Compilazione

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Linguaggio ad Alto Livello

- In ultima istanza il computer può eseguire solamente programmi nel linguaggio macchina

- Il linguaggio della macchina è un linguaggio a basso livello (dipende dall'architettura)

Linguaggio di Alto Livello

- Si può avere un linguaggio di più alto livello che ci permetta di
  - evitare di implementare più volte lo stesso programma per architetture diverse e
  - fornisca comandi più vicini al nostro modo di pensare?
- Assieme alla specifica del linguaggio, si fornisce uno strumento che traduca i nostri programmi nel linguaggio della macchina ospite: il traduttore

Traduzione: Interpretazione vs Compilazione

- Interprete: traduce una istruzione di alto livello e la esegue immediatamente
- Compilatore: traduce tutte le istruzioni assieme che vengono poi eseguite tutte assieme direttamente in linguaggio macchina (C, C++)
- Esistono soluzioni intermedie: compilazione in bytecode ed interpretazione (Java)

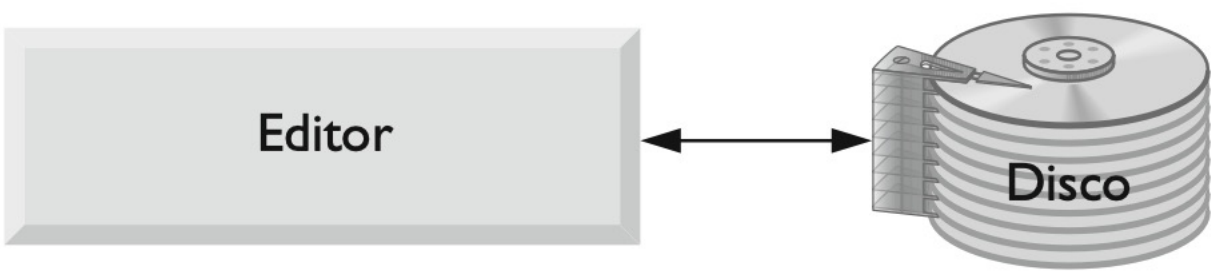
Traduzione: Interpretazione vs Compilazione

- Interprete:
  - più lenta l'esecuzione del programma.
  - necessita del traduttore per eseguire il programma
  - se si ha il traduttore ed il codice sorgente, può essere eseguito su ogni computer

- Compilatore:
  - più veloce l'esecuzione (di solito riesce anche a ottimizzare il codice)
  - non necessità del traduttore, ma ogni volta che cambio il programma devo ricompilarlo
  - il codice deve essere compilato per ogni diversa architettura

- Scelte del C: linguaggio compilato; insieme ristretto di comandi di base (ci si affida a librerie di funzioni), il compilatore è "facile" da scrivere, quindi portabilità

