Alessio Bechini
- Corso di Fondamenti di Informatica II -

Input / Output attraverso stream

(anno accademico 2002-2003)

I/O in C vs. I/O in C++

Fondamenti di Informatica II

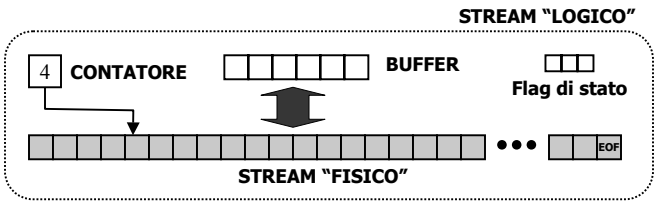
La libreria C standard per l'I/O viene acceduta con *stdio.h*

- ***stdio*** non ha una gestione OO dell'I/O, ma tutte le sue funzionalità sono contenute (e riorganizzate) nella libreria ***iostream*** ; essa viene di solito utilizzata insieme agli ADT dell'utente, con appropriata ridefinizione degli operatori << e >>.
- Per poter usare contemporaneamente *stdio* e *iostream* (ma possibilmente *non sullo stesso stream*), evitando problemi di sincronizzazione (le due librerie potrebbero usare politiche diverse di bufferizzazione), si invocano preventivamente speciali funzioni di libreria quali, p.es., *ios::synch_with_stdio()*.

Il concetto di "stream"

Fondamenti di Informatica II

Uno stream è un'astrazione che si riferisce a un flusso di dati da un'origine *(produttore)* a una destinazione *(consumatore)*.



Le classi "iostream" permettono di convertire gli oggetti in un determinato tipo in testo immediatamente leggibile (e viceversa), oltre a leggere e scrivere dati in forma binaria.

