

# Ripasso 1

# Dal problema al programma

# Problemi e strategie risolutive

Prima di cominciare ricordiamo che:

- Un **problema** può essere risolto **solo** se lo si è ben **compreso**
- Risolvere un problema significa individuare una **strada** che conduca alla sua **soluzione**, ma la strada, in particolare in informatica, deve essere **semplice** e **lineare**
- La **strategia risolutiva** ipotizzata deve essere rappresentata in una forma comprensibile al **calcolatore**.

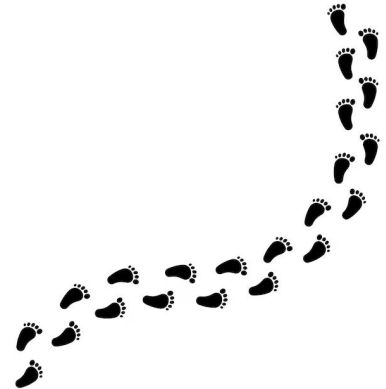
# Problemi e strategie risolutive

Ogni giorno dobbiamo affrontare problemi di varia natura, ma che cosa è un problema?

Un **problema** è un quesito che chiede di trovare degli **elementi ignoti** (la **soluzione**) partendo dagli **elementi noti** (i **dati iniziali**) da esso forniti.

Una **strategia risolutiva** è un insieme di **passi** da compiere per giungere alla **soluzione** del problema.

La **soluzione** o **risultato finale** è l'obiettivo che vogliamo raggiungere.



# Fasi del processo di risoluzione di un problema

1. **Analisi del problema:** identificare i **dati iniziali** di cui si dispone e l'**obiettivo** da raggiungere
2. **Progettazione della strategia risolutiva:** specificare le **azioni** da intraprendere per risolvere il problema, cioè per trasformare i dati iniziali in risultati finali
3. **Verifica della soluzione:** verificare che i risultati ottenuti siano rispondenti agli obiettivi iniziali. Se la verifica dà esito **negativo** bisogna **ripetere** il percorso: compiere di nuovo l'analisi, modificare il progetto della strategia risolutiva e eseguire nuovamente la verifica.

# L'algoritmo

Dopo aver analizzato il problema e identificato i **dati iniziali** su cui agire per ottenere i **risultati finali** bisogna ideare e progettare la **strategia risolutiva** cioè individuare la corretta **sequenza** di operazioni da effettuare sui dati per **risolvere** il problema (ottenendo il risultato finale) e descriverla attraverso una sequenza di passi elementari che siano **univocamente interpretabili**.  
La descrizione ottenuta viene definita **algoritmo**.



# Rappresentazione di un algoritmo

Un algoritmo è un **procedimento** che risolve un problema attraverso un numero **finito** di **passi** elementari.

Per descrivere correttamente un **algoritmo** si utilizzano 2 **strumenti** :

Diagrammi a blocchi  
o  
Flow Chart

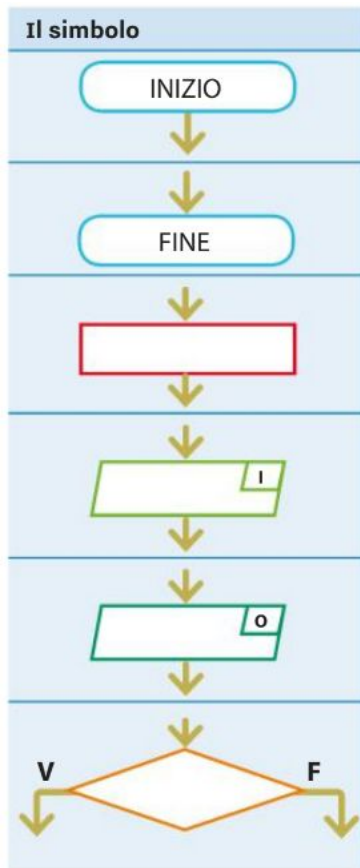
La pseudocodifica

# Diagrammi a blocchi o flow chart

- Il metodo più semplice e più diffuso per descrivere gli algoritmi è quello che utilizza i **diagrammi a blocchi** (o **flow chart**, “**diagrammi di flusso**”).
- Si tratta di una **rappresentazione grafica**, realizzata mediante appositi simboli, che mette in evidenza il flusso di esecuzione delle istruzioni.



