



**Università degli Studi  
di Cassino e del Lazio Meridionale**

**Corso di Fondamenti di  
Informatica**

*Linguaggi di Programmazione*

Anno Accademico 2012/2013

Francesco Tortorella

# Linguaggi di programmazione

- Un calcolatore basato sul modello di von Neumann permette l'esecuzione di un ***programma***, cioè di una sequenza di istruzioni descritte nel linguaggio interpretabile dal calcolatore che realizzano un particolare algoritmo, ma ...

# Linguaggio macchina

... ma quali sono le caratteristiche di tale linguaggio ?

- è codificato tramite sequenze di bit
- accede ai dati tramite gli indirizzi di memoria o i registri interni della CPU
- ogni istruzione può compiere solo azioni molto semplici
- non gestisce direttamente i tipi di dati di interesse
- è strettamente legato alla particolare macchina su cui è definito

Non a caso viene definito *linguaggio macchina*

# Scrivere un programma

- Se si volesse implementare un dato algoritmo attraverso la scrittura di un programma sarebbe quindi necessario:
  - conoscere dettagliatamente tutti i codici operativi e la loro codifica
  - decidere in quali registri (di memoria o interni alla CPU) vadano memorizzati i dati
  - determinare, per ogni singola operazione richiesta dall'algoritmo, la sequenza di istruzioni in linguaggio macchina che la realizzano
  - definire un'opportuna tecnica di codifica per ogni tipo di dati considerato
  - limitarsi a utilizzare solo i calcolatori per cui esista una tale competenza, tenendo comunque presente che il programma scritto per un certo calcolatore non è eseguibile su altre macchine

**Impresa difficile, ma non impossibile**

# Il gap semantico

## Esecutore umano

- linguaggio naturale
- gestione completa dei tipi
- istruzioni semanticamente ricche

## Calcolatore

- linguaggio rigido e complicato
- gestione dei tipi quasi nulla
- istruzioni estremamente semplici

# Linguaggio di programmazione

## Esecutore umano

- linguaggio naturale
- gestione completa dei tipi
- istruzioni semanticamente ricche

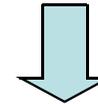


Orientato al problema

## Linguaggio di programmazione

- linguaggio formale, con costrutti precisi per la definizione dei dati e delle operazioni
- gestione completa dei tipi fondamentali; possibilità di definire tipi strutturati
- istruzioni che realizzano le principali azioni elaborative richieste

Orientato alla macchina



## Calcolatore

- linguaggio rigido e complicato
- gestione dei tipi quasi nulla
- istruzioni estremamente semplici