Utilizzo di stream per I/O 3

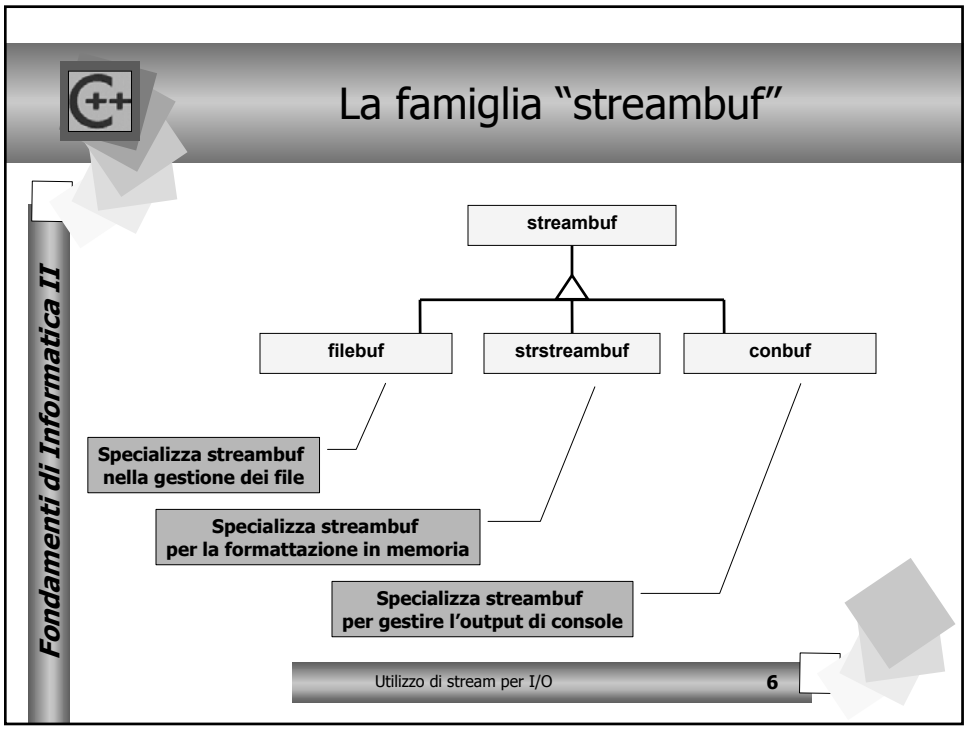
Stream di testo e binari

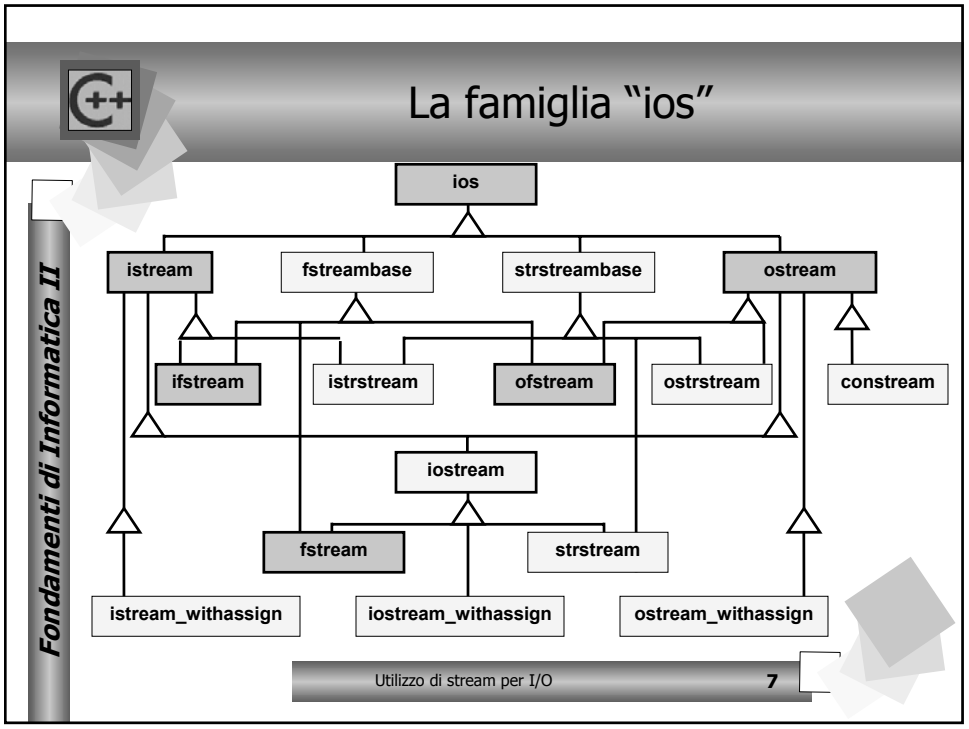
Fondamenti di Informatica II

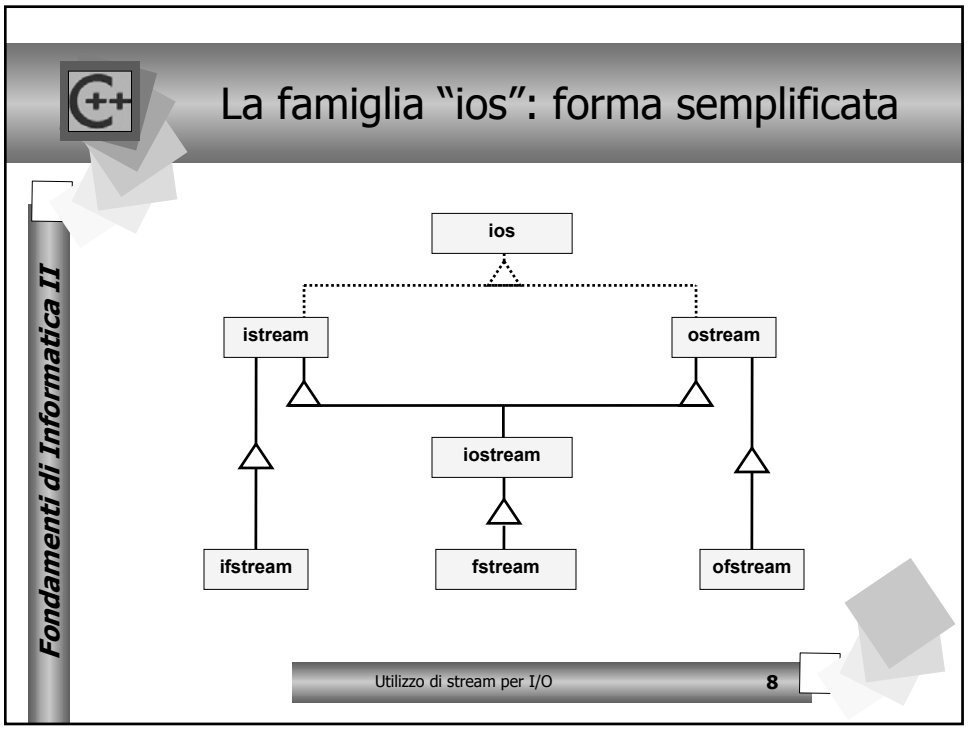
- Uno *stream di testo* è costituito da una sequenza di caratteri ed è organizzato in linee;
  ciascuna linea è terminata dal carattere '\n' (newline)
  la terminazione del file è indicata dal carattere speciale EOF
- Uno *stream binario* è costituito da una sequenza di byte che rappresentano informazioni in formato interno;
  le procedure di gestione trattano allo stesso modo tutti i caratteri in esso presenti.

