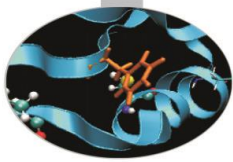
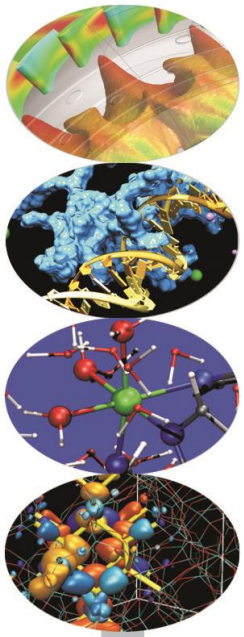
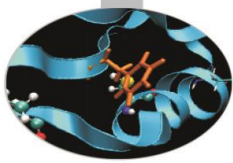


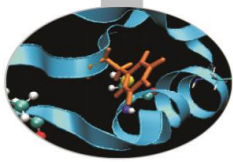
I/O da FILE



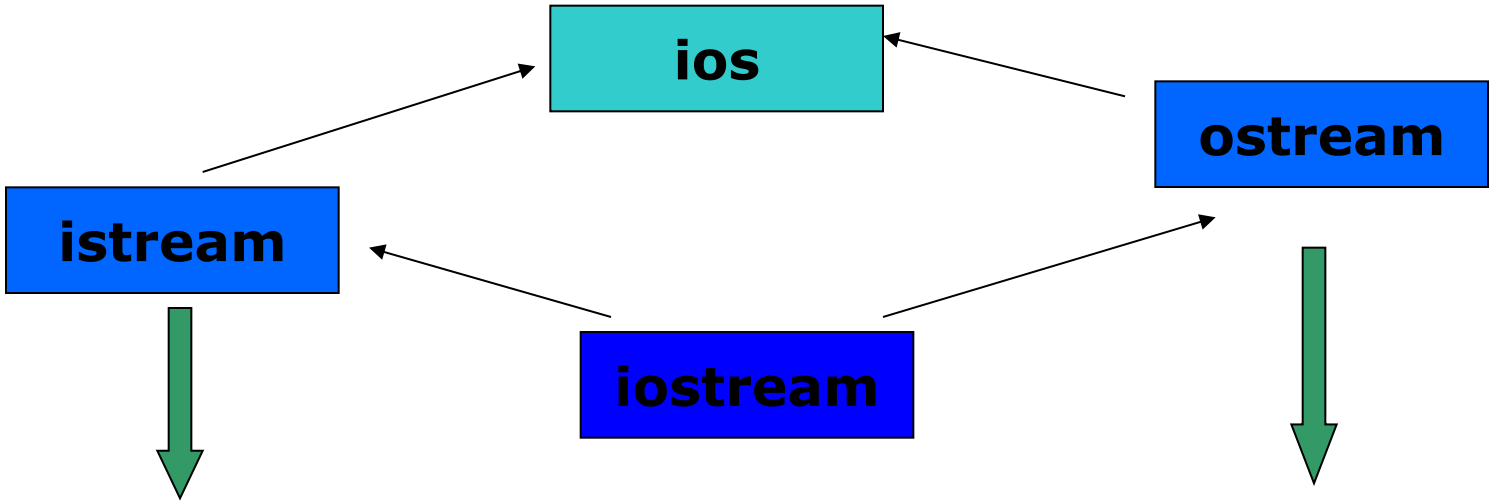
Gli stream



- Alla base delle operazioni di I/O, in C++, sta il concetto di stream (flusso) di byte tra la memoria principale ed i dispositivi di input (la tastiera, i file di sola lettura) e quelli di output (il video, i file di scrittura).
- Possiamo distinguere tra due modalità di I/O: formattato, ovvero ad alto livello, leggibile dall'utente e non formattato cioè a basso livello, comprensibile solo dalla macchina.
- La modalità non formattata è preferibile quando si debba trattare con grandi moli di dati.



Gli stream

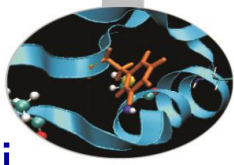


libreria **<istream>,** contiene la dichiarazione dell'oggetto **cin** per la lettura da standard input.