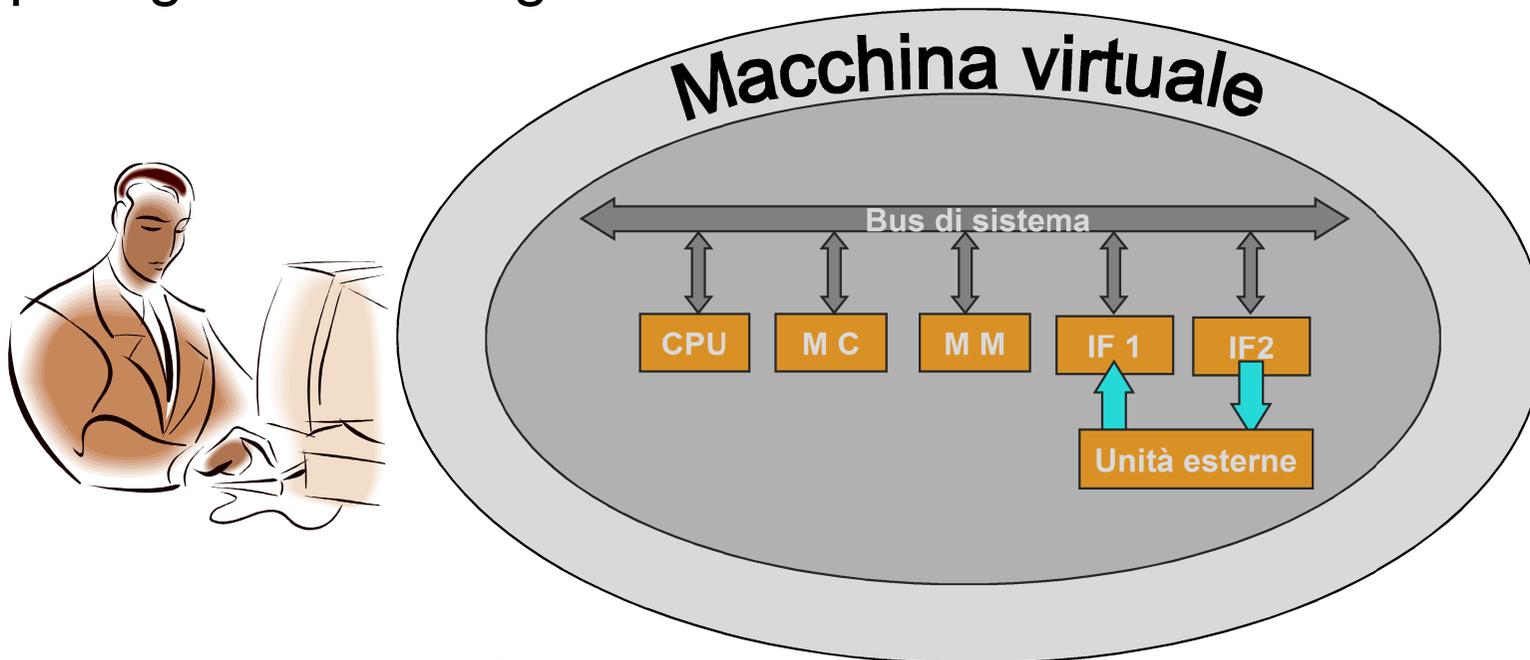
# Vantaggi

L'uso di un linguaggio di programmazione permette di :

- realizzare un programma che implementa l'algoritmo in maniera precisa ed in un linguaggio “ad alto livello”
- trascurare tutti i dettagli relativi alla rappresentazione dei dati nei registri
- definire un programma che non dipende dal particolare calcolatore su cui è stato realizzato

# Linguaggio=macchina virtuale

In effetti, l'utente non deve interagire con la macchina reale e le sue limitazioni, ma "vede" una macchina virtuale che nasconde le particolarità della macchina reale e con la quale è molto più agevole interagire.



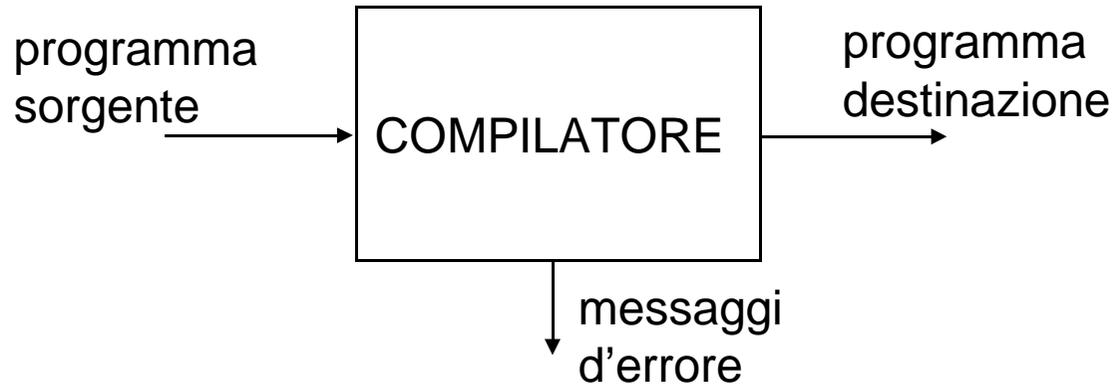
# Dal linguaggio ad alto livello al linguaggio macchina

- I linguaggi di programmazione ad alto livello sono linguaggi *formali*, in cui la forma delle frasi, cioè la sintassi, e il loro significato, la semantica, sono definiti sulla base di regole rigide e precise.
- In tal modo viene eliminata l'ambiguità e le ridondanze tipiche del linguaggio naturale ed è possibile realizzare in modo automatico l'analisi di un programma scritto in un linguaggio ad alto livello e la sua traduzione in linguaggio macchina.
- In altre parole è possibile definire algoritmi per verificare la correttezza grammaticale delle frasi e per calcolarne il significato.

# Programmi traduttori

- I programmi che svolgono il compito di tradurre un programma in linguaggio ad alto livello in un programma in linguaggio macchina sono detti *traduttori* e si dividono in due categorie:
  - **compilatori**
  - **interpreti**

# Compilatori



- Un compilatore è un **programma** che riceve in input un programma scritto in un linguaggio (*linguaggio sorgente*) e lo traduce in un **programma equivalente** scritto in un altro linguaggio (*linguaggio destinazione o target*).
- Una parte importante del processo di traduzione è il controllo per verificare la presenza di errori nel programma sorgente.

