Programación Orientada a Objetos

Unidad 6: Archivos binarios

FICH UNL
Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas

2019
Pablo Novara

¿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

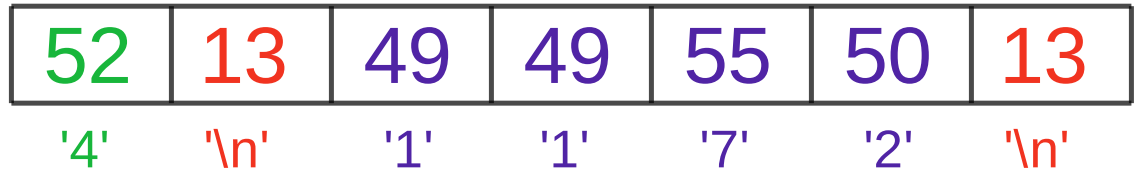
- Las categorías "de texto" y "binario" hacen referencia a la forma de "codificar" la información que se guarda:
  - **texto**: cada byte representa un carácter según el código ASCII.
  - **binario**: los datos se guardan exactamente como se guardan en memoria.

Para el sistema de archivos de sistema operativo, ***no hay diferencia*** *entre un tipo de archivo y el otro.*

ARCHIVOS DE TEXTO VS. ARCHIVOS BINARIOS

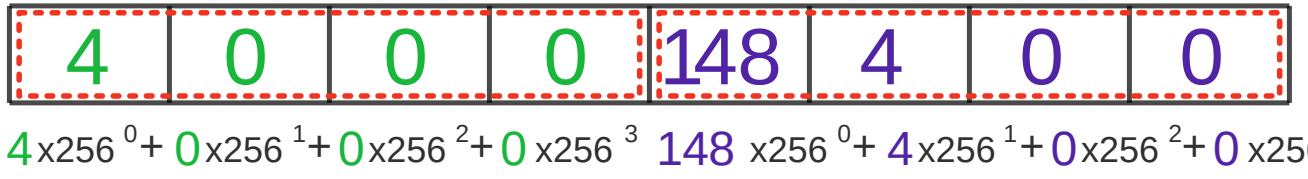
Ejemplo: Los enteros 4 y 1172:

- Archivo de texto:



no suele ser posible modificar solo un dato

- Archivo binario:

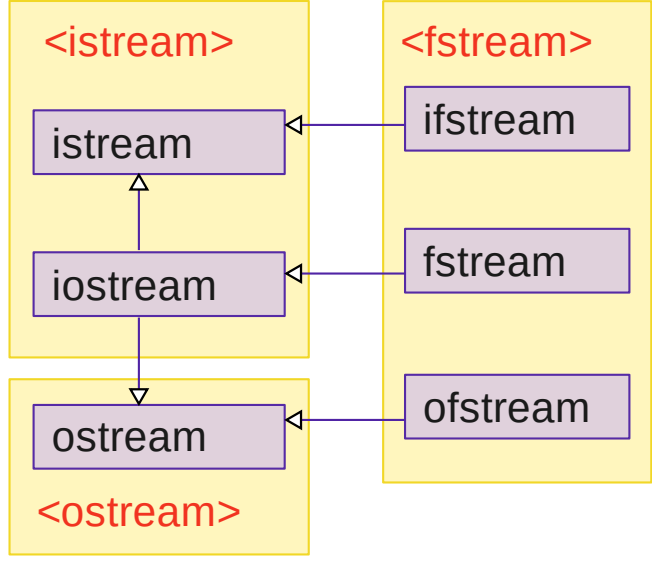


se suelen organizar igual que un arreglo

ARCHIVOS EN C++

Para acceder a archivos se utilizan las clases:

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



buscar en la referencia C++ como `basic_fstream`

ARCHIVOS BINARIOS EN C++

Para trabajar en modo binario, se debe añadir la bandera `ios::binary` al modo de apertura:

```
ofstream archi("datos.dat", ios::binary);
```

Se lee/escribe con los métodos `read` y `write`:

```
archi.write(reinterpret_cast<char*>(&x),
            sizeof(x));
```

Los argumentos son:

- un puntero de tipo char que apunta al dato que se quiere leer/escribir.
- la cantidad de bytes que se leen/escriben.

ARCHIVOS EN C++

Posibles banderas para el 2do arg del ctor:

`ios::app`	se escribe **siempre al final**
`ios::trunc`	se debe **eliminar todo** el contenido del archivo
`ios::binary`	se accederá en **modo binario**
`ios::in`	se abre para lectura, **no eliminar el contenido** previo
`ios::out`	se abre para **escritura**
`ios::ate`	el "cursor" **se ubica al final** al abrir el archivo

- *para un archivo de salida, `ios::in` es lo contrario a `ios::trunc`*

EJEMPLOS

1. Escriba un programa que permita ingresar pares de datos compuestos por un double y un int y los guarde en un archivo cuyo nombre ingresa el usuario.

¿Cada par es un struct o dos variables seperadas?

2. Escriba un programa que solicite el nombre de un archivo generado con el programa del ejemplo 2 y muestre su contenido.

¿Cómo saber cuantos pares hay en el archivo?

