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 como 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)
    - (= $61 \times 256^0 + 102 \times 256^1 + 0 \times 256^2 + 0 \times 256^3$)

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

Herramientas para trabajar con ficheros binarios en C++

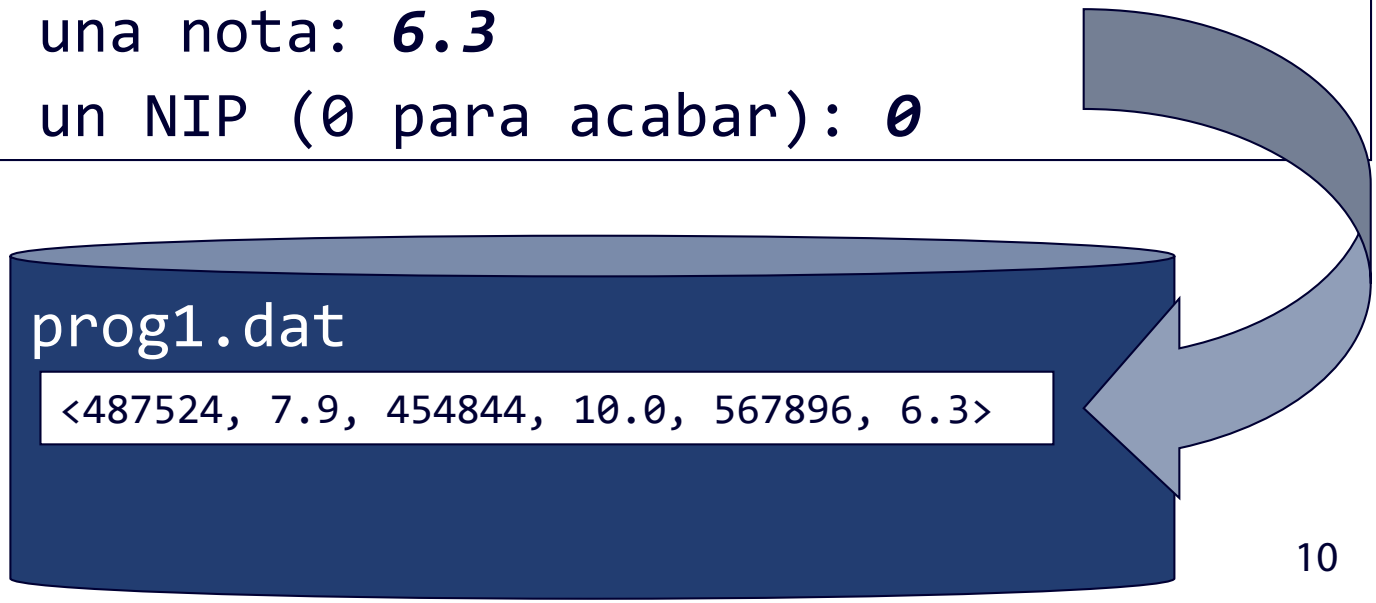
- Flujos de las clases **`ifstream`** y **`ofstream`** utilizados de una forma específica:
  - Trabajando solo con los métodos básicos para extraer o insertar *byte* a *byte*.
  - Entendiendo los valores del tipo **`char`** no como caracteres, sino como enteros de 8 bits.

