Programación Orientada a Objetos

Grado en Ingeniería Informática

Biblioteca de flujos

Departamento de Ingeniería Informática
Universidad de Cádiz

ver 0.1

Indice

1 Introducción

2 Iostream

3 Inserción/extracción

4 Formato

5 Sobrecarga

6 Ejemplos

7 Estado de los Flujos

8 Flujo de Ficheros

9 Bibliografía

Sección 1 | Introducción

Introducción

Biblioteca de Flujos

- El mecanismo de clases de C++ permite crear un sistema consistente y ampliable para los mecanismos de entrada/salida.
- Este sistema se conoce como **biblioteca de flujos**.
- Flujo de datos **(stream)**: Corriente de bytes que actúa como fuente o destino de datos según sea de entrada o salida.
- Estas clases sirven como mecanismo para manejar los tipos básicos, pero además, se pueden modificar **ampliándolos** para incorporar los tipos de datos **definidos por el usuario**.

Introducción

Comparación con C

Example

```c
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radixes: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

http://www.cplusplus.com/reference/cstdio/printf/

Introducción

Comparación con C

- Mejor comprobación de los tipos de datos de E/S.

  El tipo del objeto es conocido en C++ de forma estática por el compilador, en lugar de tener que comprobar de forma dinámica los campos **%** como sucedía en C.

- Tratamiento uniforme de todos los tipos de datos, incluyendo los definidos por los usuarios.

  En el lenguaje C sería caótico si todo el mundo se pusiera a añadir de forma simultánea nuevos campos **%** incompatibles.

- Capacidad de extensión.

  El sistema de entrada/salida de C++ es un sistema de clases. Esto significa que un usuario puede definir algo nuevo que parezca y se comporte como **streams.**

Streams

Flujos

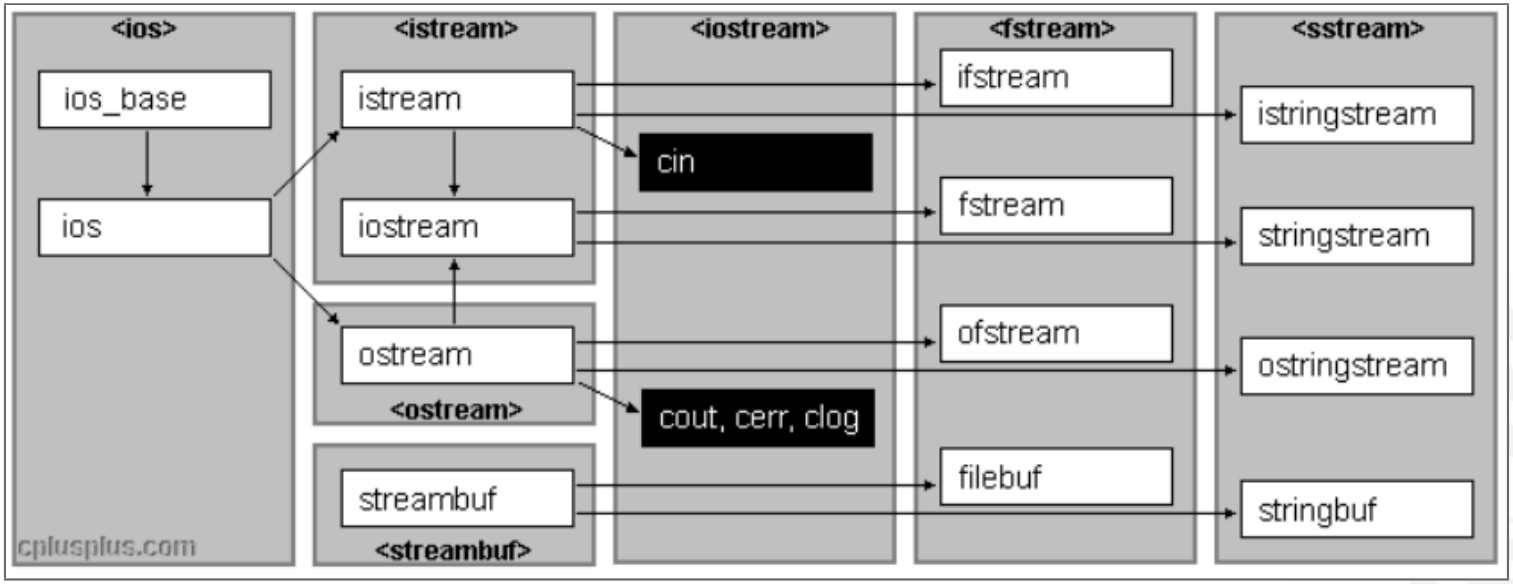
C++ da a los **streams** un significado más abstracto.

- Un programa C++ visualiza la entrada o la salida como un flujo de datos. En la entrada un programa extrae bytes de un flujo de entrada, y en la salida un programa inserta bytes en un flujo de salida.
- Este enfoque permite que un programa C++ trate la entrada de un teclado de igual forma que se trata la entrada desde un fichero.
- **Los flujos necesitan para funcionar dos conexiones, una en cada extremo**
- Tratamiento de la información en la transferencia:
  - Binaria o sin formato.
  - Entrada/salida con formato.

Sección 2 | Iostream

Iostream

Jerarquía de clases de flujos



Sobrecarga de operadores y E/S en C++ - Antonio LaTorre de la Fuente

Iostream

Información

- La biblioteca Iostream es la biblioteca de entrada/salida estándar en C++
- La biblioteca Iostream emplea para las operaciones de entrada/salida **búferes** (buffers).

