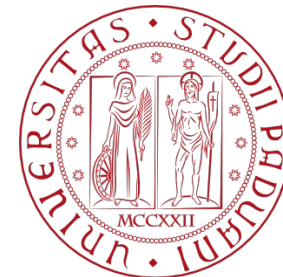


Programmazione

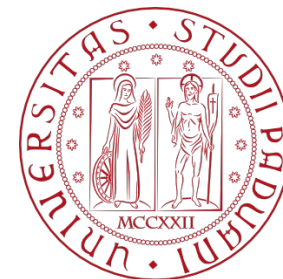
Giovanni Da San Martino

Dipartimento of Matematica, Università degli Studi di Padova
giovanni.dasanmartino@unipd.it
A.A. 2023/2024



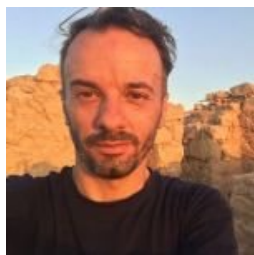
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Introduzione



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Docenti:



- Giovanni Da San Martino
(responsabile) - 40h
giovanni.dasanmartino@unipd.it



- Francesco Gavazzo -40h
francesco.gavazzo@unipd.it



- Lamberto Ballan -24h
lamberto.ballan@unipd.it

- Assistanti alla didattica:

- Elisa Sartori, Luca Parolari, Pietro Ferrazzi

- Tutor

- Giulio Umbrella, Giacomo Stevanato, Alessio Ferrarini

- Moodle del corso:

<https://stem.elearning.unipd.it/course/view.php?id=6982>

- Useremo un secondo Moodle per gli esercizi di programmazione
 - trovate l'indirizzo nel Moodle sopra (in generale ogni informazione sarà disponibile attraverso il Moodle del corso)

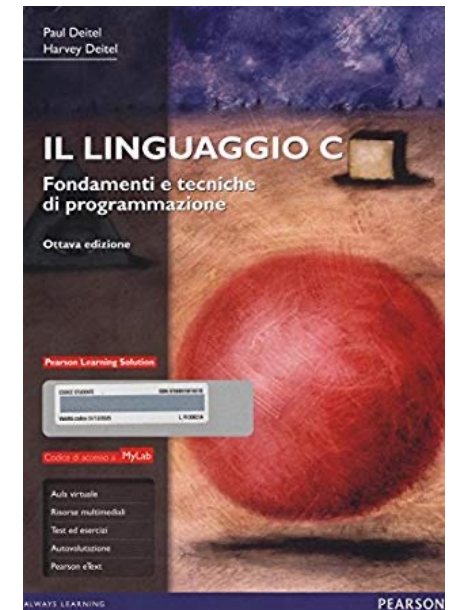
Iscrivetevi ad entrambi!

- Controllare di avere un account per i laboratori funzionante
- Ci sarà un terzo Moodle per gli esami



- Kernighan, Brian W. "Il linguaggio C : principi di programmazione e manuale di riferimento". 2. ed, Pearson Prentice Hall, 2004."

- "Il linguaggio C", Paul Deitel and Harvey Deitel (nona edizione)



- Insieme [redacted] di istruzioni [redacted]
[redacted] per risolvere un problema.

- Insieme ordinato e finito di istruzioni elementari e non ambigue, per risolvere un problema.
- Il concetto di algoritmo è generale (non ci sono riferimenti al calcolatore)

- 3 elementi
 - a) Il problema da risolvere
 - b) La sequenza di istruzioni
 - c) La soluzione del problema
- 2 attori
 - chi crea le istruzioni
 - chi le esegue



- 3 elementi
 - a) Il problema da risolvere
 - b) La sequenza di istruzioni
 - c) La soluzione del problema
- 2 attori
 - chi crea le istruzioni
 - chi le esegue

PREPARAZIONE

COME PREPARARE GLI SPAGHETTI CACIO E PEPE



Per preparare gli spaghetti cacio e pepe, come prima cosa occupatevi di grattugiare 200 g di Pecorino. Proseguite mettendo a bollire l'acqua in un tegame (mettetene circa la metà di quanto ne usate di solito per cuocere la pasta, così sarà più ricca di amido) e quando bollerà potete salare a piacere. Una volta salata, potete cuocere gli spaghetti **1**. Nel frattempo, versate i grani di pepe interi su un tagliere **2**, quindi schiacciateli pestando con un pestello per carne o un macinino **3**. In questo modo si

- 3 elementi
 - a) Il problema da risolvere
 - b) La sequenza di istruzioni
 - c) La soluzione del problema
- 2 attori
 - chi crea le istruzioni
 - chi le esegue

PREPARAZIONE

COME PREPARARE GLI SPAGHETTI CACIO E PEPE



Per preparare gli spaghetti cacio e pepe, come prima cosa occupatevi di grattugiare 200 g di Pecorino. Proseguite mettendo a bollire l'acqua in un tegame (mettetene circa la metà di quanto ne usate di solito per cuocere la pasta, così sarà più ricca di amido) e quando bollirà potete salare a piacere. Una volta salata, potete cuocere gli spaghetti **1**. Nel frattempo, versate i grani di pepe interi su un tagliere **2**, quindi schiacciateli pestando con un pestello per carne o un macinino **3**. In questo modo si

- La risoluzione di un problema spesso implica la risoluzione di una serie di sottoproblemi
- problema e sottoproblema sono concetti interscambiabili
- chiameremo i
problemi/sottoproblemi
funzioni



- Insieme ordinato e finito di istruzioni **non ambigue**, per risolvere un problema.
- non ambigue per chi le esegue
- E qua sono esseri umani ad eseguire le istruzioni...

