Entrada/Salida en C++

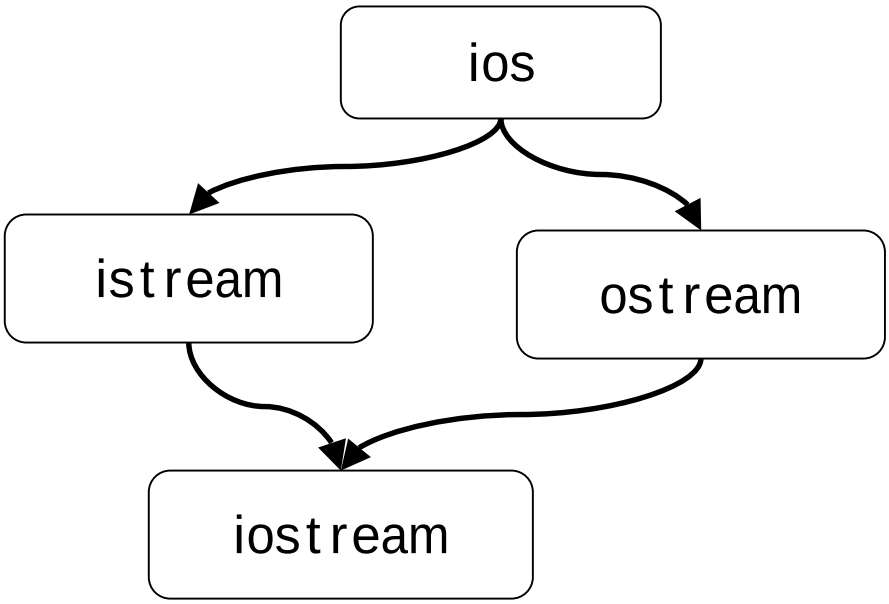
1. Gestión de E/S.
2. Clases básicas.
3. E/S estándar.
4. Manejo de ficheros.

Gestión de Entrada/Salida

- Principios generales (objetivos):
  - Independencia y transparencia entre recursos, emisores y receptores de información.
  - No aparezcan parámetros físicos.
  - Sencillo, eficiente, flexible, seguro y completo.
- En P.O. debe manejar datos básicos y estructurados:
  - conversión objetos en información comprensible y accesible desde los diferentes centros emisores y receptores.
- Mecanismo de E/S en P.O. se divide en:
  - Salida: conversión a entidades externas: usuario, otra aplicación, máquinas, etc.
  - Entrada: inversa de la salida.
  - Formato E/S: especificaciones de formato y estructura que se captura desde o se emite al exterior.
  - Ficheros y Flujos: tratamiento especializado de dispositivos y ficheros.

- En C:
  - Funciones de E/S complicadas y con importantes deficiencias.
  - No efectúan comprobación de tipos.
- En C++:
  - Funciones E/S, biblioteca `iostream.h`
  - Comprobación de los tipos de datos.
  - Tratamiento uniforme de los tipos de datos y clases de objetos.
  - Mecanismos para crear nuevas funciones de E/S para dispositivos no estándar, con mismo tipo de tratamiento.
- El concepto de *stream* en C++:
  - flujo de datos desde un origen (entrada) a un destino (salida).
  - Permite almacenamiento en una salida o extracción desde una entrada.
  - Se asocian a dispositivos físicos o ficheros.
  - Operadores inserción << y extracción >>:
    - Sobrecarga de operadores << y >>.

- Clases básicas:



- Clase `ios` contiene:
  - puntero a objeto buffer de almacenamiento temporal.
  - constantes y variables de formato de E/S y de error.
- Clase `iostream` y `ostream`:
  - definen funciones para entrada y salida con formato, respectivamente.

- E/S estándar en C++:
  - Entrada estándar:
    - Representa al dispositivo de entrada por defecto, generalmente el teclado.
    - El *stream* asociado es `cin`, de la clase `istream`.
  - Salida estándar:
    - Representa al dispositivo de salida por defecto, generalmente la pantalla.
    - El *stream* asociado es `cout`, de la clase `ostream`.
  - Salida de error:
    - Representa al dispositivo de salida por defecto donde se envían los mensajes de error, generalmente la pantalla.
    - Los *streams* asociados son `cerr` y `clog`, de la clase `ostream`.

