[ITADINFO]

2° CONVEGNO ITALIANO
SULLA DIDATTICA DELL'INFORMATICA

Diagrammi di flusso a supporto dell'apprendimento della programmazione

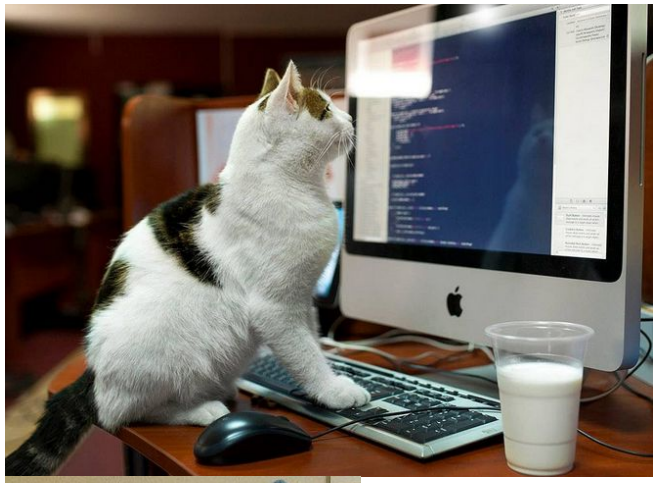
Gabriele Pozzan
Tullio Vardanega

20 Ottobre 2024, Genova

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Motivazione - Imparare a programmare (1)



Imparare a programmare è difficile?

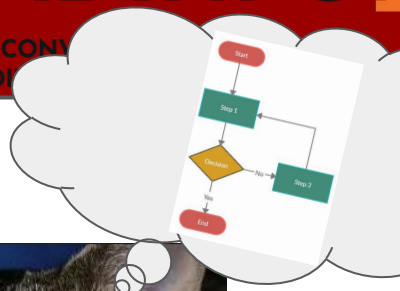
- A livello **aneddotico**: sì
- Concretamente? Uno studio del 2014*, basato su dati relativi a 161 corsi *CS1* (51 istituti, 15 paesi) riporta un tasso di superamento del **67.7%**

Spesso è stata riportata una **discrepanza** tra le **aspettative** di chi insegna e le competenze di chi impara (un compendio si trova in una tesi di dottorato del 2012**)

C'è chi dice*** che siano proprio le **aspettative** ad essere **troppo alte**.

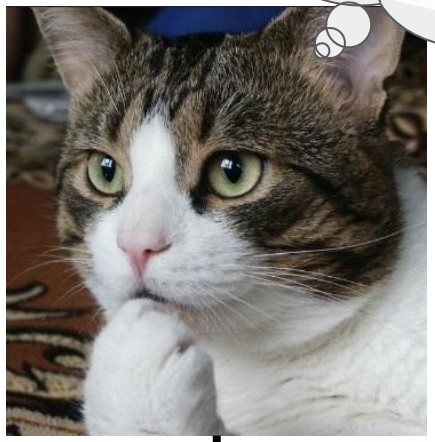
* Watson, Christopher, and Frederick WB Li. "**Failure rates in introductory programming revisited.**" *Proceedings of the 2014 conference on Innovation & technology in computer science education*. 2014.
** Sorva, Juha. ***Visual program simulation in introductory programming education.*** Aalto University, 2012. (Capitolo 3 "Students Worldwide Do Not Learn to Program")
*** Luxton-Reilly, Andrew. "**Learning to program is easy.**" *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 2016.

Motivazione - Imparare a programmare (2)



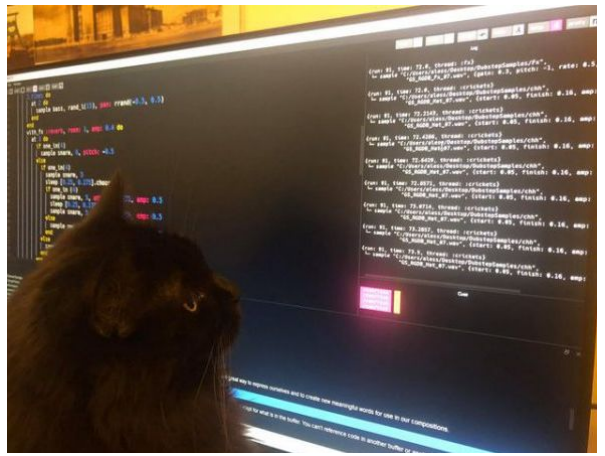
Imparare a **risolvere problemi** con soluzioni **computabili** richiede come minimo di

- Imparare a ragionare in modo **algoritmico**
- Imparare a tradurre questi ragionamenti in un **programma** effettivamente **eseguibile**



Non dimentichiamo però

- Usare un editor di testo
- Usare caratteri speciali
- Compilare un programma
- Decifrare i messaggi del compilatore
- …



Tutto **contemporaneamente**

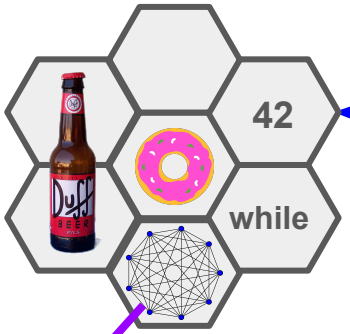
Motivazione - Carico cognitivo (1)