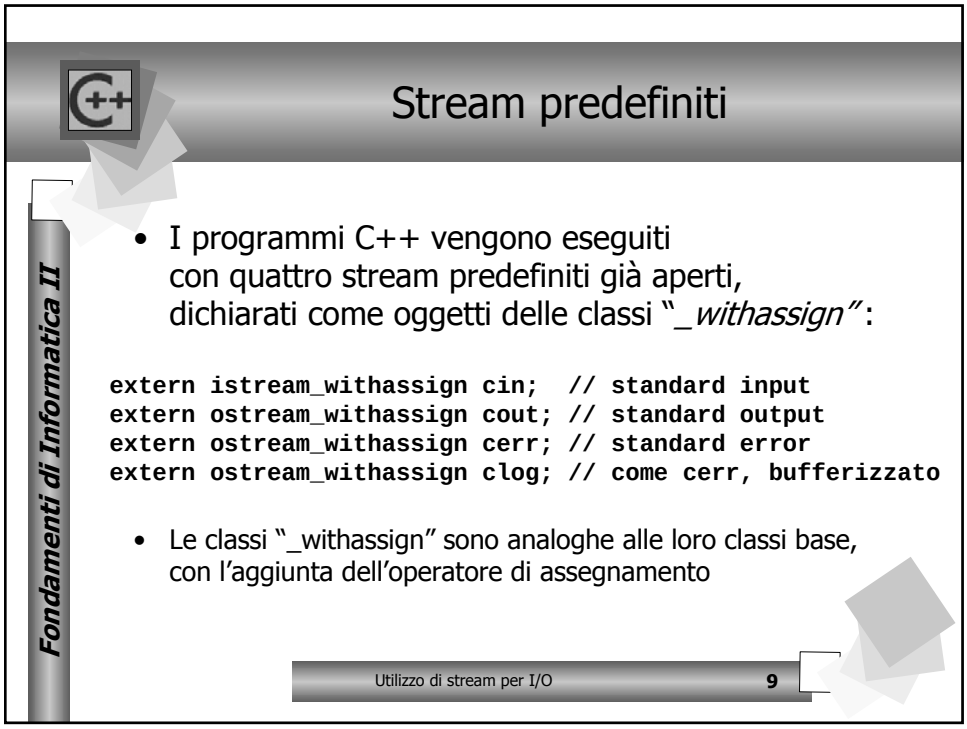
Utilizzo di stream per I/O 4

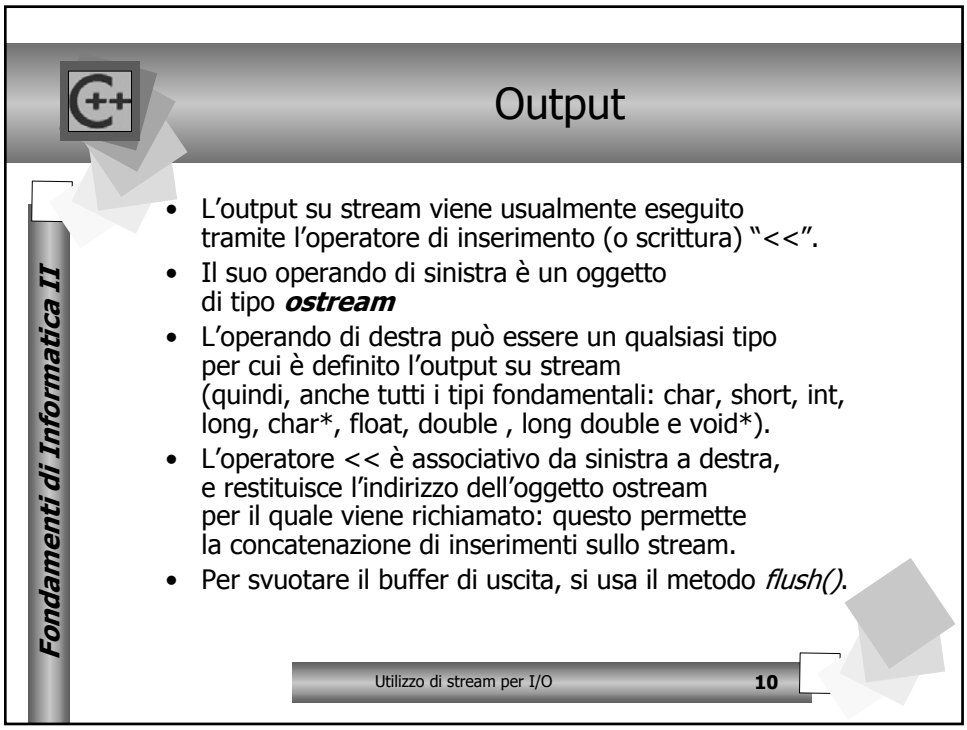

La libreria "iostream"
Fondamenti di Informatica II
• La libreria *iostream* ha due famiglie parallele di classi, con due classi radice distinte:
– ***streambuf***
– ***ios***
• La classe *streambuf* costituisce un'interfaccia verso i dispositivi fisici, e fornisce metodi per la gestione dei buffer e degli stream quando non è necessaria formattazione.
• La classe *ios* contiene un puntatore a *streambuf*, ed esegue operazioni di I/O formattato con gestione degli errori tramite *streambuf*.
Utilizzo di stream per I/O
5

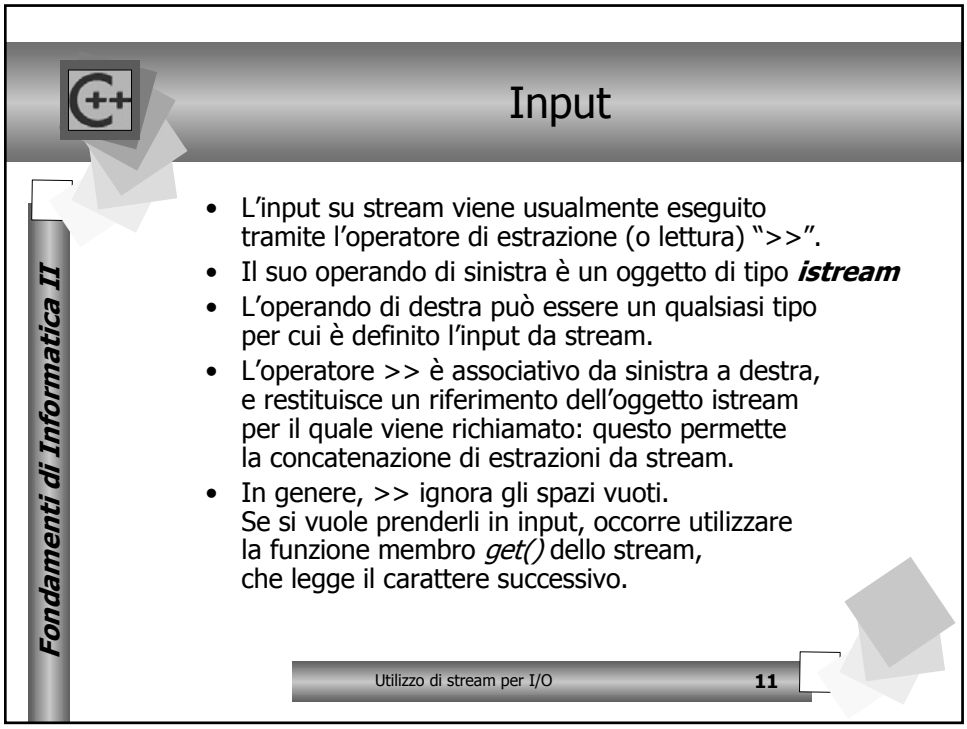

La famiglia "streambuf"
Fondamenti di Informatica II
streambuf
filebuf
strstreambuf
conbuf
Specializza streambuf nella gestione dei file
Specializza streambuf per la formattazione in memoria
Specializza streambuf per gestire l'output di console
Utilizzo di stream per I/O
6


La famiglia "ios"
Fondamenti di Informatica II
ios
istream
fstreambase
strstreambase
ostream
ifstream
istrstream
ofstream
ostrstream
constream
iostream
fstream
strstream
istream_withassign
iostream_withassign
ostream_withassign
Utilizzo di stream per I/O
7