- Parámetros y *flags* de formato:
  - `ios` define una variable que agrupa 15 bits de estado de formato.
  - `ios` proporciona 15 constantes que permiten diferenciar los bits de formato:

Flag	**Bit Field**
`ios::left`	`ios::adjustfield`
`ios::right`	
`ios::internal`	
`ios::dec`	`ios::basefield`
`ios::oct`	
`ios::hex`	
`ios::scientific`	`ios::floatfield`
`ios::fixed`	
`ios::skipws`	se ignoran espacios en blanco
`ios::showbase`	muestra indicador de la base
`ios::uppercase`	X y E mayúsculas
`ios::showpoint`	0.1 y 2.0 en lugar de .1 y 2.
`ios::unitbuf`	se vacía buffer S tras cada inserción
`ios::showpos`	muestra el signo positivo
`ios::stdio`	compatibilidad con E/S de C

- Establecimiento del formato con funciones:

Función (consulta/cambio)	**Ejemplo**
fill	cout.fill('-');
precision	cout.precision(5);
width	cout.width(30);
flags	cout.flags(ios::showbase|ios::left);
setf	cout.setf(ios::showbase|ios::left);
	cout.setf(ios::dec,ios::basefield);
unsetf	cout.unsetf(ios::showbase);

- Establecimiento del formato con manipuladores:
  - Operadores especiales que permiten la inserción de funciones de formato en las operaciones de E/S.

- Manipuladores de formato:

Ámbito	Manipulador	Descripción
E/S	dec	fija base decimal
E/S	oct	fija base octal
E/S	hex	fija base hexadecimal
E	ws	extrae espacios en blanco
S	endl	inserta salto de línea
S	ends	inserta carácter nulo
S	flush	vuelca el stream
E/S	setw(int)	fija ancho de campo
S	setfill(char)	fija el carácter de relleno
S	setprecision(int)	fija la precisión
S	setbase(int)	fija la base (8,10,16)
	setiosflag(long)	equivalente a setf()
	unsetiosflag(long)	equivalente a unsetf()

Ejemplos:

```
cout << setw(20) << setfill('-') <<
    setiosflag(ios::right)<< "CADENA";

cin >> ws >> setbase(8) >> x;
```

- Gestión de errores E/S:
  - Cuando ocurre un error en la E/S en un *stream*, este error se refleja en un atributo de estado del *stream*:

```
class ios
{
public:
  enum io_state
    { goodbit = 0x00;
      eofbit  = 0x01;
      failbit = 0x02;
      badbit  = 0x03;
    };

  inline int good() const;
  inline int eof()  const;
  inline int fail() const;
  inline int bad()  const;

...

};
```

- Ejemplo:

```
char cadena[30];
int estado;

cin.clear();  //limpia estado de errores
estado=cin.rdstate(); //lee estado de
                      //error del stream
switch(estado)
  { case ios::goodbit : ...
    case ios::eofbit  : ...
    case ios::failbit : ...
    case ios::badbit  : ...
  }

...

cin.clear(ios::fail
                      //error a mano
...

if(!cin) cout << "ERROR";
if(cin)  cout << "Todo va bien!";
```

Equivalente al switch:

```
if(cin.good()) {...}
if(cin.eof())  {...}
if(cin.fail()) {...}
if(cin.bad())  {...}
```

- La clase *ostream*:

```
class ostream
{  ...
public:
  ostream & operator<< (int);
  ostream & operator<< (long);
  ostream & operator<< (char);
  ostream & operator<< (float);
  ostream & operator<< (char *);
  ostream & put(char);
  ostream & write(const char *,int);
  ostream & flush();  // vacía buffer
...
};
```

Devuelven objeto *ostream* por referencia:
* permite concatenar llamadas a operadores y funciones.

