Capitolo 6

Input/output su file

BOZZA

6.1 Stream e file

L'input/output in C++, in particolare quello su file, avviene tramite stream.

stream. Uno *stream* è un'astrazione di un canale di comunicazione, che collega un mittente ed un ricevente. In particolare, uno stream permette la comunicazione tra un programma e un dispositivo di input/output o un file.

Nei capitoli precedenti abbiamo già utilizzato due stream predefiniti:

- `cin` per la comunicazione dal dispositivo di input standard, la tastiera, al programma (stream di input standard)
- `cout` per la comunicazione dal programma al dispositivo di output standard, il monitor (stream di output standard).

L'interazione tra questi stream, il programma ed i dispositivi di input/output ad essi associati è schematizzata in Figura 6.1.

Gli stream sono canali di comunicazione *sequenziali*: è possibile ricevere/inviare un dato alla volta, in sequenza, a partire dal primo. In ogni istante lo stream mantiene un'informazione su qual è il dato corrente da leggere/scrivere da/sul file o dispositivo a cui è connesso (si veda Figura 6.2). Questa informazione, che diremo *puntatore al dato corrente*, viene aggiornata automaticamente ad ogni operazione di lettura/scrittura, facendo avanzare il puntatore al dato successivo da leggere/scrivere.

Spesso gli stream sono usati come canali di comunicazione da/verso file allo scopo di permettere ad un programma di effettuare operazioni di let-

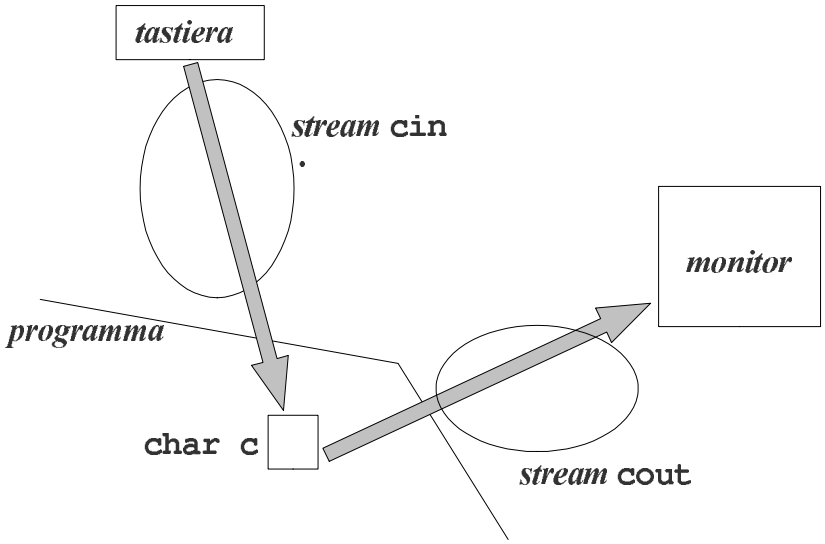

Figura 6.1: Stream di input e di output standard

tura/scrittura da/su file. Una descrizione precisa di cosa sia e come si gestisca un file esula dagli scopi di queste note. Qui possiamo limitarci a dare la seguente generica caratterizzazione di un file.

File. Un *file* è entità esterna al programma, gestito dal Sistema Operativo. Un file può vedersi come una sequenza di caratteri, identificata da un nome simbolico (una stringa), terminata da un carattere speciale detto *"end-of-file"* (`EOF`), senza limiti di dimensione massima. Inoltre un file è normalmente memorizzato in una memoria non-volatile (ad es. un hard-disk) e quindi in modo permanente.

Una possibile rappresentazione grafica di un file è mostrata in Figura 6.3. Si noti che un file può anche essere vuoto (in questo caso, nella rappresentazione grafica, il primo e unico elemento del file è `EOF`).

Su un file è possibile eseguire diverse operazioni. In generale, mettere in evidenza che per operare su un file all'interno di un programma, in un linguaggio che preveda l'utilizzo di stream, si dovranno svolgere, nell'ordine indicato, le seguenti operazioni:

1. *creare* uno stream, ad es. di nome `f`
2. *associare* lo stream `f` al file (*apertura* del file)
3. eseguire (ripetutamente) le *operazioni di input/output* su f (ad esempio operazioni `get`, `put`, `<<`, `>>`, ecc.)

