Sommario

UNIVERSITÀ DI PARMA

- Piccolo riepilogo
- Il linguaggio macchina
- Assembly
- Linguaggi di Alto Livello
- Linguaggi Interpretati e Compilati

Riepilogo: Unità di Controllo

UNIVERSITÀ DI PARMA

- Dove si trova il programma da eseguire?
  - In memoria ⟶ “numeri”
- La control unit esegue i programmi ovvero:
  - Legge da memoria la relativa istruzione da eseguire ⟶ **Fetch**
  - Decodifica il suo significato ovvero cosa deve fare ⟶ **Decode**
  - Esegue quanto richiesto pilotando gli altri sistemi ⟶ **Execute**
  - Scrive eventuali risultati elaborazione ⟶ **Store**
  - Ripete…

Riepilogo: Esempio Istruzione

UNIVERSITÀ DI PARMA

- Come è fatta una istruzione decodificata dall'unità di controllo?
- Tipicamente piú "parti"
  - Operation code → mi dice che istruzione è
  - Operando/i → mi fa capire su cosa si opera
- Instruction Set → dipende da processore!
- Esempio: ADD X, Y
  - Somma il contenuto della memoria all'indirizzo X a quello in Y e scrivi il risultato in X

Opcode (8 bit)	Indirizzo X (16 bit)	Indirizzo Y (16 bit)
00001001	0000000001100011	0000000001100100

Linguaggio Macchina

UNIVERSITÀ DI PARMA

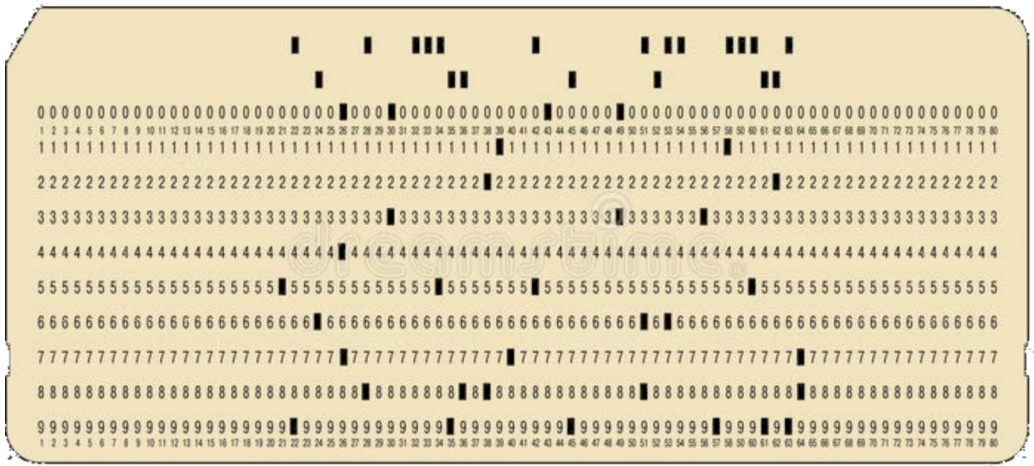
- Codice numerico che corrisponde alle possibili istruzioni eseguibili da una CPU
- Diverse per ogni tipo di processore
  - Instruction Set
- Ad esempio l'instruction set dei processori ARM è completamente differente da quello dei processori Intel x86
  - Differenti modelli della stessa famiglia di processori presentano Instruction Set con variazioni