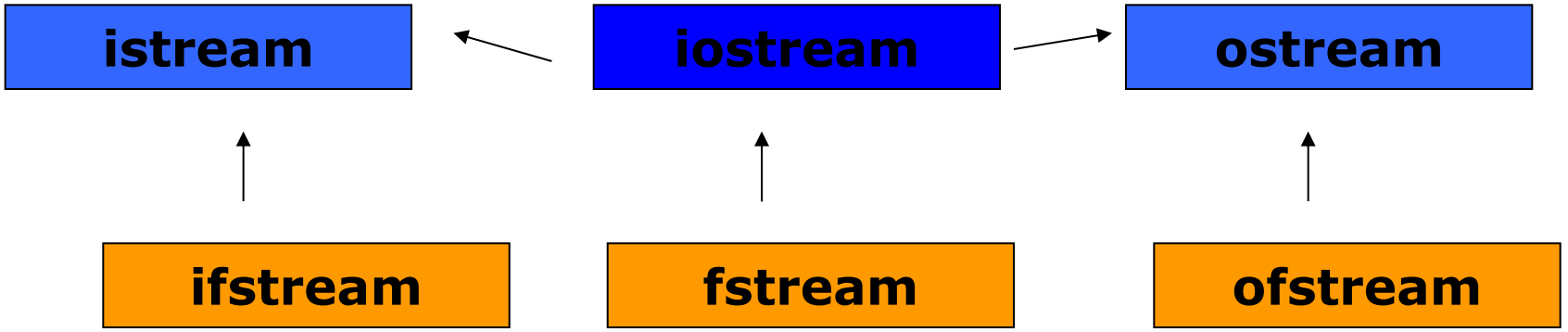
libreria **<ostream>,** ha in sé la dichiarazione degli oggetti **cout, cerr** e **clog** per la scrittura su standard output e standard error.

libreria **<iostream>**, contiene la dichiarazione di tutti e quattro gli oggetti sopra citati. E' l'unica libreria da includere nel codice per eseguire operazioni di I/O su standard device.

Gli stream

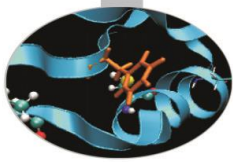


La classe **fstream** gestisce operazioni sia di input che di output da file.



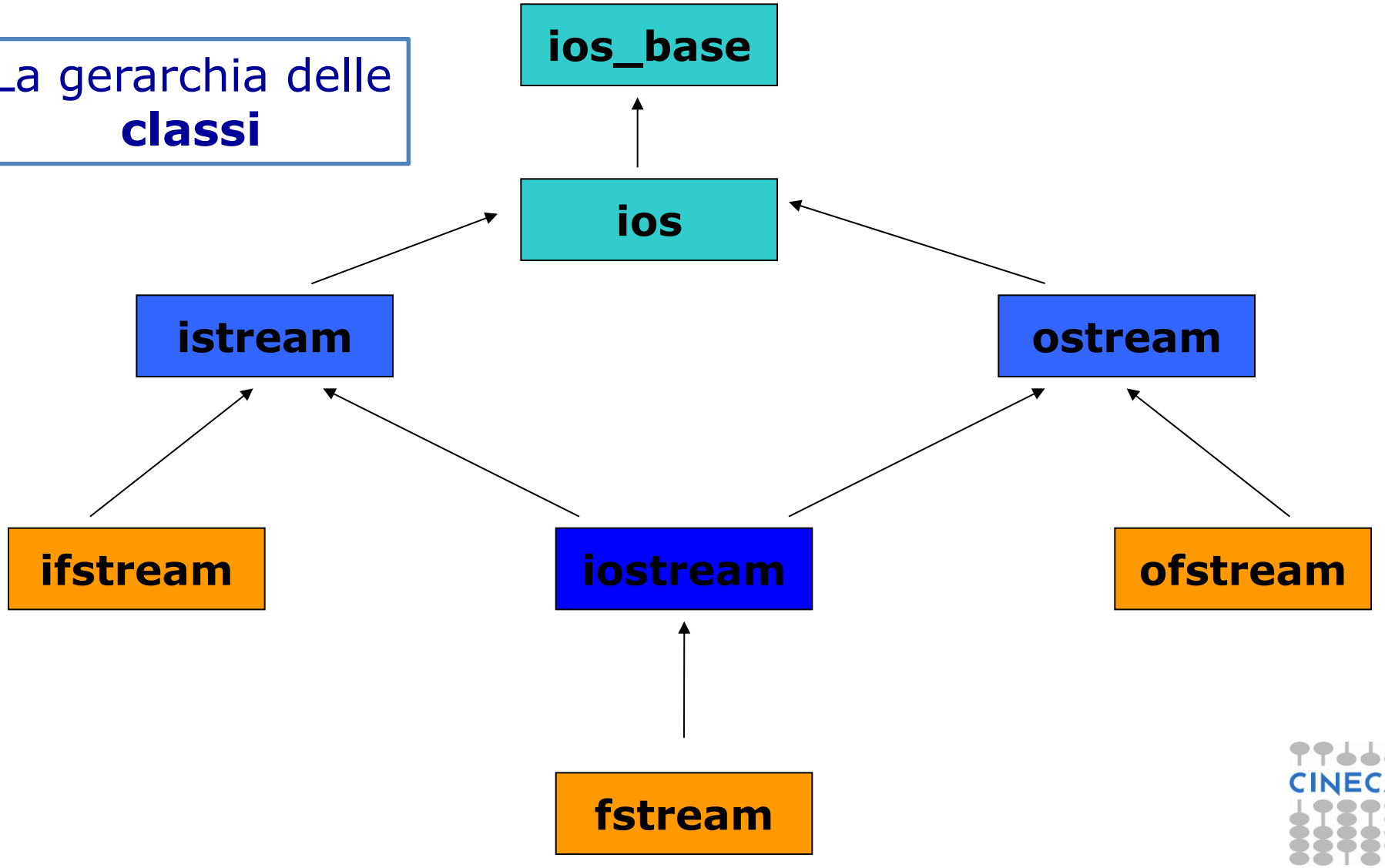
La classe **ofstream** gestisce operazioni di output su file
La classe **ifstream** gestisce operazioni di input da file

