Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno **luigi.catuogno@uniparthenope.it**
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

[FdP] H. M. Deitel, P. J. Deitel

C++ Fondamenti di programmazione

II ed. (2014) Maggioli Editore (Apogeo Education)

ISBN: 978-88-387-8571-9

3

Libri di testo

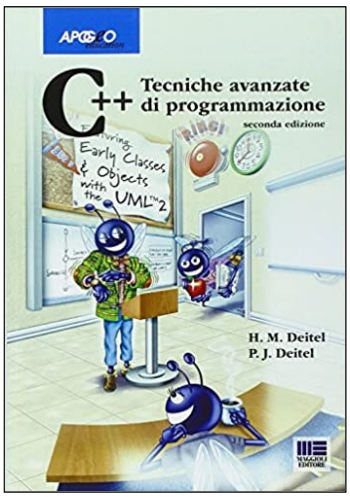
Tecniche avanzate e strutture dati elementari

[TAP] H. M. Deitel, P. J. Deitel

C++ Tecniche avanzate di programmazione

II ed. (2011) Maggioli Editore (Apogeo Education)

ISBN: 978-88-387-8572-6



4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

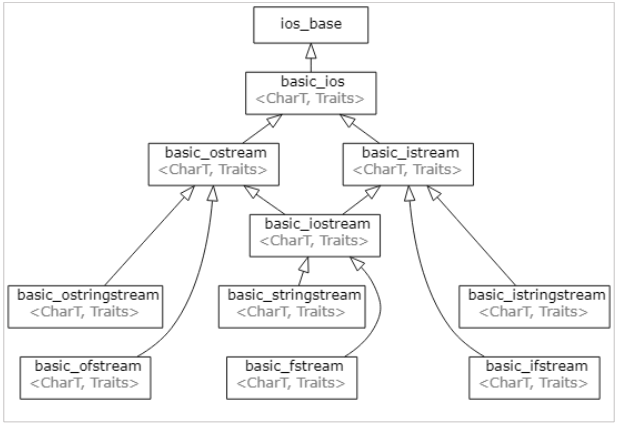
Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Input/Output su *file stream*

6

Gerarchia delle classi per lo *stream-based* I/O



7

Modalità di apertura di uno stream

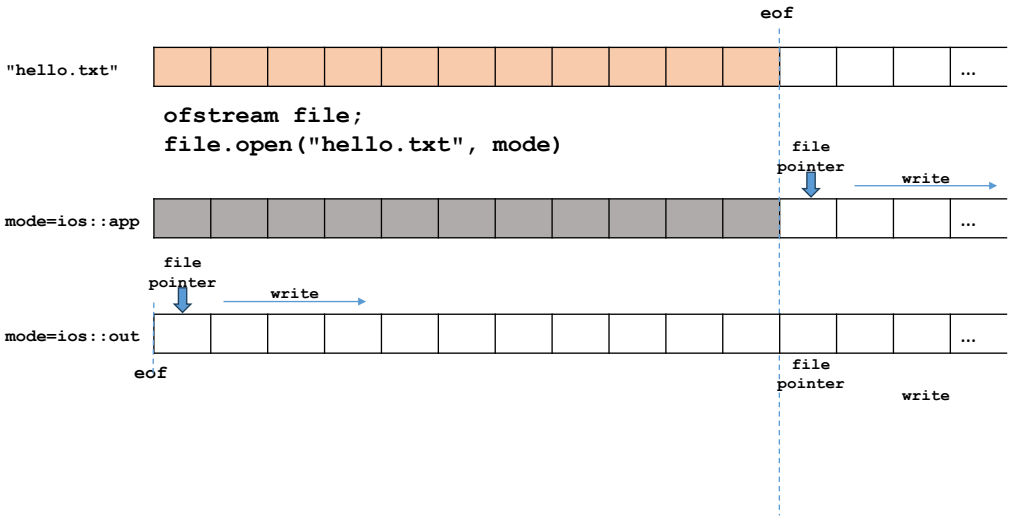
`fstream::open(filename [, ios_base::openmode mode])`

