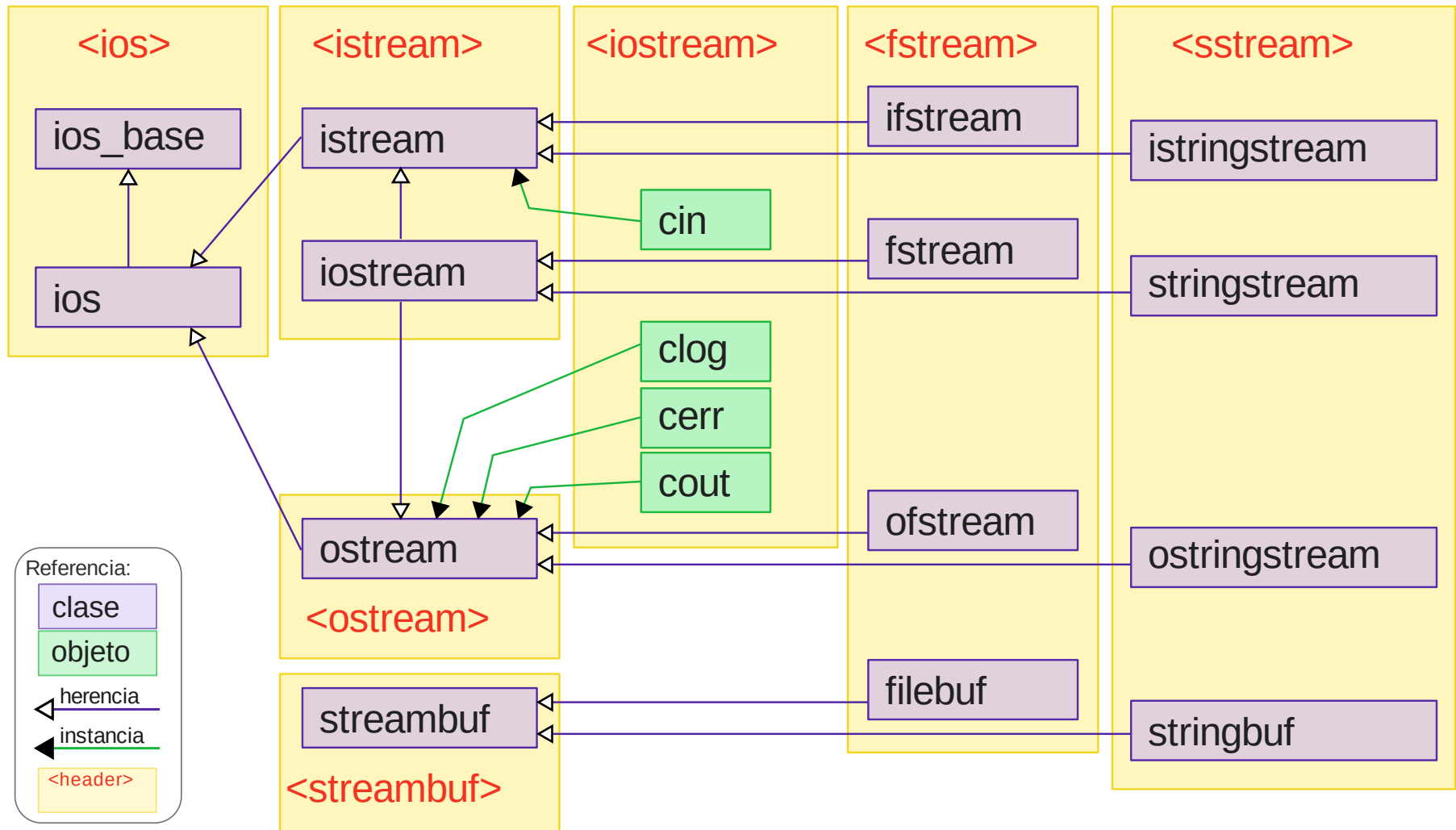
¿QUÉ SON LOS STREAMS?

El stream (flujo) es el tipo de dato básico para realizar operaciones de entrada/salida en C++.

