REDE GALEGA
de BIOINFORMÁTICA

ESTRUCTURAS DE DATOS Y C/C++

<<Streams>> y Strings en C++

Dr. *Florentino Fernández Riverola*
Escuela Superior de Ingeniería Informática
Departamento de Informática
Universidad de Vigo
riverola@uvigo.es

Índice

- **Definición y tipos de *streams***
- **Operadores de inserción y extracción**
- **Otros métodos de entrada y salida**
- **Streams en disco**
- **Streams en memoria**
- **Sobrecarga de los operadores << y >>**
- **Definición de manipuladores**
- **STRINGS**

Definición:

#Stream/flujo#

Cualquier dispositivo que produce o consume información

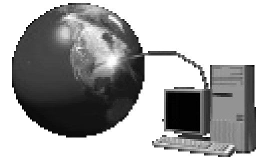
- Un flujo siempre está ligado a un dispositivo físico
- Comunica un origen y un destino mediante una *tubería*
- Todos los flujos se comportan de forma análoga independientemente del dispositivo final => *homogeneidad y abstracción*
- Cuando se ejecuta un programa en C++ se abren automáticamente:
  - ***cin***: entrada estándar (teclado)

  - ***cout***: salida estándar (pantalla)
  - ***cerr***: salida de mensajes de error (pantalla)
- C++ dispone de 2 jerarquías de clases para entrada/salida:
  - ***istream, ostream*** *e* ***iostream***: deriban de la clase ***ios***
  - ***streambuf***: manejo a bajo nivel de la entrada/salida

Tipos de Streams

- Los *streams* más normales son los que se establecen entre un programa y la consola, el teclado o la pantalla
- También existen *streams* entre un programa y un fichero, pudiendo actuar cada una de las entidades como emisor o receptor de datos
- Análogamente se puede transferir información entre zonas de memoria en forma de *stream*.
- En algunos casos el origen y el destino necesitan abrirse antes de poder ser utilizados
  - ***ficheros***: SÍ
  - ***consola***: NO
  - ***streams en memoria***: NO

Operadores de inserción y extracción



Operadores de inserción y extracción

• “<<“ Operador de inserción definido en *iostream.h* y sobrecargado en **istream**

• “>>” Operador de extracción definido en *iostream.h* y sobrecargado en **ostream**

• Devuelven siempre una referencia a sí mismos => encadenamiento

• Están sobrecargados para reconocer todos los tipos básicos de datos del lenguaje:

  - enteros cortos y largos con y sin signo
  - número de punto flotante
  - caracteres con y sin signo
  - matrices de caracteres en forma de punteros a carácter con y sin signo
  - punteros (imprimen su representación)

• Utilizando “>>” para cadenas, el espacio en blanco actúa como fin de la cadena

Stream1.cpp

Formateo de la entrada/salida (i)

• Existen dos posibilidades:

  • Ejecutar **métodos** sobre el objeto que representa el *stream*

  • Utilizar **manipuladores** con los operadores de inserción y extracción

• Existen métodos y manipuladores que realizan la misma función => Comodidad

• **ANCHURA**:

  • Por defecto el ancho se corresponde con el espacio necesario para su almacenamiento

  • Los datos se ajustan por defecto a la derecha

  • Método **width()**. Sin parámetros retorna el ancho actual [cout.width(0)]

  • Manipulador **setw()**. Después de cada inserción o extracción el ancho toma el valor 0

Stream2.cpp

Formateo de la entrada/salida (ii)

• **CARACTER DE RELLENO**:

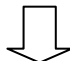
  • Por defecto el carácter de relleno utilizado es el espacio en blanco

  • Los datos se ajustan por defecto a la derecha

  • Método **fill()**. Sin parámetros retorna el carácter de relleno actual

  • Manipulador **setfill()**

  • El caracter de relleno no cambia automáticamente al espacio tras cada operación



  • Es conveniente guardar el carácter de relleno actual

Stream3.cpp

Formateo de la entrada/salida (iii)

• **PRECISIÓN**:

- Por defecto un número en punto flotante se presenta con todos sus dígitos
- Método **precision**(). Sin parámetros retorna el valor fijado anteriormente
- Manipulador **setprecision**()

Stream4.cpp

• **BASES DE NUMERACIÓN**:

- Lo normal es realizar las entradas y salidas de datos numéricos en decimal
- Se pueden utilizar los modificadores **dec**, **hex**, **oct** y **setbase**()

Stream5.cpp

Formateo de la entrada/salida (iv)

• **OTROS MANIPULADORES**:

- **endl**: Inserta un caracter de nueva línea (retorno de carro)
- **ends**: Inserta un terminador de cadena (un carácter nulo) "\0"
- **flush**: Fuerza la salida de los caracteres que estén en el *buffer*
- **flush**(): También existe como método de un *stream* de salida, (*cout*)
- **ws**: Se ignoran los espacios en blanco en las operaciones de entrada, (*cin*). Activo por defecto

Stream6.cpp

Formateo de la entrada/salida (v)

• **INDICADORES DE FORMATEO (i)**:

• **ios::skipws**: Cuando está activo indica a los objetos de entrada que ignoren los espacios en blanco iniciales

---

• **ios::left**: Activa la justificación a la izquierda en operaciones de salida

• **ios::right**: Activa la justificación a la izquierda en operaciones de salida. Por defecto

• **ios::internal**: Al activarlo permite que en las salidas numéricas el signo o el indicador de base, se muestren a la izquierda del relleno como primer caracter

---

• **ios::dec**: Activa la conversión decimal

• **ios::oct**: Activa la conversión octal

• **ios::hex**: Activa la conversión hexadecimal

• **ios::uppercase**: Al activarlo, en las salidas de datos hexadecimales se utilizarán letras mayúsculas en lugar de minúsculas

Formateo de la entrada/salida (v)

• **INDICADORES DE FORMATEO (ii)**:

• **ios::showbase**: Al activarlo, en las operaciones de salida se mostrará el indicador de base numérica, por ejemplo, **0x** para hexadecimal

• **ios::showpos**: Activándolo, en las salida de datos numéricos positivos se mostrará siempre el signo. Por defecto desactivado

• **ios::showpoint**: Causa que en las operaciones de salida de datos en punto flotante se muestre el punto decimal y los dígitos decimales necesarios para completar la salida

---

• **ios::scientific**: Cuando está activo, las salidas de punto flotante se realizan en notación científica

• **ios::fixed**: Es exclusivo con el anterior, ya que causa que las salidas de punto flotnate se realicen en notación decimal

---

• **ios::initbuf**: Activando este indicador, la entrada/slida con el objeto se realizará sin utitilizar *buffer*, volcando carácter a carácter las entradas o salidas en el dispositivo origen o destino

---

Formateo de la entrada/salida (v)

• **INDICADORES DE FORMATEO (iii): métodos**

• **setf()**: Permite activar de indicadores de formateo de forma independiente

• **setiosflags()**: Manipulador equivalente al método **setf()**

• **unsetf()**: Permite desactivar indicadores de formateo de forma independiente

• **resetiosflags()**: Manipulador equivalente al método **unsetf()**

• **flags()**: Asigna directamente el estado de todos los indicadores

---

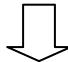
• Todos los métodos y manipuladores toman como parámetro un entero largo en el que se especifican los indicadores a activar o desactivar

• Los indicadores están definidos en la clase **ios** en forma de *enumeración*

Formateo de la entrada/salida (v)

• **INDICADORES DE FORMATEO (iv): varios**

• Los métodos **flags()**, **setf()** y **unsetf()** devuelven un entero con el estado de los indicadores antes de alterarlo



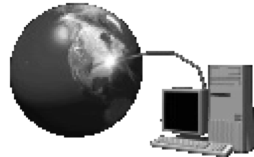
• Aconsejable almacenarlo (*long*) para restituirlos posteriormente

• El proceso habitual es habilitar/deshabilitar indicadores uniéndolos con el operador "|" mediante: **setf()**, **setiosflags()**, **unsetf()**, **resetiosflags()**

• *Ajuste a la izquierda* y *relleno interno* son incompatibles

Stream7.cpp

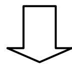
Otros métodos de Entrada / Salida



Otros métodos de Entrada/Salida

• ... Operadores de inserción y extracción realizan E/S con formato

• No obstante sigue siendo interesante la E/S sin formato para ciertas situaciones



• Métodos especiales definidos en las clases **istream** y **ostream** (*cin*, *cout*)

