



Capitolo 1

Computer, algoritmi e linguaggi

Sommario:

Introduzione

- Algoritmi

- Algoritmi e programmi

- La macchina di Von Neumann e il linguaggio assembler

- Linguaggio macchina e assembler

I linguaggi ad alto livello

- Compilatori e interpreti

Strumenti per la stesura dei programmi

- javac e Java Virtual Machine (JVM)

- Il ruolo della macchina astratta

- La programmazione strutturata

- Le strutture di controllo fondamentali

Variabili e assegnamenti

Sintassi e semantica

- Grammatiche

- Lessico del linguaggio Java

- Il primo programma Java

- ▶ **Computer**

Macchina elettronica programmabile al fine di svolgere diverse funzioni

- ▶ **Computer**

Macchina elettronica programmabile al fine di svolgere diverse funzioni

- ▶ **Programma**

Sequenza di istruzioni elementari che un computer è in grado di comprendere ed eseguire

- ▶ **Computer**

Macchina elettronica programmabile al fine di svolgere diverse funzioni

- ▶ **Programma**

Sequenza di istruzioni elementari che un computer è in grado di comprendere ed eseguire

- ▶ **Programmazione**

Attività che consiste nell'organizzare istruzioni elementari, direttamente comprensibili dall'esecutore, in strutture complesse (programmi) al fine di svolgere determinati compiti

- ▶ **Computer**

Macchina elettronica programmabile al fine di svolgere diverse funzioni

- ▶ **Programma**

Sequenza di istruzioni elementari che un computer è in grado di comprendere ed eseguire

- ▶ **Programmazione**

Attività che consiste nell'organizzare istruzioni elementari, direttamente comprensibili dall'esecutore, in strutture complesse (programmi) al fine di svolgere determinati compiti

- ▶ **Informatica**

Disciplina che si occupa dell'informazione e del suo trattamento in maniera automatica

Il trattamento dell'informazione coinvolge:

- ▶ **Mezzi fisici**: computer
- ▶ **Mezzi logici**: procedimenti di elaborazione, algoritmi

Il trattamento dell'informazione coinvolge:

- ▶ **Mezzi fisici**: computer
- ▶ **Mezzi logici**: procedimenti di elaborazione, algoritmi

Algoritmo

Insieme ordinato di passi **eseguibili** e **non ambigui**, che definiscono un processo che **termina**.

MCD fra x e y

1. Calcola il resto della divisione di x per y

MCD fra x e y

1. Calcola il resto della divisione di x per y
2. Se il resto è diverso da zero,
ricomincia dal **passo 1** utilizzando come x il valore attuale di y ,
e come y il valore del resto,

MCD fra x e y

1. Calcola il resto della divisione di x per y
2. Se il resto è diverso da zero,
ricomincia dal **passo 1** utilizzando come x il valore attuale di y ,
e come y il valore del resto,
altrimenti
prosegui con il passo successivo

MCD fra x e y

1. Calcola il resto della divisione di x per y
2. Se il resto è diverso da zero,
ricomincia dal **passo 1** utilizzando come x il valore attuale di y ,
e come y il valore del resto,
altrimenti
prosegui con il passo successivo
3. MCD è il valore attuale di y

MCD fra x e y

1. Calcola il resto della divisione di x per y
 2. Se il resto è diverso da zero,
ricomincia dal **passo 1** utilizzando come x il valore attuale di y ,
e come y il valore del resto,
altrimenti
prosegui con il passo successivo
 3. MCD è il valore attuale di y
- L'*intelligenza* necessaria per trovare la soluzione del problema è tutta codificata nell'algoritmo

MCD fra x e y

1. Calcola il resto della divisione di x per y
 2. Se il resto è diverso da zero,
ricomincia dal **passo 1** utilizzando come x il valore attuale di y ,
e come y il valore del resto,
altrimenti
prosegui con il passo successivo
 3. MCD è il valore attuale di y
- ▶ L'*intelligenza* necessaria per trovare la soluzione del problema è tutta codificata nell'algoritmo
 - ▶ Chiunque sappia comprendere ed eseguire le operazioni che costituiscono l'algoritmo di Euclide, può calcolare l'MCD

È un algoritmo ?

1. Crea un elenco di tutti i numeri primi

È un algoritmo ?

1. Crea un elenco di tutti i numeri primi
2. Ordina l'elenco in modo decrescente

È un algoritmo ?

1. Crea un elenco di tutti i numeri primi
2. Ordina l'elenco in modo decrescente
3. Preleva il primo elemento dall'elenco risultante

Un esempio dubbio

È un algoritmo ?

1. Crea un elenco di tutti i numeri primi
2. Ordina l'elenco in modo decrescente
3. Preleva il primo elemento dall'elenco risultante

Non lo è

La prima e la seconda istruzione non sono *effettivamente eseguibili* in quanto richiedono la manipolazione di infiniti elementi.

Programma

È l'espressione di un algoritmo in un linguaggio che l'esecutore è in grado di comprendere senza bisogno di ulteriori spiegazioni.

Programma

È l'espressione di un algoritmo in un linguaggio che l'esecutore è in grado di comprendere senza bisogno di ulteriori spiegazioni.

- Un algoritmo è un oggetto *astratto* (concettuale). Un programma è un' *espressione concreta* dell'algoritmo

Programma

È l'espressione di un algoritmo in un linguaggio che l'esecutore è in grado di comprendere senza bisogno di ulteriori spiegazioni.

- ▶ Un algoritmo è un oggetto *astratto* (concettuale). Un programma è un' *espressione concreta* dell'algoritmo
- ▶ Lo stesso algoritmo può essere espresso in differenti linguaggi, in base agli esecutori ai quali è destinato

Programma

È l'espressione di un algoritmo in un linguaggio che l'esecutore è in grado di comprendere senza bisogno di ulteriori spiegazioni.

- ▶ Un algoritmo è un oggetto *astratto* (concettuale). Un programma è un' *espressione concreta* dell'algoritmo
- ▶ Lo stesso algoritmo può essere espresso in differenti linguaggi, in base agli esecutori ai quali è destinato
- ▶ La scrittura del programma è una fase successiva all'individuazione dell'algoritmo per risolvere un determinato problema

La macchina di Von Neumann (1946)

- ▶ **Memoria**

Contiene il programma da eseguire e i dati da esso utilizzati

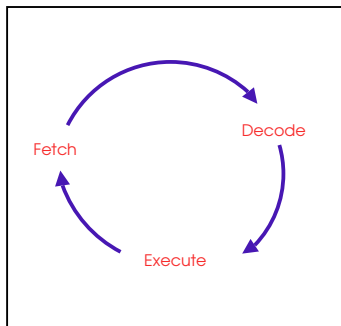
La macchina di Von Neumann (1946)

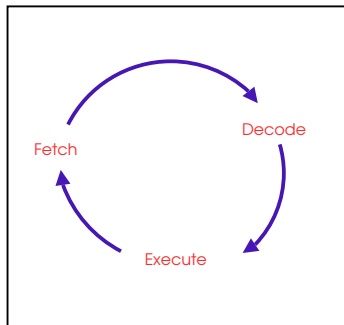
- ▶ **Memoria**

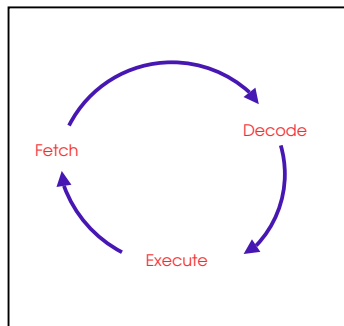
Contiene il programma da eseguire e i dati da esso utilizzati

