Gestione dei file di dati

1. Le classi di input/output

Il linguaggio C++ non dispone di proprie istruzioni per la gestione dell'interazione con l'utente. Si appoggia infatti su appositi comandi scritti per l'architettura della macchina e facenti parte delle librerie standard del linguaggio.
Le operazioni di ingresso e uscita dei dati sono gestite attraverso le risorse delle **librerie standard di I/O** (*Input/Output*): esse forniscono anche gli strumenti per leggere e scrivere negli **archivi di dati** che si trovano sui supporti di memoria di massa (dischi fissi, dischi esterni, nastri, memorie USB).

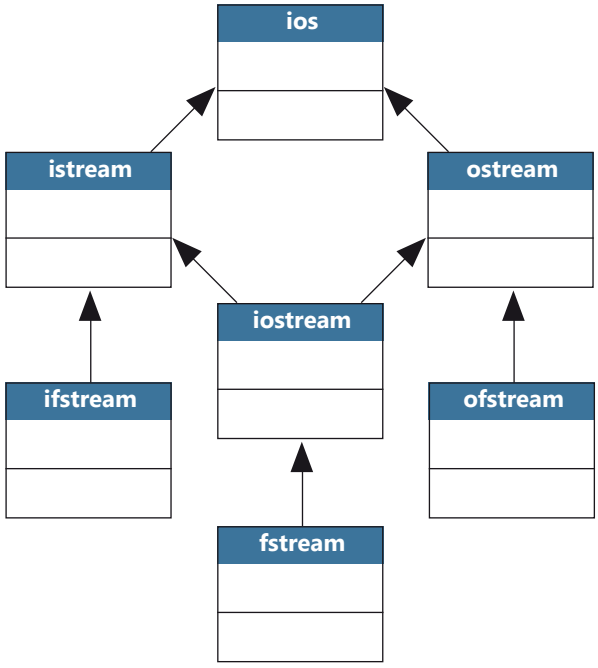
I flussi di dati dal programma alla periferica (*output*), sia essa lo schermo, la stampante o un disco, e i flussi di dati dalla periferica, la tastiera o il disco, al programma (*input*) sono gestiti attraverso gli *stream*.
Il termine *stream* si traduce in italiano con **flusso**.

> Per semplicità possiamo definire lo **stream** come il nome interno al programma utilizzato per identificare la periferica.

Dal punto di vista della programmazione ad oggetti lo *stream* è l'**oggetto periferica**, ossia l'istanza della classe definita per la gestione delle periferiche. È infatti attraverso l'uso dei metodi (funzioni membro) di uno *stream* o, più in generale, di una classe, che si gestiscono i flussi di dati da e verso una periferica.
Ogni volta che si utilizza uno *stream*, per esso viene riservata un'area di memoria centrale, detta **buffer di I/O**, utilizzata per le operazioni di lettura e scrittura sul file.

Le classi di input/output, o classi **iostream**, consentono al programma di trasformare oggetti di un certo tipo in caratteri di testo visualizzabili sullo schermo oppure di convertire i dati provenienti dalla tastiera oppure, ancora, di scrivere e leggere dati in formato binario sulle memorie di massa.

Le classi sono organizzate in una struttura gerarchica, avente come radice la **classe ios**, secondo il seguente schema:



Le dichiarazioni delle classi di input/output sono contenute nei due principali file di libreria:

- **iostream**, che contiene la definizione delle classi *ios*, *istream*, *ostream*, *iostream* per la gestione dei flussi standard di input/output;
- **fstream**, che contiene le definizione delle classi *fstream*, *ifstream*, *ofstream* per la gestione dei flussi collegati ai file registrati su memoria di massa o, in generale, ai file identificati con un nome logico.

Quindi i programmi che utilizzano i flussi devono contenere le direttive di inclusione:

```
#include <iostream>
#include <fstream>
```

2. Input e output standard

I programmi scritti in linguaggio C++ utilizzano quattro flussi o **stream predefiniti**, che vengono aperti al momento dell'esecuzione:

- **cin** è un oggetto della classe *istream* e indica il flusso associato all'unità standard di input (*standard input*), cioè la tastiera;
- **cout** è un oggetto della classe *ostream* e indica il flusso associato all'unità standard di output (*standard output*), cioè il video;
- **cerr** è un oggetto della classe *ostream* e indica il flusso associato all'unità standard per la visualizzazione dei messaggi di errore (*standard error*), cioè il video;
- **clog** è un oggetto della classe *ostream* e indica il flusso per la registrazione delle operazioni svolte dal sistema durante una sessione di lavoro (*standard log*).

