Programación 1

Tema 15

Ficheros binarios

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Índice

- Ficheros binarios
  - Diferencia con ficheros de texto
- Herramientas de C++ para trabajar con ficheros binarios
- Problemas básicos con ficheros binarios
  - Creación
  - Lectura

Ficheros binarios

- Almacenan una secuencia de datos codificados en binario.
  - Cada dato se almacena como un grupo consecutivo de *bytes*.
  - Para cada dato, se utiliza una codificación binaria similar a la que se utiliza en la memoria del computador.

Ficheros binarios

- Ejemplo:
  - Un dato de tipo **`int`** se almacena en un fichero binario como 4 *bytes* consecutivos en los que el entero está codificado en binario en complemento a 2.

Ficheros binarios

- Ventajas
  - Reducción del tamaño de los ficheros
  - Se facilitan las operaciones de lectura y escritura
    - Simplificación de las instrucciones que es necesario programar
    - Reducción del tiempo de lectura o escritura
- Desventajas
  - No legibles por seres humanos
  - Pueden aparecer problemas de portabilidad

Diferencias entre un fichero binario y un fichero de texto

- Ejemplo: 26173, dato de tipo **`int`**
  - Codificación en un **fichero de texto**:
    - 00110010 00110110 00110001 00110111 00110011
    - (= Secuencia de *bytes* 50, 54, 49, 55 y 51)
    - (= Secuencia de caracteres de códigos 50, 54, 49, 55 y 51)
    - (= Secuencia de caracteres '2', '6', '1', '7' y '3')
  - Codificación en un **fichero binario**:
    - 00111101 01100110 00000000 00000000
    - (= Secuencia de *bytes* 61, 102, 0 y 0)
    - (= 4 *bytes* que codifican el número 26173 en base 2 en complemento a 2, con el byte menos significativo en primer lugar)

Diferencias entre un fichero binario y un fichero de texto

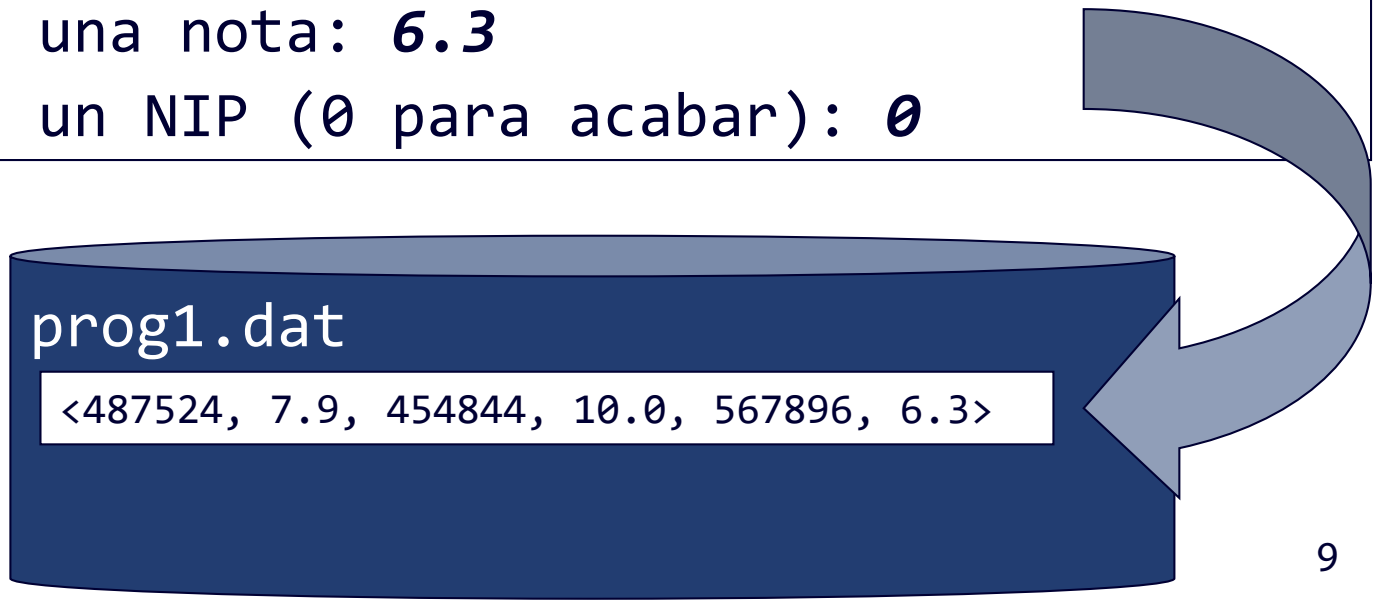
	Fichero de texto	**Fichero binario**
Interpretación de la secuencia de *bytes*	Caracteres	Codificación interna binaria de datos
¿Estructurado en líneas?	Sí	No
Necesidad de separadores entre datos	Habitualmente, sí	Habitualmente, no
Legible por una persona	Sí	No