# Il modello di compilazione

## Analisi-Sintesi

- La compilazione si può dividere in due parti: analisi e sintesi
- La parte di **analisi** spezza il programma sorgente in pezzi costituenti e ne crea una rappresentazione intermedia.
- La parte di **sintesi** costruisce il programma destinazione a partire dalla rappresentazione intermedia

# Analisi

- L'analisi consta di tre fasi:
  - **Analisi lessicale**
  - **Analisi sintattica**
  - **Analisi semantica**

# Analisi lessicale (o lineare)

- Nell'**analisi lessicale** il flusso di caratteri che costituisce il programma sorgente è letto da sinistra verso destra e raggruppato in *tokens*. Un **token** è una singola unità atomica del linguaggio (es. una *parola chiave*, un *identificatore* o un *simbolo*).

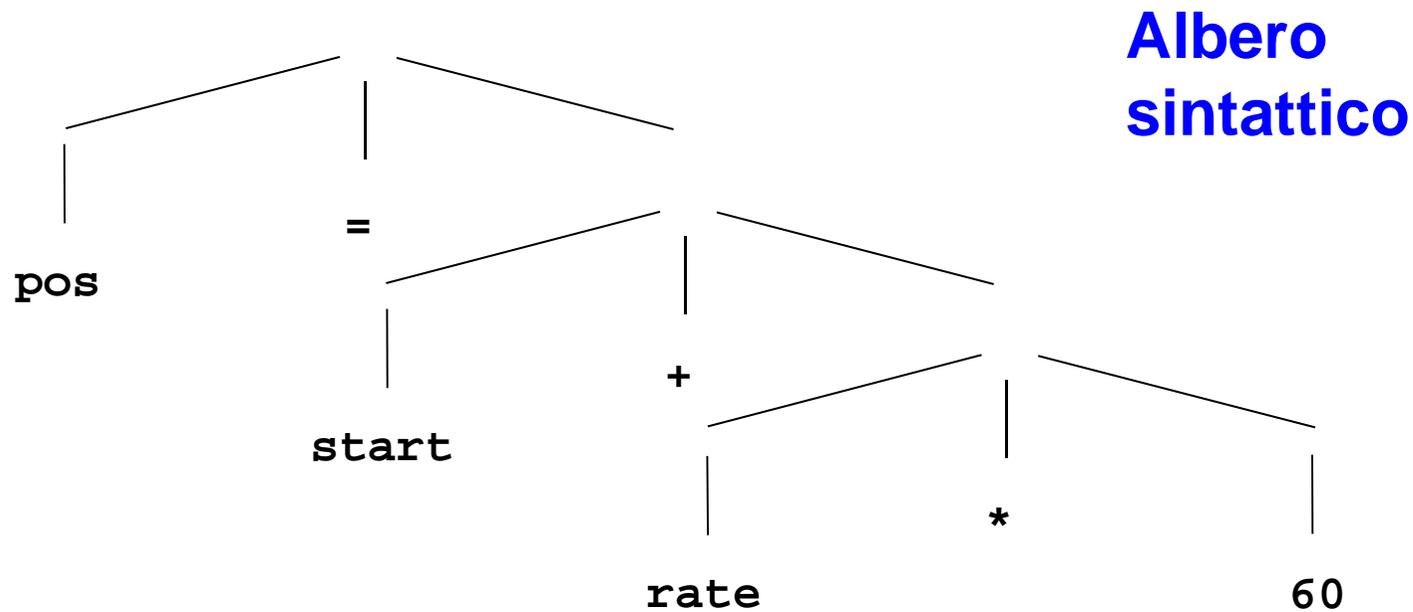
# Analisi lessicale. Esempio

- Si consideri l'istruzione di assegnazione  
`pos = start + rate * 60`
- L'analisi lessicale individua i seguenti token:
  - l'identificatore `pos`
  - il simbolo di assegnazione `=`
  - l'identificatore `start`
  - il segno `+`
  - l'identificatore `rate`
  - il segno `*`
  - la costante numerica `60`
- Sequenza di caratteri → sequenza di token

# Analisi sintattica

- Nell'**analisi sintattica** i tokens individuati nella fase precedente sono raggruppati in strutture costruite secondo le regole di una *grammatica formale* che costituisce la sintassi del linguaggio.
- Di solito tali strutture sono rappresentate sotto forma di *alberi sintattici* (parse tree).

