Universidad de Murcia

Material didáctico para la asignatura Sistemas Inteligentes de 3º de Grado en Informática

AUTORES:

- José Manuel Cadenas Figueredo
- María del Carmen Garrido Carrera
- Raquel Martínez España
- Santiago Paredes Moreno

Capítulo 4

Entrada/Salida mediante Ficheros

4.1 Introducción

En este tema vamos a tratar la lectura y escritura de datos a través de ficheros, pero para entender cómo funcionan los operadores de lectura y escritura vamos a repasar de forma breve los operadores para lectura y escritura de datos por los dispositivos de entrada y salida estándar.

Para la entrada y salida de datos C++ utiliza los stream o flujos, que no son más que un canal por el cual fluyen los datos para llegar a su destino. Tradicionalmente se ha considerado que la entrada y salida estándar son el teclado y la pantalla respectivamente. De ahí que de forma estándar C++ tenga asociado al objeto `cin` la entrada estándar, es decir, la entrada de datos a partir de teclado. Para la salida estándar C++, utiliza el objeto `cout`. Además, la salida estándar de los mensajes de error lo tiene asociado al objeto `cerr`, que por defecto muestra la información por pantalla. Si comparamos estos tres objetos con el lenguaje C, veremos que los tres objetos para la entrada y salida de datos en C++ se corresponden con los objetos `stdin`, `stout` y `stderr` del lenguaje C.

Para trabajar con la entrada y salida estándar, se debe incluir la librería de C++ `iostream`. Una vez incluida la librería se puede comenzar a hacer uso de los objetos mencionados anteriormente (`cin`, `cout`, `cerr`). Veamos un ejemplo de cómo utilizarlos:

```
1 // Se incluye la librería del estándar
2 #include <iostream>
3 int main(){
4
```

```
5   // El mensaje se imprime por pantalla.
6     std::cout << "Estoy utilizando la escritura por pantalla";
7
8   // Se muestra el error
9     std::cerr << "Esto es un error";
10
11  // Se espera leer datos de teclado.
12    std::cin.get();
13
14    return 0;
15 }
```

Un aspecto importante a recordar antes de introducirnos en la entrada y salida de datos a y desde ficheros, son los operadores de direccionamiento. Estos operadores son << y >> y se encargan de direccionar como su nombre indica el flujo de datos hacia un stream concreto. Para las salidas de datos a un stream de salida, como puede ser a fichero o a pantalla, se utiliza el operador <<. Para las entradas de datos recogidas en un flujo de entrada, por ejemplo teclado o un fichero, se utiliza el operador >>. Para ver cómo funcionan vamos a ver el siguiente ejemplo:

```
1  // Ejemplo de operadores de direccion
2  #include<iostream>
3  int main(){
4
5      char fichero[50];
6
7      //Se pide por pantalla el fichero
8      std::cout << "Que fichero quiere leer? ";
9
10     //Se lee lo que escribe el usuario por teclado
11     std::cin >> fichero;
12
13     //Se muestra por pantalla el mensaje y el contenido de la variable
           fichero
14     std::cout << "El fichero que quiere leer es " << fichero << " ";
15
16     system("pause");
17
18     return 0;
19 }
```

Además, es posible en una misma línea de código leer o escribir varios elementos a la vez, simplemente nombrando una vez al stream. También podemos enviar al stream un `endl` para provocar un salto de línea. Un ejemplo sería:

```
1  // Ejemplo de operadores de dirección
2  #include<iostream>
3  using namespace std;
4  int main(){
5      char nombre[50];
6      char apellidos[50];
7      char direccion[50];
8      char instruccion[50] = "separados por espacios ";
9
10     //Se muestra el mensaje por pantalla
11     cout << "Escribe tu nombre, apellidos y direccion " << instruccion;
12
13     //Se lee lo que escribe el usuario por teclado
14     cin >> nombre >> apellidos >> direccion ;
15
16     return 0;
17 }
```

4.2 Manejo de ficheros

Una vez que hemos repasado los streams de entrada y salida estándar y los operadores de direccionamiento, vamos a explicar cómo podemos leer y escribir datos de un fichero. Una característica de los streams estándar es que son creados y abiertos de forma automática, sin embargo, para trabajar con ficheros, los flujos o streams deben ser creados y abiertos antes de comenzar a trabajar con ellos.