Algunas instancias de streams ya conocidas son:

- `cin`: entrada de texto desde la consola (por teclado)
- `cout`: salida de texto hacia la consola (en pantalla)

STREAMS EN C++



STREAMS EN C++

Hay tres usos básicos (std) para streams en C++:

- Leer del teclado/escribir en pantalla:
 - instancias `cin/cout/cerr/clog`
- Leer/escribir en strings:
 - clase `stringstream`
- Leer escribir en **archivos**:
 - clases `fstream/ifstream/ofstream`

¿QUÉ ES UN ARCHIVO?

- En informática (según Wikipedia):
 - conjunto de bits almacenado en un dispositivo.
- Se almacena y se lee por bytes (8 bits).
- Se identifican con un nombre y una "ruta".

C:\Archivos de Programa\Zinjai\zinjai.exe
ubicación nombre

- Además, se le pueden otorgar ciertos "permisos" o "atributos" (escritura/lectura/ejecución/oculto,etc) según el sistema de archivos.

ARCHIVOS DE TEXTO VS. ARCHIVOS BINARIOS

- Las categorías "**de texto**" y "**binario**" hacen referencia a la forma de "codificar" la información que se guarda:
 - **texto**: cada byte representa un carácter según el código ASCII.
 - **binario**: los datos se guardan exactamente como se guardan en memoria.
- ! Para el sistema de archivos del sistema operativo, **no hay diferencia** entre un tipo de archivo y el otro.

ARCHIVOS DE TEXTO VS. ARCHIVOS BINARIOS

Ejemplo: Los enteros 4 y 1172:

- Archivo de texto:

52	13	49	49	55	50	13
'4'	'\n'	'1'	'1'	'7'	'2'	'\n'

! no suele ser posible modificar solo un dato

- Archivo binario:

4	0	0	0	148	4	0	0
---	---	---	---	-----	---	---	---

$$4 \times 256^0 + 0 \times 256^1 + 0 \times 256^2 + 0 \times 256^3 \quad 148 \times 256^0 + 4 \times 256^1 + 0 \times 256^2 + 0 \times 256^3$$

✓ se suelen organizar igual que un arreglo

ARCHIVOS DE TEXTO

- ▶ Se pueden leer/modificar fácilmente:
 - ▶ con cualquier editor de texto (ej: notepad),
 - ▶ desde cualquier sistema operativo/plataforma.
- ▶ Datos de igual tipo no tiene el mismo tamaño.
 - ▶ **Su acceso es secuencial.**
- ▶ Los datos que no son cadenas de texto, se convierten al escribir/leer.

ARCHIVOS BINARIOS

- Los datos se guardan directamente sin conversión.
- No se pueden leer ni modificar fácilmente.
 - Usualmente solo el programa que los creó puede operarlos correctamente.
 - Su verdadera codificación puede variar de una plataforma a otra.
- Datos de igual tipo (en general) ocupan la misma cantidad de bytes.
 - **Su acceso es aleatorio.**

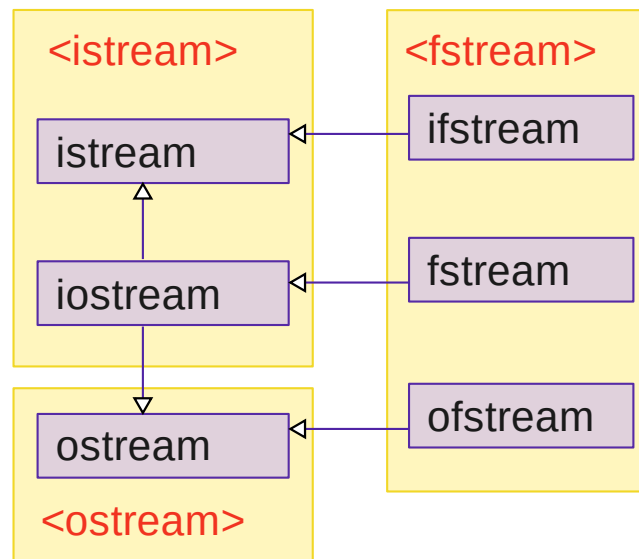
TEXTO VS BINARIOS: VENTAJAS Y DESVENTAJAS

- Los archivos de texto se pueden "*pasar*" de un programa a otro sin problemas.
- Operar con archivos binarios es más rápido porque no implica conversión y
- permiten acceso aleatorio:
 - se puede leer/modificar solo una parte del archivo.

