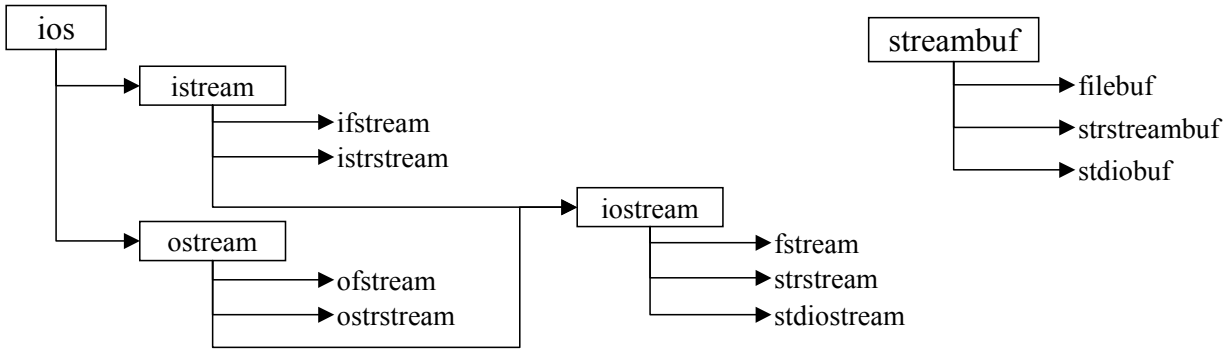
Entrada / Salida de Datos en C++

Informática II
Fundamentos de Programación

Entrada/Salida de datos en C++

- Basadas en "***clases***" y en la "***herencia***" (fáciles de extender y modificar).
- ***Flujo o stream***: dispositivo que produce o consume información
  - Un ***flujo*** está siempre ligado a un dispositivo físico
  - Todos los ***flujos*** se comportan de forma análoga
  - ***Flujos*** abiertos en todo programa:
    - **cin**: entrada estándar (teclado)
    - **cout**: salida estándar (pantalla)
    - **cerr**: salida de mensajes de error (pantalla)
- C++ dispone de dos jerarquías de clases:
  - De bajo nivel: ***streambuf*** (para usuarios más avanzados)
  - De alto nivel: ***istream***, ***ostream*** y ***iostream***, que derivan de ***ios***
  - Estas clases disponen de *variables* y *métodos* para controlar los ***flujos*** de entrada y salida



Entrada/Salida con formato

- ***Indicadores***: variables miembro *enum* de tipo ***long*** que controlan el formato al activarse o desactivarse alguno de sus bits. Su valor hexadecimal y su significado es:

```
enum {
   skipws=0x0001,   left=0x0002,   rigth=0x0004,   internal=0x0008,   dec=0x0010,
   oct=0x0020,      hex=0x0040,    showbase=0x0080, showpoint= 0x0100, fixed=0x1000,
   showpos=0x0400,  scientific=0x800, unitbuf=0x2000, uppercase=0x0200
};
```

skipws:	se descartan los blancos iniciales a la entrada
left:	la salida se alinea a la izquierda
rigth:	la salida se alinea a la derecha
internal:	se alinea el signo y los caracteres de base a la izda y las cifras a la dcha
dec:	salida decimal para enteros (defecto)
oct:	salida octal para enteros
hex:	salida hexadecimal al para enteros
showbase:	se muestra la base de los valores numéricos
showpoint:	se muestra el punto decimal
uppercase:	los caracteres de formato aparecen en mayúsculas
showpos:	se muestra el signo (+) en los valores positivos
scientific:	notación científica para coma flotante
fixed:	notación normal para coma flotante
unitbuf:	salida sin buffer (se vuelca cada operación)

- Existen unos ***indicadores*** adicionales (***adjustfield***, ***basefield*** y ***floatfield***) que actúan como combinaciones de los anteriores.