Otros métodos de E/S (i)

• **EL MÉTODO ISTREAM::GET**

• Realiza lecturas o entradas de datos desde un objeto. Está sobrecargado y posee 4 versiones:

- *int* **get()**: Toma el siguiente carácter y lo devuelve. **EOF** (*stdio.h*) al final del *stream*
- *istream&* **get(***char*, int len, char = '\n'***)**
- *istream&* **get(***signed char*, int len, char = '\n'***)**
- *istream&* **get(***unsigned char*, int len, char = '\n'***)**

El delimitador no se incluye en la cadena ni se consume del stream

- *istream&* **get(***char&***)**
- *istream&* **get(***signed char&***)**
- *istream&* **get(***unsigned char&***)**

- *istream&* **get(***streambuf&, char = '\n'***)**

Otros métodos de E/S (ii)

• **LECTURA DE LÍNEAS**

- El método **getline()** es similar a la segunda versión del método **get()**. Recibe los mismos parámetros y el mismo valor por defecto para el delimitador

- *istream&* **getline(***char*, int, char = '\n'***)**
- *istream&* **getline(***signed char*, int, char = '\n'***)**
- *istream&* **getline(***unsigned char*, int, char = '\n'***)**

- Retira el carácter de terminación del *stream*, pero no lo almacena en la cadena

- En caso de que se llegue al final del *stream* antes de completar la lectura => **gcount()** devuelve el número de bytes sin formato leídos en la última operación

- La extracción sin formato se lleva a cabo con los métodos **get()**, **getline()**, y **read()**

Stream7a.cpp

Otros métodos de E/S (iii)

• **LECTURA ABSOLUTA**

- Por medio del método **read()** es posible leer un número determinado de caracteres desde el *stream* y almacenarlo en el espacio de memoria indicado
- *istream&* **read(***char*, int***)**
- *istream&* **read(***signed char*, int***)**
- *istream&* **read(***unsigned char*, int***)**
- Primer parámetro: puntero con la dirección dónde almacenar la información
- Segundo parámetro: Longitud de la información a leer
- En caso de que se llegue al final del *stream* antes de completar la lectura => **gcount()** devuelve el número de bytes leídos

Otros métodos de E/S (iv)

• **MOVIMIENTO EN EL STREAM**

- Aunque con los objetos *cout* y *cin* no tenga mucho sentido, es posible desplazar el puntero de lectura de un *stream*
- *long* **tellg()**: Devuelve la posición actual en el *stream*
- **seekg()** permite desplazar el puntero de lectura a una posición absoluta o relativa
  - *istream&* **seekg(***streampos pos***)**
  - *istream&* **seekg(***streamoff offset, seek_dir dir***)**
    - offset: desplazamiento
    - dir: *ios::beg*, *ios::cur* e *ios:end*

Stream7b.cpp

Otros métodos de E/S (v)

• **OTRAS OPERACIONES DE LECTURA**

- *int* **peek()**: Retorna el código del siguiente caracter sin consumirlo del *stream*
- *istream&* **putback(***char***)**: Devuelve un caracter al *stream*. Debe existir siempre una lectura previa
- *istream&* **ignore(***int n = 1, int delim = EOF***)**: Ignora caracteres hasta alcanzar el número *n* o encontrar el delimitador especificado

• **ESCRITURA SIN FORMATO (i)**

- *ostream&* **put(***char ch***)**
- *ostream&* **put(***signed char ch***)**
- *ostream&* **put(***unsigned char ch***)**
- *ostream&* **write(***const char*, int n***)**
- *ostream&* **write(***const signed char*, int n***)**
- *ostream&* **write(***const unsigned char*, int n***)**

Otros métodos de E/S (vi)

• **ESCRITURA SIN FORMATO (ii)**

- *long* **tellp()**: Devuelve la posición actual en el *stream*
- **seekp()** permite desplazar el puntero de escritura a una posición absoluta o relativa
  - *ostream&* **seekp(***streampos pos***)**
  - *ostream&* **seekp(***streamoff offset, seek_dir dir***)**

• offset: desplazamiento

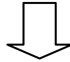
• dir: *ios::beg*, *ios::cur* e *ios:end*