- ▶ **Processore**

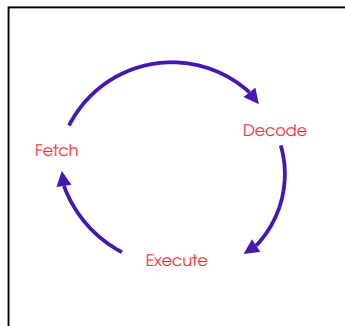
È l'esecutore che opera ripetendo il ciclo



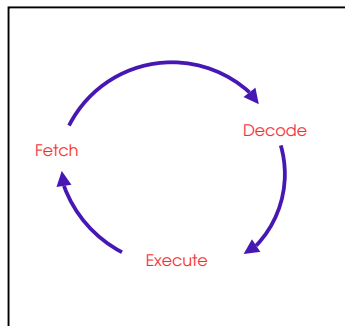




- **Fetch**: preleva dalla memoria la prossima istruzione da eseguire



- ▶ **Fetch**: preleva dalla memoria la prossima istruzione da eseguire
- ▶ **Decode**: interpreta l'istruzione, cioè ne riconosce il significato



- ▶ **Fetch**: preleva dalla memoria la prossima istruzione da eseguire
- ▶ **Decode**: interpreta l'istruzione, cioè ne riconosce il significato
- ▶ **Execute**: esegue le operazioni elementari corrispondenti all'istruzione

- ▶ Ogni processore ha un proprio *linguaggio macchina* con un proprio formato delle istruzioni

- ▶ Ogni processore ha un proprio *linguaggio macchina* con un proprio formato delle istruzioni
- ▶ Le istruzioni sono *sequenze di bit* che codificano:
 - ▶ l'operazione da eseguire
 - ▶ gli operandi su cui tale operazione deve essere eseguita (registri, locazioni di memoria, costanti ...)

- ▶ Ogni processore ha un proprio *linguaggio macchina* con un proprio formato delle istruzioni
- ▶ Le istruzioni sono *sequenze di bit* che codificano:
 - ▶ l'operazione da eseguire
 - ▶ gli operandi su cui tale operazione deve essere eseguita (registri, locazioni di memoria, costanti ...)
- ▶ Il linguaggio *assembler* di un processore è la *versione simbolica del linguaggio macchina*

Un assembler “giocattolo”

- ▶ Celle di memoria individuate da interi, contengono interi

Un assembler “giocattolo”

- ▶ Celle di memoria individuate da interi, contengono interi
- ▶ Registri del processore $R1, R2, \dots$, contengono interi

Un assembler “giocattolo”

- ▶ Celle di memoria individuate da interi, contengono interi
- ▶ Registri del processore $R1, R2, \dots$, contengono interi
- ▶ Istruzioni di trasferimento:
 - ▶ **LOAD R, X** : copia il contenuto della cella X nel registro R
 - ▶ **STORE R, X** : copia il contenuto del registro R nella cella X

Un assembler “giocattolo”

- ▶ Celle di memoria individuate da interi, contengono interi
- ▶ Registri del processore **R1, R2, ...**, contengono interi
- ▶ Istruzioni di trasferimento:
 - ▶ **LOAD R, X** : copia il contenuto della cella **X** nel registro **R**
 - ▶ **STORE R, X** : copia il contenuto del registro **R** nella cella **X**
- ▶ Istruzioni aritmetiche su interi:
 - ▶ **ADD R1, R2** : somma fra **R1** e **R2**, risultato in **R1**
 - ▶ **MUL R1, R2** : moltiplicazione...
 - ▶ **SUB R1, R2** : sottrazione...
 - ▶ **DIV R1, R2** : divisione...

Un assembler “giocattolo”

- ▶ Celle di memoria individuate da interi, contengono interi
- ▶ Registri del processore **R1, R2, ...**, contengono interi
- ▶ Istruzioni di trasferimento:
 - ▶ **LOAD R, X** : copia il contenuto della cella **X** nel registro **R**
 - ▶ **STORE R, X** : copia il contenuto del registro **R** nella cella **X**
- ▶ Istruzioni aritmetiche su interi:
 - ▶ **ADD R1, R2** : somma fra **R1** e **R2**, risultato in **R1**
 - ▶ **MUL R1, R2** : moltiplicazione...
 - ▶ **SUB R1, R2** : sottrazione...
 - ▶ **DIV R1, R2** : divisione...
- ▶ Istruzioni di salto:

incondizionato

```
JUMP alfa
```

```
...
```

```
alfa:...
```

Salta all'istruzione con etichetta **alfa**.

Un assembler “giocattolo”

- ▶ Celle di memoria individuate da interi, contengono interi
- ▶ Registri del processore **R1, R2, ...**, contengono interi
- ▶ Istruzioni di trasferimento:
 - ▶ **LOAD R, X** : copia il contenuto della cella **X** nel registro **R**
 - ▶ **STORE R, X** : copia il contenuto del registro **R** nella cella **X**
- ▶ Istruzioni aritmetiche su interi:
 - ▶ **ADD R1, R2** : somma fra **R1** e **R2**, risultato in **R1**
 - ▶ **MUL R1, R2** : moltiplicazione...
 - ▶ **SUB R1, R2** : sottrazione...
 - ▶ **DIV R1, R2** : divisione...
- ▶ Istruzioni di salto:

incondizionato

JUMP alfa

...

alfa:...

Salta all'istruzione con etichetta **alfa**.

condizionato

JZERO R1, alfa

...

alfa:...

Se **R1** contiene **0** salta all'istruzione con etichetta **alfa**, altrimenti prosegui.

MCD fra x e y

1. Calcola il *resto* della divisione di x per y

MCD fra x e y

1. Calcola il *resto* della divisione di x per y
2. Se $\text{resto} \neq 0$,
ricomincia dal **passo 1** con
 $x \leftarrow y$ e $y \leftarrow \text{resto}$

MCD fra x e y

1. Calcola il *resto* della divisione di x per y
2. Se $\text{resto} \neq 0$,
ricomincia dal **passo 1** con
 $x \leftarrow y$ e $y \leftarrow \text{resto}$
altrimenti
prosegui con il passo successivo

MCD fra x e y

1. Calcola il *resto* della divisione di x per y
2. Se $\text{resto} \neq 0$,
 ricomincia dal **passo 1** con
 $x \leftarrow y$ e $y \leftarrow \text{resto}$
 altrimenti
 prosegui con il passo successivo
3. MCD è il valore di y

MCD fra x e y

1. Calcola il *resto* della divisione di x per y
2. Se *resto* $\neq 0$,
 ricomincia dal **passo 1** con
 $x \leftarrow y$ e $y \leftarrow \text{resto}$
 altrimenti
 prosegui con il passo successivo
3. MCD è il valore di y

```
        LOAD  R1, 101
        LOAD  R2, 102
alfa:   DIV   R1, R2
        MUL   R1, R2
        LOAD  R2, 101
        SUB   R2, R1
        JZERO R2, fine
        LOAD  R1, 102
        STORE R1, 101
        STORE R2, 102
        JUMP  alfa
fine:   LOAD  R1, 102
        STORE R1, 103
```

