1. Che cos’è un programma

Ogni **programma** (o **app**-licazione) è costituito da tante istruzioni scritte una dopo l’altra, definite come algoritmo:

> **Un algoritmo è un insieme finito di istruzioni che, eseguite in sequenza, permettono di elaborare i dati in ingresso per ottenere in uscita i risultati richiesti dall’elaborazione, in modo da risolvere un determinato problema**

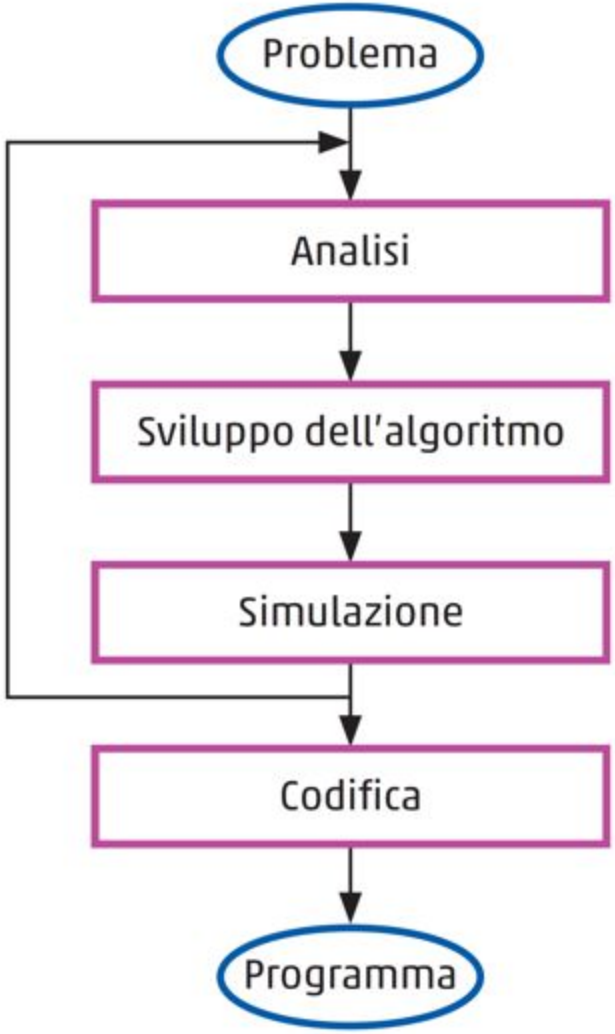
1. Che cos’è un programma

2. Dal problema al programma

Il processo di formalizzazione

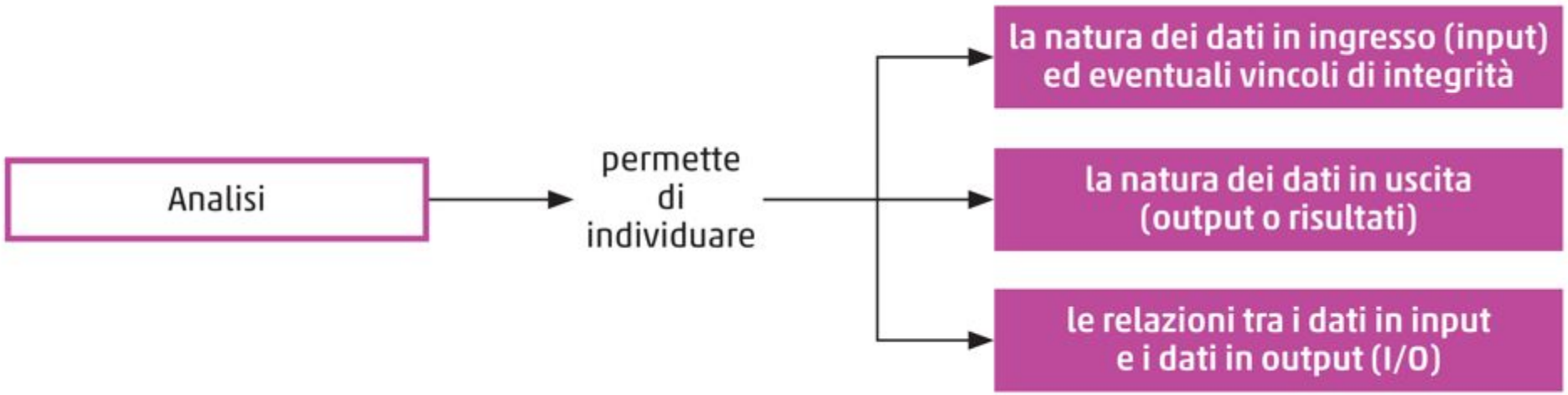
Serve a:

- focalizzare meglio gli obiettivi
- definire il modo per raggiungerli



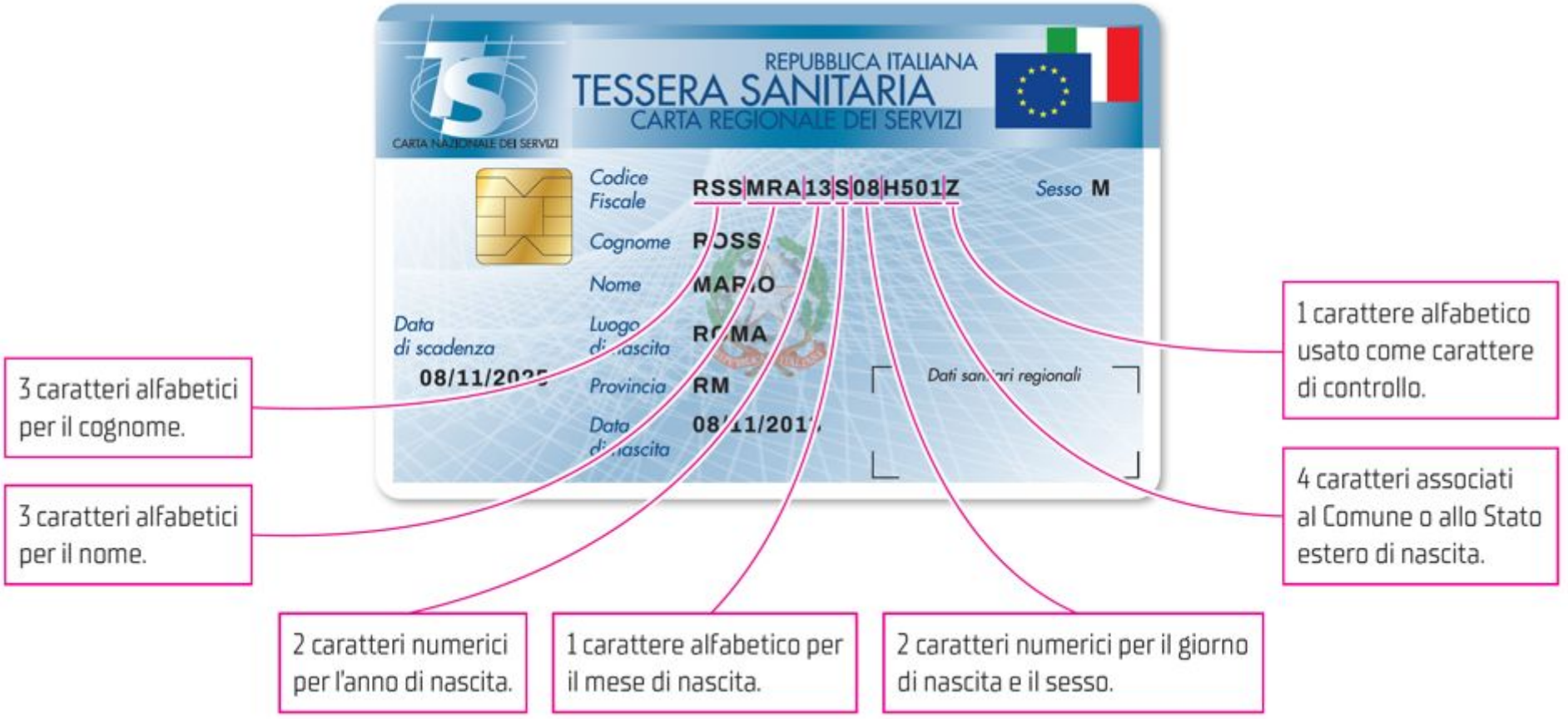
L’analisi

La prima fase consiste nell’**analisi** approfondita del problema da risolvere



2. Dal problema al programma

Durante la fase dell'analisi, vanno individuati eventuali vincoli di integrità, cioè le condizioni che gli input al programma devono rispettare per essere accettati come validi



Lo sviluppo dell'algoritmo

Un algoritmo è un insieme finito di azioni che risolvono un determinato problema, trasformando i dati di input in dati di output (o risultati) attraverso le relazioni esistenti tra input e output

L'**algoritmo risolutore ottimale** segue criteri di:

- generalità
- sintesi

La simulazione

La terza fase del processo di formalizzazione consiste nella **simulazione**

- si controlla se la sequenza di azioni ricavate è funzionante

La codifica

È la **traduzione** dell'algoritmo in un insieme di istruzioni comprensibili all'elaboratore mediante un **linguaggio di programmazione**