La famiglia "ios": forma semplificata
Fondamenti di Informatica II
ios
istream
ostream
iostream
ifstream
fstream
ofstream
Utilizzo di stream per I/O
8


Stream predefiniti
Fondamenti di Informatica II
• I programmi C++ vengono eseguiti con quattro stream predefiniti già aperti, dichiarati come oggetti delle classi "_withassign":
extern istream_withassign cin; // standard input
extern ostream_withassign cout; // standard output
extern ostream_withassign cerr; // standard error
extern ostream_withassign clog; // come cerr, bufferizzato
• Le classi "_withassign" sono analoghe alle loro classi base, con l'aggiunta dell'operatore di assegnamento
Utilizzo di stream per I/O
9

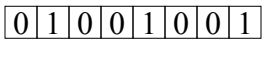

Output
Fondamenti di Informatica II
• L'output su stream viene usualmente eseguito tramite l'operatore di inserimento (o scrittura) "<<".
• Il suo operando di sinistra è un oggetto di tipo ostream
• L'operando di destra può essere un qualsiasi tipo per cui è definito l'output su stream (quindi, anche tutti i tipi fondamentali: char, short, int, long, char*, float, double , long double e void*).
• L'operatore << è associativo da sinistra a destra, e restituisce l'indirizzo dell'oggetto ostream per il quale viene richiamato: questo permette la concatenazione di inserimenti sullo stream.
• Per svuotare il buffer di uscita, si usa il metodo flush().
Utilizzo di stream per I/O
10


Input
Fondamenti di Informatica II
• L'input su stream viene usualmente eseguito tramite l'operatore di estrazione (o lettura) ">>".
• Il suo operando di sinistra è un oggetto di tipo ***istream***
• L'operando di destra può essere un qualsiasi tipo per cui è definito l'input da stream.
• L'operatore >> è associativo da sinistra, e restituisce un riferimento dell'oggetto istream per il quale viene richiamato: questo permette la concatenazione di estrazioni da stream.
• In genere, >> ignora gli spazi vuoti. Se si vuole prenderli in input, occorre utilizzare la funzione membro *get()* dello stream, che legge il carattere successivo.
Utilizzo di stream per I/O
11

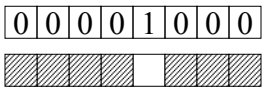

Condizioni e flag degli stream
Fondamenti di Informatica II
Lo *stato di funzionamento* di uno stream è codificato su tre flag, e dunque è rappresentato da un intero da 0 a 7. I flag sono conosciuti come:
• eofbit: indica che si è tentato di leggere la marca di fine stream (errore recuperabile)
• failbit: se 1, la precedente operazione sullo stream è fallita
• badbit: indica se l'errore è recuperabile (0) o no (1)
- Il metodo *rdstate()* restituisce lo stato dello stream; si può controllare se uno specifico flag è settato utilizzando i metodi *good(), eof(), fail(), bad()*
- Il metodo *clear()* setta i flag di stato secondo il valore (int) passatogli come argomento.
Utilizzo di stream per I/O
12

Manipolazione di flag

- Registro dei flag F



- Maschera di selezione di un flag "X"
  (configurazione di F con il solo flag "X" settato, corrispondente al valore che chiamiamo *VALFLAG_X)*



- Per controllare se il flag X è settato (con AND bit a bit):
  ((F & VALFLAG_X) == VALFLAG_X)
- Per settare i flag "X" e "Y" su F (con OR bit a bit):
  F = F | VALFLAG_X | VALFLAG_Y