Trabajo con ficheros binarios

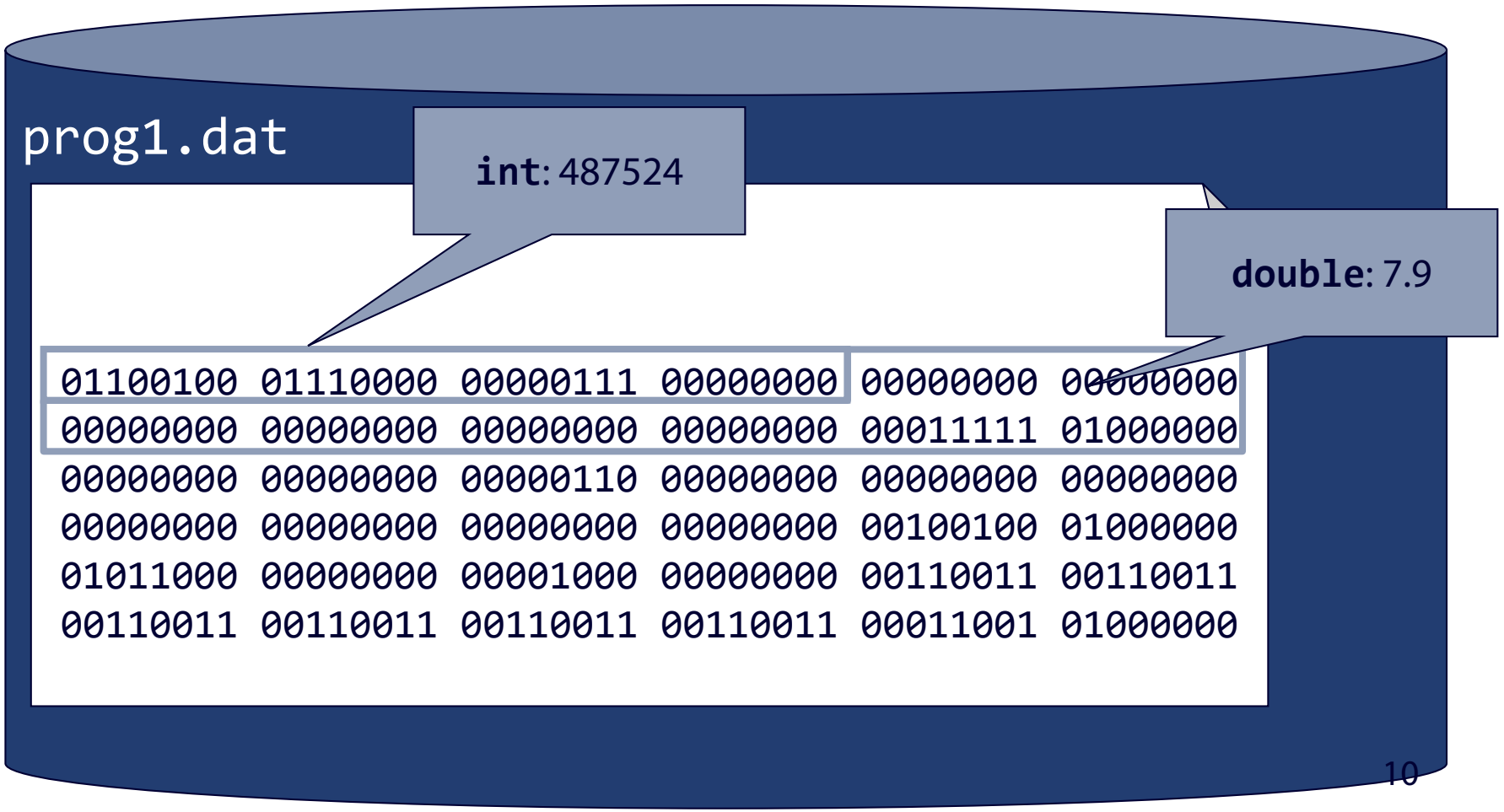
- Asociación
  - `f.open(const char cadena[], ios::binary)`
- Lectura
  - `f.read(char buffer[], streamsize n)`
    - `f.read(reinterpret_cast<char*>(&dato), sizeof(dato))`
- Escritura
  - `f.write(const char buffer[], streamsize n)`
    - `f.write(reinterpret_cast<const char*>(&dato), sizeof(dato))`