- C Language, C++, C#, Python, Java, JavaScript, Visual Basic, Scratch

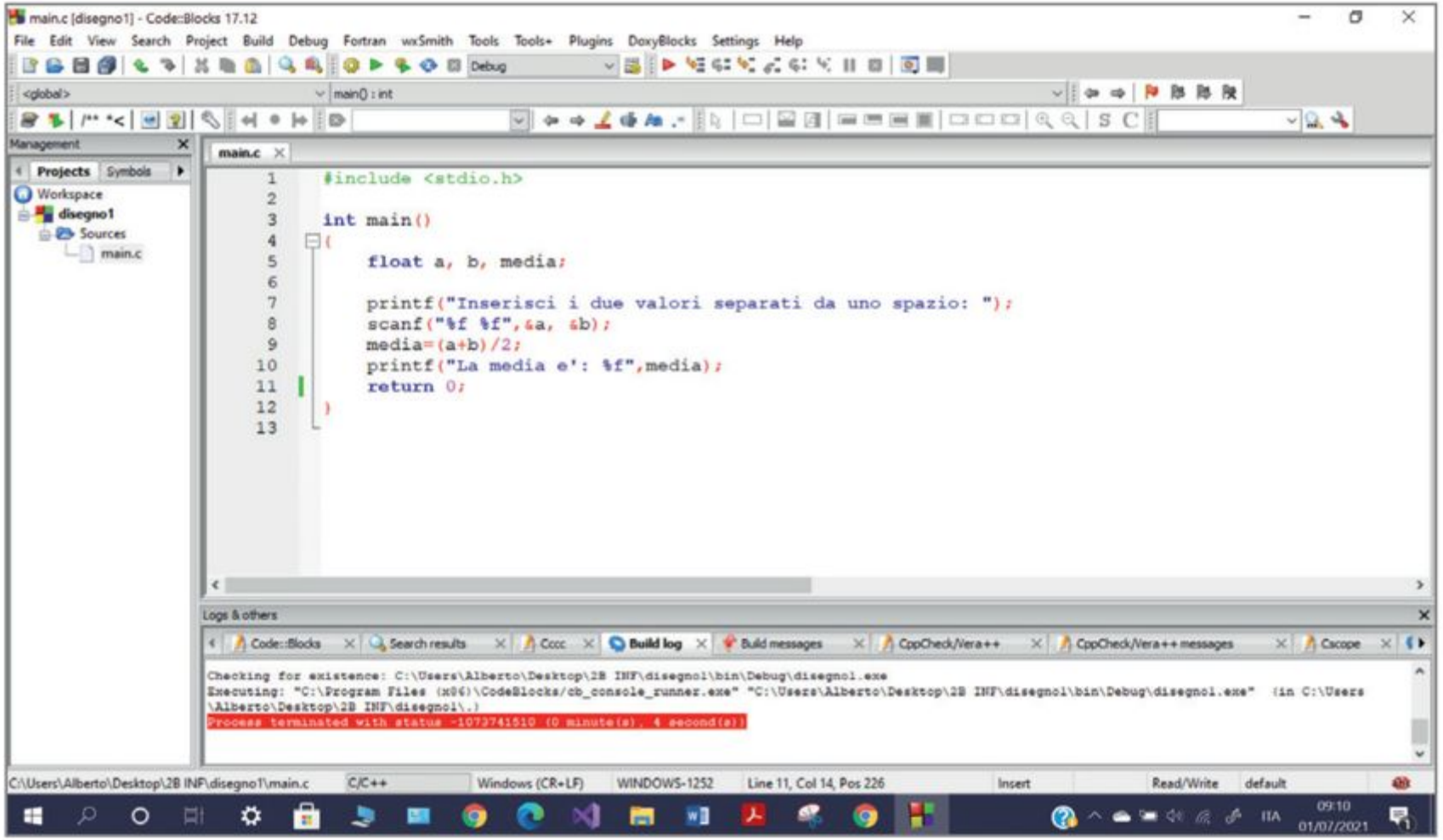
Editare il programma

Ogni linguaggio di programmazione offre un proprio ambiente di sviluppo chiamato **IDE** (*Integrated Development Environment*)

Questo include un **editor** con cui:

- scrivere il codice nel linguaggio di programmazione scelto
- salvarlo in memoria ricavando il **programma sorgente**

2. Dal problema al programma



2. Dal problema al programma

Fra i programmi messi a disposizione dall'**IDE** troviamo:

- Il **traduttore**
  - traduce il programma editato in linguaggio macchina per ottenere il **programma eseguibile**
- Il **debugger**
  - aiuta a individuare e correggere eventuali errori nel codice

3. Concetto di variabile

Le variabili sono gli oggetti elaborati dalle istruzioni del programma

Esse risiedono nella memoria dell'elaboratore e corrispondono a contenitori dei valori che sono utilizzati durante l'esecuzione del programma

Alle variabili è associato un nome, detto identificatore, che le individua univocamente all'interno del programma

Sono chiamate variabili perché a loro viene associato un valore che può cambiare durante l'esecuzione del programma

3. Concetto di variabile

L'operazione tipica che viene eseguita su una variabile è detta **assegnazione**

- È l'inserimento di un valore nel contenitore associato all'identificatore della variabile
- Il simbolo utilizzato negli algoritmi per indicare l'assegnazione è ←

Oltre alle variabili, le istruzioni di un programma possono elaborare oggetti chiamati **costanti**

- Il loro valore, assegnato a inizio programma, non cambia mai durante l'esecuzione

3. Concetto di variabile

1

A inizio esecuzione

I tre contenitori sono vuoti.

2

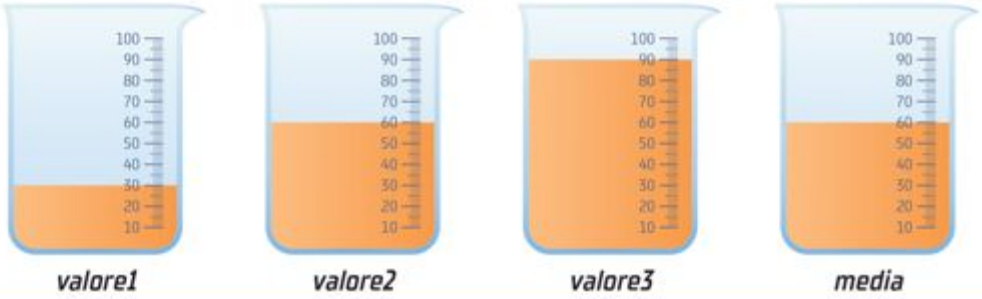
Dopo l'input dei dati

I tre contenitori contengono i valori in ingresso al programma.

3

Dopo il calcolo della media

I primi tre contenitori continuano a contenere i valori precedentemente caricati, mentre il contenitore della media contiene il valore appena calcolato.



$$media = \frac{valore1 + valore2 + valore3}{3}$$

3. Concetto di variabile

Tabella 1 Operazioni di assegnazione

Assegnazione	Risultato		Commento
Assegnare un valore a una variabile $A \leftarrow 9$	**Prima:** A	**Dopo:** A 9	Nel contenitore associato all'identificatore della variabile *A* viene inserito il valore 9. Nel caso in cui il contenitore associato ad *A* contenga un precedente valore, questo viene perso e sostituito dal nuovo valore
Assegnare a una variabile il valore di un'altra variabile $B \leftarrow A$	**Prima:** A 9 B	**Dopo:** A 9 B 9	Nel contenitore identificato da *B* e corrispondente alla variabile *B* viene inserito il valore contenuto nel contenitore associato alla variabile *A*. In questo caso anche *B* assume il valore 9. Il contenitore associato ad *A* non perde dunque il suo valore di partenza
Incrementare il valore di una variabile $A \leftarrow A+1$	**Prima:** A 9	**Dopo:** A 10	È calcolato il valore dell'espressione a destra della freccia, leggendo il contenuto del contenitore identificato da *A* e aumentandolo di una unità, per poi inserire il risultato ottenuto nuovamente nel contenitore della variabile *A*. La variabile *A* perde il vecchio valore che è sostituito con quello nuovo
Modificare il valore di una variabile $A \leftarrow A+B$	**Prima:** A 10 B 9	**Dopo:** A 19 B 9	Il contenuto della variabile *A* è addizionato al contenuto della variabile *B* e il risultato dell'espressione è inserito nel contenitore della variabile *A* (a sinistra della freccia). La variabile *A* perde il vecchio valore che viene sostituito con quello nuovo, mentre la variabile *B* mantiene il suo valore inalterato
Scambiare il valore tra due variabili $AUS \leftarrow A$ $A \leftarrow B$ $B \leftarrow AUS$	**Prima:** A 19 B 9 AUS	**Dopo:** A 9 B 19 AUS 19	Per poter scambiare il valore contenuto nelle variabili *A* e *B* occorre utilizzare una variabile ausiliaria che identifichiamo con *AUS*. In questo modo non si perde nessun valore

