Apéndice A

streams en C++

La gestión de la entrada/salida (*E/S*) en *C++* se implementa mediante un conjunto de clases conocidas de forma genérica como biblioteca de `streams`. Se trata de una estructura jerarquizada de clases que puede ser modificada y expandida por el usuario para incorporar nuevas clases. La figura A.1 muestra la organización jerárquica de las clases `stream`. En la versión 2.96 de *GNU gcc*, los ficheros cabecera con la definición de estas clases se encuentran en `/usr/include/g++-3`.

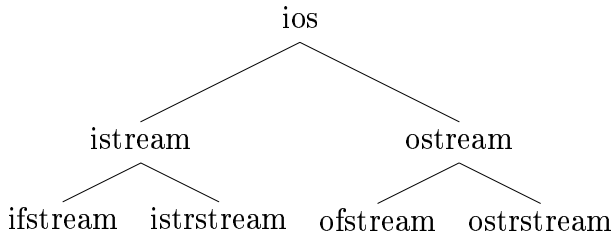

Figura A.1: Jerarquía de clases `stream`

A.1. Clase `ios`

La clase `ios`, que se define en el fichero `streambuf.h`, es la clase base para definir la jerarquía que permite representar a cualquier dispositivo de E/S in *C++*. Los objetos instanciados de la clase `ios` se denominan streams, y sus principales funciones son:

- Controlar la conexión con los buffers donde se insertan y/o extraen datos de un dispositivo. Un objeto stream está asociado a un objeto `streambuf` que le provee de un buffer.

- Mantener la información sobre el estado de un dispositivo. Esto permitirá inspeccionar las condiciones de error en las operaciones de E/S.
- Controlar la forma en que los caracteres se insertan y se extraen del buffer. Esto es, dar el formato a lo datos.

A.1.1. Modos de apertura

La figura A.2 contiene la definición del tipo enumerado `open_mode` que representa los modos de apertura de un dispositivo.

```
1     class ios  {
2         // ...
3         enum open_mode {
4              in        = 0x01,
5              out       = 0x02,
6              ate       = 0x04,
7              app       = 0x08,
8              trunc     = 0x10,
9              nocreate  = 0x20,
10             noreplace = 0x40,
11             binary    = 0x80
12        };
13        // ...
14    };
15
```

Figura A.2: Modos de apertura de un dispositivo

ios::in Abre para lectura

ios::out Abre para escritura.

ios::ate El primer byte que se escribe se añade al final, pero los bytes siguientes se escriben en la posición actual.

ios::app Los nuevos bytes se escriben al final.

ios:trunc Si el fichero existe se destruye su contenido. Está implícito en el modo `ios::out` cuando no va acompañado de `ios::in`, `ios::ate` o `ios::app`.

ios::nocreate La función falla si no existe el fichero.

ios::noreplace La función falla si el fichero existe.

ios::binary Abrir en modo binario.

A.1.2. Estados

La figura A.3 muestra el tipo enumerado `io_state` que representa los estados de un stream, y los métodos que permiten inspeccionar dicho estado.

```
1     class ios {
2       // ...
3       public:
4       enum io_state {
5            goodbit = 0x0,
6            eofbit  = 0x1,
7            failbit = 0x2,
8            badbit  = 0x4
9       };
10
11      int good() const { return _state == 0; }
12      int eof() const { return _state & ios::eofbit; }
13      int fail() const { return _state & (ios::badbit|ios::failbit); }
14      int bad() const { return _state & ios::badbit; }
15      iostate rdstate() const { return _state; }
16      void clear(iostate state = 0);
17      // ...
18    };
19
```

Figura A.3: Representación del estado en la clase `ios`

El estado se almacena como un campo de bits en el atributo `_state`, y puede tomar los valores:

ios::good No hay error (bits de error a cero).

ios::eof Se encontró el final de fichero.

ios::fail Posible error recuperable, de formato o conversión.

ios::bad Error irrecuperable.

Los siguientes métodos permiten acceder al estado de un stream:

int good() Devuelve un valor distinto de cero si todos los bits de error están a cero.

int eof() Devuelve un valor distinto de cero si en la operación de lectura previa se encontró el final de fichero.

int fail() Devuelve un valor distinto de cero si en la operación previa ocurrió cualquier error recuperable.

int bad() Devuelve un valor distinto de cero si sucedió un error irrecuperable.

iostate rdstate() Devuelve el contenido de la variable de estado.

void clear() Toma como argumento un valor que escribe en la variable de estado. Por defecto es el valor `ios::good`.