Creación de un fichero binario

```
Introduzca un NIP (0 para acabar): 487524
Introduzca una nota: 7.9
Introduzca un NIP (0 para acabar): 454844
Introduzca una nota: 10.0
Introduzca un NIP (0 para acabar): 567896
Introduzca una nota: 6.3
Introduzca un NIP (0 para acabar): 0
```



Creación de un fichero binario



Creación de un fichero binario. Sintaxis

```
<fichero_de_notas> ::= { <nota> }
<nota> ::= <nip> <calificación>
<nip> ::= int
<calificación> ::= double
```

Creación de un fichero binario

```
/*
 * Pre:  ---
 * Post: Ha creado un fichero binario de
 *       nombre «nombreFichero» compuesto
 *       por una secuencia de pares (NIP,
 *       nota) solicitados
 *       interactivamente al usuario.
 */
void crearFicheroNotas(
                const char nombreFichero[]);
```

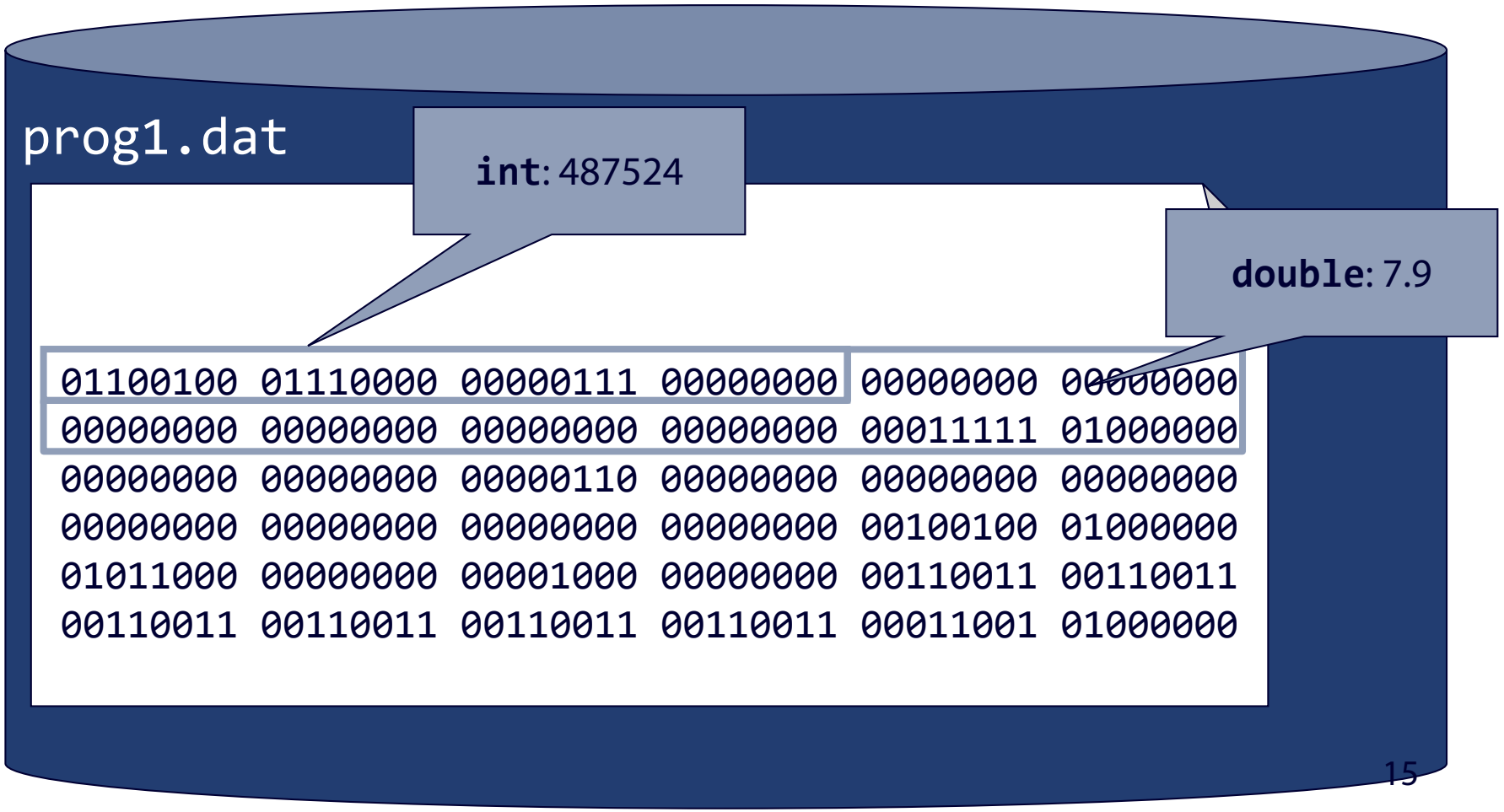
Creación de un fichero binario

```
void crearFicheroNotas(const char nombreFichero[]) {
    ofstream f(nombreFichero, ios::binary);
    if (f.is_open()) {
        cout << "Introduzca un NIP (0 para acabar): ";
        int nip;
        cin >> nip;
        while (nip != 0) {
            cout << "Introduzca una nota: ";
            double nota;
            cin >> nota;
            f.write(reinterpret_cast<const char*>(&nip),
                    sizeof(nip));
            f.write(reinterpret_cast<const char*>(&nota),
                    sizeof(nota));
            cout << "Introduzca un NIP (0 para acabar): ";
            cin >> nip;
        }
        ...
```