L'algoritmo di Euclide in assembler

```
// input: #101 contiene x, #102 contiene y
    LOAD  R1, 101    // R1 <- x
    LOAD  R2, 102    // R2 <- y
    // calcola x % y
alfa: DIV   R1, R2    // R1 <- (x / y)
    MUL   R1, R2    // R1 <- (x / y) * y
    LOAD  R2, 101    // R2 <- x
    SUB   R2, R1    // R2 <- x - ((x / y) * y) (resto)

    JZERO R2, fine    // se (resto = 0) vai a fine
    // altrimenti (resto != 0) continua
    LOAD  R1, 102    // R1 <- y
    STORE R1, 101    // x <- y
    STORE R2, 102    // y <- resto
    JUMP  alfa        // torna ad alfa

fine: LOAD  R1, 102    // R1 <- y      (y = MCD)
    STORE R1, 103    // 103 <- MCD

// output: #103 contiene MCD
```

- È necessario **conoscere i dettagli dell'architettura** del processore utilizzato e il relativo linguaggio

Svantaggi del linguaggio macchina

- ▶ È necessario **conoscere i dettagli dell'architettura** del processore utilizzato e il relativo linguaggio
- ▶ Risulta **impossibile trasportare i programmi** da una macchina ad una differente

Svantaggi del linguaggio macchina

- ▶ È necessario **conoscere i dettagli dell'architettura** del processore utilizzato e il relativo linguaggio
- ▶ Risulta **impossibile trasportare i programmi** da una macchina ad una differente
- ▶ Il programmatore si specializza nell'uso di **“trucchi” legati alle caratteristiche specifiche della macchina**
I programmi risultano difficili da comprendere e da modificare

Svantaggi del linguaggio macchina

- ▶ È necessario **conoscere i dettagli dell'architettura** del processore utilizzato e il relativo linguaggio
- ▶ Risulta **impossibile trasportare i programmi** da una macchina ad una differente
- ▶ Il programmatore si specializza nell'uso di **“trucchi” legati alle caratteristiche specifiche della macchina**
I programmi risultano difficili da comprendere e da modificare
- ▶ La **struttura logica del programma è nascosta**
Difficile comprendere il programma e correggerlo in presenza di errori

Obiettivo

Rendere la programmazione indipendente dalle caratteristiche peculiari della macchina utilizzata.

- ▶ Non sono pensati per essere compresi direttamente da macchine reali...
...ma da macchine *“astratte”*, in grado di effettuare operazioni più ad alto livello rispetto alle operazioni elementari dei processori reali

Obiettivo

Rendere la programmazione indipendente dalle caratteristiche peculiari della macchina utilizzata.

- ▶ Non sono pensati per essere compresi direttamente da macchine reali. . .
... ma da macchine “*astratte*”, in grado di effettuare operazioni più ad alto livello rispetto alle operazioni elementari dei processori reali
- ▶ L'attività di programmazione viene svincolata dalla conoscenza dei dettagli architetturali della macchina utilizzata

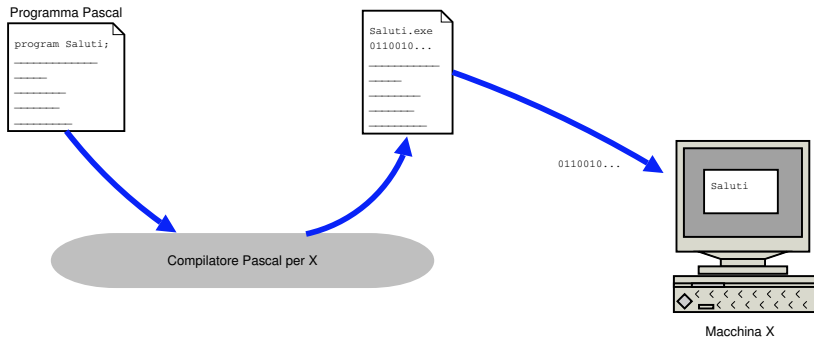
- La macchina astratta viene implementata sulla macchina reale **M** da un opportuno strumento di “traduzione”

- ▶ La macchina astratta viene implementata sulla macchina reale **M** da un opportuno strumento di “traduzione”
 - ▶ **Compilatore**
È un programma che *traduce* un programma del linguaggio **L** in un programma equivalente nel linguaggio macchina di **M**

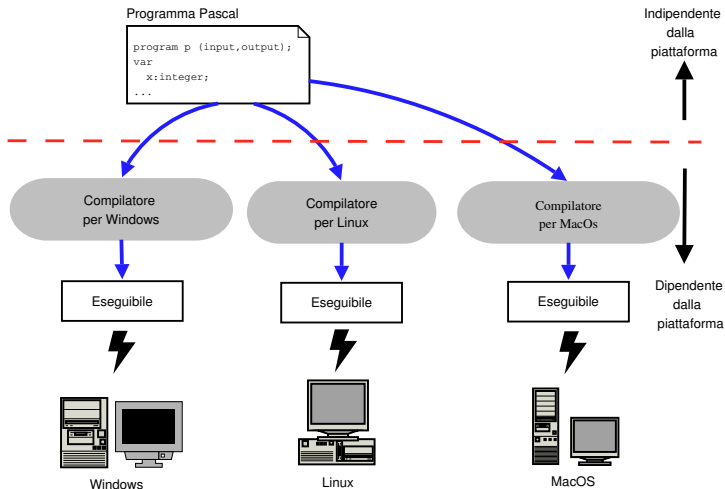
- ▶ La macchina astratta viene implementata sulla macchina reale **M** da un opportuno strumento di “traduzione”
 - ▶ **Compilatore**
È un programma che *traduce* un programma del linguaggio **L** in un programma equivalente nel linguaggio macchina di **M**
 - ▶ **Interprete**
È un programma che simula direttamente la macchina astratta:
 - legge un'istruzione del programma P

- ▶ La macchina astratta viene implementata sulla macchina reale **M** da un opportuno strumento di “traduzione”
 - ▶ **Compilatore**
È un programma che *traduce* un programma del linguaggio **L** in un programma equivalente nel linguaggio macchina di **M**
 - ▶ **Interprete**
È un programma che simula direttamente la macchina astratta:
 - legge un'istruzione del programma P
 - effettua le operazioni del linguaggio macchina corrispondenti al suo significato

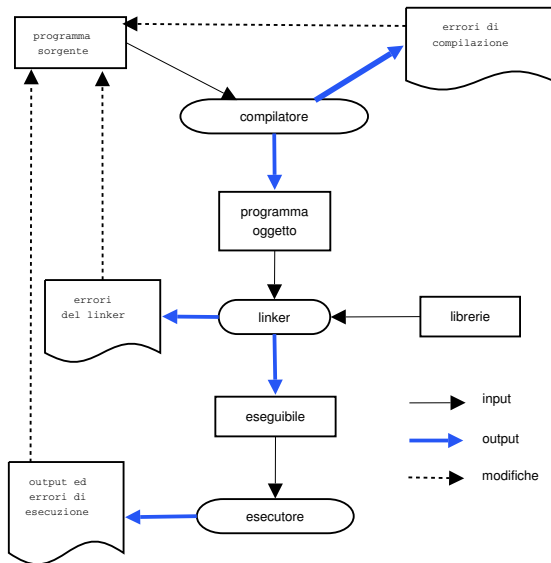
- ▶ La macchina astratta viene implementata sulla macchina reale **M** da un opportuno strumento di “traduzione”
 - ▶ **Compilatore**
È un programma che *traduce* un programma del linguaggio **L** in un programma equivalente nel linguaggio macchina di **M**
 - ▶ **Interprete**
È un programma che simula direttamente la macchina astratta:
 - legge un'istruzione del programma P
 - effettua le operazioni del linguaggio macchina corrispondenti al suo significato
 - passa a considerare l'istruzione successiva