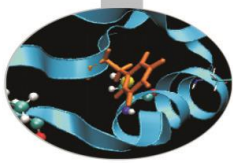
libreria **<fstream>**. Da includere nel codice per eseguire ogni operazione di I/O su file. Non contiene la dichiarazioni di particolari oggetti.



Gli stream

La gerarchia delle **classi**





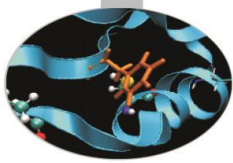
cout

- E' un oggetto della classe ostream; la sua dichiarazione risiede all'interno della libreria <ostream>.
- Tipicamente viene utilizzato insieme con l'operatore di inserimento nello stream `<<`; può far uso del metodo pubblico `put(char)` di ostream, per la scrittura di un singolo carattere su standard output.

- esempio: uso di `<<` e di `put(char)` con `cout`.

```
#include<iostream>
using namespace std;

int main(){
        cout << "Hi!" << endl;
        cout.put('H').put('i').put('!').put('\n');
        return 0;}
```



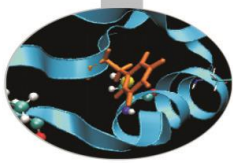
cout

- esempio: uso delle funzioni width, fill e precision .

```
#include<iostream>
using namespace std;
int main(){
   double a=11, b=3;
   for(int i=0; i<5; i++){
        cout.width(10);
        cout.fill('*');
        cout.precision(i+1);
        cout << a/b << endl;
   }
   return 0;
}
```

- output:

```
*********4
*******3.7
******3.67
*****3.667
****3.6667
```

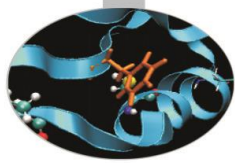


cin

- E' un oggetto della classe istream; la sua dichiarazione è posta all'interno della libreria <istream>.
- Viene utilizzato, di solito, insieme all'operatore di estrazione dallo stream >>, ma può anche chiamare alcuni dei metodi pubblici della classe istream, come: get(), per la lettura di un carattere); getline(char*), per la lettura di una stringa in cui è ammesso anche il carattere di spazio; eof(), che restituisce 1 o 0 (true o false) a seconda che sia stato raggiunto o meno il carattere di End Of File (EOF).
- Per mezzo della funzione width(int) della classe **ios** è possibile definire l'ampiezza del campo di input.

Esempio: uso di get().

```
#include<iostream>
using namespace std;
int main(){
      char c;
      cout << "Insert a sentence:" << endl;
      while( (c=cin.get()) != EOF)
         cout << c;
      return 0;}
```



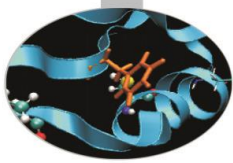
I manipolatori di stream

- Sono funzioni che servono per controllare la formattazione dei dati nelle procedure di I/O e rappresentano un'alternativa alle funzioni della classe ios viste in precedenza.
- Si dividono in due categorie:
  - semplici;
  - parametrizzati.
- Per fare uso dei manipolatori parametrizzati è necessario includere la libreria **<iomanip>**; per i manipolatori semplici è sufficiente <iostream>.
- In generale ogni manipolatore può essere visto come un operando degli operatori << e >> che influenza la stampa o la lettura degli oggetti e delle variabili che li seguono nell'istruzione in cui compaiono.