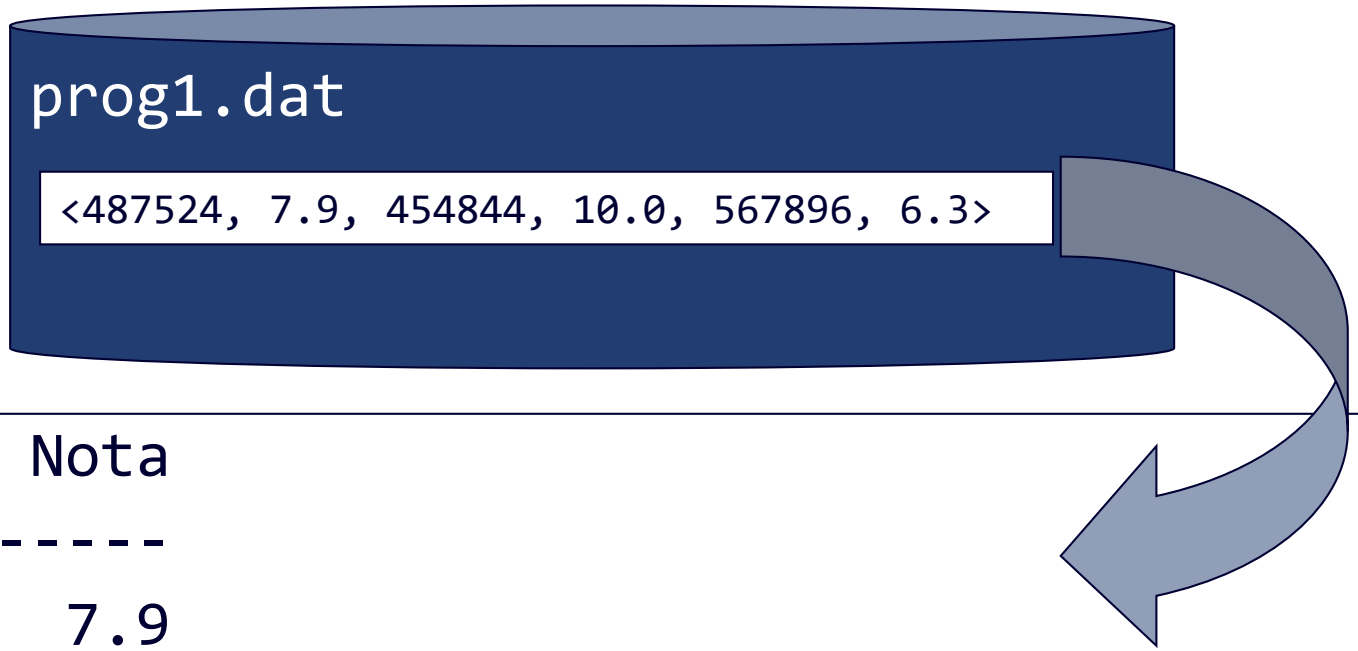
Creación de un fichero binario

```
void crearFicheroNotas(
                        const char nombreFichero[]) {
        ...
        f.close();
    }
    else {
        cerr << "No se ha podido escribir en el"
             << " fichero \"" << nombreFichero
             << "\"" << endl;
    }
}
```