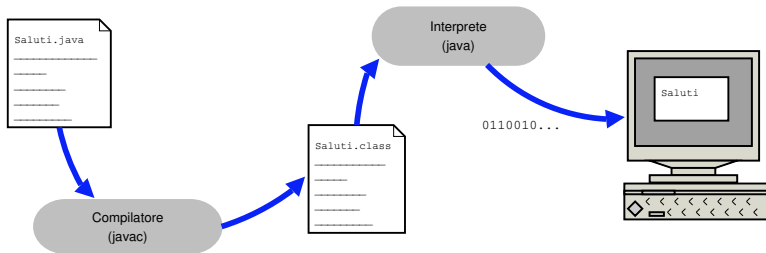
Vantaggi: portabilità del sorgente



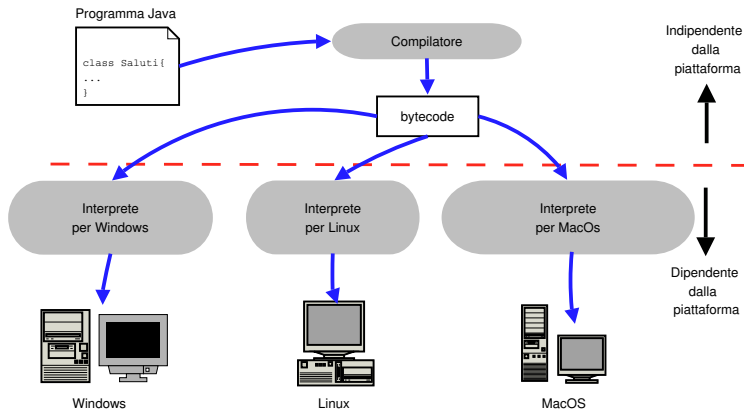
Strumenti per la stesura dei programmi



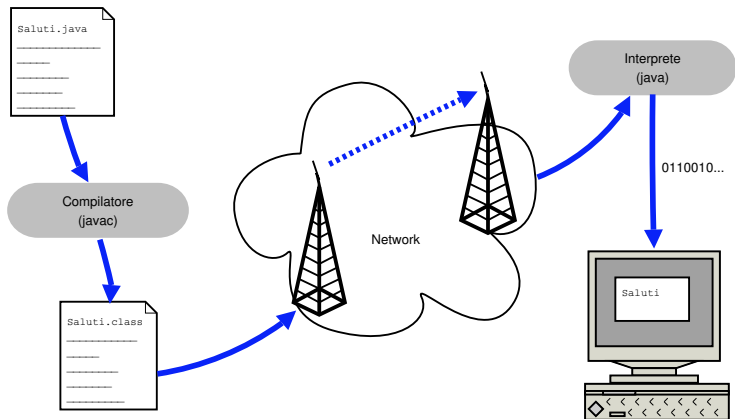
Java Virtual Machine (JVM)



Vantaggi: portabilità del bytecode



Il vantaggio di questo meccanismo



- ▶ Il linguaggio assembler ha una *semantica* (significato delle istruzioni) descritta in termini delle operazioni del processore (*semantica operativa*)

- Il linguaggio assembler ha una *semantica* (significato delle istruzioni) descritta in termini delle operazioni del processore (*semantica operativa*)

```
LOAD R1, 102
```

Copia il valore della locazione di memoria 102 nel registro R1

- Il linguaggio assembler ha una *semantica* (significato delle istruzioni) descritta in termini delle operazioni del processore (*semantica operativa*)

LOAD R1, 102

Copia il valore della locazione di memoria 102 nel registro R1

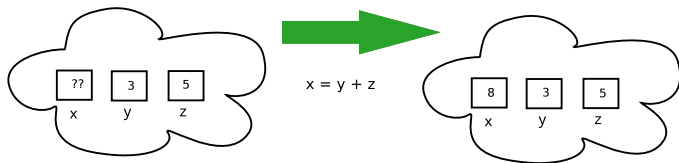
- Per descrivere il significato delle istruzioni di un linguaggio ad alto livello possiamo fare riferimento ad una *macchina astratta*

- L'introduzione dei linguaggi ad alto livello comporta *l'aggiunta di un livello di astrazione* rispetto all'architettura della macchina

- ▶ L'introduzione dei linguaggi ad alto livello comporta *l'aggiunta di un livello di astrazione* rispetto all'architettura della macchina
- ▶ Un programmatore Java *scrive i programmi facendo riferimento alla macchina astratta Java...*

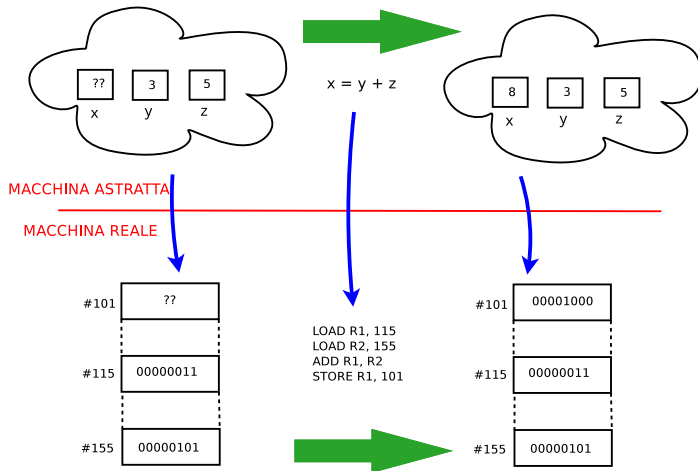
...cioè pensa i programmi in termini delle operazioni della macchina astratta non in termini delle operazioni del processore

La macchina astratta



MACCHINA ASTRATTA

La macchina astratta



- ▶ Metodologia introdotta agli inizi degli anni settanta

- ▶ Metodologia introdotta agli inizi degli anni settanta
- ▶ L'esecutore è guidato alla sequenza di esecuzione opportuna, tra tutte quelle possibili, mediante tre *strutture di controllo* fondamentali:

- ▶ Metodologia introdotta agli inizi degli anni settanta
- ▶ L'esecutore è guidato alla sequenza di esecuzione opportuna, tra tutte quelle possibili, mediante tre *strutture di controllo* fondamentali:
 - ▶ **Sequenza**: permette di eseguire le istruzioni secondo l'ordine in cui sono scritte

- ▶ Metodologia introdotta agli inizi degli anni settanta
- ▶ L'esecutore è guidato alla sequenza di esecuzione opportuna, tra tutte quelle possibili, mediante tre *strutture di controllo* fondamentali:
 - ▶ **Sequenza**: permette di eseguire le istruzioni secondo l'ordine in cui sono scritte
 - ▶ **Selezione**: permette di scegliere l'esecuzione di un blocco di istruzioni tra due possibili in base al valore di una condizione

- ▶ Metodologia introdotta agli inizi degli anni settanta
- ▶ L'esecutore è guidato alla sequenza di esecuzione opportuna, tra tutte quelle possibili, mediante tre *strutture di controllo* fondamentali:
 - ▶ **Sequenza**: permette di eseguire le istruzioni secondo l'ordine in cui sono scritte
 - ▶ **Selezione**: permette di scegliere l'esecuzione di un blocco di istruzioni tra due possibili in base al valore di una condizione
 - ▶ **Iterazione**: permette di ripetere l'esecuzione di una o più istruzioni in base al valore di una condizione

- ▶ L'impiego di queste strutture migliora la leggibilità dei programmi

- ▶ L'impiego di queste strutture migliora la leggibilità dei programmi
 - ▶ Ogni struttura di controllo ha un solo *punto di ingresso* e un solo *punto d'uscita*

- ▶ L'impiego di queste strutture migliora la leggibilità dei programmi
 - ▶ Ogni struttura di controllo ha un solo *punto di ingresso* e un solo *punto d'uscita*
 - ▶ Il flusso di esecuzione è evidente dalla struttura del codice

- ▶ L'impiego di queste strutture migliora la leggibilità dei programmi
 - ▶ Ogni struttura di controllo ha un solo *punto di ingresso* e un solo *punto d'uscita*
 - ▶ Il flusso di esecuzione è evidente dalla struttura del codice

Completezza

Tutti i programmi esprimibili tramite istruzioni di salto (*goto*) o diagrammi di flusso (*flow-chart*) possono essere riscritti utilizzando esclusivamente le tre strutture di controllo fondamentali.

