File di testo
in C++

File di testo in C++

Il linguaggio C++ mette a disposizione le seguenti **classi** per operare sui file (è necessario includere nel programma l'**header <fstream>**), consentendo tipo di *accesso sequenziale* o *diretto* a seconda del tipo di file (sequenziale o binario).

Classi: ***ifstream*** per **leggere** dati da file (input)

ofstream per **scrivere** dati su file (output)

fstream per **leggere** e/o **scrivere** dati in momenti diversi.
Può essere **scelta la modalità in fase di apertura del file**, oppure
la **prima operazione su file ne determinerà la modalità** (se si scrive→ scrittura, se si legge →lettura).
Il cambio di modalità si può fare solo chiudendo il file e riaprendolo.

File di testo in C++

Dichiarazioni e istruzioni di base per l'uso dei file

Per operare su un file di testo sono necessari i seguenti passaggi:

1. Dichiarazione della **libreria** da includere
2. Dichiarazione di un **oggetto** di una delle classi che utilizzerà il programma.
3. Apertura del file con il metodo **open** e specificando il **nome fisico** del file

   L'apertura del file può anche essere contestuale alla dichiarazione dell'oggetto.

   Es. ifstream *fin* ("nomefile.txt");
   ofstream *fout*("dati.txt", ios::app);
   fstream *finout*("file.txt", ios::out);

   Subito dopo l'apertura del file (soprattutto per i file in lettura) è consigliato **testare se il file è stato trovato** e aperto correttamente.

.....

```
#include <iostream>
#include <fstream>
...
int main() {
 ifstream fin;   // fin nome logico del file
 ofstream fout;
 fstream  finout;

 fin.open ("nomefile.txt");
 fout.open ("dati.txt");
 finout.open ("file.txt", ios::out);  //ios::in
                                      //ios::app

 if (!fin ) {      //oppure if (fin.fail()) {
    cout<<"Errore in apertura file"<<endl;
    return -1;
 }
 else {
        ...   // operazioni sul file
        ...
 }
....
}
```

File di testo in C++

Per operare su un file di testo sono necessari i seguenti passaggi:

4. Scrittura: le operazioni di scrittura avvengono usando l'operatore: **<<**

   Se il file esiste già le operazioni di scrittura lo sovrascriveranno a meno che non sia stata specificata la modalità **ios::app**.

5. Lettura: le operazioni di lettura avvengono in modalità sequenziale e si usa l'operatore: >>

   Quando si legge da file si deve verificare se si è **raggiunta la fine del file**.

6. Chiusura del file con il metodo **close**

```
#include <fstream>
...
int main() {
 int anno=2018, eta;
 char cognome [15];
 ...                                  // scrittura
 fout << "Ciao mondo ! " << anno << endl;



 fin >> cognome ;                     // lettura primo dato
 while (!fin.eof() ) {                // ciclo di lettura
    fin >> eta;
    cout<<"Studente "<<cognome
          <<"  eta' "<<eta<<endl;
    fin >> cognome ;
 }

// in alternativa
 while (fin >> cognome) {   // ciclo di lettura
    fin >> eta;
    cout<<"Studente "<<cognome
          <<"  eta' "<<eta<<endl;
  }

fin.close() ;
fout.close() ;
finout.close() ;
}
```

File di testo in C++ e gli errori

Gli oggetti delle classi: ***ifstream***, ***ofstream***, ***fstream*** **restituiscono** **FALSE** in corrispondenza di errori che si possono verificare in fase di apertura, lettura e scrittura del file.

Queste **situazioni di errore** andrebbero gestite e per farlo può essere utile leggere lo **stream state**.
Ciascun (i/o)stream si trova in uno STATO memorizzato in un **insieme di flag** (valori booleani) all'interno dell'oggetto associato a quello stream.

Esempio
Apertura in lettura: controllo esistenza file

```
fstream miofile ("prova.txt", ios::in);
if(!miofile){
        cout<<"Errore nell'apertura del file"<<endl;
}
else{
  //operazioni sul file
  .....
  .....
}
```

