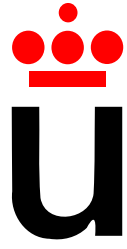
10. Manejo de ficheros

Julio Vega

julio.vega@urjc.es



Universidad
Rey Juan Carlos

Contenidos

- Variables y colecciones almacenan datos temporalmente en RAM.
- Los archivos almacenan datos permanentemente en disco.
- Vamos a tratar con archivos secuenciales y de acceso aleatorio.
- Los datos en un fichero pueden ser crudos o con formato.

- La ud. más pequeña de dato que maneja un ordenador es el **bit**.
  - Y cada bit puede contener el valor 0 o el valor 1.
- Un **byte** a su vez está compuesto por ocho bits.
- Pero programar a bajo nivel, con bits, es muy difícil.
  - Más fácil, con formatos: dígitos, letras y símbolos especiales.
  - Este es el conjunto de caracteres que maneja un ordenador.
- Un programador usa caracteres; C++ proporciona el tipo `char`.
  - Un `char` ocupa un byte; para Unicode, en C++ se usa `wchar_t`.

- C++ considera cada archivo como una secuencia de bytes.
- Cada archivo termina con un caracter especial de terminación.
  - O con un patrón específico de bytes reconocido por el sistema.
- Al abrir un archivo se crea un objeto al cual se asocia un flujo.
  - Algunos objetos especiales son `cin`, `cout`, `cerr` y `clog`.
    - Estos se crean al usar la librería `<iostream>`.
    - `cin`: objeto flujo de entrada estándar (teclado).
    - `cout`: objeto flujo de salida estándar (pantalla).
    - `cerr` y `clog`: objetos flujo de error estándar.

- Para tratar ficheros en C++ necesitamos `<fstream>` e `<iostream>`.
- `<fstream>`: `basic_ifstream`, `basic_ofstream` y `basic_fstream`.
  - Para la entrada, salida y e/s de ficheros respectivamente.
- Además, `<fstream>` ofrece alias `typedef` para estas funciones.
  - E.g. `typedef ifstream` es una especialización de `basic_ifstream`.
    - `ifstream` permite la entrada de valores `char` desde fichero.
  - Y, de igual forma, `typedef ofstream` y `typedef fstream`.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
using namespace std;

int main() {
   ofstream outUsersFile ("users.dat", ios::out); // open file

   if (!outUsersFile) { // file couldn't be opened
      cerr << "File could not be opened" << endl;
      exit (1);
   }
```

```
    cout << "Enter data according to 'name surname phone' (press
        CTRL+C to finish):\n";

    string name;
    string surname;
    int phone;
    while (cin >> name >> surname >> phone) {
       outUsersFile << name << ' ' << surname << ' ' << phone <<
           endl;
    }
}
```

- Los objetos `ofstream` se abren por defecto en modo salida.
  - La línea `ofstream outUsersFile (''users.dat'', ios::out);`
    - Es equivalente a `ofstream outUsersFile (''users.dat'');`
  - En este modo, el fichero se trunca: se sobreescriben los datos.
  - También se puede crear el objeto `ofstream` y después abrirlo.
    - `ofstream outUsersFile;`
    - `outUsersFile.open(''users.dat'');`
- Un objeto `ofstream` también puede ser abierto en modo `app`.
  - Con `ios::app` se añaden los datos al final del fichero.

- Al modificar datos en un archivo secuencial, se puede corromper.
  - E.g. si tenemos fichero con contenido *[...] Julio Vega 677387389 [...]*.
  - Si queremos modificar *Vega* por *Vega Pérez*:
    - El nuevo registro quedaría así: *Julio Vega Pérez 677387389*.
    - Pero este nuevo registro ocupa seis caracteres más que el original.
    - Serían seis caracteres que se sobrescribirían del siguiente registro.
- Para actualizar un registro en un archivo secuencial correcta/.
  - Copiar a otro fich. todos los registros antes del que hay que modificar.
  - Añadir el registro nuevo a ese nuevo fichero.
  - Y final/ copiar todos los registros que hay después del reg. modificado.

- `ios::app` añade toda la salida al final del archivo.
- `ios::ate` puede añadir los datos en cualquier parte.
- `ios::in` abre un archivo en modo de entrada.
- `ios::out` abre un archivo en modo de salida.
- `ios::trunc` ignora el contenido del archivo (como `ios::out`).
- `ios::binary` abre un archivo para entrada/salida binaria.

- La clase `ios` define una variable enum `io_state`.
  - Cada flujo tiene este flag para comprobar los posibles errores.
  - Posibles valores: `goodbit`, `eofbit`, `badbit` o `failbit`.
  - Para comprobar: `int good()`, `int eof()`, `int bad()` o `int fail()`.
  - Con `int clear()` se borran los bits de error que se hayan activado.
- El operador `!` está sobrecargado en la librería `ios`.
  - E.g. nos informa sobre si la operación `open` tuvo éxito o no.
  - `(!outUsersFile)=true` $\Longleftrightarrow$ (`failbit=1` o `badbit=1`) para `open`.
    - $\nexists$ fichero-r, no tenemos permisos (r o w), no queda espacio (w).
- Otra función sobrecargada de `ios` es `void*` de `cin`.
  - Permite convertir el flujo de entrada (`cin`) en un puntero.
  - *puntero* = *null* $\Longrightarrow$ `cin=0` $\Longleftrightarrow$ (`failbit=1` o `badbit=1`) para `cin`.
    - Es decir, `while cin` es `true` mientras no se activen bits de fallo.
  - Al introducir fin de archivo $\Longrightarrow$ `failbit=1` $\Longrightarrow$ fin while de `cin`.
    - Fin de archivo: sistemas UNIX `<Ctrl+d>`, Windows `<Ctrl+z>`.
  - `<Ctrl+d>` $\Longrightarrow$ `main` termina $\Longrightarrow$ destructor objeto `ofstream`.
    - También se puede cerrar explícita/ con `outUsersFile.close();`