- ▶ Le istruzioni sono eseguite nello stesso ordine in cui compaiono nel programma, cioè secondo la *sequenza* in cui sono scritte.

- Le istruzioni sono eseguite nello stesso ordine in cui compaiono nel programma, cioè secondo la *sequenza* in cui sono scritte.

Somma di due numeri

leggi i numeri a , b

calcola $a + b$

scrivi il risultato

Sintassi

SE *condizione*

ALLORA

blocco1

ALTRIMENTI

blocco2

FINESE

Sintassi

```
SE condizione  
  ALLORA  
    blocco1  
  ALTRIMENTI  
    blocco2  
FINESE
```

Esecuzione

(1) Viene valutata *condizione*

Sintassi

```
SE condizione  
  ALLORA  
    blocco1  
  ALTRIMENTI  
    blocco2  
FINESE
```

Esecuzione

(1) Viene valutata *condizione*

- ▶ *se è vera*, vengono eseguite le istruzioni del *blocco1*

Sintassi

```
SE condizione  
  ALLORA  
    blocco1  
  ALTRIMENTI  
    blocco2  
FINESE
```

Esecuzione

(1) Viene valutata *condizione*

- ▶ *se è vera*, vengono eseguite le istruzioni del *blocco1*
- ▶ *se è falsa*, vengono eseguite quelle del *blocco2*

Sintassi

SE *condizione*

ALLORA

blocco1

ALTRIMENTI

blocco2

FINESE

Esecuzione

(1) Viene valutata *condizione*

- ▶ *se è vera*, vengono eseguite le istruzioni del *blocco1*
- ▶ *se è falsa*, vengono eseguite quelle del *blocco2*

(2) L'esecuzione procede con l'istruzione che segue immediatamente la fine del costrutto di selezione (FINESE)

Selezione (senza ALTRIMENTI)

Sintassi

SE *condizione*

ALLORA

blocco1

FINESE

Selezione (senza ALTRIMENTI)

Sintassi

```
SE condizione  
  ALLORA  
    blocco1  
FINESE
```

Esecuzione

Viene valutata *condizione*:

- ▶ *se è vera*, vengono eseguite le istruzioni del *blocco1*

Selezione (senza ALTRIMENTI)

Sintassi

SE *condizione*

ALLORA

blocco1

FINESE

Esecuzione

Viene valutata *condizione*:

- ▶ *se è vera*, vengono eseguite le istruzioni del *blocco1*
quindi l'esecuzione riprende dalla prima istruzione che segue il costrutto di selezione

Selezione (senza ALTRIMENTI)

Sintassi

SE *condizione*

ALLORA

blocco1

FINESE

Esecuzione

Viene valutata *condizione*:

- ▶ *se è vera*, vengono eseguite le istruzioni del *blocco1*
quindi l'esecuzione riprende dalla prima istruzione che segue il costrutto di selezione
- ▶ *se è falsa*, l'esecuzione prosegue direttamente dalla prima istruzione che segue il costrutto di selezione

- ▶ Calcolo della divisione tra due numeri controllando che il divisore sia diverso da zero:

leggi il dividendo e il divisore

SE il divisore è diverso da zero

ALLORA

calcola dividendo/divisore

scrivi il risultato

ALTRIMENTI

scrivi "errore: divisione per zero"

FINESE

Esempio: calcolo delle radici di $ax^2 + bx + c = 0$

leggi i valori dei parametri a , b , c

calcola il discriminante

SE il discriminante è minore di zero

ALLORA

scrivi "nessuna soluzione reale"

ALTRIMENTI

SE il discriminante è uguale a zero

ALLORA

calcola $\frac{-b}{2a}$

scrivi "Due soluzioni coincidenti: ", il risultato

ALTRIMENTI

scrivi "Due soluzioni: ",

calcola $\frac{-b - \sqrt{b^2 - 4ac}}{2a}$

scrivi il risultato

calcola $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$

scrivi il risultato

FINESE

FINESE

Iterazione (schema 1)

Sintassi

ESEGUI

blocco

QUANDO condizione

Iterazione (schema 1)

Sintassi

ESEGUI

blocco

QUANDO condizione

Esecuzione

(1) Viene eseguito *blocco*

Iterazione (schema 1)

Sintassi

ESEGUI

blocco

QUANDO condizione

Esecuzione

- (1) Viene eseguito *blocco*
- (2) Viene valutata *condizione*:

Iterazione (schema 1)

Sintassi

ESEGUI

blocco

QUANDO condizione

Esecuzione

- (1) Viene eseguito *blocco*
- (2) Viene valutata *condizione*:
 - ▶ *se è vera*, si ritorna al punto (1)

Iterazione (schema 1)

Sintassi

ESEGUI

blocco

QUANDO condizione

Esecuzione

- (1) Viene eseguito *blocco*
- (2) Viene valutata *condizione*:
 - ▶ *se è vera*, si ritorna al punto (1)
 - ▶ *se è falsa*, si prosegue con la prima istruzione scritta dopo il costrutto iterativo

Iterazione (schema 1)

Sintassi

ESEGUI

blocco

QUANDO condizione

Esecuzione

(1) Viene eseguito *blocco*

(2) Viene valutata *condizione*:

- ▶ *se è vera*, si ritorna al punto (1)
- ▶ *se è falsa*, si prosegue con la prima istruzione scritta dopo il costrutto iterativo

- ▶ il *blocco* è eseguito *almeno una volta*
- ▶ termina quando la *condizione* diventa falsa

Esempio: somma dei primi 100 numeri interi

- Calcolo iterativo, senza utilizzare la formula di Gauss ($\sum_{i=1}^n i = \frac{n(n+1)}{2}$)

poni il valore della somma a zero

inizia a considerare il numero 1

ESEGUI

aggiungi alla somma il numero che stai considerando

considera il numero successivo

QUANDO il numero che stai considerando non supera 100

scrivi la somma

Iterazione (schema 2)

Sintassi

QUANDO condizione ESEGUI

blocco

RIPETI

Iterazione (schema 2)

Sintassi

QUANDO condizione ESEGUI
 blocco
RIPETI

Esecuzione

(1) Viene valutata la *condizione*:

Iterazione (schema 2)

Sintassi

QUANDO condizione ESEGUI
 blocco
RIPETI

Esecuzione

- (1) Viene valutata la *condizione*:
 - ▶ *se è vera*, viene eseguito *blocco*
 quindi si torna al punto (1)

Iterazione (schema 2)

Sintassi

QUANDO condizione ESEGUI
 blocco
RIPETI

Esecuzione

(1) Viene valutata la *condizione*:

- ▶ *se è vera*, viene eseguito *blocco*
 quindi si torna al punto (1)
- ▶ *se è falsa*, l'esecuzione riprende dalla prima l'istruzione che segue il
 costrutto iterativo