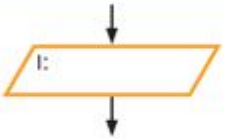
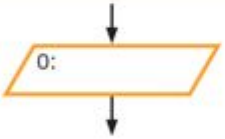
4. Gli schemi di flusso

Per rappresentare graficamente un algoritmo si utilizzano gli **schemi di flusso**, ottenendo una descrizione delle azioni più efficace e chiara, standard e senza ambiguità interpretative

Uno schema di flusso (o diagramma a blocchi o flow chart) è una rappresentazione grafica di un algoritmo realizzata mediante l'utilizzo di simboli, la cui forma dipende dal tipo di azione che si vuole descrivere, uniti da frecce che rappresentano il flusso dell'esecuzione delle istruzioni che compongono l'algoritmo

4. Gli schemi di flusso

Tabella 2 Principali simboli associati a ogni azione e loro significato

Tipo di istruzione	Simbolo	Significato
Azione		*Blocco di azione*: esegue l'azione descritta all'interno del rettangolo
Controllo (Condizionale)		*Blocco di controllo*: verifica la condizione e se il risultato è vero passa a eseguire le istruzioni sul ramo corrispondente a vero, altrimenti passa a eseguire le istruzioni sul ramo corrispondente a falso
Comunicazione (Trasmissione)		*Blocco di input dati*: chiede in ingresso all'utente un valore che verrà memorizzato in una variabile in memoria
		*Blocco di output dati*: fornisce in uscita all'utente un valore che verrà visualizzato a video
Salto		*Salto condizionato o incondizionato*: va a eseguire l'istruzione indirizzata dal flusso delle frecce. È rappresentato da una freccia che si innesta in un'altra freccia nel punto dell'algoritmo cui si deve saltare
Inizio algoritmo		*Blocco di Inizio algoritmo*
Fine algoritmo		*Blocco di Fine algoritmo*

Gli schemi di composizione fondamentali

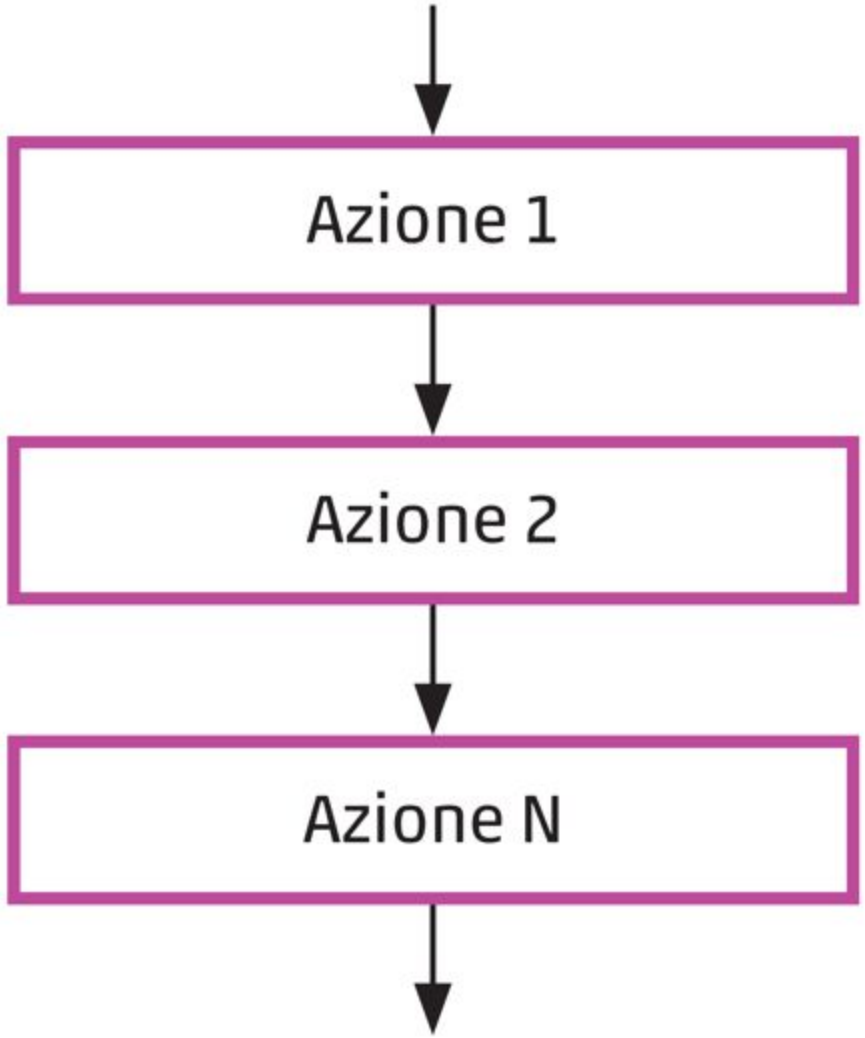
Gli schemi di composizione fondamentali (SCF) sono schemi di flusso che rappresentano le possibili situazioni che si possono incontrare nello sviluppo di algoritmi