6502 Machine Language Hex Codes

	Immediate	Accumulator	Zero Page	Zero Page,X	Zero Page,Y	Absolute	Absolute,X	Absolute,Y	Indirect	(Zero Page,X)	(Zero Page),Y	Implied	Relative	Affects Flags
Add with Carry	69		65	75		6D	7D	79		61	71			N,V,Z,C
Bitwise AND with Accumulator	29		25	35		2D	3D	39		21	31			N,Z
Arithmetic Shift Left		0A	06	16		0E	1E							N,Z,C
Branch on Carry Clear													90	
Branch on Carry Set													B0	
Branch on Equal													F0	
Test Bits			24			2C								N,V,Z
Branch on Minus													30	
Branch on Not Equal													D0	
Branch on Plus													10	
Break												00		B
Branch on Overflow Clear													50	
Branch on Overflow Set													70	
Clear Carry												18		C
Clear Decimal												D8		D
Clear Interrupt												58		I
Clear Overflow												B8		V
Compare Accumulator	C9		C5	D5		CD	DD	D9		C1	D1			N,Z,C
Compare X Register	E0		E4			EC								N,Z,C
Compare Y Register	C0		C4			CC								N,Z,C
Decrement Memory			C6	D6		CE	DE							N,Z
Decrement X												CA		N,Z
Decrement Y												88		N,Z
Bitwise Exclusive OR	49		45	55		4D	5D	59		41	51			N,Z
Increment Memory			E6	F6		EE	FE							N,Z
Increment X												E8		N,Z
Increment Y												C8		N,Z
Jump						4C			6C					
Jump to Subroutine						20								
Load Accumulator	A9		A5	B5		AD	BD	B9		A1	B1			N,Z
Load X Register	A2		A6	B6		AE		BE						N,Z
Load Y Register	A0		A4	B4		AC	BC							N,Z
Logical Shift Right		4A	46	56		4E	5E							N,Z,C
No Operation												EA		
Bitwise OR with Accumulator	09		05	15		0D	1D	19		01	11			N,Z
Push Accumulator to Stack												48		
Push Processor Status to Stack												08		
Pull Accumulator off Stack												68		
Pull Processor Status off Stack												28		All
Rotate Left		2A	26	36		2E	3E							N,Z,C
Rotate Right		6A	66	76		6E	7E							N,Z,C
Return from Interrupt												40		All
Return from Subroutine												60		
Subtract with Carry	E9		E5	F5		ED	FD	F9		E1	F1			N,V,Z,C
Set Carry												38		C
Set Decimal												F8		D
Set Interrupt												78		I
Store Accumulator			85	95		8D	9D	99		81	91			
Store X Register			86		96	8E								
Store Y Register			84	94		8C								
Transfer A to X												AA		N,Z
Transfer A to Y												A8		N,Z
Transfer Stack Pointer to X												BA		
Transfer X to A												8A		N,Z
Transfer X to Stack Pointer												9A		
Transfer Y to A												98		N,Z

Linguaggio Macchina

- Non mnemonico ⟶ difficoltà di utilizzo
- Non portabile
  - Codice sviluppato per una famiglia di CPU non utilizzabile altrove
- All'atto pratico non usato direttamente



- Sostituzione codice numerico con simboli mnemonici
- ~1:1

```
START: MOV   AX,  BX
       CMP   AX,  12h
       JZ    EQUAL
       INT   21h
       RET
EQUAL: MOV   BL,  82h
```