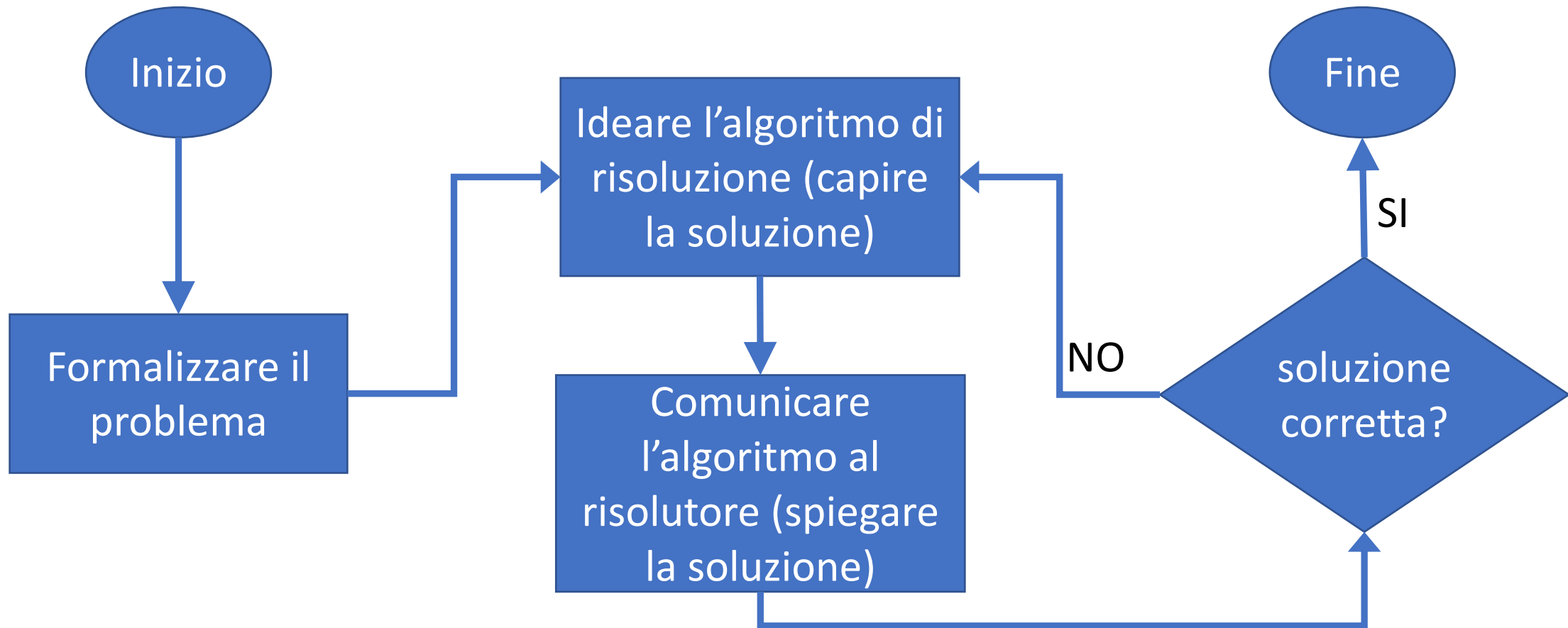


- Problema: fare un panino al burro di arachidi
 - Figli: creano le istruzioni
 - Padre: esegue le istruzioni
- I figli sanno come fare il panino, ma non riescono a comunicarlo in modo non ambiguo
- Il padre non si sforza per comprenderli

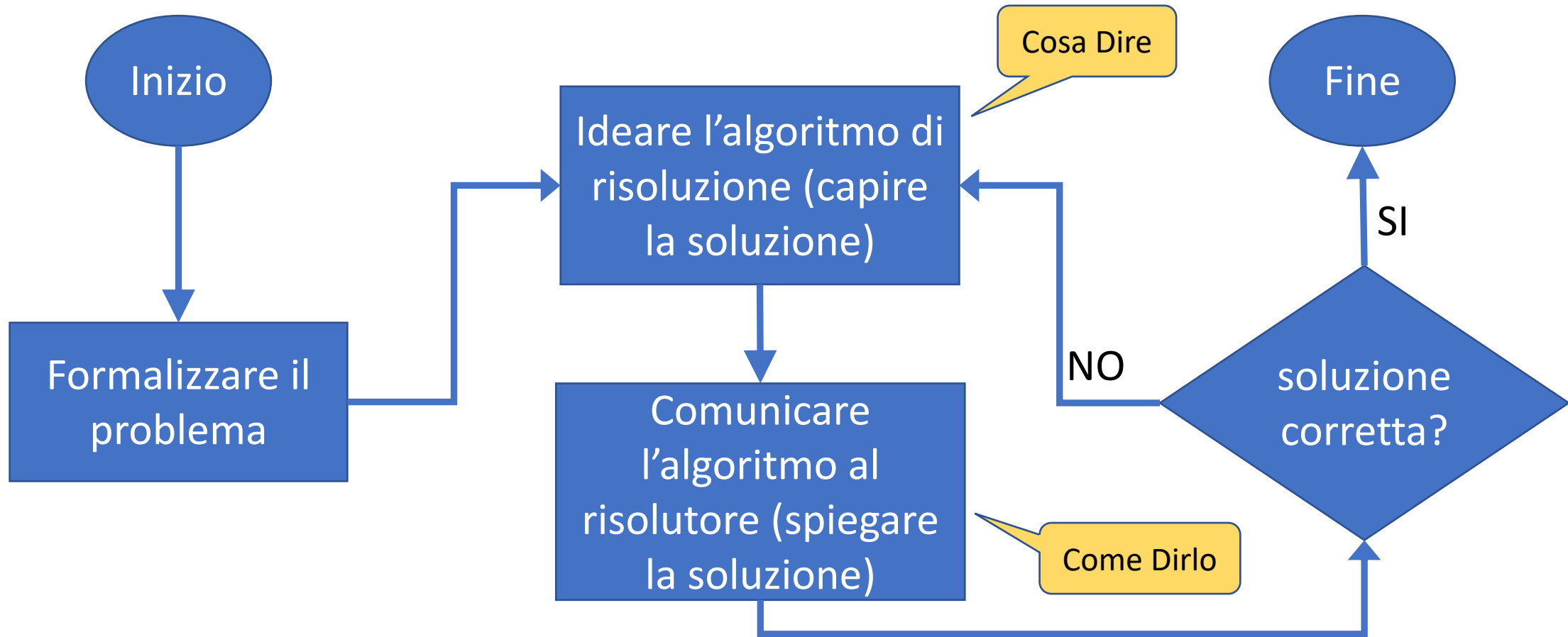


<https://www.youtube.com/watch?v=Ct-IOOUqmyY>

L' "Algoritmo" per Creare Algoritmi

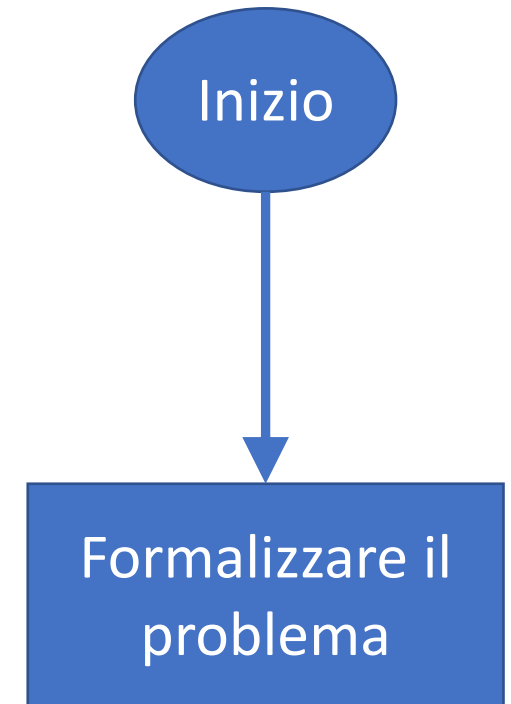


L' "Algoritmo" per Creare Algoritmi



"Coding is to programming what typing is to writing"
Leslie Lamport

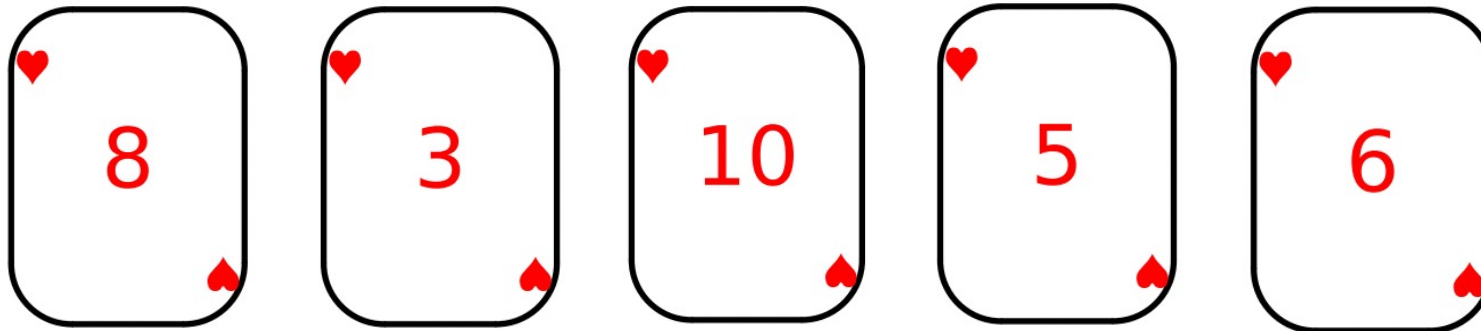
- Ovvero **descrivere Ingressi (Input) e Uscite (Output)**
- Input: come rappresentare i dati in ingresso, cosa si assume su di essi
 - Per quali input garantiamo di poter calcolare la soluzione
 - Useremo il termine **Precondizione** per le assunzioni sui dati
- Output: cosa calcola il nostro algoritmo
 - deve essere descritto in modo non ambiguo per chi utilizzerà il nostro algoritmo
 - in generale associamo una **Postcondizione**
 - un'asserzione (formula che può essere vera o falsa) che esprime cosa calcola un frammento di codice



