

# Programación 1

## Tema 16

---

Ficheros: otras posibilidades



Escuela de  
Ingeniería y Arquitectura  
**Universidad** Zaragoza





# Objetivos

---

- Trabajo de forma no secuencial con ficheros
  - Modo *append*
  - Acceso directo
  - Modo entrada y salida



# Objetivos

---

- Trabajo de forma no secuencial con ficheros
  - **Modo *append***
  - Acceso directo
  - Modo entrada y salida

# Añadir datos a un fichero

```
/*  
 * Pre: «nombreFichero» hace referencia a un  
 * fichero de texto existente y  
 * modificable.  
 * Post: Inserta al final del fichero de  
 * denominado «nombreFichero» una línea  
 * completa cuyo contenido sea la  
 * secuencia de caracteres de «linea».  
 */  
void unaLineaAdicional(  
    const char nombreFichero[],  
    const char linea[]);
```

# Una solución

```
void unaLineaAdicional(const char nombreFichero[],
                      const char linea[]) {
    const char FICHERO_TEMPORAL[] = "temporal.tmp";
    ifstream fOriginal;
    fOriginal.open(nombreFichero);
    if (fOriginal.is_open()) {
        ofstream fTemporal;
        fTemporal.open(FICHERO_TEMPORAL);
        if (fTemporal.is_open()) {
            ...
        }
        else {cerr << "No se ha podido escribir el fi..." << endl;
              fOriginal.close(); }
    }
    else { cerr << "No se ha podido leer el fichero ..." << endl;}
}
```

# Una solución

```
void unaLineaAdicional(const char nombreFichero[],  
                       const char linea[]) {
```

```
    ...
```

```
    char c;  
    fOriginal.get(c);  
    while (!fOriginal.eof()) {  
        fTemporal.put(c);  
        fOriginal.get(c);  
    }  
    fTemporal << linea << endl;  
    fTemporal.close();  
    fOriginal.close();  
    remove(nombreFichero);  
    rename(FICHERO_TEMPORAL, nombreFichero);
```

```
    ...
```

```
}
```

Funciones remove y rename  
definidas en <cstdio>

# Función rename

- Biblioteca <stdio>
- **int rename ( const char oldname[], const char newname[] );**
  - **Rename file**
  - Changes the name of the file or directory specified by *oldname* to *newname*.
  - This is an operation performed directly on a file; No streams are involved in the operation.
  - If *oldname* and *newname* specify different paths and this is supported by the system, the file is moved to the new location.
  - If *newname* names an existing file, the function may either fail or override the existing file, depending on the specific system and library implementation.
  - Proper file access shall be available.
- **Parameters**
  - *oldname*: C string containing the name of an existing file to be renamed and/or moved. Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system).
  - *newname*: C string containing the new name for the file. [...]
- **Return value**
  - If the file is successfully renamed, a zero value is returned.
  - On failure, a nonzero value is returned.

# Función remove

---

- Biblioteca <stdio>
- **int remove ( const char filename[] );**
  - **Remove file**
  - Deletes the file whose name is specified in *filename*.
  - This is an operation performed directly on a file identified by its *filename*; No streams are involved in the operation.
  - Proper file access shall be available.
- **Parameters**
  - **filename**: C string containing the name of the file to be deleted. Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system).
- **Return value**
  - If the file is successfully deleted, a zero value is returned.
  - On failure, a nonzero value is returned.



# Una solución mejor

```
void unaLineaAdicional(const char nombreFichero[],
                       const char linea[]) {
    ofstream f;
    f.open(nombreFichero, ios::app);
    if (f.is_open()) {
        f << linea << endl;
        f.close();
    }
    else {
        cerr << "No se ha podido escribir en el fichero"
              << "\"" << nombreFichero << "\"." << endl;
    }
}
```



# Objetivos

---

- Trabajo de forma no secuencial con ficheros
  - Modo *append*
  - **Acceso directo**
  - Modo entrada y salida

# Acceso directo a los datos de un fichero

```
/*  
 * Pre: El de fichero de nombre «nombreFichero»  
 * almacena la siguiente información  
 * codificada en binario que representa  
 * una secuencia de NIF: en primer lugar  
 * el número de NIF almacenados en el  
 * fichero; este número es mayor que cero  
 * y mayor que «i». Le sigue una secuencia  
 * de datos de tipo «Nif».  
 * Post: Asigna a «nif» el NIF «i»-ésimo  
 * almacenado en el fichero  
 */  
void leerUnNifBin(const char nombreFichero[],  
                  const int i, Nif& nif);
```

# Acceso directo a los datos de un fichero

```
<ficheroNIFBinarioAlt1>  
    ::= <número> { <nif> }  
<número> ::= int  
<nif> ::= Nif
```

# Una solución

```
void leerUnNifBin(const char nombreFichero[], const int i,
                  Nif& nif) {
    ifstream f;
    f.open(nombreFichero, ios::binary);
    if (f.is_open()) {
        int n;
        f.read(reinterpret_cast<char*>(&n), sizeof(n));
        for (int j = 0; j < i; j++) {
            f.read(reinterpret_cast<char*>(&nif),
                    sizeof(nif));
        }
        f.close();
    }
    else {
        cerr << "No se ha podido leer el fichero \""
              << nombreFichero << "\"" << endl;
    }
}
```

# Una solución mejor

```
void leerUnNifBin(const char nombreFichero[], const int i,
                  Nif& nif) {
    ifstream f;
    f.open(nombreFichero, ios::binary);
    if (f.is open()) {
        f.seekg(sizeof(int) + (i - 1) * sizeof(Nif));
        f.read(reinterpret_cast<char*>(&nif),
                sizeof(nif));
        f.close();
    }
    else {
        cerr << "No se ha podido leer el fichero \""
              << nombreFichero << "\"\" << endl;
    }
}
```



# Objetivos

---

- Trabajo de forma no secuencial con ficheros
  - Modo *append*
  - Acceso directo
  - **Modo entrada y salida**

# Lectura y escritura de datos de un mismo fichero

```
/*  
 * Pre: El fichero de nombre «nombreFichero» almacena la  
 *       siguiente información codificada en binario que  
 *       representa una secuencia de NIF: En primer lugar el  
 *       número de NIF almacenados en el fichero; este número  
 *       es mayor que cero y mayor o igual que «i».  
 *       Le sigue una secuencia de datos de tipo «Nif».  
 * Post: Si ha sido necesario, ha corregido el valor de la letra  
 *       del «i»-ésimo NIF almacenado en el fichero de nombre  
 *       «nombreFichero».  
 */  
void corregirUnNifBin(const char nombreFichero[], const int i);
```



# Lectura y escritura de datos de un mismo fichero

```
void corregirUnNifBin(const char nombreFichero[], const int i) {  
    fstream f;  
    f.open(nombreFichero, ios::binary | ios::in | ios::out);  
    if (f.is_open()) {  
        int posicion = sizeof(int) + (i - 1) * sizeof(Nif);  
        f.seekg(posicion);  
        Nif nif;  
        f.read(reinterpret_cast<char*>(&nif), sizeof(nif));  
        if (!esValido(nif)) {  
            nif.letra = calcularLetra(nif.dni);  
            f.seekp(posicion);  
            f.write(reinterpret_cast<const char*>(&nif),  
                    sizeof(nif));  
        }  
        f.close();  
    }  
    else { cerr << "No se ha podido abrir el..." << endl; }  
}
```