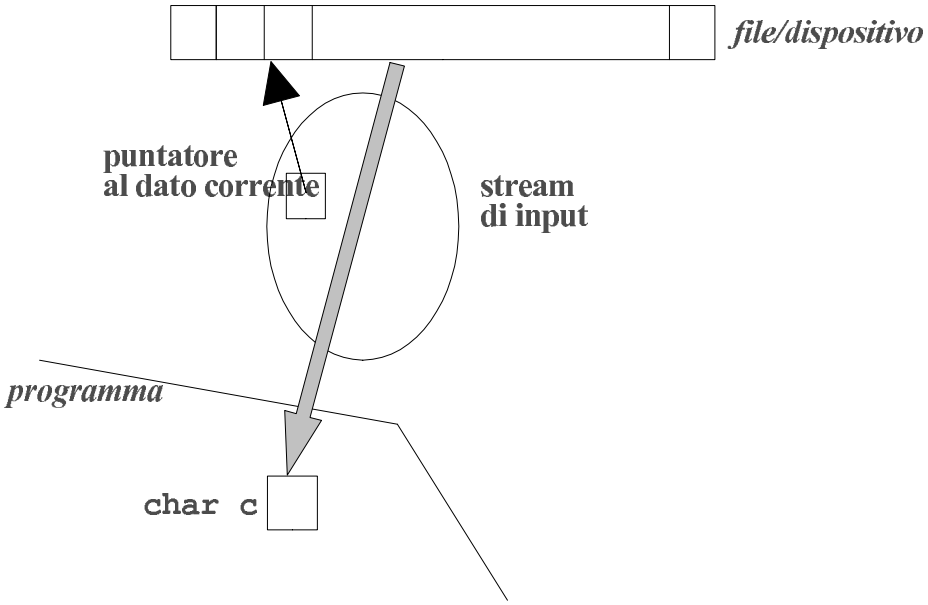

Figura 6.2: Puntatore al dato corrente

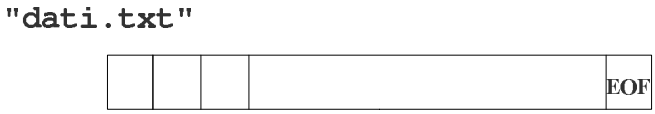

Figura 6.3: Un file di nome "`dati.txt`"

4. al termine, *eliminare* l'associazione tra lo stream e il file (*chiusura* del file).

Analizziamo più in dettaglio queste diverse operazioni nel caso specifico del C++.

6.2 Gestione di stream e file in C++

Per poter creare e gestire stream all'interno di un programma C++ bisogna prima di tutto includere nel programma il file `fstream` con la direttiva

```
#include <fstream>
```

Questo file contiene tutte le dichiarazioni necessarie a creare stream e ad operare su essi.

1. Creazione dello stream

Uno stream può essere creato tramite un'opportuna dichiarazione.

- Dichiarazione di stream di input

  ```
  ifstream f1;
  ```

  crea uno stream di input di nome `f1` (= un oggetto di nome `f1` e tipo `ifstream`).

- Dichiarazione di stream di output

  ```
  ofstream f2;
  ```

  stream di output di nome `f2` (= un oggetto di nome `f2` e tipo `ofstream`).

`ifstream` e `ofstream` sono classi derivate dalla classe `fstream`, che a sua volta è derivata dalla classe `ios`.

2. Associazione dello stream al file (= apertura del file)

Uno stream (di input o di output) può essere associato ad uno specifico file tramite un'operazione di `open` (funzione propria delle classi `ifstream` e `ofstream`).

Apertura stream di input

L'esecuzione di

```
f1.open("dati");
```

associa lo stream di input `f1` al file di nome `"dati"` (n.b.: il nome del file è una stringa tipo C). Da questo punto in poi è possibile operare in lettura sul file `"dati"` tramite lo stream `f1`.

Nota. Il nome del file può essere un "pathname" assoluto (ad esempio, `"C:\Users\Utente1\Documenti\prova.txt"`) o relativo (ad esempio `"prova.txt"`). In quest'ultimo caso il file viene cercato a partire dalla cartella di lavoro, che coincide normalmente con quella contenente il codice sorgente ed eseguibile del programma.