Creación de un fichero binario



Lectura de un fichero binario




```
  NIP    Nota
------------
487524    7.9
454844   10.0
567896    6.3
```

Lectura de un fichero binario. Sintaxis

```
<fichero_de_notas> ::= { <nota> }
<nota> ::= <nip> <calificación>
<nip> ::= int
<calificación> ::= double
```

Lectura de un fichero binario

```
/*
 * Pre:  «nombreFichero» es el nombre de un fichero existente
 *       binario cuya estructura consiste en una secuencia de
 *       pares (NIP, nota), de tipos int y double,
 *       respectivamente.
 * Post: Ha mostrado en la pantalla del contenido del fichero de
 *       nombre «nombreFichero», de acuerdo con el siguiente
 *       formato de ejemplo:
 *
 *         NIP   Nota
 *       ------------
 *       487524   7.9
 *       454844  10.0
 *       567896   6.3
 */
void mostrarFicheroNotas(const char nombreFichero[]);
```

Lectura de un fichero binario

```
void mostrarFicheroNotas(const char nombreFichero[]) {
    ifstream f(nombreFichero, ios::binary);
    if (f.is_open()) {
        cout << "  NIP   Nota" << endl;
        cout << "------------" << endl;
        cout << fixed << setprecision(1);
        int nip;
        f.read(reinterpret_cast<char*>(&nip), sizeof(nip));
        while (!f.eof()) {
            double nota;
            f.read(reinterpret_cast<char*>(&nota),
                   sizeof(nota));
            cout << setw(6) << nip << " " << setw(5) << nota
                 << endl;
            f.read(reinterpret_cast<char*>(&nip), sizeof(nip));
        }
        ...
```

Lectura de un fichero binario

```
void mostrarFicheroNotas(
                        const char nombreFichero[]) {
        ...
        f.close();
    }
    else {
        cerr << "No se ha podido leer el "
             << "fichero \"" << nombreFichero
             << "\"" << endl;
    }
}
```

Otro ejemplo

- En una aplicación criptográfica necesitamos trabajar con un vector que contenga los primeros 5 000 000 números primos.
- Calcularlos cada vez que se inicia la aplicación es costoso en tiempo.
- Se ha decidido calcularlos solo una vez y almacenarlos en un fichero.

Fichero binario de números primos

```
#include <fstream>
#include <iostream>
#include "calculos.h"
using namespace std;

const int MAX_PRIMOS = 5000000;
const char NOMBRE_FICHERO_PRIMOS[] = "../primos.dat";

int main() {
    // En el repo GitHub se explica por qué se usa el modificador «static»
    static int primos[MAX_PRIMOS];
    inicializar(primos, MAX_PRIMOS);

    cout << "Vector con los primeros " << MAX_PRIMOS << " cargado." << endl;
    return 0;
}
```