# Diagrammi a blocchi o flow chart

Il simbolo	indica nei diagrammi a blocchi	Osserva bene...
 <p>INIZIO</p> <p>↓</p> <p>FINE</p> <p>↓</p> <p></p> <p>↓</p> <p>I</p> <p>↓</p> <p>O</p> <p>↓</p> <p>V F</p>	<p><b>Blocco di inizio:</b> da questo blocco parte l'algoritmo</p> <p><b>Blocco di fine:</b> con questo blocco termina l'algoritmo</p> <p><b>Blocco di azione:</b> per scrivere istruzioni che effettuano trasformazione di dati</p> <p><b>Blocco di input:</b> per inserire un valore all'interno dell'algoritmo</p> <p><b>Blocco di output:</b> per visualizzare informazioni e/o risultati</p> <p><b>Blocco di decisione:</b> per effettuare operazioni di confronto. Al suo interno, si riassumono domande che possono avere come risposta solo due valori: <b>SÌ</b> (cioè Vero) e <b>NO</b> (cioè Falso)</p>	<p>La freccia presente in questi due blocchi: all'inizio si può seguire una sola direzione e alla fine si può giungere da un'unica strada</p> <p>I blocchi di azione, di input e di output hanno una sola freccia che vi arriva e una sola che parte da essi</p> <p>È l'unico simbolo che prevede due frecce in uscita, ma ne ha sempre solo una in entrata</p>

# Pseudocodifica - esempio

Un altro modo per rappresentare un algoritmo è utilizzare lo **pseudolinguaggio** che può essere definito come un **linguaggio formale** che utilizza simboli ai quali corrisponde uno ed un solo significato in qualsiasi contesto.

La descrizione formale dell'algoritmo in pseudolinguaggio si dice **pseudocodice**.

L'attività di scrittura dello pseudocodice prende il nome di **pseudocodifica**.

# Esempio

Calcola l'area di un quadrato, prendendo in input la misura del suo perimetro.

Analisi del problema:

Dati iniziali?

Risultato finale?

Strategia risolutiva?

# Esempio

Calcola l'area di un quadrato, prendendo in input la misura del suo perimetro.

**Analisi del problema:**

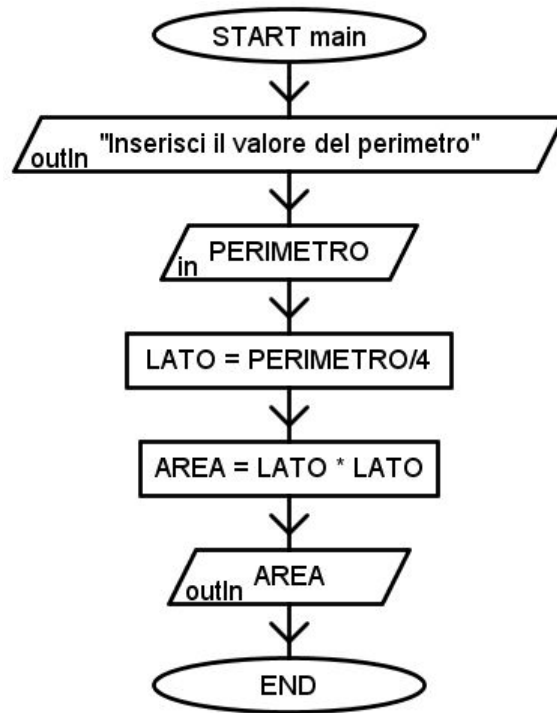
Dato iniziale (o di input): PERIMETRO

Risultato finale (o dato di output): AREA

**Strategia risolutiva:**

1. Calcolo del LATO=PERIMETRO/4
2. Calcolo dell'AREA=LATO\*LATO

# Soluzione



# Pseudocodifica - esempio

*Calcola l'area di un quadrato,  
Prendendo in input la misura del suo perimetro.*  
Lo pseudocodice è il seguente:

```
PROGRAM main
|
| OUTLN "Inserisci il valore del perimetro"
|
| INPUT PERIMETRO
|
| LATO = PERIMETRO/4
|
| AREA = LATO * LATO
|
| OUTLN AREA
|
END
```

# Variabili

Nell'esercizio appena visto abbiamo preso

- come dato di input il **PERIMETRO**
- come dato di output l'**AREA**
- e per calcolare l'area abbiamo dovuto trovare un altro dato... Quale?