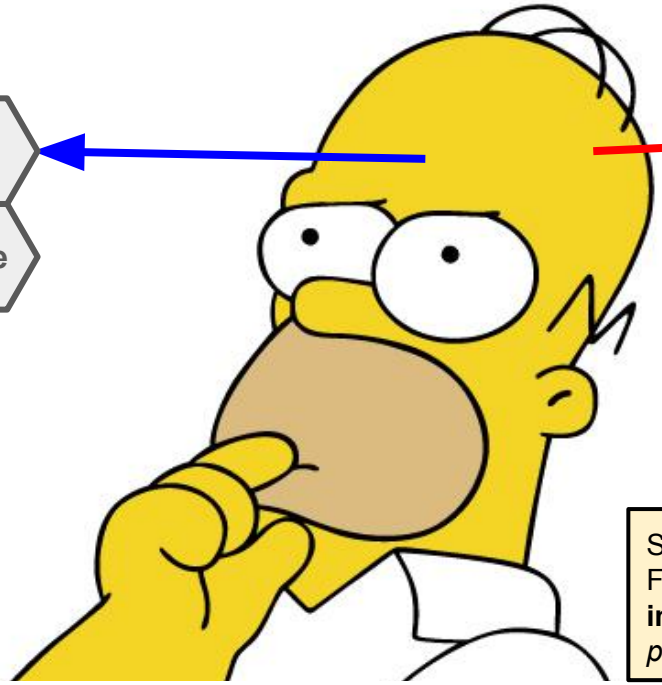
Memoria di lavoro

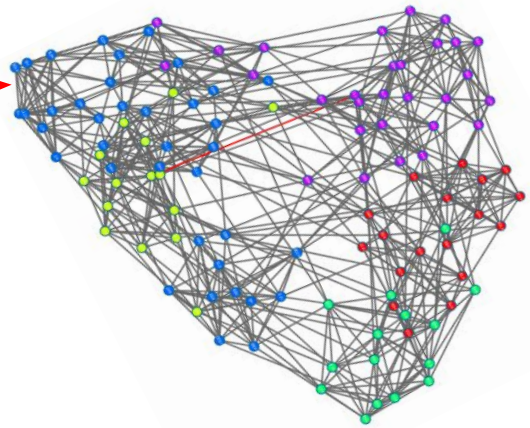
- Limitata
  - **Pochi** elementi contemporaneamente
  - **Breve** durata
- **Collo di bottiglia** durante l'apprendimento

Memoria a lungo termine

- Virtualmente illimitata
- Funziona per **schemi**







Gli **schemi** occupano poco spazio

Sweller, John, Jeroen JG Van Merrienboer, and Fred Paas. "**Cognitive architecture and instructional design: 20 years later.**" *Educational psychology review* 31 (2019): 261-292.

Motivazione - Carico cognitivo (2)

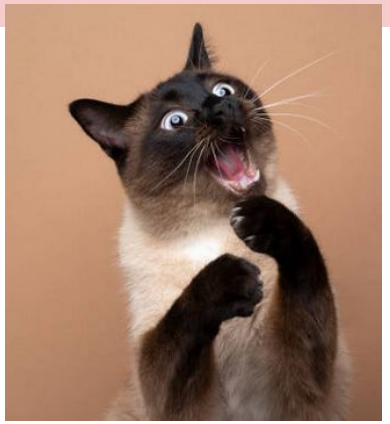
Carico cognitivo **intrinseco**

- Legato alla **complessità** dell'informazione
- Legato alle **conoscenze pregresse** del discente
- In generale "non può" essere ridotto

Carico cognitivo **estrinseco**

- Legato a come l'informazione è **presentata**
- Legato a quello che il discente deve **fare** mentre impara
- <u>Può spesso essere ridotto</u> lavorando sui due punti precedenti

Il **sovraccarico** cognitivo implica non solo apprendere **meno** ma apprendere **peggio**