- esempio:

```
cout << n << " " << hex << n << endl;
cout << n << " " << setbase(16) << n << endl;
```

La libreria <iomanip>



- E' costituita da sei differenti manipolatori parametrici:

  - **resetiosflags(**ios::*nome_flag*)**:** annulla il flag di formattazione ios::*nome_flag* attivato in precedenza;

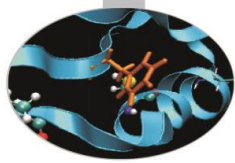
  - **setiosflag(**ios::*nome_flag*)**:** attiva il particolare flag di formattazione ios::*nome_flag*;

  - **setbase(**int)**:** impone la scrittura di numeri in una determinata base. I valori consentiti sono 8, 10 e 16;

  - **setfill(**char)**:** indica il carattere di riempimento per i campi giustificati;

  - **setprecision(**int)**:** determina il numero totale delle cifre (intere e decimali) con cui deve essere scritto un numero reale;

  - **setw(**int)**:** specifica il numero minimo di caratteri da utilizzare nella scrittura della successiva espressione.



La libreria <iomanip>

Esempio: uso di setw, setfill e setprecision.

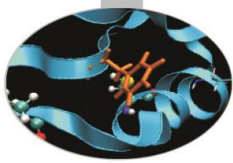
```
#include<iostream>
#include<iomanip>
using namespace std;

int main(){
   double a=11, b=3;
   for(int i=0; i<5; i++){
        cout << setw(10) << setfill('*')
             << setprecision(i) << a/b << endl;
   }
   return 0;
}
```

output:

```
*********4
*********4
*******3.7
******3.67
*****3.667
```

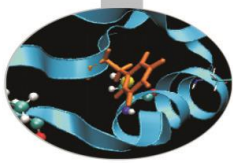
I flag di formattazione



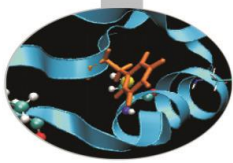
- Sono definiti tramite l'istruzione enum all'interno della classe **ios** (ios_base per i compilatori più datati).
- Vengono usati come argomento da passare a due dei manipolatori parametrici ed alle funzioni della classe ios che si occupano della formattazione dell'output.
- I principali sono:

`ios::skipws`	salta gli spazi bianchi in input
`ios::left`	giustifica a sinistra
`ios::right`	giustifica a destra
`ios::internal`	giustifica il segno di un numero a sinistra ed il suo valore a destra
`ios::dec`	interi in base decimale
`ios::oct`	interi in base ottale
`ios::hex`	interi in base esadecimale
`ios::showbase`	mostra la base di un intero (0 per ottale 0x per esadecimale)

I flag di formattazione



ios::showpoint	punto decimale sempre presente nel formato dei numeri reali
ios::uppercase	lettera maiuscola E nella notazione scientifica e 0X per la base esadecimale
ios::showpos	mostra il segno di un numero
ios::scientific	notazione scientifica
ios::fixed	numero fisso di cifre decimali
ios::floatfield	formato di default per numeri reali(numero variabile di cifre decimali)



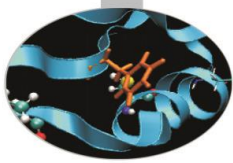
I flag di formattazione

- Esempio2: flag di formattazione e manipolatore `setiosflags`

```
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
   double a=3;
   for(int i=0; i<5; i++){
        cout << setiosflags(ios::showpoint | ios::fixed)
             << setiosflags(ios::right | ios::showpos
                           |ios::internal)
             << (i+1)/a << endl;
   }return 0;
}
```

- output:

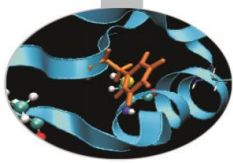
```
+0.333333
+0.666667
+1.000000
+1.333333
+1.666667
```



