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

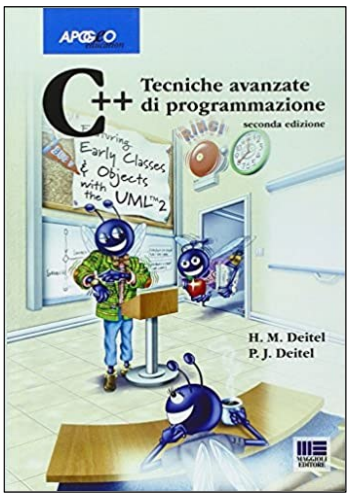
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

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

Input/Output su *file stream*

Le funzionalità di I/O del C++ si basano sull'astrazione degli *stream*: che rappresentano un *flusso di dati* stabilito:

- da una sorgente (un dispositivo di input, un file in lettura) alla memoria del programma: è il flusso in input o *istream*
- dalla memoria del programma verso una destinazione esterna: è il flusso di output o *ostream*

7

Input/Output su *file stream*

L'interfaccia esposta dagli stream offre il vantaggio di trattare l'I/O dei dati nel programma in maniera uniforme, indipendentemente dalla tipo di entità associata all'altra estremità del flusso.

- Per stabilire un flusso di dati con una entità esterna al programma (sia essa un file o un dispositivi di I/O) è necessario *aprire lo stream.* Le API degli stream forniscono, allo scopo, il metodo **`open()`**
- Al termine del trasferimento dei dati, lo stream deve essere *chiuse* mediante il metodo **`close()`**

8

Input/Output su *file stream*

Principalmente, con la **`open()`**, il programma indica :

- l'entità con cui si entra in comunicazione mediante lo stream (tipicamente un file)
- la direzione del flusso: input, output o entrambi;
- altre proprietà del flusso (*e.g.* tipo di dati in transito, modalità di accesso…)

9

Input/Output su *file stream*

Una volta inizializzato lo stream, il programma può:

- inviare dati dalla sua memoria al dispositivo esterno tramite lo stream utilizzando l'operatore di *immissione* **<<**
- caricare in memoria i dati provenienti dalla sorgente esterna, attraverso lo stream, mediante l'operatore di *estrazione* **>>**
- interrogare lo *stato* dello stream (errori, altre condizioni di interesse…)
- gestire il trasferimento dei dati con metodi più *«specializzati»* e in diverse *modalità*

10

Input/Output su *file stream*

Quando un qualsiasi programma C++ è avviato, il Sistema Operativo gli fornisce tre stream già aperti. Questi sono istanziati da oggetti che conosciamo già:

- **`cin`** che rappresenta lo stream di input standard da tastiera;
- **`cout`**, che rappresenta l'output standard su video
- **`cerr`**, che rappresenta un secondo canale di output su console, dedicato ai messaggi di errore.

11

Esempio: *scrivere in un file di testo…*

```
1  #include<iostream>
2  #include<fstream>
3  using namespace std;
4  int main() {
5          ofstream fout;
6          string nome;
7          float altezza;
8          int anno;
9          cout<<"Nome: ";
10         cin>>nome;
11         cout<<"Altezza: ";
12         cin>>altezza;
13         cout<<"Anno: ";
14         cin>>anno;
15
16         fout.open("dati.txt");
17         fout<<"Nome: "<<nome<<endl;
18         fout<<"Altezza: "<<altezza<<endl;
19         fout<<"Anno di nascita: "<<anno <<endl;
20         fout.close();
21 }
```

Header file da includere per le operazioni su file stream.

Dichiariamo un oggetto di tipo **`ofstream`**, (*i.e.* output file stream). **`fout`**

Apriamo il file **`dati.txt`**: d'ora in avanti, tramite l'oggetto **`fout`**, scriveremo dati nel file con l'operatore di immissione **<<;**

12

Esempio: *scrivere in un file di testo…*

```
1  #include<iostream>
2  #include<fstream>
3  using namespace std;
4  int main() {
5          ofstream fout;
6          string nome;
7          float altezza;
8          int anno;
9          cout<<"Nome: ";
10         cin>>nome;
11         cout<<"Altezza: ";
12         cin>>altezza;
13         cout<<"Anno: ";
14         cin>>anno;
15
16         fout.open("dati.txt");
17         fout<<"Nome: "<<nome<<endl;
18         fout<<"Altezza: "<<altezza<<endl;
19         fout<<"Anno di nascita: "<<anno <<endl;
20         fout.close();
21 }
```

Scrittura dei dati nel file.

Chiusura del file. L'oggetto **`fout`**, viene distrutto (dopo aver svuotato il buffer). Il flusso viene interrotto.