A.1.3. Formato

La clase `ios` controla el formato de representación de los datos para los tipos básicos. Para ello utiliza las banderas (*flags*) indicados en la figura A.4. **Ver descripción en [1].**

A.2. Clase `ostream`

La clase `ostream` implementa la funcionalidad necesaria para acceder de forma secuencial o aleatoria a un stream abierto para escritura. Hereda las características de la clase ios, y necesita estar conectado a un objeto `streambuf`. La definición se encuentra en el fichero cabecera `iostream.h`.

La funcionalidad propia se implementa mediante las funciones mostradas en la figura A.5.

ostream() El constructor que crea un objeto sin inicializar.

ostream(streambuf* sb, ostream* tied=NULL) Se crea un objeto conectándolo al buffer apuntado por `sb`. De forma opcional, se le puede ligar un objeto `ostream`.

ostream& flush() Fuerza una sincronización del buffer con el dispositivo.

ostream& put(char c) Escribe el carácter `c` en el buffer.

ostream& write(const void *s, streamsize n) En sus diferentes variantes, el método write escribe `n` caracteres en el buffer sacandolos de la dirección de memoria dada en el parámetro `s`.

ostream& seekp(streampos) Posiciona el dispositivo de escritura en la posición especificada mediante un parámetro de tipo `long`.

ostream& seekp(streamoff, seekdir) Desplaza el dispositivo de escritura el número de bytes dados en el primer parámetro de tipo `long` respecto al punto de referencia dado en el segundo argumento. Los posibles valores del punto de referencia se definen en la clase `ios`. (Ver figura A.6.)

ios::beg Comienzo del stream.

ios::cur Posición actual del dispositivo.

ios::end Final del stream.

ostreampos tellp() Devuelve la posición actual del dispositivo.

ostream& operator<<(T) El operador de reenvío está sobrecargado para manejar los tipos elementales.

A.2.1. Clase `istream`

La figura A.7 muestra la clase `istream` que implementa la funcionalidad de acceso a stream de entrada. Hereda de la clase `ios` y añade los siguientes métodos específicos.

istream() El constructor que crea un objeto sin inicializar.

istream(streambuf* sb, ostream* tied=NULL) Se crea un objeto conectándolo al buffer apuntado por `sb`. De forma opcional, se le puede ligar un objeto `ostream`.

istream& get(...) Este método extrae un dato del stream y lo almacena en la dirección pasada como argumento.

istream& getline(...) Este método extrae una cadena de caracteres del stream y lo almacena en la dirección pasada como argumento primer parámetro. El segundo parámetro indica el tamaño en bytes de la zona de memoria apuntada por `ptr`, de forma que el número máximo de caracteres leidos es `len-1`. El último carácter se pone a '
0'. Si se encuentra el carácter `delim` se extrae pero no se almacena.

istream& read(void *ptr, streamsize n) Este método, en sus múltiples variantes, lee un bloque de `n` caracteres en modo binario y los guarda en la zona de memoria apuntada por `ptr`.

int peek() Inspecciona el siguiente carácter de la entrada sin extraerlo.

istream& ignore(int n=1, int delim = EOF) Este método lee e ignora los siguientes `n` caracteres de la entrada, o hasta que encuentra el carácter `delim`.

_IO_size_t gcount() Devuelve el número de caracteres leidos en la última operación.

istream& seekg(streampos) Posiciona el dispositivo de lectura en la posición dada como parámetro.

istream& seekg(streamoff, seekdir) Desplaza el dispositivo de lectura el número de bytes indicado por el primer parámetro respecto al punto de referencia indicado en el segundo parámetro. (Ver figura A.6).

streampos tellg() Devuelve la posición del dispositivo de lectura.

istream& putback(char ch) Reintroduce el caracter especificado en el stream.

istream& operator>>(T&) Se define la sobrecarga del operador de extracción para los tipos elementales.

A.2.2. Clase `iostream`

Esta clase hereda de las clases `istream` y `ostream` las funcionalidades para manejar un stream en modo lectura/escritura. La definición se encuentra en el fichero cabecera `iostream.h`. Ver figura A.8.