- Ejemplos:

```
int c='Q';
cout.put(c);  // la salida es Q
cout.put(48); // carácter '0', ascii 48

char *str="ABCDEFGH";
cout.write(str+1,4); //la salida es "BCDE"
```

- La clase *istream*:

```
class istream
{  ...
public:
  istream & operator>> (int);
  istream & operator>> (long);
  istream & operator>> (char);
  istream & operator>> (float);
  istream & operator>> (char *);
  istream & get(char &c);
  istream & getline(char *s,int n,
                    char c='\n');
  istream & read(const char *,int n);
  istream & putback(char c);
  istream & ignore(int n,int t=EOF);
   int sync();
...
};
```

get()	extrae un carácter del stream.
getline()	extrae hasta n caracteres del stream o hasta que encuentra el carácter delimitador.
read()	extrae n caracteres del stream.
putback()	devuelve un carácter al stream.
ignore()	extrae y descarta n caracteres del o hasta encontrar fin de stream.
sync()	vacía el buffer de entrada.

- Sobrecarga de los operadores >> y <<

```
class punto
{ int x,y;

public:
  friend ostream & operator<< (ostream &,
                               punto);
  friend istream & operator>> (istream &,
                               punto &);
};
```

Se sobrecargan como funciones *friend*.

Son operadores binarios:
* el primer operando es siempre un *stream* por referencia y
* el segundo un objeto de la clase en cuestión.

```
ostream & operator<< (ostream & os,
                      punto p)
{ os << '(' << p.x << ',' << p.y << ')';
  return(os);
}
```

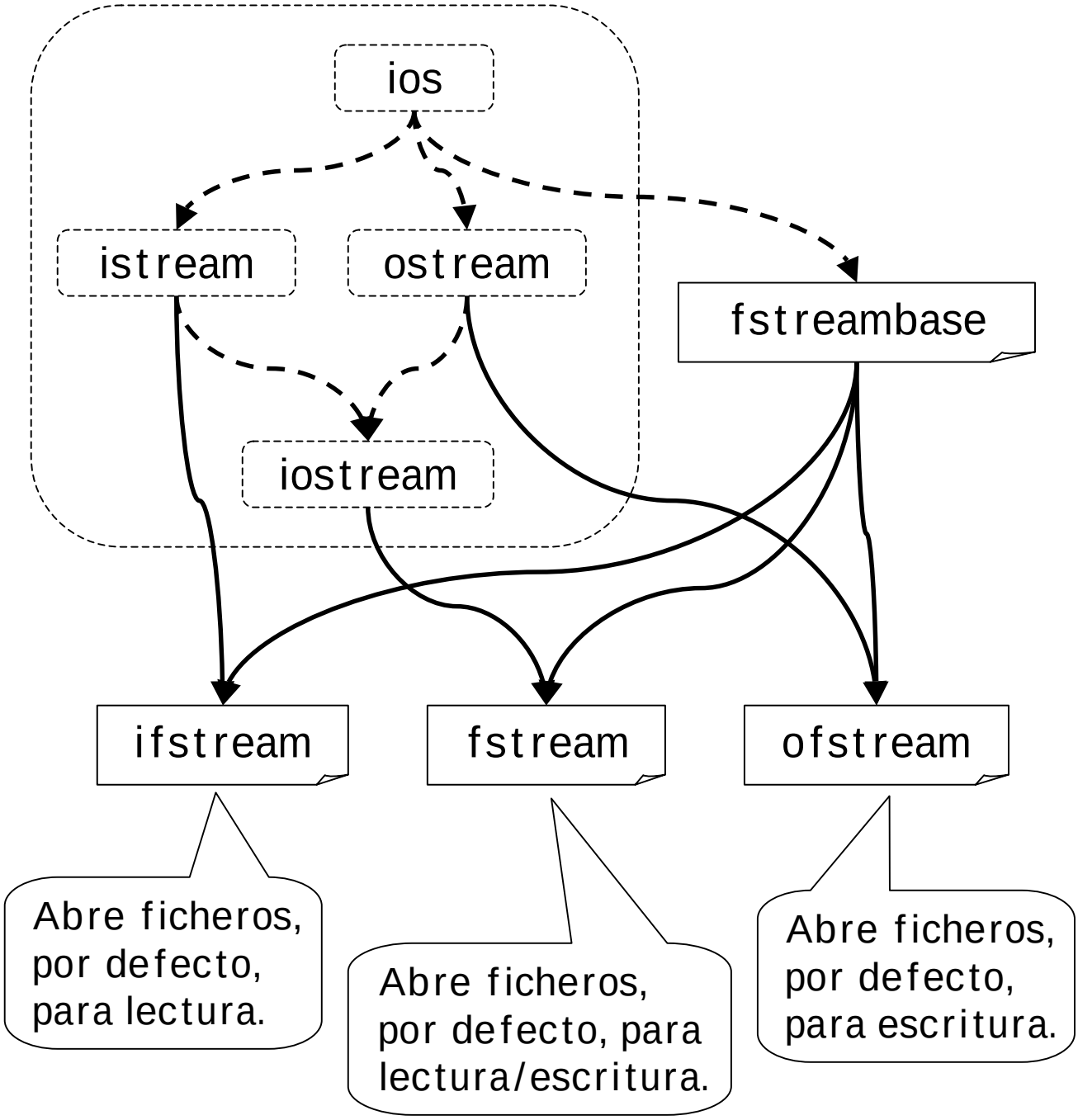
Se devuelve el mismo *stream* para otras operaciones de concatenación

```
istream & operator>> (istream & is,
                      punto & p)
{ char c;

is >> c;
if(c!='(')
  is.clear(ios::failbit);
else
  { is >> p.x >> c;
    if(c!=',') is.clear(ios::failbit);
    else
      { is >> p.y >> c;
        if(c!=')')
          is.clear(ios::failbit);
      }
  }
return(is);
}
```