  Recordad que un búfer es un bloque de memoria que se emplea como almacenamiento temporal o intermedio para la transferencia de información de un dispositivo a un programa o viceversa. Típicamente, dispositivos como unidades de disco transfieren información en bloques de 512 B, 1 kB, o incluso mayores
- El sistema de entrada/salida en C++, está definido para trabajar con diferentes dispositivos. Cada dispositivo se convierte en un **dispositivo lógico** y se denomina flujo (stream).

Iostream

Flujos predefinidos

Los siguientes flujos u objetos se abren de forma automática cuando se ejecuta un programa C++ y se ha incluido el fichero de cabecera iostream.

- **cin** Flujo de entrada estándar.
- **cout** Flujo de salida estándar.
- **cerr** Flujo de error estándar que se emplea para visualizar mensajes de error.

  Las salidas hacia el objeto cerr no se almacena en el búfer. Esto significa que cada inserción de flujo de cerr causa que su salida aparezca inmediatamente; esto es adecuado para notificar al usuario algún error en forma inmediata.
- **clog** Flujo de error estándar a través del búfer; este flujo es más adecuado para grandes cantidades de error.

  Las salidas de clog se almacenan en el búfer. Esto significa que cada inserción a clog puede causar que su salida se conserve en un búfer hasta que éste se llene o vacíe.

Iostream

Ficheros de cabecera

La biblioteca Iostream es francamente potente, está formada por unas 250 funciones y por aproximadamente 20 clases. Esta biblioteca está compuesta por varios ficheros de cabecera, debido a su gran tamaño. y son:

- **iostream** Si se realiza cualquier operación de entrada/salida se necesita utilizar este fichero. Contiene los objetos cin, cout, cerr y clog. Manipula operaciones de entrada/salida con o sin formato, y contiene los manipuladores más comunes.
- **fstream** Si se va a abrir, posicionar y cerrar ficheros.
- **sstream** Se tiene que formatear una cadena (string).
- **iomanip** Si se están utilizando manipuladores, o creando unos propios, para adecuar el formato de entrada/salida a unas necesidades particulares.

Sección 3 | Inserción/extracción

El operador de inserción <<

- El operador << se denomina **operador de inserción** en lugar de operador de desplazamiento de bits a la izquierda.
- **Está sobrecargado** para reconocer todos los tipos básicos de C++ y string.
- **Se puede encadenar**, y los valores del operador se visualizan en el orden que se tiene en la línea de izquierda a derecha.
- No realiza un salto de línea, esto hay que hacerlo de forma explícita, bien al estilo C mediante el carácter \n , bien al estilo C++ con el manipulador endl.

  El manipulador endl tiene una ventaja sobre el carácter \n, ya que además de insertar una nueva línea en el flujo, también vacía el búfer de salida. Por lo tanto es equivalente a utilizar \n seguido de flush().

ostream: put() y write()

La clase ostream proporciona el método **put()** para visualizar caracteres

ostream& put(char);