# Variabili

Il **LATO** che non è né di input né di output.

In pratica abbiamo trovato i **dati** che dovevamo utilizzare per **sviluppare** gli **algoritmi** e abbiamo dato loro dei **nomi**: PERIMETRO, AREA, LATO...

È un po' come se avessimo creato dei **contenitori** all'interno dei quali inserire i valori: questi contenitori sono chiamati **variabili** perché possono cambiare durante l'esecuzione dell'algoritmo.



# Variabili

Una **variabile** è un **contenitore** di memoria utilizzato per contenere dei dati che possono essere **modificati** durante l'esecuzione del procedimento risolutivo.

Una variabile è caratterizzata da:

- **Identificatore** (cioè un nome) che identifica quella cella di memoria e la distingue da tutte le altre (p. es. PERIMETRO,...)
- **Valore**: il valore che il dato può assumere (p. es. PERIMETRO=40)
- **Tipo**: l'insieme di valori che il dato può assumere (p. es. intero, reale, ...)

# Istruzione di assegnazione

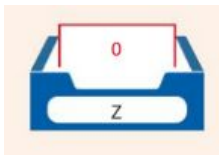
L'istruzione di **assegnazione** è quella particolare istruzione che permette di **definire** il valore attuale di una variabile.

Il valore rimane **inalterato** fino ad una nuova assegnazione alla variabile.

Per esempio: per attribuire il valore 0 alla variabile Z facciamo uso della seguente istruzione di assegnazione:

**Z = 0**

Vuol dire che la variabile Z “contiene” il valore 0:



# Costanti

Oltre alle **variabili** esistono anche le **costanti**.

Una **costante** è un **contenitore** di memoria utilizzato per contenere un dato che **non** può essere **modificato** durante l'esecuzione del procedimento risolutivo.

La costante può essere pensata come un contenitore di memoria a cui è associato un **identificatore** e un **valore** che potrà essere utilizzato ma non modificato, rimanendo così fisso per tutta l'esecuzione del procedimento risolutivo.

# Classificazione dei dati

**Variabili** e **costanti** sono utilizzate per rappresentare **dati**, i quali, all'interno dell'algoritmo, possono essere classificati in base alla loro **funzione**:

- **Dati di input**: sono i dati che vengono forniti dall'esterno e servono per la risoluzione del problema (p. es. PERIMETRO)
- **Dati di output**: sono i dati che vengono comunicati all'esterno e rappresentano la soluzione del problema (p. es. AREA)
- **Dati di lavoro** (o dati intermedi): sono quelli che vengono utilizzati durante l'esecuzione del processo risolutivo (p. es. LATO).

# Classificazione delle istruzioni

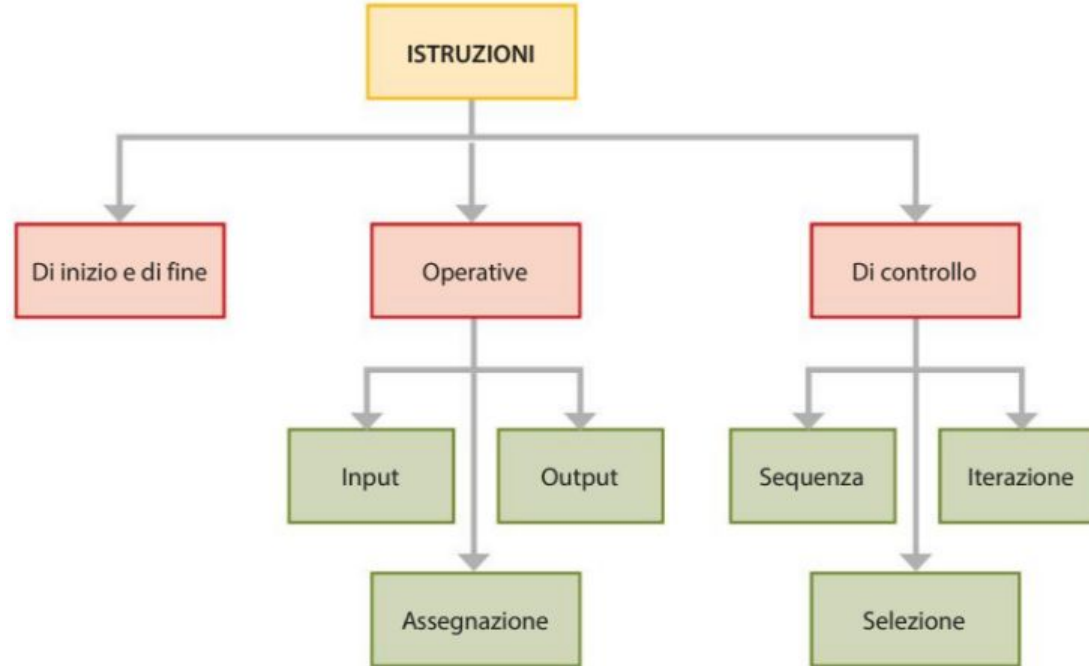
Quali sono le **istruzioni** possibili in un calcolatore moderno?