Stream8.cpp

Los *streams* y los *buffers (i)*

• Un *buffer* es una zona de memoria que se utiliza como almacenamiento intermedio para evitar así tener que acceder continuamente al dispositivo correspondiente al *stream*

• Como norma general, a cada *stream* le corresponde un buffer desde el momento en que se crea o abre



• Mejores tiempo de acceso a los *stream*, sobre todo cuando está relacionado con un fichero en disco

• Un *buffer* en C++ es un objeto de la clase *streambuf* o alguna de sus subclases

• Los métodos de *streambuf* permiten añadir, extraer y posicionar el puntero de L/E

• **rdbuf**(): Devuelve un puntero al *streambuf* correspondiente al *stream*

• *void* **setbuf**(*char*, int*): Permite cambiar el *buffer* por defecto del *stream*

Los *streams* y los *buffers (ii)*

• **LECTURA DE BUFFER**

• **in_avail**(): Devuelve un entero indicando el número de caracteres que hay actualmente en el *buffer* de entrada, tomados desde el *stream* de lectura

• **sbumpc**(): Lee un caracter del *buffer* de entrada, lo devuelve y avanza una posición

• **snextc**(): Avanza una posición, lee un caracter del *buffer* y lo devuelve

• **sgetc**(): Lee el caracter actualmente apuntado en el buffer de entrada y lo devuelve, pero no avanza al siguiente

• **stossc**(): Avanza al siguiente carácter, pero no lee el actual ni lo devuelve

• *int* **sgetn**(*char*, int n*): Lee el número indicado de caracteres del *buffer* de entrada, almacenándolos en la posición indicada por el puntero

• *int* **sputbackc**(*char*): Devuelve el carácter que se le pasa como parámetro al *buffer* de entrada

Los *streams* y los *buffers (iii)*

• **ESCRITURA EN BUFFER**

• **out_waiting()**: Devuelve un entero indicando el número de caracteres que hay actualmente en el *buffer* de salida, esperando a escribirse en el *stream*

• *int* **sputc(***int***)**: Permite añadir el caracter que se le pasa como parámetro al *buffer* de salida

• *int* **sputn(***const char*, int n***)**: Permite escribir un determinado número de caracteres en el *buffer* de salida

• **SINCRONIZACIÓN DE STREAMS**

• Se puede conectar cualquier *stream* de entrada a uno de salida para que cuando se vaya a realizar una entrada, se vuelque el *buffer* del *stream* de salida hacia su destino

• *ostream** **tie(***ostream****)**: Concatena el *stream* sobre el que se llama el método con el *stream* que se le pasa como parámetro. Devuelve el *stream* previamente conectado

• *cin*.**tie**(*cout*): Por defecto. Conecta el objeto de entrada *cin* al objeto de salida *cout*

Los *streams* y los *buffers (iv)*

• **POSICIONAMIENTO EN EL BUFFER**

•*streampos* **seekpos(***streampos, int* = (*ios::in* | *ios::out*)**)**:

Facilita el posicionamiento del puntero en una localización absoluta. El segundo parámetro permite indicar el puntero que se va a fijar: lectura o escritura

• *streampos* **seekoff(***streamoff offset, ios::seek_dir, int mode***)**:

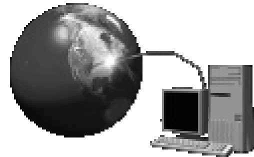
Posiciona el puntero de forma relativa, según el segundo parámetro que puede ser *ios::beg*, *ios::cur* o *ios:end*

El tercer parámetro pude tomar los valores del método anterior

El primer parámetro puede ser un long negativo, desplazándose hacia atrás

Stream9.cpp

Streams
en
Disco



Streams en disco

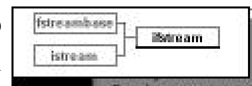
• Un *stream* en disco puede ser tanto de entrada como de salida, pudiendo realizar operaciones en los dos sentidos, algo que no es posible con *cout* y *cin*

• El objeto que permite manejar un *stream* en disco pude ser creado a partir de tres clases distintas, definidas en el archivo de cabecera *fstream.h*:

- *ifstream*: *Stream* de entrada asociado a un fichero en disco

open(), close(), rdbuf(), setbuf()

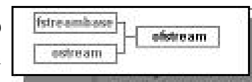
operadores de extracción



- *ofstream*: *Stream* de salida asociado a un fichero en disco

open(), close(), rdbuf(), setbuf()