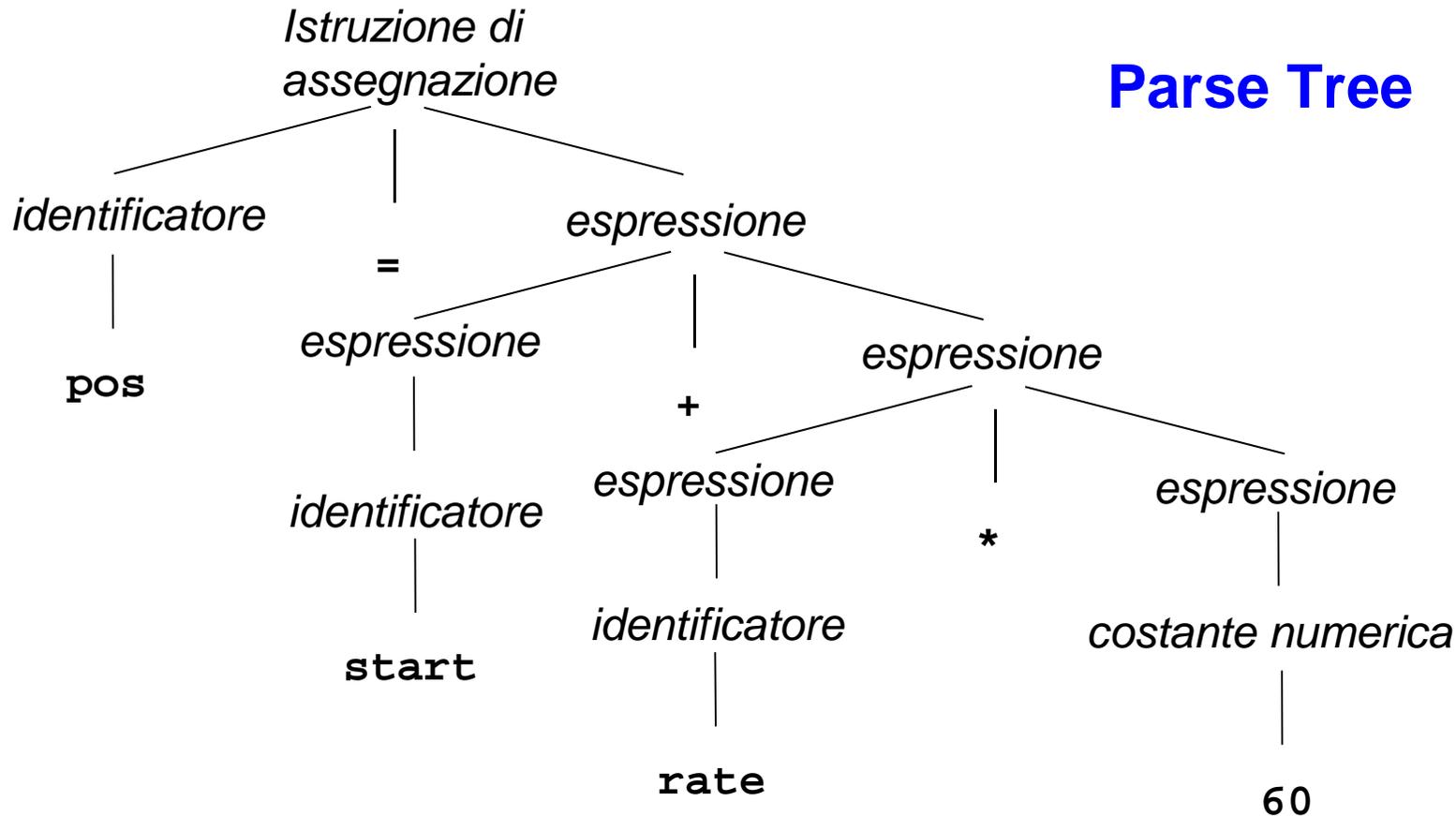
# Analisi sintattica. Esempio.



# Analisi sintattica. Esempio.

- Regole utilizzate:
  - Se  $id$  è un identificatore ed  $expr$  è un'espressione, allora  $id = expr$  è un'istruzione di assegnazione
  - Un identificatore è un'espressione
  - Una costante numerica è un'espressione
  - Se  $expr1$  ed  $expr2$  sono espressioni, allora sono espressioni anche:
    - $expr1 + expr2$
    - $expr1 * expr2$

