Generazioni dei linguaggi di programmazione

I generazione (anni ’50):

il **linguaggio macchina** formato da sequenze di **numeri binari** che corrisponde al livello più basso di descrizione[1] di un programma.

II generazione (anni ’60):

il **linguaggio assemblativo** o **ASSEMBLY** dove ancora esiste una **corrispondenza biunivoca** tra istruzioni in linguaggio macchina (**numeri binari**) e istruzioni in linguaggio ASSEMBLY (**insiemi di caratteri**).
Ad esempio, se l’istruzione in linguaggio macchina **11010001** ha il significato di “SOMMA”, una possibile istruzione in linguaggio ASSEMBLY potrebbe essere **ADD**.
Tale linguaggio è detto ***mnemonico*** proprio perché ogni istruzione è formata da una sequenza di caratteri che *ricordano* il tipo di operazione da eseguire ed è necessario un programma (**assemblatore** o **ASSEMBLER**) che lo ***traduca*** nel linguaggio comprensibile da una calcolatore (codice esadecimale decodificato all’interno della specifica[2] CPU).

III generazione (a partire dagli anni ’70):

i **linguaggi procedurali** a medio-alto livello (come il **C**), dove il controllo del sistema diventa sempre più logico e meno meccanico (si accentua il distacco dall’hardware) con evoluzione verso linguaggi ad alto livello dove è possibile descrivere nuovi tipi di dati adatti alle proprie esigenze e manipolarli più agevolmente; appartengono a quest’ultima categoria linguaggi come **Java** (dal ‘95), il **C++** (‘98) o **C#** (nel 2001).

IV generazione (a partire dagli anni ’80):

i **linguaggi** per realizzare **ambienti di sviluppo** (linguaggi **macro** in ambienti Word o Excel**,** linguaggi di interrogazione nella gestione di Data Base come **SQL** …)

V generazione (a partire dagli anni ’90):

i **linguaggi** sviluppati in progetti di **Intelligenza Artificiale** (già predecessori linguaggi funzionali quale il **Lisp** nel **‘58**, linguaggi logici quali il **Prolog** nel **70**) in cui si sono incorporate alcune tecniche di ***ragionamento deduttivo*** ed appartengono ad un livello gerarchicamente ancora più elevato di descrizione di programmi sempre più distante dall’hardware.

---

[1] In realtà all’interno della CPU un *circuito decodificatore* permette di usare il *codice esadecimale* (ogni 4 bit sono espressi con un'unica cifra espressa con un codice a base 16 che usa i valori tipici della codifica decimale da 0 a 9 ed i caratteri da A ad F. Ad esempio il numero in codice binario **1100** corrisponde a 12 in codice decimale ed a **C** in esadecimale)

[2] Ad esempio un’operazione di **assegnamento** che copi nel *registro Accumulatore* (zona di memoria interna alla CPU dove vengono memorizzati i risultati delle operazioni aritmetico-logiche) un *dato* si descriverà col codice mnemonico LD A, *dato* nel caso di CPU Z80 (dove il registro *Accumulatore* può contenere solo 1 byte) corrispondente al codice esadecimale 3E *dato* mentre nel caso di CPU 80x86 si potrà descrivere con MOV al, *dato* (codice esadecimale B0 *dato*) oppure MOV ah, *dato* (codice esadecimale B4 *dato*) o ancora MOV ax, *dato* (codice esadecimale B8 *dato*)

SCHEMA EVOLUTIVO DELLE GENERAZIONI DEI LINGUAGGI DI PROGRAMMAZIONE

1GL linguaggi di Prima generazione

Basati sul codice macchina (in **binario**)

```
11100101
11110000
11001101
```

2GL linguaggi di Seconda generazione

Simbolici di tipo **Assembly**

Esempi: IBM BAL, VAX Macro.

```
calc: decL R5
      pushL R5
       calls #1, recfib
       movL R0, R6
       decL R5
        pushL R5
       calls #1, recfib
       addL R6, R0
        ret
.end
```

3GL linguaggi di Terza generazione

Linguaggi ***procedurali***, con termini in lingua inglese.I programmatori devono specificare il "cosa" e "come" desiderano mandare in output.

Esempi: COBOL, RPG, FORTRAN, Pascal, Ada, **C**, BASIC, PL/I.

4GL linguaggi di Quarta generazione

Non procedurali, ma ancora basati sulla lingua inglese. Contengono dizionari di dati integrati, database relazionali dinamici, consentono rapidi sviluppi del sistema tramite anche prototipizzazioni. I programmatori (e a volte gli utenti) specificano il "cosa" dell'output desiderato, lasciando al software il compito di stabilire il "come".

Esempi: FOCUS, Powerhouse

```
access admissions
set report device printer
set report device disc
set report name AD32
select if term = "961" and (app-status="AC" or app-status="PP")
sort on ad-source on name
report id name ad-source class hs-code entrance-code app-status
footing at ad-source skip 2 "Total for: " ad-source "=" count skip 3
final footing "Total enrollment " count
set report nolimit
go
```

?GL linguaggi della prossima generazione?

L'evoluzione dei linguaggi di programmazione e la disponibilità di tools di sviluppo non finiscono certo con la 4^ generazione. Si è detto che la programmazione Object-Oriented rappresenta una nuova generazione di strumenti.
Lo sviluppo O-O si basa sull'uso di "oggetti" che rappresentano sia la struttura dei dati che dei processi che possono agire su di essi. La maggior valenza degli oggetti deriva dal fatto che una volta definiti sono riutilizzabili infinite volte, riducendo drasticamente lo sforzo di programmazione.
Successive generazioni di programmi potranno apparire sempre meno vincolati ad un linguaggio di tipo tradizionale, con le sue regole grammaticali e la sua sintassi.

Ricky Spelta http://www.windoweb.it/edpstory_new/el_programmi.htm