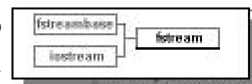
operadores de inserción



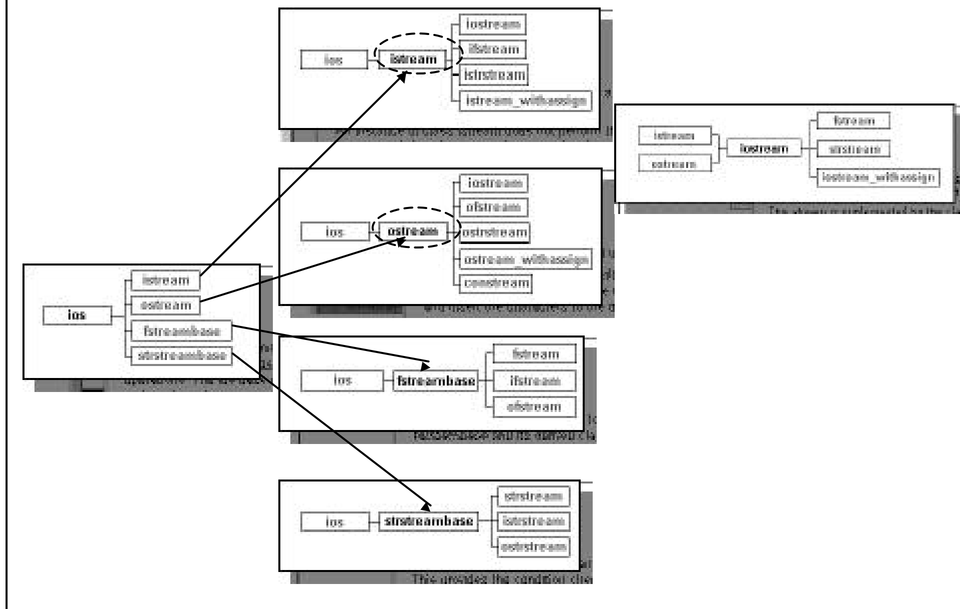
- *fstream*: *Stream* que permite tanto entradas como salidas al fichero asociado

open(), close(), rdbuf(), setbuf()

operadores de extracción e inserción



Jerarquía de clases



Apertura de un stream en disco

• Se puede llevar a cabo en el momento de la creación del objeto (constructor) o bien después, mediante el método **open()**

• *void* **open(***const char *name, int mode, int prot=filebuf::openprot***)**

<u>*name*</u>: nombre del fichero

<u>*mode*</u>: establece el modo de apertura (definido como una enumeración en la clase *ios*)

<u>*openprot*</u>: permisos de acceso DOS. Usado si no se especifica *ios::nocreate* en *mode*

• *in*: ______ Apertura para entrada

• *out*: ______ Apertura para salida

• *app*: ______ Apertura para añadir

• *ate*: ______ Posición inicial del puntero al final del fichero

• *trunc*: ______ Eliminar el contenido del fichero si ya existe

• *noreplace*: ______ Abrir fichero inexistente, no sustituir

• *nocreate*: ______ Abrir fichero existente, no crearlo

• *binary*: ______ Abrir fichero en modo binario, no de texto

Apertura de un stream en disco

• **EJEMPLOS (i)**

// Abre fichero existente para lectura, genera error si el fichero no existe

• **ifstream** F("Fichero");

// Abre fichero para escritura creándolo, si existe trunca su contenido

• **ofstream** F("fichero");

// Abre fichero para lectura/escritura, si existe no lo trunca, si no existe lo crea

• **fstream** F("fichero", **ios::int | ios::out**);

// Abre fichero para escritura, si ya existe genera error, no lo trunca

ofstream F("fichero", **ios::noreplace**);

// Abre fichero existente para lectura/escritura, si no existe no lo crea y genera error

• **fstream** F("fichero", **ios::nocreate**);

// Abre fichero para lectura/escritura en modo binario, si existe trunca el contenido

• **fstream** F("fichero", **ios::trunc | ios::binary**);

Apertura de un stream en disco

• **EJEMPLOS (ii)**

• Es posible crear un objeto *stream* sin asociarlo directamente a un fichero en disco para posteriormente utilizar el método **open()**