Assembly

UNIVERSITÀ DI PARMA

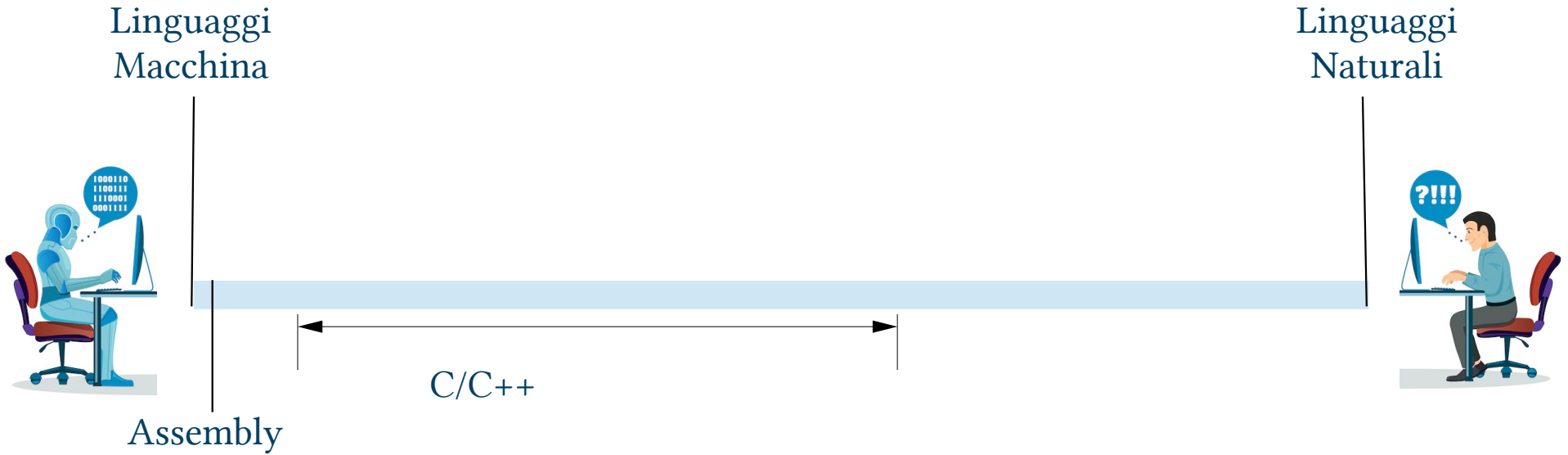
- Molto piú facile del codice macchina!
  - Sia da scrivere che da leggere
- Ma devo comunque ottenere il codice macchina
- Assembler: programma che legge le istruzioni assembly e le traduce in istruzioni macchina
  - Quasi traduzione 1 a 1

- Comunque linguaggio complesso
  - Molte istruzioni anche per operazioni semplici
- Non portabile

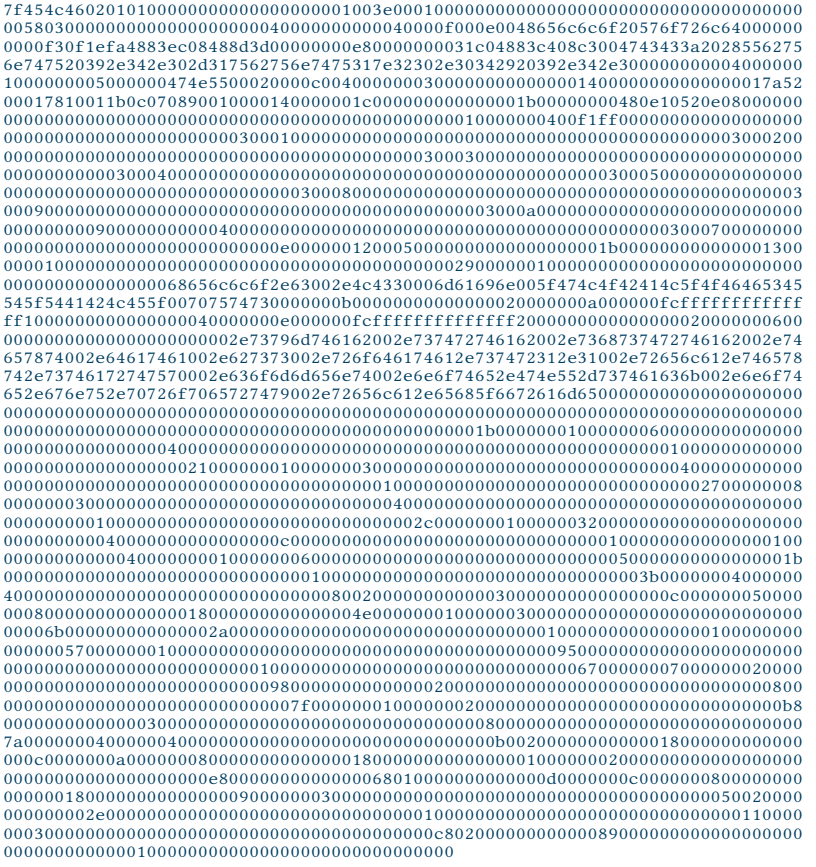
Linguaggi ad Alto Livello

UNIVERSITÀ DI PARMA

- Non piú corrispondenza 1:1 con istruzioni linguaggio macchina
  - 1 istruzione può corrispondere a decine di istruzioni linguaggio macchina