Trabajo con ficheros binarios

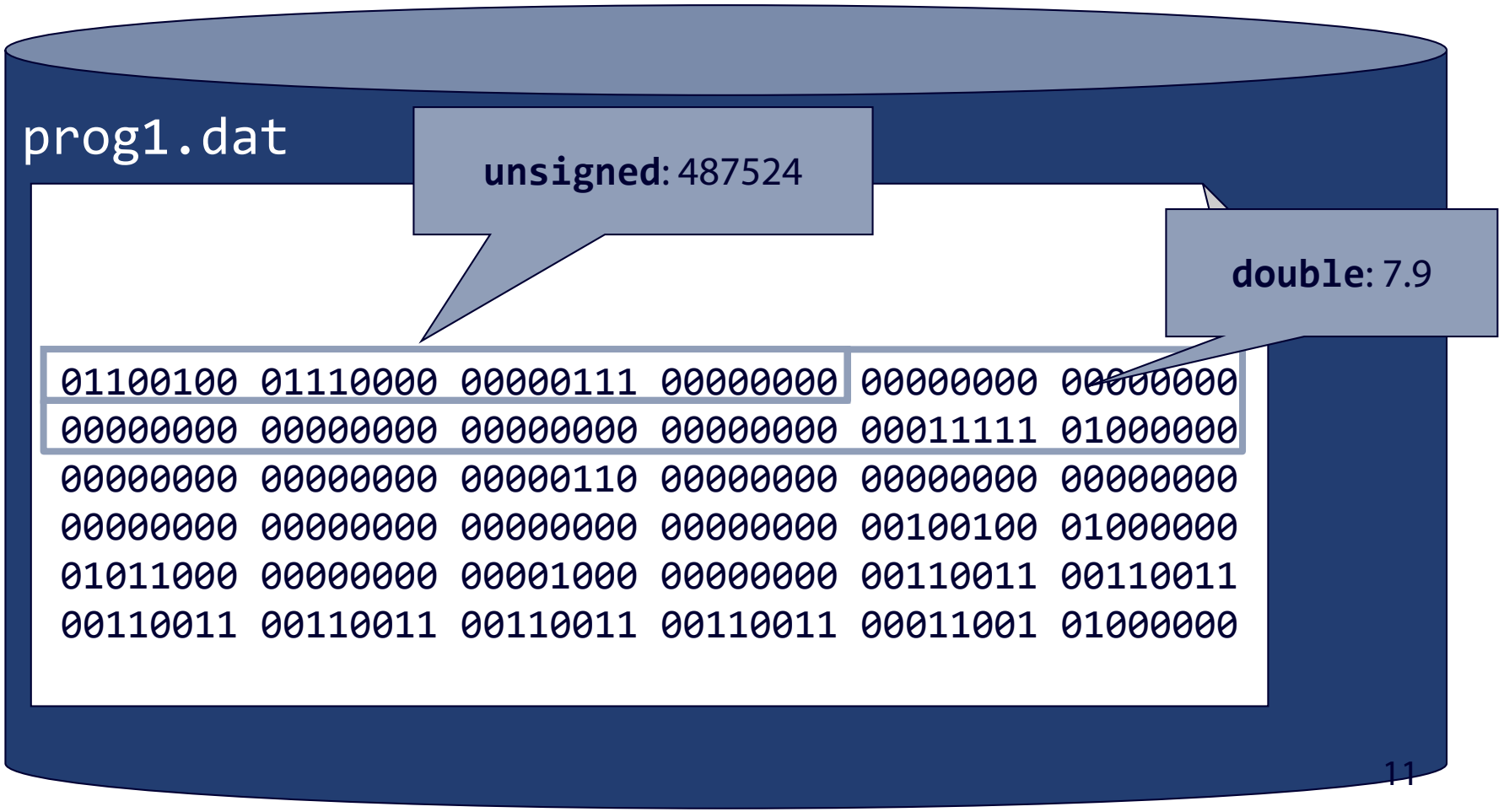
- Asociación
  - `f.open(const string cadena, ios::binary)`
- Lectura
  - `f.get(char& c)`
  - `f.read(char buffer[], streamsize n)`
    - `f.read(reinterpret_cast<char*>(&dato), sizeof(dato))`
- Escritura
  - `f.put(const char c)`