```
#include <iostream>
using namespaces std;
int main (void)
{
cout.put('F').put('R').put('A').put('N') << endl;
cout.put(3) << endl;
cout.put(77).put(67).put(82).put(85).put(90) << endl;
return 0;
}
```

ostream: put() y write()

El método **write()** para visualizar cadenas.

ostream& write (const signed char *, int);
ostream& write (const unsigned char *, int);

El operador de extracción >>

El operador >> se llama en este contexto **operador de extracción**, y es el opuesto del operador de inserción **<<**; su misión es leer datos de un flujo de entrada.

- Este operador se puede utilizar en cascada también `cint >> a >> b >> c;`
- Los espacios no significativos (espacios, tabuladores, retornos de carro) no se tendrán en cuenta.
- A diferencia de **scanf()**, al operador de extracción no se le debe especificar la dirección de la variable.
- El operador de entrada **>>**, al igual que ocurre con **scanf()**, tiene un inconveniente que lo hace inadecuado cuando se trata de leer cadenas con espacios en blanco en su interior, ya que solo lee una cadena hasta que encuentra la primera ocurrencia del espacio en blanco. El carácter nulo que marca el final de las cadenas en C es añadido al final de la cadena que se lea.

istream: get() y getline()

La clase istream contiene la función miembro **get()** para la introducción de cadenas completas de caracteres

istream& get (char *cad, streamsize n, char fin='\n');

- **cad** es una variable de cadena de caracteres
- **n** es la longitud máxima de la cadena
- **fin** es el delimitador, si no se suministra es el carácter nueva-linea. Este carácter no se se guarda, y se añade siempre al final un carácter nulo '\0' .

istream: get() y getline()

¿Por qué no funciona?