Ideare l'algoritmo di risoluzione



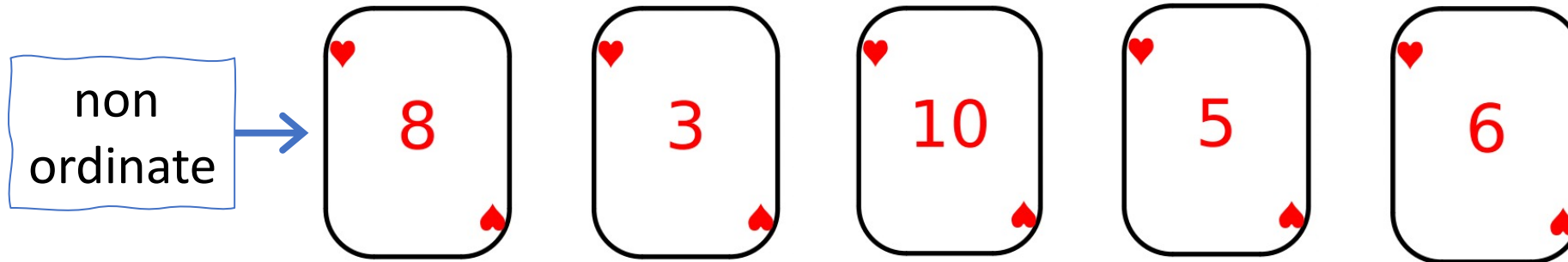
- Intuire la soluzione e comunicarla a me stesso (o ad un mio collega) in un linguaggio/formalismo a noi familiare – per esempio l'italiano
 - ci concentriamo sulla parte di ideazione della soluzione, siamo più liberali sulle operazioni elementari (purché comprensibili e meno ambigue possibile)
- Esempio: ordinare le carte



Ideare l'algoritmo di risoluzione



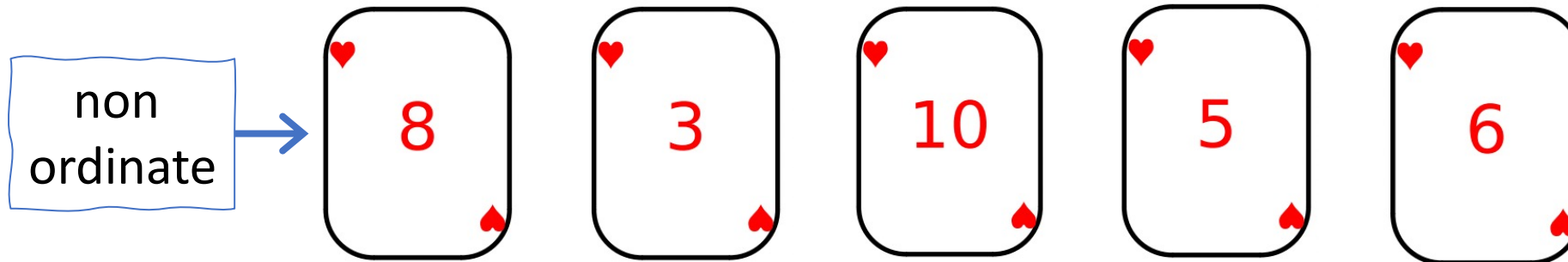
- Lista carte “non ordinate” e lista carte “ordinate”
- POST: la lista “ordinate” contiene le carte ordinate in modo crescente
- finché la lista “carte non ordinate” è non vuota
 1. selezionare la carta minima nella lista “carte non ordinate”
 2. spostarla alla fine della lista “carte ordinate”



Ideare l'algoritmo di risoluzione



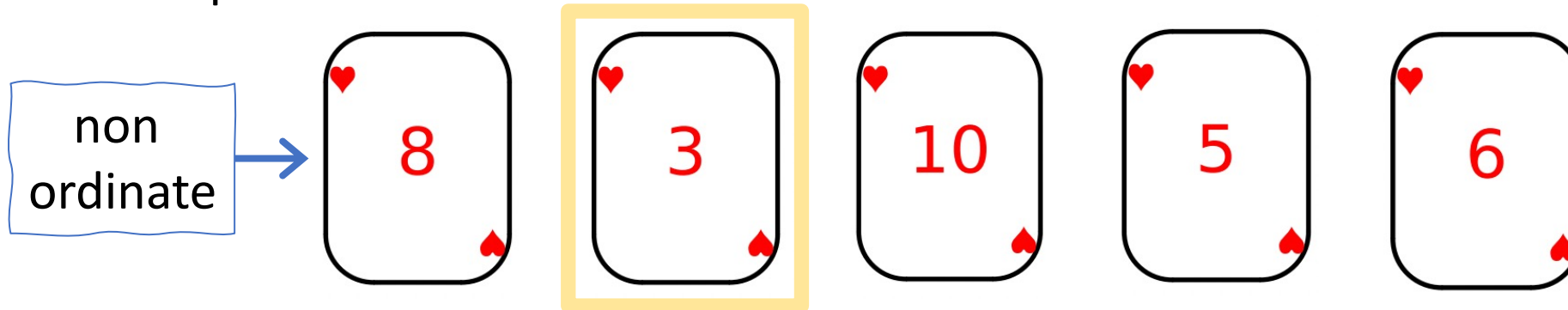
- Lista carte “non ordinate” e lista carte “ordinate”
- finché la lista “non ordinate” è non vuota
 1. selezionare la carta minima nella lista “carte non ordinate”
 2. spostarla alla fine della lista “carte ordinate”



Ideare l'algoritmo di risoluzione



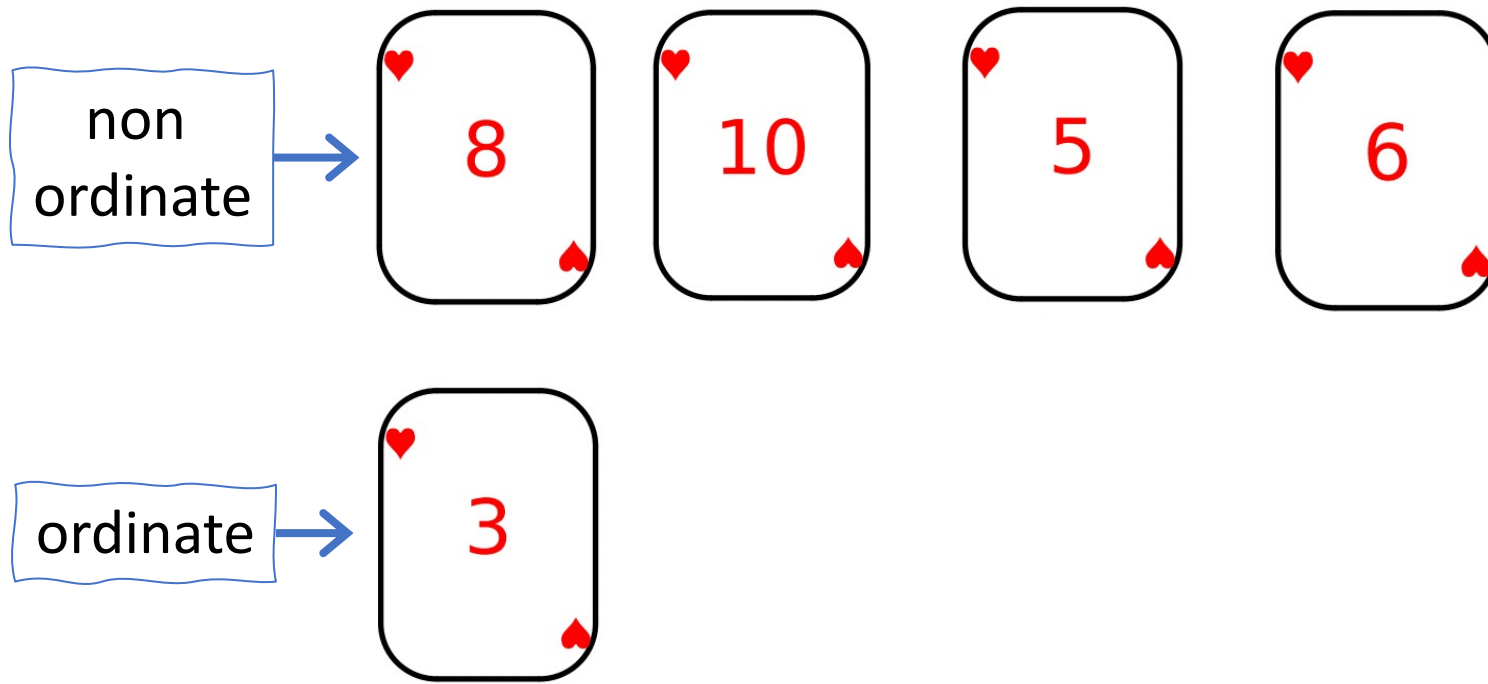
- Lista carte “non ordinate” e lista carte “ordinate”
- finché la lista “non ordinate” è non vuota
 1. **selezionare la carta minima nella lista “carte non ordinate”**
 2. spostarla alla fine della lista “carte ordinate”