  - `f.write(const char buffer[], streamsize n)`
    - `f.write(reinterpret_cast<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> ::= unsigned
<calificación> ::= double
```

Creación de un fichero binario

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

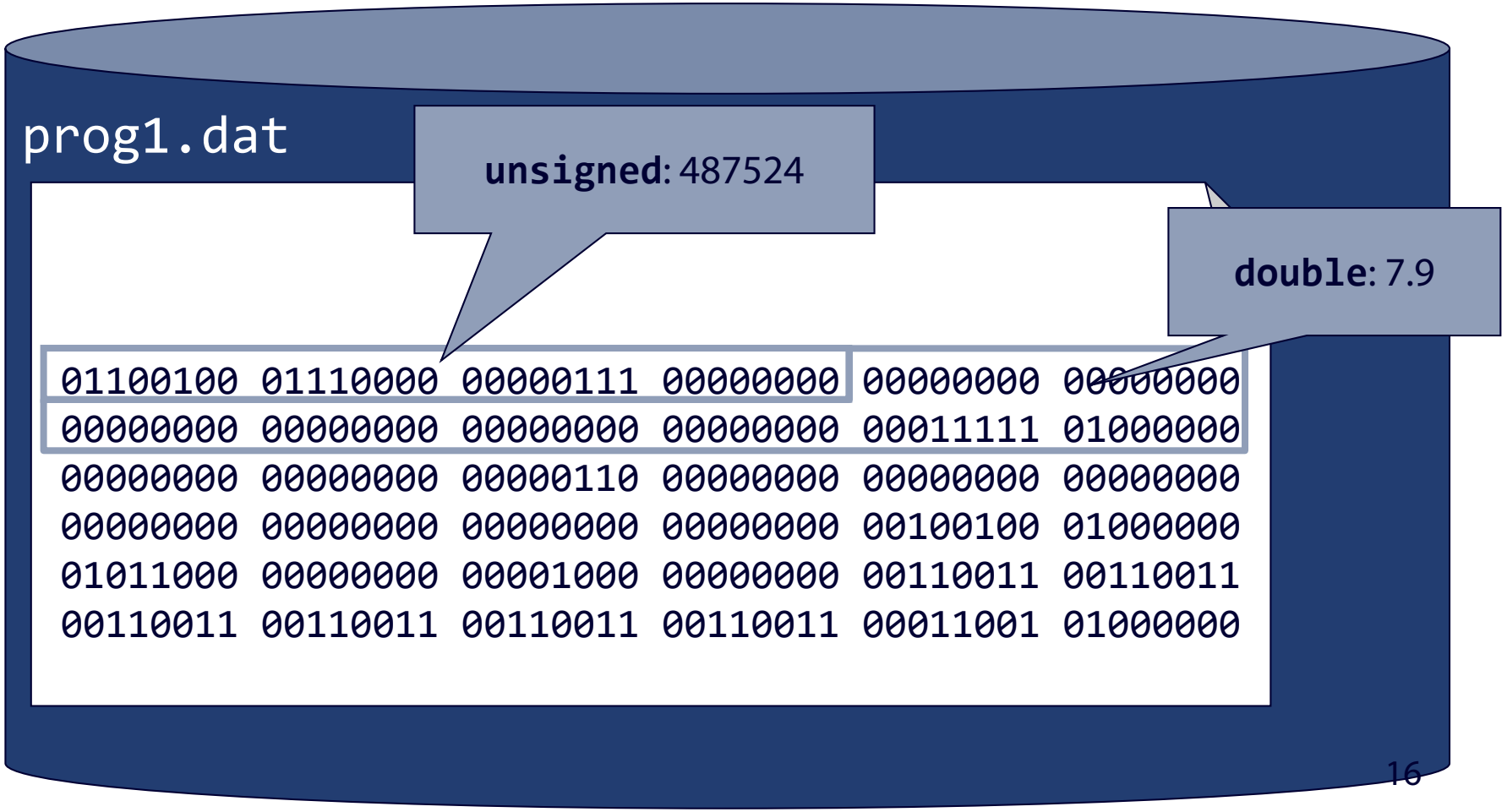
Creación de un fichero binario