Per leggere da standard input singoli caratteri o array di caratteri si può usare anche il metodo **get()**.

Per esempio, per leggere un nome avente al massimo 19 caratteri, si possono usare le istruzioni:

```
char nome[20];
cin.get(nome, 20);
```

Occorre precisare che l'operatore **>>** considera gli spazi, le tabulazioni e i caratteri di fine riga ('\n') tutti allo stesso modo, cioè simboli di fine input. Per un corretto funzionamento delle operazioni di input che riguardano stringhe di caratteri, che possono contenere anche spazi bianchi (per esempio, cognome e nome di una persona acquisiti sulla stessa riga), è più appropriato utilizzare il metodo **getline()** dell'oggetto *cin*.

La lettura con *get()* o *getline()* termina con '\n', che viene convertito in '\0', oppure con la lettura del numero di caratteri specificati meno 1, aggiungendo in coda il carattere '\0' di fine stringa.

Solitamente anche una lettura con >> termina con il carattere '\n' (*newline*). L'operatore >> non inserisce tale carattere nella stringa di input e lo lascia nel *buffer di I/O*, cioè nella memoria temporanea, associato allo stream *cin*.

Questo significa che una successiva operazione di input con *get()* o *getline()*, trovando il carattere di terminazione, lo scarica dal buffer di I/O, lo converte in '\0' e non esegue l'acquisizione dei dati. Occorre quindi eliminare il carattere di terminazione con il metodo **ignore**:

```
cin.ignore(80,'\n');
```

Rimuove dal buffer di input tutti i caratteri, fino a quando incontra il primo carattere **\n** (*newline*). Il numero 80 indica il numero massimo di caratteri di una riga.

Nell'esempio seguente, i dati acquisiti da tastiera vengono subito dopo visualizzati. Il programma mostra un esempio di uso del metodo **getline()**.

```
// Tastiera.cpp: acquisizione di caratteri

#include <iostream>
using namespace std;

int main()
{
   char codice[5],nome[40],telefono[24];
```

```
    // acquisizione dei dati
    cout << "Inserimento nuovo cliente " << endl;
    cout << "------------------------ " << endl;
    cout << "Codice cliente: ";
    cin >> codice;
    cin.ignore(80,'\n');
    cout << "Nome Societa': ";
    cin.getline(nome,40);
    cout << "Telefono: ";
    cin >> telefono;
    cin.ignore(80,'\n');
    // visualizza
    cout << codice << endl;
    cout << nome << endl;
    cout << telefono << endl;
    cout << endl;
    // attesa
    cout << "premi Invio per continuare" << endl;
    cin.get();
}
```

3. Archivi di dati

Comunemente per indicare un archivio di dati si usa il termine inglese **file**.

In generale il termine *file* sta ad indicare tutto ciò che può essere registrato su un supporto di memoria di massa (testo, lettera, relazione, programma, archivio di dati o immagine).

Il file può essere anche considerato come un insieme di record legati tra loro da un nesso logico, cioè un insieme di informazioni strutturate, organizzate in un archivio e registrate su un supporto di memoria esterna.

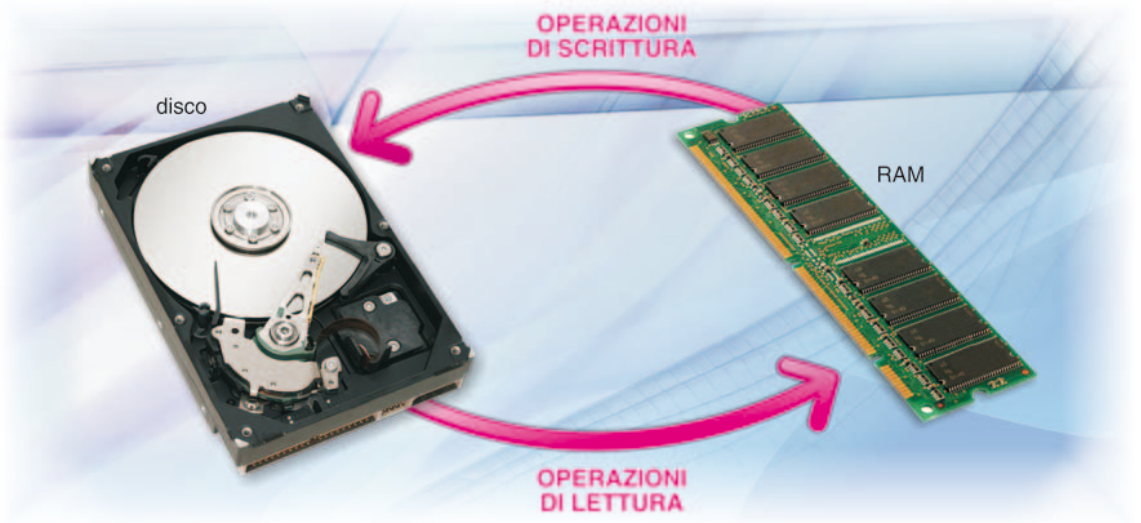
I file vengono identificati attraverso un nome di file: di solito i nomi sono significativi, cioè ricordano il tipo di dati che contengono.