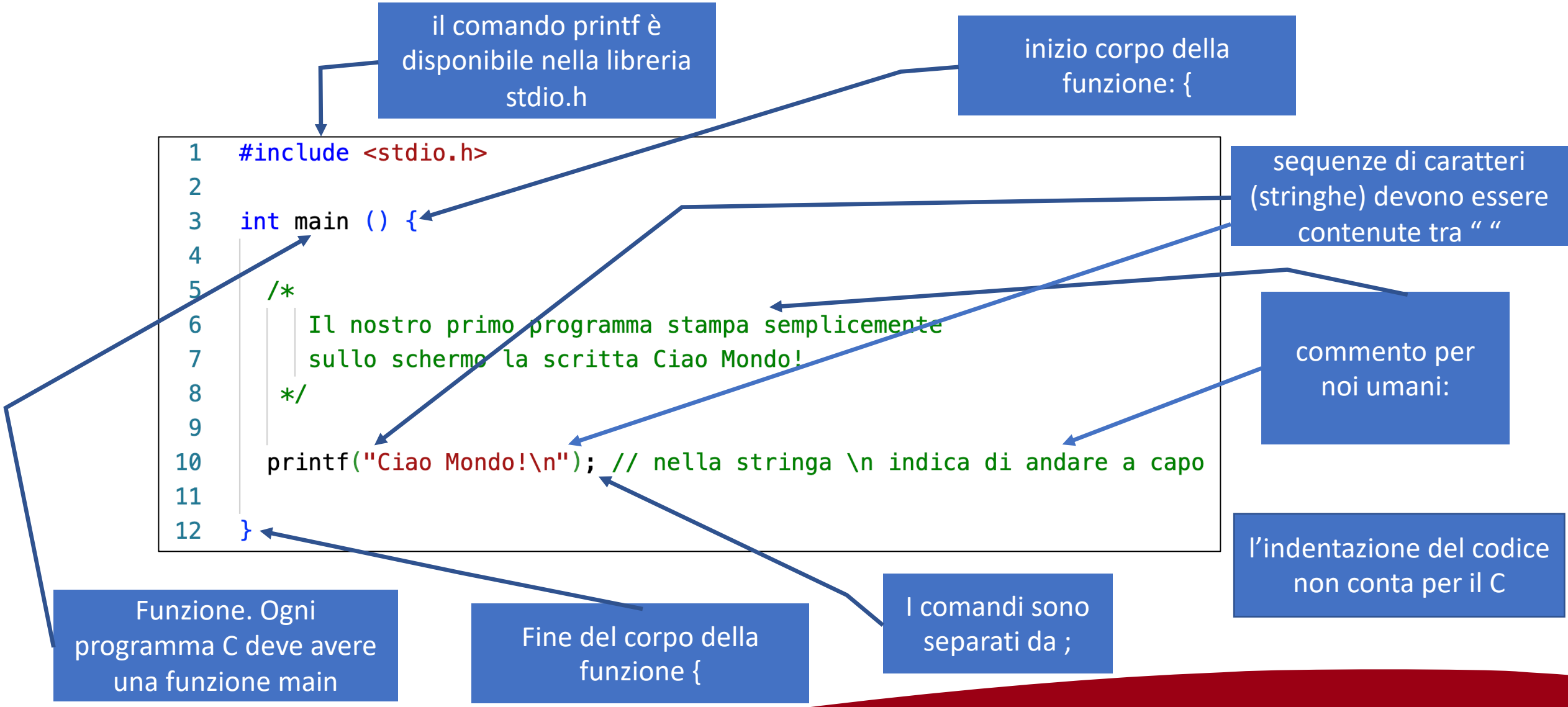
Traduzione



Fase1:
Il programmatore crea il programma con l'editor e lo memorizza su disco.

```c
#include <stdio.h>

int main () {

  /*
     Il nostro primo programma stampa semplicemente
     sullo schermo la scritta Ciao Mondo!
   */

  printf("Ciao Mondo!\n"); // nella stringa \n indica di andare a capo

}
```

Programma in C




```
1   #include <stdio.h>
2
3   int main () {
4
5     /*
6       Il nostro primo programma stampa semplicemente
7       sullo schermo la scritta Ciao Mondo!
8      */
9
10    printf("Ciao Mondo!\n"); // nella stringa \n indica di andare a capo
11
12  }
```