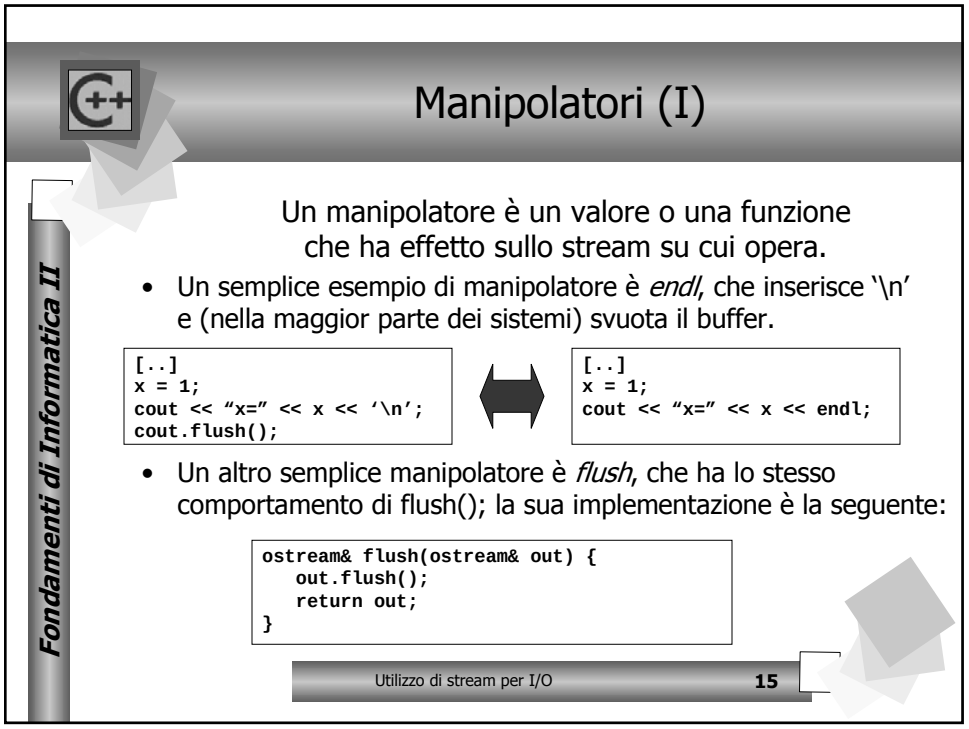
Stream ed eccezioni

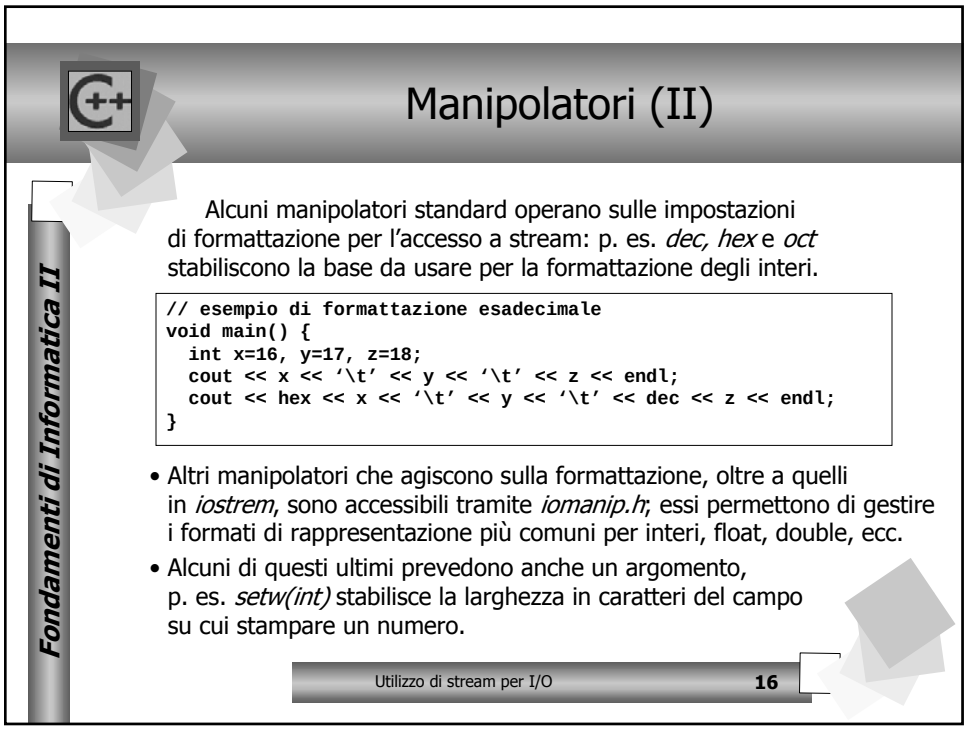
Per usare eccezioni nella gestione di stream, ci si riferisce a una versione generalizzata di iostream, accessibile includendo <iostream> (invece di <iostream.h>)

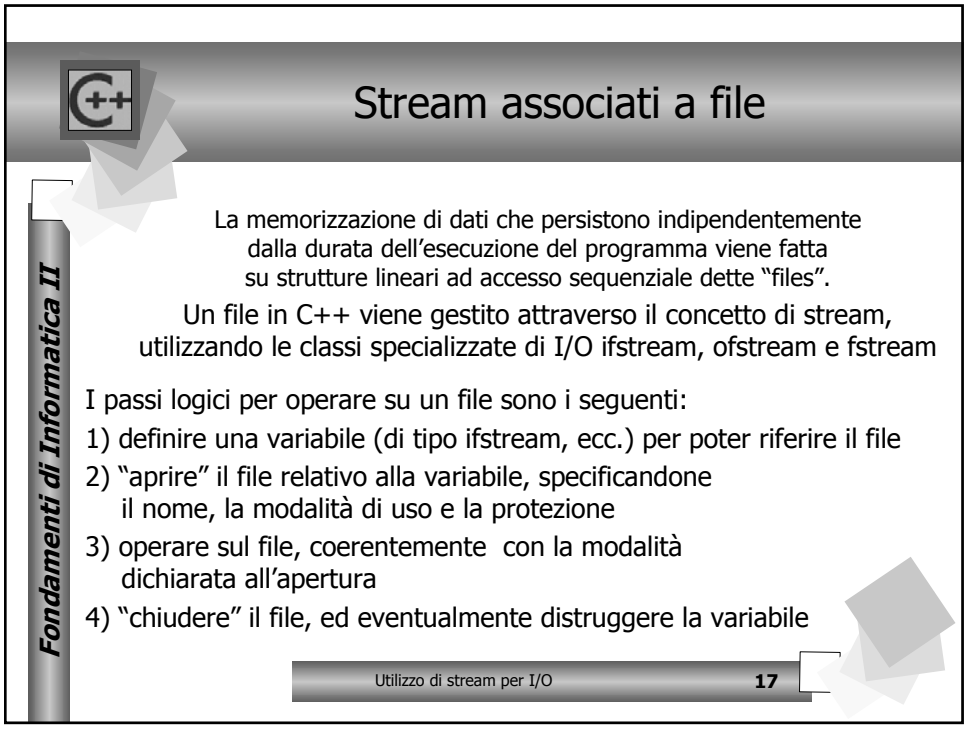
In questa libreria, gli operatori di I/O lanciano un'eccezione del tipo ***ios_base::failure*** se, dopo un'operazione, alcuni flag di stato (selezionabili) risultano settati.