Se il file specificato nella `open` non esiste allora siamo in una situazione d'errore. Questa situazione può essere rilevata tramite la funzione `fail` (funzione propria delle classi `ifstream` e `ofstream`). L'esecuzione di

```
f1.fail()
```

restituisce `true` se, in generale, l'ultima operazione eseguita sullo stream `f1` ha avuto esito negativo, `false` altrimenti.[1] Se l'ultima operazione eseguita è stata quella di apertura del file, allora il fatto che la funzione `fail` restituisca `true` significa che l'apertura è fallita, in particolare perchè il file specificato non è stato trovato.

Mostriamo ora la porzione di programma C++ che effettua la creazione e l'apertura di un file di nome `"dati"`.

[1] Di fatto la funzione `fail` restituisce il valore del campo booleano `failbit` contenuto nello stream a cui essa viene applicata; questo campo viene posto a `true` ogni qualvolta lo stream viene a trovarsi in una situazione di errore.

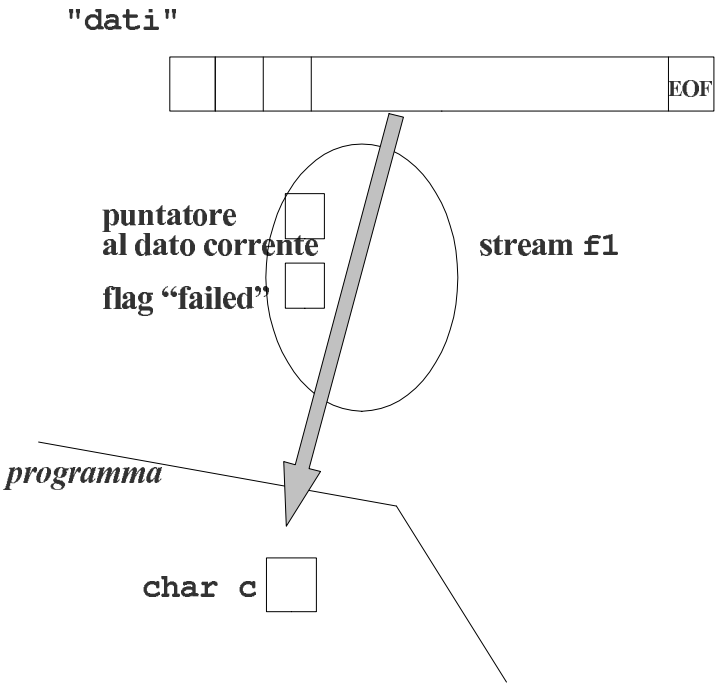

Figura 6.4: Apertura in input del file "dati"

```
#include <fstream>
using namespace std;
int main() {
   //Crea lo stream
   ifstream f1;
   //Apre il file associato allo stream
   f1.open("dati");
   //Controlla errori di apertura del file.
   if (f1.fail()) {
      cout << "Impossibile aprire il file" << endl;
      return 0;
   }
   "operazioni di input sul file tramite f1"
   ...
}
```

Apertura stream di output

L'esecuzione di

```
f2.open("risultati");
```

associa lo stream di output `f2` al file di nome `"risultati"`. Da questo punto in poi è possibile operare in scrittura sul file `"risultati"` tramite lo stream `f2`.

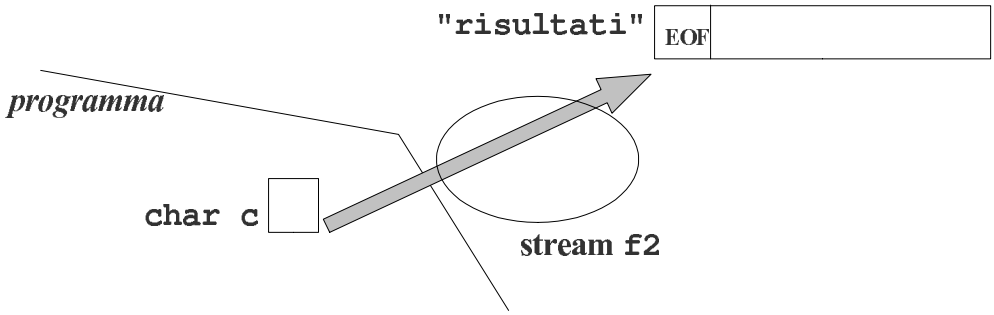

Figura 6.5: Apertura in output del file "`risultati`"

Nel caso di uno stream di output, se il file indicato nella `open` non esiste allora viene creato un file con il nome specificato.[2] Il file viene creato inizialmente vuoto. Il file potrà crescere successivamente in seguito ad eventuali operazioni di scrittura su di esso.