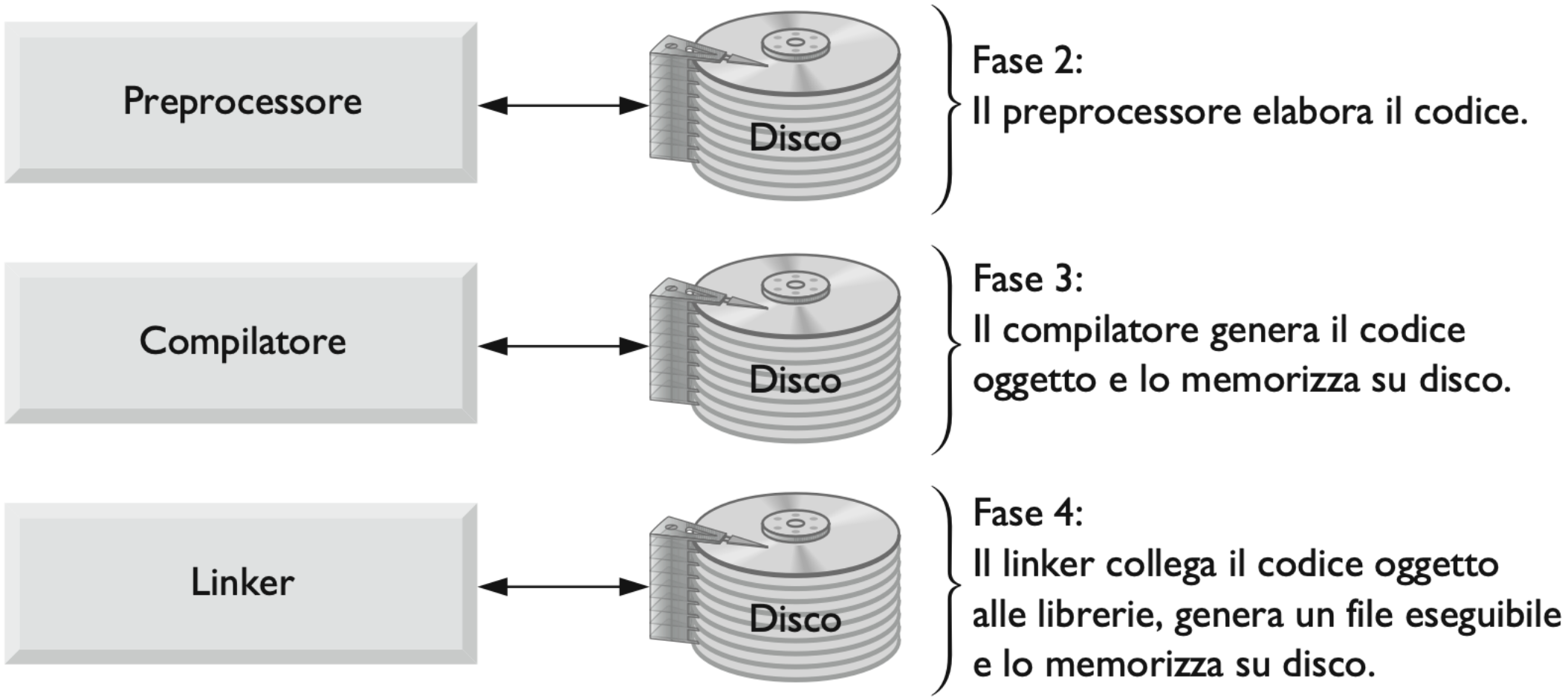
Commenti

- Commenti: descrizione ad alto livello di cosa fa (o come) un frammento di codice o un intero programma
- Obiettivo dei commenti: far comprendere il più velocemente possibile il codice a chi lo leggerà
- I commenti non devono essere banali: 3+2; //somma 3 e 2
- Se usate un algoritmo inusuale per risolvere un problema, indicatelo

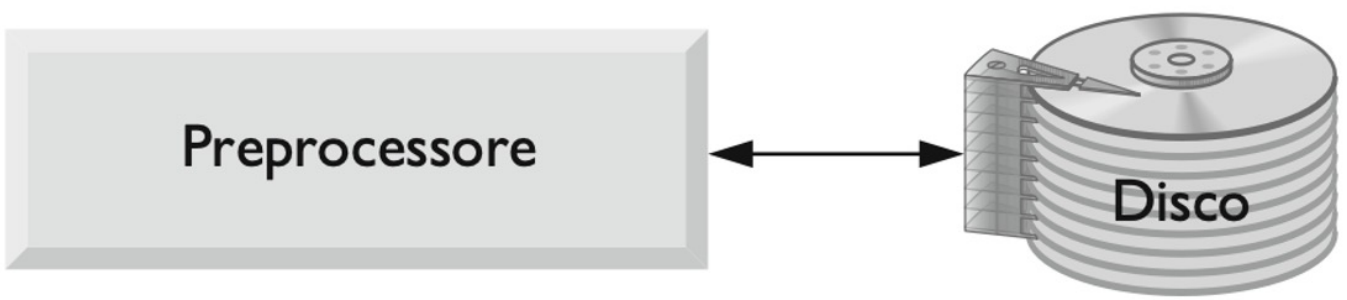
```c
#include <stdio.h>

int main () {

  /*
     Il nostro primo programma stampa semplicemente
     sullo schermo la scritta Ciao Mondo!
   */

  printf("Ciao Mondo!\n"); // nella stringa \n indica di andare a capo

}
```

