# Programación Orientada a Objetos

## Unidad 5: Flujos de I/O - texto

Pablo Novara

2024

UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

FICH UNL

# Parte 1
# Streams

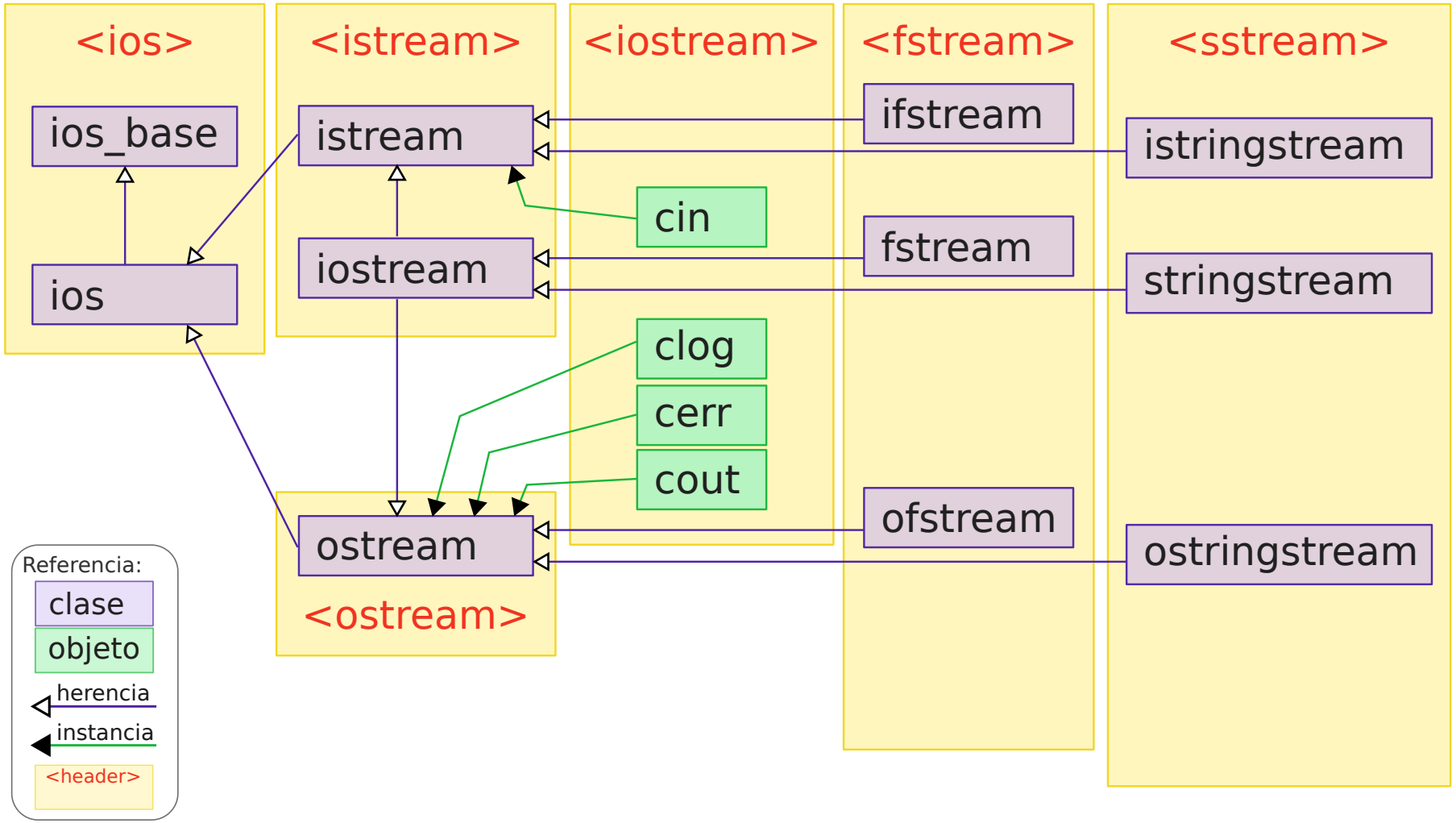
# ¿QUÉ SON LOS STREAMS?

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 terminal (por teclado)
- `cout`: salida de texto hacia la terminal (en pantalla)

# STREAMS EN C++


<ios>
ios_base
ios
<istream>
istream
iostream
<iostream>
cin
clog
cerr
cout
<ostream>
ostream
<fstream>
ifstream
fstream
ofstream
<sstream>
istringstream
stringstream
ostringstream
Referencia:
clase
objeto
herencia
instancia
<header>

# STREAMS EN C++

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

- Leer del teclado/escribir en la terminal:
  - 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

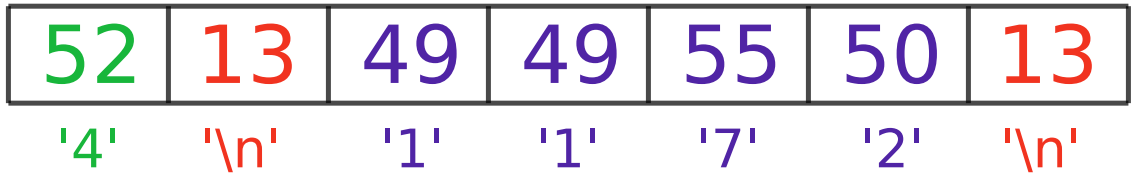
- Las categorías "de texto" y "binario" hacen referencia a la forma de "codificar" la información que se guarda:
  - **texto**: cada byte representa un carácter según el código ASCII (o algo similar como UTF8)
  - **binario**: los datos se guardan exactamente como se guardan en memoria.