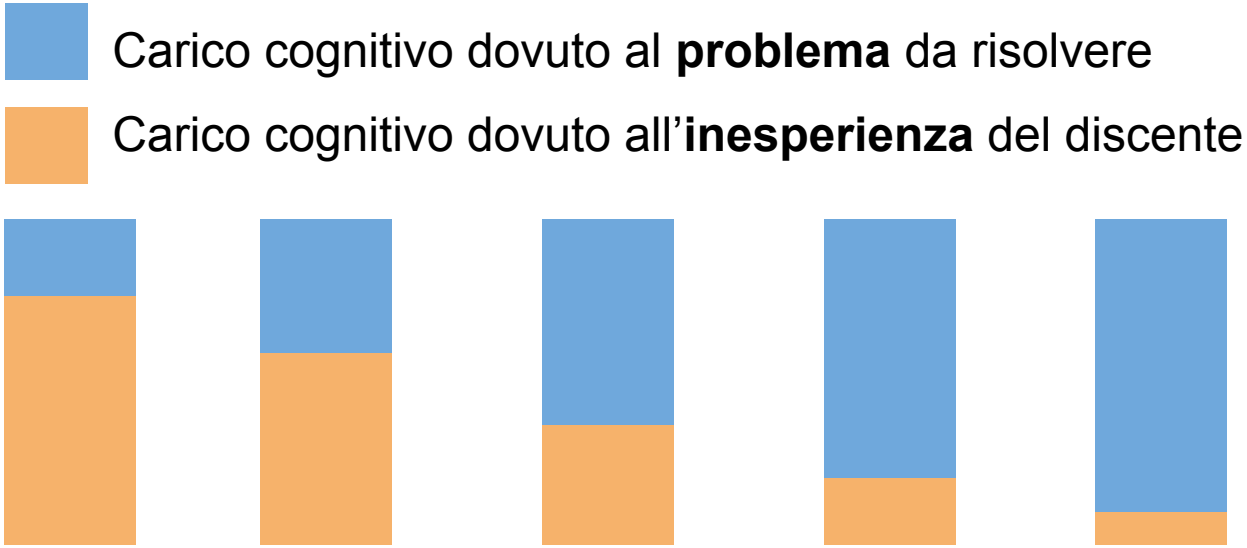


Un possibile strumento didattico

Come strumento per **controllare il carico cognitivo** durante **esercitazioni** di programmazione:

Esempi da completare (*completion problems*)

- Soluzioni **algoritmiche parziali** al problema, con pezzi mancanti da completare
- Rappresentate tramite **diagrammi di flusso** "con buchi"
- Il supporto "scompare" mano a mano in esercizi successivi *sullo stesso argomento*



Dettagli della sperimentazione

Perché?

- Valutare il possibile impatto di un supporto “a scomparsa” basato su **esempi da completare** nella forma di **diagrammi di flusso**

Dove?

- Corso di Programmazione (*CS1*), Laurea Triennale in Informatica (**primo anno**), Università di Padova

Quando?

- Primi **quattro laboratori** pratici del corso (Marzo - Aprile 2024)

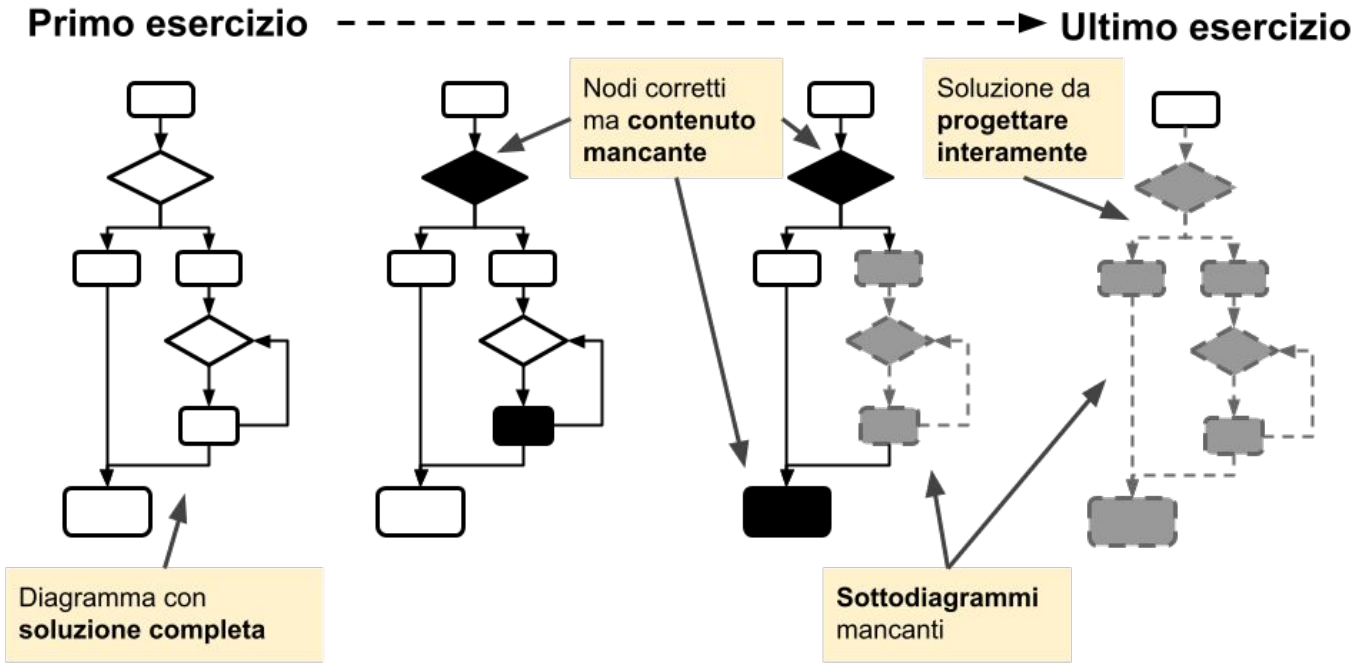
Cosa?