Se pone el error manualmente.

```
void main(void)
{ punto p;

cin >> p;
if(!cin)
  { cout << "ERROR de formato de entrada";
    cin.clear(); //limpia registro de estado
  }
};
```

- Ficheros:


ios
istream
ostream
fstreambase
iostream
ifstream
fstream
ofstream
Abre ficheros, por defecto, para lectura.
Abre ficheros, por defecto, para lectura/escritura.
Abre ficheros, por defecto, para escritura.

```
class fstreambase
{  ...

public:
  fstreambase();
  fstreambase(const char *,int,int);
  ~fstreambase();
  void open(const char*,int,int);
  int is_open();
  close();
...
};
```

Nombre del fichero a abrir

Modo de apertura

Modo de protección:
- S_IREAD
- S_IWRITE

Por defecto, se permite compartirlo para lectura/escritura.

Modos de apertura

ios::in	lectura.
ios::out	escritura.
ios:ate	abrir y saltar al final.
ios::app	añadir.
ios::trunc	destruir el fichero si existe.
ios::nocreate	error si el fichero no existe.
ios::noreplace	error si el fichero existe.
ios::binary	modo binario

- Ejemplo (fichero de escritura):

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h> //manipuladores formato

void main(void)
{

// fichero de escritura
ofstream out("ARCHIVO");

out << setw(20) << setfill('.') <<
       setiosflag(ios::left) <<
       "Linea" << 1 << endl;

out << setw(20) << setfill('.') <<
       setiosflag(ios::left) <<
       "Linea" << setiosflag(ios::showpos) <<
       2 << endl;

out.close();
}
```

Equivale a:
`out("ARCHIVO",ios::out);`

Ancho de campo, 20.
Carácter relleno: '.'
Justificación izquierda.
Mostrar signo positivo.

- Ejemplo (fichero de lectura):

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h> //manipuladores formato

int main(void)
{

// fichero de lectura
ifstream in;
char cadena[30];

in.open("ARCHIVO");
if(!in)
  { cerr << "ERROR en la apertura.";
    return 1;
  }

in.getline(cadena,30);
cout << cadena << "Caracteres leídos " <<
   in.gcount() << endl;

in.close();
return 0;
}
```

Se crea un objeto fichero pero no se abre ninguno aún.

Devuelve el número de caracteres leídos.