Codice Macchina



```
7f454c4602010100000000000000000001003e0001000000000000000000000000000000000000
0058030000000000000000000040000000000040000f000e0048656c6c6f20576f726c64000000
0000f30f1efa4883ec08488d3d00000000e80000000031c04883c408c3004743433a2028556275
6e747520392e342e302d317562756e7475317e32302e30342920392e342e300000000004000000
1000000005000000474e5500020000c0040000000300000000000000140000000000000017a52
00017810011b0c0708900100001400000001c00000000000001b00000000480e10520e08000000
0000000000000000000000000000000000000000000000010000000400f1ff000000000000000000
0000000000000000000000030001000000000000000000000000000000000000000003000200
0000000000000000000000000000000000000000030003000000000000000000000000000000
0000000000300040000000000000000000000000000000000000000000003000500000000000000
0000000000000000000000000000300080000000000000000000000000000000000000000003
0009000000000000000000000000000000000000000000003000a000000000000000000000000
0000000009000000000040000000000000000000000000000000000000000003000700000000
0000000000000000000000000e0000001200050000000000000000001b0000000000000001300
0000100000000000000000000000000000000000000029000000100000000000000000000000
0000000000000000068656c6c6f2e63002e4c4330006d61696e005f474c4f42414c5f4f46465345
545f5441424c455f00707574730000000b0000000000000002000000a000000fcfffffffffffff
ff1000000000000000040000000e000000fcfffffffffffff2000000000000000020000000600
0000000000000000000002e73796d746162002e737472746162002e736873747274616200 2e74
657874002e64617461002e627373002e726f646174612e737472312e31002e72656c612e746578
742e737461727475700002e636f6d6d656e74002e6e6f74652e474e552d737461636b002e6e6f74
652e676e752e70726f7065727479002e72656c612e65685f6672616d650000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000001b0000000100000006000000000000000
0000000000000004000000000000000000000000000000000000000000000001000000000000
0000000000000000210000001000000030000000000000000000000000000400000000000
0000000000000000000000000000000000000001000000000000000000000000000002700000008
0000000300000000000000000000000000000400000000000000000000000000000000000000
0000000010000000000000000000000000000002c000000010000003200000000000000000000
0000000004000000000000000c000000000000000000000000000000010000000000000000100
0000000000040000000010000000600000000000000000000000000000500000000000000001b
0000000000000000000000000000010000000000000000000000000000003b00000004000000
400000000000000000000000000000080020000000000030000000000000000c000000050000
0008000000000000001800000000000004e0000000100000300000000000000000000000000000
00006b00000000000002a000000000000000000000000000000010000000000000010000000
0000005700000001000000000000000000000000000000000000095000000000000000000000
0000000000000000000000010000000000000000000000000000067000000700000020000
0000000000000000000000009800000000000002000000000000000000000000000000000800
0000000000000000000000000007f000000010000000200000000000000000000000000000b8
0000000000000030000000000000000000000000000000800000000000000000000000000000
7a000000040000004000000000000000000000000000000b00200000000000180000000000000
000c0000000a000000080000000000000180000000000000010000000200000000000000000
0000000000000000000e800000000000006801000000000000d0000000c0000000800000000
0000001800000000000000900000003000000000000000000000000000000000000050020000
000000002e000000000000000000000000000001000000000000000000000000000110000
0003000000000000000000000000000000000000c802000000000008900000000000000000
0000000000001000000000000000000000000000
```

Assembly

```
        .file   "hello.asm"
        .text
        .section        .rodata
.LC0:
        .string "Hello World"
        .text
        .globl  hello
        .type   hello, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        leaq    .LC0(%rip), %rdi
        call    puts@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
```

FORTRAN

```
program hello
  print *, 'Hello, World!'
end program hello
```

Elenco (ampiamente incompleto!)