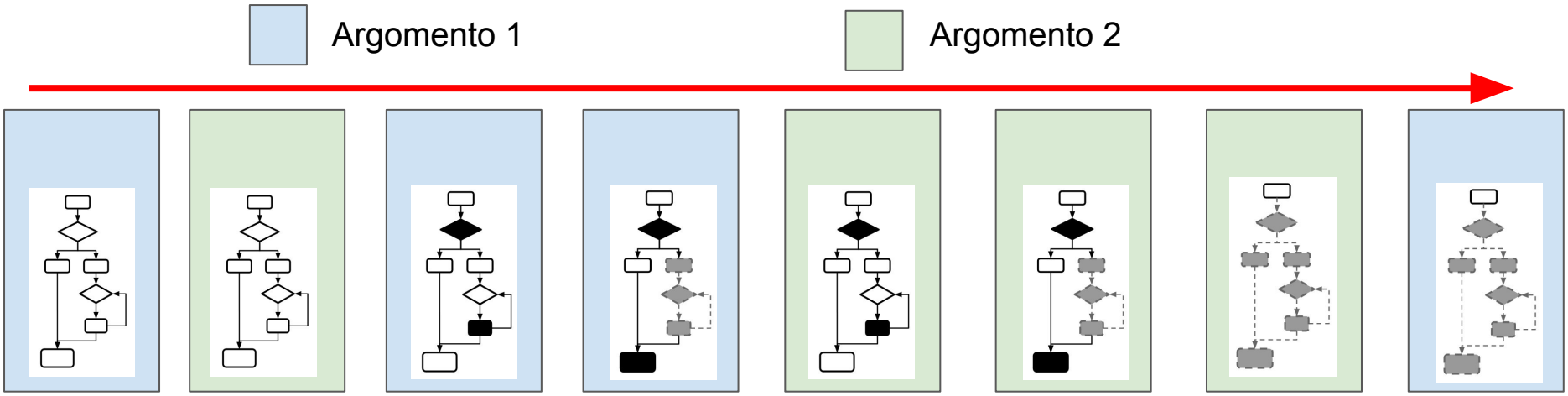
- Il corso insegna la programmazione in linguaggio C
- **Gruppo sperimentale**: ha ricevuto supporto con esempi da completare
- **Gruppo di controllo**: non ha ricevuto esempi da completare
- **Pretest** a monte per dividere in due gruppi omogenei
- **Posttest** a valle (quinto laboratorio) per valutare gli effetti

Chi?

- 263 studentesse e studenti in totale (registrati al corso)
- 173 hanno partecipato al Pretest
- 83 hanno partecipato al Posttest
- **68** fanno parte dell’analisi dei dati

Materiali di supporto (gruppo sperimentale)