13

Esempio: *scrivere in un file di testo…*

```
1  #include<iostream>
2  #include<fstream>
3  using namespace std;
4  int main() {
5          ofstream fout;
6          string nome;
7          float altezza;
8          int anno;
9          cout<<"Nome: ";
10         cin>>nome;
11         cout<<"Altezza: ";
12         cin>>altezza;
13         cout<<"Anno: ";
14         cin>>anno;
15
16         fout.open("dati.txt");
17         fout<<"Nome: "<<nome<<endl;
18         fout<<"Altezza: "<<altezza<<endl;
19         fout<<"Anno di nascita: "<<anno <<endl;
20         fout.close();
21 }
```

```
$ g++ fdati.cpp -o fdati
$./fdati
Nome: Pippo
Altezza: 1.86
Anno: 1932
$ cat dati.txt
Nome: Pippo
Altezza: 1.86
Anno: 1932
$
```

14

Esempio: *leggere da un file di testo…*

```
 5  int main() {
 6          ifstream fin;
 7          string tag, nome;
 8          float altezza;
 9          int anno;
10          fin.open("dati.txt")
11          if(!fin) {
12                  cerr << "Impossibile aprire il file"<<endl;
13                  exit(0);
14          }
15          fin >> tag;
16          cout << "leggo "<<tag;
17          fin >> nome;
18          cout << nome << endl;
19          fin >> tag;
20          cout << "leggo "<<tag;
21          fin >> altezza;
22          cout << altezza <<endl;
23          fin >> tag >> altezza;
24          cout << "leggo "<<tag << altezza <<endl;
25          fin.close();
26  }
```

Apertura del file in lettura (**fin** è un'istanza dell'oggetto **ifstream**)

Il metodo **open()**, restituisce **0** se, quacosa va storto durante l'apertura (*e.g.*il file non esiste).

Operazioni di lettura con l'operatore di estrazione **>>;**

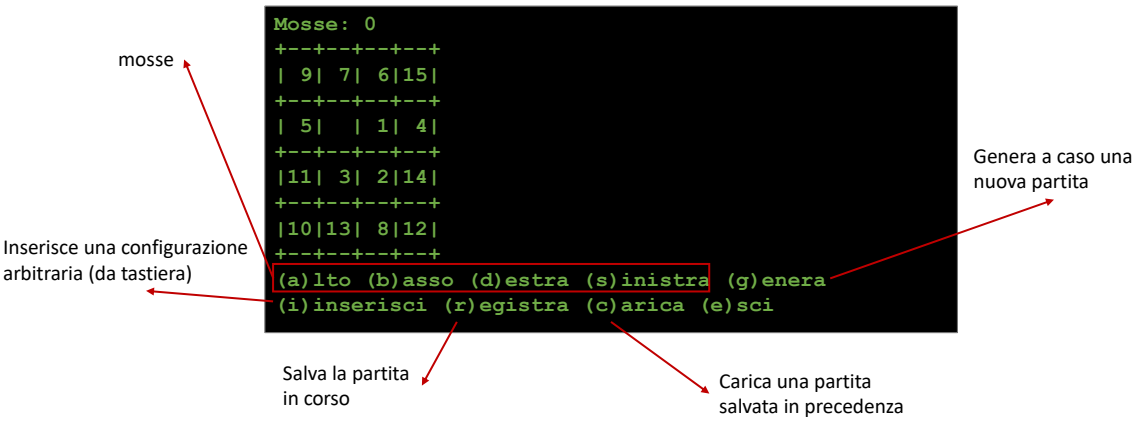
15

Il gioco del 15 *(reprise)*

16

Esempio: *Il gioco del 15*

Aggiungiamo al gioco del 15 la possibilità di salvare/ricaricare una partita su file.



Esempio: *Il gioco del 15…*

file: schema15.hpp

```
 1  class schema15 {
 2          private:
 3                  int pad[16], num_mosse, vX, vY;
 4                  const int maxiter=1000;
 5          public:
 6                  schema15();
 7                  schema15(int);
 8                  bool inizializza();
 9                  bool inizializza(int *);
10                  bool inizializza(int *, int);
11                  bool genera(int);
12                  void mischia(int);
13                  bool valido();
14                  bool vinto();
15                  bool alto();
16                  bool basso();
17                  bool destra();
18                  bool sinistra();
19                  int mosse();
20                  void conf(int *);
21                  void mostra();
22  };
```

Inizializza lo schema riempendolo con l'array dato e impostando un numero di mosse arbitrario.

Restituisce una copia dell'array privato **pad[]**. Il parametro deve puntare a un array già allocato.