*Para el sistema de archivos del sistema operativo **no hay diferencia** entre un tipo de archivo y el otro (ambos son solo un conjunto de bytes).*

# ARCHIVOS DE TEXTO VS. ARCHIVOS BINARIOS

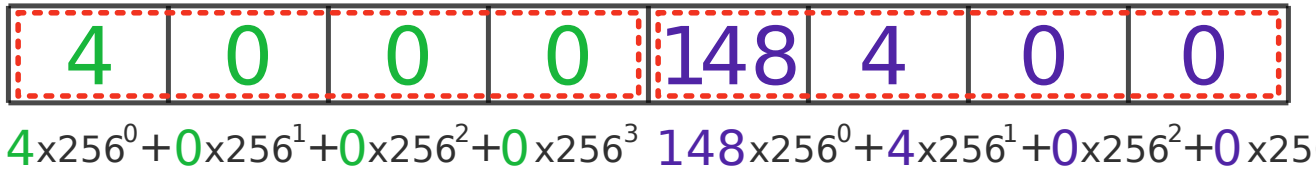
Ejemplo: Los enteros 4 y 1172:

- Archivo de texto:




*no suele ser posible modificar solo un dato*

- Archivo binario:




*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 tienen siempre el mismo tamaño.
  - **Su acceso es secuencial**.

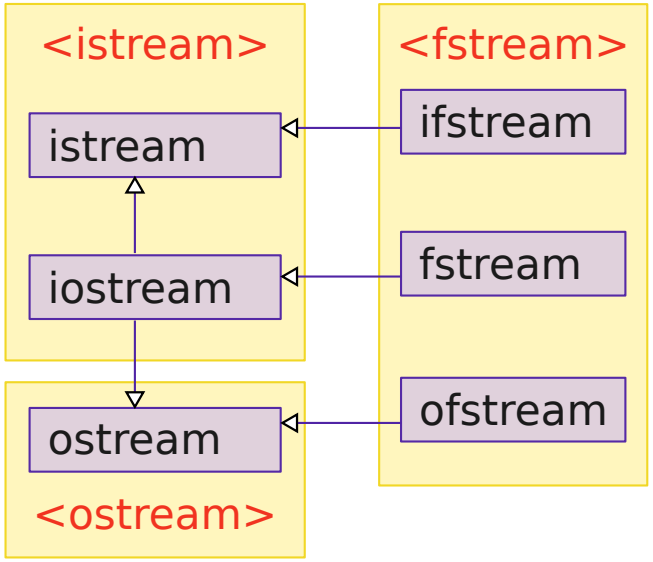
- Los datos que no son cadenas de texto, se convierten al escribir/leer.

# ARCHIVOS EN C++

Para acceder a archivos se utilizan las clases:

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

*se suele usar solo para binarios*



# 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 fstream:

```
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

  *`trunc` es la bandera por defecto para `ofstream`*

- `ios::app` se escribirá agregando siempre **al final**

# ARCHIVOS EN C++

- ¿Cómo verificar si se abrió correctamente?

```
ifstream archi("notas.txt");
if (not archi.is_open())
    throw runtime_error("No se pudo abrir notas.txt");
```

*¿Por qué podría fallar?*

- Nota: la fórmula `throw runtime_error("blah")` finalizará el programa.

*No tendremos tiempo de ver manejo de errores y excepciones, usar como receta "mágica" para errores irrecuperables.*

- 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 la terminal:

- Se escribe con <<:

```
ofstream archi("notas.txt");
if (not archi.is_open()) ...

for (size_t i=0; i<v.size(); ++i) {
    archi << v[i].nombre << endl; // string
    archi << fixed << set_precision(2) << v[i].promedio << endl; // float
}
```

*Se pueden usar manipuladores de flujo.*

- Se debe escribir "algo" (ej:`endl`) para separar entre dato y dato.

# LECTURA/ESCRITURA CON ARCHIVOS DE TEXTO

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

- Se lee con **>>** y/o **getline**:

```
ifstream archi("notas.txt");
if (not archi.is_open()) ...

for (size_t i=0; i<v.size(); ++i) {
    getline(archi, nombre);
    archi >> nota;
    archi.ignore();
}
```

*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

```
ifstream archi("numeros.txt");
if (not archi.is_open()) ...
int cant = 0;
float aux, suma = 0;
while ( archi>>aux ) {
    suma += aux;
    ++cant;
}
cout << "Promedio: " << suma/cant;
```

- *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.

```cpp
// 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");
if (not fin.is_open()) throw runtime_error("No se pudo leer 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", ios:trunc);
if (not fin.is_open()) throw runtime_error("No se pudo escribir notas.txt.");
fout << fixed << setprecision(2);
for (size_t i=0; i<v.size(); ++i) {
    fout << v[i].nom << endl;
    fout << (v[i].n1+v[i].n2+v[i].n3)/3.0 << endl;
}
```

*importante: cerrar antes de volver a abrir para escritura*

*sería mejor hacer cada paso en una función diferente*

# 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 !=** comparan 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 remando de 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*