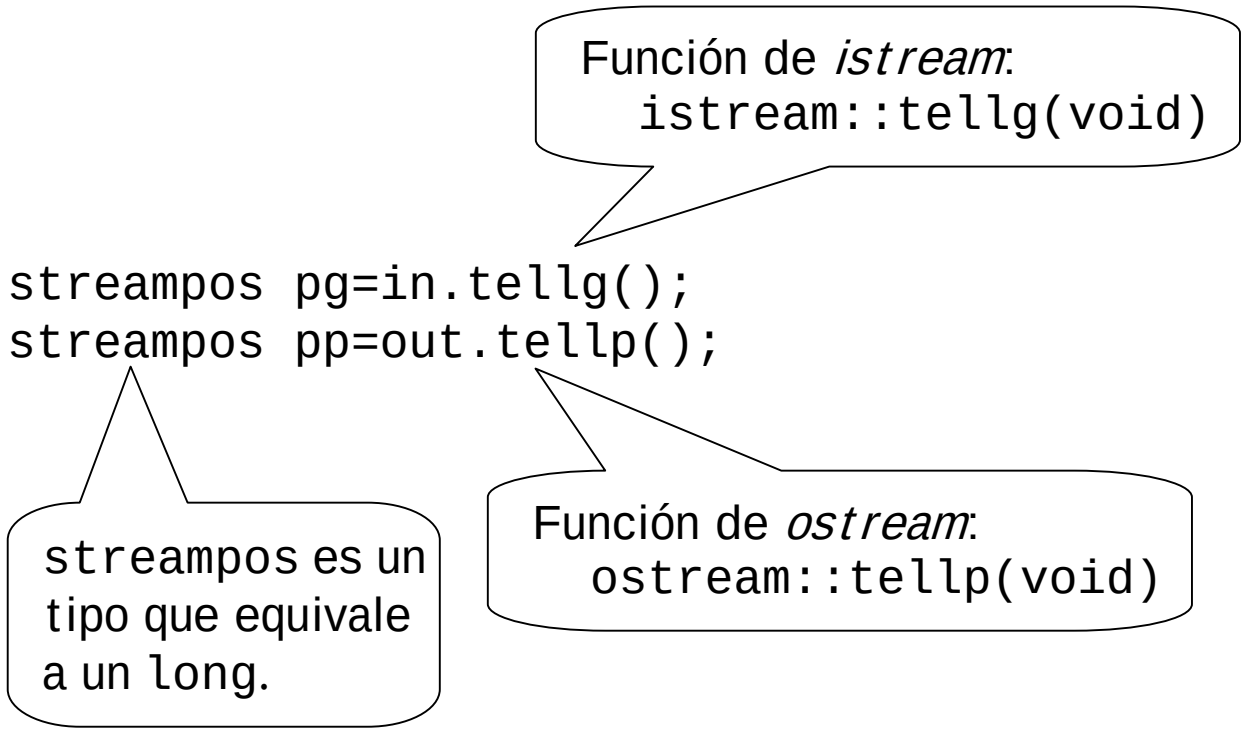
- Ejemplo (fichero de lectura/escritura):

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h> //manipuladores formato

int main(void)
{
// fichero de
// lectura/escritura
fstream io("ARCHIVO",
          ios:in | ios::out | ios::nocreate);
char c;

if(!io)
  { cerr << "ERROR en la apertura.";
    return 1;
  }
io >> c;
if(io.eof()) cout << "FICHERO VACIO" << endl;