Creazioni di file

- Un file è trattato dal C++ come uno *stream* (flusso) sequenziale di byte.
- Quando un file viene aperto, ad esso è associato automaticamente uno stream che rappresenta un canale di comunicazione tra il file stesso ed il programma.
- La fine di un file è segnata da un *marcatore di end of file* o da uno specifico numero di byte registrato in una struttura dati gestita dal sistema.
- Per far uso di file è necessario includere all'interno del programma l'header file `<fstream>` che contiene la definizione delle classi `ifstream` (input da file), `ofstream` (output da file) e `fstream` (input/output) da file.
- L'**apertura** di un nuovo file richiede la **creazione di un oggetto di una classe stream**.
- Al costruttore della classe selezionata vengono inviati il nome del file ed, eventualmente, la modalità di apertura che coincide con un metodo della classe **ios**:

```
xfstream nome_oggetto(nome_file,ios::modalità_apertura);
```



Apertura di file

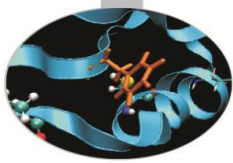
- Ogni file può essere aperto in due modi differenti: uno ricalca la regola generale di sintassi per la creazione di file vista in precedenza, l'altro fa uso della funzione membro **open** presente in ognuna delle classi stream.
- **File di output**
- L'apertura di un file di output, associato all'oggetto outFile per es., può essere realizzata attraverso la notazione:

```
ofstream outFile ("nome_file.dat", ios::out);
```

- oppure usando la funzione membro **open**(*lista_argomenti*) della classe ofstream:

```
ofstream outFile; // dichiarazione dell'oggetto di classe ofstream
outFile.open("nome_file.dat", ios::out);
```

- Il parametro **ios::out** indica che la modalità di apertura del file è "in sola scrittura". Di default, ogni oggetto appartenente alla classe ofstream gode di questa proprietà.



Apertura di file

- **File di input**

Un file di input è associato ad un oggetto della classe ifstream (inpFile per es.) e viene dichiarato seguendo la regola generale:

```
ifstream inpFile("nome_file.dat", ios::in);
```

oppure:

```
ifstream inpFile; // dichiarazione dell'oggetto di classe ifstream
inpFile.open("nome_file.dat", ios::in);
```

Il parametro **ios::in** indica che la modalità di apertura del file è "in sola lettura". Ogni oggetto della classe ifstream è creato, di default, con questa modalità.

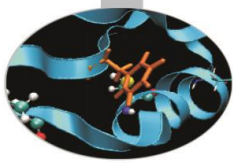
- **File di input/output**

I file di input/output sono associati ad oggetti della classe fstream (ioFile per es.) e vengono aperti nel modo seguente:

```
fstream ioFile("nome_file.dat", ios::in|ios::out);
```

ovvero:

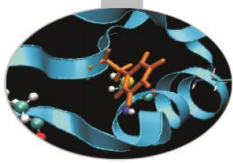
```
fstream ioFile; // dichiarazione dell'oggetto di classe fstream
ioFile.open("nome_file.dat", ios::in|ios::out);
```



Chiusura di file

- Tutti i file vengono chiusi *automaticamente* quando il programma termina.
- Se l'oggetto associato ad un file ancora aperto viene distrutto, il file è chiuso automaticamente quando il *distruttore* viene invocato.
- E', tuttavia, possibile chiudere esplicitamente un file utilizzando la funzione membro **`close()`**, comune a tutte le classi stream, ad es.

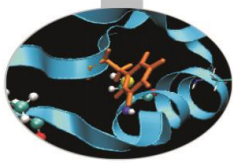
```
outFile.close();
inpFile.close();
```



Modalità apertura dei file

- `ios::app` Aggiunge l'output alla fine del file.
- `ios::ate` Apre un file e si sposta alla fine di esso
- `ios::in` Apre un file in input.
- `ios::out` Apre un file in output.
- `ios::trunc` Elimina il contenuto del file se esiste.
  Di default si comporta così anche ios::out.
- `ios::nocreate` Se il file non esiste, l'operazione open fallisce.
- `ios::noreplace` Se il file esiste, l'operazione open fallisce.