- I programmi e le funzioni dovrebbero indicare come essere invocati.
- Nota: nel corso userò // per commenti che normalmente non metterei nel codice, ma che aggiungo per motivi didattici, /* */ per i commenti che metterei normalmente in un programma

Traduzione


Preprocessore
Disco
Fase 2:
Il preprocessore elabora il codice.
Compilatore
Disco
Fase 3:
Il compilatore genera il codice
oggetto e lo memorizza su disco.
Linker
Disco
Fase 4:
Il linker collega il codice oggetto
alle librerie, genera un file eseguibile
e lo memorizza su disco.

Traduzione



Fase 2:
Il preprocessore elabora il codice.

- Rimozione dei commenti
- Ogni linea che inizia per # indica una direttiva per il preprocessore
- #include <x>: il contenuto del file x viene ricopiato in questo punto del file
  - #include <x> permette di accedere ai comandi messi a disposizione dalla libreria x
  - Es. stdio.h permette di utilizzare il comando printf

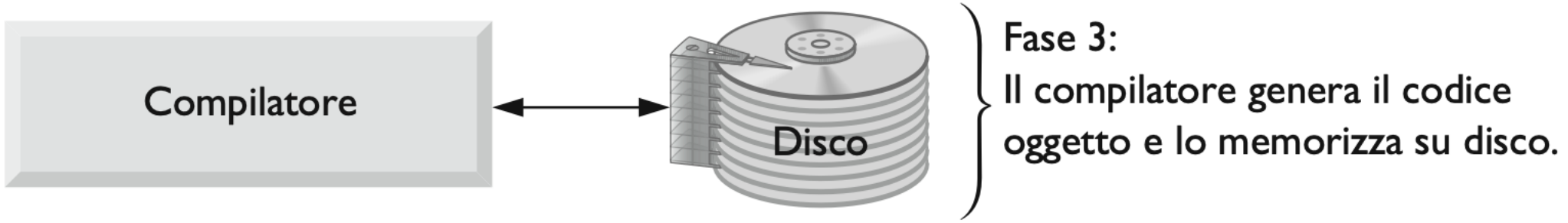
```c
#include <stdio.h>

int main () {

  printf("Ciao Mondo!\n");

}
```

Traduzione

- Espansione delle macro (le vedremo a breve)
  - #define X 3, sostituisce ogni occorrenza di X nel file con 3

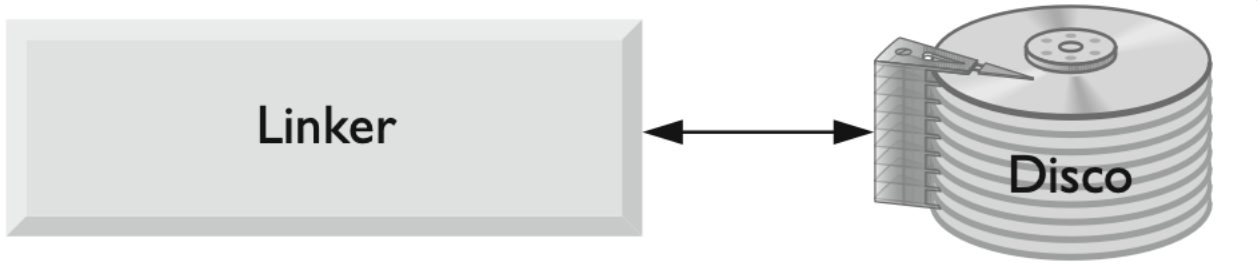
- Compilazione condizionale (utile se alcune librerie hanno nomi diversi in diversi sistemi operativi)