Cosa riesce a fare direttamente?

Quali sono i blocchi elementari che possiamo comporre per costruire espressioni sempre più complesse, programmi sempre più sofisticati?

Le istruzioni presenti in un algoritmo possono essere classificate in diverse categorie in base al loro **comportamento** (o tipo).

# Classificazione delle istruzioni



# Istruzioni di inizio e fine

**Istruzioni di inizio e fine:** Indicano quale istruzione dell'algoritmo debba essere eseguita inizialmente e quale determini la fine dell'esecuzione.



# Istruzioni operative

**Istruzioni operative:** Istruzioni che servono per:

- acquisire dati iniziali (**input**)
- effettuare elaborazioni (**assegnazione**)
- comunicare i risultati finali (**output**)



# Istruzioni di input e output

**Istruzioni di input e output:** indicano una trasmissione di dati o messaggi fra l'algoritmo e tutto ciò che è **esterno** all'algoritmo.

Tali istruzioni si dicono:

- di **input** o ingresso o **lettura** quando l'algoritmo riceve dati dall'esterno;
- di **output** o uscita o **scrittura** quando i dati sono comunicati dall'algoritmo all'esterno.

# Istruzione operativa di assegnazione

L'**assegnazione** ha sempre la forma:

**variabile = espressione;**

Ad esempio:

**a = 2\*b + c;**

A differenza di quanto avviene in matematica, la scrittura

**2\*b + c = a;**

non ha senso.

**Attenzione:** l'assegnazione viene eseguita solo se il risultato della valutazione di **espressione** posta a destra del simbolo **=** appartiene all'insieme dei possibili valori che può assumere la **variabile** posta a sinistra di **=**.

Altrimenti l'assegnazione potrebbe non essere effettuata o generare un **errore**.

# Istruzione operativa di assegnazione

Esempi di istruzioni di assegnazione:

l'istruzione: **a = 5+3;**

l'istruzione: **x = 2;**

l'istruzione: **i = i+1;**

l'istruzione: **w = 3 % 2;**

(dove % sta per modulo, ovvero resto della divisione intera).

# Istruzione operativa di assegnazione

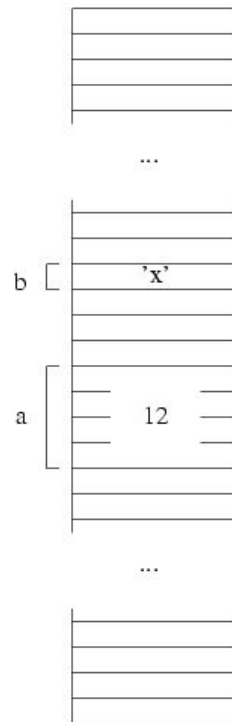
Ogni volta che si dichiara una **variabile**, viene riservato uno spazio di **memoria** per poter memorizzare il suo **valore**.

La **dimensione** di questo spazio dipende dal **tipo** della variabile: per esempio, le variabili di tipo **char** (caratteri alfabetici, punteggiatura, etc.) occupano un solo byte, mentre le variabili di tipo **int** (numeri interi) ne occupano 4.

# Istruzione operativa di assegnazione

Il disegno mostra come potrebbe essere strutturata la memoria se un programma usa una **variabile intera a** e una **variabile carattere b**: la variabile **a** occupa quattro byte consecutivi nella memoria, mentre la variabile **b** ne occupa uno solo. Ogni volta che si assegna ad **a** un valore, questo viene scritto nei byte che le sono assegnati.

