Il compilatore C

Astrazione

- Linguaggio macchina
  implica la conoscenza dei metodi usati per la rappresentazione dell informazioni nella CPU
- Linguaggio Assembly
  è il linguaggio delle istruzioni che possono essere eseguite su una CPU
  implica la conoscenza dettagliata delle caratteristiche della CPU (es. registri, set istruz.)

Astrazione

- Un linguaggio ad alto livello è astratto dai dettagli legati all'architettura della CPU
  E' possibile esprimere i propri algoritmi in modo simbolico
- Passaggio da Assembly a ling. macchina: banale
- Passaggio da alto livello ad Assembler: difficile

Esempio

- Linguaggio Macchina:
  0100 0000 0000 1000
  0100 0000 0000 1001
  0000 0000 0000 1000

- Linguaggio Assembly:
  movl 0, ebp
  cmpl 0, ebp
  jne L2

- Linguaggio di alto livello:

```
int main(void)
{
    int i = 0;
    if (i == 0)
       i = i + 1;
```

Le istruzioni corrispondono a quelle di sopra, ma sono scritte in modo simbolico

E' indipendente dalla machina

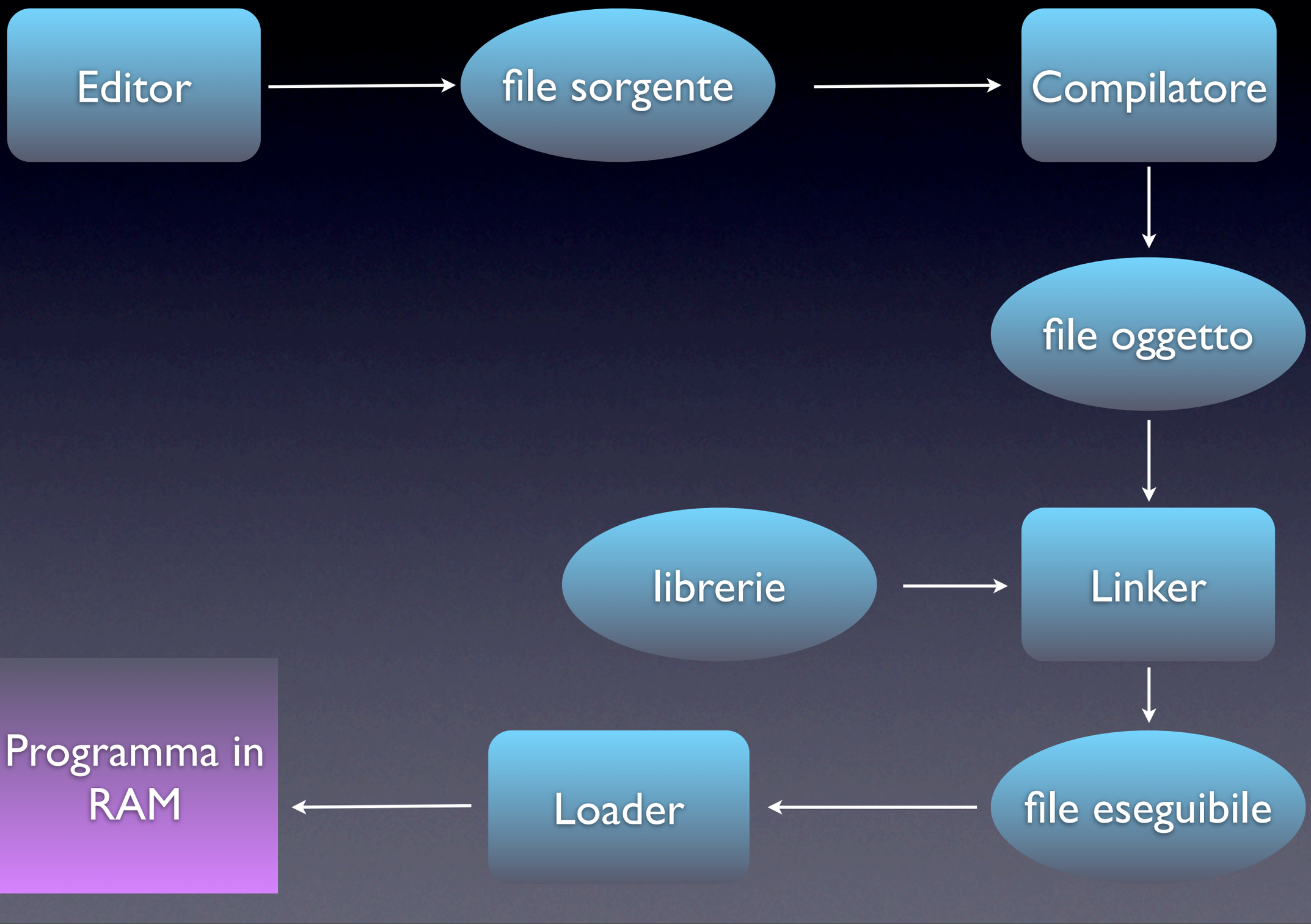
Esecuzione

- Per eseguire sulla macchina hardware un programma scritto in un linguaggio di alto livello è necessario tradurre il programma in sequenze di istruzioni di basso livello, direttamente eseguite dal processore, attraverso:
- interpretazione (ad es. JavaScript, Python)
- compilazione (ad es. C, C++, Delphi)

