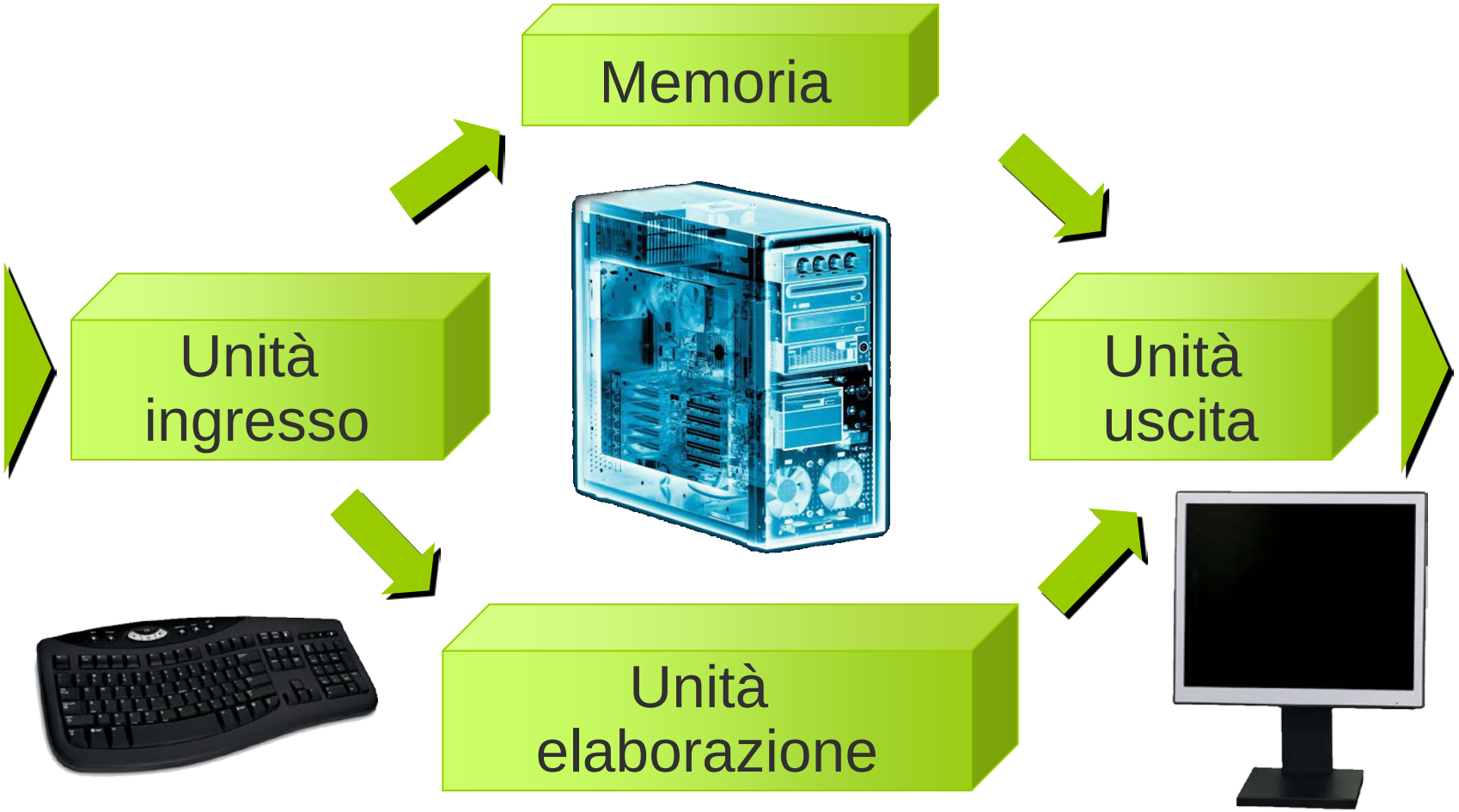
INFORMATICA

Introduzione alla Programmazione

Premessa: cos'è un calcolatore



Memoria centrale (RAM)