Esempio: *Il gioco del 15…*

file: main.hpp

```
…
int main()
{
        int seed,tmp[16],nm=0;
…
        cout << "seed: ";
        cin >>seed;
        schema15 x(seed);
        do {
                x.mostra();
                cout << "(a)lto (b)asso (d)estra (s)inistra (g)enera"<<endl;
                cout << "(i)inserisci (r)egistra (c)arica (e)sci"<<endl;
                cin >> scelta;
                switch (scelta) {
                        case 'a':
                            if(!x.alto())
                                    cout << "Mossa non valida!"<<endl;
                            break;
…
                        case 'g':
                            cout<<"Genero una nuova configurazione. Seed? ";
                            cin >> seed;
                            if(!x.genera(seed)){
…
```

19

Esempio: *Il gioco del 15…*

file: main.cpp

```
…
…
 case 'r':
                                  outp.open(savedgame) ;
                                  outp << x.mosse()<<endl;
                                  x.conf(tmp);
                                  for(int i=0;i<16;i++)
                                          outp<<tmp[i]<<endl;
                                  cout<<"Partita salvata"<<endl;
                                  outp.close();
                                  break;
…
```

Apre il file delle partite salvate **savedgame** è una costante (*e.g.* **savegame.dat**).

Chiude il file

Scrive il numero di mosse giocate e la configurazione nel file.

20

Esempio: *Il gioco del 15…*

file: main.cpp

```
71  case 'c':
72                          inp.open(savedgame);
73                          if(!inp){
74                                  cout << "Nessuna partita salvata!"<<endl;
75                                  break;
76                          }
77                          inp>>nm; // numero mosse
78                          for(int i=0;i<16;i++)
79                                  inp>>tmp[i];
80                          if(!x.inizializza(tmp,nm)) {
81                              cout << "Conf. errata o non
    …                                                       risolvibile"<<endl;
82                              x.inizializza();
83                          }
84                          else
85                              cout << "Partita caricata" <<endl;
86                          inp.close();
87                          break;
```

Apre il file delle partite salvate **savedgame** e segnala un errore, nel caso non sia possibile aprirlo (non c'è?)

21

Esempio: *Il gioco del 15…*

file: main.cpp

```
71  case 'c':
72                          inp.open(savedgame);
73                          if(!inp){
74                                  cout << "Nessuna partita salvata!"<<endl;
75                                  break;
76                          }
77                          inp>>nm; // numero mosse
78                          for(int i=0;i<16;i++)
79                                  inp>>tmp[i];
80                          if(!x.inizializza(tmp,nm)) {
81                              cout << "Conf. errata o non
    …                                                       risolvibile"<<endl;
82                              x.inizializza();
83                          }
84                          else
85                              cout << "Partita caricata" <<endl;
86                          inp.close();
87                          break;
```

Legge dal file il numero di mosse compiute nella partita salvata e la configurazione.

Inizializza la partita con la configurazione caricata.
In caso di errore, resetta lo schema con «1,2,3,4…15»

Chiude il file.

22

Interrogare lo *stato* di uno stream

Uno stream possiede uno *stato* che può essere *corretto* o *errato.* Uno stream in stato di errore non può essere usato.

Errori in lettura:

- Apertura di un file che non esiste
- Fine del file (**`end-of-file o EOF`**)
- Discordanza nel tipo dei dati in lettura (*e.g.* lettura di un intero, quando il dato letto non lo è)
- Apertura di un file in cui non è concesso leggere.

23

Interrogare lo *stato* di uno stream

Uno stream possiede uno *stato* che può essere *corretto* o *errato.* Uno stream in stato di errore non può essere usato.

Errori in scrittura:

- Apertura di un file in scrittura in una directory in cui non è consentito scrivere;
- Apertura di un file in cui non è permesso scrivere
- ... Altri errori del FS (*e.g.* spazio insufficiente…)

24

Interrogare lo *stato* di uno stream

Interrogare lo stato di uno stream:

```
istream fin("file-input");
…
If (!fin) {
   // … gestione errore …
}
```

Si può valutare lo stream interpretandolo come un tipo **`bool`**. Per esempio, la condizione su riportata è vera quando si verifica una condizione d'errore.

25

Interrogare lo *stato* di uno stream

Sono definiti una serie di metodi che forniscono, all'occorrenza, una diagnosi più precisa:

`fin.eof()`	**`true`** quando viene effettuata una lettura oltre la fine del file
`fin.fail()`	**`true`** quando si verifica un *errore di formattazione*, quando un programma sta operando l'input di un valore numerico e invece trova un carattere. In questo caso, i dati ancora nello stream non sono perduti. E' possibile recuperare.
`fin.bad()`	**`true`** quando si verifica un errore *non recuperabile*. In questo caso, i dati ancora nello stream sono perduti e lo stream stesso è compromesso.
`fin.good()`	**`true`** quando lo stream non è in nessuno degli stati precedenti.