Il parametro **`mode`** indica la modalità in cui il file viene aperto e le azioni compiute in fase di apertura (*e.g.* posizionamento del *file pointer*, cancellazione dei file, proprietà…) :

`ios::app`	Aggiunge l'output alla fine del file
`ios::ate`	Apre un file e si sposta alla fine di esso, ma i dati possono essere scritti ovunque nel file
`ios::in`	Apre un file in input
`ios::out`	Apre un file in output (se il file esiste già, ne cancella il contenuto)
`ios::trunc`	Elimina il contenuto del file se esiste (è l'azione di default di **ios::out**)
`ios::binary`	Apre un file per l'input e l'output binario (non di testo)

8

Modalità di apertura di uno stream



9

Esempio: *scrivere in un file di testo #2*

file: outfile.cpp

```
 5  int main()
 6  {
 7          ofstream fout;
 8          string data;
 9          char scelta;
10          int count=0;
11          ios_base::openmode mode;
12
13          cout<<"*** Output su file ***"<<endl;
14          cout<<"Mode: (o)ut, (a)pp."<<endl;
15          cin >>scelta;
16          switch(scelta){
17                  case 'a':
18                          mode=ios_base::app;
19                          cout<<"append"<<endl;
20                          break;
21                  default:
22                  case 'o':
23                          mode=ios_base::out;
24                          cout<<"out"<<endl;
25                          break;
26          }
```

10

Esempio: *scrivere in un file di testo #2*

file: outfile.cpp

```
27  fout.open("data.txt",mode);
28
29          cout<<"Inserisci i dati: (exit per terminare)"<<endl<<endl;
30
31          do{
32                  cout<<"data> ";
33                  cin>>data;
34                  if(data=="exit")
35                          break;
36                  fout<<data<<endl;
37                  count++;
38          } while(1);
39
40          cout<<"Operazione conclusa. ";
41          cout<<"Salvate "<<count<<" righe"<<endl;
42          fout.close();
43  }
```

```
$ g++ outfile.cpp -o outfile
$
```

11

Esempio: *scrivere in un file di testo #2*

```
$ g++ outfile.cpp -o outfile
$ ./outfile
*** Output su file ***
Mode: (o)ut, (a)pp.
o
out
Inserisci i dati: (exit per terminare)

data> prima
data> seconda
data> terza
data> riga
data> exit
Operazione conclusa. Salvate 4 righe
$
```

Apriamo in modo «out» un file inesistente;
Il programma crea il file e vi salva i dati immessi dall'utente

12

Esempio: *scrivere in un file di testo #2*

```
$ g++ outfile.cpp -o outfile
$  cat data.txt
prima
seconda
terza
Riga
$./outfile
*** Output su file ***
Mode: (o)ut, (a)pp.
a
append
Inserisci i dati: (exit per terminare)

data> quarta
data> exit
Operazione conclusa. Salvate 1 righe
$
```

Controlliamo: i dati immessi sono nel file.

Apriamo il file in scrittura in modalità *append* e inseriamo una nuova riga…

13

Esempio: *scrivere in un file di testo #2*

```
$ g++ outfile.cpp -o outfile
$  cat data.txt
prima
seconda
terza
Riga
$./outfile
*** Output su file ***
Mode: (o)ut, (a)pp.
a
append
Inserisci i da

data> quarta
data> exit
Operazione cor
$
```

```
$  cat data.txt
prima
seconda
terza
riga
quarta
$
```

Visualizziamo il contenuto del file launa nuova riga è stata *aggiunta in coda* ai dati già presenti nel file.