Iterazione (schema 2)

Sintassi

QUANDO condizione ESEGUI
 blocco
RIPETI

Esecuzione

(1) Viene valutata la *condizione*:

- ▶ *se è vera*, viene eseguito *blocco*
 quindi si torna al punto (1)
- ▶ *se è falsa*, l'esecuzione riprende dalla prima l'istruzione che segue il
 costrutto iterativo

- ▶ il *blocco* può essere eseguito anche zero volte
- ▶ termina quando la *condizione* diventa falsa

- Il comportamento dello schema **QUANDO...RIPETI** può essere simulato combinando **ESEGUI...QUANDO...** e selezione

- Il comportamento dello schema **QUANDO...RIPETI** può essere simulato combinando **ESEGUI...QUANDO...** e selezione

```
SE condizione
  ALLORA
    ESEGUI
      blocco
    QUANDO condizione
  FINESE
```

- ▶ Una **variabile** è un *contenitore* preposto a contenere dei valori

- Una **variabile** è un *contenitore* preposto a contenere dei valori

Istruzione di assegnamento

variabile ← espressione

- Una **variabile** è un *contenitore* preposto a contenere dei valori

Istruzione di assegnamento

variabile \leftarrow espressione

Semantica operativa

(1) Viene calcolato il valore dell'espressione scritta a destra del simbolo \leftarrow

- Una **variabile** è un *contenitore* preposto a contenere dei valori

Istruzione di assegnamento

variabile \leftarrow espressione

Semantica operativa

- (1) Viene calcolato il valore dell'espressione scritta a destra del simbolo \leftarrow
- (2) Il risultato ottenuto è assegnato alla variabile (quindi posto nel contenitore) il cui nome è scritto a sinistra del simbolo \leftarrow , sovrascrivendo il valore precedentemente contenuto prima.

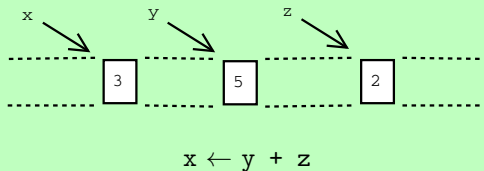
- ▶ Una **variabile** è un *contenitore* preposto a contenere dei valori

Istruzione di assegnamento

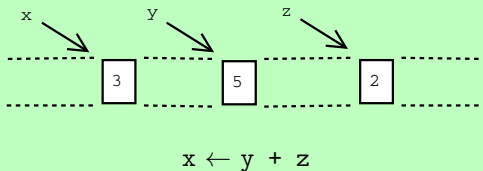
variabile ← espressione

Semantica operativa

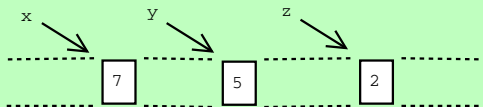
- (1) Viene calcolato il valore dell'espressione scritta a destra del simbolo ←
 - (2) Il risultato ottenuto è assegnato alla variabile (quindi posto nel contenitore) il cui nome è scritto a sinistra del simbolo ←, sovrascrivendo il valore precedentemente contenuto prima.
- ▶ **Osservazione:** Molti linguaggi, tra cui anche Java, utilizzano per l'assegnamento il simbolo =, usato comunemente per indicare l'uguaglianza.



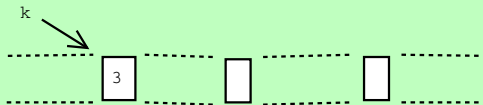
- Si *valuta l'espressione* $y+z$, recuperando i valori presenti in y e z e facendone la somma



- Si *valuta l'espressione* $y+z$, recuperando i valori presenti in y e z e facendone la somma

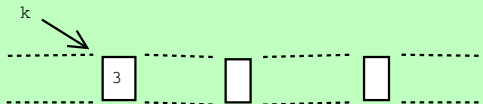


- Si pone il risultato nel contenitore di cui x è il nome sovrascrivendo il valore precedentemente contenuto



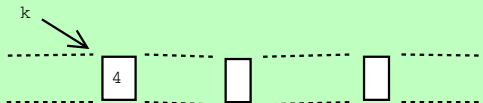
$k \leftarrow k + 1$

- Si valuta l'espressione $k+1$, recuperando il valore di k (3) e sommandogli 1



$k \leftarrow k + 1$

- Si valuta l'espressione $k+1$, recuperando il valore di k (3) e sommandogli 1



- Si pone il risultato nel contenitore di cui k è il nome sovrascrivendo il valore precedentemente contenuto

- ▶ Il **tipo** di una variabile specifica:
 - l'insieme dei valori che in essa possono essere memorizzati
 - l'insieme delle operazioni che possono essere effettuate su di essa

- ▶ Il **tipo** di una variabile specifica:
 - l'insieme dei valori che in essa possono essere memorizzati
 - l'insieme delle operazioni che possono essere effettuate su di essa

Esempio: x di tipo **intero**

- ▶ può assumere come valori solo *numeri interi*
- ▶ su di essa possono essere effettuate soltanto *le operazioni consentite per i numeri interi*

Variabili

- ▶ Il concetto di **variabile** è un'*astrazione* del concetto di locazione di memoria
- ▶ L'**assegnamento** di un valore a una variabile è un'astrazione dell'operazione STORE

Variabili

- ▶ Il concetto di **variabile** è un'*astrazione* del concetto di locazione di memoria
- ▶ L'**assegnamento** di un valore a una variabile è un'astrazione dell'operazione STORE

Tipi

- ▶ Tutte le variabili siano rappresentate nella memoria come *sequenze di bit*, tali sequenze sono *interpretate diversamente a seconda del tipo della variabile*

Variabili

- ▶ Il concetto di **variabile** è un'*astrazione* del concetto di locazione di memoria
- ▶ L'**assegnamento** di un valore a una variabile è un'astrazione dell'operazione STORE

Tipi

- ▶ Tutte le variabili siano rappresentate nella memoria come *sequenze di bit*, tali sequenze sono *interpretate diversamente a seconda del tipo della variabile*
 - ▶ 00001010 come **intero** 10
 - ▶ 00001010 come **char** 'A'

Variabili

- ▶ Il concetto di **variabile** è un'*astrazione* del concetto di locazione di memoria
- ▶ L'**assegnamento** di un valore a una variabile è un'astrazione dell'operazione STORE

Tipi

- ▶ Tutte le variabili siano rappresentate nella memoria come *sequenze di bit*, tali sequenze sono *interpretate diversamente a seconda del tipo della variabile*
 - ▶ 00001010 come *intero* 10
 - ▶ 00001010 come *char* 'A'
- ▶ La nozione di tipo fornisce un'astrazione rispetto alla *rappresentazione effettiva dei dati*, il programmatore può utilizzare variabili di tipo differente, senza la necessità di conoscerne l'effettiva rappresentazione

- ▶ Molti linguaggi richiedono di **dichiarare** le variabili prima del loro utilizzo.
 - ▶ Alcuni linguaggi (ad esempio Pascal) richiedono che le variabili siano dichiarate tutte all'inizio del programma
 - ▶ Alcuni linguaggi richiedono che siano dichiarate prima del loro utilizzo (ad esempio Java)

- ▶ Molti linguaggi richiedono di **dichiarare** le variabili prima del loro utilizzo.
 - ▶ Alcuni linguaggi (ad esempio Pascal) richiedono che le variabili siano dichiarate tutte all'inizio del programma
 - ▶ Alcuni linguaggi richiedono che siano dichiarate prima del loro utilizzo (ad esempio Java)
- ▶ **Vantaggi:**
 - ▶ accresce la leggibilità dei programmi
 - ▶ diminuisce la possibilità di errori
 - ▶ facilita la realizzazione di compilatori efficienti

Esempio: calcolo delle radici di $ax^2 + bx + c = 0$

variabili a, b, c, discriminante, x, x1, x2: numeri reali

leggi a, b, c

discriminante $\leftarrow b^2 - 4 * a * c$

SE discriminante < 0

ALLORA

scrivi "nessuna soluzione reale"

ALTRIMENTI

SE discriminante == 0

ALLORA

x $\leftarrow -b / (2 * a)$

scrivi "Due soluzioni coincidenti: ", x

ALTRIMENTI

x1 $\leftarrow (-b - \sqrt{\text{discriminante}}) / (2 * a)$

x2 $\leftarrow (-b + \sqrt{\text{discriminante}}) / (2 * a)$

scrivi "Due soluzioni: ", x1, x2

FINESE

FINESE

Sintassi di un linguaggio

Sintassi

Specifica *come si scrivono* le frasi del linguaggio.

