

UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

## Librerie e linking

Stefano Ghidoni



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE



# Agenda

- File header
  - [Preprocessore](#)
- Genesi di una libreria
- Il ruolo del linker
- Linking statico e dinamico



## File header

- Molte funzioni che usiamo non sono scritte da noi
  - Es: sqrt(), operatore << su cout
- Serve uno strumento per importare funzioni scritte da altri ( librerie)
- In C++, questo meccanismo si attiva mediante i **file header**



## File header

- Un file header è un insieme di dichiarazioni (alcune delle quali definizioni) di entità
  - Tali entità sono utilizzabili da chi include il file header
  - Es: math.h dichiara double sqrt(double x);
- Questo è il file header visto dalla **prospettiva dell'utente**

Molto spesso l'header contiene la dichiarazione di una funzione, ma non il suo codice, che si trova da un'altra parte.

L'header di solito contiene un pezzo di libreria, non tutta.



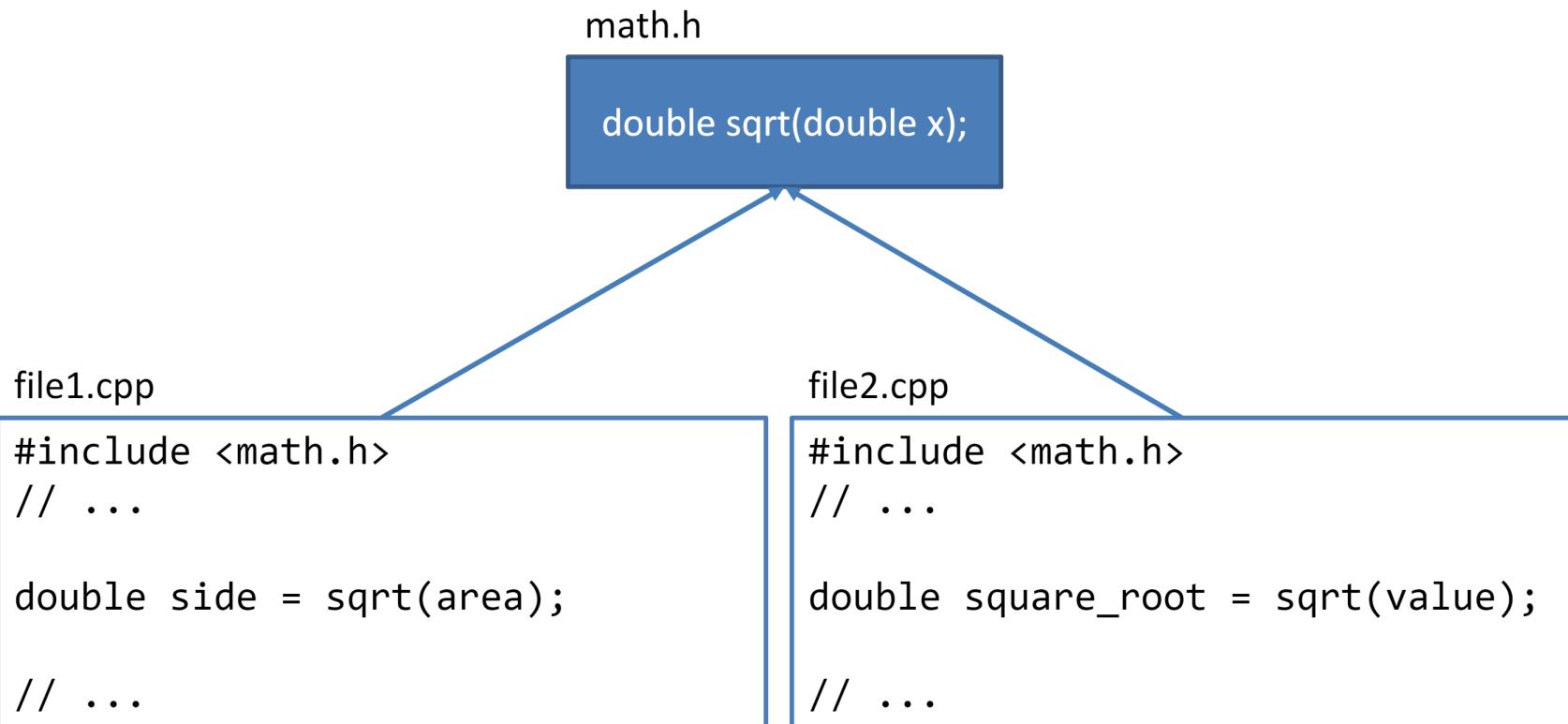
## File header

- Due meccanismi di inclusione:
  - Header di sistema, inclusi con <>
    - Es: `#include <iostream>`
  - Header definiti dall'utente, inclusi con ""
    - Es: `#include "my_header.h"`  
`#include "opencv2/core.h"`
- Differenza nel path di ricerca dei file header
  - Path di sistema (dipende dal sistema) vs path locale + sistema