```
int main() {
   ifstream inUsersFile ("users.dat", ios::in);

   if (!inUsersFile) {
      cerr << "File could not be opened" << endl;
      exit (1);
   }

   string name;
   string surname;
   int phone;
   cout << "Name\tSurname\tPhone\n";

   while (inUsersFile >> name >> surname >> phone)
      cout << name << "\t" << surname << "\t" << phone << endl;
}
```

- Para leer datos secuencial/ de un fichero, se leen de ppio. a fin.
- A veces se necesita leer todos los datos secuencial/ varias veces.
- `istream` y `ostream` ofrecen funciones para reposicionar puntero.
  - Un objeto `istream` tiene puntero *get* y, un `ostream`, *put*.
    - Para saber sus posiciones, están las funciones `tellg()` y `tellp()`.
  - En `istream` está la función `seekg` (*seek get*, buscar obtener).
    - E.g. instrucción `inUsersFile.seekg(0);` ubica puntero en pos. 0.
    - El arg. de `seekg` es un entero `long`. Y un 2.º arg. puede ser:
    - `ios::beg` opción por defecto para posiciona/ desde inicio del flujo.
    - `ios::cur` para un posiciona/ relativo a la pos. actual del flujo.
    - `ios::end` para un posiciona/ desde final (hacia atrás) del flujo.
  - En `ostream` está la función `seekp` (*seek put*, buscar poner).
    - Y su uso es similar al descrito para `seekg`, con los mismos posibles args.

```
int option = getUserOption(); // e.g. 1=showNames,
    2=showSurnames, 0=end...

while (option != 0) { // option != end
  while (!inUsersFile.eof()) { // shows the selected field
    inUsersFile >> name >> surname >> phone;
    if (option == 1)
      cout << name << endl;
    if (option == 2)
      cout << surname << endl;
    if (option == 3)
      cout << phone << endl;
  }

  inUsersFile.clear (); // reset eof for next loop
  inUsersFile.seekg (0); // pointer reposition to the beginning
  option = getUserOption(); // get a new request
}
```

- Estos archivos se usan para acceder a registro de forma instantánea.
  - El acceso a registro en un fich. secuencial tiene complejidad $O(n)$.
  - El acceso a registro en un fich. aleatorio tiene complejidad $O(1)$.
- C++ no ofrece una estructura para el manejo de ficheros.
  - Si se quieren usar ficheros con acceso aleatorio, hay que crearlos.
  - Las técnicas a emplear para crear un archivo aleat. son muy variadas.
  - E.g. usar registros de la misma long. fija. $\Longrightarrow$ fácil hallar un registro.

- Se puede escribir un `int` (cuatro bytes) con operador `<<`.
  - E.g. `outputFile << intNumber;`
  - Así se podría escribir desde un dígito hasta 11 (10 dígitos más signo).
    - Cada uno de estos requeriría un solo byte de almacenamiento.
- Otra opción podría ser usando función `write`.
  - E.g. `outputFile.write(reinterpret_cast<const char*> (&intNumber), sizeof (intNumber));`
  - Siempre escribiría la versión binaria de los cuatro bytes del tipo `int`.
  - `write` trata a su primer argumento como un grupo de bytes...
    - ...al ver el objeto en memoria como un puntero a byte (`const char*`).
  - Y desde esa posición, `write` envía el *n.º* de bytes según 2.º parám.
  - `write` necesita que el primer parámetro sea `const char*`
    - Y la expresión `&intNumber` devuelve puntero tipo `int*`.
    - `reinterpret_cast` es usado para convertir tipos de punteros.