Essi si distinguono in:

- **SCF di sequenza**
- **SCF di selezione**
- **SCF di ripetizione**

Lo SCF di sequenza

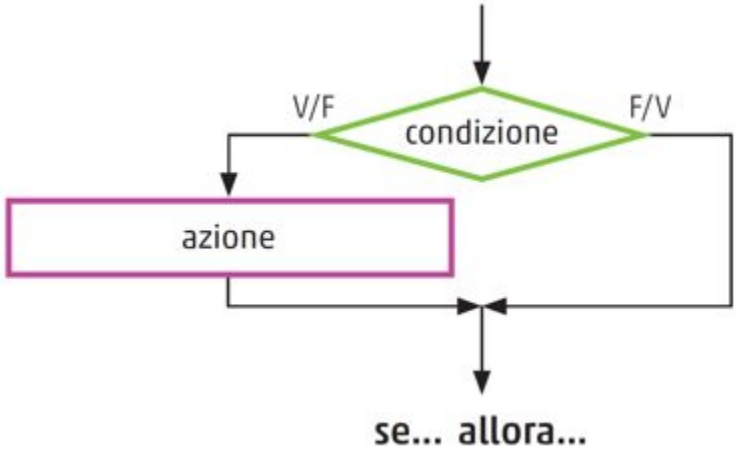
Rappresenta una serie di istruzioni di tipo azione che vengono eseguite in successione

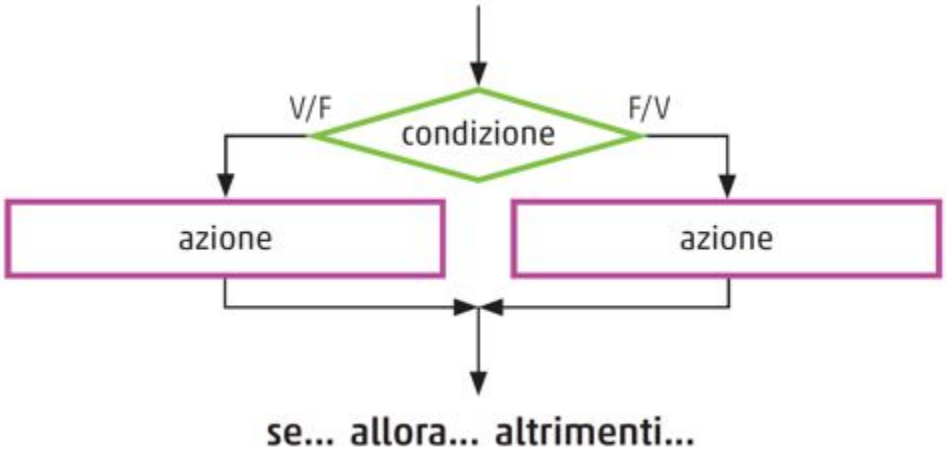


Lo SCF di selezione

Rappresenta la situazione in cui:

- si incontra un blocco di controllo
- a seconda del risultato della condizione che contiene, si decide se proseguire per una strada o per un'altra





Lo SCF di selezione permette di scegliere quali azioni dell'algoritmo eseguire a seconda del risultato fornitoci da una condizione, spostando il flusso dell'esecuzione verso le azioni corrispondenti al risultato Vero del test oppure verso le azioni corrispondenti al risultato Falso del test

Più SCF di selezione possono essere inseriti uno dentro l'altro formando una serie di **blocchi di controllo annidati**

5. Equivalenza tra algoritmi

Uno degli obiettivi fondamentali della programmazione consiste nel trovare l'**algoritmo risolutore ottimale**

Tuttavia, nella maggior parte dei casi, per un problema esistono più **algoritmi risolutori equivalenti**

> **Due o più algoritmi si dicono equivalenti se, pur usando metodi risolutori diversi, ricevuti gli stessi input, forniscono in uscita gli stessi output**

5. Equivalenza tra algoritmi

Analizziamo, per esempio, il seguente algoritmo nelle tre versioni risolutive equivalenti

Determinare il valore più grande tra tre valori numerici in ingresso al programma.