A.2.3. Clase `fstreambase`

La clase `fstreambase` que se muestra en la figura A.9 implementa la funcionalidad básica para trabajar con ficheros almacenados en dispositivos de almacenamiento secundario. Los ficheros son un tipo particular de objetos stream, por lo que la clase `fstreambase` hereda de la clase `ios`.

fstreambase() Instancia un objeto sin inicializar.

fstreambase(int fd) Instancia un objeto que asigna al dispositivo identificado en el sistema operativo mediante el manejador `fd`.

fstreambase(const char *name, int mode, int prot=0664) Instancia un objeto y lo asigna al fichero con nombre `name`. Abriéndolo en el modo especificado en el segundo parámetro (ver figura A.2), y asignando los permisos indicados en el tercer parámetro.

void close() Cierra el fichero.

void open(const char *name, int mode, int prot=0664) Abre un fichero asignándolo a un objeto creado previamente.

int is_open() Devuelve un valor distinto de cero si el objeto está asignado a un fichero abierto.

void setbuf(char *ptr, int len) Asigna un buffer para trabajar con el fichero.

A.2.4. Clase `ofstream`

La clase `ofstream` hereda de `fstreambase` y de `ostream` las funcionalidades para manejar ficheros abiertos en modo de escritura. La figura A.10 contiene la especificación que se encuentra en `fstream.h`.

ofstream() El constructor se limita a invocar al constructor de la clase `fstreambase` con los parámetros adecuados.

void open(const char *name, int mode=ios::out, int prot=0664) Invoca al método `open` de la clase `fstreambase` con los parámetros adecuados.

A.2.5. Clase `ifstream`

La clase `ifstream` hereda de `fstreambase` y de `istream` las funcionalidades para manejar ficheros abiertos en modo de lectura. La figura A.11 contiene la especificación que se encuentra en `fstream.h`. Tanto el método `open()` como el constructor se implementan invocando a los correspondente métodos de la clase `fstreambase` con los parámetros adecuados.

A.2.6. Clase `fstream`

La clase `fstream` hereda de `fstreambase` y de `iostream` las funcionalidades para manejar ficheros abiertos en cualquier modo. La figura A.12 contiene la especificación que se encuentra en `fstream.h`. Tanto el método `open()` como el constructor se implementan invocando a los correspondente métodos de la clase `fstreambase` con los parámetros adecuados.

```
1     class ios {
2         // ...
3         enum {
4              skipws     = 0x0001,
5              left       = 0x0002,
6              right      = 0x0004,
7              internal   = 0x0008,
8              dec        = 0x0010,
9              oct        = 0x0020,
10             hex        = 0x0040,
11             showbase   = 0x0080,
12             showpoint  = 0x0100,
13             uppercase  = 0x0200,
14             showpos    = 0x0400,
15             scientific = 0x0800,
16             fixed      = 0x1000,
17             unitbuf    = 0x2000,
18             stdio      = 0x4000
19          };
20         enum {                          // Masks.
21               basefield   = dec+oct+hex,
22               floatfield  = scientific+fixed,
23               adjustfield = left+right+internal
24         };
25
26         inline long flags() const;
27         inline long flags(long _lf);
28
29         // ...
30    };
31
```

Figura A.4: Representación del formato en la clase`ios`

```
1       class ostream : virtual public ios {
2         // ...
3         public:
4         ostream() { }
5         ostream(streambuf* sb, ostream* tied=NULL);
6         ostream& flush();
7         ostream& put(char c);
8         ostream& write(const char *s, streamsize n);
9         ostream& write(const unsigned char *s, streamsize n);
10        ostream& write(const signed char *s, streamsize n);
11        ostream& write(const void *s, streamsize n);
12        ostream& seekp(streampos);
13        ostream& seekp(streamoff, _seek_dir);
14        streampos tellp();
15
16        ostream& operator<<(char c);
17        ostream& operator<<(unsigned char c);
18        ostream& operator<<(signed char c);
19        ostream& operator<<(const char *s);
20        ostream& operator<<(const unsigned char *s);
21        ostream& operator<<(const signed char *s);
22        ostream& operator<<(const void *p);
23        ostream& operator<<(int n);
24        ostream& operator<<(unsigned int n);
25        // ...
26    };
27
```

Figura A.5: Clase `ostream`

```
1     class ios  {
2         // ...
3         enum seek_dir {beg, cur, end};
4         typedef enum seek_dir seekdir;
5         // ...
6     };
7
```