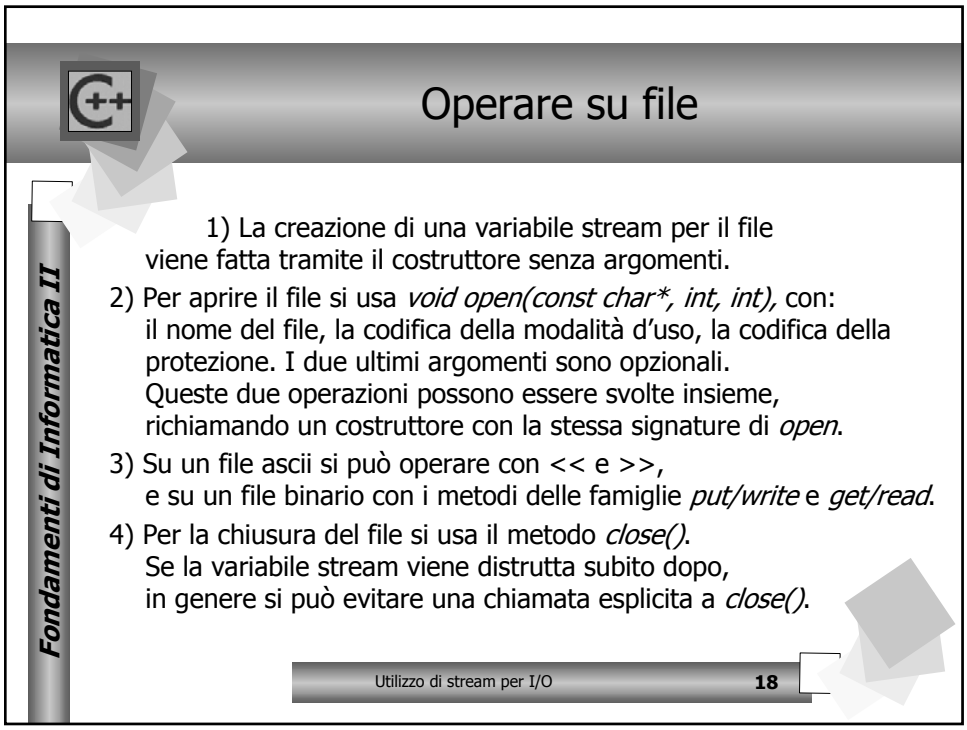
I flag da considerare si possono specificare con ***exceptions(int exp)***

```
#include<iostream>
using namespace std;
void main() {
  int i; int vett[5];
  cin.exceptions(ios::failbit | ios::eofbit); // ecc: con failbit o eofbit
  try {  for(i=0; i<10; i++) cin >> vett[i];  }
  catch(ios_base::failure) {
    int s = cin.exceptions();
    if (s & ios::eofbit) cout << "Fine di cin" << endl;
    if (s & ios::failbit) cout << "Errore di lettura" << endl;
  }
}
```


Manipolatori (I)
Fondamenti di Informatica II
Un manipolatore è un valore o una funzione che ha effetto sullo stream su cui opera.
• Un semplice esempio di manipolatore è endl, che inserisce '\n' e (nella maggior parte dei sistemi) svuota il buffer.
[..]
x = 1;
cout << "x=" << x << '\n';
cout.flush();
[..]
x = 1;
cout << "x=" << x << endl;
• Un altro semplice manipolatore è flush, che ha lo stesso comportamento di flush(); la sua implementazione è la seguente:
ostream& flush(ostream& out) {
out.flush();
return out;
}
Utilizzo di stream per I/O
15


Manipolatori (II)
Fondamenti di Informatica II
Alcuni manipolatori standard operano sulle impostazioni di formattazione per l'accesso a stream: p. es. dec, hex e oct stabiliscono la base da usare per la formattazione degli interi.
// esempio di formattazione esadecimale
void main() {
int x=16, y=17, z=18;
cout << x << '\t' << y << '\t' << z << endl;
cout << hex << x << '\t' << y << '\t' << dec << z << endl;
}
• Altri manipolatori che agiscono sulla formattazione, oltre a quelli in iostrem, sono accessibili tramite iomanip.h; essi permettono di gestire i formati di rappresentazione più comuni per interi, float, double, ecc.
• Alcuni di questi ultimi prevedono anche un argomento, p. es. setw(int) stabilisce la larghezza in caratteri del campo su cui stampare un numero.
Utilizzo di stream per I/O
16


Stream associati a file
Fondamenti di Informatica II
La memorizzazione di dati che persistono indipendentemente dalla durata dell'esecuzione del programma viene fatta su strutture lineari ad accesso sequenziale dette "files".
Un file in C++ viene gestito attraverso il concetto di stream, utilizzando le classi specializzate di I/O ifstream, ofstream e fstream
I passi logici per operare su un file sono i seguenti:
1) definire una variabile (di tipo ifstream, ecc.) per poter riferire il file
2) "aprire" il file relativo alla variabile, specificandone il nome, la modalità di uso e la protezione
3) operare sul file, coerentemente con la modalità dichiarata all'apertura
4) "chiudere" il file, ed eventualmente distruggere la variabile
Utilizzo di stream per I/O
17


Operare su file
Fondamenti di Informatica II
1) La creazione di una variabile stream per il file viene fatta tramite il costruttore senza argomenti.
2) Per aprire il file si usa void open(const char*, int, int), con: il nome del file, la codifica della modalità d'uso, la codifica della protezione. I due ultimi argomenti sono opzionali. Queste due operazioni possono essere svolte insieme, richiamando un costruttore con la stessa signature di open.
3) Su un file ascii si può operare con << e >>, e su un file binario con i metodi delle famiglie put/write e get/read.
4) Per la chiusura del file si usa il metodo close(). Se la variabile stream viene distrutta subito dopo, in genere si può evitare una chiamata esplicita a close().
Utilizzo di stream per I/O
18

Operare su file: esempio

Fondamenti di Informatica II

```
void doppiaLinea(ifstream& orig, ofstream& dest) {
  char c;
  while(orig.get(c)) {
    dest.put(c);
    if(c=='\n') dest.put(c);
  }
}

void main() {

  ifstream fOrig("mioTesto.txt");
  ofstream fDest("mioTesto2.txt");


  doppiaLinea(fOrig, fDest);
}
```

ifsream(const char*,
int = ios::in,
int prot = filebuf::openprot)

ofsream(const char*,
int = ios::out,
int prot = filebuf::openprot)