Activar y desactivar indicadores

- Activación de ***indicadores***: método ***setf()*** de ***ios***:
  - Su prototipo es:

```
long setf(long indic);
```

    el valor de retorno es la configuración anterior; ***indic*** es el ***long*** que contiene los indicadores que se deesean establecer
  - Se permite activar varios indicadores a la vez con el operador OR binario. Ejemplo:

```
cout.setf(ios::showpoint | ios::fixed)
```

  - es necesario determinar el flujo afectado (***cout***) y la clase del indicador (***ios***)
- Para desactivar los indicadores se utiliza la función ***unsetf()*** de modo similar a ***setf()***
- La función ***flags()*** devuelve un ***long*** con la configuración de todos los indicadores:
  - Su prototipo es:

```
long flags();
```

  - Existe otra definición que permite cambiar todos los indicadores a la vez pasando un ***long*** como argumento:

```
long flags(long indic);
```

    el valor de retorno es un ***long*** con la configuración anterior
- La función ***setf()*** cambia sólo los indicadores que se le pasan como argumentos, mientras que ***flags()*** cambia la configuración por completo

Funciones *width()*, *precision()* y *fill()*

- Función miembro ***width()***: establece la anchura de campo mínima para un dato de salida
  - Su prototipo es: `int width(int n);`
    donde el valor de retorno es la anchura anterior.
  - La anchura establecida es la mínima. Siempre que sea necesario el sistema la aumenta de modo automático.
- Función miembro ***precision()***: establece el número de cifras para un dato de salida
  - Su prototipo es: `int precision(int n);`
    donde el valor de retorno es la precisión anterior.
  - La precisión por defecto es 6 dígitos.
- Función miembro ***fill()***: establece el carácter de relleno para un dato de salida
  - Su prototipo es: `char fill(char ch);`
    donde el valor de retorno es el carácter de relleno anterior.
  - Por defecto el carácter de relleno es el blanco ' '.
- El efecto de ***precision()*** y ***fill()*** es permanente. ***width()*** debe ser llamada para cada dato.

```
#include <iostream.h>
void main()
{
  double coef[]={5198.0,3.21,46.32,506.5};
  char *prov[] ={"Madrid","Guipuzcoa",
                 "Cantabria","Alava"};
  //salida alineados
  const int formato = ios::fixed | ios::left;
  cout.flags(formato);
  cout.fill('.');    //carac. relleno
  for (int i=0;i<sizeof(coef)/sizeof(double);i++){
      cout.width(15);    //ancho para texto
      cout << prov[i];       //escribe texto
      cout.unsetf( ios::left);//suprime just. izq.
      cout.width(10);    //ancho para cantidades
      cout.precision(2); //2 decimales
      cout << coef[i] << endl;
      cout.setf(ios::left);
    }
    cout.unsetf(ios::left);
    cout.setf(ios::scientific|ios::right );
    cout.width(25);      //ancho para texto
    cout << coef[0]<<endl;
}//fin de main
```

Ejercicio 1

Manipuladores de entrada/salida

- Los ***manipuladores*** son una alternativa a los ***indicadores***. Se pueden introducir en la propia sentencia de entrada o salida
  - Los manipuladores pueden tener argumentos o no tenerlos. Si los tienen hay que incluir el fichero ***iomanip.h***
  - Un manipulador sólo afecta al flujo (***cin***, ***cout***, etc.) al que se aplica
- El manipulador ***setiosflag()*** equivale al indicador ***setf()***, y ***unsetiosflag()*** equivale a ***unsetf()***
- Algunos manipuladores de entrada/salida
  - ***dec***, ***hex*** y ***oct***: Establecen base para enteros
  - ***ws***: se saltan los blancos iniciales
  - ***endl***: se imprime un **'\n'** y se vacía el buffer de salida
  - ***flush***: se vacía el buffer de salida
  - ***setw(int w)***: establece la anchura mínima de campo (sólo para el siguiente dato)
  - ***setprecision(int p)***: establece el número de cifras
  - ***setfill(char ch)***: establece el carácter de relleno
- Ejemplos de uso de los manipuladores:

```
cout << hex << 100;
cout <<  setw(10) << mat[i][j] << endl;
```

- El efecto de los manipuladores permanece en el flujo correspondiente hasta que se cambian con otro manipulador (excepto *setw()*)