Fichero binario de números primos

```
/*
 * Pre:  El vector «primos» tiene al menos «n»
 *       componentes.
 * Post: Las primeras «n» componentes del vector
 *       «primos» almacenan, en orden creciente, los
 *       primeros «n» números primos.
 *       Existe ahora también un fichero binario
 *       denominado «NOMBRE_FICHERO_PRIMOS» que
 *       almacena al menos los primeros «n» números
 *       primos en orden creciente.
 */
void inicializar(int primos[], const int n);
```

Fichero binario de números primos

```
void inicializar(int primos[], const int n) {
    bool ok = leerFicheroPrimos(primos, n);
    if (!ok) {
        calcular(primos, n);
        guardar(primos, n);
    }
}
```

Fichero binario de números primos

```
/*
 * Pre:  El vector «primos» tiene al menos «n» componentes.
 * Post: Si existe un fichero binario denominado «NOMBRE_FICHERO_PRIMOS» que
 *       contiene al menos los primeros «n» números primos almacenados en
 *       orden creciente, ha copiado los primeros «n» datos del fichero en
 *       las primeras «n» componentes del vector «primos», de forma que
 *       almacenan, en orden creciente, los primeros «n» números primos y ha
 *       devuelto «true». En caso contrario (si el fichero no existe o si
 *       contiene menos de «n» números primos), ha devuelto «false».
 */
bool leerFicheroPrimos(int primos[], const int n);
```

Fichero binario de números primos

```
bool leerFicheroPrimos(int primos[], const int n) {
    ifstream f;
    f.open(NOMBRE_FICHERO_PRIMOS, ios::binary);
    if (f.is_open()) {
        f.read(reinterpret_cast<char*>(primos),
               n * sizeof(int));
        bool ok = !f.eof();
        f.close();
        return ok;
    }
    else {
        return false;
    }
}
```

Fichero binario de números primos

```
/*
 * Pre:  El vector «primos» tiene al menos
 *       «n» componentes.
 * Post: Las primeras «n» componentes del
 *       vector «primos» almacenan, en orden
 *       creciente, los primeros «n» números
 *       primos, que han sido calculados por
 *       esta función.
 */
void calcular(int primos[], const int n);
```

Fichero binario de números primos

```
void calcular(int primos[], const int n) {
    primos[0] = 2;
    int i = 1;
    int p = 3;
    while (i < n) {
        if (esPrimo(p)) {
            primos[i] = p;
            i++;
        }
        p += 2;
    }
}
```

La función `esPrimo(int)` es la presentada en la transparencia 21 del tema 6 (enteros).

Fichero binario de números primos

```
/*
 * Pre:  El vector «primos» tiene al menos «n»
 *       componentes que almacenan, en orden
 *       creciente, los primeros «n» números primos.
 * Post: Ha escrito en un fichero binario denominado
 *       «NOMBRE_FICHERO_PRIMOS» los
 *       primeros «n» números primos en orden
 *       creciente, tal y como aparecen en
 *       el vector «primos».
 */
void guardar(const int primos[], const int n);
```

Fichero binario de números primos

```
void guardar(const int primos[], const int n) {
    ofstream f;
    f.open(NOMBRE_FICHERO_PRIMOS, ios::binary);
    if (f.is_open()) {
        f.write(reinterpret_cast<const char*>(primos),
        n * sizeof(int));
        f.close();
    }
    else {
        cerr << "Error creando el fichero \""
             << NOMBRE_FICHERO_PRIMOS << "\"." << endl;
    }
}
```

Resultados

- Ejecuciones con fichero binario
  - Intel Core i5-6500 a 3,20 GHz,
    16 GB de RAM, Windows 10
    - Cuando el fichero no existe: t ≈ 66 s
    - Cuando existe: t ≈ 0,25 s
  - Intel Celeron N3350 a 1,10 GHz
    4 GB de RAM, Windows 10
    - Cuando el fichero no existe: t ≈ 142 s
    - Cuando existe: t ≈ 0,64 s
- Tamaño del fichero:
  - Binario: ≈ 19 MB
  - Del fichero equivalente de texto: ≈ 46,9 MB

¿Cómo se puede estudiar este tema?

- Repasando estas transparencias
- Trabajando con el código de estas transparencias
  - https://github.com/prog1-eina/tema-15-ficheros-binarios
- Leyendo
  - Capítulo 15 de los apuntes del profesor Martínez
    - Disponibles en Moodle