Ogni volta che il valore di **a** viene usato (per esempio per fare un calcolo come **y=a+c;**), il calcolatore va a cercare il valore in quei quattro byte.



# Strutture di controllo

**Strutture di controllo:** consentono di scegliere percorsi differenti durante l'esecuzione in base al verificarsi o meno di determinate condizioni.

Si suddividono in strutture di:

- sequenza
- selezione
- iterazione

Esse consentono di specificare **se, quando, in quale ordine e quante volte** devono essere eseguite le istruzioni che compongono un programma.

# Strutture di controllo

**Sequenza:** è la struttura di controllo fondamentale ed è data dalla semplice **successione** delle istruzioni, che vengono quindi eseguite in sequenza, una dopo l'altra.

**Selezione:** questa struttura permette di **scegliere** la sequenza di esecuzione tra due (o più) **alternative**. Consente quindi di specificare che una data istruzione o un dato blocco di istruzioni devono essere eseguiti **"(solo) se"** vale una certa condizione.

## Strutture di controllo

**Iterazione (o ciclo):** questa struttura di controllo detta anche "iterativa" consente di specificare che una data istruzione o un dato blocco di istruzioni devono essere eseguiti **ripetutamente**.

Ogni struttura di controllo di questo tipo deve consentire di specificare sotto quali **condizioni** l'iterazione (ripetizione) di tali istruzioni debba continuare.

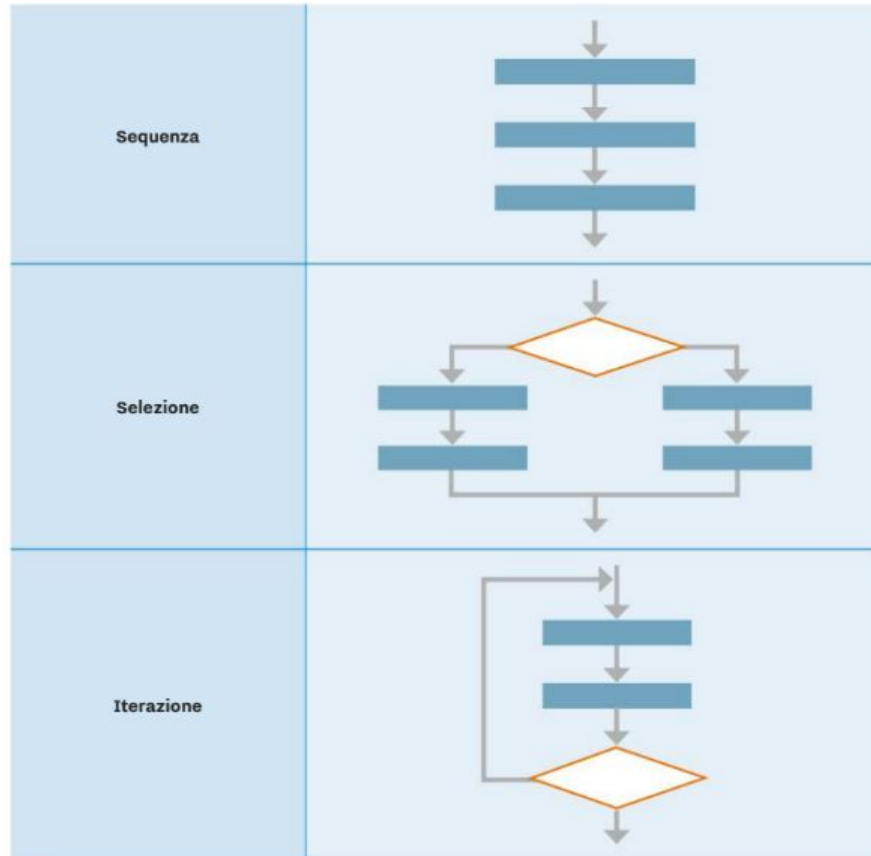


# Teorema di Böhm Jacopini

Il teorema di **Böhm-Jacopini**, enunciato nel 1966 dagli informatici Corrado Böhm e Giuseppe Jacopini, afferma che qualunque **algoritmo** può essere realizzato utilizzando le sole tre strutture di controllo fondamentali: la **sequenza**, la **selezione** e l'**iterazione**.