ANALISI

Dati in input a, b, c

Dati in output il valore più grande fra a, b, c

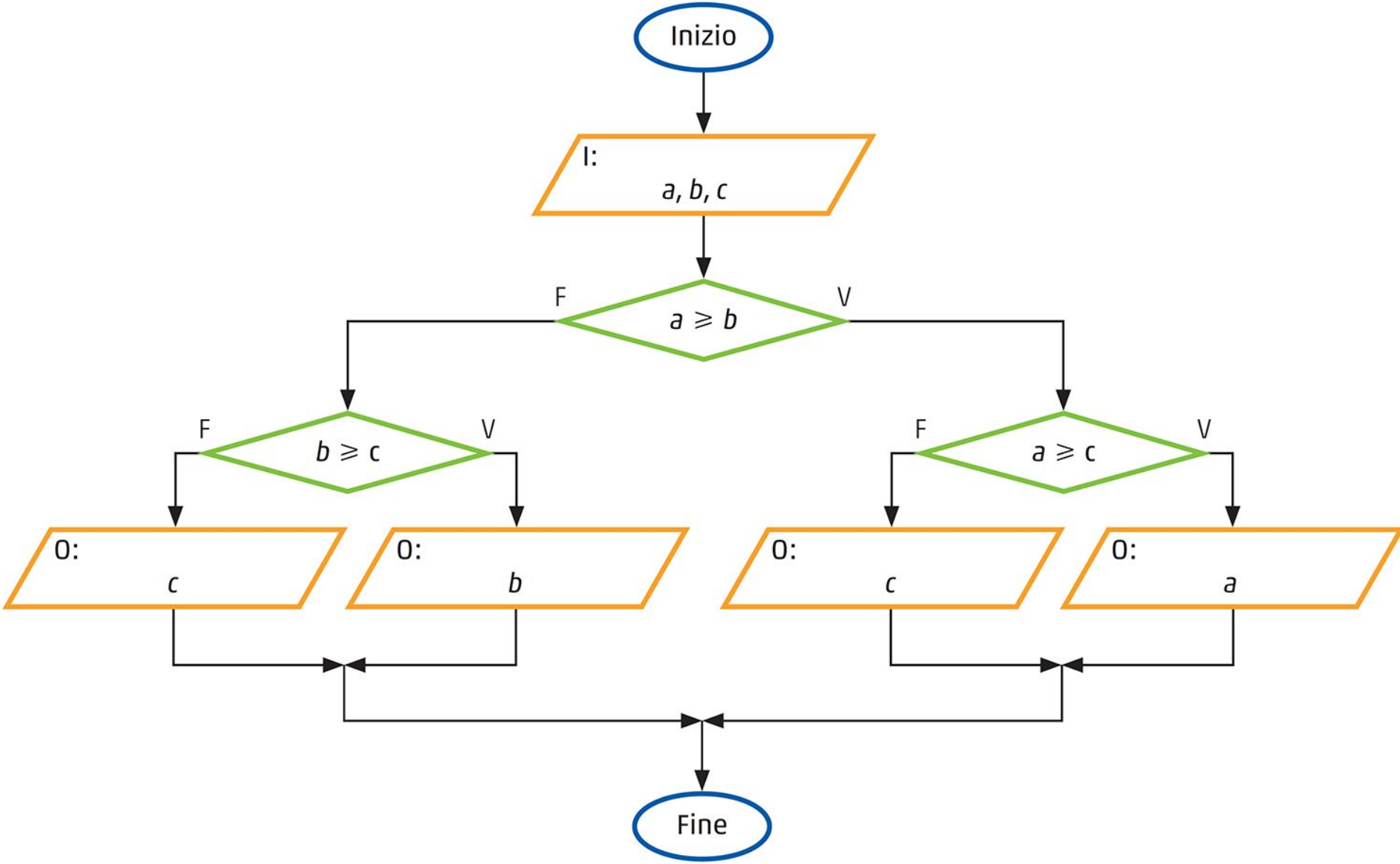
Relazione tra I/O

Se $a \geqslant b$ e $b \geqslant c$ allora il valore più grande è contenuto in a

altrimenti se $b \geqslant a$ e $b \geqslant c$ allora il valore più grande è contenuto in b

altrimenti se $c \geqslant a$ e $c \geqslant b$ allora il valore più grande è contenuto in c

5. Equivalenza tra algoritmi



Gli operatori booleani

Possiamo ottenere lo stesso risultato con meno blocchi di controllo mediante l'utilizzo degli **operatori booleani AND, OR, NOT**