```
#include <iostream>
using namespace std;
int main (void)
{
    char c1[80];
    char c2[4];
    cout << "Introduce una cadena:" << endl;
    cin.get (c1, 80);
    cout << "Introduce una cadena:" << "\n";
    cin.get(c2,4);
    cout << "\n" << c1 << "\n" << c2 << "\n";
}
```

istream: get() y getline()

getline

Su diferencia con la función miembro **get()** es que el carácter de terminación se lee antes de que se añada el carácter '\0'. En consecuencia esta función no deja el carácter de terminación en una cola.

istream& getline (char *, int, char ='\n');

read

Al contrario que **get() y getline(), read()** no añade ningún carácter a la entrada, por lo que no convierte la entrada a formato cadena.

Sección 4 | Formato

Entrada/salida con formato

En C++ existen dos formas propias de manipular el formato de la salida:

- Emplear las funciones miembro de la clase **ios**.
- Emplear unas funciones especiales que reciben el nombre de **manipuladores**

Clase ios

- **Ajustar la anchura de los campos: width()**

  streamsize width() const; Informa de la anchura actual de un campo
  streamsize width(streamsize ancho); Estable el ancho y devuelve el ancho anterior

- **Precisión: precision()**

  int precision() const; Estado del valor de la variable de precisión.
  int precision(int p); Estable una nueva precisión.

- **Carácter de relleno: fill()**

  char fill() const; Devuelve el carácter de relleno
  char fill (char c); Establece el carácter de relleno.

Clase ios

Establecer formato

Cada objeto derivado de ios contiene una variable de estado de indicadores. Estos se pueden establecer con:

long	ios::flags()	Informa de todos
long	ios::flags(long flags)	Informa de todos, establece todos
long	ios::setf(long flags)	Informa de todos, establece máscara
long	ios::setf(long bis, long flags)	Informa de todos, establece un grupo (pone a 0 los primeros, después pone a 1 los segundos)

Clase ios

Indicadores de formato

ios::skipws	Saltar sobre espacios en blanco, sólo entrada
ios::left	Justificar a la izquierda
ios::right	Justificar a la derecha
ios::internal	Rellenar números con espacios después de los indicadores de base
ios::dec	Formato en base 10. Conversión decimal
ios::oct	Formato en base 8. Conversión octal
ios::hex	Formato en base 16. Conversión hexadecimal
ios::showbase	Visualizar indicador de base numérica. Sólo salida
ios::showpoint	Visualizar el punto decimal. Salida float
ios::uppercase	Visualizar dígitos hexadecimales en mayúsculas
ios::showpos	Añade un signo + a los números positivos
ios::scientific	Utilizar notación científica para reales
ios::fixed	Utilizar notación coma fija para reales
ios::unitbuf	Limpia los flujos después de la inserción
ios::stdio	Limpia stdout y stderr después de la inserción

Manipuladores

Hasta ahora se han visto varias formas de manejo del formato, bastante potentes, pero un poco engorrosas de emplear. C++ ofrece un enfoque más amigable con lo que se denominan **manipuladores**.

Manipulador	Entrada/Salida	Significado
`dec`	E / S	Formato de datos numéricos en decimal.
`endl`	S	Envía carácter de nueva línea.
`ends`	S	Envía un nulo \0.
`flush`	S	Vacía un flujo.
`hex`	E / S	Formato de datos numéricos en hexadecimal.
`oct`	E / S	Formato de datos numéricos en octal.
`resetiosflags(long l)`	E / S	Desactiva los indicadores de l.
`setbase(int i)`	S	Formato de los números en la base i.
`setfill(char c)`	E / S	Establece c como carácter de relleno.
`setiosflags(long l)`	E / S	Activa los indicadores de l.
`setprecision(int i)`	E / S	Establece los dígitos decimales.
`setw(int i)`	E / S	Anchura de un campo.
`ws`	E	Ignora los caracteres en blanco iniciales.

Sección 5 | Sobrecarga

Sobrecarga

- Los operadores << y >> pueden ser sobrecargados para permitir la inserción y extracción de nuevos tipos de datos definidos por el programador .
- El método de sobrecarga del operador siempre tiene que devolver una referencia al flujo sobre el que trabaja => encadenamiento de varias operaciones.
- Deben definirse externos a la clase, y si es preciso, **friend**.
- El primer parámetro es una referencia al stream sobre el que se va a trabajar. El segundo representa el dato que se va a insertar o extraer.

Sobrecarga

```
class Complejo {

  public:
      friend ostream& operator<<
      (ostream& os, const Complejo& c);
      ...

  private:
      double real;
      double imag;

}

ostream& operator<< (ostream& os,
      const Complejo& c) {

      os << c.real << ...

}
```

Sección 6 | Ejemplos

01 Ejemplo: read, write, gcount

```
#include <iostream>

using namespace std;
int  main()
{
  const int TAMANO = 80;
  char buffer[TAMANO];

  cout << "Introduzca una oracion: "; //DABALE ARROZ A LA ZORRA EL ABAD
  cin.read(buffer, 20);
  cout << "\nLa oracion introducida fue:\n";
  cout.write( buffer, cin.gcount() ); //DABALE ARROZ A LA ZO
  cout << endl;
}
```

02 Ejemplo: manipulador de flujo

```
#include <iostream>
#include <iomanip>

using namespace std;
int main ()
{
   int n;
   cout << "Introduzca un numero entero: "; //Introducimos: 456
   cin >> n;
   cout << n << " en hexadecimal es: " << hex << n << '\n'
        << dec << n << " en octal es: " << oct << n << '\n'
        << setbase(10) << n << " es decimal es: " << n << endl;
}
```

03 Ejemplo: flush

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Esta linea aparece inmediatamente." << flush;
    cout << "\nLo mismo sucede con esta..." << flush;
}
```

04 Ejemplo: uso de manipuladores

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   cout << "Octal: " << oct << 10 <<  " " << 20 << endl;
   cout << "Hexadecimal: " << hex << 10 << ' ' << 20 << endl;
   cout << "Decimal: "<< dec << 10 << ' ' << 20 << endl;

   cout << "Octal:   " << oct << 10 << endl;
   cout << "Decimal: "   << dec << 0xFF << endl;
   cout << "Decimal: " << dec << 012 << endl;
   cout << "Hexadecimal: "   << hex << 255 << endl;