Viceversa, se il file specificato nella `open` già esiste, allora la `open` cancella tutto il precedente contenuto del file (lo 'tronca a zero").

Per evitare il "troncamento" si può specificare che l'apertura del file avvenga con modalità "append". Ad esempio,

```
f2.open("risultati", ios:app);
```

associa allo steam di output `f2` il file `"risultati"`, ma senza cancellare il precedente contenuto del file; le eventuali nuove scritture sul file andranno ad aggiungere i dati di seguito a quelli già presenti.[3]

Apertura file: modalità alternative

- Possibile creare lo stream e associarlo ad un file in un'unica dichiarazione. Ad esempio:

  ```
  ifstream f1("risultati");
  ```

- Possibile testare l'avvenuta apertura del file utilizzando direttamente il nome dello stream come espressione booleana (invece di ricorrere alla funzione `fail`). Ad esempio:

[2] Il file viene creato nella cartella di lavoro corrente a meno che non venga specificato come nome file un percorso che contenga una diversa cartella; ad esempio, "`..\dati_programmi\risultati`" specifica che il file "`risultati`" deve essere creato nella cartella "`dati_programmi`" che si trova nel livello superiore alla cartella corrente.

[3] Con questa modalità di apertura, però, il fatto che il file specificato non esista viene visto come un errore (come nel caso dell'apertura su uno stream di input) e quindi non comporta la creazione del file.

```
if (!f1)
    cout << "Errore apertura file" << endl;
```

(l'operatore `!` è definito in C++in modo tale che quando il parametro è uno stream restituisce come risultato il valore booleano del campo `failbit` contenuto nello stream).

3. Operazioni di input/output

Per eseguire operazioni di input/output su uno stream collegato ad un file è possibile utilizzare le funzioni e gli operatori già incontrati in precedenza a proposito dell'input/output standard (da tastiera e su monitor). In particolare distinguiamo tra:

- input/output a caratteri – funzioni `get`, `put` e `getline`
- input/output "tipato" – operatori `>>` e `<<`.

Descriveremo queste due modalità di input/output su file rispettivamente nei capitoli 6.3 e 6.5.

4. Chiusura di un file

Una volta che il collegamento tra uno stream ed un file non è più utilizzato è opportuno eliminarlo. Questo può essere realizzato tramite la funzione `close` (funzione propria delle classi `ifstream` e `ofstream`). L'esecuzione di:

```
f1.close()
```

provoca l'eliminazione dell'associazione tra lo stream `f1` e il file a cui tale stream era collegato. Si noti che questa operazione non elimina il file e nemmeno lo stream: elimina soltanto il collegamento tra i due. Una volta eseguita la `close` su uno stream è possibile collegare lo stesso stream ad un altro file, con una nuova `open`.

Al termine dell'esecuzione del programma tutti i file eventualmente ancora aperti sono comunque chiusi automaticamente.

6.3 Input/output a caratteri

6.3.1 Lettura di caratteri

L'operazione di lettura di caratteri da uno stream di input avviene principalmente tramite la funzione `get` già incontrata in precedenza (si veda cap. 2.8.3). L'esecuzione di `s.get()` legge (= estrae) dallo stream di input `s` il carattere corrente e lo restituisce come suo risultato (n.b.: il carattere corrente è quello indicato dal puntatore al dato corrente contenuto nello stream `s`).