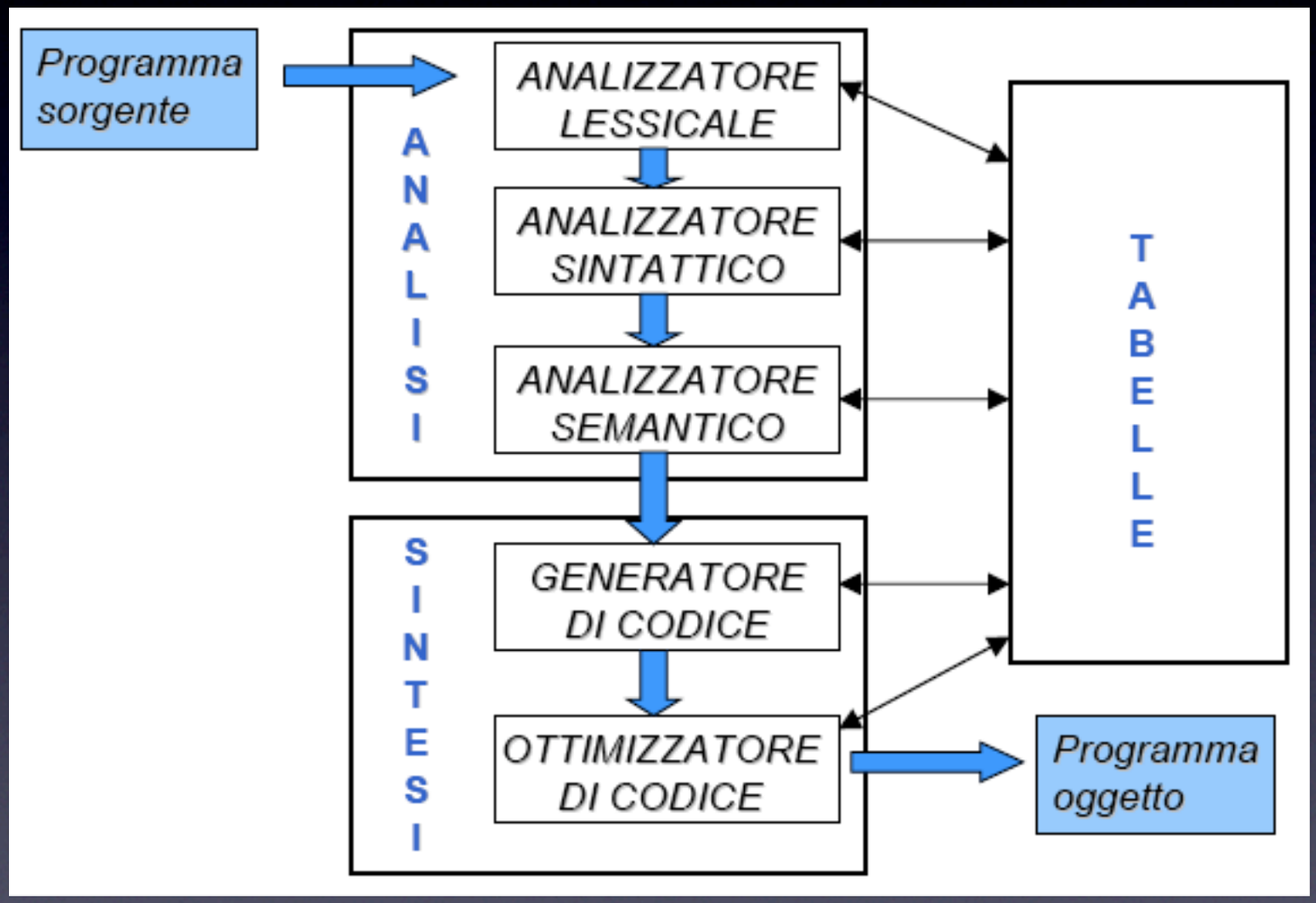
Compilatori e interpreti

- I compilatori traducono automaticamente un programma dal linguaggio L a quello macchina (per un determinato elaboratore)
- Gli interpreti sono programmi capaci di eseguire direttamente un programma in linguaggio L istruzione per istruzione
  - il codice può essere compilato in un linguaggio intermedio: es. Java
  - i programmi compilati sono in generale più efficienti di quelli interpretati

Compilazione: schema



Compilatore: modello



- Il programma sorgente viene analizzato
- Il programma oggetto viene sintetizzato

Analisi

- Il compilatore nel corso dell'analisi del programma sorgente verifica la correttezza sintattica e semanticadel programma:
  - ANALISI LESSICALE verifica che i simboli utilizzati siano legali cioè appartengano all'alfabeto
  - ANALISI SINTATTICA verifica che le regole grammaticali siano rispettate => albero sintattico
  - ANALISI SEMANTICA verifica i vincoli imposti dal contesto

Sintesi

- Generatore di codice: trasla la forma intermedia in linguaggio assembler o macchina
- Prima della generazione di codice:
  - allocazione della memoria
  - allocazione dei registri
- Eventuale passo ulteriore di ottimizzazione del codice

Il linguaggio C caratteristiche

- Imperativo: l'istruzione è un comando esplicito, che opera su una o più variabili oppure sullo stato interno della macchina, e le istruzioni vengono eseguite in un ordine prestabilito.