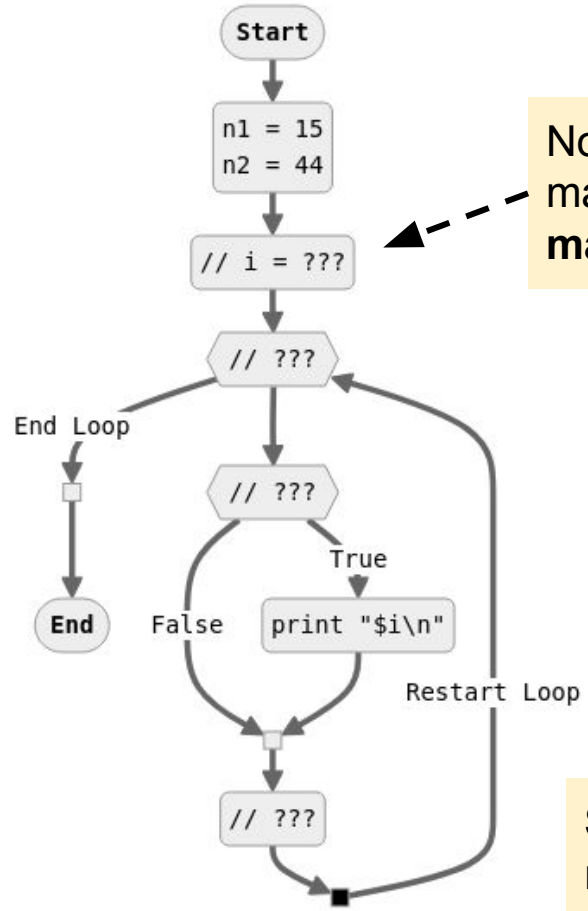




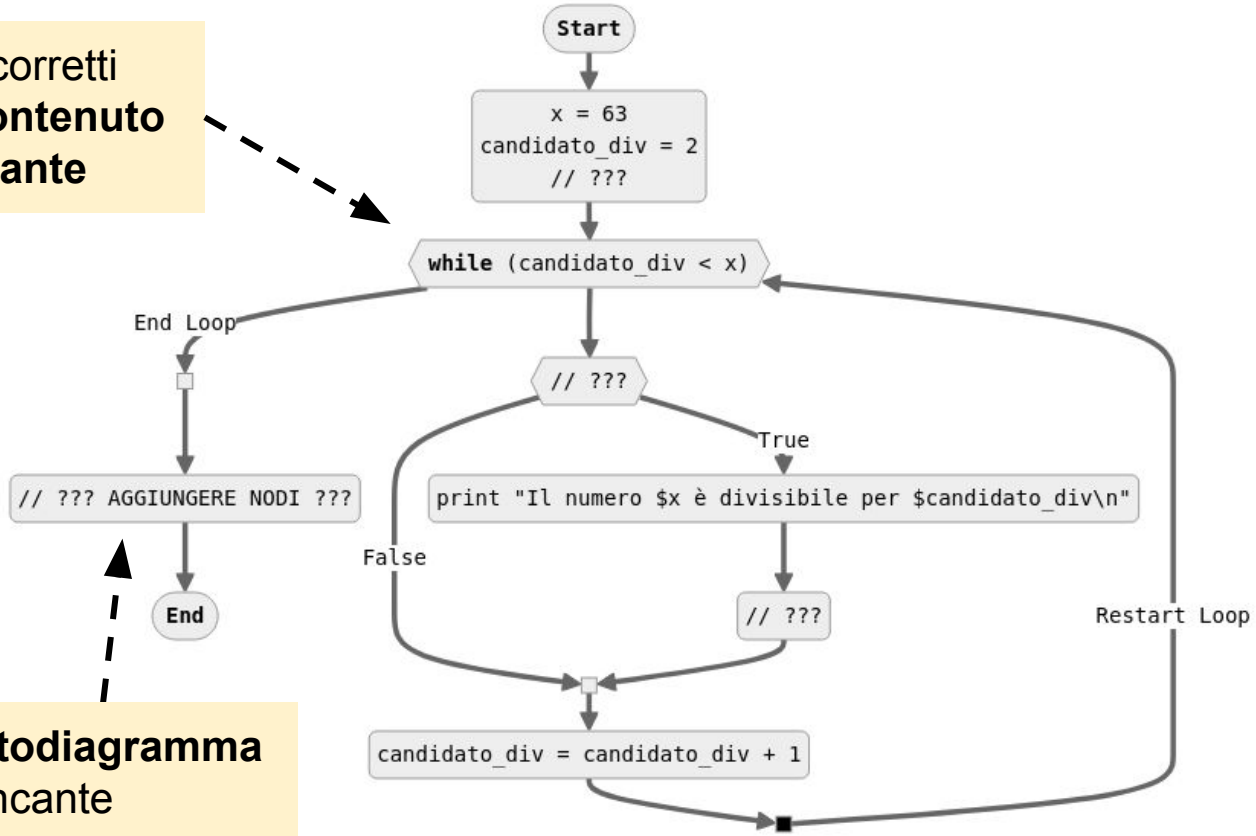
Materiali di supporto (gruppo sperimentale)

Argomenti

- Selezione
- Iterazione
- Funzioni
- Array



Nodi corretti ma **contenuto mancante**



Sottodiagramma mancante

Pretest (1)

Segnalare **esperienze precedenti** con la programmazione (sì/no)

- Scuole superiori
- Università
- Professionale

[0, 9]

+

Autovalutare competenze (nessuna, poca, abbastanza, ampia)

- Variabili
- Logica Booleana
- Selezione
- Iterazione
- Funzioni
- Ricorsione

[0, 9]

+

Predire l'esito dell'esecuzione di tre diagrammi di flusso*

[-9, 9]

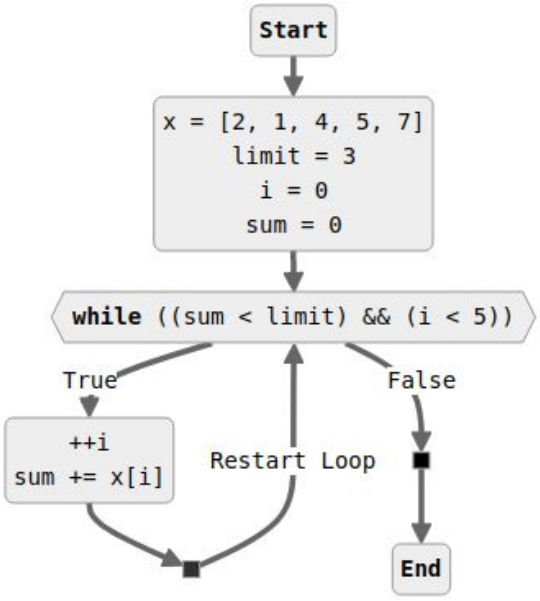
=

[-9, 27]