   // Los siguientes ejemplos muestran el uso del manipulador setbase( )
   cout << "Octal: " << setbase( 8 ) << 10 << ' ' << 20 << endl;
   cout << "Hexadecimal: " << setbase( 16 ) << 10 << ' ' << 20 << endl;
   cout << "Decimal:" << setbase( 10 ) << 10 << ' ' << 20 << endl;
}
```

05 Ejemplo: uso de manipuladores

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   float valor = 1.2345;
   cout << setiosflags( ios::fixed );
   cout << setprecision( 0 ) << valor << endl;
   cout << setprecision( 1 ) << valor << endl;
   cout << setprecision( 2 ) << valor << endl;
   cout << setprecision( 3 ) << valor << endl;
   cout << setprecision( 4 ) << valor << endl;
   cout << setprecision( 5 ) << valor << endl;
   cout << setprecision( 6 ) << valor << endl;
}
```

06 Ejemplo: uso de manipuladores

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{
   double raiz2 = sqrt( 2.0 );
   int posiciones;
   cout  << setiosflags( ios::fixed )
   << "Raiz cuadrada de 2 con precision 0-9.\n"
     "Precision establecida por la "
     "funcion miembro precision():" << endl;
   for( posiciones = 0; posiciones <= 9; posiciones++ )
   {
   cout.precision( posiciones );
   cout << raiz2 << '\n';
   }//Fin del for
   cout  << "\nPrecision establecida por el "
         "manipulador setprecision( ):\n";
   for( posiciones = 0; posiciones <= 9; posiciones++ )
    cout << setprecision( posiciones ) << raiz2 << '\n';
}
```

07 Ejemplo: width

```
#include <iostream>

using namespace std;
int main()
{
   int ancho = 5;
   char cadena[ 10 ];
   cout << "Introduzca una oracion:\n";
   cin.width( 5 );
   while( cin >> cadena )
   {
     cout.width( ancho++ );
     cout << cadena << endl;
     cin.width( 5 );
   }
}
```

08 Ejemplo: showpoint

```
#include <iostream>

using namespace std;
int main()
{
   cout
   << "Antes de establecer el indicador de ios::showpoint\n"
   << "9.9900 se imprime como: " << 9.9900
   << "\n9.9000 se imprime como: " << 9.9000
   << "\n9.0000 se imprime como: " << 9.0000
   << "\n\nDespues de establecer el indicador de ios::showpoint\n";
   cout.setf( ios::showpoint );

   cout
   << "9.9900 se imprime como: " << 9.9900
   << "\n9.9000 se imprime como: " << 9.9000
   << "\n9.0000 se imprime como: " << 9.0000 << endl;

}
```

09 Ejemplo: manipuladores

```
#include <iostream>
#include <iomanip>

using namespace std;
int main()
{
   int x = 12345;
   cout
   << "La justificacion a la derecha esta predeterminada:\n"
   << setw( 10 ) << x << "\n\nUSO DE FUNCIONES MIEMBRO"
   << "\nUse setf( ) para establecer ios::left:\n" << setw( 10 );
   cout.setf( ios::left, ios::adjustfield );
   cout
   << x << "\ nUse unsetf( ) para restablecer el predeterminado:\n";
   cout.unsetf( ios::left );
   cout
   << setw( 10 ) << x
   << "\n\nUSO DE MANIPULADORES CON PARAMETROS"
   <<"\nUse setiosflags( ) para establecer ios::left:\n"
   << setw( 10 ) << setiosflags( ios::left ) << x
   << "\nUse resetiosflags( ) para restablecer el predeterminado:\n"
   << setw( 10 ) << resetiosflags( ios::left )
   << x << endl;
}
```

10 Ejemplo: manipuladores