Le caratteristiche dei nomi di file (lunghezza, set di caratteri utilizzabili) dipendono dal *file system* del sistema operativo in uso.

Le operazioni fondamentali riguardanti il trattamento di un file sono:

- **Apertura** del file: si stabilisce un collegamento tra la memoria centrale e l'unità di memoria di massa che contiene il file sul quale si vogliono fare operazioni di lettura e scrittura.
- **Chiusura** del file: si chiude il collegamento tra la memoria centrale e l'unità di memoria di massa che contiene il file. Dopo la chiusura non è più possibile effettuare alcuna operazione su quel file a meno che non si effettui una nuova apertura.
- **Lettura** dal file: le operazioni che trasferiscono dati dal file alla memoria centrale.
- **Scrittura** sul file: le operazioni che trasferiscono dati dalla memoria centrale al file.

La lettura si indica anche con *input* e la scrittura con *output*, e le operazioni che trasferiscono informazioni dalla memoria centrale al file e viceversa si chiamano, in generale, **operazioni di I/O** (*Input/Output*) sul file.



L'area di memoria centrale, detta **buffer di I/O**, utilizzata per velocizzare le operazioni di lettura e scrittura sul file, viene rilasciata con l'operazione di chiusura o con specifici comandi di svuotamento del buffer.

4. File di testo e file binari

I file trattati dal *file system* di un sistema operativo si possono classificare secondo due tipologie, in base all'organizzazione dei dati in essi contenuti:

- file di testo o file ASCII
- file binari.

La sostanziale differenza tra questi due tipi di file riguarda la loro leggibilità: i **file di testo** o **file ASCII** contengono i caratteri di fine riga (**EOL**, *End of Line*) per fornire un'impaginazione delle informazioni contenute e ciascun byte rappresenta un carattere della codifica ASCII, mentre i **file binari** sono considerati come un flusso continuo di bit, anch'essi raggruppati in byte, ma non necessariamente distinti l'uno dall'altro.

Storicamente la classificazione in file ASCII e in file binari era molto rilevante in quanto le informazioni riguardanti l'utente, cioè i dati, i documenti e i programmi sorgenti realizzati dal programmatore, venivano memorizzati in file prontamente leggibili da parte dell'utente, senza bisogno di un apposito programma di lettura.

Viceversa le informazioni riguardanti il computer, per esempio i programmi applicativi, i comandi di sistema, gli interpreti dei comandi (*shell*) o, in generale, i **programmi eseguibili**, venivano memorizzati in file prontamente comprensibili dal processore della macchina, dato che le informazioni erano memorizzate in **linguaggio macchina**.

Un tempo quindi i dati a disposizione dell'utente erano organizzati in file di testo. Con l'evolversi e lo specializzarsi dei programmi per l'automazione dell'ufficio (*office automation*), i file utilizzati da tali pacchetti sono stati personalizzati in base al programma che li ha creati. Per questa ragione, i file creati da un programma di elaborazione di testi (per esempio *Word*) sono leggibili e modificabili esclusivamente da quel programma. La stessa cosa accade con un foglio elettronico o un database: infatti, nell'ambiente *Windows*, per visualizzare in modo leggibile un file creato con *Word* o con *Access* non si utilizza il programma *Blocco Note*, che consente di visualizzare correttamente solo file di testo.

Oggi, quindi, i file di testo vengono impiegati solamente per i codici sorgente dei linguaggi di programmazione e nei casi in cui il trasferimento dei dati da un applicativo all'altro sia incompatibile. I programmi applicativi professionali, invece, per una maggiore efficienza conservano le informazioni, da essi trattate, in file binari. In questa evoluzione i programmi eseguibili hanno mantenuto le caratteristiche proprie dei file binari.

Per la loro caratteristica i **file di testo** risultano particolarmente adatti per memorizzare dati di lunghezza diversa. Inoltre occupano più spazio su disco, però sono più facili da gestire e possiedono una maggiore portabilità, nel senso che possono essere letti su computer aventi sistemi operativi diversi.
I **file binari** sono invece adatti per archivi di record a lunghezza fissa, occupano meno spazio su disco e sono più veloci nelle operazioni di lettura e scrittura.