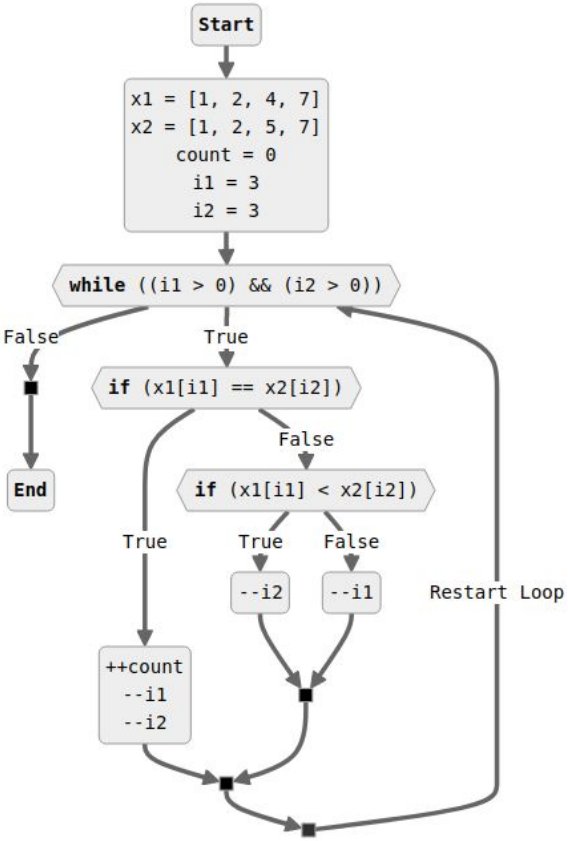
* selezionati da Lister, Raymond, et al. "**A multi-national study of reading and tracing skills in novice programmers.**" *ACM SIGCSE Bulletin* 36.4 (2004): 119-150.

Domanda: “Indicare il valore della variabile … dopo l’esecuzione”

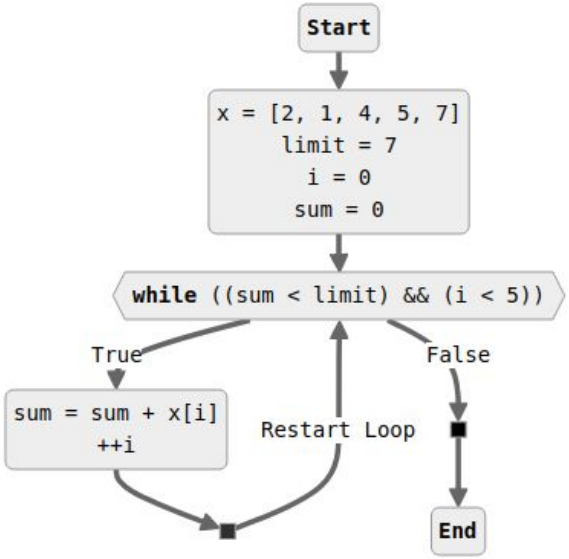
1) variabile **i** (risposta: 2)



2) variabile **count** (risposta: 2)

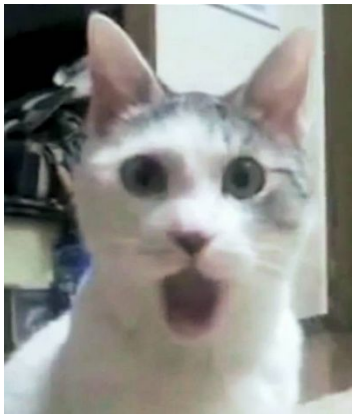


3) variabile **i** (risposta: 3)



Pretest (3)

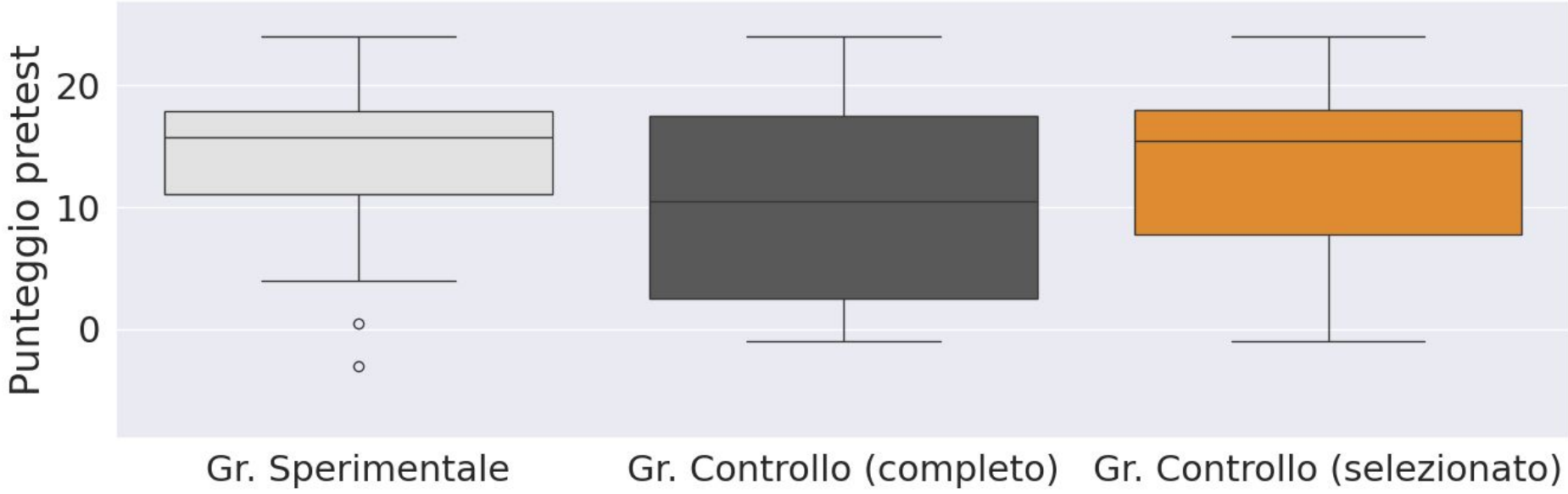
- I risultati del Pretest hanno permesso di dividere 173 studentesse e studenti in **due gruppi omogenei**
- **83** di questi hanno poi partecipato al Posttest
  - 34 del gruppo sperimentale
  - 49 del gruppo di controllo
- Questi due "sottogruppi" non erano confrontabili a livello di Pretest
- Abbiamo quindi selezionato un sottoinsieme del gruppo di controllo che fosse confrontabile a Pretest con il gruppo sperimentale