- Strutturato: si usano strutture di controllo come la sequenza, la selezione e il ciclo, evitando le istruzioni di salto incondizionato

Il linguaggio C caratteristiche

- Strutturato a blocchi: una o più istruzioni sono raggruppate insieme, come un paragrafo in un testo
- Procedurale: è possibile scrivere procedure che contengono sequenze di istruzioni. Le procedure possono essere richiamate, quando necessario, da altre procedure

Il linguaggio C caratteristiche

- Linguaggio di sistema: adatto per scrivere software di base, sistemi operativi, compilatori, driver
- Portabile, efficiente e sintetico: compilatori per quasi tutte le CPU, di medio livello, notazione sintetica
  (es.: `printf("%c\n", ( i == 0 ? '0' : '1') );`)

Il linguaggio C caratteristiche

- Basato su pochi concetti elementari
  - dati (tipi primitivi, tipi di dato)
  - espressioni
  - dichiarazioni / definizioni
  - istruzioni / blocchi
  - funzioni

- Il C è orientato alla programmazione strutturata (top-down):
  - per risolvere un problema, lo si scompone in sottoproblemi, ognuno dei quali può essere risolto in modo indipendente (raffinamento per passi successivi).
- Un programma è un insieme di funzioni. Una funzione può essere vista come un programma che, dati dei valori in ingresso, compie delle operazioni e restituisce un risultato.