# File header

- Esempio di uso dell'header lato utente



Funzioni, header,  
compilazione



# Funzioni in C++

```
int f(int i);
```

Dichiarazione 2

```
int main(void)
{
    int i = 0;

    i = f(i);

    return 0;
}
```

Chiamata 1

Il pattern comune in C++ è dichiarare, chiamare senza averla ancora definita, poi definirla

```
int f(int i)
{
    return i + 2;
}
```

Definizione 3



# Progetti software in file multipli

- Progetti grandi prevedono:
  - Molte dichiarazioni
  - Molte definizioni
- Necessità:
  - Raggruppare e spostare altrove le dichiarazioni
  - Raggruppare e spostare altrove le definizioni
- Questo porta a:
  - Avere più di un file sorgente
  - Creare header
- Questo è il file header visto dalla **prospettiva del progettista**



# Progetti software in file multipli

- Come possiamo distribuire un SW in molti file?

```
int f(int i);
```

Dichiarazione



Header file  
(my\_func.h)

```
int main(void)
{
    int i = 0;

    i = f(i);

    return 0;
}
```

Chiamata

Quindi passo da uno a tre file (il main rimane nel file dove si trovava)

```
int f(int i)
{
    return i + 2;
}
```

Definizione



File sorgente di libreria  
(my\_func.cpp)



# Header multipli

- Un file sorgente può includere più header

```
#include <iostream>
#include "date.h"
#include "book.h"
```

- Gli header possono includere altri header
  - Esempio: book.h include date.h
- Questo può generare problemi
  - Inclusioni cicliche: a.h include b.h, e viceversa



# Include guards

- Ogni header **deve** essere protetto dalle **include guards**

```
#ifndef RATIONAL_H
#define RATIONAL_H

// ...

#endif // RATIONAL_H
```

"se la costante non è stata definita  
nel processo di compilazione"

- Talvolta rese con:

```
#pragma once          // NON standard!!!
```

– Non è standard! Quindi non è corretto usarlo

→ se ne occupa il preprocessore



# Compilare progetti software

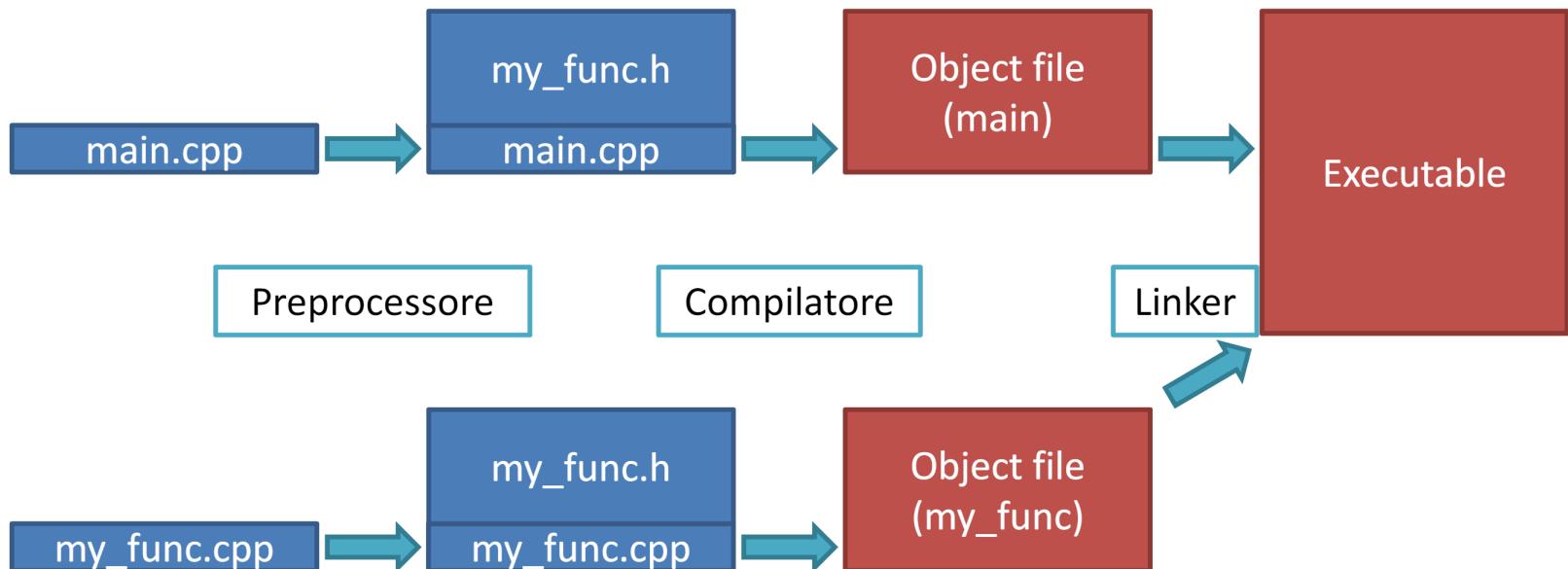
- Come compilare un progetto composto da molti file?
- Preprocessore, compilatore, linker!
  - Ora agiscono su più file
- Vediamo in dettaglio come...