14

Esempio: *scrivere in un file di testo #2*

```
$./outfile
*** Output su file ***
Mode: (o)ut, (a)pp.
o
out
Inserisci i dati: (exit per terminare)

data> quinta
data> exit
Operazione conclusa. Salvate 1 righe
$cat data.txt
quinta
$
```

```
$  cat data.tx
prima
seconda
terza
Riga
$./outfile
*** Output su
Mode: (o)ut, (
a
append
Inserisci i da

data> quarta
data> exit
Operazione conclusa. Salvate 1 righe
$
```

Riaprendo il file in modalità *out* si cancella il suo contenuto precedente. Al termine dell'esecuzione, il file contiene soltanto gli ultimi dati immessi.

15

Esempio: *leggere da un file di testo #2*

file: infile.cpp

```
 5  int main()
 6  {
 7          ifstream fin;
 8          string data;
 9          char scelta;
10          int count=0;
11          ios_base::openmode mode;
12
13
14          cout<<"*** Output su file ***"<<endl;
15          cout<<"Mode: (i)n, at(e)."<<endl;
16          cin >>scelta;
17          switch(scelta){
18                  case 'i':
19                          mode=ios_base::in;
20                          cout<<"in"<<endl;
21                          break;
22                  case 'e':
23                          mode=ios_base::ate;
24                          cout<<"at end"<<endl;
25                          break;
26          }
```

16

Esempio: *scrivere in un file di testo #2*

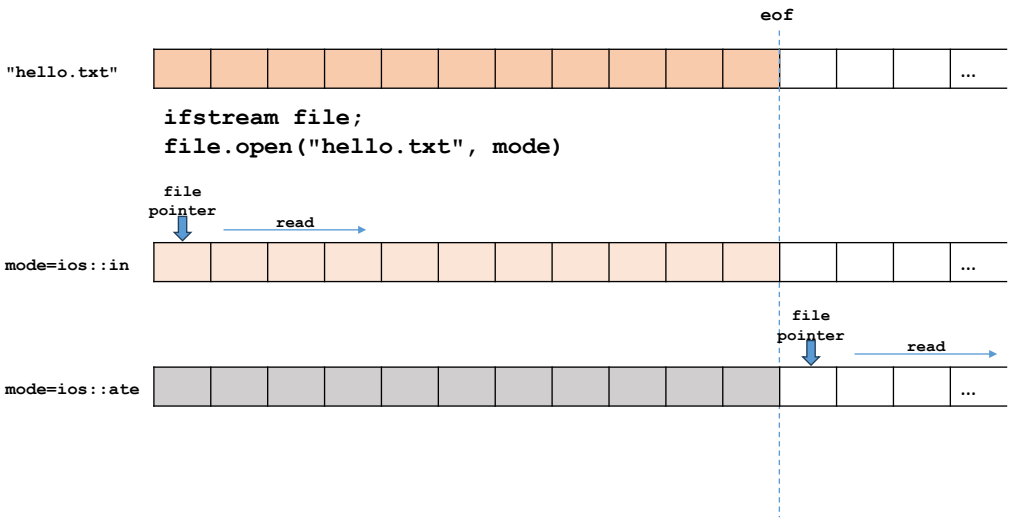
file: infile.cpp

```
27  fin.open("data.txt",mode);
28          if(!fin){
29                  cout<<"Impossibile aprire il file"<<endl;
30                  exit(0);
31          }
32
33          while(fin>>data) {
34                  cout << "data: "<<data<<endl;
35                  count++;
36          }
37
38          cout<<"Operazione conclusa. ";
39          cout<<"lette "<<count<<" righe"<<endl;
40          fin.close();
41  }
```

```
$ g++ infile.cpp -o infile
$
```

17

Modalità di apertura di uno stream



18

Esempio: *leggere da un file di testo #2*

```
$./infile
*** Input da file ***
Mode: (i)n, at(e).
i
in
data: prima
data: seconda
data: terza
data: riga
data: quarta
$
$./infile
Mode: (i)n, at(e).
e
at end
Operazione conclusa. lette 0 righe
$
```

19

I/O sequenziale in un file di testo

Scrittura:

- I nuovi dati sono aggiunti sempre *in coda* al file;
- L'I/O sequenziale è orientato ai byte. Pertanto, qualsiasi strutturazione dei dati deve essere gestita dall'applicazione (*e.g.* separazione dei campi e dei record…)

20

I/O sequenziale in un file di testo

Lettura:

- Si inizia la lettura dall'inizio del file e si prosegue verso la fine;
- Per raggiungere i dati richiesti, occorre *«scorrere»* tutti quelli che li precedono

21

Esempio: *Una rubrica telefonica*

I contatti:

- In memoria, sono rappresentati dagli oggetti di classe **`contatto`**, che utilizzano due attributi di classe **`string`**: **`nome`** (comprensivo di spazi) e **`numero`**;
- Nel «database» (un file di testo sequenziale), i contatti sono rappresentati ciascuno su una singola riga: un *record*, col seguente formato:

```
nome:numero
```

22

Esempio: *Una rubrica telefonica*

Funzionalità:

- Inserimento di nuovi contatti
- Ricerca di uno specifico contatto (per nome o per numero)
- Elenco di tutti i contatti

23

Esempio: *Una rubrica telefonica*

file: RT_contatto.hpp

```
 5  class contatto {
 6  private:
 7          string nome;
 8          string numero;
 9  public:
10          contatto();
11          contatto(string);
12          contatto(string, string);
13          ~contatto();
14
15          static const int  maxnome=20;
16          static const int  maxnumero=16;
17          static const char sep=':';
18          static string extrnome(string);
19          static string extrnumero(string);
20
21          contatto&  setnome(string);
22          contatto&  setnumero(string);
23          string getnome();
24          string getnumero();
25          string getcontact();
26  };
```

Costruttori e distruttore

Costanti e metodi di classe:
lunghezza massima dei campi;
separatore dei campi nei record (righe) nel file;
metodi per l'estrazione dei campi dai record.

24

Esempio: *Una rubrica telefonica*

file: RT_contatto.hpp

```
 5  class contatto {
 6  private:
 7          string nome;
 8          string numero;
 9  public:
10          contatto();
11          contatto(string);
12          contatto(string, string);
13          ~contatto();
14
15          static const int  maxnome=20;
16          static const int  maxnumero=16;
17          static const char sep=':';
18          static string extrnome(string);
19          static string extrnumero(string);
20
21          contatto&  setnome(string);
22          contatto&  setnumero(string);
23          string getnome();
24          string getnumero();
25          string getcontact();
26  };
```

Metodi di accesso agli attributi privati. Metodo di «codifica» del record (rigadel contatto per il file)

25

Esempio: *Una rubrica telefonica*

file: RT_contatto.cpp

```
 5  contatto::contatto(string no, string num)
 6  {
 7          nome=no;
 8          numero=num;
 9  }
10
11  contatto::contatto(string co)
12  {
13          nome=extrnome(co);
14          numero=extrnumero(co);
15  }
16
17  contatto::contatto() {}
18
19  contatto::~contatto(){}
```

Questo costruisce un'istanza della classe **contatto** a partire da un record letto dal file.

Questi estraggono dal record i due campi e li assegnano all'attributo corrispondente.

26

Esempio: *Una rubrica telefonica*

file: RT_contatto.cpp

```
20  string contatto::extrnome(string riga)
21  {
22          int possep;
23
24          possep=riga.find(sep);
25          return riga.substr(0,possep);
26  }
27  string contatto::extrnumero(string riga)
28  {
29          int possep;
30
31          possep=riga.find(sep);
32          return riga.substr(possep+1,riga.size());
33  }
34  string contatto::getcontact()
35  {
36          return nome+sep+numero;
37  }
```

Cerca la posizione in cui si trova la *prima occorrenza da sx* del carattere separatore (**':'**) nella stringa **riga**;

Estrae la *sottostringa* compresa tra le posizioni **0** e **possep** dalla stringa **riga** (che contiene il campo **nome**).

Costruisce un record a partire dagli attributi dell'oggetto.