En el lenguaje C++ para manipular ficheros debemos hacer uso de las clases `fstream`, `ifstream` y `ofstream`. Estas tres clases nos proporcionan la lectura y escritura, sólo la lectura y sólo la escritura de un fichero respectivamente.

La sintaxis para utilizar estas clases es la siguiente:

- Para lectura y escritura: Clase `fstream`
  - fstream();
  - fstream(const char * name, int mode, int = filebuf::openprot);

  - fstream(int);
  - fstream(int _f, char*, int);

- Sólo para lectura : Clase `ifstream`
  - ifstream();
  - ifstream(const char * name, int mode, int = filebuf::openprot);
  - ifstream(int);
  - ifstream(int _f, char*, int);

- Sólo para escritura : Clase `ofstream`
  - ofstream();
  - ofstream(const char * name, int mode, int = filebuf::openprot);
  - ofstream(int);
  - ofstream(int _f, char*, int);

Como podemos apreciar, cada clase tiene cuatro constructores para crear el objeto. Dependiendo del utilizado debemos de especificar unos parámetros u otros. El primer constructor crea un objeto que aún no tiene asociado un fichero, por ello para utilizar este constructor tendremos que utilizar el método `open("nombreFichero")`, para establecer la conexión entre el nombre del fichero y el stream. El segundo constructor lo que hace es crear un objeto que se asocia al fichero en disco. Al utilizar este constructor el fichero ya queda abierto y asociado al stream correspondiente, donde el primer parámetro char * queda apuntando a la cadena de caracteres con el nombre del fichero. El tercer constructor crea un stream pero como parámetro hay que pasarle el identificador de un fichero. Por último, el cuarto constructor crea un stream utilizando el identificador del fichero y pasándole un buffer y el tamaño de éste.

Un aspecto a comentar son los diferentes parámetros de apertura de un fichero:

- ios::app Para añadir información.
- ios::ate Para colocar el puntero a final de fichero.

- ios::in Para realizar una lectura del fichero. Esta opción está por defecto al usar un objeto de la clase ifstream.
- ios::out Para realizar la escritura en un fichero. Esta es la opción por defecto al usar un objeto de la clase ofstream.
- ios::nocreate Esta operación no crea el fichero si éste no existe.
- ios::noreplace Esta operación crea un fichero si existe uno con el mismo nombre.
- ios::trunc Esta operación crea un fichero y si existe uno con el mismo nombre lo borra.
- ios::binary Para lectura y escritura de ficheros binarios.

Como hemos comentado, los streams no son más que objetos de cierta clase y esta clase nos ofrece una serie de métodos para comprobar y consultar las condiciones y el estado en el que se encuentran. En la siguiente tabla podemos consultar algunos de los métodos más utilizados:

Método	Descripción
bad	Devuelve true si ha ocurrido un error
clear	Sirve para limpiar los flag de estado
close	Cierra el stream
eof	Indica si se ha alcanzado el final de un fichero devolviendo true
fail	Devuelve true si ha ocurrido un error
fill	Establecer manipulador de carácter de relleno
flush	Vaciar el buffer de un stream
gcount	Indica el número de caracteres leidos en la última operación de entrada
get	Método para ir leyendo carácter a carácter
getline	Método para leer una línea completa, hasta un \n ó \r
ignore	Método para leer y descartar caracteres
open	Método para abrir un stream tanto de entrada como de salida
precision	Manipula la precisión del stream
put	Método para escribir caracteres
read	Método para leer datos de un stream hacia un buffer
seekg	Método para acceder de forma aleatoria sobre un stream de entrada
seekp	Método para acceder de forma aleatoria sobre un stream de salida
tellg	Método que devuelve el puntero del stream de entrada
tellp	Método que devuelve el puntero del stream de salida
write	Método para escribir datos desde un buffer hacia un stream

Para visualizar cómo funcionan los métodos que hemos comentado y concretamente cómo se lee y se escribe en un fichero, vamos a mostrar varios ejemplos, donde encima de cada línea se indica mediante un comentario cuál es la función que realiza el método utilizado.