Ideare l'algoritmo di risoluzione



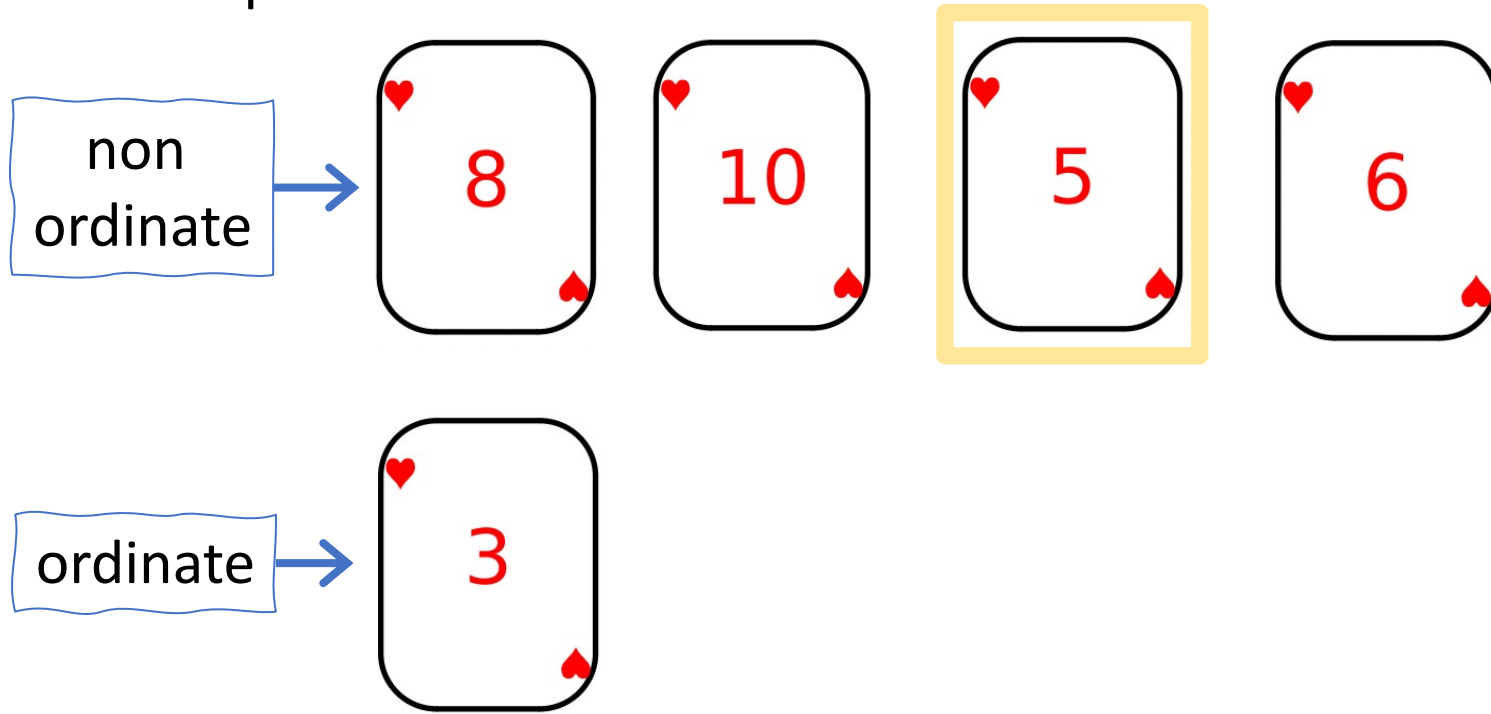
- Lista carte “non ordinate” e lista carte “ordinate”
- finché la lista “non ordinate” è non vuota
 1. selezionare la carta minima nella lista “carte non ordinate”
 2. **spostarla alla fine della lista “carte ordinate”**



Ideare l'algoritmo di risoluzione



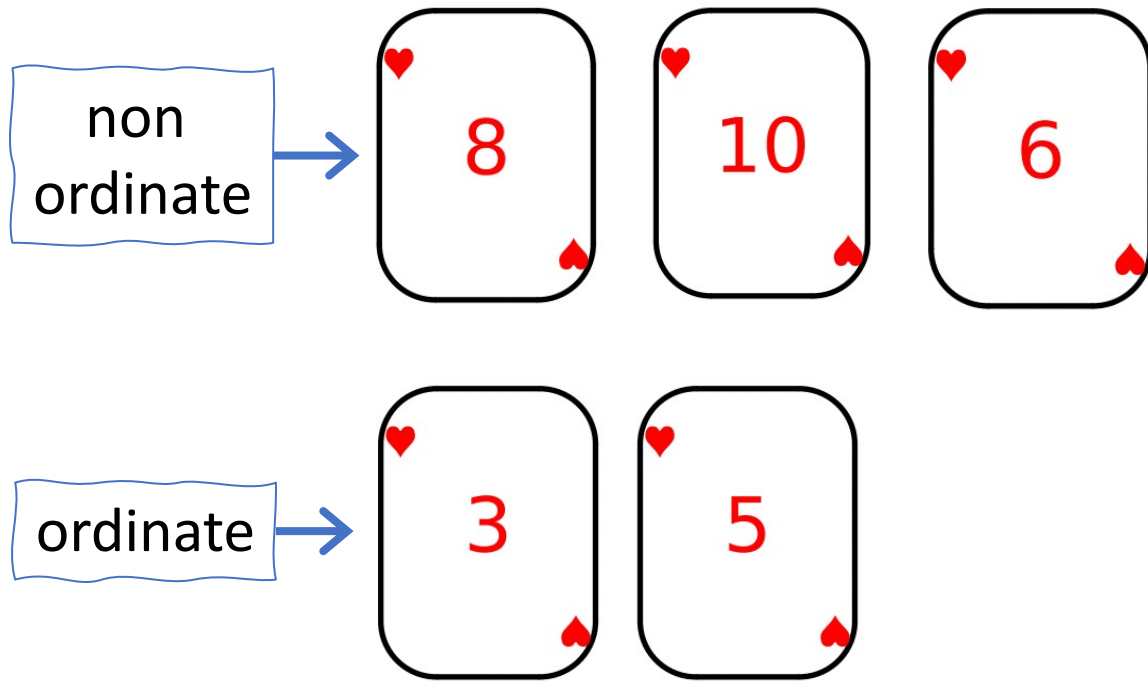
- Lista carte “non ordinate” e lista carte “ordinate”
- finché la lista “non ordinate” è non vuota
 1. **selezionare la carta minima nella lista “carte non ordinate”**
 2. spostarla alla fine della lista “carte ordinate”