27

...metodi della classe **string**

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

```
string substr( size_type pos = 0, size_type count = npos ) const;
```

estrare e restituisce una nuova stringa dall'istanza data.

pos: posizione del primo carattere della sottostringa

npos: numero dei caratteri (lunghezza) della sottostringa

Nota bene: la libreria C++ offre più versioni di questi metodi, a seconda di diversi modi di impiego e dei tipi di stringhe coinvolte.
https://cppreference.com

28

…metodi della classe **string**

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

```
size_type find( CharT ch, size_type pos = 0 ) const;
```

Cerca la prima occorrenza del carattere ch (eventualmente a partire da pos) ne restituisce la posizione.

ch: carattere cercato

pos: posizione da cui iniziare la ricerca

Nota bene: la libreria C++ offre più versioni di questi metodi, a seconda di diversi modi di impiego e dei tipi di stringhe coinvolte. https://cppreference.com

29

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
1  #include<iostream>
2  #include<string>
3  #include<iomanip>
4  #include<fstream>
5  #include<cstdlib>
6  #include "RT_contatto.hpp"
7
8  using namespace std;
9
10 int main()
11 {
12         fstream RTfile;
13         char scelta;
14         string nome, numero, riga,temp;
15         contatto persona;
…  …
```

Utilizziamo stream di classe **fstream** poiché accederemo al file simultaneamente in lettura e in scrittura.

30

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
16      RTfile.open("RT.dat",(ios::in|ios::app));
17
18      if(!RTfile) {
19              cout<<"Impossibile aprire il file"<<endl;
20              exit(0);
21      }
22      do {
23              cout<<"------------ RT Menu' Principale   ------------"<<endl;
24              cout<<"(i)nserisci, (c)erca nome, cerca (n)umero,"<<endl;
25              cout<<"(l)ista contatti, (e)sci"<<endl;
26              cout<<"Scelta: ";
27              cin>>scelta;
28              cin.ignore();
29
30              switch(scelta) {
…
```

Apriamo il file RT.dat in lettura e con scrittura in coda (append).

- Si può scorrere il file dall'inizio fino a eof
- Si possono aggiungere nuovi dati solo alla fine del file (che viene di conseguenza spostata più avanti…)

31

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
16      RTfile.open("RT.dat",(ios::in|ios::app));
17
18      if(!RTfile) {
19              cout<<"Impossibile aprire il file"<<endl;
20              exit(0);
21      }
22      do {
23              cout<<"------------ RT Menu' Principale   ------------"<<endl;
24              cout<<"(i)nserisci, (c)erca nome, cerca (n)umero,"<<endl;
25              cout<<"(l)ista contatti, (e)sci"<<endl;
26              cout<<"Scelta: ";
27              cin>>scelta;
28              cin.ignore();
29
30              switch(scelta) {
…
```

Quando immettiamo un carattere, nella variabile scelta, ci va solo questo. Il newline successivo resta nel buffer e può disturbare le operazioni di input successive. Il metodo **`ignore`** scarta il primo carattere dal buffer dello stream.

32

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
31                          case 'i':
32                                  cout<<"Nome: ";
33                                  getline(cin,nome);
34                                  cout<<"Numero: ";
                                    cin>>numero;
                                    persona.setnome(nome).setnumero(numero);
                                    RTfile << persona.getcontact()<<endl;
                                    break;
```

