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
 *       cadena de caracteres «linea».
 */
void unaLineaAdicional(
                        const string nombreFichero,
                        const string linea);
```

Una solución

```
void unaLineaAdicional(const string nombreFichero,
                       const string linea) {
    const string 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 …" << endl;
            fOriginal.close(); }
    } else { cerr << "No se ha podido leer el ficher…" << endl;}
}
```

Una solución

```
void unaLineaAdicional(const string nombreFichero,
                       const string linea) {
    ...
    char c;
    while (fOriginal.get(c)) {
        fTemporal.put(c);
    }
    fTemporal << linea << endl;
    fTemporal.close();
    fOriginal.close();
    remove(nombreFichero);
    rename(FICHERO_TEMPORAL, nombreFichero);
    ...
}
```

Funciones remove y rename definidas en <cstdio>

Función `rename`

- Biblioteca `<cstdio>`
- `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.

Fuente: «rename». *Cplusplus.com.* 2000–2017. http://www.cplusplus.com/reference/cstdio/rename/ (consultado el 10-12-2018).

Función `remove`

- Biblioteca `<cstdio>`
- `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.

Fuente: «remove». *Cplusplus.com*. 2000–2017. http://www.cplusplus.com/reference/cstdio/remove/ (consultado el 10-12-2018).

Una solución mejor

```
void unaLineaAdicional(const string nombreFichero,
                       const string 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

Fichero de primos

- Supongamos que disponemos de un fichero binario que contiene los primeros 5 000 000 números primos, codificados en binario como datos de tipo **`unsigned`**

Acceso directo a los datos de un fichero

```
/*
 * Pre:  El fichero binario de nombre
 *       «nombreFichero» almacena al menos los
 *       primeros «i» números primos, almacenados en
 *       orden ascendente.
 * Post: Devuelve el «i»-ésimo (comenzando a
 *       contar por 1) número primo, según el
 *       contenido del fichero. Si no se puede
 *       abrir el fichero, escribe un mensaje de
 *       error en «cerr» y devuelve 1.
 */
unsigned leerUnPrimo(const string nombreFichero,
                     const unsigned i);
```

Una solución

```
unsigned leerUnPrimo(const string nombreFichero,
                     const unsigned i) {
    ifstream f(nombreFichero, ios::binary);
    if (f.is_open()) {
        unsigned primo;
        for(unsigned j = 1; j <= i; j++) {
            f.read(reinterpret_cast<char*>(&primo),
                   sizeof(primo));
        }
        f.close();
        return primo;
    } else {
        cerr << "No se ha podido leer el fichero \""
             << nombreFichero << "\"" << endl;
        return 1;
    }
}
```

Una solución mejor

```
unsigned leerUnPrimo(const string nombreFichero,
                     unsigned i) {
    ifstream f(nombreFichero, ios::binary);
    if (f.is_open()) {
        f.seekg((i - 1) * sizeof(unsigned));
        unsigned primo;
        f.read(reinterpret_cast<char*>(&primo),
               sizeof(primo));
        f.close();
        return primo;
    } else {
        cerr << "No se ha podido leer el fichero \""
             << nombreFichero << "\"" << endl;
        return 1;
    }
}
```

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» contiene los
 *       primeros números primos, almacenados en orden
 *       ascendente. El fichero contiene al menos dos primos.
 * Post: Añade al fichero el número primo que sigue al
 *       último que tenía inicialmente almacenado. Si no puede,
 *       escribe un mensaje de error en «cerr».
 */
void agregarSiguientePrimo(const string nombreFichero);
```

Lectura y escritura de datos de un mismo fichero

```
void agregarSiguientePrimo(const string nombreFichero) {
    fstream f(nombreFichero, ios::binary | ios::in | ios::out );
    if (f.is_open()) {
        unsigned primo;
        f.seekg(-1 * int(sizeof(unsigned)), ios_base::end);
        f.read(reinterpret_cast<char*>(&primo), sizeof(primo));

        primo += 2;
        while (!esPrimo(primo)) {
            primo += 2;
        }

        f.seekp(0, ios_base::end);
        f.write(reinterpret_cast<const char*>(&primo),
                sizeof(primo));
        f.close();
    } else { ... }
}
```