ARCHIVOS EN C++

Para acceder a archivos se utilizan las clases:

- `ifstream`: para lectura
- `ofstream`: para escritura
- `fstream`: para lectura y/o escritura



ARCHIVOS EN C++

Antes de utilizar un archivo, se debe "abrir":
asociar el **stream** con una **ruta del sist. de arch.**

- Mediante el constructor de ifstream:

```
ifstream archi("notas.txt");
```

¿Dónde se encuentra "notas.txt"?

- Mediante el método open:

```
ifstream archi;  
archi.open("notas.txt");
```

⊗ Nunca utilizar rutas fijas y absolutas
como "c:\\miprograma\\misdatos.txt"

ARCHIVOS EN C++

Constructor y open aceptan una combinación de banderas como segundo argumento opcional:

Útiles para ofstreams en modo texto:

- `ios::trunc` **elimina** todo el contenido previo del archivo
- `ios::app` se escribirá siempre **al final**

ARCHIVOS EN C++

- ¿Cómo verificar si se abrió correctamente?

```
ifstream archi("notas.txt");  
if (!archi.is_open()) {  
    cerr << "No se pudo abrir el archivo";  
    return 1;  
}
```

❓ ¿Por qué podría fallar?

- Luego de utilizarlo, se lo debe cerrar:

```
archi.close();
```

✅ Si nos "olvidamos", lo hará el destructor de `fstream`

LECTURA/ESCRITURA CON ARCHIVOS DE TEXTO

Se utilizan las mismas funciones, y los mismos operadores que para leer/escribir en consola:

- Se escribe con <<, se pueden usar manipuladores:

```
ofstream archi("notas.txt");  
archi << nombre << endl;  
archi << set_precision(2) << prom << endl;
```

- Se lee con >> y/o getline:

```
ifstream archi("notas.txt");  
getline(archi, nombre);  
archi >> nota;
```

! recordar el problema del "ignore"
al combinar >> con getline

EJEMPLO

1. Escriba un programa que permita ingresar 10 enteros y los guarde en un archivo de texto.
2. Escriba otro programa que permita leer y mostrar en pantalla la información guardada por el programa anterior.

❓ ¿Qué pasaría si en lugar de 10 fueran N ?

¿HASTA CUANDO LEER DESDE UN ARCHIVO?

- **utilizar la mismísima acción de lectura como condición** de control para la iteración

```
int n=0;
float x, suma=0;
ifstream archi("numeros.txt");
while ( archi>>x ) {
    suma+=x;
    n++;
}
cout << "Promedio: " << suma/n;
```

- ✓ Aplica tanto para >> como para getline
 - ✓ equivale a preguntar si la lectura se pudo realizar correctamente

EJEMPLO

Escriba un programa para leer una lista de nombres y notas de 3 parciales de un curso de POO de un archivo de texto como el siguiente:

```
Carmack, Juan
10 9 10
Gates, Guillermo
6 5 7
Stallman, Ricardo
7 10 8
...
```

y reemplace las notas por el promedio.

¿CÓMO OPERAR SOBRE LOS DATOS DE UN ARCHIVO?

- Opción 1: **cargarlo en memoria** al comienzo, operar en memoria, guardarlo completo antes de finalizar
 - Texto o datos de longitud variable
 - Bases de datos pequeñas
 - Archivos sin formato
- Opción 2: operar **directamente sobre el archivo**
 - Bases de datos grandes (ej: históricos, logs)
 - Si hay que modificar, solo con archivos binarios
 - No se puede borrar físicamente

```
// estructuras de datos para guardar todo el contenido inicial
struct Alumno { string nom; int n1,n2,n3; };
vector<Alumno> v;
// Paso 1: cargar todo el archivo
ifstream fin("notas.txt"); Alumno aux;
while ( getline(fin,aux.nom) &&
        fin>>aux.n1>>aux.n2>>aux.n3 )
{
    v.push_back(aux);
    fin.ignore();
}
fin.close();
// Paso 2: reescribirlo completamente desde cero
ofstream fout("notas.txt");
for(size_t i=0;i<v.size();i++)
    fout << v[i].nom << endl
        << fixed << setprecision(2)
        << (v[i].n1+v[i].n2+v[i].n3)/3.0
        << endl;
fout.close();
```