Traduzione

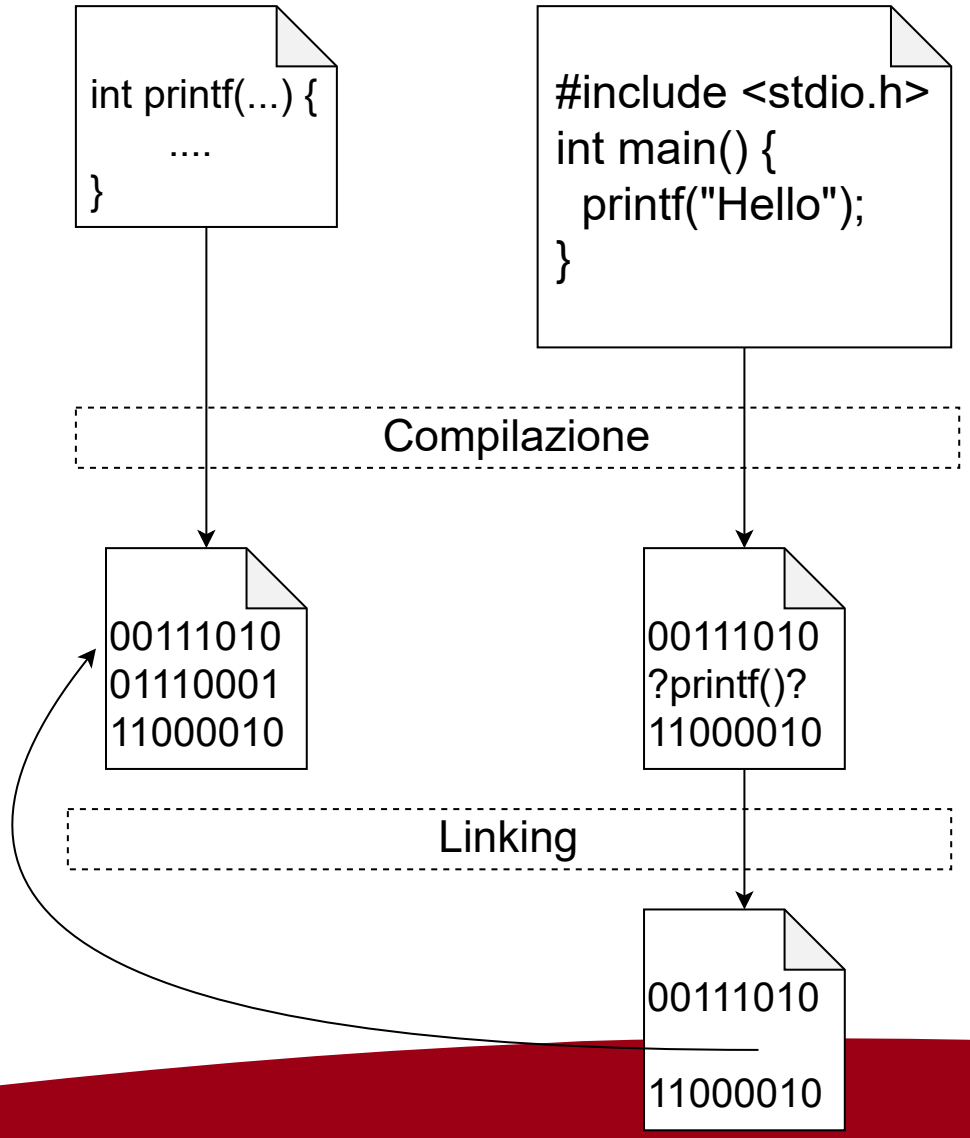


- Il compilatore analizza il file con il codice traducendolo in istruzioni del linguaggio a basso livello
- Le istruzioni devono seguire rigorosamente la sintassi definita dal linguaggio C.
- Un errore viene generato se il compilatore non riesce ad analizzare il nostro codice
- Se ci riesce, un file con le istruzioni nel linguaggio a basso livello viene generato