Ideare l'algoritmo di risoluzione



- Lista carte “non ordinate” e lista carte “ordinate”
- finché la lista “non ordinate” è non vuota
 1. selezionare la carta minima nella lista “carte non ordinate”
 2. **spostarla alla fine della lista “carte ordinate”**

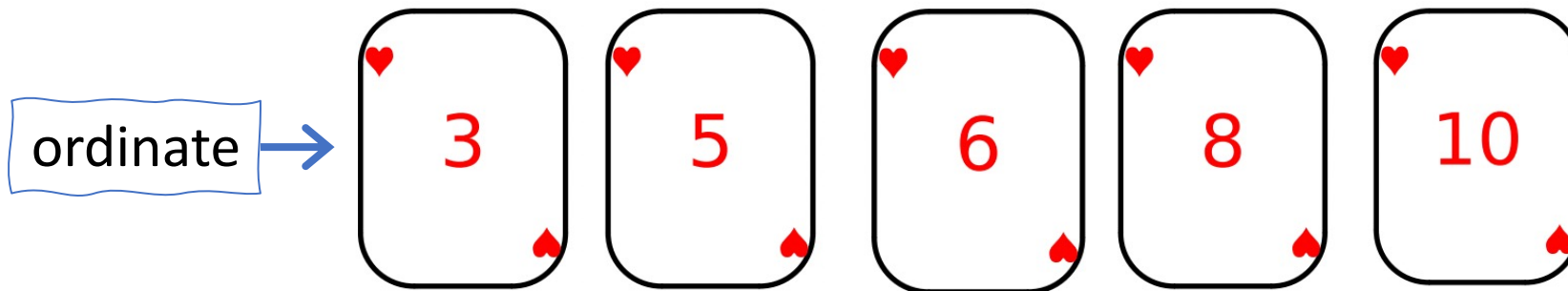


Ideare l'algoritmo di risoluzione



- Lista carte “non ordinate” e lista carte “ordinate”
- **finché la lista “non ordinate” è non vuota**
 1. selezionare la carta minima nella lista “carte non ordinate”
 2. spostarla alla fine della lista “carte ordinate”

non
ordinate →



↓

Ideare
l'algoritmo di
risoluzione

↓

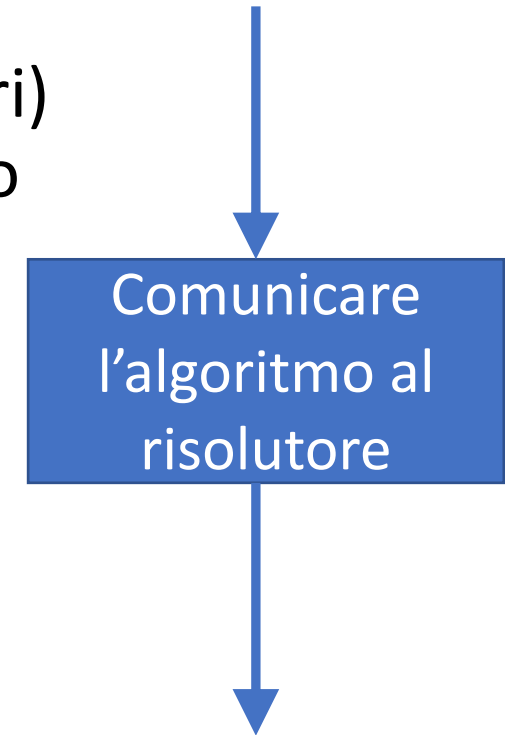
- In generale pensare un nuovo algoritmo è difficile
- A volte richiede creatività (è un'arte?) e conoscenze extra-informatiche
 - Difatti è un'abilità utile in varie discipline, non solo per informatici
- In questo corso
 - Vedremo schemi di risoluzione di problemi
 - non vi è richiesto di inventare nuovi algoritmi
 - Ma dovrete capirli, ovvero essere in grado di riapplicare schemi di risoluzione a problemi simili



Comunicare l'Algoritmo al Risolutore



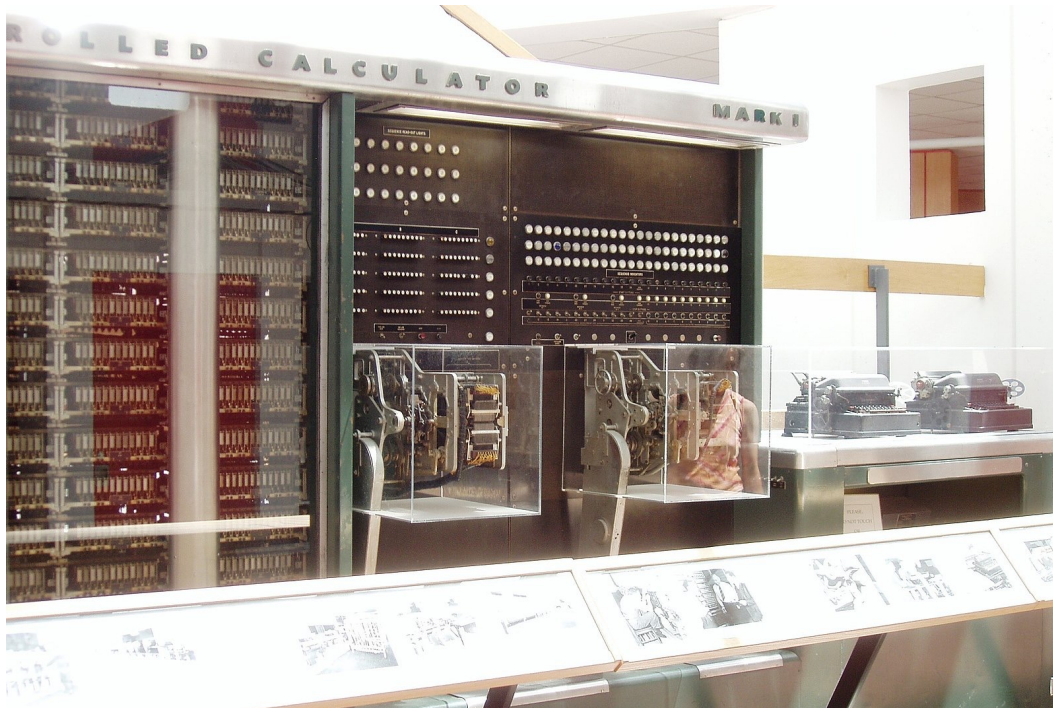
- Programmare: comunicare algoritmi al calcolatore
- Linguaggio di Programmazione: insieme di istruzioni (elementari) che possono essere eseguite dal calcolatore (e regole per la loro composizione)
- Implementare un algoritmo:
 1. Conoscere le istruzioni elementari messe a disposizione dal linguaggio di programmazione
 2. Esprimere la soluzione al passo precedente (in italiano) nel linguaggio di programmazione
 - se la distanza dei 2 linguaggi è ampia, può avere elementi di difficoltà paragonabili al passo precedente, si impareranno schemi implementativi
 - eseguendo i due passi separatamente teniamo sotto controllo la difficoltà



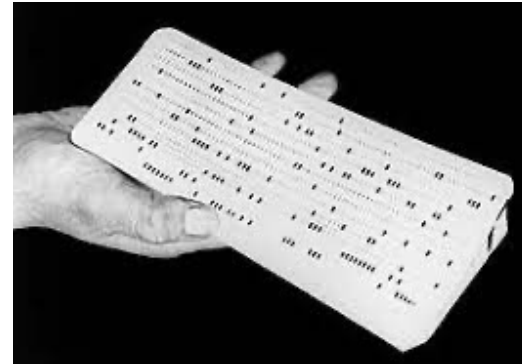
Linguaggi di basso livello (linguaggio macchina)