```
void crearFicheroNotas(const string nombreFichero) {
    ofstream f(nombreFichero, ios::binary);
    if (f.is_open()) {
        cout << "Introduzca un NIP (0 para acabar): ";
        unsigned 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;
        }
        ...
```

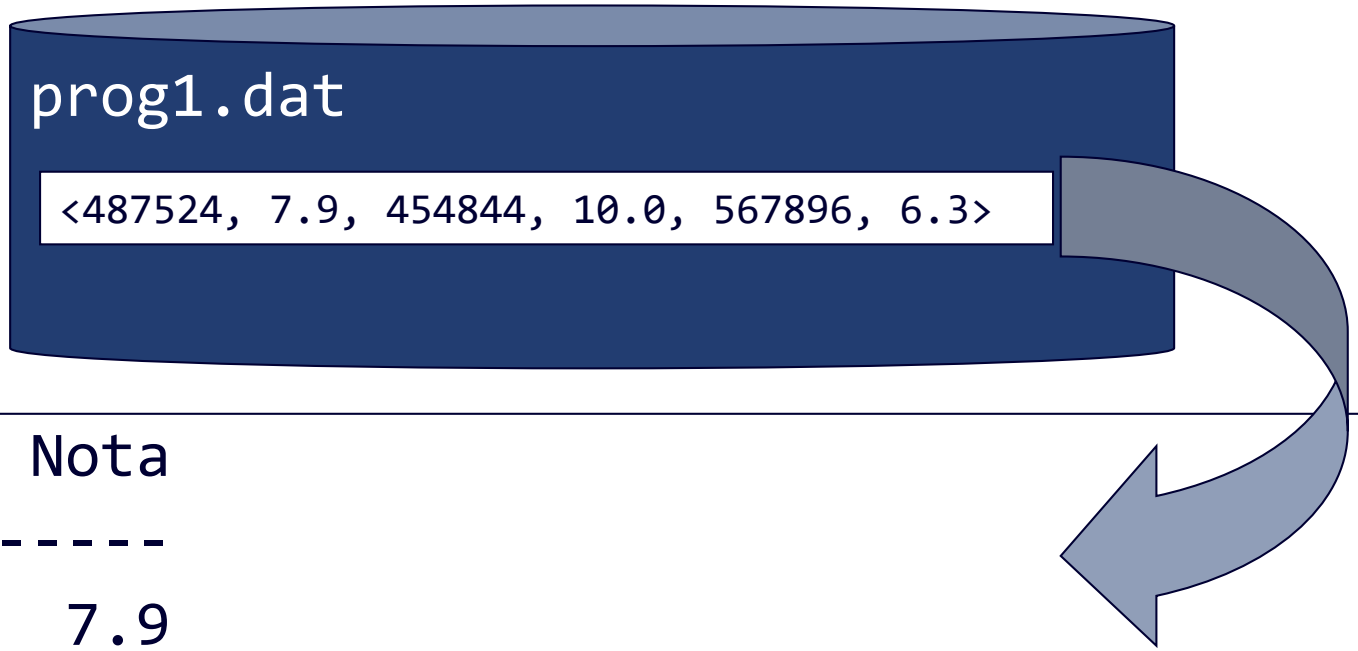
Creación de un fichero binario