L'operatore **>>** di uno stream di input accetta i dati necessari a riempire il suo operando fino a un separatore (spazio per le stringhe)
La funzione di libreria **getline()** consente di riempire la stringa nome con una sequenza che possa contenere uno o più spazio. Qui il separatore è **'\n'**;

https://en.cppreference.com/w/cpp/string/basic_string/getline

33

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
31                          case 'i':
32                                  cout<<"Nome: ";
33                                  getline(cin,nome);
34                                  cout<<"Numero: ";
                                    cin>>numero;
                                    persona.setnome(nome).setnumero(numero);
                                    RTfile << persona.getcontact()<<endl;
                                    break;
```

L'operatore **>>** di uno stream di input accetta i dati necessari a riempire il suo operando fino a un separatore (spazio per le stringhe)
La funzione di libreria **getline()** consente di riempire la stringa nome con una sequenza che possa contenere uno o più spazio. Qui il separatore è **'\n'**;

Scriviamo il record composto dall'utente in coda al file. E' necessario ricordare di mettere **endl** poiché questo carattere funge da separatore tra i record.

https://en.cppreference.com/w/cpp/string/basic_string/getline

34

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
39        case 'l':
40                RTfile.seekg(0);
41                while(getline(RTfile,riga)){
42                     cout << "| "<<setw(contatto::maxnome)<<left;
43                     cout <<contatto::extrnome(riga)<<" | ";
44                     cout << setw(contatto::maxnumero)<<left;
45                     cout <<contatto::extrnumero(riga)<<" |"<<endl;
46                }
47
48                RTfile.clear();
49                break;
…
```

Per effettuare la scansione in lettura dell'intero file, occorre posizionare ogni volta il *file pointer* all'inizio del file. Infatti, potrebbe trovarsi all'eof per effetto di una scansione precedente… In questo caso, il metodo **`seekg()`**, posiziona il file pointer all'*offset* (posizione nel file) indicata dal parametro.

35

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
39        case 'l':
40                RTfile.seekg(0);
41                while(getline(RTfile,riga)){
42                     cout << "| "<<setw(contatto::maxnome)<<left;
43                     cout <<contatto::extrnome(riga)<<" | ";
44                     cout << setw(contatto::maxnumero)<<left;
45                     cout <<contatto::extrnumero(riga)<<" |"<<endl;
46                }
47
48                RTfile.clear();
49                break;
…
```

La fuzione di libreria **`getline()`**, effettua l'input dell'intera riga dallo stream indicato dal primo parametro e restituisce il riferimento allo stesso stream. Si ricordi che questo, a sua volta, risulterà **`false`**, quando sarà raggiunta la fine del file.

36

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
39        case 'l':
40                RTfile.seekg(0);
41                while(getline(RTfile,riga)){
42                     cout << "| "<<setw(contatto::maxnome)<<left;
43                     cout <<contatto::extrnome(riga)<<" | ";
44                     cout << setw(contatto::maxnumero)<<left;
45                     cout <<contatto::extrnumero(riga)<<" |"<<endl;
46                }
47
48                RTfile.clear();
49                break;
…
```

Si ricordi che una volta raggiunta la fine del file, lo stato dello stream non è più good. Occorre quindi ripristinarlo prima di effettuare qualsiasi altra operazione.

La fuzione di libreria **getline()**, effettua l'… parametro e restituisce il riferimento allo stesso stream. Si ricordi che questo, a sua volta, risulterà **false**, quando sarà raggiunta la fine del file.

37

Esempio: *Una rubrica telefonica*

file: RT_main.cpp

```
…
50        case 'c':
51                cout<<"Nome: ";
52                getline(cin,nome);
53                RTfile.seekg(0);
54                while(getline(RTfile,riga)){
55                     temp=contatto::extrnome(riga);
56                     if(temp.compare(0,nome.size(),nome)==0){
57                          cout <<"trovato: "<<contatto::extrnome(riga)<<" ";
                            cout <<contatto::extrnumero(riga)<<endl;


                     lear();
…
```

Input del nome da cercare (event. spazi compresi)

Posizionamento all'inizio del file

Scorriamo il file una riga alla volta. Fino a eof. Da ciascun record letto estraiamo il nome

38

Esempio: *Una rubrica telefonica*

```
…
50          case 'c':
51                  cout<<"Nome: ";
52                  getline(cin,nome);
53                  RTfile.seekg(0);
54                  while(getline(RTfile,riga)){
55                       temp=contatto::extrnome(riga);
56                       if(temp.compare(0,nome.size(),nome)==0){
57                             cout <<"trovato: "<<contatto::extrnome(riga)<<" ";
58                             cout <<contatto::extrnumero(riga)<<endl;
59                       }
60                  }
61                  RTfile.clear();
62                  break;
…
```

Confronto «parziale». Qui si verifica se il nome inserito è almeno un *prefisso* del nome contenuto nel record in esame. Nel caso, la corrispondenza è trovata.