4.2.1 Ejemplo1: Escribiendo en un fichero

```
1 #include<iostream>
2 #include<fstream>
3 using namespace std;
4 int main(){
5     // se crea el objeto de la clase ofstream
6     ofstream mifichero;
7
8     // se abre el fichero
9     mifichero.open("ejemplo.txt");
10
11    // se escribe en el fichero
12    mifichero << "Escribo la primera línea" << endl;
13    mifichero << "Escribo la segunda línea" << endl;
14
15    // se cierra el fichero
16    mifichero.close();
17
18    return 0;
19 }
```

4.2.2 Ejemplo 1.1: Escribiendo en un fichero utilizando otro constructor

```
1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4 int main()
5 {
6     // se crea el objeto de la clase ofstream y se abre el fichero,
7     // con la opción de añadir en el fichero ya existente
8     ofstream mifichero("ejemplo.txt",ios::app);
9
10    // se comprueba si hay algún error
11    if (mifichero.bad()){
```

```
12      cout << "Error al abrir el archivo";
13      return -1;
14     }
15     // se escribe en el fichero
16     mifichero << "Escribo la primera línea" << endl;
17     mifichero << "Escribo la segunda línea" << endl;
18
19     // se cierra el fichero
20     mifichero.close();
21
22     return 0;
23 }
```

4.2.3 Ejemplo 2: Leyendo de un fichero línea a línea

```
1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4 int main(){
5     // se crea un objeto de la clase ifstream y abro el fichero
6     ifstream mifichero("ejemplo.txt");
7     char aux[128];
8
9     // se comprueba que todo es correcto
10    if(mifichero.fail()){
11        cerr << "Se ha producido un error al abrir el fichero" << endl;
12        return -1;
13    }
14
15    // se lee el fichero hasta final de fichero
16    while(!mifichero.eof()){
17        // se lee línea a línea
18        mifichero.getline(aux,sizeof(aux));
19        cout << aux << endl;
20    }
21    // se cierra el fichero
22    mifichero.close();
23
24    system("pause");
25
26    return 0;
27 }
```

4.2.4 Ejemplo 3: Leyendo un fichero y copiándolo en otro, utilizando las funciones `write` y `getline`

```
1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4
5 int main(){
6     // se crea un objeto de la clase ifstream y abre el fichero
7     ifstream mifichero("ejemploAleer.txt");
8     // se crea un objeto de la clase ofstream y abre el fichero
9     ofstream mifichero2("ejemploAcopiar.txt");
10
11    char aux[128] = "";
12
13    // se comprueba que todo es correcto
14    if(mifichero.fail())
15        cerr << "Se ha producido un error al abrir el fichero" << endl;
16    else{
17        // se lee el fichero hasta final de fichero
18        while(!mifichero.eof()){
19              // se lee línea a línea
20              mifichero.getline(aux,sizeof(aux),'\n');
21
22              // Conforme se lee se escribe
23              mifichero2.write(aux,sizeof(aux));
24        }
25        // se cierran los ficheros
26        mifichero.close();
27        mifichero2.close();
28    }
29
30    return 0;
31 }
```

4.2.5 Ejemplo 3.1: Leyendo un fichero y copiándolo en otro, utilizando los operadores de dirección

```
1 #include<iostream>
2 #include<fstream>
3 using namespace std;
4
```

```
5  int main(){
6      // se crea un objeto de la clase ifstream y abre el fichero
7      ifstream mifichero("ejemploAleer.txt");
8      // se crea un objeto de la clase ofstream y abre el fichero
9      ofstream mifichero2("ejemploAcopiar.txt");
10
11     char aux[128] = "";
12
13     // se comprueba que todo es correcto
14     if (mifichero.fail())
15         cerr << "Se ha producido un error al abrir el fichero" << endl;
16     else{
17         // se lee el fichero hasta final de fichero
18         while(!mifichero.eof()){
19             // se lee
20             mifichero >> aux;
21             // se escribe
22             mifichero2 << aux << endl;
23         }
24         // se cierran los ficheros
25         mifichero.close();
26         mifichero2.close();
27     }
28     return 0;
29 }
```