UNIVERSITÀ DI PARMA

- FORTRAN (1954)
- LISP (1958)
- ALGOL (1958) & ALGOL68 (1968)
- PASCAL (1970)
- C (1969)
- C++ (1979)
- JAVA (1991)
- Python (1991)

Caratteristiche Linguaggi ad Alto Livello

UNIVERSITÀ DI PARMA

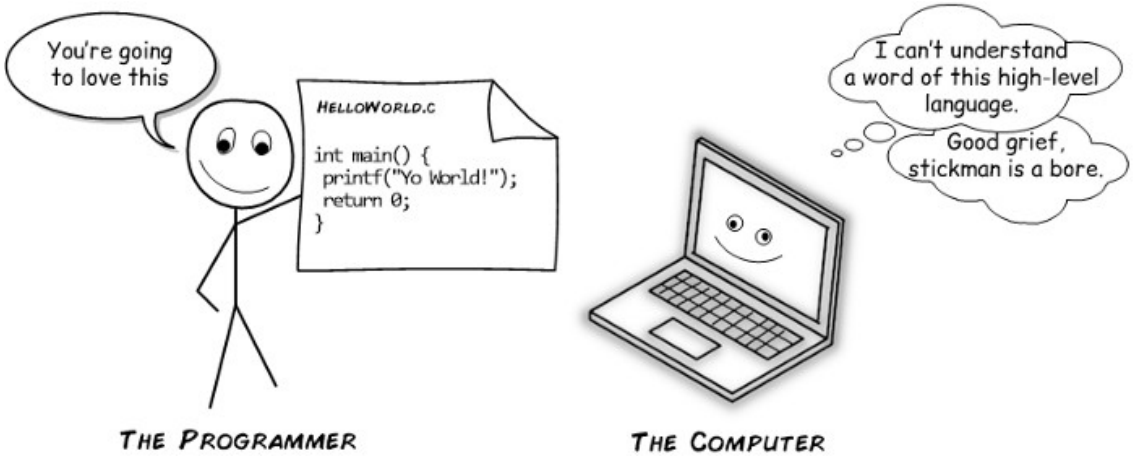
- Pro
  - Mascherano il calcolatore
  - Maggiore leggibilità e comprensibilità
  - Portabilità

- Contro
  - Necessità di traduzione

Linguaggi interpretati e compilati

UNIVERSITÀ DI PARMA

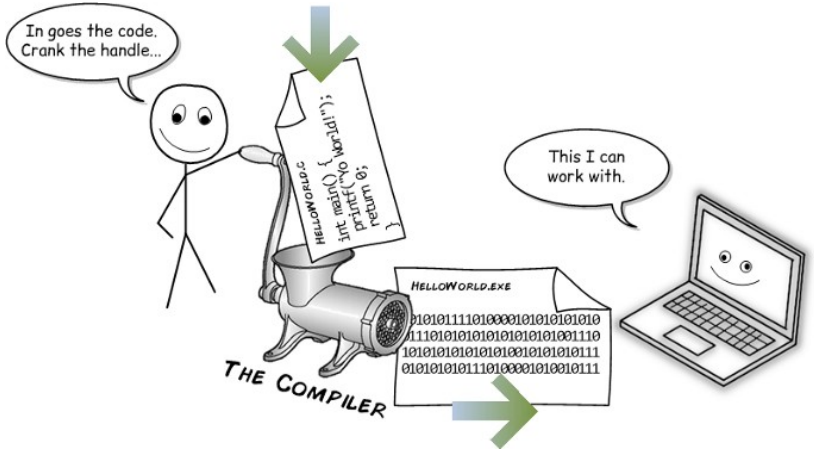
- I linguaggi ad alto livello richiedono una traduzione piú complessa
  - Da codice sorgente a linguaggio macchina
- Due possibilità
  - Compilazione
  - Interpretazione