Sintassi

Specifica *come si scrivono* le frasi del linguaggio.

- ▶ Esistono varie notazioni per descrivere la sintassi dei linguaggi:
 - ▶ BNF (Bakus-Naur Form)
 - ▶ Carte sintattiche
 - ▶ ...

Sintassi

Specifica *come si scrivono* le frasi del linguaggio.

- ▶ Esistono varie notazioni per descrivere la sintassi dei linguaggi:
 - ▶ BNF (Bakus-Naur Form)
 - ▶ Carte sintattiche
 - ▶ ...

Esempio: sintassi dei numeri reali (3.14)

```
<reale> ::= <seq_cifre> . <seq_cifre>  
<seq_cifre> ::= <cifra> | <cifra> <seq_cifre>  
<cifra> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Semantica di un linguaggio

Semantica

Specifica il *significato* di un programma.

Semantica di un linguaggio

Semantica

Specifica il *significato* di un programma.

Esempio: sintassi date

```
<data> ::= <c><c> . <c><c> . <c><c><c><c>  
      <c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Semantica

Specifica il *significato* di un programma.

Esempio: sintassi date

`<data> ::= <c><c> . <c><c> . <c><c><c><c>`

`<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

- 01.02.2015 è una data. Il giorno a cui questa data si riferisce *non è identificato dalla sintassi*

Semantica

Specifica il *significato* di un programma.

Esempio: sintassi date

```
<data> ::= <c><c> . <c><c> . <c><c><c><c>  
      <c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

- 01.02.2015 è una data. Il giorno a cui questa data si riferisce *non è identificato dalla sintassi*

