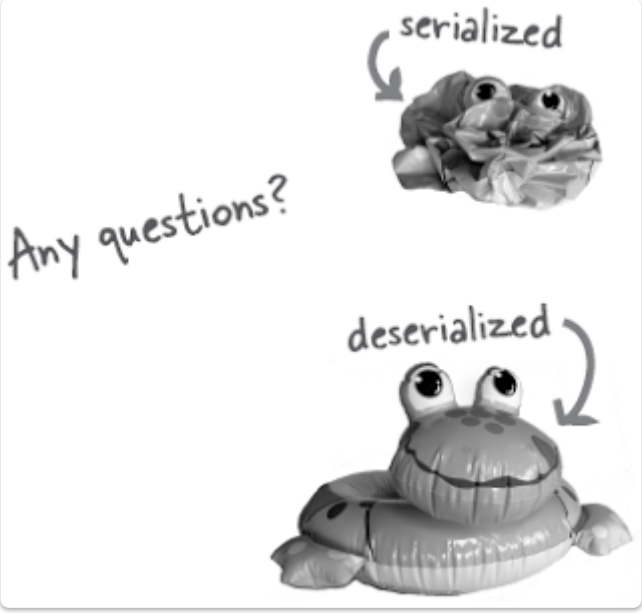
9. Persistenca de datos con archivos binarios

Tutor Isaac Palma Medina, CC BY-SA 4.0
Versión 12/07/2023
Karla Verónica Quiros Delgado



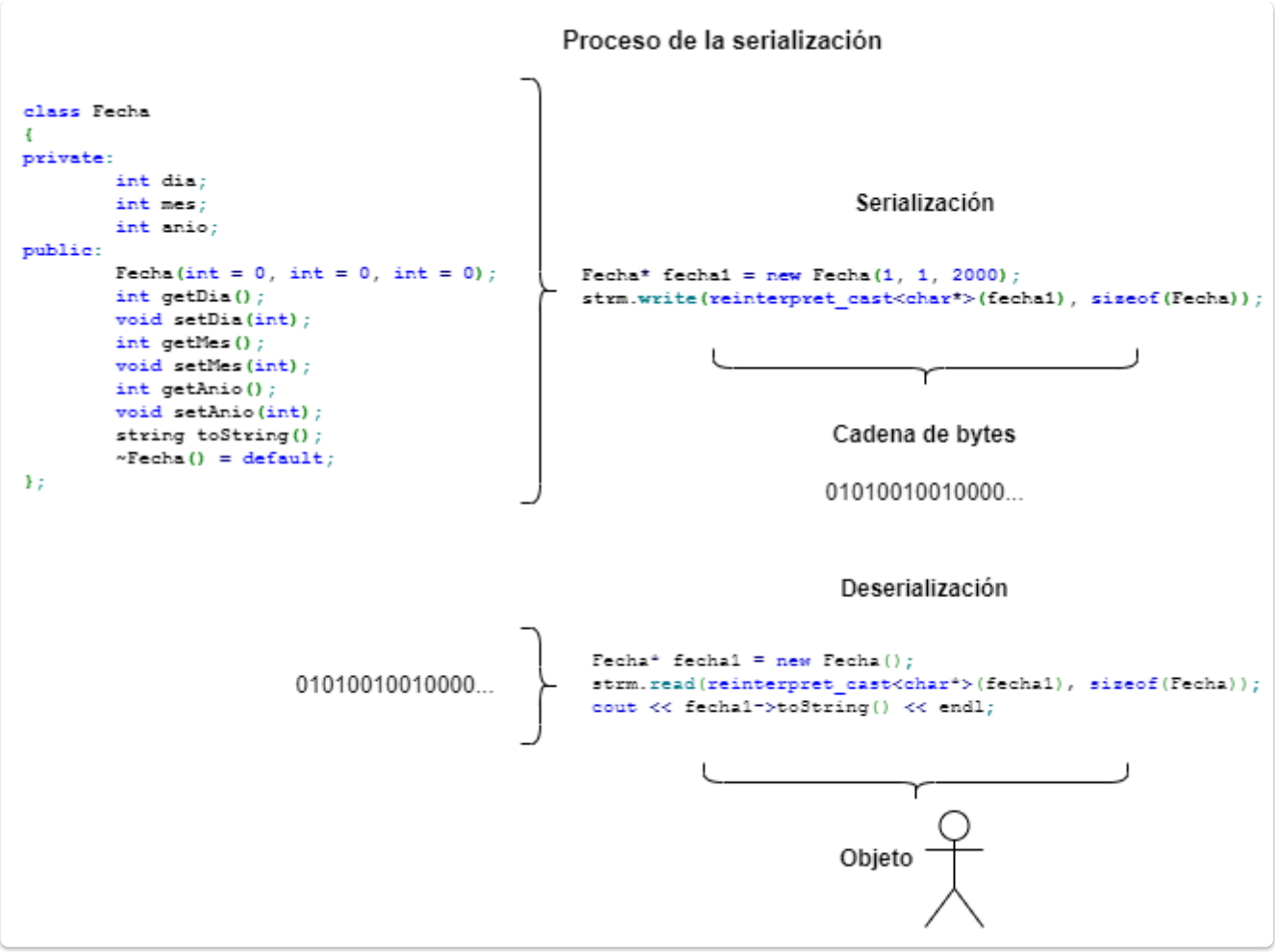
Los archivos binarios se utilizan para almacenar datos de forma directa en formato binario, representados por unos y ceros. Esta diferencia fundamental con respecto a los archivos de texto, donde los datos se guardan como caracteres legibles, tiene importantes implicaciones.