```
void crearFicheroNotas(
                        const string 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> ::= unsigned
<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 unsigned y double,
 *       respectivamente.
 * Post: Muestra 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 string nombreFichero);
```

Lectura de un fichero binario

```
void mostrarFicheroNotas(const string nombreFichero) {
   ifstream f(nombreFichero, ios::binary);
   if (f.is_open()) {
      cout << "  NIP   Nota" << endl;
      cout << "------------" << endl;
      cout << fixed << setprecision(1);
      unsigned nip;
      while (                                                    ){
         double nota;


         cout << setw(6) << nip << " " << setw(5) << nota
              << endl;
      }
      ...
```

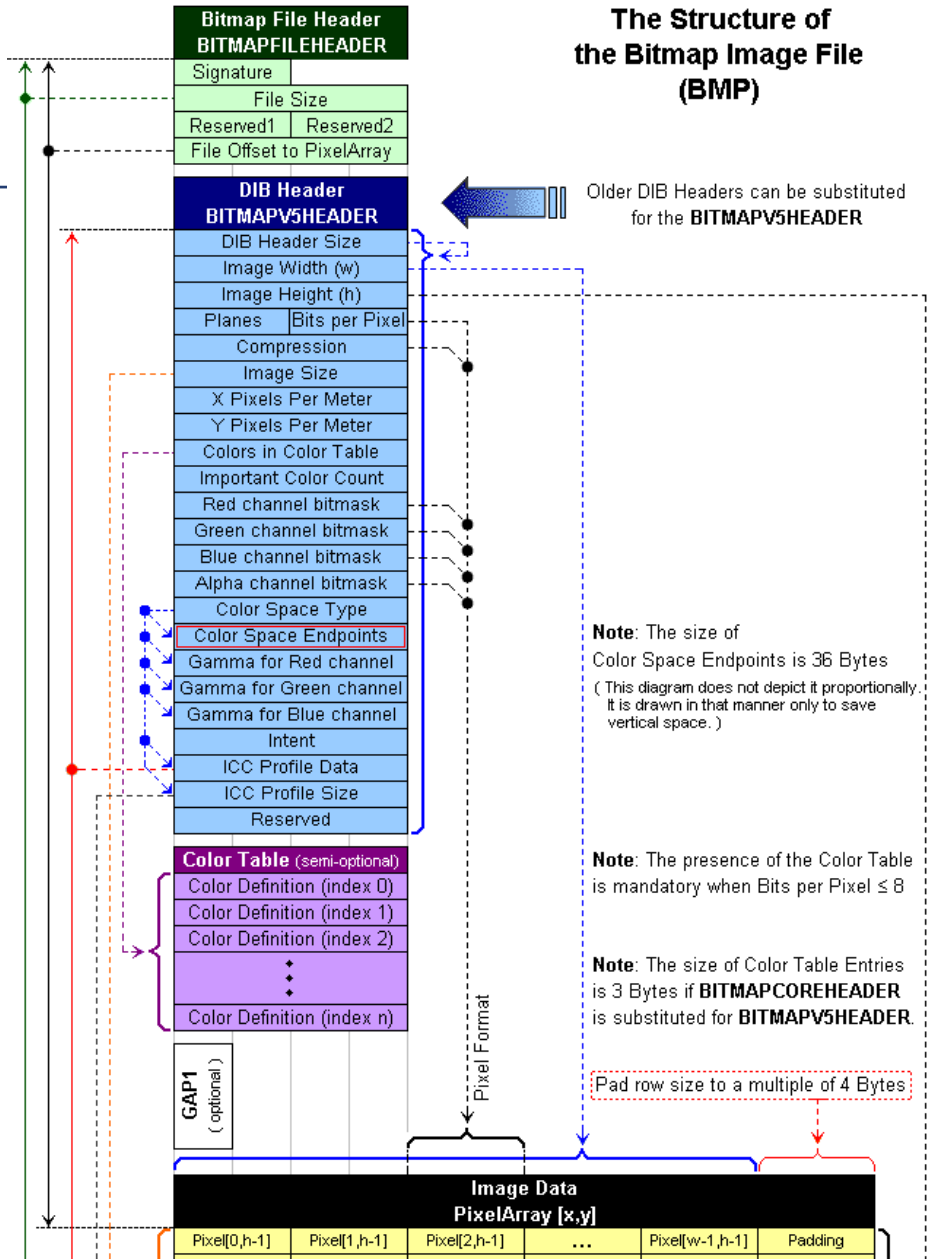
Lectura de un fichero binario