5. Lettura e scrittura in un file di testo

Un file può essere utilizzato per scrivere nuove registrazioni, per leggere dati o per aggiungere dati in coda a quelli già registrati.

Un file può essere aperto per le seguenti operazioni:
- scrivere caratteri in un file (*output*)
- leggere caratteri da un file (*input*)
- aggiungere caratteri al file in coda a quelli già esistenti (*append*).

La scrittura a partire dall'inizio, eseguita su un file già esistente, provoca la cancellazione di eventuali dati presenti nell'archivio; è per questa ragione che dobbiamo distinguere l'operazione di scrittura (*output*) dall'operazione di aggiunta in coda (*append*).
Nelle operazioni di lettura in un archivio con *accesso sequenziale*, occorre introdurre un opportuno controllo per segnalare il raggiungimento della *fine del file*.

L'esempio seguente descrive il **programma di scrittura** di un file di testo, in generale un file avente record di lunghezza variabile. Poiché la scrittura avviene dall'inizio, di fatto il programma esegue la *creazione* di un nuovo archivio, anche nei casi in cui l'archivio già esista.

ESEMPIO

Creare un archivio con i nomi degli amici.

Si vuole creare un archivio su memoria di massa che contenga i nomi dei propri amici. Viene inserito in input l'elenco dei nomi degli amici: i nomi devono essere uno a uno registrati in un file di testo, ogni riga del file contiene un nome di amico. L'archivio è aperto in scrittura e ciò comporta la cancellazione di un eventuale archivio preesistente con lo stesso nome e la scrittura dei dati inseriti a partire dall'inizio del file.
All'inizio del programma viene definito l'oggetto *fout* della classe **ofstream**, associato al nome fisico del file, *amici.dat*, cioè al nome con cui il file è registrato su disco.
Il costruttore contiene l'**apertura implicita del file**: vedremo in seguito l'apertura esplicita, tramite il metodo *open*().

Programma C++

```
// CreaAmici.cpp: creazione archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
```

Direttiva di inclusione della libreria **fstream** per utilizzare le classi di I/O.

```
int main()
{
   string nome;
   ofstream fout("amici.dat");      // apre il file in scrittura
   cout << "inserisci i nomi" << endl;
   cout << "uno per riga (Ctrl+Z per terminare)" << endl;
   while (cin >> nome) {
      fout << nome << endl;         // scrive sul file
   }
   fout.close();                    // chiude il file

   return 0;
}
```

La lettera iniziale della classe (lettera 'o', di *ofstream*) indica che *fout* è un oggetto di output.

Scrittura sul file: **operatore <<** applicato all'oggetto *fout*.

La fine dell'inserimento dei nomi dalla tastiera è segnalata dall'utente con la combinazione dei tasti **Ctrl + Z** (nei sistemi Linux, **Ctrl + D**), che indica la fine dell'input da linea comandi.

Per controllare il contenuto di un file di testo già esistente si può utilizzare un **programma di lettura**.
Poiché i dati registrati vengono letti dal file uno di seguito all'altro, a partire dall'inizio, nello stesso ordine con il quale erano stati scritti, cioè con *accesso sequenziale*, occorre inserire all'interno del programma di lettura un controllo sul raggiungimento della **fine del file**.

ESEMPIO

Visualizzare il contenuto dell'archivio degli amici precedentemente creato.

Per la visualizzazione del contenuto dell'archivio degli amici si procede utilizzando una ripetizione che comprende la lettura sequenziale di ogni riga del file e la sua visualizzazione. La ripetizione si arresta dopo che è stata letta l'ultima registrazione del file.
All'inizio del programma viene definito un oggetto *fin* della classe **ifstream**, associato al nome fisico del file, *amici.dat*.

Programma C++

```
// LeggiAmici.cpp: lettura archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
   string nome;
   ifstream fin("amici.dat");   // apre il file in lettura
   while (fin >> nome) {        // legge dal file fino alla fine
      cout << nome << endl;
   }
   fin.close();                 // chiude il file

   return 0;
}
```

Direttiva di inclusione della libreria **fstream** per utilizzare la classe di I/O *ifstream*.

La lettera iniziale della classe (lettera 'i', di *ifstream*) indica che *fin* è un oggetto di input.

Lettura sul file: **operatore >>** applicato all'oggetto *fin*.