• El cierre del fichero normalmente se deja en manos del destructor (cuando el objeto se quede fuera de ámbito o se destruya explícitamente)

• *void* **close()**: Cierra el fichero y el *filebuf* asociado

```
ifstream F; // Se crea el stream
char Nombre [ 12 ] ;

cout << "Archivo a mostrar: " ;
cin >> Nombre ;

F.open(Nombre);
```

• **fstream(***int fd***)**: Constructor que permite crear un *stream* a partir de un descriptor de fichero ya abierto

• **fstream(***int fd, char *buf, int n***)**:

Se especifica el buffer a utilizar y su tamaño. Si *buf* es NULL ó *n* < 0 => SIN BUFFER

Stream10.cpp

Stream11.cpp

Stream en disco y *buffers*

• Cuando se abre un *stream* en disco se asigna un *buffer* por defecto, aunque es posible especificar el *buffer* que se va a utilizar (último constructor visto)

• Se disponde de **rdbuf()** y **setbuf()** para obtener y asignar *buffers*

• El puntero devuelto por el método **rdbuf()** es un objeto de la clase *filebuf* que deriva de *streambuf*

• El objeto de la clase *filebuf* devuelto por **rdbuf()** contiene dos nuevos métodos:

  • *int* **fd()**: Devuelve el descriptor del fichero asociado con el *stream* o EOF

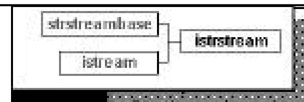
  • *int* **is_open()**: Devuelve un valor distinto de cero si el fichero está abierto

• Se debe tener precaución cuando se manipulan los *buffers* de los *streams* en disco, puesto que puede suceder que la información no llegue a su destino

filebuf .cpp

Streams en Memoria

Streams en memoria (E)



• La información se almacena y se extrae utilizando la memoria como origen y destino de la información

• Disponen de todas las funcionalidades vistas hasta ahora, excepto de los métodos de apertura y cierre del *stream*

• Un *istrstream* se utiliza para realizar sobre él operaciones de lectura, por lo que inicialmente tendrá que tener algún contenido

• **istrstream(***unsigned char ****)**;
• **istrstream(***char ****)**;
• **istrstream(***signed char ****)**;

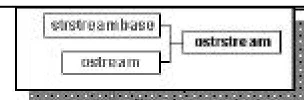
El puntero a caracter especifica el *buffer* del *stream*, que contendrá información en el momento de su creación

• **istrsteam(***signed char *str, int n***)**;
• **istrstream(***char *str, int n***)**;
• **istrstream(***unsigned char *str, int n***)**;

Permite especificar el tamaño de la cadena que se utilizará como *buffer*

Stream12.cpp

Streams en memoria (S1)



• Un objeto de la clase *ostrstream* puede crearse con un buffer inicial, que a priori puede o no tener información

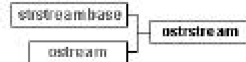
• **ostrstream(***char *buf, int len, int mode = ios::out***)**;
• **ostrstream(***signed char *buf, int len, int mode = ios::out***)**;
• **ostrstream(***unsigned char *buf, int len, int mode = ios::out***)**;

buf es la dirección del *buffer*

len especifica la longitud del *buffer*

mode **ios::out** (sobrescribe el contenido del buffer), **ios::app** o **ios:ate** (puntero al final)

Streams en memoria (S2)



• Puede crearse un objeto de la clase *ostrstream* con un *buffer* dinámico que va creciendo a medida que es necesario

• **ostrstream()**: Crea un *stream* con un *buffer* dinámico

• El *buffer* se va asignando de forma dinámica y hay que liberarlo explícitamente antes de destruir el objeto *ostrstream*

• *ostrstream* dispone de 2 métodos a mayores de los comentados anteriormente:

  • *int* **pcount()**: Devuelve un entero indicando el número de caracteres actualmente almacenados en el stream ⇔llamada a **out_waiting()** del *buffer* asociado

  • *char* ***str()**: Devuelve un puntero a carácter con la dirección dónde está almacenada la información del *stream*. El usuario debe liberarlo si el *buffer* es dinámico

Streams en memoria y *buffers*

• Cuando se crea un *stream* en memoria se asigna un *buffer* (dinámico o especificado)