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

Lectura de un fichero binario

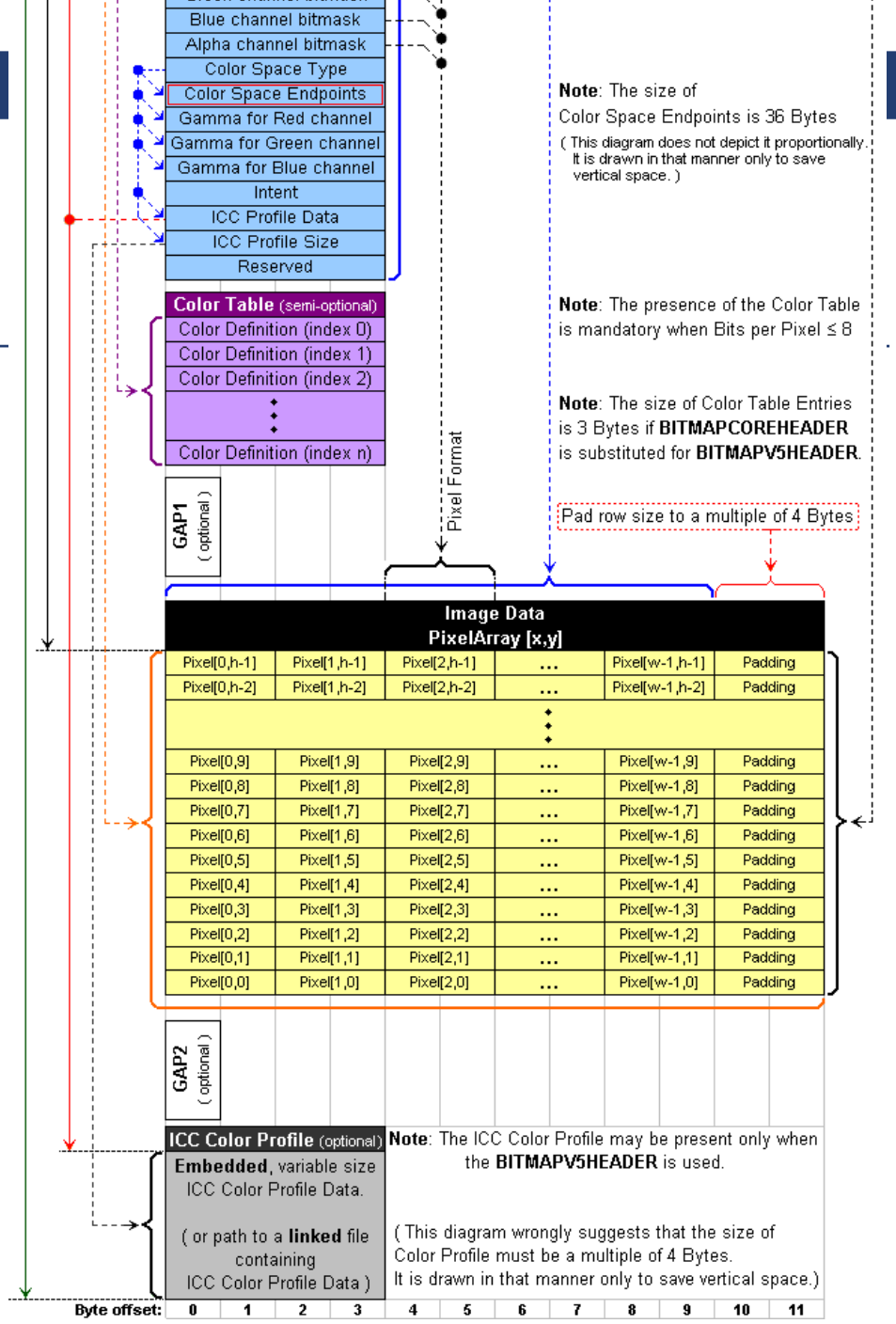
```
void mostrarFicheroNotas(
                        const string nombreFichero) {
        ...
        f.close();
    } else {
        cerr << "No se ha podido leer el "
             << "fichero \"" << nombreFichero
             << "\"" << endl;
    }
}
```

Ejemplo.
Ficheros BMP



Fuente: File:BMPfileFormat.png. (2020, September 9). *Wikimedia Commons, the free media repository*. Retrieved 18:31, December 4, 2020 from https://commons.wikimedia.org/w/index.php?title=File:BMPfileFormat.png&oldid=452662221.

Ejemplo.
Ficheros BMP



Fuente: File:BMPfileFormat.png. (2020, September 9). *Wikimedia Commons, the free media repository*. Retrieved 18:31, December 4, 2020 from https://commons.wikimedia.org/w/index.php?title=File:BMPfileFormat.png&oldid=452662221.

Ejemplo. Ficheros BMP