La funzionalità elenca tutte le occorrenze nel file del nome dato, fino alla fine del file. Al termine, occorre ripristinare lo stato dello stream.

39

I/O sequenziale in un file di testo

Cancellazione:

- La cancellazione di dati in un file sequenziale non è un'operazione molto frequente e in generale è piuttosto scomoda

40

I/O sequenziale in un file di testo

Cancellazione:

- La cancellazione
  molto frequent

Un po' come sfilare un foglio nel mezzo di una pila di documenti…

41

I/O sequenziale in un file di testo

Cancellazione:

- La cancellazione di dati in un file sequenziale non è un'operazione molto frequente e in generale è piuttosto scomoda.
- In linea di principio si procede a:

  Cancellare solo i dati *in coda i.e.* da una certa posizione, fino alla fine del file (*troncamento*)

  Cancellare dei dati *interni* e procedere allo scorrimento *verso il basso* di tutti i dati successivi.

42

I/O sequenziale in un file di testo

Cancellazione:

- La cancellazione di dati in un file sequenziale non è un'operazione molto frequente e in generale è piuttosto scomoda.
- In linea di principio si procede a:

In C++ standard non si può fare (si usano le system call del SO)

Cancellare solo i dati *in coda i.e.* da una certa posizione, fino alla fine del file (*troncamento*)

Cancellare dei dati *interni* e procedere allo scorrimento *verso il basso* di tutti i dati successivi.

43

I/O sequenziale in un file di testo

Cancellazione:

- La cancellazione di dati in un file sequenziale non è un'operazione molto frequente e in generale è piuttosto scomoda.
- In linea di principio si procede a:

Cancellare solo i dati *in coda i.e.* da una certa posizione, fino alla fine del file (*troncamento*)

Questo è più laborioso e ci porta al seguente...

Cancellare dei dati *interni* e procedere allo scorrimento *verso il basso* di tutti i dati successivi.

44

Esercizio: *…cancellare un record in RT*

Si estenda l'applicazione RT con la funzionalità «cancella record» che:

- Chiede all'utente di indicare il nome del/dei contatti da cancellare
- Scorre dall'inizio la rubrica e per ogni corrispondenza, chiede all'utente se vuole cancellarla o meno.
- In caso di assenso procede alla cancellazione.

45

Esercizio: *…cancellare un record in RT*

Schema di soluzione:

Si crea un nuovo file (**`RT.dat.new`**)

Si procede alla ricerca dei record nel vecchio file. Tutti i record per cui non c'è corrispondenza con *la chiave* (*i.e.* il nome indicato dall'utente) vengono trascritti nel nuovo file.

Quando si verifica la corrispondenza, il programma visualizza il record trovato e chiede all'utente se vuole cancellarlo.

In caso di assenso, il programma continua la scansione del vecchio file, senza trascrivere il record nel nuovo file

In caso contrario, il record viene anch'esso trascritto nel nuovo file.

Si procede fino a raggiungere EOF nel vecchio file. Quindi si chiudono entrambi i file….

46

Esercizio: *…cancellare un record in RT*

Schema di soluzione (continua…)

Si cancella il vecchio file

Si rinomina il nuovo file da **`RT.dat.new`** in **`RT.dat`**

Possono essere utili le funzioni di libreria

```
int rename( const char* old_filename, const char* new_filename );
```
https://en.cppreference.com/w/cpp/io/c/rename

```
int remove( const char* pathname );
```
https://en.cppreference.com/w/cpp/io/c/remove

47