Esempio 6.1 (Input tramite `get`) *Dato il seguente frammento di codice C++*

```
ifstream f1;
f1.open("dati.txt");
char c;
c = f1.get();
```

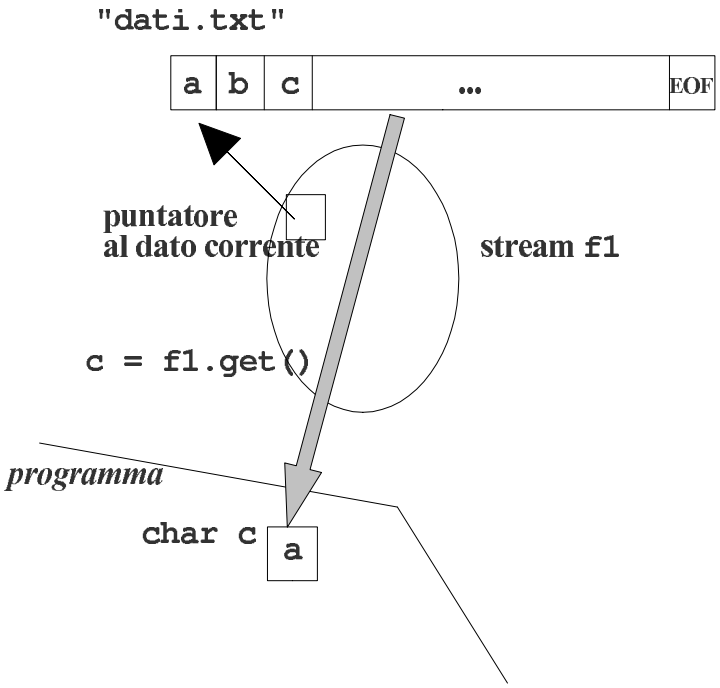

Figura 6.6: Lettura da file tramite `get`

e supponendo che il contenuto del file sia quello indicato in figura 6.6 l'esecuzione della `get` *estrae il primo carattere presente sul file (carattere* `'a'`*). Una successiva* `get`

```
c = f1.get();
```

estrae il carattere successivo (carattere `'b'`*), che viene assegnato alla variabile* `c`*. Una terza* `get`

```
c = f1.get();
```

estrae ed assegna a `c` *il carattere* `'c'`*, e così via.*

Come ci si accorge quando si raggiunge l'end-of-file? Uno stream di input ha associato anche un altro flag (variabile booleana) che indica se si è raggiunto l'end-of-file del file associato allo stream. Questo flag può essere testato con la funzione `eof` (funzione propria della classe `ifstream`). L'esecuzione di

```
s.eof()
```

dove `s` è uno stream di input, restituisce come suo risultato `true` se l'ultima operazione di lettura su `s` (ad es., una `get`) ha letto l'end-of-file; `false` altrimenti.

Vediamo il funzionamento e l'utilizzo della funzione `eof` con un semplice (ma completo) programma C++.

Esempio 6.2 (Conteggio dei caratteri contenuti in un file)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   ifstream in_file;
   in_file.open("dati.txt");
   if (in_file.fail()) {
       cout << "Il file non esiste" << endl;
       return 0;
   }

   int num_car = 0;
   char c;
   c = in_file.get();
   while (!in_file.eof()) {
       num_car++;
       c = in_file.get();
   }
   cout << "Contiene " << num_car << " caratteri" << endl;
   in_file.close();
   return 0;
}
```

Si noti che se il file è vuoto il programma stampa (correttamente) che il file contiene 0 caratteri.

Si presti attenzione al fatto che la funzione `eof` restituisce `true` soltanto dopo aver eseguito un'operazione che ha comportato la lettura del carattere di end-of-file. Se ad esempio il ciclo di lettura da file del programma precedente venisse modificato nel modo seguente

```
...
char c;
while (!in_file.eof()) {
   c = in_file.get();
   num_car++;
}
...
```

(cioè eliminando la `get` che precede il ciclo `while`), allora il numero di caratteri calcolato risulterebbe (erroneamente) incrementato di 1 rispetto al numero di caratteri realmente contenuti nel file. In particolare, se il file fosse vuoto, il risultato stampato sarebbe 1.

Per l'input di caratteri si può utilizzare anche la funzione `getline` già vista in precedenza (cap. 4.5.2), specificando come stream di input quello collegato al file da cui si vogliono leggere dati. La lettura della stringa termina non solo quando si incontra il carattere delimitatore o si raggiunge la dimensione massima, ma anche quando si incontra l'end-of-file.

Test di end-of-file - modalità alternative

- La funzione `get()` restituisce la costante predefinita `EOF` quando legge l'end-of-file. Pertanto è possibile riscrivere il ciclo di lettura da file dell'esempio precedente nel modo seguente:

  ```
  c = in_file.get();
  while (c != EOF) {
     num_car++;
     c = in_file.get();
  }
  ```

  oppure in modo equivalente, ma più sintetico:

  ```
  while ((c = in_file.get()) != EOF)
     num_car++;
  ```

  Si noti l'utilizzo dell'assegnamento come (sotto-)espressione nella condizione del `while`: l'esecuzione di `c = in_file.get()`, oltre ad assegnare a `c` il carattere letto dalla `get`), restituisce questo valore come suo risultato.

  Si presti attenzione al fatto che la modalità di test dell'end-of-file tramite la costante `EOF` può essere usata soltanto in combinazione con la `get`. Il test tramite la funzione `eof()` invece può essere usato in modo più generale, sia con la `get`, che con la `getline`, che con l'operatore `<<`.

- Esistono anche altre forme di `get`, che prevedono che il carattere letto sia un parametro (passato per riferimento) della `get` stessa. In particolare, l'esecuzione di

  ```
  s.get(c)
  ```

  equivale all'esecuzione della `c = s.get()`, ma la `s.get(c)` restituisce anche un risultato che può essere interpretato come un valore booleano, con il seguente significato: se è `true` significa che la `get` è riuscita a leggere correttamente il carattere corrente dallo stream di input `s`; altrimenti, se è `false`, significa che c'è stato qualche problema di lettura, in particolare che il carattere corrente era l'end-of-file.

  Pertanto è possibile riscrivere il ciclo di lettura dal file dell'esempio precedente nel modo seguente:

  ```
  while (in_file.get(c))
     num_car++;
  ```

6.3.2 Scrittura di caratteri

L'operazione di scrittura di caratteri su uno stream di output avviene principalmente tramite la funzione `put` già incontrata in precedenza (si veda cap. 2.8.3). L'esecuzione di `s.put(e)` scrive (= inserisce) sullo stream di output `s` il carattere ottenuto dalla valutazione dell'espressione `e` (n.b.: il carattere viene scritto nella posizione indicata dal puntatore al dato corrente contenuto nello stream `s`, che viene poi aggiornato di conseguenza).

Esempio 6.3 (Output tramite `put`) *Dato il seguente frammento di codice C++*

```
ofstream f2;
f2.open("risultati.txt");
char c = 'a';
f2.put(c);
```

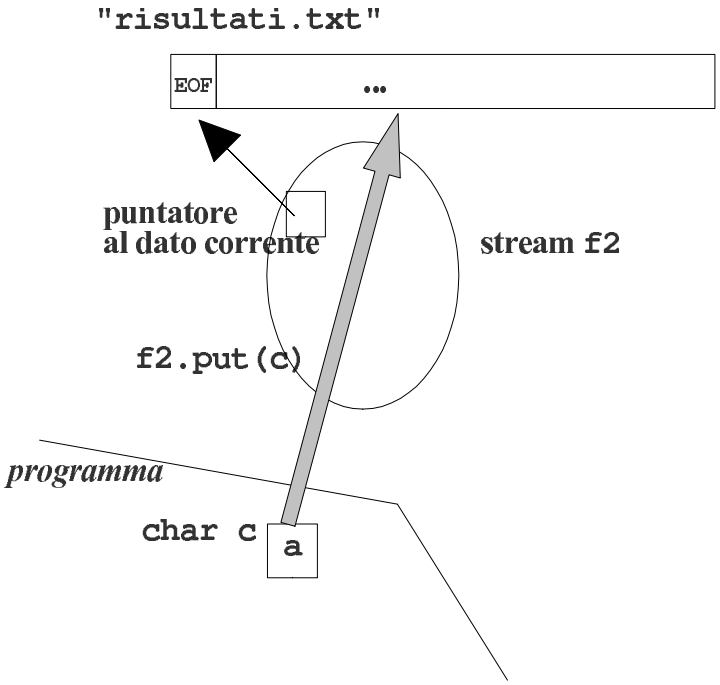

Figura 6.7: Scrittura su file tramite `put`

e supponendo che il contenuto del file sia quello indicato in figura 6.7 l'esecuzione della `put` *inserisce il carattere* `'a'` *nel file al posto di* `EOF` *che avanza alla posizione successiva. Il puntatore al dato corrente è aggiornato e punta alla nuova posizione contenente* `EOF`*. Una successiva*