```
const unsigned ANCHO = 4096;
const unsigned ALTO = 2160;
const unsigned TAM_CABECERA_1 = 18;
const unsigned TAM_CABECERA_2 = 28;

struct Pixel {
    char rojo, verde, azul;
};

struct Imagen {
    unsigned ancho, alto;
    Pixel pixels[ANCHO][ALTO];
    char cabecera_parte1[TAM_CABECERA_1];
    char cabecera_parte2[TAM_CABECERA_2];
};
```

Ejemplo.
Ficheros BMP

```
/*
 * Pre:  «nombreFichero» es un fichero binario en formato
 *       BMP.
 * Post: Si se ha encontrado el fichero y este tiene unas
 *       dimensiones correctas, tras ejecutar este
 *       procedimiento, «imagen» almacena en memoria la
 *       imagen almacenada en un fichero binario en
 *       formato BMP y la función ha devuelto true. En
 *       caso contrario, ha devuelto false y ha escrito en
 *       la pantalla un mensaje de error indicando la
 *       causa del mismo.
 */
bool leerImagen(const string nombreFichero,
                Imagen& imagen);
```

Ejemplo. Ficheros BMP

```
bool leerImagen(const string nombreFichero, Imagen& imagen) {
    ifstream f(nombreFichero, ios::binary);
    if (f.is_open()){
        f.read(imagen.cabecera_parte1, TAM_CABECERA_1);
        if (imagen.cabecera_parte1[0] == 'B'
                && imagen.cabecera_parte1[1] == 'M') {
            f.read(reinterpret_cast<char*>(&imagen.ancho), sizeof(unsigned));
            if (imagen.ancho <= MAX_ANCHO && imagen.ancho % 4 == 0) {
                f.read(reinterpret_cast<char*>(&imagen.alto),
                        sizeof(unsigned));
                if (imagen.alto <= MAX_ALTO && imagen.alto % 4 == 0) {
                    f.read(imagen.cabecera_parte2, TAM_CABECERA_2);
                    leerPixeles(f, imagen);
                    cout << "Imagen leída con éxito." << endl;
                    f.close();
                    return true;
                }
    // «elses» correspondientes a situaciones de error
}
```

Ejemplo. Ficheros BMP

```
/*
 * Pre:  «f» está asociado con un fichero externo de formato
 *       bitmap de dimensiones múltiplo de 4 y máximo
 *       800 píxeles y ya se ha extraído del flujo la cabecera,
 *       estándose en disposición de extraer el primer píxel;
 *       imagen.alto e imagen.ancho representan el ancho y alto
 *       de la imagen, en píxeles.
 * Post: Extre los píxeles de «f» y se los asigna
 *       a las primeras imagen.alto filas y imagen.ancho
 *       columnas del registro «imagen».
 */
void leerPixeles(ifstream& f, Imagen& imagen);
```

Ejemplo.
Ficheros BMP

```
void leerPixeles(ifstream& f, Imagen& imagen) {
    for (unsigned i = 0; i < imagen.alto; i++) {
        for (unsigned j = 0; j < imagen.ancho; j++){
            f.get(imagen.pixels[i][j].rojo);
            f.get(imagen.pixels[i][j].verde);
            f.get(imagen.pixels[i][j].azul);
        }
    }
}
```

Ejemplo.
Ficheros BMP

```
/*
 * Pre:  ---
 * Post: Tras ejecutar este procedimiento,
 *       almacena en disco en un fichero
 *       de nombre «nombreFichero» la
 *       imagen BMP de «imagen».
 */
void guardarImagen(
                  const string nombreFichero,
                  const Imagen& imagen);
```

Ejemplo.
Ficheros BMP

```
void guardarImagen(const string nombreFichero, const Imagen& imagen) {
    ofstream f(nombreFichero, ios::binary);
    if (f.is_open()) {
        f.write(imagen.cabecera_parte1, TAM_CABECERA_1);
        f.write(reinterpret_cast<const char*>(&imagen.ancho), sizeof(int));
        f.write(reinterpret_cast<const char*>(&imagen.alto), sizeof(int));
        f.write(imagen.cabecera_parte2, TAM_CABECERA_2);
        for (unsigned i = 0; i < imagen.alto; i++){
            for (unsigned j = 0; j < imagen.ancho; j++){
                f.write(&imagen.pixels[i][j].rojo,  sizeof(char));
                f.write(&imagen.pixels[i][j].verde, sizeof(char));
                f.write(&imagen.pixels[i][j].azul,  sizeof(char));
            }
        }
        f.close();
    }
    ...
}
```