File ad accesso sequenziale



- Nei file ad accesso *sequenziale*, in prima approssimazione, possiamo dire che i record sono acceduti *nell'ordine in cui sono stati scritti* dal primo fino all'ultimo.
- La scrittura di dati su file ad accesso sequenziale avviene semplicemente per mezzo dell'operatore di inserimento nello stream **<<**, ad.es:

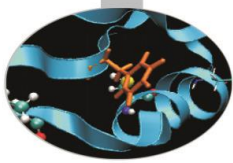
```
outFile << nome_variabile1 << nome_variabile2 << endl;
```

- La lettura da file ad accesso sequenziale si avvale, invece, dell'operatore di estrazione dallo stream **>>**:

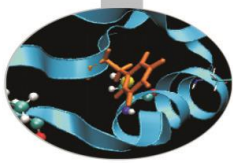
```
inpFile >> nome_variabile1 >> nome_variabile2;
```

- L'uso degli operatori **>>** e **<<** è, sostanzialmente, ciò che permette di capire che un file è ad accesso sequenziale ed implica l'utilizzo di un modello ***formattato*** di input/output ove *la dimensione dei record non è costante*.

File ad accesso sequenziale



- Per recuperare un dato in maniera sequenziale, il programma dovrebbe tutte le volte cominciare a vagliare i dati uno dopo l'altro, a partire da quello iniziale, finché non trova quello desiderato.
- Per velocizzare queste operazioni, le classi istream e ostream forniscono metodi che permettono di conoscere e di modificare il valore associato al *puntatore di posizione del file*, ovvero il numero d'ordine (0,1,2,…,n; di tipo *long int*) del byte corrispondente alla locazione di memoria, attualmente puntata sul file, dalla quale leggere o sulla quale scrivere.
- La classe `istream` mette a disposizione le funzioni membro `tellg` e `seekg` (**`get`**). La prima serve per conoscere la locazione di memoria attualmente puntata sul file e la seconda consente di specificare la posizione da cui deve cominciare la successiva operazione di input.
- Analogamente la classe ostream annovera fra le sue funzioni membro `tellp` e `seekp` (**`put`**).



File ad accesso sequenziale

- La sintassi seguita per l'uso delle funzioni seekg e seekp è la seguente:

```
nome_oggetto.seekx(numero_byte, direzione_ricerca);
```

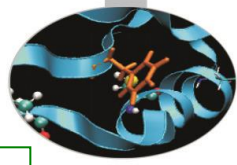
- Le modalità di posizione possibili, associate alla direzione di ricerca, sono:

`ios::beg`	posizionamento relativo all'inizio dello stream (default);
`ios::cur`	posizionamento relativo alla locazione corrente;
`ios::end`	posizionamento relativo alla fine dello stream.

- **Esempi:**

`inpFile.seekg(0);`	posizionamento all'inizio del file associato a inpFile
`inpFile.seekg(n);`	posizionamento sull'n-simo byte del file
`inpFile.seekg(n, ios::cur);`	posizionamento in avanti di n byte dalla posizione corrente
`inpFile.seekg(n, ios::end);`	posizionamento all'indietro di n byte dalla fine del file
`inpFile.seekg(0, ios::end);`	posizionamento alla fine del file

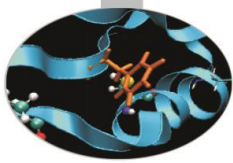
Esempio: scrittura



```
#include<fstream>
#include<iomanip>
int main(){
   int intero=12345;
   double decimale=12.345;
   ofstream outFile;
   outFile.open("myfile.dat",ios::out);
   outFile << setiosflags(ios::left) << setw(15) <<"Numero" << setw(15)
           << endl;
   outFile << setw(8) <<setfill('*') << setiosflags(ios::right)
           << setiosflags(ios::showbase) << intero << endl;
   outFile << setw(5) << hex << setfill('^') << setiosflags(ios::right )
           << setiosflags(ios::showbase) << intero << endl;
   outFile << setw(7) << setbase(8) << setfill('%')
           << setiosflags(ios::right) << setiosflags(ios::showbase)
           <<intero << endl;
   outFile << setw(10) << setiosflags(ios::internal | ios::showpos |
           ios::fixed) << setfill('$') << decimale << endl;
   outFile << setw(10) << setiosflags(ios::scientific) << setfill('$')
           << decimale << endl;
   return 0;
}
```

- myfile.dat:

```
Numero
***12345
0x3039
%030071
+12.345000
+12.345$$$
```



Esempio lettura

```
#include <iostream>
#include <fstream>
#include<vector>
using namespace std;
int main()
{
        ifstream input;
        input.open("data.dat", ios::in);
        double a[3];
        if(input.fail())
        {
                cout<<"error opening file data.dat"<<endl;
                return 1;
        }
        int i=0;
        while(!input.eof() && i<3)
        {
                input>>a[i];
                i++;

        }
        input.close();
        cout<<"Letto"<<endl;
        for(int i=0;i<3;i++)
                cout<<a[i]<<endl;
        return 0;
}
```