```
#include <iostream>
#include <iomanip>

using namespace std;
int main()
{
   int x = 10000;
   cout << x << " impreso como un int justificado a derecha e\n"
   << "izquierda y como hex con justificacion interna.\n"
   << "Utilizando el caracter predeterminado de relleno (espacio): \n";
   cout.setf( ios::showbase );
   cout << setw( 10 ) << x << '\n';
   cout.setf( ios::left, ios::adjustfield );
   cout << setw( 10 ) << x << '\n';
   cout.setf( ios::internal, ios::adjustfield );
   cout << setw( 10 ) << hex << x;
   cout << "\n\nUtilizando varios caracteres de relleno:\n";
   cout.setf( ios::right, ios::adjustfield );
   cout.fill( '*' );
   cout << setw( 10 ) << dec << x << '\n';
   cout.setf( ios::left, ios::adjustfield );
   cout << setw( 10 ) << setfill( '%' ) << x << '\n';
   cout.setf( ios::internal, ios::adjustfield );
   cout << setw( 10 ) << setfill( '^' ) << hex << x << endl;
}
```

11 Ejemplo: manipuladores

```
#include <iostream>
#include <iomanip>

using namespace std;
int main()
{
   double x = .001234567, y = 1.946e9;
   cout
   << "Desplegado en formato predeterminado:\n"
   << x << '\t' << y << '\n';
   cout.setf( ios::scientific, ios::floatfield );
   cout
   << "Desplegado en formato cientifico:\n"
   << x << '\t' << y << '\n';
   cout.unsetf( ios::scientific );
   cout
   << "Desplegado en formato predeterminado despues de unsetf( ):\n"
   << x << '\t' << y << '\n';
   cout.setf( ios::fixed, ios::floatfield );
   cout
   << "Desplegado en formato fijo:\n"
   << x << '\t' << y << endl;
}
```

Sección 7 | Estado de los Flujos

Estado de los Flujos

Cuando se realizan operaciones de entrada y salida se pueden producir errores, para poder controlarlo podemos consultar su estado del flujo

- **eofbit** Se activa cuando se encuentra final-de-archivo. Para comprobar su estado utilizaríamos **eof()**.
- **badbit** Se activa cuando se produce un error fatal, como escribir en un fichero no abierto para escritura. Para comprobar su estado **bad()**.
- **failbit** Se activa cuando se produce un error de E-S, como podría ser leer un entero cuando se esperaba un carácter. Para comprobar su estado podemos utilizar **fail()**, cual se activa también cuando **badbit** esta activo.
- **goodbit** Esta activada cuando ninguna de las tres anteriores esta activada. **good()** nos indicará si el flujo esta bien.

12 Ejemplo: Estado Flujo

```
#include <iostream>

using namespace std;

int main (void)
   {
    int dato;
    cout << "Introduce un enter" << endl;
    cin >> dato;
    if (!cin.fail())
       cout << "Dato introducido: " << dato << endl;
     else
       cout << "Dato introducido incorrecto, no es un entero" << endl;
   }
```

13 Ejemplo: Estado Flujo

El lenguaje permite usar directamente un objeto istream o ostream en una expresión booleana, por lo que facilita la comprobación de errores.

```
#include <iostream>

using namespace std;

int main (void)
   {
    int dato;
    cout << "Introduce un enter" << endl;
    if (cin >> dato)
       cout << "Dato introducido: " << dato << endl;
     else
       cout << "Dato introducido incorrecto, no es un entero" << endl;
   }
```

13 Ejemplo: Modificar estado flujo

Es posible modificar de forma explicita el estado de un flujo con la funcion **setstate**, esto podríamos utilizarlo por ejemplo para una sobrecarga del operador de extracción

```
istream& operator >> (istream& entrada, Complejo& c)
{
double r, i;
char car = '/0';

entrada >> car;

if (car == '(') {
entrada >> r >> car;
if (car== ',') {
entrada >> i >> car;
if (car == ')')
c=Complejo(r,i);
else
entrada.setstate(ios::failbit);
}
else
entrada.setstate(ios::failbit);
}
entrada.setstate(ios::failbit);

return entrada;
}
```

Sección 8 | Flujo de Ficheros

Flujo de Ficheros

Para la E/S de ficheros se tiene que incluir la cabecera **fstream** donde están definidas las cabeceras **ifstream** y **ofstream**

Flujo de Ficheros

Apertura

En la biblioteca C hay una función de propósito general fopen(), debiendo definir primero un objeto ifstream o ofstream. Los constructores son