# Da sorgente a eseguibile

Codice sorgente

Codice macchina



Qui non c'è l'uso.  
La libreria offre degli strumenti



## Librerie esterne

- Caso precedente: un progetto diviso in più file
  - La libreria era parte del progetto SW, è stata compilata
- Le librerie però spesso sono fornite da terzi
- Lo stesso meccanismo funziona quando la libreria e il codice che la usa appartengono a progetti diversi
  - Spesso non sono forniti i file .cpp, ma file pre-compilati



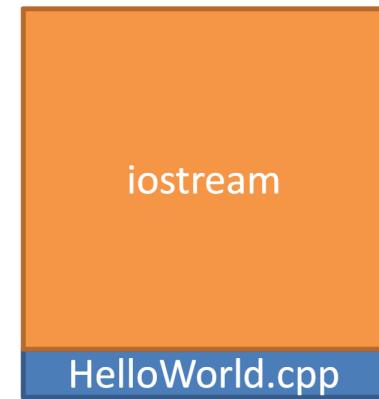
# Un caso semplice

- Un solo file sorgente (es., hello world)

```
#include <iostream>

int main(void)
{
    std::cout << "Hello world!\n";

    return 0;
}
```

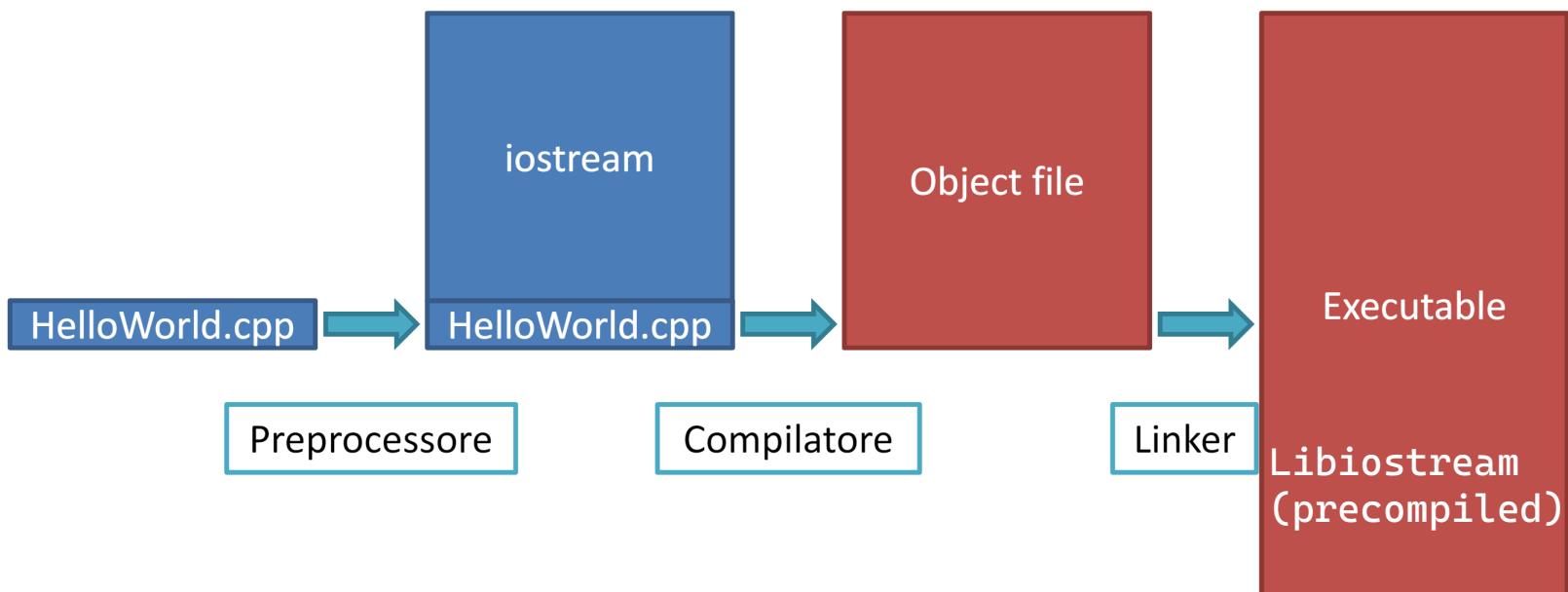




# Da sorgente a eseguibile

Codice sorgente

Codice macchina





# Librerie statiche

- Quello che abbiamo appena visto è un **linking statico (libreria statica)**
- Cosa succede se molti programmi linkano la stessa libreria?