# Analisi sintattica. Esempio.



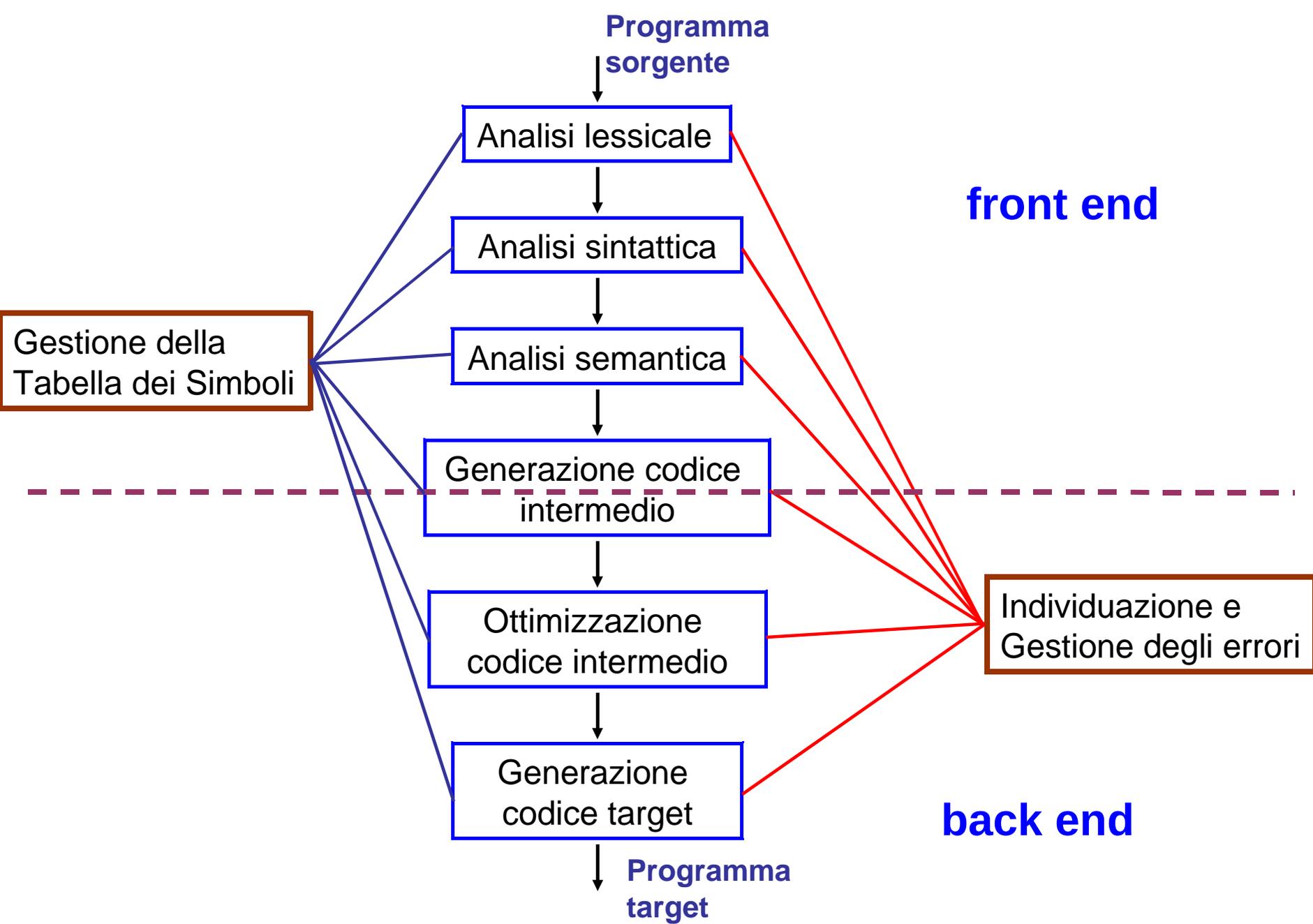
# Analisi semantica

- Nella fase di analisi semantica il compilatore aggiunge informazioni semantiche all'albero sintattico e costruisce la tabella dei simboli.
- In questa fase si operano verifiche semantiche quali la verifica dei tipi (**type checking**).
- In accordo al type checking, il compilatore verifica che ogni operatore ha operandi che sono permessi dalle specifiche del linguaggio sorgente.

# Sintesi

Nella sintesi si possono generalmente identificare tre fasi:

- **Generazione codice intermedio**
- **Ottimizzazione del codice intermedio**
- **Generazione del codice target**



**front end**

**back end**

- **Tabella dei simboli:**  
Raccoglie le informazioni su tutti gli identificatori utilizzati nel programma sorgente (nome, tipo, indirizzo di memoria)
- **Individuazione e gestione degli errori:**  
Al verificarsi di un errore (p.es. in una delle fasi di analisi) si valuta la sua gravità e si decide se proseguire o meno con la compilazione (*warning* / *fatal error*)