```
#ifndef USER_H
#define USER_H
#include <string>
using namespace std;
class User {
public:
   User (string = "", string = "", int = 0);
   void setNumRecord (int);
   int getNumRecord () const;
   void setName (string);
   string getName () const;
   void setSurname (string);
   string getSurname () const;
   void setPhone (int);
   int getPhone () const;
private:
   int numRecord; char name [10]; char surname [15]; int phone;
};
#endif
```

```
#include <string>
#include "User.h"
using namespace std;

User::User (string myName, string mySurname, int myPhone) {
  setName (myName);
  setSurname (mySurname);
  setPhone (myPhone);
}

int User::getNumRecord() const {
  return numRecord;
}
void User::setNumRecord (int record) {
  numRecord = record;
}
string User::getName() const {
  return name;
}
```

```
void User::setName (string myName) { // copy at most 10 char
  int length = myName.size();
  length = (length < 10 ? length : 9);
  myName.copy (name, length);
  name [length] = '\0'; // null character
}
string User::getSurname() const {
  return surname;
}
void User::setSurname (string mySurname) { // copy at most 15
    char
  int length = mySurname.size();
  length = (length < 15 ? length : 14);
  mySurname.copy (surname, length);
  surname [length] = '\0';
}
int User::getPhone() const { return phone; }
void User::setPhone (int myPhone) { phone = myPhone; }
```

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

int main() {
  ofstream outUsersFile ("users.dat", ios::out | ios::binary);

  if (!outUsersFile) { // ofstream could not open file
    cerr << "File could not be opened." << endl;
    exit (1);
  }

  User user; // fill with zeros each data member
  for (int i = 0; i < 10; i++) // write 10 empty records to file
    outUsersFile.write (reinterpret_cast <const char *> (&user),
       sizeof (User));
}
```

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

int main() {
  int numRecord;
  string name;
  string surname;
  int phone;
  fstream outUsersFile ("users.dat", ios::in | ios::out |
      ios::binary); // ios::in will require an existing file
  // Uses: "users.dat", generated in previous example
  if (!outUsersFile) { // fstream could not open file
    cerr << "File could not be opened." << endl;
    exit (1);
  }
  cout << "Enter record number (1 to 10, 0 to end)\n> ";
```

```
  User user;
  cin >> numRecord;
  while (numRecord > 0 && numRecord <= 10) {
    cout << "Enter name, surname and phone\n> ";
    cin >> name;
    cin >> surname;
    cin >> phone;
    user.setNumRecord (numRecord);
    user.setName (name);
    user.setSurname (surname);
    user.setPhone (phone);
    outUsersFile.seekp ((user.getNumRecord() - 1 ) *
       sizeof (User));
    outUsersFile.write (reinterpret_cast <const char *> (&user),
       sizeof (User));
    cout << "Enter record number (1 to 10, 0 to end)\n> ";
    cin >> numRecord;
  }
}
```

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include "User.h"
using namespace std;
void displayRecord (ostream &, const User &);

int main() {
  ifstream inUsersFile ("users.dat", ios::in | ios::binary);
  // Uses: "users.dat", generated in previous example

  if (!inUsersFile) { // fstream could not open file
    cerr << "File could not be opened." << endl;
    exit (1);
  }
  cout << left << setw (10) << "Record" << setw (11)
    << "Name" << setw (16) << "Surname" << left
    << setw (10) << right << "Phone" << endl;
```

```
  User user;
  inUsersFile.read (reinterpret_cast <char *>(&user),
    sizeof (User));

  while (inUsersFile && !inUsersFile.eof()) {
    if (user.getNumRecord() != 0)
       displayRecord (cout, user);

    inUsersFile.read (reinterpret_cast <char *>(&user),
      sizeof (User));
  }
}

void displayRecord (ostream &output, const User &record) {
   output << left << setw (10) << record.getNumRecord ()
      << setw (11) << record.getName()
      << setw (16) << record.getSurname()
      << setw (10) << right << record.getPhone() << endl;
}
```

10. Manejo de ficheros

Julio Vega

julio.vega@urjc.es