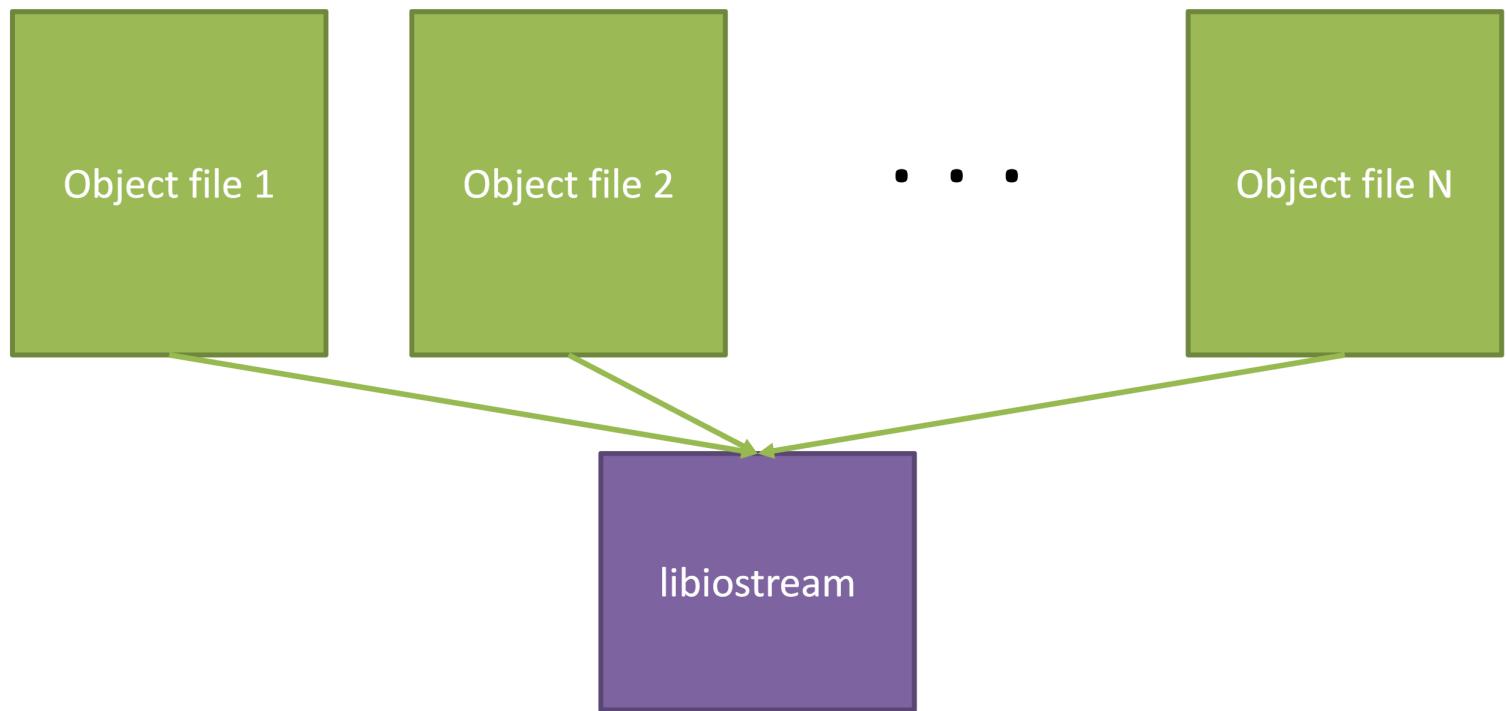
• • •





# Librerie dinamiche

- Nel caso di **linking dinamico (librerie dinamiche)** esiste una sola copia della libreria





# Librerie statiche e dinamiche

- Quali sono i vantaggi e gli svantaggi?



# Librerie statiche e dinamiche

- Quali sono i vantaggi e gli svantaggi?
  - **Librerie dinamiche**
    - Riduzione dello spazio disco occupato (una sola copia funziona per tutti gli eseguibili)
    - Possono essere ricompilate senza toccare gli eseguibili
    - Chiamate SO (Shared Object) sotto Linux e DLL (Dynamic Linking Library) sotto Windows
  - **Librerie statiche**
    - Generano eseguibili che non possono essere spezzati in seguito
    - Sono self-contained
      - Più adatte alla distribuzione di software monolitico
- lib iostream**



# Recap

- File header lato utente
- File header lato progettista
- Processo di compilazione
  - Più in dettaglio
  - Con più file
- Librerie statiche e dinamiche



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

## Librerie e linking

Stefano Ghidoni