La principal ventaja de los archivos binarios radica en su eficiencia en términos de espacio de almacenamiento y tiempo de acceso. Dado que los datos se almacenan directamente en el formato que el programa comprende, no hay necesidad de realizar conversiones, lo que agiliza tanto la lectura como la escritura de datos. Sin embargo, esta eficiencia tiene un costo: los archivos binarios no son legibles para los humanos. Esto implica que no se pueden abrir en un editor de texto convencional para comprender su contenido, ya que la información está codificada en una serie de ceros y unos que no tienen un significado claro para el ojo humano.

Serialización y deserialización

El proceso de serialización implica codificar un objeto en un medio de almacenamiento, como un archivo o un buffer de memoria. Este objeto se transforma en una serie de bytes que permiten crear una réplica idéntica del original.

La serialización puede aplicarse directamente a los objetos. Por ejemplo:



La forma anterior funcionaría y permitiría la serialización correcta de un objeto de tipo Fecha. Sin embargo, este proceso puede enfrentar problemas al manejar punteros en los objetos o cadenas de caracteres tipo `string`1.

Soluciones

> *Código referenciado: EjemploArchivosBinarios1*

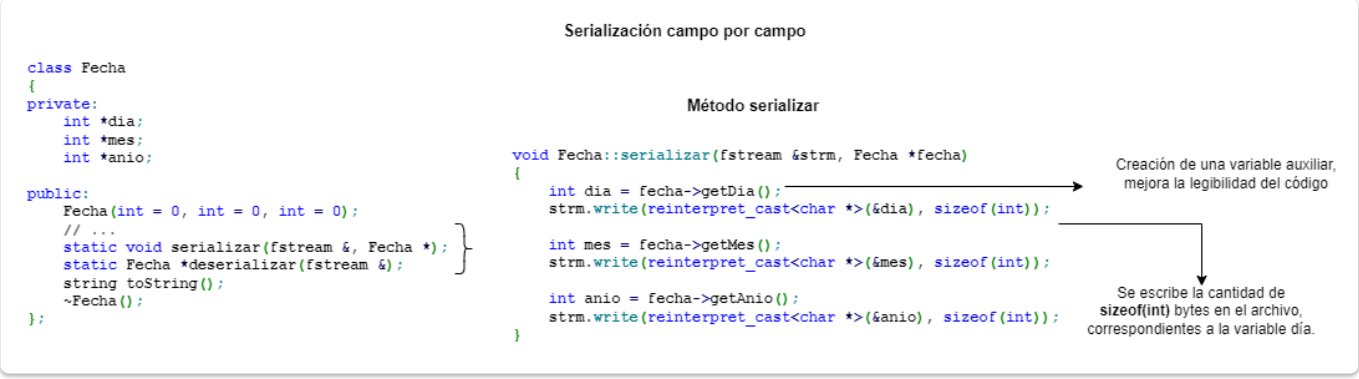
La solución más práctica sería serializar de forma independiente cada uno de los campos de un objeto. Considérese esta nueva implementación de la clase Fecha:

```
class Fecha
{
private:
    int *dia;
    int *mes;
    int *anio;

public:
    Fecha(int = 0, int = 0, int = 0);
    // ...
    static void serializar(fstream &, Fecha *);
```

```
    static Fecha *deserializar(fstream &);
    string toString();
    ~Fecha();
};
```

En el caso del método de serialización, se realiza una operación para cada uno de los componentes de la clase. Es importante observar que se indica cuántos bytes deben escribirse en el archivo. Esto puede presentar complicaciones al trabajar con strings, ya que estos tienen una longitud variable.