Gruppo	N. studenti	Punteggio medio	Deviazione standard
Sperimentale	34	14.13	6.31
Controllo (completo)	49	10.64	7.97
Controllo (selezionato)	34	13.76	6.71

Pretest (4)

Gruppo	N. studenti	Punteggio medio	Deviazione standard
Sperimentale	34	14.13	6.31
Controllo (completo)	49	10.64	7.97
Controllo (selezionato)	34	13.76	6.71



Posttest (1)

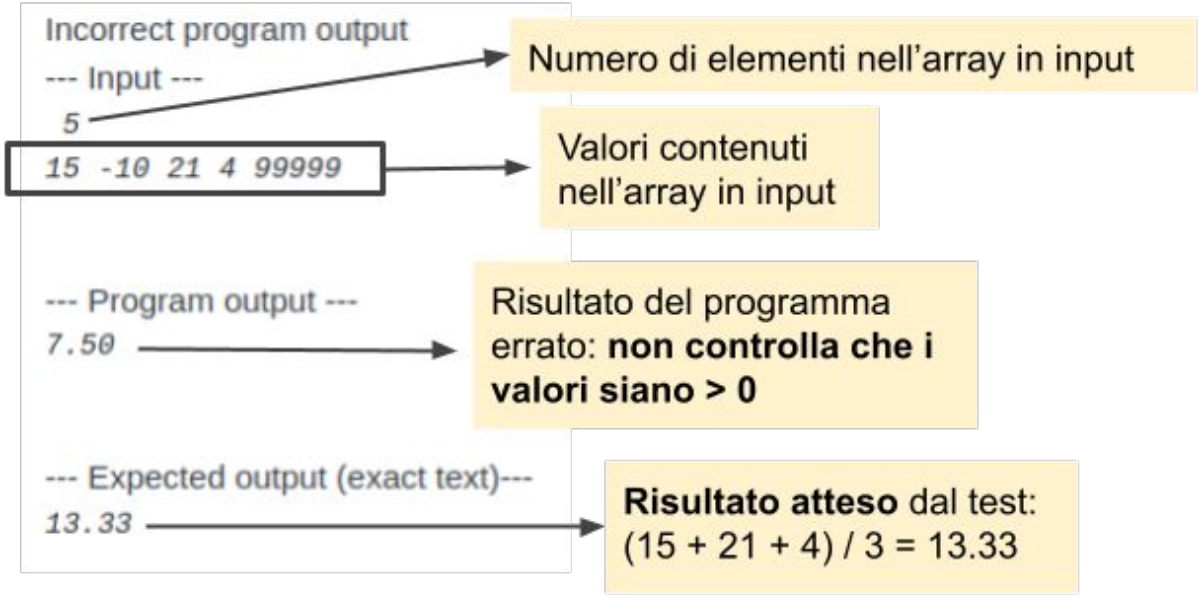
Risolvere con un programma C il “classico” *Rainfall Problem*:

Scrivete un programma che, dato un **array di N > 0 valori interi** (una qualsiasi combinazione di numeri interi, cioè maggiori, minori o uguali a zero), legga questi valori uno ad uno fino ad incontrare il **valore di guardia 99999**. Dopo aver incontrato il valore di guardia il programma deve stampare la **media dei valori positivi** (maggiori di 0) letti fino a quel momento, **escluso il valore di guardia**.