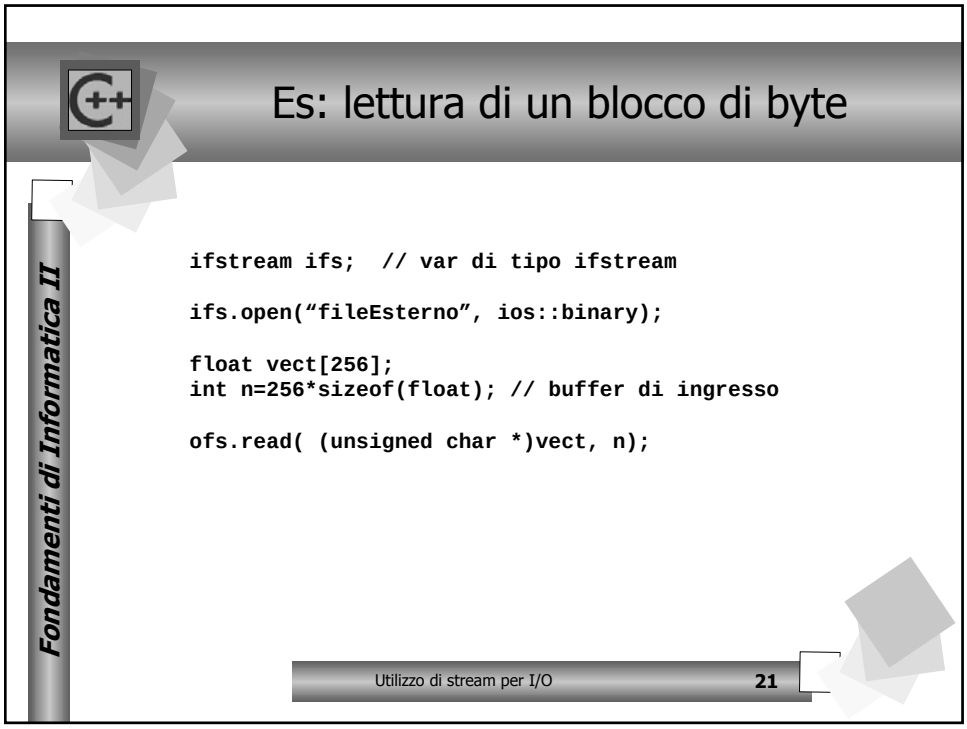
Utilizzo di stream per I/O 19

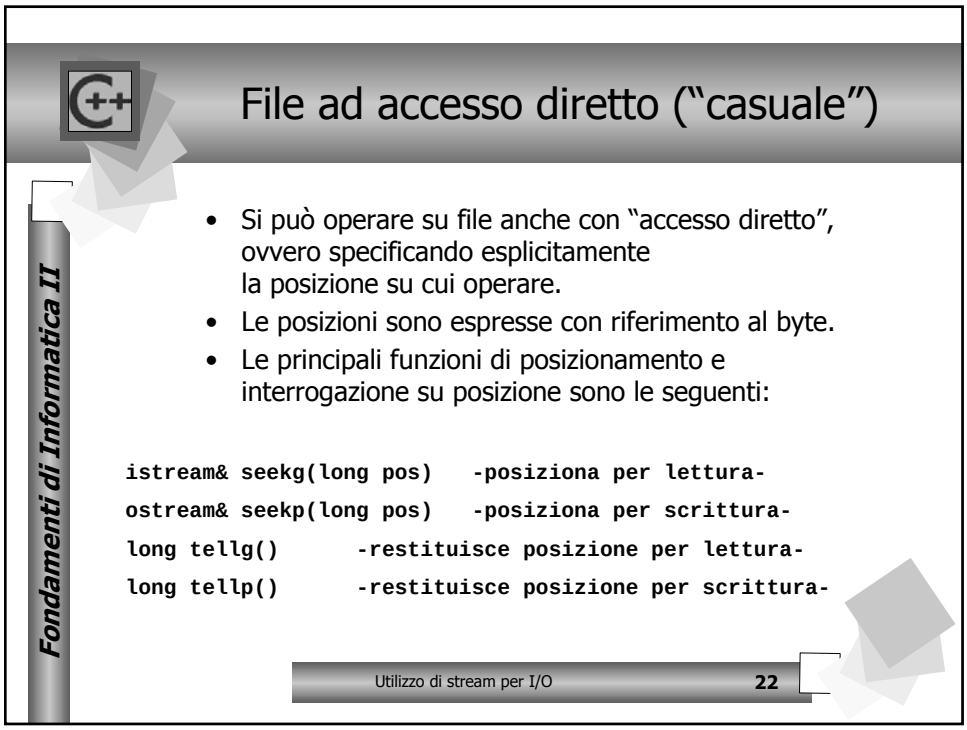
Es: scrittura di un blocco di byte

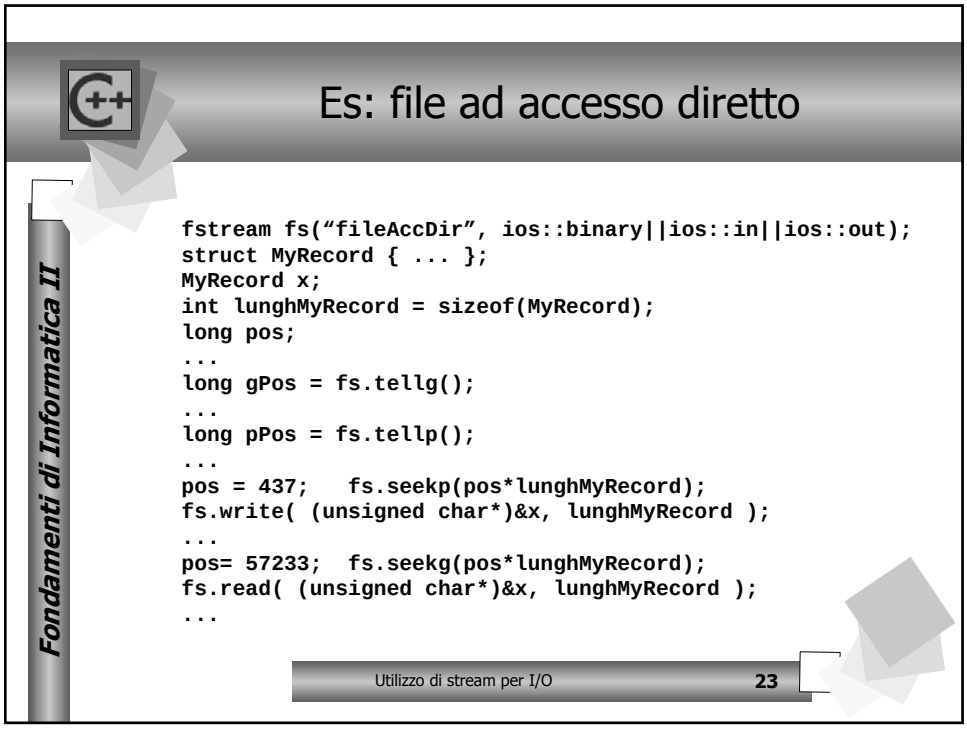
Fondamenti di Informatica II

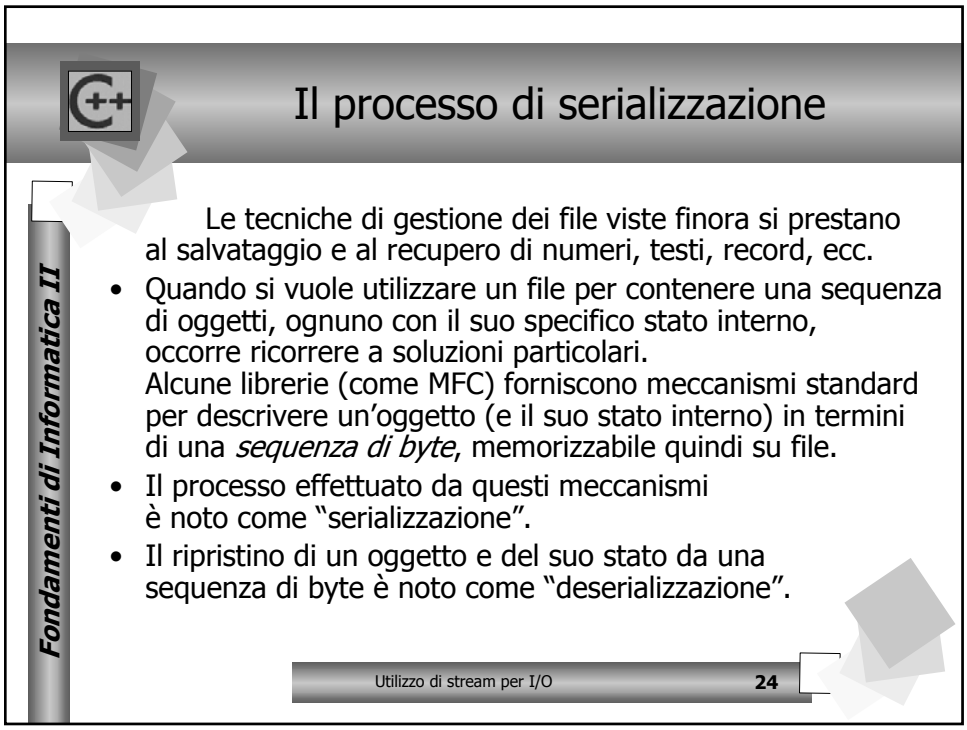
```
ofstream ofs;  // var di tipo ofstream

ofs.open("fileEsterno", ios::binary);

float vect[256];
int n=256*sizeof(float);

ofs.write( (unsigned char *)vect, n);
```

Utilizzo di stream per I/O 20


Es: lettura di un blocco di byte
Fondamenti di Informatica II
ifstream ifs; // var di tipo ifstream
ifs.open("fileEsterno", ios::binary);
float vect[256];
int n=256*sizeof(float); // buffer di ingresso
ofs.read((unsigned char *)vect, n);
Utilizzo di stream per I/O
21


File ad accesso diretto ("casuale")
Fondamenti di Informatica II
• Si può operare su file anche con "accesso diretto", ovvero specificando esplicitamente la posizione su cui operare.
• Le posizioni sono espresse con riferimento al byte.
• Le principali funzioni di posizionamento e interrogazione su posizione sono le seguenti:
istream& seekg(long pos) -posiziona per lettura-
ostream& seekp(long pos) -posiziona per scrittura-
long tellg() -restituisce posizione per lettura-
long tellp() -restituisce posizione per scrittura-