- istruzioni sono veramente elementari, molto lontane dal linguaggio parlato, quindi è più difficile pensare algoritmi complessi direttamente nel linguaggio macchina

Comunicare
l'algoritmo al
risolutore

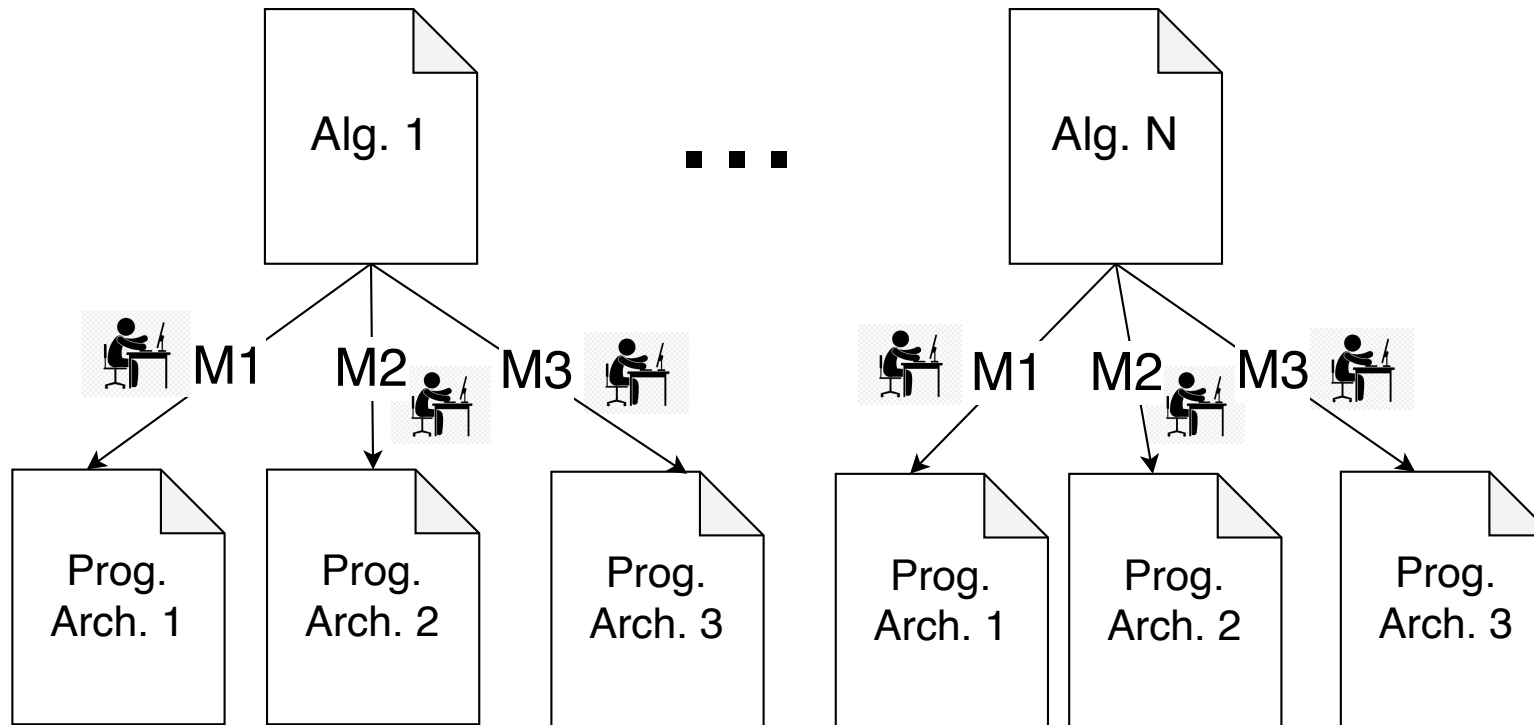


Harvard Mark I

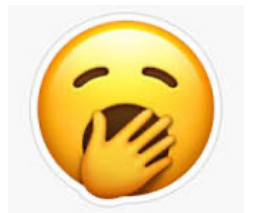


Linguaggi di basso livello

- dipendono dall'architettura



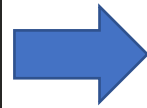
- Per ogni programma che invento (potenzialmente molto lungo), devo implementare una versione per ogni architettura





- Linguaggi di alto livello: alcune nuove istruzioni vengono implementate nel linguaggio macchina
 - l'utente esprime un programma tramite queste nuove istruzioni

```
1 #include <stdio.h>
2
3 int main (void) {
4     printf ("Hello, World!\n");
5
6     return 0;
7 }
```



