Linguaggio compilato e interpretato

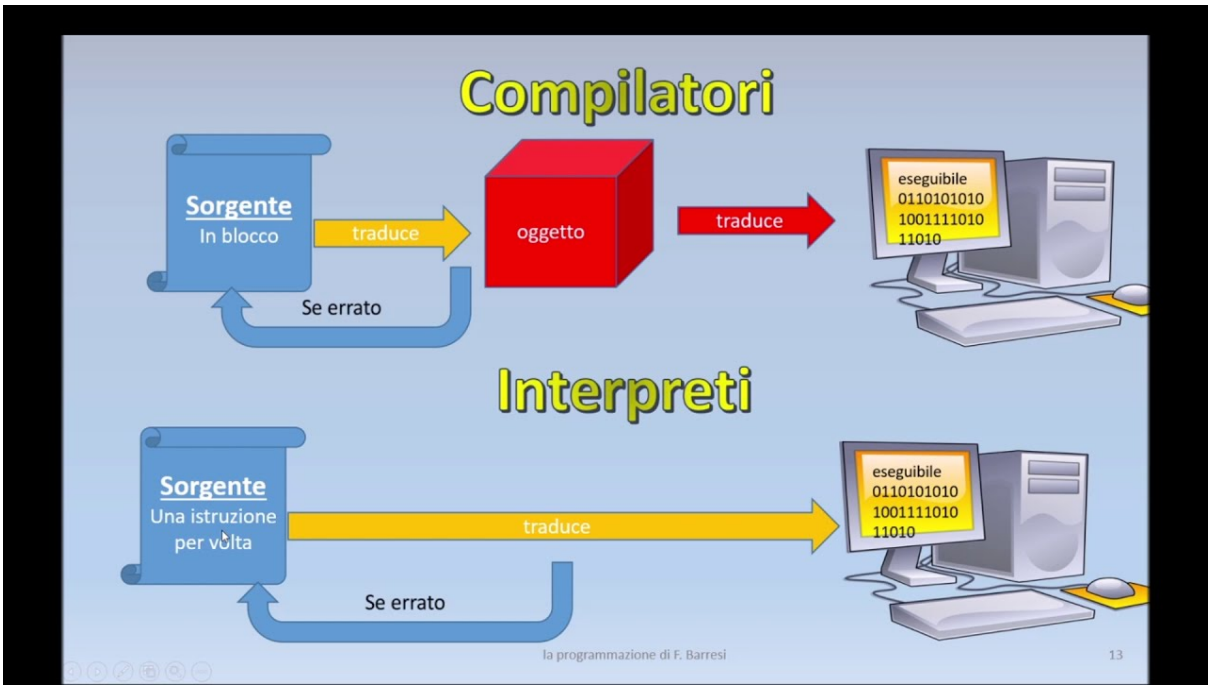
Introduzione

I programmi che noi scriviamo sono formati da istruzioni in linguaggio umano, incomprensibili per la macchina. Essa infatti comprende solo il linguaggio binario (0 o 1). Allora, avremo bisogno di qualcuno che traduca per noi le istruzioni che vogliamo eseguire. Per i linguaggi di alto livello, esistono due tipi di traduttori:

- **il compilatore**
- **l'interprete**

Differenze ed esempio

Il compilatore prima traduce tutto il blocco di istruzioni e poi decide se è possibile eseguire il programma o no (in base a se ci sono stati degli errori), mentre l'interprete esegue una istruzione per volta finchè o finisce le istruzioni oppure trova un errore.



Facendo un esempio, mettiamo che il computer sia un cuoco italiano e il codice sia una ricetta in giapponese. Partendo dal presupposto che il cuoco non sappia il giapponese, gli serve un modo per poter fare quella ricetta.

Nel caso del compilatore, è come se il cuoco avesse un traduttore di giapponese che gli riscrivesse tutta la ricetta in italiano. Così facendo, il cuoco poi legge direttamente la ricetta in italiano e la esegue in autonomia.
Nel caso dell'interprete invece, è come se il cuoco avesse un amico giapponese che si siede accanto a lui e legge passo per passo la ricetta dicendogli cosa significa.