> *Estos métodos son estáticos, lo que permite serializar cualquier fecha, no únicamente la instancia actual de la clase.*

Serializar objetos relacionados con otros

> *Se recomienda leer el tema 8. Persistencia de datos para aclarar los tipos de relaciones al momento de guardarlos en archivos. En este caso, se trata de una relación simple de 1 a 1; no obstante, los conceptos expresados en el tema 8 son aplicables a esta situación.*

Tomando en cuenta la siguiente clase (que tiene un objeto interno de tipo Fecha, y todos sus atributos son punteros):

```
#define LONGITUD_MAXIMA_STRING 50

class Persona
{
private:
    int *id;
    string *nombreCompleto;
    Fecha *fechaNacimiento;

public:
    Persona(int = 0, string = "", Fecha * = nullptr);
    // ...
    static void serializar(fstream &, Persona *);
    static Persona *deserializar(fstream &);
    string toString();
```

```
    ~Persona();
};
```

El método de serialización sería el siguiente:

```
void Persona::serializar(fstream &strm, Persona *persona)
{

    int id = persona->getId();
    strm.write(reinterpret_cast<char *>(&id), sizeof(int));

    string nombreCompleto = persona->getNombreCompleto();
    const char *nombreCompletoCStr = nombreCompleto.c_str();
    strm.write(nombreCompletoCStr, LONGITUD_MAXIMA_STRING);

    Fecha::serializar(strm, persona->getFechaNacimiento());
}
```

1. Escritura de la identificación de la persona:

```
int id = persona->getId();
strm.write(reinterpret_cast<char *>(&id), sizeof(int));
```

2. Escritura del nombre de la persona:

```
string nombreCompleto = persona->getNombreCompleto(); // Se recupera el nombre de la persona.
const char *nombreCompletoCStr = nombreCompleto.c_str(); // Se transforma de un string a una cadena de carecteres más "pura".
strm.write(nombreCompletoCStr, LONGITUD_MAXIMA_STRING); // Se escribe la cadena de caracteres, asumiendo una longitud máxima de tamaño LONGITUD_MAXIMA_STRING (50)
```

Implicaciones de usar `LONGITUD_MAXIMA_STRING`

- *Ventajas*
  - Simplicidad: simplifica la serialización y deserialización, ya que elimina la necesidad de calcular dinámicamente las longitudes de los strings.
  - Eficiencia: al usar tamaños de datos fijos, las operaciones de lectura y escritura en archivos son más rápidas
- *Desventajas*
  - Desperdicio de memoria: puede llevar al desperdicio de memoria (de disco) si la mayoría de los strings que se almacenan son más cortos que el tamaño máximo especificado.
  - Limitación en la longitud de los strings.

> *Este enfoque es adecuado cuando se está seguro de que los strings nunca excederán el tamaño máximo definido y cuando la eficiencia en la lectura y escritura de datos es una preocupación importante.*

3. Escritura de la fecha:

```
Fecha::serializar(strm, persona->getFechaNacimiento()); // Se llama al método serializar de la clase Fecha.
```

En cuanto a la deserialización, el método posee la siguiente forma:

```
Persona *Persona::deserializar(fstream &strm)
{
    int id;
    strm.read(reinterpret_cast<char *>(&id), sizeof(int));

    char nombreCompleto[LONGITUD_MAXIMA_STRING];
    strm.read(nombreCompleto, LONGITUD_MAXIMA_STRING);

    Fecha *fechaNacimiento = Fecha::deserializar(strm);

    return new Persona(id, nombreCompleto, fechaNacimiento);
}
```

Para comprender cómo utilizar estos métodos en conjunto con una estructura de datos, como un arreglo, consulte el código *EjemploArchivosBinarios1*.

Referencias

[1]: rcplusplus. (2012). *Serializing a class with a pointer in C++.* Recuperado de https://stackoverflow.com/questions/9778806/serializing-a-class-with-a-pointer-in-c