Figura A.6: Punto de referencia de un posicionamiento

```
1   class istream : virtual public ios
2   {
3     //...
4     public:
5       istream();
6       istream(streambuf* sb, ostream*tied=NULL);
7       istream& get(char& c);
8       istream& get(unsigned char& c);
9       istream& getline(char* ptr, int len, char delim = '\n');
10      istream& getline(unsigned char* ptr, int len, char delim = '\n');
11      istream& get(signed char& c);
12      istream& get(signed char* ptr, int len, char delim = '\n');
13      istream& getline(signed char* ptr, int len, char delim = '\n');
14      istream& read(char *ptr, streamsize n);
15      istream& read(unsigned char *ptr, streamsize n);
16      istream& read(signed char *ptr, streamsize n);
17      istream& read(void *ptr, streamsize n);
18      istream& get(streambuf& sb, char delim = '\n');
19      int peek();
20      _IO_size_t gcount();
21      istream& ignore(int n=1, int delim = EOF);
22      istream& seekg(streampos);
23      istream& seekg(streamoff, seekdir);
24      streampos tellg();
25      istream& putback(char ch);
26      istream& operator>>(char*);
27      istream& operator>>(unsigned char* p);
28      istream& operator>>(signed char*p);
29      istream& operator>>(char& c);
30      istream& operator>>(unsigned char& c);
31      istream& operator>>(signed char& c);
32      istream& operator>>(int&);
33      istream& operator>>(long&);
34      istream& operator>>(short&);
35      istream& operator>>(unsigned int&);
36      istream& operator>>(unsigned long&);
37      istream& operator>>(unsigned short&);
38      istream& operator>>(bool&);
39      istream& operator>>(float&);
40      istream& operator>>(double&);
41      istream& operator>>(long double&);
42      istream& operator>>(streambuf*);
43    //...
44  };
45
```

Figura A.7: Clase `istream`

```
1     class iostream : public istream, public ostream
2     {
3       public:
4         iostream() { }
5         iostream(streambuf* sb, ostream*tied=NULL);
6     };
7
```

Figura A.8: Clase `iostream`

```
1     class fstreambase : virtual public ios {
2       //...
3       public:
4         fstreambase();
5         fstreambase(int fd);
6         fstreambase(const char *name, int mode, int prot=0664);
7         void close();
8         void open(const char *name, int mode, int prot=0664);
9         int is_open() const;
10        void setbuf(char *ptr, int len);
11      //...
12    };
13
```

Figura A.9: Clase `fstreambase`

```
1     class ofstream : public fstreambase, public ostream {
2       public:
3         ofstream() : fstreambase() { }
4         ofstream(int fd) : fstreambase(fd) { }
5         ofstream(const char *name, int mode=ios::out, int prot=0664)
6           : fstreambase(name, mode | ios::out, prot) { }
7         void open(const char *name, int mode=ios::out, int prot=0664) {
8            fstreambase::open(name, mode | ios::out, prot);
9         }
10    };
11
```

Figura A.10: Clase `ofstream`

```
1   class ifstream : public fstreambase, public istream {
2     public:
3       ifstream() : fstreambase() { }
4       ifstream(int fd) : fstreambase(fd) { }
5       ifstream(const char *name, int mode=ios::in, int prot=0664)
6         : fstreambase(name, mode | ios::in, prot) { }
7       void open(const char *name, int mode=ios::in, int prot=0664) {
8           fstreambase::open(name, mode | ios::in, prot);
9       }
10  };
11
```

Figura A.11: Clase `ifstream`

```
1   class fstream : public fstreambase, public iostream {
2     public:
3       fstream() : fstreambase() { }
4       fstream(int fd) : fstreambase(fd) { }
5       fstream(const char *name, int mode, int prot=0664)
6         : fstreambase(name, mode, prot) { }
7       void open(const char *name, int mode, int prot=0664) {
8           fstreambase::open(name, mode, prot);
9       }
10  };
11
```

Figura A.12: Clase `fstream`

Bibliografía

[1] Francisco Javier Ceballos. *Programación Orientada a Objetos con C++*. Rama, second edition, 1997. ISBN 84-7897-268-4.

[2] Michael J. Folk, Bill Zoellick, and Greg Riccardi. *File Structures. An Object-Oriented Approach with C++*. Addison Wesley, 1998. ISBN 0-201-87401-6.

[3] Valerie Illingwoth, editor. *Diccionario de Informática*. Díaz de Santos, second edition, 1990. ISBN 84-7978-068-1.

[4] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, second edition, 1991. ISBN 0-201-53992-6.