Struttura di un programma C

- <programma> ::=
  {<unità-di-traduzione>}
  <main>
  {<unità-di-traduzione>}
- <main> ::=
  int main()
  {
  [<dichiarazioni-e-definizioni>]
  [<sequenza-istruzioni>]
  }

Struttura di un programma C

- <dichiarazioni-e-definizioni>
  introducono i nomi di costanti, variabili, tipi definiti dall'utente
- <sequenza-istruzioni>
  sequenza di frasi del linguaggio ognuna delle quali è un'istruzione
- main() è una particolare unità di traduzione (una *funzione*)

Struttura di un programma C

- **set di caratteri** ammessi in un programma dipende dall'implementazione; solitamente ASCII + estensioni

- **identificatori**
  sequenze di caratteri tali che
  <Identificatore> ::=
  <Lettera> { <Lettera> | <Cifra> }

Struttura di un programma C

- **Commenti**
  sequenze di caratteri racchiuse fra i delimitatori /* e */

- <Commento> ::= /* <frase> */
- <frase> ::= { <parola> }
- <parola> ::= { <carattere> }

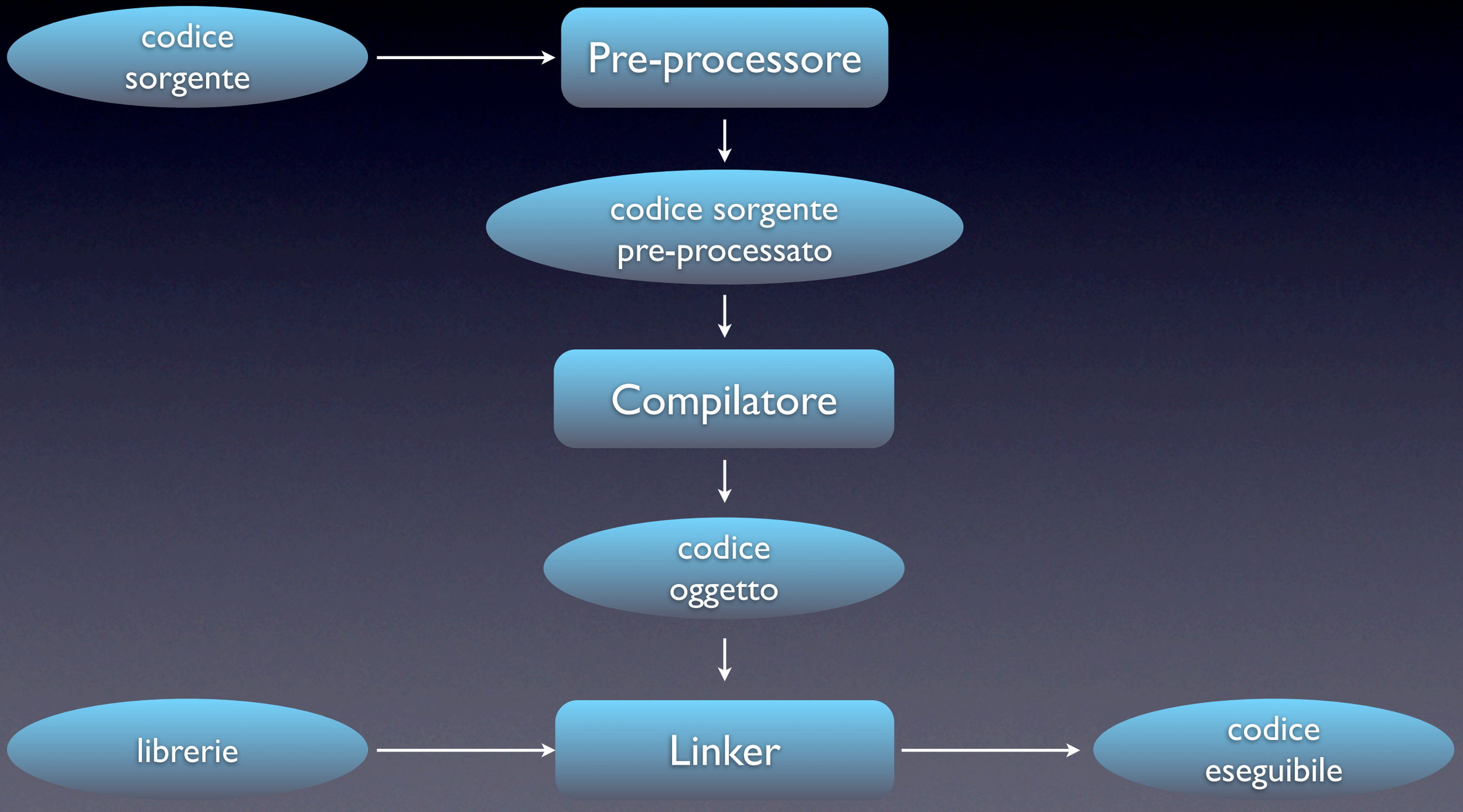
  i commenti non possono essere innestati