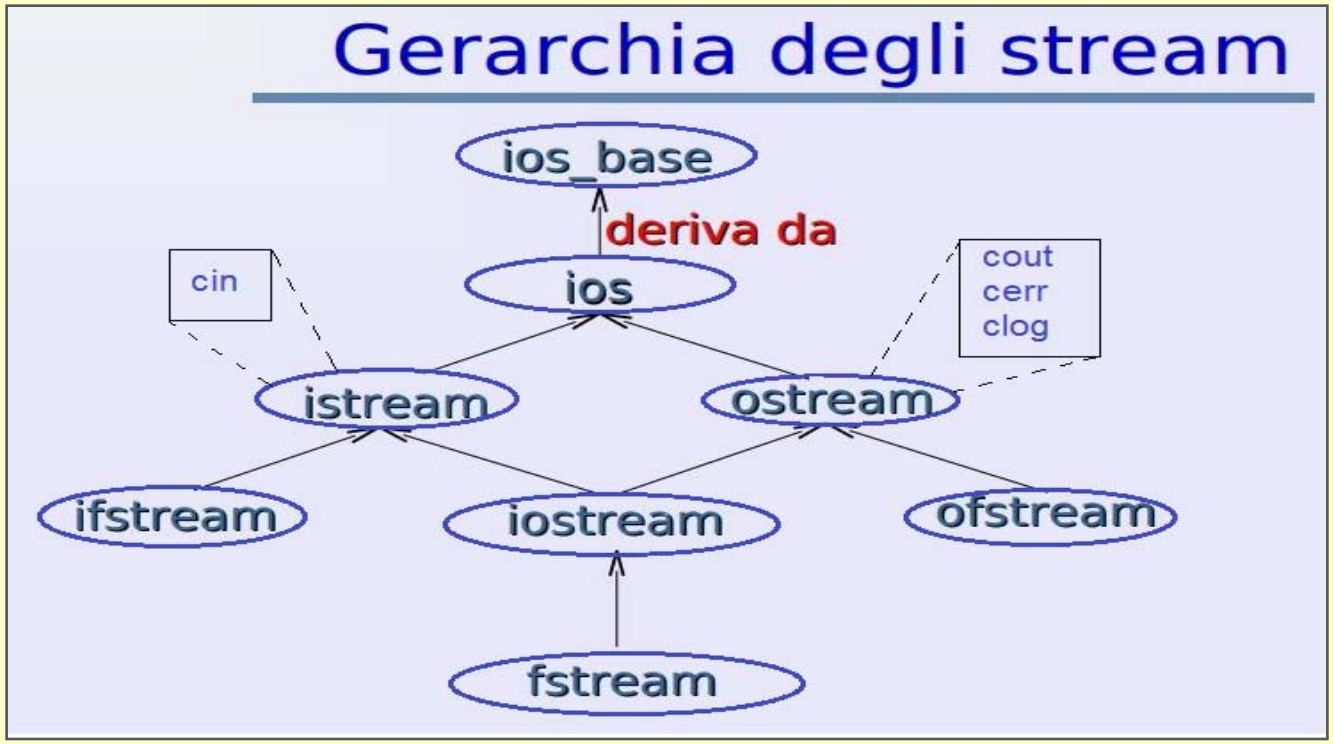
```
fstream miofile ("prova.txt", ios::in);
if(miofile.fail() ){
        cout<<"Errore nell'apertura del file"<<endl;
}
else{
  //operazioni sul file
  .....
  .....
}
```

File di testo in C++ e gli errori

Tutti gli oggetti di uno stream, **mettono a disposizione** i seguenti **metodi** per recuperare e testare il loro **stato interno** (stream state):

nomefile.**eof()**	Tale metodo restituisce il valore booleano TRUE quando viene letto il carattere EOF **di fine file**.
nomefile.**fail()**	restituisce TRUE se si è verificato un **errore nell'apertura** dello stream, FALSE altrimenti;
nomefile.**good()**	restituisce TRUE se **l'operazione effettuata** sullo stream è **andata a buon fine**, FALSE altrimenti;
nomefile.**bad()**	restituisce TRUE se **l'operazione effettuata** sullo stream ha prodotto **un errore irreversibile**, FALSE altrimenti;
nomefile.**is_open()**	restituisce TRUE se il **file è aperto**, FALSE altrimenti.

Quando uno stream si porta in uno **stato di errore**, esso ci rimane finché i **flag** non sono **esplicitamente resettati** e in quel frattempo le operazioni di input su quello stream sono operazioni nulle.
Quindi, prima di effettuare la successiva operazione di input, bisogna *resettare lo stato* dello stream utilizzando il metodo *nomefile*.**clear()**. Inoltre, quando un'operazione di input fallisce, dallo stream di input non viene rimosso alcun carattere, pertanto bisogna ripulire anche lo stream richiamando il metodo *nomefile*.**ignore()**.



La classe ***istream*** gestisce gli **stream di INPUT** e implementa i meccanismi per INTERPRETARE e CONVERTIRE sequenze di caratteri in valori di diverso tipo. Lo STANDARD INPUT è **cin** ed è associato alla TASTIERA.

La classe ***ostream*** gestisce gli **stream di OUTPUT** e implementa i meccanismi per CONVERTIRE valori di vario tipo in sequenze di caratteri.
Lo STANDARD OUTPUT è **cout** e lo STANDARD ERROR è **cerr** e sono entrambi associati al MONITOR.

Standard Input e controllo errore

```
#include <iostream>

using namespace std;
int main() {
  int eta;

  cout << "\nQuanti anni hai?  ";
  cin >> eta;

 while(cin.fail ( )) {
      if(cin.bad( ) ) {
          cerr << " --> ERRORE Stdin!" << endl;
          system("pause");
          return 0;
      }
     cerr<< " --> Non hai inserito un numero!" << endl;

     cin.clear();
     cin.ignore(100, '\n');

     cout << "\nQuanti anni hai?  ";
     cin >> eta;
 }

 cout<<"\nTu hai "<<eta<<" anni"<<endl;
 system("pause");
}
```

Esegue il controllo (**cin.fail()**) su un input da tastiera che deve essere numerico.
Il programma recupera l'eventuale errore logico nel caso in cui si siano scritte **lettere anziché cifre**.

- Se l'errore è irrecuperabile (**cin.bad()**)
  → messaggio d'errore e il programma termina
- Se invece l'errore è logico
  (inseriti dei caratteri anziché numeri)
  → viene emesso un avvertimento a video,
  e con: **cin.clear()** fa il reset della condizione di errore,
  cin.ignore() svuota il buffer di input()
  e si chiede nuovamente il dato