# DAGLI ALGORITMI AI LINGUAGGI

## Linguaggi di Programmazione

E` una notazione con cui e` possibile descrivere gli algoritmi.

**Programma:**

e` la rappresentazione di un algoritmo in un particolare linguaggio di programmazione.

In generale, ogni linguaggio di programmazione dispone di un insieme di "**parole chiave**" (keywords), attraverso le quali e` possibile esprimere il flusso di azioni descritto dall'algoritmo.

Ogni linguaggio e` caratterizzato da una **sintassi** e da una **semantica**:

- **sintassi**: e' l'insieme di regole formali per la composizione di programmi nel linguaggio scelto. Le regole sintattiche dettano le modalita` di combinazione tra le parole chiave del linguaggio, per costruire correttamente istruzioni (**frasi**).

- **semantica**: e` l'insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio scelto.

# Linguaggi di programmazione

| | |
|---|---|
| **1930-40** | Macchina di Turing Diagrammi di flusso (Von Neumann) |
| **1940-50** | Linguaggi macchina ed ASSEMBLER |
| **1950-55** | FORTRAN (Backus, IBM) |
| **1959** | COBOL |
| **1950-60** | LISP (McCarthy) |
| **1960-65** | ALGOL'60 (blocco, stack) |
| **1965** | PL/I |
| **1966** | SIMULA (tipo di dato astratto, classe) |
| **1970-71** | PASCAL (Wirth) |
| **1973** | PROLOG (Kowalski - Colmerauer) |
| **1975** | SETL |
| **1980** | SMALLTALK (oggetti) |
| **1983** | C++ (C con oggetti) |
| **1988** | Visual Basic |
| **1994** | Java |

# Il linguaggio macchina

☞ Il linguaggio macchina e` direttamente eseguibile dall'elaboratore, senza nessuna traduzione.

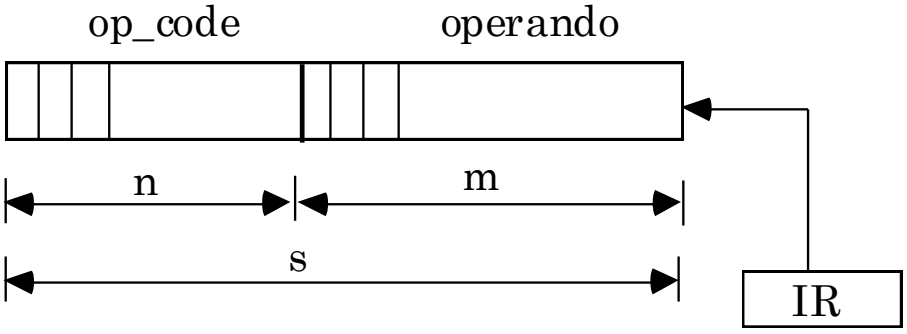
**Istruzioni:**

Si dividono in due parti: un **codice operativo** ed, eventualmente, uno o piu' **operandi**:

- Il **codice operativo** specifica l'operazione da compiere
- gli **operandi** individuano le celle di memoria a cui si riferiscono le operazioni.

Per semplicita`, consideriamo istruzioni **ad un solo operando**.

☞ istruzioni ed operandi relativi al programma in esecuzione sono caricati in memoria e quindi sono memorizzati in forma binaria.




**s**, lunghezza di un'istruzione (in bit)

$$s = n + m$$

# Il linguaggio ASSEMBLER

E' difficile leggere e capire un programma scritto in forma binaria:

| | | | |
|---|---|---|---|
| 0 0100 | 0000 | 0000 | 1000 |
| 1 0100 | 0000 | 0000 | 1001 |
| 2 0000 | 0000 | 0000 | 1000 |
| 3 0001 | 0000 | 0000 | 1001 |
| 4 1000 | 0000 | 0000 | 0000 |
| 5 0010 | 0000 | 0000 | 1000 |
| 6 0101 | 0000 | 0000 | 1000 |
| 7 1101 | 0000 | 0000 | 0000 |
| 8 0000 | 0000 | 0000 | 0000 |
| 9 0000 | 0000 | 0000 | 0000 |

**Linguaggi assemblatori (Assembler):**

- Le istruzioni corrispondono univocamente a quelle macchina, ma vengono espresse tramite nomi simbolici (parole chiave).

- I riferimenti alle celle di memoria sono fatti mediante nomi simbolici (identificatori).

- Identificatori che rappresentano **dati** (costanti o variabili) oppure istruzioni (**etichette**).

☞ Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina (**assemblatore**).

# ESEMPIO LINGUAGGIO ASSEMBLER

MACCHINA:

| | | | | |
|---|---|---|---|---|
| 0 | 0100 | 0000 | 0000 | 1000 |
| 1 | 0100 | 0000 | 0000 | 1001 |
| 2 | 0000 | 0000 | 0000 | 1000 |
| 3 | 0001 | 0000 | 0000 | 1001 |
| 4 | 1000 | 0000 | 0000 | 0000 |

ASSEMBLER:

| | | | | |
|---|---|---|---|---|
| 0 | ADD | X | Y | Z |
| 1 | STO | Z | | |
| 2 | MUL | Z | 36 | W |
| 3 | STO | W | | |
| 4 | JMP | 1 | | |

# Linguaggi di programmazione di alto livello

## Linguaggio Macchina:

- conoscenza dei metodi di rappresentazione delle informazioni utilizzati.

## Linguaggio Macchina ed Assembler:

- necessita` di conoscere dettagliatamente le caratteristiche della macchina (registri, dimensioni dati, set di istruzioni)
- semplici algoritmi implicano la specifica di molte istruzioni

## Linguaggi di Alto Livello:

Il programmatore puo` astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico.