# Front end e back end

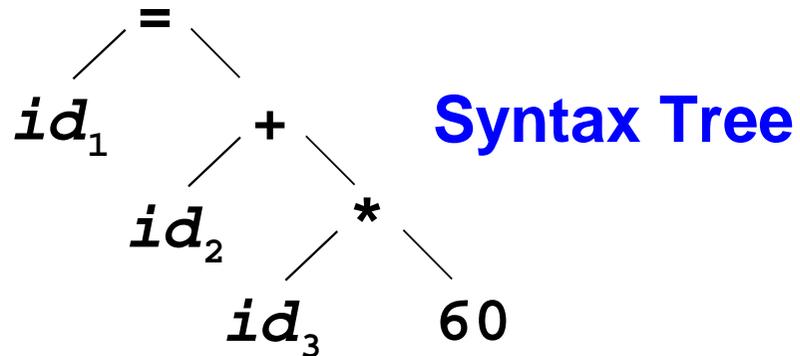
- Di solito si considera il compilatore formato da due sezioni: *front end* e *back end*.
- Il front end comprende quelle porzioni del compilatore che si occupano delle fasi che dipendono strettamente dal linguaggio sorgente e sono praticamente indipendenti dal linguaggio target. Tipicamente, fasi di analisi e generazione di codice intermedio
- Il back end include le parti del compilatore che dipendono dal linguaggio target e dal linguaggio intermedio.
- Ciò permette di riutilizzare un front end definito su un certo linguaggio sorgente per costruire più compilatori, ognuno con un diverso back end.