Compiled Language

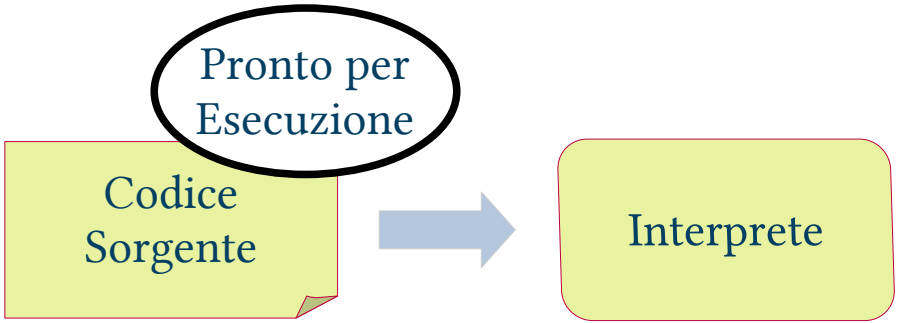
UNIVERSITÀ DI PARMA

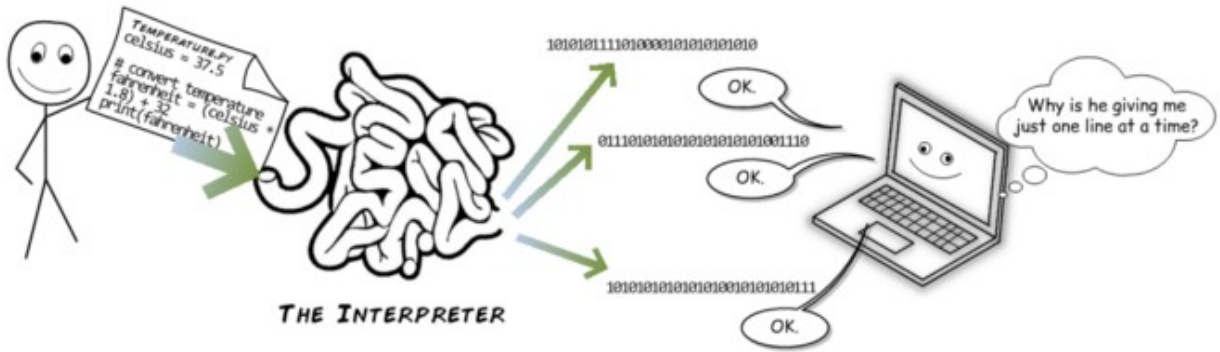
- Traduzione in codice macchina necessaria
- Compilatore ⟶ il “traduttore”
- Produce un “file eseguibile”



Interpreted Language

- Il codice sorgente non deve essere tradotto
- Interprete che lo "legge" al volo
  - Macchina virtuale





Compiled vs Interpreted

UNIVERSITÀ DI PARMA

- Compilazione
  - Piú veloce
  - Permette progetti multifile con piú linguaggi
  - Posso nascondere codice sorgente
- Interpretazione
  - Esecuzione immediata
    - Scripting
  - Multipiattaforma
    - Web e altro

Perché così tanti linguaggi?

UNIVERSITÀ DI PARMA

- Ricerca costante di sistemi migliori per aggiornamento tecnologia
- Non solo “General Purpose” ma problemi specifici
  - Web $\longrightarrow$ Javascript, PHP, GO...
  - Mobile $\longrightarrow$ JAVA
  - Firmware $\longrightarrow$ C
  - Sistemi real time $\longrightarrow$ C++
- Performance vs Rapidità di sviluppo
  - GO/C $\longleftrightarrow$ Javascript/Python

Elementi del linguaggio

UNIVERSITÀ DI PARMA

- Lessico
  - Parole chiave: istruzioni, operatori, simboli…
- Sintassi
  - Come si possono combinare i precedenti elementi
    - Parsing
  - Si può distinguere tra
    - Sintassi necessaria e
    - *Zucchero Sintattico*

Errore di Compilazione o Interpretazione

- Semantica
  - Significato di quanto scritto

Errore di Esecuzione

Riepilogo

UNIVERSITÀ DI PARMA

- Computer: sistema che esegue una serie di istruzioni in "sequenza" ⟶ automa
- Programmare:
  - Individuare algoritmo risolutivo del problema dato
    - Passi e loro ordine di esecuzione
  - Tradurre l'algoritmo individuato in codice
    - Lessico, Sintassi e Semantica specifici del linguaggio di programmazione