Parte 2

Strings

LECTURA Y ESCRITURA

```
string palabra;  
cout << "Ingrese una palabra: ";  
cin >> palabra;
```

```
string frase;  
cout << "Ingrese una frase: ";  
getline(cin, frase);
```

```
cout << palabra << endl;  
cout << frase << endl;
```

OPERADORES SOBRECARGADOS

- El **operator=** copia/asigna:

```
string s1, s2;  
s1 = "Estudiar"; s2 = "suerte";
```

- El **operator+** concatena:

```
string s3 = s1+" trae "+s2;  
cout << s3; // muestra "Estudiar trae suerte"
```

- Los **operadores ==, <, <=, >, >= y !=** comapanan lexicográficamente (según el código ASCII)

"EXTRACCIÓN" DE SUB-CADENAS

```
string s1 = "Al infinito y más allá!";  
  
string s2 = s1.substr(3, 8);  
cout << s2; // muestra "infinito"  
  
string s3 = s1.substr(14);  
cout << s3; // muestra "más allá!"  
  
cout << s1; // muestra "Al infinito y más allá!"
```

- ✓ Siempre que un método reciba un rango, será dado como: posición inicial y longitud
- ✓ Si solo se indica posición inicial, va hasta el final de la cadena

MODIFICACIÓN (I)

- Borrar una parte intermedia

```
string s1 = "Es menester que sea rock!";  
s1.erase(3,17);  
cout << s1; // muestra "Es rock!"
```

- Borrar desde un punto hasta el final

```
string s2="Vengo remandode larga distancia";  
s2.erase(13);  
cout << s2; // muestra "Vengo remando"
```

- Borrar todo

```
string s3 = "Andarás bien por la 66"  
s3.clear();  
cout << s3; // muestra ""
```

MODIFICACIÓN (II)

- Reemplazar un fragmento por otro

```
string s1 = "Hola a todos, yo soy el león!";  
s1.replace(5, 7, "mundo");  
cout << s1; // muestra "Hola mundo, yo soy el león!"
```

- Insertar un string en medio de otro

```
string s2 = "Maderas de nogal";  
s2.insert(11, "viejo ");  
cout << s2; // muestra "Maderas de viejo nogal"
```

MANIPULACIÓN DE CARACTERES

- Se puede operar como si fuera un `vector<char>`

```
string corregir(string str) {  
    if (!str.empty()) {  
        str[0] = toupper(str[0]);  
        for (size_t i=1; i<str.size(); i++)  
            str[i] = tolower(str[i]);  
    }  
    return str;  
}  
  
int main() {  
    string s1 = "sIGue AL COnEjo BlaNCo.";  
    string s2 = corregir(s1);  
    cout << s2; // muestra "Sigue al conejo blanco."  
}
```

BÚSQUEDA (I)

```
string frase = "Ser el elegido es como estar  
enamorado";  
  
string palabra;  
cout << "Ingrese una palabra: ";  
cin >> palabra;  
  
size_t p = frase.find(palabra);  
if (p==string::npos)  
    cout << "La palabra no esta en la frase";  
else  
    cout << "La palabra comienza en la  
posición " << p;
```

❗ **Si no se encuentra**, find retorna el valor especial ***string::npos***

BÚSQUEDA (II)

```
string frase = "Ser el elegido es como estar  
enamorado";  
  
string silaba;  
cout << "Ingrese una silaba: ";  
cin >> silaba;  
  
int cant = 0;  
size_t p = frase.find(silaba, 0);  
while (p!=string::npos) {  
    cant++;  
    p = frase.find(silaba, p+silaba.size());  
}  
cout << "La silaba está " << cant  
    << " veces" << endl;
```

✓ El 2do argumento de *find* indica desde dónde comenzar la búsqueda

MÁS FUNCIONES Y MÉTODOS

En una referencia de la biblioteca estándar
buscar como:

- ▶ `std::basic_string`

⚠ Es el verdadero nombre de la clase desde la cual se "especializa" `string`. El concepto de "especialización" se verá en la unidad 7.