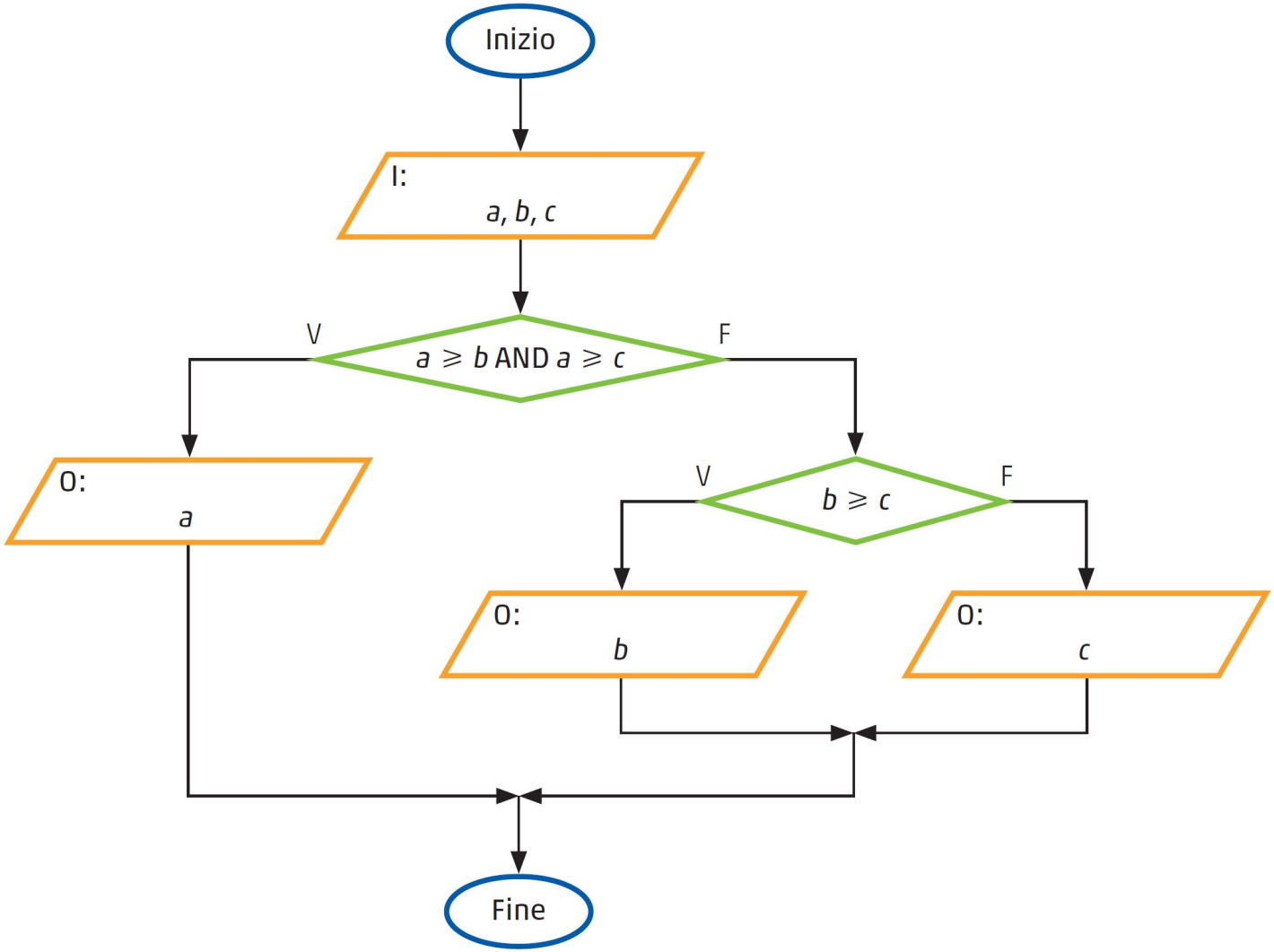
Le **tavole di verità** ci aiutano a capirne il funzionamento

C_1	C_2	C_1 AND C_2
Vero	Vero	Vero
Vero	Falso	Falso
Falso	Vero	Falso
Falso	Falso	Falso

C_1	C_2	C_1 OR C_2
Vero	Vero	Vero
Vero	Falso	Vero
Falso	Vero	Vero
Falso	Falso	Falso

C	NOT C
Vero	Falso
Falso	Vero

5. Equivalenza tra algoritmi



Introduzione di una variabile ausiliaria

Introducendo la variabile ausiliaria **max** è possibile semplificare le condizioni contenute nei blocchi di controllo