Utilizzo di stream per I/O
22


Es: file ad accesso diretto
Fondamenti di Informatica II
fstream fs("fileAccDir", ios::binary||ios::in||ios::out);
struct MyRecord { ... };
MyRecord x;
int lunghMyRecord = sizeof(MyRecord);
long pos;
...
long gPos = fs.tellg();
...
long pPos = fs.tellp();
...
pos = 437; fs.seekp(pos*lunghMyRecord);
fs.write((unsigned char*)&x, lunghMyRecord);
...
pos= 57233; fs.seekg(pos*lunghMyRecord);
fs.read((unsigned char*)&x, lunghMyRecord);
...
Utilizzo di stream per I/O
23


Il processo di serializzazione
Fondamenti di Informatica II
Le tecniche di gestione dei file viste finora si prestano al salvataggio e al recupero di numeri, testi, record, ecc.
• Quando si vuole utilizzare un file per contenere una sequenza di oggetti, ognuno con il suo specifico stato interno, occorre ricorrere a soluzioni particolari.
Alcune librerie (come MFC) forniscono meccanismi standard per descrivere un'oggetto (e il suo stato interno) in termini di una sequenza di byte, memorizzabile quindi su file.
• Il processo effettuato da questi meccanismi è noto come "serializzazione".
• Il ripristino di un oggetto e del suo stato da una sequenza di byte è noto come "deserializzazione".
Utilizzo di stream per I/O
24