Entrada/Salida de ficheros

- Los ***ficheros*** se utilizan para la lectura y/o escritura de datos en unidades de almacenamiento permanente como los disquetes, discos duros, etc.
- Las clases necesarias para la utilización de ficheros son ***ifstream***, ***ofstream*** y ***fstream***, que derivan de ***istream*** y ***ostream***, que a su vez derivan de la clase ***ios*** (ver figura). Para utilizarlas se debe incluir el fichero <**fstream.h**>.
- Antes de abrir un fichero hay que crear un ***flujo*** o ***stream***, es decir un *objeto* de las clases ***ifstream***, ***ofstream*** o ***fstream*** e indicar el modo de apertura (lectura, escritura, …).
- **Clase ofstream**:
  - Es una clase derivada de ***ostream***, especializada en manipular ficheros en el disco abiertos ***para escribir***.
  - Al construir un objeto de esta clase, el constructor lo conecta automát. con un objeto ***filebuf*** (un *buffer*).
  - La funcionalidad de esta clase está soportada por las siguientes funciones miembro, entre otras:
    - **ofstream(const char *_nombre_fichero_, int _modo_=ios::out, int _proteccion_=filebuf::openprot);**
    - **void open(const char *_nombre_fichero_, int _modo_=ios::out, int _proteccion_=filebuf::openprot);**
    - **void close();** //esta función cierra el fichero
    - **int is_open();** //verifica si el fichero está abierto(=1). Si no lo está devuelve un 0.

```
char file[]="Prueba.txt";
ofstream fout(file);
if (!fout) cerr<<"No se puede abrir "<<file;
cout << "Fichero "<< file << " abierto bien"
fout.close();
```

```
char file[]="Prueba.txt";
ofstream fout;
fout.open(file);
if (!fout) cerr<<"No se puede abrir "<< file;
cout << "Fichero "<< file << " abierto bien";
fout.close();
```

• Para ***escribir*** en el fichero se utiliza el operador de inserción "<<" sobrecargado. Para ***leer*** del fichero se usa el operador de extracción ">>". Esta forma de Escritura es sólo en formato texto.
• El Ejercicio 1 (slide 5) sustituir ***cout*** por ***fout***. El efecto es el mismo, sólo que en un fichero.

Entrada/Salida de ficheros

- **Clase ifstream**:
  - Es una clase derivada de ***istream***, especializada en manipular ficheros en el disco abiertos ***para leer***.
  - Al construir un objeto de esta clase, el constructor lo conecta automáticamente con un objeto ***filebuf*** (un *buffer*).
  - La funcionalidad de esta clase está soportada por las siguientes funciones miembro, entre otras:
    - **ifstream(const char *_nombre_fichero_, int _modo_=ios::in, int _proteccion_=filebuf::openprot);**
    - **void open(const char *_nombre_fichero_, int _modo_=ios::in, int _proteccion_=filebuf::openprot);**
    - **void close();** //esta función cierra el fichero
    - **int is_open();** //verifica si el fichero está abierto(=1). Si no lo está devuelve un 0.

```
#include <fstream.h>
void main()
{
      int x=20;
      char file[]="Prueba.txt";
      ofstream file1(file);
      if (!file1) cerr << "No se puede abrir " << file;
      cout << "Fichero "<< file << " abierto bien" <<endl;
      file1 << "Hola" << endl << "Me despido " << "Adios ";
      file1 << hex << x;
      file1.close();

      char tmp[120];
      ifstream file2(file);
      if (!file2) { cerr << "No se puede abrir " << file; return;}
      cout << "Vamos a leer de " << file << endl;
      while(!file2.eof()){
            file2 >> tmp;
            cout << tmp << " ";
      }
      file2.close();
}
```

Entrada/Salida de ficheros

Clase fstream:

- Es una clase derivada de ***iostream***, especializada en manipular ficheros en el disco abiertos ***para leer y/o escribir***.
- Al construir un objeto de esta clase, el constructor lo conecta automáticamente con un objeto ***filebuf*** (un *buffer*).
- La funcionalidad de esta clase está soportada por las siguientes funciones miembro, entre otras:
  - **fstream(const char** **nombre_fichero*, **int** *modo*, **int** *proteccion*=**filebuf::openprot);**
  - **void open(const char** **nombre_fichero*, **int** *modo*, **int** *proteccion*=**filebuf::openprot);**
  - **void close();** //esta función cierra el fichero
  - **int is_open();** //verifica si el fichero está abierto(=1). Si no lo está devuelve un 0.

```
//generar datos y escribir en formato texto
double PI=3.141592,val;
fstream fout("datos.txt",ios::out);
if (!fout) { cerr << "Error"; return; }
for (double i=0;i<2*PI;i+=PI/10)   //20 ptos
{
   val=5.0*sin(i);
   fout << val << endl;  //graba en formato texto
}
fout.close();
cout << "Datos escritos bien. Ahora a leer.." << endl;
//leer datos
fstream fin("datos.txt",ios::in);
if (!fin) { cerr << "Error"; return; }
cout.setf(ios::fixed|ios::showpos);  //con pto. decimal
cout.precision(3);
while( !fin.eof()){
   fin >> val;
   cout << val << endl;
}
fin.close();
```

Si el fichero se abre con el modo:

ios::app

Entonces, todo lo que se escriba se agregará a partir del final del fichero.

Entrada/Salida de ficheros

- Otras posibilidades de leer y escribir en un fichero son:
  - ***getline()***: lee de fichero un número de caracteres especificado en la variable ***nCount*** o hasta que encuentre el carácter fin de línea '\n'. Devuelve un NULL cuando encuentra el final del fichero. Su prototipo es:

    ```
    istream& getline(unsigned char* puch, int nCount, char dlm = '\n');
    ```

  - ***read() y write():*** leen y escriben, respectivamente, bloques de bytes o ***datos binarios***. Sus prototipos son:

    ```
    istream& read( unsigned char* bif, int num);
    ostream& write( unsigned char* bif, int num);
    ```

- Ejemplo:

```
#include <fstream.h>
#include <iostream.h>
void main()
{
   char frase[81];
   fstream fichero;
   fichero.open("datos.txt", ios::in);
   while(fichero.getline(frase,80) != NULL)
       cout << frase;
}
```

```
//generar datos binarios
double PI=3.141592,val;
fstream fout("datos.dat",ios::out);
if (!fout) { cerr << "Error"; return; }
for (double i=0;i<2*PI;i+=PI/10){ //20 ptos
   val=5.0*sin(i);
   fout.write((char*)&val,sizeof(double));
}
fout.close();
//leer datos binarios
fstream fin("datos.dat",ios::in);
if (!fin) { cerr << "Error"; return; }
cout.setf(ios::fixed|ios::showpos);
cout.precision(3);
while(true){
   if (!fin.read((char*)&val,8) break;
   cout << val*2 << endl;
}
fin.close();
```

Entrada/Salida de ficheros

Acceso Aleatorio:

- Hasta ahora hemos estado trabajando con ficheros de ***acceso secuencial***: leer o escribir desde el inicio del fichero o escribir a partir del final.
- El acceso aleatorio de ficheros permite leer o escribir a partir de una determinada posición del fichero. Esto tiene una gran ventaja, ya que se pueden modificar algunos de los valores contenidos en el fichero.
- C++ nos da unas funciones para el acceso aleatorio:

  Para la ***clase istream***:
  - `istream &seekg(streamoff desp, ios::seek_dir pos);`
  - `streampos tellg();`

  Para la ***clase ostream***:
  - `ostream &seekp(streamoff desp, ios::seek_dir pos);`
  - `streampos tellp();`

  Donde: ***streampos*** es un typedef de long.

  desp es la nueva posición, desplazada ***desp bytes***, desde la posición dada por ***pos***, el cual puede ser:

ios::beg	Principio del fichero
ios::cur	Posición actual del puntero del stream
ios::end	Final del stream.
seekg	se usa para desplazarse en un fichero para lectura
seekp	se usa para desplazarse en un fichero para escritura
tellg, tellp	dan la posición actual del puntero de lectura y escritura, respectivamente.