No siempre es lo mismo dos variables separadas que las mismas dos como un solo struct

¿CÓMO DETERMINAR HASTA CUANDO LEER?

```
struct Par { int i; double d; };
...
// empezar posicionandose al final con ios::ate
ifstream archi("datos.dat",
                ios::binary|ios::ate);
// obtener tamaño (en bytes) del archivo
int tam_bytes = archi.tellg();
// calcular cantidad de datos guardados
int cant_ints = tam_bytes/sizeof(Par);
// volver al comienzo para comenzar a leer
archi.seekg(0);
```

✅ *recomendable para archivos binarios*

❌ *no válido para archivos de texto*

ACCESO ALEATORIO EN ARCHIVOS BINARIOS

- Si no se especifica otra cosa, los datos se leen o escriben uno a continuación de otro.

- Hay métodos para modificar o consultar la posición donde se leerá o escribirá la próxima vez:
  - **`tellg()/tellp()`**: consulta la posición
  - **`seekg(...)/seekp(...)`**: cambia la posición

> *No se debe asumir que los punteros de lectura y escritura son el mismo, ni que son independientes*

EJEMPLOS

3. Escriba un programa para abrir el archivo generado en el primer ejemplo, buscar el par de datos cuyo double sea mayor, informar su posición, y poner su entero en 0.

4. Escriba una clase para gestionar la tabla de mejores puntuaciones de un juego. La clase debe permitir almacenar y recuperar mediante un archivo binario los 10 mejores puntajes y los nombres de quienes los obtuvieron.

¿CÓMO OPERAR SOBRE LOS DATOS DE UN ARCHIVO?

1. Operar directamente sobre el archivo:
   - Si solo hay que modificar posiciones particulares, solo con archivos binarios
   - Bases de datos grandes (ej: históricos, logs)

   ⊗ *no se puede borrar*

2. Cargarlo todo en memoria, operar en memoria, guardarlo completo al finalizar:
   - Bases de datos pequeñas
   - Archivos sin formato
   - Texto o datos de longitud variable

BINARIOS Y OBJETOS DINÁMICOS

```
void foo() {
    string s1 = "Hola Mundo!", s2;
    ofstream a1("archi.dat", ios::binary);
    a1.write(reinterpret_cast<char*>(&s1),
             sizeof(s1));
    a1.close();

    ifstream a2("archi.dat", ios::binary);
    a2.read(reinterpret_cast<char*>(&s2),
            sizeof(s2));
    cout << s2; // muestra "Hola Mundo!"
} // KBOOM!!!
```

Aunque parezca que "funciona"
está horriblemente mal

BINARIOS Y OBJETOS DINÁMICOS

```
int main() {
    string str = "Hola Mundo!";
    ofstream arch("archi.dat", ios::binary);
    arch.write(reinterpret_cast<char*>(&str),
                sizeof(str));
}
```

```
int main() {
    string str;
    ifstream arch("archi.dat",ios::binary);
    arch.read(reinterpret_cast<char*>(&str),
                sizeof(str));
    cout << s2; //KBOOM!!!
}
```

BINARIOS Y OBJETOS DINÁMICOS

No es correcto guardar/leer (con `read`/`write`) instancias de clases que utilizan punteros en un archivo binario.

⊗ *Se guardan los punteros que hay dentro del string, pero no los datos apuntados*

Ejemplos de clases que usan punteros:

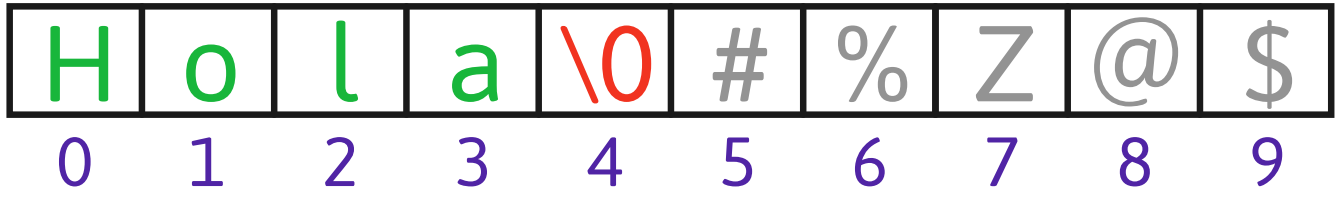
`vector`, `string`, `list`, …

`std::string` VS CSTRIGNS

Un cstring (string estilo C) es:

- un arreglo de caracteres (`char[N]` o `char*`) que contiene una cadena
- con un caracter especial al final para indicar dónde termina: `'\0'`

```
char str[10] = "Hola";
```



`std::string` VS CSTRIGNS

Pasar de `std::string` a `char*`:

```
strcpy(c, s.c_str());
```

- **`strcpy`** copia de un cstring a otro
- `string::`**`c_str`**`()` es un método de `string` que retorna su contenido como cstring (con el `'\0`)

¿Qué pasa si c no es suficientemente grande?

Pasar de `char*` a `std::string`:

```
string s1(c), s2;
s2 = c;
```

- Operador de **asignación** y el **constructor** de `string` están sobrecargados para aceptar un cstring.

ARCHIVOS BINARIOS Y `std::string`S

Escribir:

```
string str = "Hola Mundo!";
char aux[256];
strcpy ( aux, str.c_str() );
ofstream arch("archi.dat", ios::binary);
arch.write( aux, sizeof(aux) );
```

Leer:

```
char aux[256];
ifstream arch("archi.dat", ios::binary);
arch.read( aux, sizeof(aux) );
string str = aux;
cout << str;  // muestra "Hola Mundo!"
```

`sizeof` solo es aplicable a arreglos ***estáticos***

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.