```
f2.put('b');
```

inserisce `'b'` *sul file in modo analogo. Dopo queste istruzioni il file sarà pertanto diventato:*

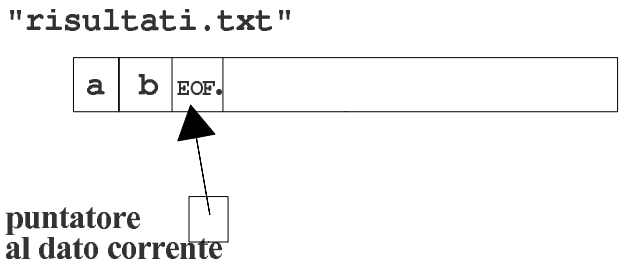

Figura 6.8: File "`risultati.txt`" modificato

6.4 Un esempio completo: copia di un file

Problema. Copiare il contenuto di un file di nome `"sorgente.txt"` in un altro file di nome `"copia.txt"`. Al termine stampare su standard output il numero di caratteri copiati.

Procedimento risolutivo. I caratteri presenti nel file `"sorgente.txt"` vengono, uno alla volta, letti dal file, memorizzati in una variabile `c`, e quindi scritti sul file `"copia.txt"`. Questo procedimento continua fino a quando si incontra l'end-of-file sul file di input.

La soluzione proposta è illustrata schematicamente in Figura 6.9.

Programma C++. Il seguente programma C++ realizza il procedimento sopra descritto.

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

const int max_nome = 100;
int main() {
   // Apre un file per l'input.
   ifstream in_file;
   in_file.open("sorgente.txt");
   // Controllo errori di apertura del file.
   if (in_file.fail()) {
      cout << "Il file non esiste!" << endl;
      return 0;
   }

   // Apre un file per l'output.
```

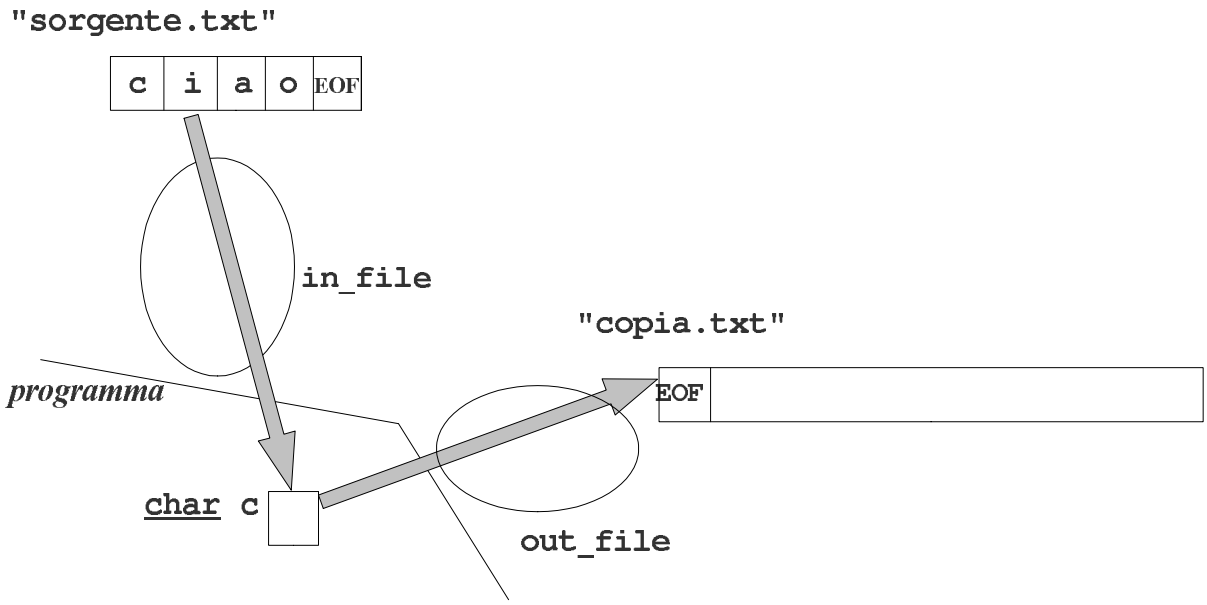

Figura 6.9: Copia di un file

```
    ofstream out_file;
    out_file.open("copia.txt");

    int num_bytes = 0;
    char c;
    c = in_file.get();
    while (!in_file.eof()) {
       out_file.put(c);
       c = in_file.get();
       num_bytes++;
    }

    cout << "Copiati " << num_bytes << " bytes.\n" << endl;

    in_file.close();
    out_file.close();
    return 0;
}
```