Esempio di programma

```
#include <stdio.h> /* istruzione
per pre-processore */

int main(void)
{
        printf("Hello world !\n");
        return 0;
}
```

La compilazione



La compilazione: passo I

- Il preprocessor (preprocessore) elimina i commenti e tratta le linee che iniziano con # (direttive al preprocessore).
- Esempio:
  #include "file.h"
- Il preprocessore sostituisce tale linea con il contenuto del file di nome file.h contenuto nella directory corrente. Se il file va ricercato altrove (in locazioni note al sistema) il nome del file va scritto fra le parentesi angolari < e >
- Esempio:
  #include <stdio.h> /* stdio.h e' un file presente nel sistema */
- Il preprocessore produce un file ASCII che verrà tradotto in codice oggetto dal compilatore. Esercizio: gcc -E prog.c

La compilazione: passo 2

- Il compiler (compilatore) traduce il programma sorgente in codice oggetto (file binario).
- Esercizio: gcc -c nomeprogramma.c
- Esercizio: gcc -S nomeprogramma.c

La compilazione: passo 3

- Il linker (o loader) carica e "mette insieme" i file oggetto prodotti al passo 2 e quelli già presenti nel sistema (ad esempio, il file oggetto che contiene printf()) e produce un unico file eseguibile.
- Es.: in gcc tutti e tre i passi possono essere eseguiti con un solo comando:
  gcc prog.c -o prog

Esempio

- gcc -c hello.c
  1) preprocessore, 2) compilatore
  produce hello.o
- ld -lc -lcrt1.10.5.o hello.o -o hello
  3) linker
  produce hello (eseguibile)
  Nota: -lcrt1.10.5.o serve sotto OSX, se il programma hello contiene *printf()*

Funzioni di libreria

- Le funzioni di libreria sono contenute in particolari file oggetto (quindi, file già compilati) posti in directory note al sistema.
- Sono inglobate nel programma dal linker nel passo 3.
- Per poter eseguire il passo 2 il compilatore deve però conoscere il prototipo delle funzioni usate, ossia:
  - numero e tipo dei parametri della funzione
    - tipo del valore restituito dalla funzione.
- Queste informazioni sono contenute negli header file (file di intestazione).

Debug

- Il compilatore può aggiungere informazioni addizionali nei file oggetto ed eseguibili per collegare le istruzioni macchina alle istruzioni del programma sorgente
- gcc -g

Ottimizzazione

- Il processo di compilazione non è univoco
- Si può richiedere al compilatore di ottimizzare il codice
  - il processo è più lento
  - dipende dalla bontà del compilatore
- Es.: gcc -O*X* con *X* = 1..3

Header file

- Sono file ASCII che contengono le definizioni che servono al compilatore per tradurre il codice contenuto in un file (prototipi di funzione, definizioni, . . .)
- Per convenzione hanno suffisso .h.
- Gli header file vanno inclusi nel sorgente mediante #include (quindi sono inclusi dal preprocessore prima della traduzione in codice oggetto). Vengono inclusi all'inizio del file in modo che le definizioni contenute in essi siano visibili in tutto il file.

File sorgenti

- Si può suddividere un programma in più file separati
- Il linker li ricollega
- Vantaggio: se si modifica il codice di un solo file si deve ricompilare solo quello (ma linkare tutto!)

Makefile

- Quando si suddivide un programma su più file può diventare macchinoso compilare il programma
- Si usa un programma (*make*) che esegue le istruzioni di compilazione e linking scritte con un apposito linguaggio, salvate in un file chiamato *Makefile*