26

Interrogare lo *stato* di uno stream

Lo stato di uno stream è codificato in una *parola di stato* in cui, ciascun bit indica una delle condizioni appena viste.

L'intera parola può essere ottenuta mediante il metodo **`rdstate()`**

`fin.rdstate()`	restituisce un valore di tipo intero (o simile) che è possibile ispezionare mediante *maschere di bit*, per rilevare le condizioni dello stream.
`failbit`	**`(std::ios_base::failbit)`** vero nelle condizioni di errore recuperabile.
`eofbit`	vero quando si è raggiunta fine del file
`badbit`	vero nelle condizioni di errore non recuperabile
`goodbit`	vero se **`failbit`**, **`eofbit`** e **`badbit`** sono falsi

27

Esempio: lettura da file di lunghezza arbitraria

La condizione di EOF si verifica quando si effettua un'operazione di lettura che oltrepassa la fine de lfile

```
ifstream fin("interi.txt");
if (!fin)
    cout << "Errore nell'apertura del file" << endl;
else {
    int n;
    fin >> n;
    while (fin) {
        cout << n << endl;
        fin >> n;
    }
}
fin.close();
```

Legge nel file fino che non si veifica un errore (*e.g.* end of file)

28

Esempio: lettura da file di lunghezza arbitraria

```
file: numeri.txt
3 42 -7 .
```

```
10      ifstream fin("numeri.txt");
11      if (!fin)
12          cout << "Errore nell'apertura del file" << endl;
13      else {
14          int n;
15          fin >> n;
16          while (fin) {
17              cout << n << endl;
18              fin >> n;
19          }
20          fin.clear();
21          char c;
22          fin >> c;
23          if (!fin)
24              cout << "Errore nella lettura" << endl;
25          else
26              cout << c << endl;
27  }
28  fin.close();
```

Queste letture proseguono fino a quando lo stream incontra un dato che non può essere contenuto nella variabile di destinazione

Prima di provare a leggere il carattere, occorre resettare lo stato dello stream.

29

Esempio: *Interrogare lo stato di uno stream*

```
1   #include<iostream>
2   #include<iomanip>
3   #include<string>
4   #include<fstream>
5   #include<cstdlib>
6   using namespace std;
7
8   void diag(ifstream &x)
9   {
10          cout<<"eof    :"<<x.eof()<<endl;
11          cout<<"fail   :"<<x.fail()<<endl;
12          cout<<"bad    :"<<x.bad()<<endl;
13          cout<<"good   :"<<x.good()<<endl;
14          cout<<"rdstate:"<<x.rdstate()<<endl;
15  }
16  …
```

30

Esempio: *Interrogare lo stato di uno stream*

```
…
17 int main()
18 {
19         int dato,cnt=0;
20
21         ifstream fin;
22         fin.open("intsequenza.dat");
23         if(!fin) {
24                 cerr << "errore nell'apertura del file"<<endl;
25                 exit(0);
26         }
27         while(fin>>dato)
28                 cnt++;
29         cout << "Letti "<<cnt<<" dati."<<endl;
30         diag(fin);
31         fin.close();
32 }
```

31

Esempio: *Interrogare lo stato di uno stream*

```
…
17 int main()
18 {
19         int dato,cnt=0;
20
21         ifstream fin;
22         fin.open("intsequenza.d
23         if(!fin) {
24                 cerr << "errore
25                 exit(0);
26         }
27         while(fin>>dato)
28                 cnt++;
29         cout << "Letti "<<cnt<<
30         diag(fin);
31         fin.close();
32 }
```

file: intsequenza.dat

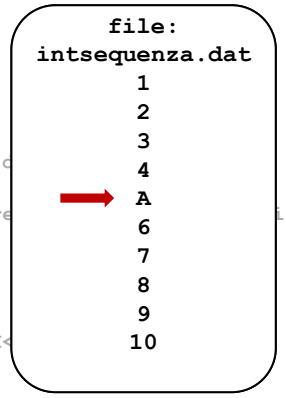
1
2
3
4
5
6
7
8
9
10

```
Letti 10 dati.
eof    :1
fail   :1
bad    :0
good   :0
rdstate:6
```

32

Esempio: *Interrogare lo stato di uno stream*

```
…
17 int main()
18 {
19         int dato,cnt=0;
20
21         ifstream fin;
22         fin.open("intsequenza.d
23         if(!fin) {
24                 cerr << "errore
25                 exit(0);
26         }
27         while(fin>>dato)
28                 cnt++;
29         cout << "Letti "<<cnt<<
30         diag(fin);
31         fin.close();
32 }
```




```
Letti  4 dati.
eof    :0
fail   :1
bad    :0
good   :0
rdstate:4
```

33

34