Al termine dell'esecuzione del programma, il file `"copia.txt"` conterrà una copia esatta del contenuto del file `"sorgente.txt"`. Si noti che il file `"copia.txt"`, se non esistente, viene creato ex-novo durante l'esecuzione del programma, mentre se già esistente verrà comunque sovrascritto.

In alternativa, il ciclo di lettura-scrittura da/su file può essere realizzato utilizzando un costrutto `do-while` nel seguente modo:

```
char c;
do {
    c = in_file.get();
    if (in_file.eof()) break;
    out_file.put(c);
    num_bytes++;
} while (true);
```

Una variante (più realistica) di questo problema prevede che il nome del file da copiare non sia prefissato, ma venga fornito in input dall'utente, mentre il nome del file di output sarà creato a partire da quello di input aggiungendo ad esso qualche altra stringa. Ad esempio, se il nome del file di input è `"dati.txt"` il nome del file di output potrebbe essere ottenuto anteponendo la stringa `"copia di "` e quindi essere `"copia di dati.txt"`.

Per realizzare questo nuovo funzionamento il programma C++ mostrato sopra viene modificato come segue (in evidenza le sole parti modificate):

```
...
#include <cstring>
...
int main() {
   // Legge il nome del file da copiare
   char sorgente[max_nome];
  ifstream in_file;
  cout << "Immettere il nome del file da copiare: ";
  cin.getline(sorgente,max_nome);
  // Apre un file per l'input
  in_file.open(sorgente);
  // Controlla errori di apertura del file
  if (in_file.fail()) {
     cout << "Impossibile aprire il file ''" << sorgente << endl;
     return 0;
  }
  // Costruisce il nome del nuovo file
  char destinazione[max_nome+9] = "Copia di ";
  strcat(destinazione,sorgente);
  // Apre un file per l'output
  ofstream out_file;
  out_file.open(destinazione);
  int num_bytes = 0;
  ...  // come versione precedente
}
```

Si osservi che nel caso in cui l'apertura del file di input fallisca abbiamo finora supposto di far semplicemente terminare il programma. In alternativa, si può far in modo che in caso di fallimento dell'apertura venga richiesto

all'utente di provare ad immettere un nuovo nome di file. Per ottenere questo, modifichiamo il programma mostrato sopra come segue:

```
...
ifstream in_file;
do {
   cout << "Immettere il nome del file da copiare: ";
   cin.getline(sorgente,max_nome);
   in_file.clear();  //reset del flag 'failbit' modificato da open
   in_file.open(sorgente);
   if (in_file.fail())
      cout << "Impossibile aprire il file. Ripetere!" << endl;
   else
      cout << "File aperto correttamente." << endl;
}
while (in_file.fail());

// Costruisce il nome del nuovo file
...
```

6.5 Input/output "tipato"

In C++ è possibile leggere/scrivere da/su file non solo singoli caratteri, ma anche valori di qualsiasi tipo primitivo *t*, utilizzando gli operatori `>>` e `<<` applicati a stream di input e di output.

In questo caso l'esecuzione dell'operazione di input/output legge/scrive caratteri da/sul file fino a comporre (se possibile) un valore di tipo *t* sintatticamente corretto (come per altro gia' illustrato nel caso degli stream di input e ouput standarxd nel cap. 2.8.1).

. . . in preparazione . . .

6.6 *Domande per il Capitolo 6*

1. Che cosa è, in generale, uno stream in C++ (definizione, uso, caratteristiche principali)?
2. Qual la sequenza tipica delle operazioni necessarie per accedere ed operare su un file tramite stream allinterno di un programma C++?.

. . . in preparazione . . .