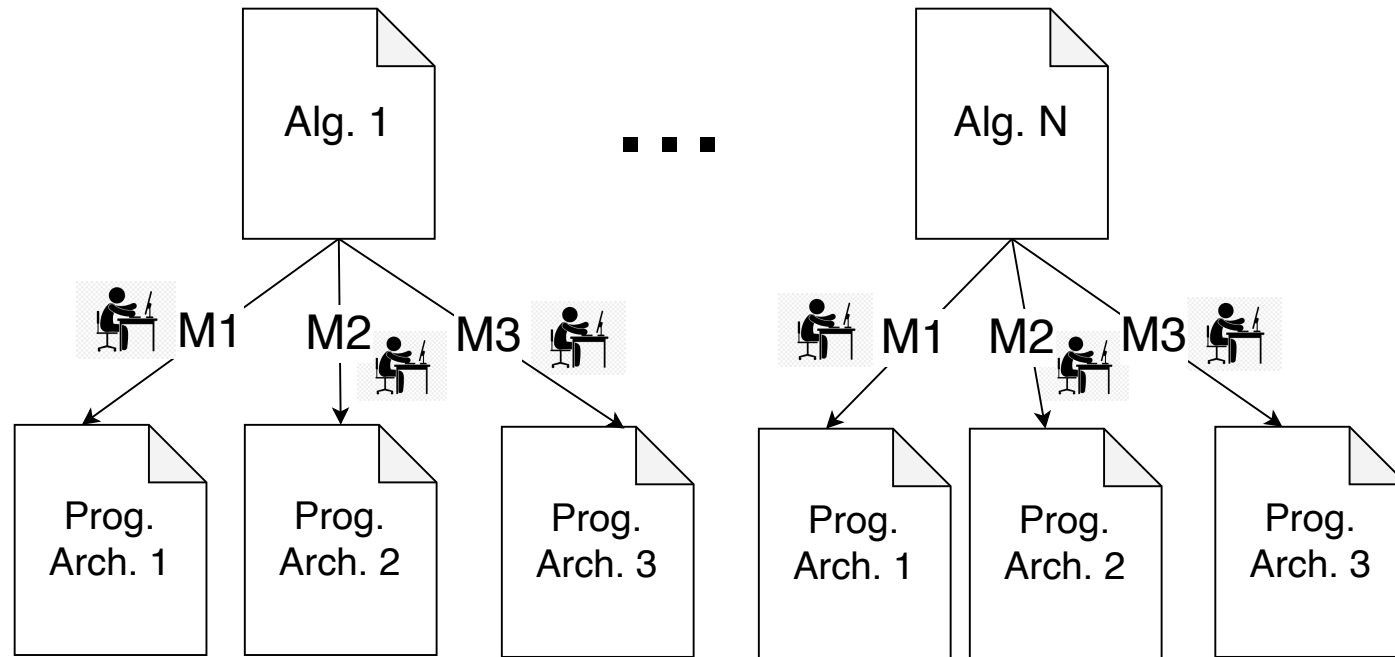
Traduttore
Architettura x



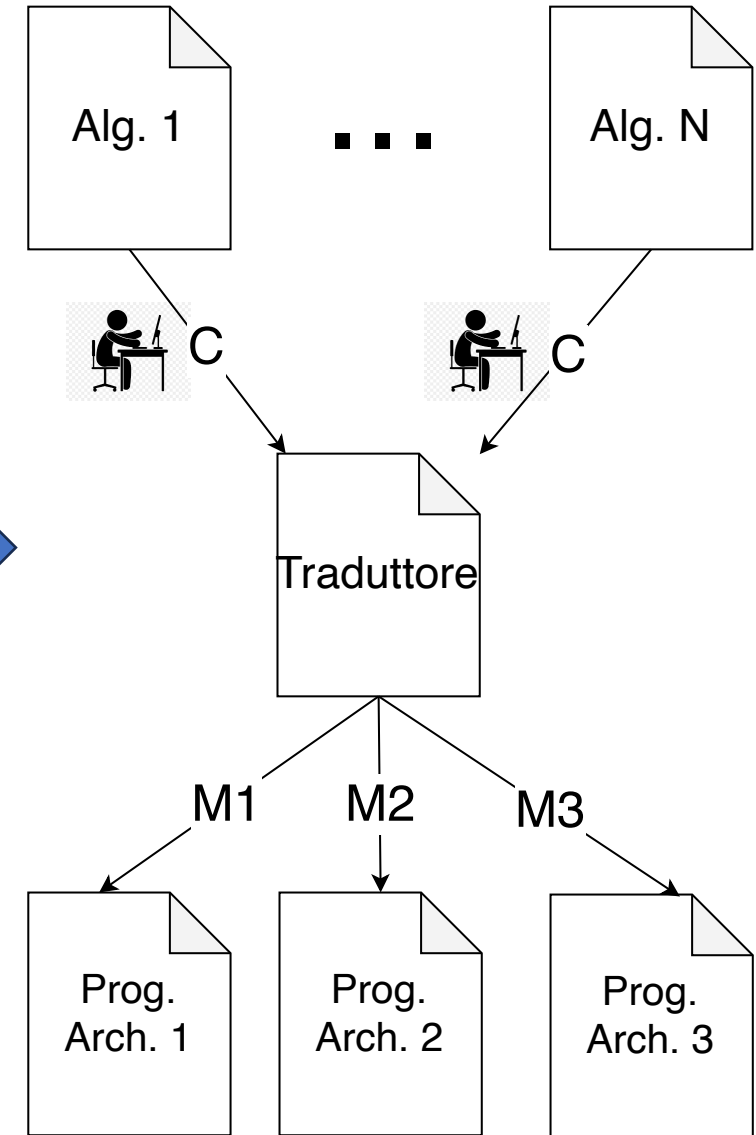
Comunicare
l'algoritmo al
risolutore



- le nuove istruzioni vengono tradotte automaticamente nel linguaggio macchina, tramite un programma chiamato traduttore (compilatore)



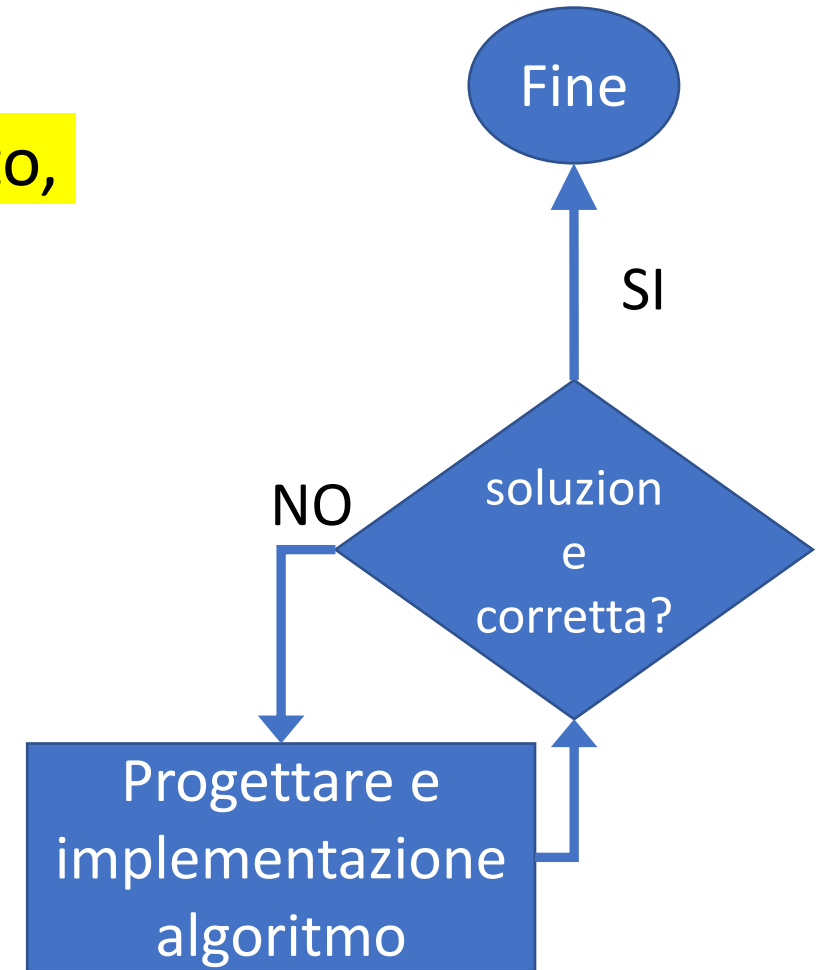
- Se cambia l'architettura, qualcuno deve solamente fornire un traduttore per essa (gli algoritmi non necessitano di interventi, dobbiamo solo riapplicare il traduttore ad essi)



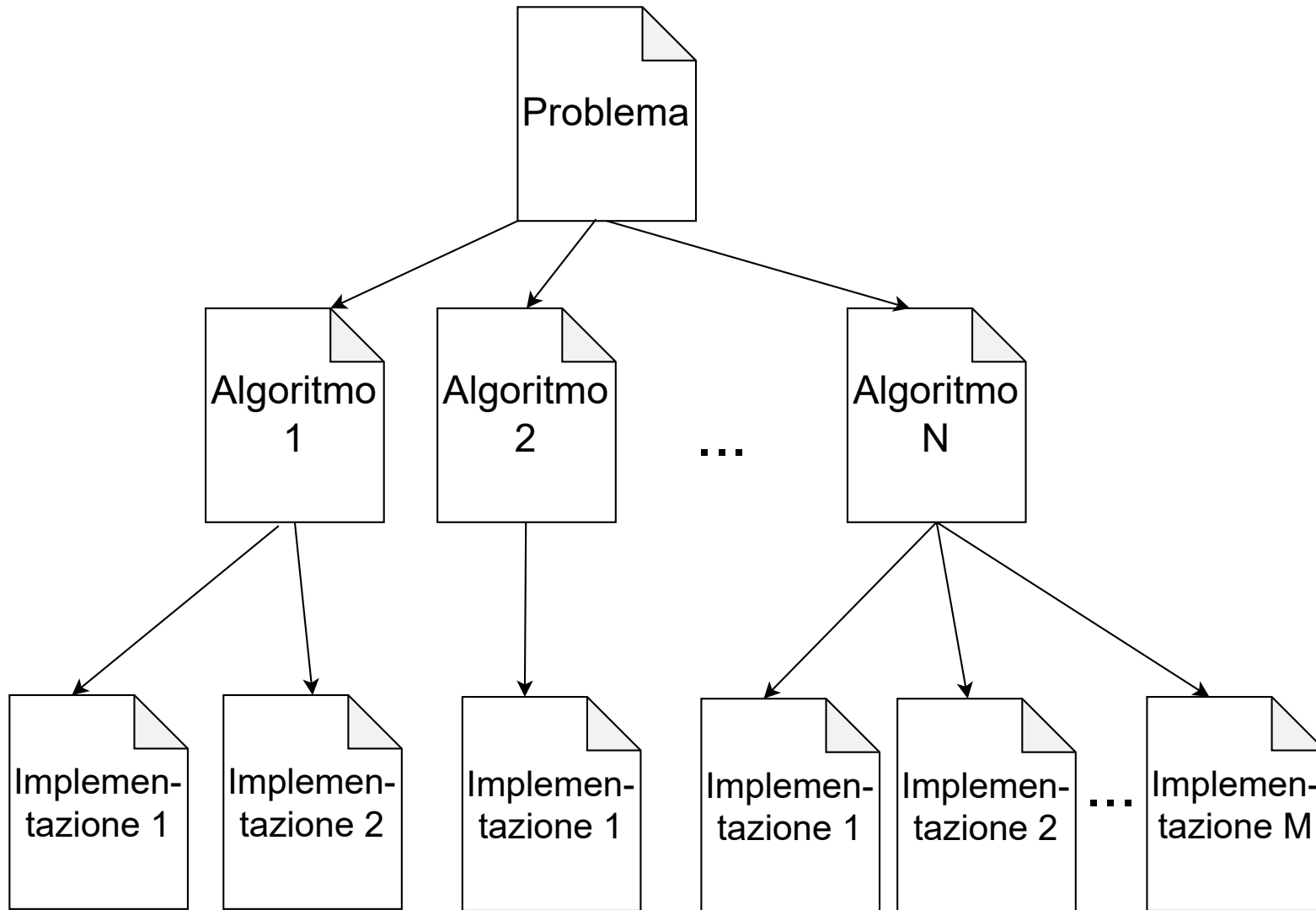
- Se cambia architettura, dobbiamo solamente fornire un traduttore per essa (e ritradurre i nostri programmi)
- Desiderata per un linguaggio:
 - Bassa complessità del traduttore \leftarrow il numero di nuove istruzioni implementate nel linguaggio di basso livello è ridotto
 - Potenza del linguaggio \leftarrow si creano ulteriori “istruzioni” (funzioni) direttamente nel nuovo linguaggio
 - tali istruzioni sono raccolte in librerie
 - conoscerle permette di risparmiare tempo e non “reinventare la ruota”
- Esempi di linguaggi di alto livello: C, C++, Java, Python