while(!io.eof())
  { cout << c << flush;
    io >> c;
  }
return 0;
}
```

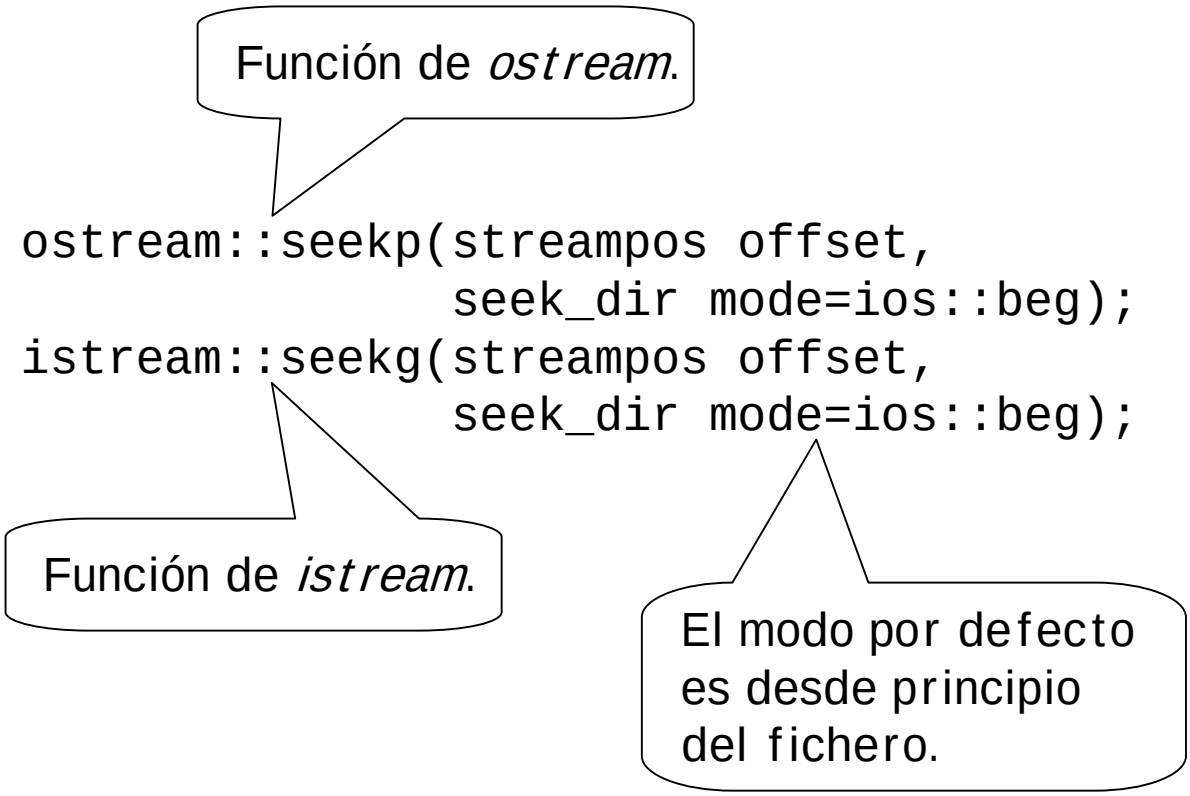
Se modifican modos para lectura, escritura y apertura si está creado.

Se vuelca contenido del buffer en el stream De salida.

- Acceso directo a ficheros:
  - Averiguar posición actual del fichero:



– Saltar a una posición del fichero:



Modos de posición inicial del salto

ios::beg	desde principio del fichero.
ios::cur	desde posición actual del fichero.
ios::end	desde final del fichero.

- Ejemplo (acceso directo a ficheros):

```
struct articulo { char codigo;
                  int precio; };

int main(void)
{ articulo rart;
  fstream fart("ARTICULOS",ios::in |
                           ios::binary);
if(!fart)
  { cout << "ERROR en la apertura\n";
    return 1;
  }

int index=0;
while(index>=0)
  { cout << "Número de registro: ";
    cin >> index;
    if(index>=0)
      { streampos sp=sizeof(articulo)*index;
        fart.seekg(sp);
        fart.read((char*)&rart,
                  sizeof(articulo));
        cout << rart.codigo   <<
                rart.precio   << endl;
      }
  }
}
```

Fichero de lectura binario.

Calcular posición del registro indicado.

Saltar al registro indicado.

Leer el registro.

Mostrar el contenido del registro.

- Excepción:
  - Error ocurrido en tiempo de ejecución
  - Ejemplos:
    - Desbordamiento de un vector.
    - Error en la apertura de un fichero.
    - Falta de memoria.
    - etc.
- Manejo de excepciones:
  - detectar excepciones y desviar la ejecución del programa hacia un manipulador de la excepción.
- Manipulador de excepción:
  - fragmento de código que se ejecuta cuando se produce una determinada excepción.
  - Se encarga de:
    - devolver el programa a la situación correcta, o
    - finalizarlo, si la situación es irreversible.

- Estrategia de la “patata caliente”:
  - Cuando en una función se produce una excepción:
    - Si la función dispone de un manejador para esa excepción:
      - lo ejecuta.
    - Si no, se pasa la excepción a la función de nivel superior.
    - Si la excepción llega a la función main():
      - Si no existe un manipulador para esa excepción:
        - » se aborta la ejecución del programa.

- Instrucciones:
  - `throw` envía el mensaje de excepción.
  - `catch` captura el mensaje de excepción (manejador).
  - `try` marca el fragmento de código donde se lanzarán las excepciones.

- Utilización:


```
try
  { ...
    if(excepcion_n) throw objeto_n;
  }

catch(clase_1 m)
  { // código del manejador;
    ...
  }
...