- Para escribir en un fichero de acceso aleatorio, éste debe ser abierto de modo lectura/escritura, usando para ello: **`ios::in | ios::out`**

Ejemplo de acceso Aleatorio

```
#include <fstream.h>
#include <math.h>
void main()
{
        cout << "Cambiar datos" << endl;
        //cambiar un dato con acceso aleatorio
        fstream fout;
        fout.open("datos.dat",ios::out|ios::in);
        if (!fout) { cerr << "Error"; return; }
        fout.seekp(8*2,ios::beg);       //desde el inicio
        val=25.5;
        fout.write((char*)&val,sizeof(double));
        fout.seekp(8*2,ios::cur);       //desde pos. actual
        val=15.5;
        fout.write((char*)&val,sizeof(double));
        fout.close();

        //leer datos binarios
        fstream fin("datos.dat",ios::in);
        if (!fin) { cerr << "Error"; return; }
        cout.setf(ios::fixed|ios::showpos);  //con pto. decimal
        cout.precision(3);
        //ir al segundo elemento
        //fin.seekg(8*2,ios::beg);      //desde el inicio
        while(true){
                if ( !fin.read((char*)&val,sizeof(double))) break;;
                cout << val << endl;
        }
        fin.close();
}
```

Leer y Escribir Objetos

- Para leer y escribir objetos en formato binario, se deben sobrecargar los operadores de extracción “>>” e inserción “<<“, en los cuales pondremos el código necesario para usar las funciones ***read*** y ***write*** de las clases ***ofstream***, ***ifstream*** o ***fstream***.
- Veamos a continuación el ejemplo con la clase ***Complejo***:

```
class Complejo
{
private:
     float r,i;
public:
     Complejo(float =0, float =0);
     void Asignar(float =0, float =0);
     friend fstream &operator<<(fstream &fo, const Complejo &c);
     friend ifstream &operator>>(ifstream &ci, const Complejo &c);
     friend ostream &operator<<(ostream &co, const Complejo &c);
};
void Complejo::Asignar(float pr, float pi)
{
     r = pr;  i = pi;
}
Complejo::Complejo(float pr, float pi){
     r = pr; i=pi;
}
ifstream &operator>>(ifstream &ci, const Complejo &c){
     ci.read( (char*)&c,sizeof(c));
     return ci;
}
fstream &operator<<(fstream &fo, const Complejo &c){
     fo.write( (char *)&c, sizeof(c));
     return fo;
}
ostream &operator<<(ostream &co, const Complejo &c){  //para cout
     return(co << c.r << "," << c.I);
}
```


Leer y Escribir Objetos (cont.)

```
#include <fstream.h>
#include <stdlib.h>
#include “complejo.h”
void Escribir();
void Leer();

void main()
{
     char ch;

     cout << "Leer o escribir (l/e)?:";
     cin >> ch;
     if (ch == 'l' || ch=='L') Leer();
     else Escribir();
}
void Leer()  //leer objetos
{
     Complejo c;
     char file[80];

     cout << "Nombre del fichero:";
     cin >> file;
     ifstream fin(file);
     if ( !fin) { cerr << "Error al abrir fichero"; return; }
     int num=0;
     while (true){
          fin >> c;
          if ( fin.eof() ) break;
          cout << "C["<< num <<"]=" << c << endl;
          num++;
     }
     fin.close();
     cout << "Complejos leidos:" << num << endl;

}//fin de Leer()
```

```
void Escribir()
{
     Complejo *c;
     int num;
     char file[80];
     cout << "Nombre del fichero:"; cin >> file;
     cout << "Numero de complejos:" ; cin>>num;
     c = new Complejo[num];
     for (int i=0;i<num;i++) {
          float x = (float)rand()/10000;
          float y = (float)rand()/10000;
          c[i].Asignar(x,y);
     }
     fstream fout(file,ios::out);
     if ( !fout) { cerr << "Error al abrir fichero";
          return; }
     for (i=0;i<num;i++) fout << c[i];
     fout.close();
     delete [] c;
     cout << "Datos escritos en: " << file << endl;
}//fin de Escribir
```