Vantaggi e svantaggi

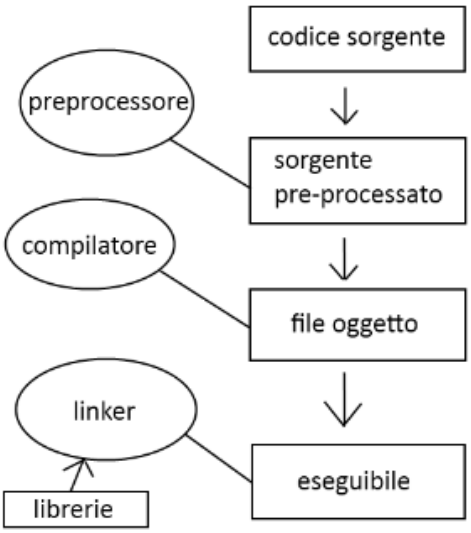
Quindi, da un lato prima si traduce tutto e poi si esegue, mentre dall'altro si traduce e poi si esegue dei pezzi più piccoli. Ma quindi, alla fine dei conti, **quale dei due metodi è il migliore?**
La risposta è che **dipende** da ciò che ti serve. Osserviamo vantaggi e svantaggi di entrambi.

	Linguaggio compilato	**Linguaggio interpretato**
Vantaggi	Più veloce da eseguire in quanto le istruzioni sono già scritte in linguaggio macchina.	Si adatta ad ogni piattaforma e il programma parte subito.
Svantaggi	Ci vuole più tempo per far partire il programma (durata maggiore della fase di compilazione) e il codice binario generato deve dipendere dalla piattaforma su cui ci troviamo.	Si possono verificare errori durante l'esecuzione del programma e il processore ha un carico di lavoro molto maggiore (prestazioni peggiori).

Come funziona la compilazione?

Abbiamo detto che il compilatore è come un traduttore a nostro servizio ma, nella pratica, **come fa a tradurre le istruzioni in linguaggio macchina?**
Per fare ciò, possiamo individuare 5 fasi principali:

1. **Preprocessing**
2. **Analisi lessicale (scanning)**
3. **Analisi sintattica (parsing)**
4. **Analisi semantica**
5. **Linking**



Vediamo le fasi nel dettaglio.

Prima fase: preprocessing

La prima fase è quella che viene detta di pre processione (preprocessing appunto).
In questa parte, non viene fatto altro che rimuovere i commenti e gli spazi presenti nel codice sorgente. Inoltre, vengono inclusi i file scritti ed espanse le definizioni che abbiamo dato (nel caso del C con il comando #define).

Seconda fase: scanning

Nella seconda fase (analisi lessicale) viene controllato che siano stati usati dei comandi e dei simboli comprensibili al compilatore di quel linguaggio. Per esempio, se in un file C il compilatore trova come tipo di una variabile un bool, non lo riconosce e da errore.

Terza fase: parsing

Nella terza fase (analisi sintattica) viene controllato il corretto ordine dei comandi e dei simboli, in modo tale che vengono formate delle "frasi" di senso compiuto per il compilatore. Ad esempio, se nel programma in C trovo una dichiarazione di variabile dove c'è scritto "int 6 = integerNumber;" questo non ha senso per il compilatore e da errore.

Quarta fase: analisi semantica

Nella quarta fase vengono determinati la compatibilità dei tipi, dei parametri, delle funzioni ed il significato da attribuire ad ogni "frase". In poche parole, il compilatore capisce per ogni "frase" quale sia l'istruzione o istruzioni da eseguire. Una volta finita questa fase, se non ci sono stati errori, viene creato il codice oggetto del sorgente, il quale avrà lo stesso nome del file originale ma con l'estensione .o (es. Helloworld.c diventerà HelloWorld.o).

Quinta fase: linking

Nell'ultima fase dobbiamo creare un file eseguibile (.exe) che possa leggere il nostro sistema operativo per poi eseguirlo. Per fare ciò, avremo bisogno del **linker**, ovvero un programma che collega i file oggetto creati precedentemente alle librerie che vengono utilizzate nel codice sorgente (un esempio può essere la libreria stdio.h in C). Ora che abbiamo creato il file eseguibile, il sistema operativo può far partire il programma ed eseguire tutte le istruzioni.