catch(clase_n m)
{ // código ...
}
```

- Ejemplo:

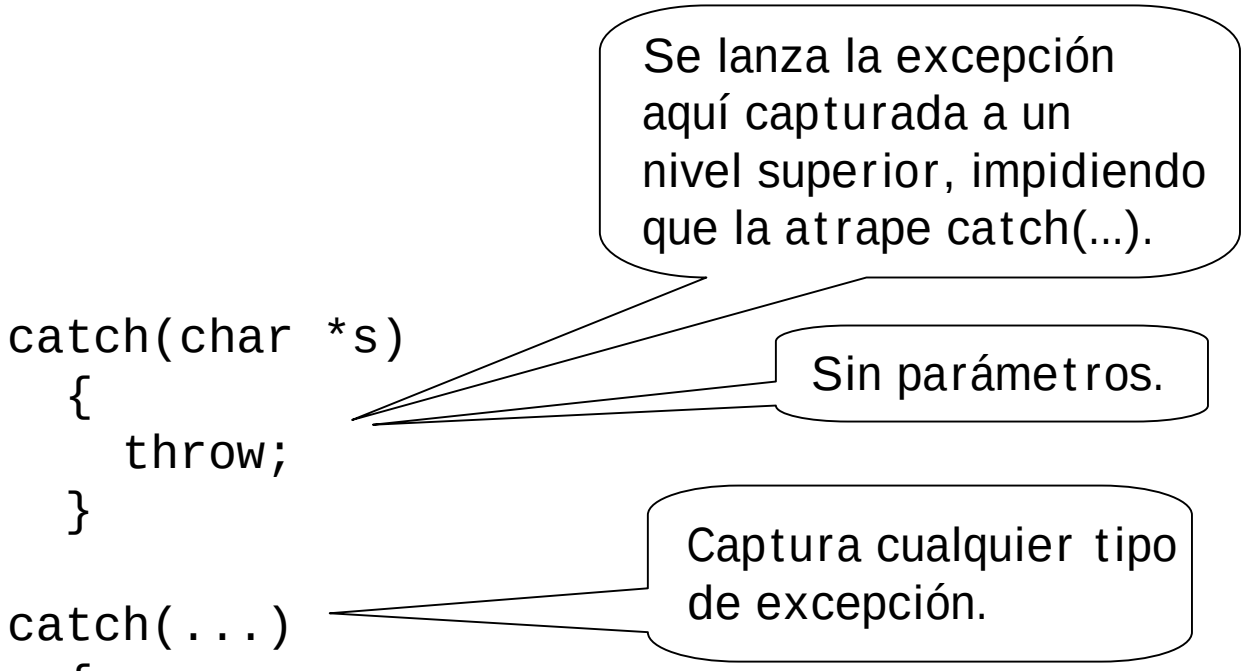
```
void main(void)
{ f(600);
  cout << "Final programa\n";
}
```

```
void f(int v)
{ float b;

try
  { if(v>500) throw "Valor fuera de rango";
    if(v==0) throw 1;
    b=50.0/(float)v;
    if(b<0) throw 2;
    b=sqrt(b);
  }

catch(int i)
  { switch(i)
      { case 1: cout<<"División por cero\n";
                break;
        case 2: cout<<"Negativo en sqrt\n";
                break;
      }
  }

catch(char *s)
  { cout << s << endl; }

cout << "Final de la función\n";
}
```

SALIDA:

```
Valor fuera de rango
Final función
Final programa
```

- Relanzar una excepción:




```
catch(char *s)
  {
    throw;
  }

catch(...)
  {
    cout << “Error indeterminado\n”;
  }
```

- Anidar manipuladores:

```
catch(char *s)
  { try
      { ... // código de tratamiento
      }

    catch(char *s)
      { ... }
  }
```

Ha fallado el código del anterior try.

- Lista `throw`:
  - permite especificar que tipo de excepciones puede generar (pasar a nivel superior) una función.

Se especifica en el prototipo o en la cabecera de la función.

```
void f() throw (char *s);
void f() throw (char *s,int);
void f() throw (int,vector);
```

Pueden emitir sólo excepciones del tipo especificado.

```
void f() throw ();
void f();
```

No puede emitir ningún tipo de excepción.

Puede emitir cualquier tipo de excepción.