☞ Sono indipendenti dalla macchina (astrazione).

## Esecuzione:

Sono tradotti in sequenze di istruzioni di basso livello, direttamente eseguite dal processore, attraverso:

- interpretazione (ad es. BASIC)
- compilazione (ad es. C, FORTRAN, Pascal)

# ESEMPIO LINGUAGGI DI ALTO LIVELLO:

## Linguaggio C:

```
#include <stdio.h>
#define dim 10

main()
{int V[dim], i,j, max, tmp, quanti;

/* lettura dei dati */
for (i=0; i<dim; i++)
  { printf("valore n. %d: ",i);
    scanf("%d", &V[i]);
   }
/*ordinamento */

for(i=0; i<dim; i++)
  {quanti=dim-i;
  max=quanti-1;
  for( j=0; j<quanti; j++)
    { if (V[j]>V[max])

  max=j;
    }
  if (max<quanti-1)
    { /*scambio */
    tmp=V[quanti-1];
    V[quanti-1]=V[max];
    V[max]=tmp;
    }
  }
/*stampa */
for(i=0; i<dim; i++)
  printf("Valore di V[%d]=%d\n", i, V[i]);
}
```

# Ambiente di programmazione

È l'insieme dei programmi che consentono la scrittura e la verifica di nuovi programmi (fasi di sviluppo).

## Sviluppo di un Programma

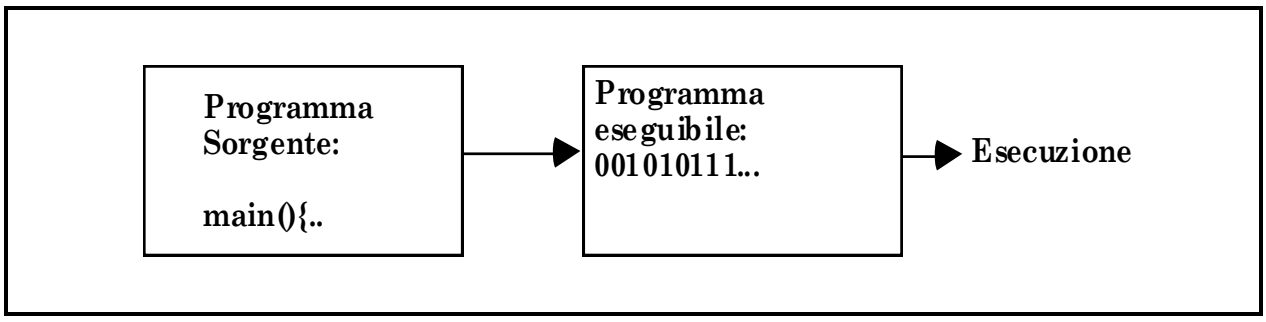
- Affinchè un programma scritto in un qualsiasi linguaggio di programmazione (ad es. il C) sia comprensibile (e quindi eseguibile) da un calcolatore, è necessaria un'azione di traduzione dal linguaggio originario al linguaggio macchina

- Questa operazione viene normalmente svolta da speciali programmi, detti **traduttori**.

| **Programma** | **Traduzione** |
|---|---|
| `main()` | |
| `{ int A;` | `00100101` |
| `...` | |
| `A=A+1;` | `11001..` |
| `if....` | `1011100..` |

I traduttori provvedono a convertire il codice di programmi scritti in un particolare linguaggio di programmazione (programmi *sorgenti*), nella corrispondente rappresentazione in linguaggio macchina (programmi *eseguibili*).

# Sviluppo di Programmi




**Due categorie di traduttori, a secondo del linguaggio di programmazione:**

- i **Compilatori**: accettano in ingresso l'intero programma e producono in uscita la rappresentazione dell'intero programma in linguaggio macchina.
- gli **Interpreti**: traducono ed eseguono direttamente ciascuna istruzione del *programma sorgente*, istruzione per istruzione.

**Quindi:**

- **Compilatore**: per ogni programma da tradurre, lo schema viene percorso una volta sola prima dell'esecuzione.
- **Interprete**: lo schema viene attraversato tante volte quante sono le istruzioni che compongono il programma; a ogni attivazione dell'interprete su una particolare istruzione, segue l'esecuzione dell'istruzione stessa.

☞ l'esecuzione di un programma compilato è più veloce dell'esecuzione di un programma mediante interprete.

# Ambiente di programmazione: componenti

- **Editor:** serve per la costruzione di file che contengono testi (cioè sequenze di caratteri). In particolare tramite un editor si scrive il *programma sorgente*.

- **Compilatore:** opera la traduzione di un programma sorgente scritto in un linguaggio ad alto livello in un *programma oggetto* scritto in un linguaggio direttamente eseguibile dal calcolatore.

- **Linker:** (collegatore o correlatore) nel caso in cui il programma sia suddiviso in moduli (*oggetto*) separati compilati separatamente provvede a collegarli per formare un unico *programma eseguibile.*

- **Debugger:** (scopritore di bachi, cioè errori) serve per scoprire ed eliminare errori presenti durante l'esecuzione di un programma, ma non rilevati in fase di compilazione.

In un ambiente dove l'esecuzione avviene tramite interpretazione i programmi non vengono compilati. In questo caso è presente un interprete.

- **Interprete:** traduce ed esegue direttamente ciascuna istruzione del *programma sorgente*, istruzione per istruzione. È alternativo al compilatore (non sono sempre presenti entrambi).