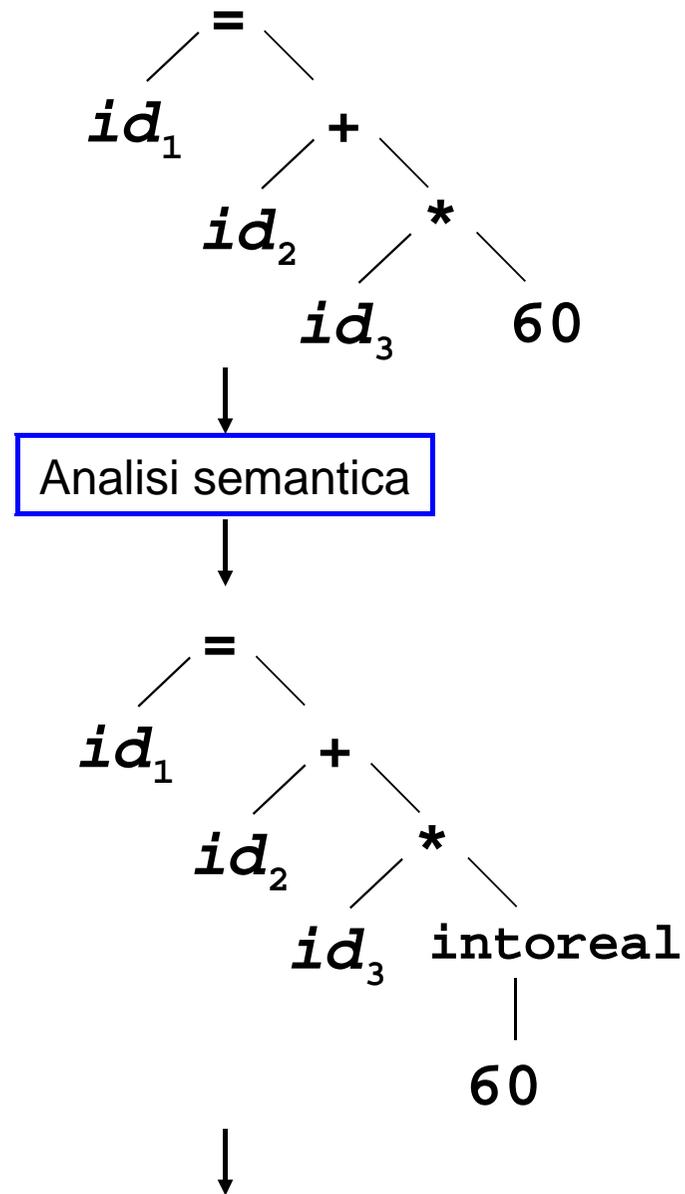
`pos = start + rate * 60`

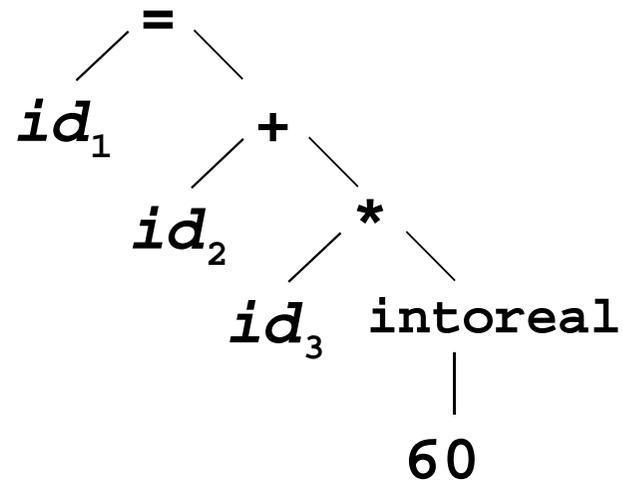
↓  
Analisi lessicale

`id1 = id2 + id3 * 60`

↓  
Analisi sintattica







↓

Generazione codice  
intermedio

↓

```
temp1=intoreal(60)
temp2=id3*temp1
temp3=id2+temp2
id1=temp3
```

```
temp1=intoreal(60)
temp2=id3*temp1
temp3=id2+temp2
id1=temp3
```