- In generale i linguaggi ad alto livello si bilanciano tra due obiettivi contrastanti:
 - far fare più controlli al linguaggio di programmazione per evitare errori all'utente (motto: non fidarsi del programmatore)
 - rimanere efficienti
- C: sviluppato nel 1970 da Ken Thompson e Dennis Ritchie
 - poca memoria disponibile nei calcolatori, nell'ordine dei Kb → efficienza come obiettivo principale
 - è piuttosto vicino ai linguaggi di basso livello (alcuni principi, per esempio la programmazione ad oggetti, verranno sviluppati solo in seguito) → buono come primo linguaggio da studiare
- Le specifiche del linguaggio vengono pubblicate nel 1978
 - molto popolare, vengono creati molti compilatori, anche con comportamenti diversi
 - viene creato uno standard, ANSI C, aggiornato fino ad oggi.

- Dopo che si è implementato l'algoritmo, si deve fornire evidenza che il nostro programma sia corretto, ovvero realizzi la consegna (o meglio la postcondizione)
- Il tipo di prova dipende dal contesto
 - test (controllare che, per certi input otteniamo l'output desiderato)
 - prove di correttezza
- Se il programma non è corretto, si analizza il funzionamento tramite il debugger



I Programmi Non Sono Tutti Uguali



- Come valutare diversi algoritmi e diverse implementazioni?

I Programmi non sono tutti uguali

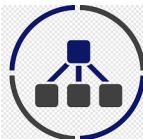


- Criterio più importante: Correttezza



A parità di correttezza, come si può migliorare un programma?

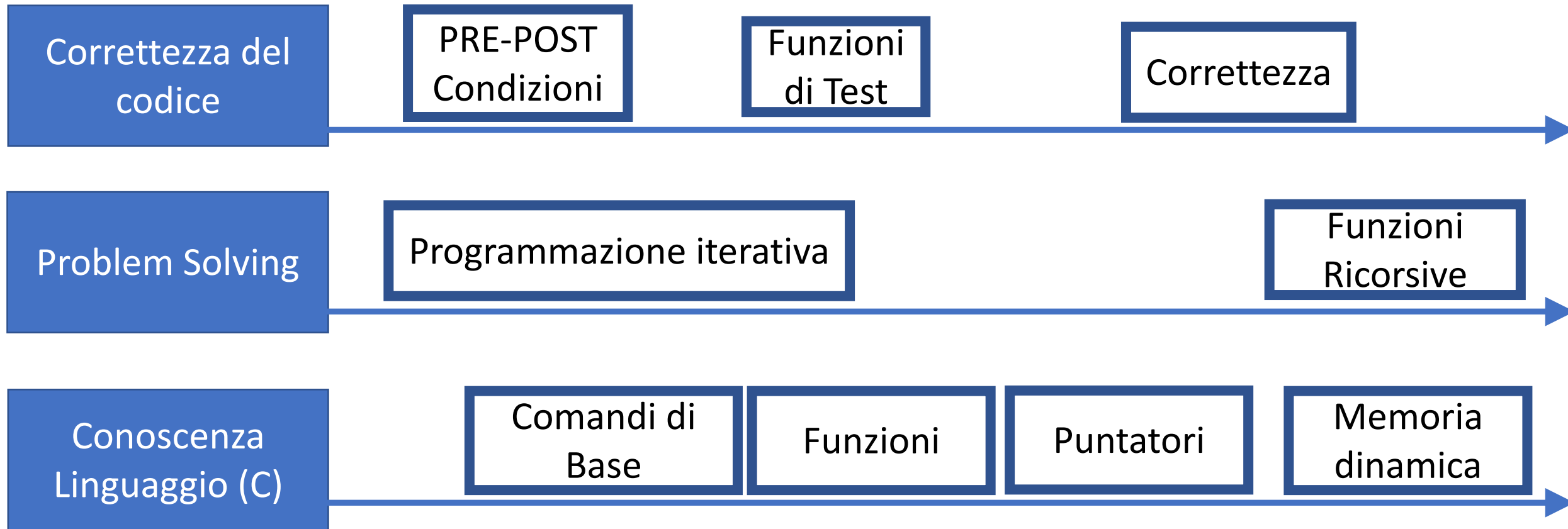
1. Efficienza (tempo e spazio), sia a livello algoritmico che implementativo
 - Il secondo aspetto è preponderante nel corso, ma tratteremo entrambi
2. Organizzazione: si codifica (per bene) una volta, poi si riusa il codice
 - Si divide il problema in sottoproblemi e si cerca di riusare le soluzioni già implementate in passato
3. Stile
 - Il codice deve essere comprensibile ai colleghi e a noi stessi tra 3 mesi.



[Dettagli su Moodle]

- A conoscere la semantica dei comandi e la struttura tipica di un linguaggio di programmazione di alto livello
- A non rimanere inermi davanti ad un errore e a fornire evidenza della correttezza del nostro codice.
- Non è richiesto di inventare nuovi algoritmi per avere un buon voto, ma di imparare gli schemi di programmazione in modo da saperli riapplicare a problemi simili.

Macroargomenti di Valutazione



- Discuteremo anche di efficienza, organizzazione e stile del codice.
- Lezioni frontali e in laboratorio (mediamente una a settimana)

- Generalmente ogni argomento interagisce con i successivi, complicandosi a posteriori - ovvero mettendo in risalto nuove problematiche
- I concetti del linguaggio e gli schemi di risoluzione richiedono tempo per essere assimilati. Gli esercizi sono parte integrante del corso (a volte necessari per capire le ramificazioni della teoria)
- Lezioni di laboratorio: 2 canali con docenti diversi (canale pari e dispari)
- Esame: ci saranno due compitini (il primo verso la metà del semestre)
 - Prove di esame: quiz con domande sui temi del corso e prove di programmazione
 - i codici che non compilano non verranno corretti
 - L'esame sarà effettuato al calcolatore
 - le richieste per ufficio inclusione devono essere fatte per tempo