```
01.02.2015  USA      2 Gennaio 2015  
01.02.2015  Europa  1 Febbraio 2015
```


- **Tutorial**

Si tratta di “visite guidate al linguaggio”. Lo scopo è quello di introdurre la semantica e la sintassi gradualmente, in genere tramite esempi.

- ▶ **Tutorial**

Si tratta di “visite guidate al linguaggio”. Lo scopo è quello di introdurre la semantica e la sintassi gradualmente, in genere tramite esempi.

- ▶ **Manuali di riferimento**

Descrivono in modo preciso (ma non formale) la sintassi e la semantica del linguaggio. La presentazione del linguaggio ruota attorno alla sintassi. Sono rivolti ai programmatori.

- ▶ **Tutorial**

Si tratta di “visite guidate al linguaggio”. Lo scopo è quello di introdurre la semantica e la sintassi gradualmente, in genere tramite esempi.

- ▶ **Manuali di riferimento**

Descrivono in modo preciso (ma non formale) la sintassi e la semantica del linguaggio. La presentazione del linguaggio ruota attorno alla sintassi. Sono rivolti ai programmatori.

- ▶ **Definizioni formali**

Sono rivolte agli specialisti, in genere la sintassi e la semantica vengono presentate tramite linguaggi formali (o semi-formali) allo scopo di eliminare le possibili ambiguità del linguaggio naturale. Sono utilizzati nella costruzione dei compilatori.

$$G = (T, N, P, S)$$

$$G = (T, N, P, S)$$

- ▶ T insieme finito dei *simboli terminali*, cioè dei simboli che costituiranno le sentenze del linguaggio

$$G = (T, N, P, S)$$

- ▶ T insieme finito dei *simboli terminali*, cioè dei simboli che costituiranno le sentenze del linguaggio
- ▶ N insieme finito dei *simboli non terminali*, o *metasimboli*, utilizzati nella costruzione delle sentenze del linguaggio

$$G = (T, N, P, S)$$

- ▶ T insieme finito dei *simboli terminali*, cioè dei simboli che costituiranno le sentenze del linguaggio
- ▶ N insieme finito dei *simboli non terminali*, o *metasimboli*, utilizzati nella costruzione delle sentenze del linguaggio
- ▶ P insieme finito delle *regole di produzione*

$$G = (T, N, P, S)$$

- ▶ T insieme finito dei *simboli terminali*, cioè dei simboli che costituiranno le sentenze del linguaggio
- ▶ N insieme finito dei *simboli non terminali*, o *metasimboli*, utilizzati nella costruzione delle sentenze del linguaggio
- ▶ P insieme finito delle *regole di produzione*
- ▶ S *simbolo iniziale*, $S \in N$ ed è il punto di partenza nella costruzione delle sentenze

$$G = (T, N, P, S)$$

- ▶ T insieme finito dei *simboli terminali*, cioè dei simboli che costituiranno le sentenze del linguaggio
- ▶ N insieme finito dei *simboli non terminali*, o *metasimboli*, utilizzati nella costruzione delle sentenze del linguaggio
- ▶ P insieme finito delle *regole di produzione*
- ▶ S *simbolo iniziale*, $S \in N$ ed è il punto di partenza nella costruzione delle sentenze

Linguaggio generato da G

L'insieme di tutte le sequenze di simboli terminali ottenibili applicando le regole di produzione dell'insieme P , a partire dal simbolo iniziale S .

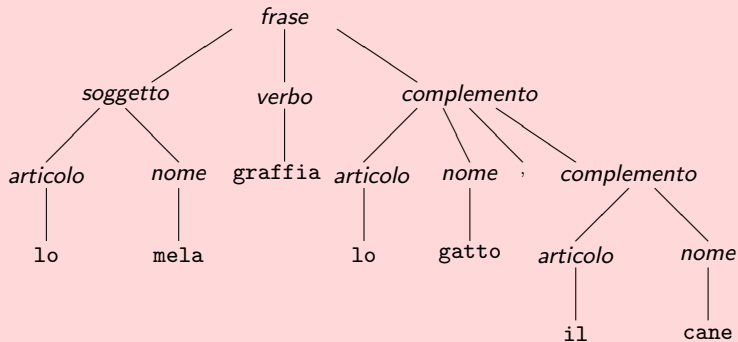
(1) $T = \{\text{il, lo, la, cane, mela, gatto, mangia, graffia, ,}\}$

- (1) $T = \{\text{il, lo, la, cane, mela, gatto, mangia, graffia, ,}\}$
- (2) $N = \{\text{frase, soggetto, verbo, complemento, articolo, nome}\}$

- (1) $T = \{\text{il, lo, la, cane, mela, gatto, mangia, graffia, ,}\}$
- (2) $N = \{\text{frase, soggetto, verbo, complemento, articolo, nome}\}$
- (3) P , regole espresse in *BNF* (forma di *Backus-Naur*):
 - ▶ $\text{frase} ::= \text{soggetto verbo complemento}$
 - ▶ $\text{soggetto} ::= \text{articolo nome}$
 - ▶ $\text{articolo} ::= \text{il} \mid \text{la} \mid \text{lo}$
 - ▶ $\text{nome} ::= \text{cane} \mid \text{mela} \mid \text{gatto}$
 - ▶ $\text{verbo} ::= \text{mangia} \mid \text{graffia}$
 - ▶ $\text{complemento} ::= \text{articolo nome} \mid \text{articolo nome , complemento}$

- (1) $T = \{\text{il, lo, la, cane, mela, gatto, mangia, graffia, ,}\}$
- (2) $N = \{\text{frase, soggetto, verbo, complemento, articolo, nome}\}$
- (3) P , regole espresse in *BNF* (forma di *Backus-Naur*):
 - ▶ $\text{frase} ::= \text{soggetto verbo complemento}$
 - ▶ $\text{soggetto} ::= \text{articolo nome}$
 - ▶ $\text{articolo} ::= \text{il} \mid \text{la} \mid \text{lo}$
 - ▶ $\text{nome} ::= \text{cane} \mid \text{mela} \mid \text{gatto}$
 - ▶ $\text{verbo} ::= \text{mangia} \mid \text{graffia}$
 - ▶ $\text{complemento} ::= \text{articolo nome} \mid \text{articolo nome , complemento}$
- (4) $S = \text{frase}$

Esempio di derivazione



- ▶ Formalismo grafico per la descrizione delle regole di produzione

- ▶ Formalismo grafico per la descrizione delle regole di produzione
- ▶ Si specifica una carta sintattica per ciascun simbolo non terminale della grammatica

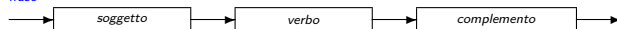
- ▶ Formalismo grafico per la descrizione delle regole di produzione
- ▶ Si specifica una carta sintattica **per ciascun simbolo non terminale della grammatica**
- ▶ In una carta sintattica:
 - ▶ i **rettangoli** indicano simboli non terminali (che andranno espansi con le carte sintattiche corrispondenti)

- ▶ Formalismo grafico per la descrizione delle regole di produzione
- ▶ Si specifica una carta sintattica **per ciascun simbolo non terminale della grammatica**
- ▶ In una carta sintattica:
 - ▶ i **rettangoli** indicano simboli non terminali (che andranno espansi con le carte sintattiche corrispondenti)
 - ▶ gli **ovali** indicano simboli terminali, che quindi non devono essere espansi ulteriormente

- ▶ Formalismo grafico per la descrizione delle regole di produzione
- ▶ Si specifica una carta sintattica **per ciascun simbolo non terminale della grammatica**
- ▶ In una carta sintattica:
 - ▶ i **rettangoli** indicano simboli non terminali (che andranno espansi con le carte sintattiche corrispondenti)
 - ▶ gli **ovali** indicano simboli terminali, che quindi non devono essere espansi ulteriormente
 - ▶ ogni **biforcazione** indica un'alternativa

L'esempio precedente

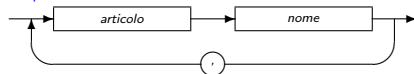
frase



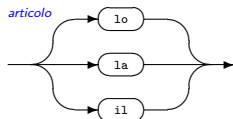
soggetto



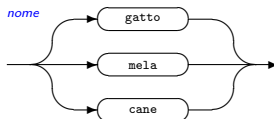
complemento



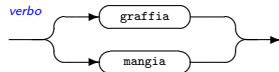
articolo



nome



verbo



► Alfabeto

L'alfabeto utilizzato per scrivere i programmi si chiama *Unicode*, ed è un insieme di caratteri rappresentati su 16 bit

- ▶ **Alfabeto**

L'alfabeto utilizzato per scrivere i programmi si chiama *Unicode*, ed è un insieme di caratteri rappresentati su 16 bit

- ▶ **Parole riservate (o parole chiave)**

Sono parole che nel linguaggio hanno un significato predeterminato. Non possono essere utilizzate diversamente e non possono essere ridefinite

► Alfabeto

L'alfabeto utilizzato per scrivere i programmi si chiama *Unicode*, ed è un insieme di caratteri rappresentati su 16 bit

► Parole riservate (o parole chiave)

Sono parole che nel linguaggio hanno un significato predeterminato. Non possono essere utilizzate diversamente e non possono essere ridefinite

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	extends	final
finally	float	for	goto	if	implements
import	instanceof	int	interface	long	native
new	package	private	protected	public	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	void
volatile	while				

► **Identificatori**

Sono nomi impiegati all'interno del programma per indicare variabili, classi, riferimenti a oggetti, e così via

Un identificatore è costituito da una sequenza di lettere e cifre che inizia con una lettera

► **Identificatori**

Sono nomi impiegati all'interno del programma per indicare variabili, classi, riferimenti a oggetti, e così via

Un identificatore è costituito da una sequenza di lettere e cifre che inizia con una lettera

► **Separatori**

Sono caratteri che permettono di separare o raggruppare parti di codice

() { } [] ; , @ ::

Linguaggio Java: il lessico (2)

► Identificatori

Sono nomi impiegati all'interno del programma per indicare variabili, classi, riferimenti a oggetti, e così via

Un identificatore è costituito da una sequenza di lettere e cifre che inizia con una lettera

► Separatori

Sono caratteri che permettono di separare o raggruppare parti di codice

() { } [] ; , @ ::

► Operatori

Sono simboli o sequenze di simboli che denotano alcune operazioni.

=	>	<	!	~	?	:	->			
==	<=	>=	!=	&&		++	--			
+	-	*	/	&		^	%	<<	>>	>>>
+=	-=	*=	/=	&=	=	^=	%=	<<=	>>=	>>>=

► **Letterali**

Sequenze di caratteri utilizzate all'interno dei programmi per rappresentare valori di tipi primitivi e stringhe

Linguaggio Java: il lessico (3)

- ▶ **Letterali**

Sequenze di caratteri utilizzate all'interno dei programmi per rappresentare valori di tipi primitivi e stringhe

- ▶ **Commenti**

- ▶ `/*...*/`

Il compilatore ignora tutto il testo compreso tra questi caratteri; possono estendersi per più righe

► Letterali

Sequenze di caratteri utilizzate all'interno dei programmi per rappresentare valori di tipi primitivi e stringhe

► Commenti

► `/*...*/`

Il compilatore ignora tutto il testo compreso tra questi caratteri; possono estendersi per più righe

► `/**...*/`

Il compilatore ignora tutto il testo compreso tra questi caratteri; possono estendersi per più righe. Sono chiamati *commenti di documentazione*, vengono interpretati da javadoc

► Letterali

Sequenze di caratteri utilizzate all'interno dei programmi per rappresentare valori di tipi primitivi e stringhe

► Commenti

► `/*...*/`

Il compilatore ignora tutto il testo compreso tra questi caratteri; possono estendersi per più righe

► `/**...*/`

Il compilatore ignora tutto il testo compreso tra questi caratteri; possono estendersi per più righe. Sono chiamati *commenti di documentazione*, vengono interpretati da javadoc

► *commenti a fine riga*

Si aprono con la coppia di caratteri `//` e si chiudono alla fine della riga; il compilatore ignora il testo che inizia dai caratteri `//` fino alla fine della riga

Il primo programma Java

BuonInizio.java

```
/* Il nostro primo programma */
class BuonInizio {

    // il metodo main
    public static void main(String [] args) {
        System.out.println("Ti auguro una buona giornata!");
    }
}
```