Si possono aggiungere altri dati ad un archivio creato in precedenza specificando la modalità **append** nell'apertura del file.

Per esempio, se si vogliono aggiungere altri nomi di amici all'archivio, si può usare un programma del tutto simile a quello usato per l'inserimento dei nomi.

Programma C++

```
// AggiungiAmici.cpp: aggiunta di nuovi nomi
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
   string nome;
   // apre il file in append
   ofstream fout("amici.dat", ios::app);
   cout << "inserisci nuovi nomi" << endl;
   cout << "uno per riga (Ctrl+Z per terminare)" << endl;
   while (cin >> nome) {
      fout << nome << endl;      // scrive sul file
   }
   fout.close();                 // chiude il file

   return 0;
}
```

Aggiunta della costante **ios::app** nella dichiarazione del file, per specificare la modalità di apertura in *append*.

La costante **ios::app** appartiene all'insieme delle **costanti enumerative** per l'apertura del file: esse sono definite nella classe **ios**, contenuta nella libreria *iostream*. Le altre costanti, presentate in seguito, sono **ios::in**, per la modalità di lettura, e **ios::out**, per la modalità di scrittura.

Dopo aver costruito il programma per l'aggiunta di nuovi nomi, si può verificare che i nomi già esistenti nell'archivio siano stati mantenuti, eseguendo di nuovo il programma di lettura del file.

6. Apertura esplicita dei file

L'operazione di apertura di un file può essere ottenuta anche tramite il metodo **open()** applicato all'oggetto *stream*. In questo caso l'apertura si compone di due passi:

- creazione di una variabile *stream* con costruttore senza argomenti, per esempio:

```
ofstream fout;
```

- connessione dello *stream* all'archivio di dati tramite il metodo *open()*, specificando il nome del file e la modalità di apertura; per esempio, nel programma di aggiunta di nuovi dati, si usa la seguente istruzione:

```
fout.open("amici.dat", ios::app);
```

Di seguito, con riferimento all'archivio di nomi degli esempi precedenti, sono descritte le variazioni da apportare alle prime istruzioni dei programmi per la creazione del file, la lettura sequenziale e l'aggiunta di nuovi dati in coda a quelli già esistenti, utilizzando il metodo *open()*:

```
ofstream fout;
fout.open("amici.dat", ios::out);
```

```
ifstream fin;
fin.open("amici.dat", ios::in);
```

Le costanti *ios::out* e *ios::in* possono essere tralasciate perché sono i valori di *default*, rispettivamente per gli oggetti di tipo *ofstream* e *ifstream*.

```
ofstream fout;
fout.open("amici.dat", ios::app);
```

A titolo di esempio viene riportato il programma completo di creazione dell'archivio, modificato con l'introduzione dell'apertura esplicita dell'archivio, tramite il metodo *open()*.

Programma C++

```
// CreaAmici2.cpp: apertura esplicita
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  string nome;

  ofstream fout;              // dichiara lo stream
  fout.open("amici.dat");     // connette lo stream all'archivio

  cout << "inserisci i nomi" << endl;
  cout << "uno per riga (Ctrl+Z per terminare)" << endl;
  while (cin >> nome) {
      fout << nome << endl;   // scrive sul file
  }
  fout.close();               // chiude il file

  return 0;
}
```

Apertura esplicita dell'archivio.

7. Controllo della fine del file

Come si è già visto in precedenza, nel programma di lettura sequenziale di un file, occorre inserire un controllo sul raggiungimento della fine del file. Il modo più semplice è stato presentato negli esempi precedenti con l'uso di *while (fin >> variabile)*. In alternativa si può utilizzare il metodo **eof()**, *end of file*, che assume il valore *true* quando viene raggiunta la fine del file.
La ripetizione per la lettura sequenziale del file viene realizzata con la seguente struttura *while*:

```
nomestream >> variabile;
while (!nomestream.eof()) {
  cout << variabile << endl;
  nomestream >> variabile;
}
```

Due operazioni di lettura dal file, una esterna prima di *while* e una interna alla ripetizione.

Di seguito viene presentata un'ulteriore versione del programma di lettura sequenziale del file, che utilizza il metodo *eof()* per il controllo di fine file.

```
// LeggiAmici4.cpp: lettura archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
   ifstream fin("amici.dat");  // apre il file in lettura
   string nome;
   fin >> nome;                // legge dal file
   while (!fin.eof()) {
      cout << nome << endl;
      fin >> nome;
   }
   fin.close();                // chiude il file

   return 0;
}
```

La ripetizione viene eseguita mentre il valore restituito dal metodo *eof()* è diverso da *true*.