Ottimizzazione  
codice intermedio

```
temp1=id3*60.0
id1=id2+temp1
```

Generazione  
codice target

```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

# Interpreti

- Anche gli interpreti eseguono una fase di analisi e produzione di linguaggio macchina.
- La differenza rispetto ai compilatori è che l'interprete prende in esame un'istruzione alla volta, realizzandone la traduzione e l'esecuzione.
- Quindi, per ogni istruzione del programma in linguaggio ad alto livello, l'interprete
  - analizza l'istruzione e verifica la presenza di errori
  - in caso di errori, la traduzione viene arrestata; altrimenti, si producono e si eseguono le istruzioni in linguaggio macchina corrispondenti.

# Differenze tra compilatori e interpreti

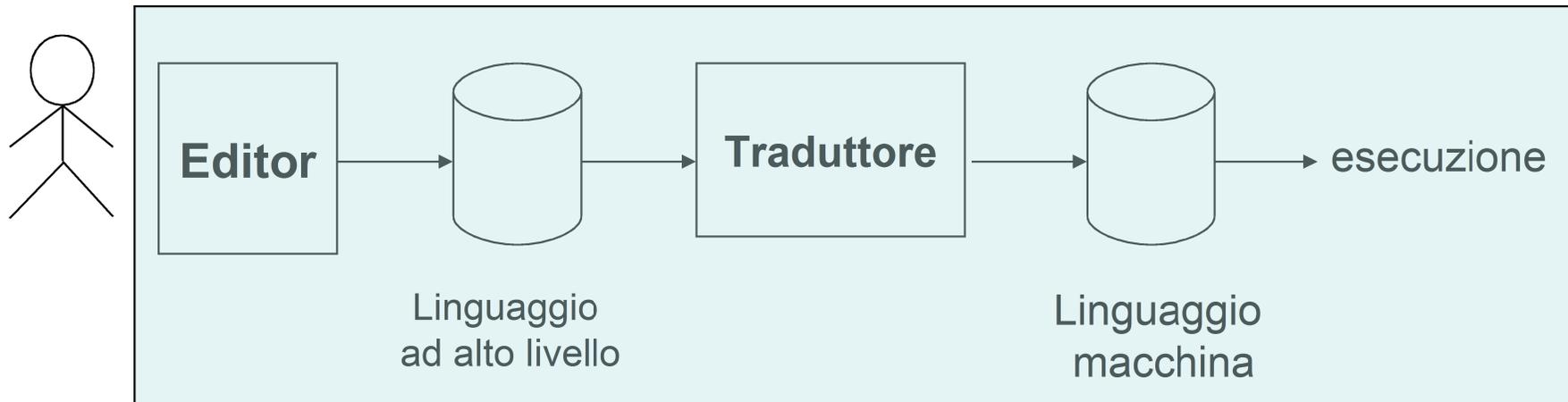
- Dal punto di vista dell' efficienza, l'esecuzione di un programma compilato è molto più veloce di quella di un programma interpretato.
- L'uso di un interprete potrebbe essere di aiuto nelle prime fasi di sviluppo di un programma, in quanto permette un immediata verifica della funzionalità del codice realizzato.

# Fasi di produzione di un programma

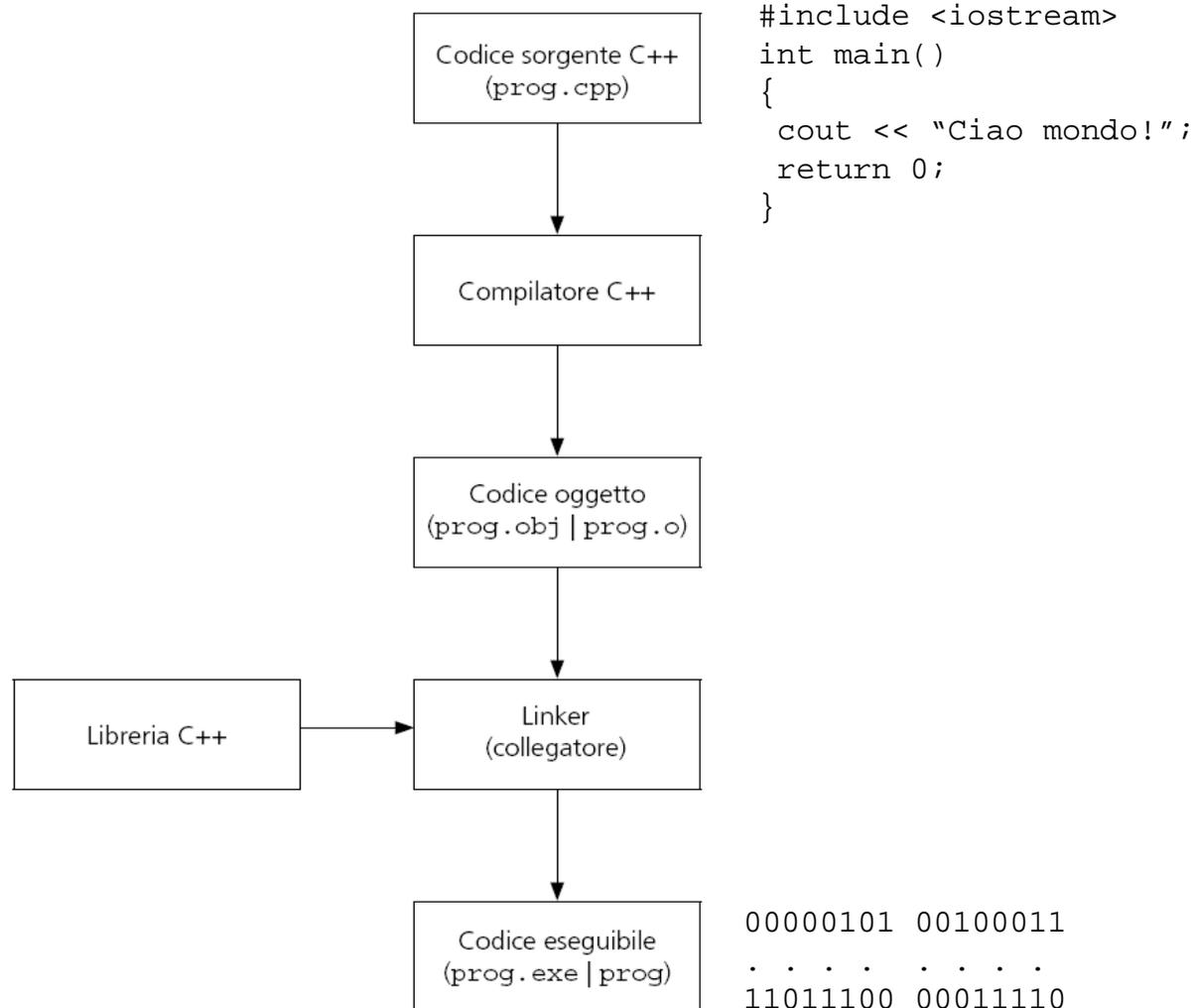
- Le fasi per la produzione di un programma che possa essere eseguito da un calcolatore sono quindi:
  - definizione dell'algoritmo e dei dati
  - implementazione dell'algoritmo tramite un programma in un linguaggio ad alto livello
  - traduzione in linguaggio macchina
  - esecuzione del programma in linguaggio macchina

# L'ambiente di sviluppo

- Nelle varie fasi c'è l'ausilio di opportuni programmi per:
  - la scrittura del programma in linguaggio ad alto livello (text editors)
  - la traduzione (compilatori o interpreti)
  - la verifica in esecuzione (debuggers)



# Traduzione di un programma C++



# Editor

- E' un programma che permette la scrittura e la memorizzazione su file delle istruzioni in linguaggio ad alto livello.
- Il file prodotto è un file di *tipo testo*: contiene, cioè, i caratteri scritti codificati in codice ASCII. In questo modo, è modificabile da altri programmi di *text editing* (es. NotePad) e trasferibile tra macchine diverse.
- Alcuni editors sono *orientati al linguaggio*, nel senso che organizzano il testo del programma in modo che sia resa più efficace la visualizzazione del codice. Tipiche caratteristiche di tali editors sono:
  - Indentazione
  - Syntax highlighting

# Debugger

- E' un programma che permette l'esecuzione *passo-passo* del programma eseguibile, visualizzando lo stato corrente dei dati del programma.
- In questo modo è possibile verificare la rispondenza del codice prodotto all'algoritmo da implementare.