Traduzione



- Un programma è generalmente composto da molti file ed utilizza funzioni già scritte da altri (printf).
- Per evitare di duplicare il codice di tali funzioni, si caricano in memoria una volta e si collegano al nostro programma (linking)
- il linker viene invocato passandogli il file che usa printf ed il file dove printf è definita (entrambi compilati)



Compilazione Programma in C

- Ma in pratica come si compila un programma? Dal terminale digitare:

```
gcc -o ciao hello_world.c
```

- Il comando esegue tutte le fasi della compilazione
- -o indica il nome del file eseguibile
  - se si omette "–o ciao" viene creato il file a.out
- Il codice tradotto è a questo punto eseguibile: [su linux] ./ciao

```c
#include <stdio.h>

int main () {

  printf("Ciao Mondo!\n");

}
```

file: hello_world.c

Compilatori

- Esiste un unico compilatore? NO
- Il C è nato negli anni 70, ha avuto molto successo, per cui molti compilatori sono stati creati indipendentemente
- ANSI C: una serie di specifiche che standardizzano il comportamento del compilatore
  - C89, C90 – ISO C 1990
  - C17 – ISO C 2017
  - C2x – in lavorazione
- In alcuni casi particolari non è specificato il comportamento atteso, e quindi ogni compilatore può fare quello che vuole!
  - se abbiamo un dubbio sul comportamento di un elemento del linguaggio, non dobbiamo solamente provarlo sulla nostra macchina, ma controllare lo standard!
- Noi useremo quello di default usato dal compilatore del laboratorio

Opzioni del Compilatore

- Potete vedere le opzioni disponibili per il vostro compilatore digitando il comando

  man gcc su linux, man clang su MacOs

- -std=c89 indica quale standard ISO seguire
  - Qual è il valore di default sul vostro sistema?

- -c esegue tutte le fasi della compilazione fino al linking escluso (genera file con estensione .o)

Compilazione: Esempio

file: hello_world_stripped.c

```c
#include <stdio.h>

int main () {

  printf("Ciao Mondo!\n")

}
```

- gcc hello_world_stripped.c
- genera un errore alla riga 5 colonna 26
- Non necessariamente l'errore è esattamente dove indicato, la posizione indica dove il compilatore si è "arreso"

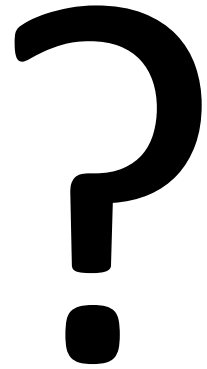
```
                                dasan$ gcc hello_world_stripped.c
hello_world_stripped.c:5:26: error: expected ';' after expression
  printf("Ciao Mondo!\n")
                         ^
                         ;
1 error generated.
```

Compilazione: Esempio

file: hello_world_stripped_w2.c

```c
#include <stdio.h>

int main () {

    printf("Ciao Mondo!\n");

}
```

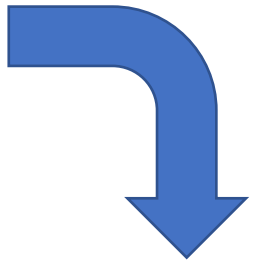
- gcc hello_world_stripped_w2.c



Compilazione: Esempio

file: hello_world_stripped_w2.c

```c
#include <stdio.h>

int main () {

  printf("Ciao Mondo!\n);

}
```



```
$ gcc hello_world_stripped_w2.c
hello_world_stripped_w2.c:5:10: warning: missing terminating '"' character [-Winvalid-pp-token]
  printf("Ciao Mondo!\n);
         ^
hello_world_stripped_w2.c:5:10: error: expected expression
hello_world_stripped_w2.c:8:1: error: expected '}'
^
hello_world_stripped_w2.c:3:13: note: to match this '{'
int main () {
           ^
1 warning and 2 errors generated.
```

- warning: non un errore ma qualcosa di insolito o ”rischioso”
- Tanti errori in cascata, si inizia dal primo (che di solito genera anche gli altri)
- Ci siamo dimenticati i doppi apici alla fine di -Ciao Mondo!\n- !