Questo teorema riveste un'importanza fondamentale per lo sviluppo della **programmazione strutturata**, che considera l'algoritmo come un insieme di blocchi di istruzioni, ognuno fornito di un solo ingresso e di una sola uscita. e organizzati tra di loro secondo le tre strutture di controllo, opportunamente combinate.

# Teorema di Böhm Jacopini



# Gli operatori logici

# Operatori Relazionali e Logici

Le condizioni sottoposte al controllo nei test di selezione, dette anche predicati, sono espressioni che coinvolgono una o più variabili.

I predicati si esprimono usando gli operatori relazionali che sono uguale, maggiore, minore, maggiore uguale, minore uguale, diverso.

Nei predicati spesso è utile usare anche gli **operatori o connettivi logici**: **AND, OR, NOT, XOR** in particolare quando si vogliono controllare simultaneamente più condizioni.

# Operatori Relazionali e Logici

Questi operatori sono anche detti **booleani**, dal nome di **Boole** l'inglese che nell'800 ha fondato la logica matematica.

Se chiamiamo C1 e C2 due condizioni ciascuna delle quali può essere vera (V) oppure falsa (F), allora la **congiunzione logica AND** e la **disgiunzione OR** sono definite dalle seguenti tabelle di verità e porte logiche (Una **porta logica** è un circuito digitale in grado di implementare, cioè di realizzare, simulando la "logica matematica" mediante opportuni controlli su segnali elettrici, una particolare operazione logica di una o più variabili booleane).

# Tabelle di Verita' - AND ( $\wedge$ )



<b>C1</b>	<b>C2</b>	<b>C1 AND C2</b>
<b>V</b>	<b>V</b>	<b>V</b>
<b>V</b>	<b>F</b>	<b>F</b>
<b>F</b>	<b>V</b>	<b>F</b>
<b>F</b>	<b>F</b>	<b>F</b>

# Tabelle di Verita'- OR ( V )



C1	C2	C1 OR C2
V	V	V
V	F	V
F	V	V
F	F	F

# Operatori Logici

Dunque la condizione composta **C1 AND C2** è vera soltanto se sono vere sia C1 sia C2.

Invece perchè **C1 OR C2** sia vera, basta che sia vera almeno una tra le condizioni C1 e C2.

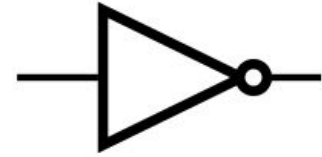


# Operatori Logici

Può accadere di controllare simultaneamente anche più di due condizioni. Quindi:

- se uso la condizione AND il predicato composto è **vero** soltanto se **tutte** le condizioni sono vere mentre è **falso** anche se **una sola** condizione non è verificata.
- se uso la condizione OR il predicato composto è **vero** se **almeno una** tra le condizioni è soddisfatta mentre è **falso** soltanto quando **nessuna** delle singole condizioni è soddisfatta.

# Operatore Logico - NOT -

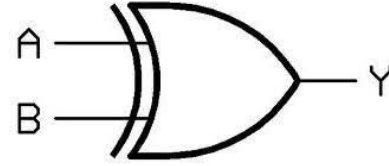


L'operatore logico NOT nega la condizione a cui è applicato.

Applicato a una condizione C è definito da questa tabella di verità:

C	NOT C
V	F
F	V

# Operatore Logico XOR ( $\oplus$ )



L'operatore XOR o disgiunzione esclusiva, afferma una verità se una sola delle proposizioni è vera.

C1	C2	C1 XOR C2
V	V	F
V	F	V
F	V	V
F	F	F

# Proposizioni Complesse

Ovviamente posso unire più operatori logici in una condizione formando **proposizioni complesse**. Per capire il valore di falsità o di verità di una proposizione complessa si costruisce una **tabella di verità** dove vengono inserite le proposizioni semplici fino a quelle composte.

# Contraddizioni e Tautologie

Le **contraddizioni** sono proposizioni che risultano sempre false (sto parlando e non sto parlando).

$P \text{ AND NOT } P$

Le **tautologie** sono proposizioni che risultano sempre vere (sto parlando oppure non sto parlando).

$P \text{ OR NOT } P$

# Esercizi