```
ifstream();
ifstream(const char* camino, openmode modo =ios_base::in | ios_base ::trunc);

ofstream();
ofstream(const char* camino, openmode mod = ios_base::out | ios_base::trunc);
```

Si se ha utilizado el constructor sin parámetros, entonces debemos abrir el fichero para ello utilizaremos:

```
void ifstream::open(const char* camino, openmode modo =ios_base::in | ios_base ::trunc);
void ofstream::open(const char* camino, openmode mod = ios_base::out | ios_base::trunc);
```

- **camino** Nombre o camino del fichero que se quier abrir
- **modo** Modo de apertura. (Se pueden aplicar varios a la vez)

Flujo de Ficheros

Modos de apertura

Los distintos modos de apertura son:

- ios_base::in Lectura (entrada)
- **ios_base::out** Escritura (salida)
- **ios_base::ate** Colocación al final del fichero
- **ios_base::app** Modo anexado. Se posiciona el cursor al final del fichero.
- **ios_base::bin** El archivo se abre en modo binario
- **ios_base::nocreate** Genera un error si el fichero no existe
- **ios_base::noreplace** Genera un error si el fichero ya existe

Flujo de Ficheros

Cierre

Consiste en la desconexión del fichero físico del flujo. Podemos cerrar el fichero de las siguiente maneras

- **ifstream() - ofstream(), fstream()** Destructor de la clase asociada
- **close()**
- Mediante el comando **exit()** de la biblioteca estándar de C (en <stdlib>), o mediante el **return** de la función **main()** que llamaría al destructor

14 Ejemplo: Ejemplo de ficheros

```
#include <fstream> // Biblioteca para el manejo de ficheros
#include <iostream> // Biblioteca para la entrada−salida estandar (NO HACE FALTA PUES ESTA INCLUIDA EN FSTREAM)
int main(){
ofstream fichout("EJEMPLO5.TXT",ios::out);
if (!fichout)
cout << "\n Incapaz de crear este o abrir el fichero \n";
else {
fichout << 1 << " " << 5.0 << " APROBADO" << endl;
fichout << 2 << " " << 1.1 << " SUSPENSO" << endl;
fichout << 3 << " " << 8.0 << " NOTABLE " << endl;
fichout.close();
}
} // Fin del main
```

15 Ejemplo: Ejemplo de ficheros

```
#include <fstream> // Libreria para el manejo de ficheros
#include <iostream> //(NO HACE FALTA PUES ESTA INCLUIDA EN FSTREAM
typedef char TCadena[30];
int main(){
int i;
char c;
float r;
TCadena cad;
ifstream fichin("EJEMPLO5.TXT"); // declaracion y apertura del fichero
if (!fichin)
cout << "\n Incapaz de crear este o abrir el fichero ";
else{
fichin >> i; // Observese la lectura adelantada!!!
while (!fichin.eof()){
cout << i << " ";
fichin >> r;
cout << r << " ";
fichin >> cad;
cout << cad << "\n";
fichin >> i;
}
fichin.close();
} // Fin del main
```

16 Ejemplo: Ejemplo de ficheros

```
#include<iostream>
#include<fstream>
void main ( )
{
using namespace std;
// create a read/write file−stream object on tiny char and attach it to the file "filebuf.out"
ofstream out("filebuf.out",ios_base::in |
ios_base::out);
// tie the istream object to the ofstream object
istream in(out.rdbuf());
out << "Il errait comme un ame en peine"; // output to out
in.seekg(0); // seek to the beginning of the file
cout << in.rdbuf() << endl; // output in to the standard output
out.close(); // close the file "filebuf.out"
// open the existing file "filebuf.out" and truncate it
out.open("filebuf.out",ios_base::in | ios_base::out |
ios_base::trunc);
out.rdbuf()−>pubsetbuf(0,4096); // set the buffer size
ifstream ins("filebuf.cpp"); // open the source code file
out << ins.rdbuf(); //output it to filebuf.out
out.seekp(0); // seek to the beginning of the file
cout << out.rdbuf(); // output the all file to the standard output
}
```

Sección 9 | Bibliografía

Bibliografía

Introducción

- Fundamentos de C++