• Se dispone de los métodos **rdbuf()** y **setbuf()**. El puntero que devuelve **rdbuf()** es la dirección de un objeto de la clase *strstreambuf* (derivada de *streambuf*)

• **strstreambuf** dispone de 2 nuevos métodos:

  • *void* **freeze**(*int = 1*): Bloquea (1) / desbloquea (0) la lectura en el *stream*

  • *char* ***str()**: Devuelve un puntero a carácter al contenido del *stream*

Histo.cpp

Control de estado del *stream* (i)

• Independientemente del tipo de *stream* se dispone de una serie de indicadores que controlan su estado

• Es posible obtener el estado de un *stream* por medio del método **rdstate()**, que devuelve un entero en el que estarán activados los bits indicadores:

• *goodbit*: Indica que no hay error

• *eofbit*: Indicador de fin de fichero

• *failbit*: Indica que la última operación ha fallado

• *badbit*: Indica que la última operación no es válida

• *hardfail*: Indicador de error irrecuperable

• El único error sin solución es *hardfail*. Si este indicador se activa, lo más probable es que el *stream* haya quedado inservible

Control de estado del *stream* (ii)

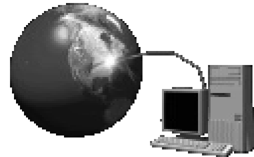
• Estos indicadores se pueden comprobar por medio de operaciones lógicas o por métodos definidos para ello:

- *int* **good()**: Devuelve un valor distinto de 0 si está activado *goodbit*
- *int* **eof()**: Devuelve un valor distinto de cero si está activado *eofbif*
- *int* **bad()**: Devuelve un valor distinto de cero si está activado *badbit* o *hardfail*
- *int* **fail()**: Devuelve un valor distinto de cero si está activado *failbit*, *badbit* o *hardfail*

• *void* **clear(***int = 0***)**: Permite activar los indicadores de estado. Sin parámetros borra su valor

• Es posible comprobar el estado de un *stream* con el operador de negación, que está sobrecargado para este fin

Sobrecarga de los operadores de inserción y extracción



Sobrecarga de << y >>

• 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 la *stream* sobre el que trabaja => encadenamiento de varias operaciones

• Si los datos forman parte de una clase, lo más normal es que los operadores sobrecargados se creen como *friend* de la clase para que puedan manipular sus datos

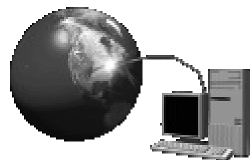
• **friend** *ipstream&* **operator** >> (*ipstream& is, string& str*);

• **friend** *opstream&* **operator** << (*opstream& ps, signed char ch*);

• 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

Stream14.cpp

Definición de manipuladores



Definición de manipuladores

• Además de los manipuladores existentes, se puede definir cualquier otro que se necesite

• Útil porque se ahorran las interrupciones de la cadena para llamar a un método y a continuación seguir con la inserción o extracción

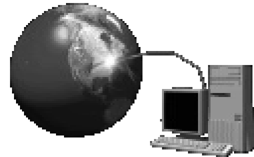
• El tipo de *stream* que recibe y devuelve un manipulador como parámetro será *ostream* o *istream* para extracción e inserción respectivamente

```
ostream& tab ( ostream& ptrStream )
{
     return ptrStream << '\t';
}
```

• Se pueden definir manipuladores que reciban parámetros

Stream15.cpp

Cadenas en C++ (strings)



string

• El lenguaje C no dispone de ningún tipo de dato cadena => Se utilizan los **char ***
(arrays de caracteres terminados con un caracter nulo '\0')

• C++ incorpora las cadenas como objetos de tipo **string**, los cuales disponen de un buen número de métodos listos para su ejecución:

Introducción: método length()	String01.cpp
Asignación y concatenación	String02.cpp
Operadores booleanos	String03.cpp
Un problema grave con las cadenas de C	String04.cpp
Extracción de una subcadena de un *string*	String05.cpp
Encontrar una subcadena dentro de un *string*	String06.cpp

REDE GALEGA de BIOINFORMÁTICA

ESTRUCTURAS DE DATOS Y C/C++

<<Streams>> y Strings en C++

Dr. *Florentino Fernández Riverola*

Escuela Superior de Ingeniería Informática

Departamento de Informática

Universidad de Vigo

riverola@uvigo.es