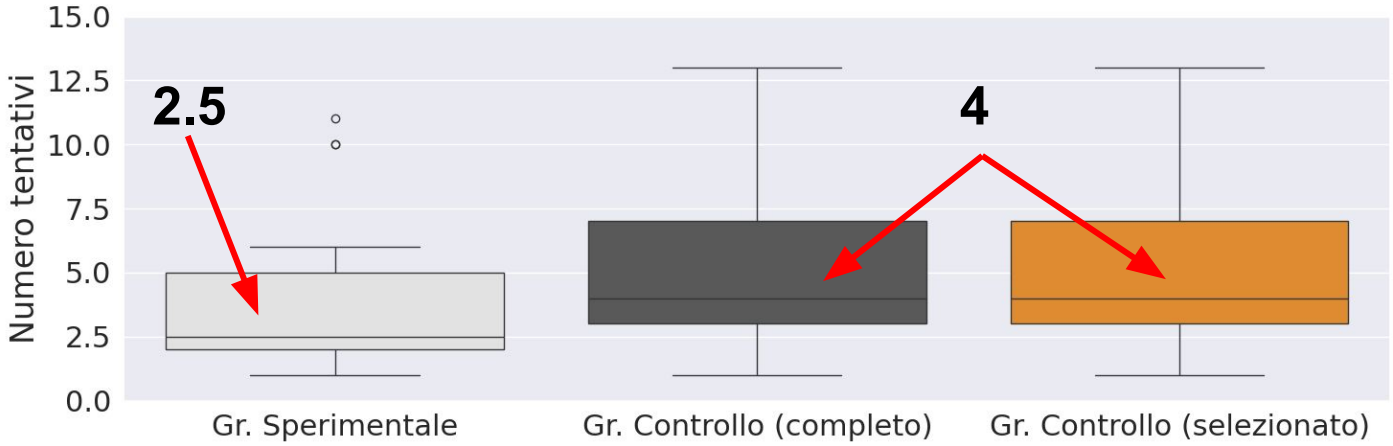
- Nessun limite al numero di tentativi
- Nessun obbligo di consegna
- Ogni “consegna” veniva testata con una serie di test di unità
- L’esito dei test veniva mostrato

Errori controllati con i test di unità

- Il programma non controlla di aver raggiunto la guardia (ma prende tutti gli elementi dell'array)
- Il programma include i valori uguali a 0 nella media
- Il programma include i valori negativi nella media
- Il programma controlla che il valore sia inferiore alla guardia (invece di diverso)
- Il programma assume la lunghezza dell'array (non la legge da input)
- Il programma sbaglia a calcolare la media (invertendo i valori della frazione)
- Il programma stampa solo la parte intera della media
- Il programma non controlla che siano stati sommati più di 0 valori nel totale (e fa quindi una divisione per 0)

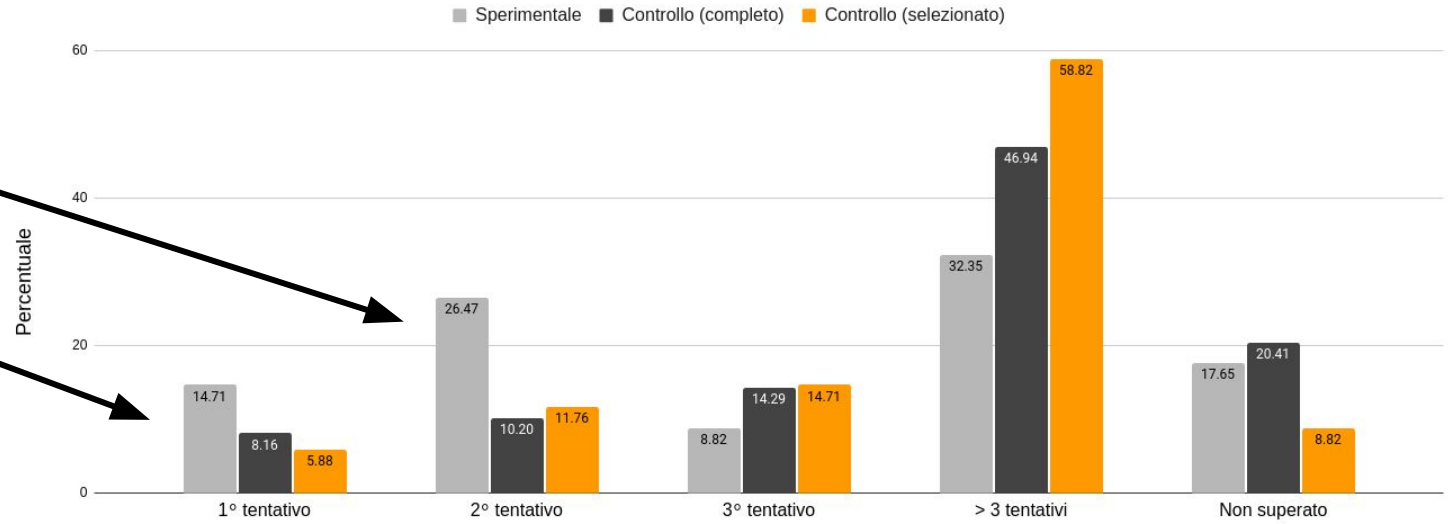


Posttest (3)



Risolto <u>entro i primi tre tentativi</u>:
SP: 50%
CT: 32%

Chi supera del gruppo sperimentale mediamente lo fa **prima**



Conclusioni

Limiti

- Pochissime persone hanno risolto il Posttest al primo tentativo (parlavamo di aspettative…)
- Non abbiamo un dato solido che ci dica chi ha effettivamente fatto uso degli esempi a completamento

Risultati

- A livello statistico il gruppo sperimentale ha avuto risultati migliori nel Posttest
- Il risultato sicuramente non è “schiacciante” ma è incoraggiante

Implicazioni

- Non c’è svantaggio, anzi sembra esserci vantaggio nel:
  - Separare inizialmente il ragionamento algoritmico dalla codifica
  - Usando strumenti di supporto “a scomparsa”