Memoria

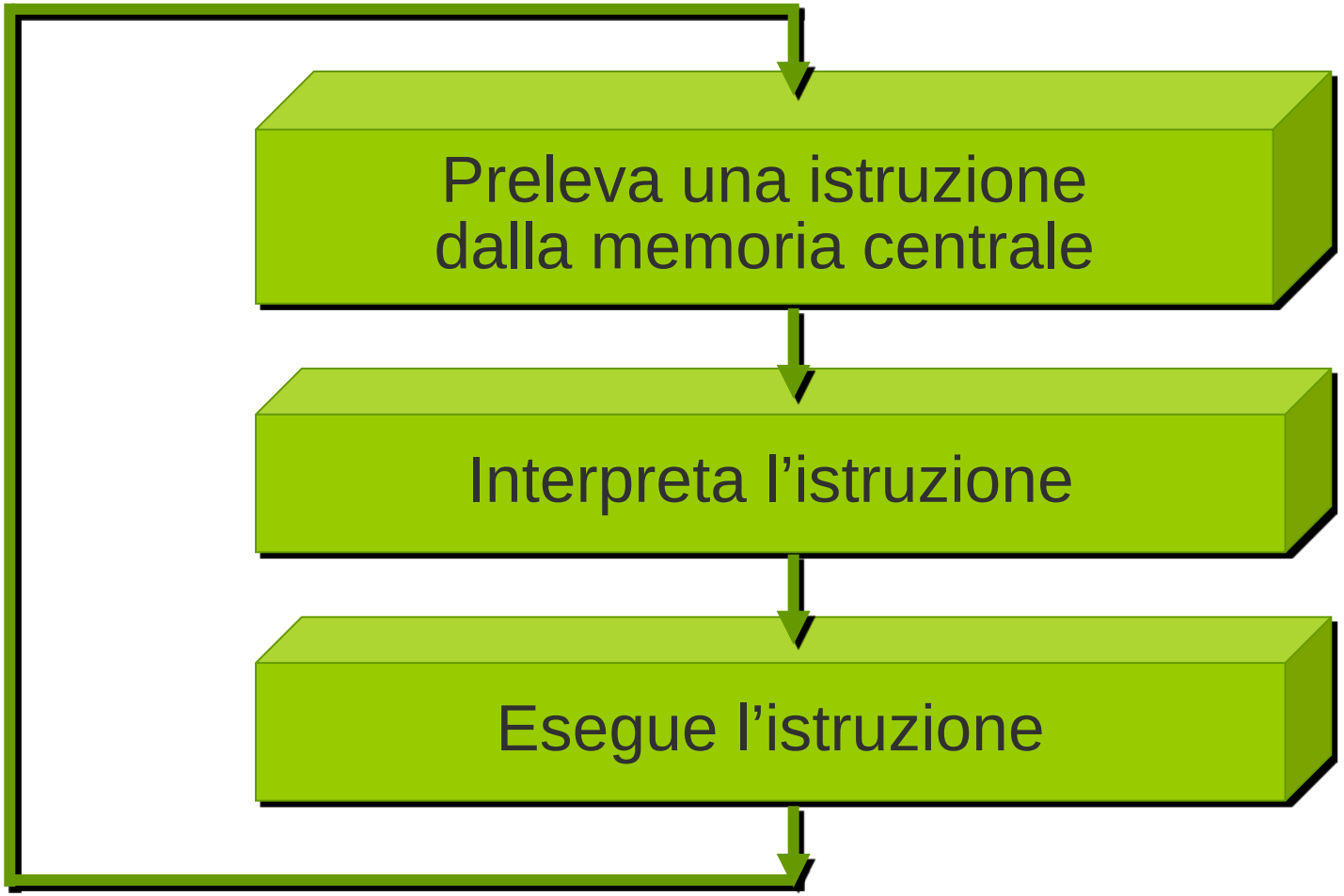
Memoria di massa (dischi)

Unità ingresso

Unità uscita

Unità elaborazione

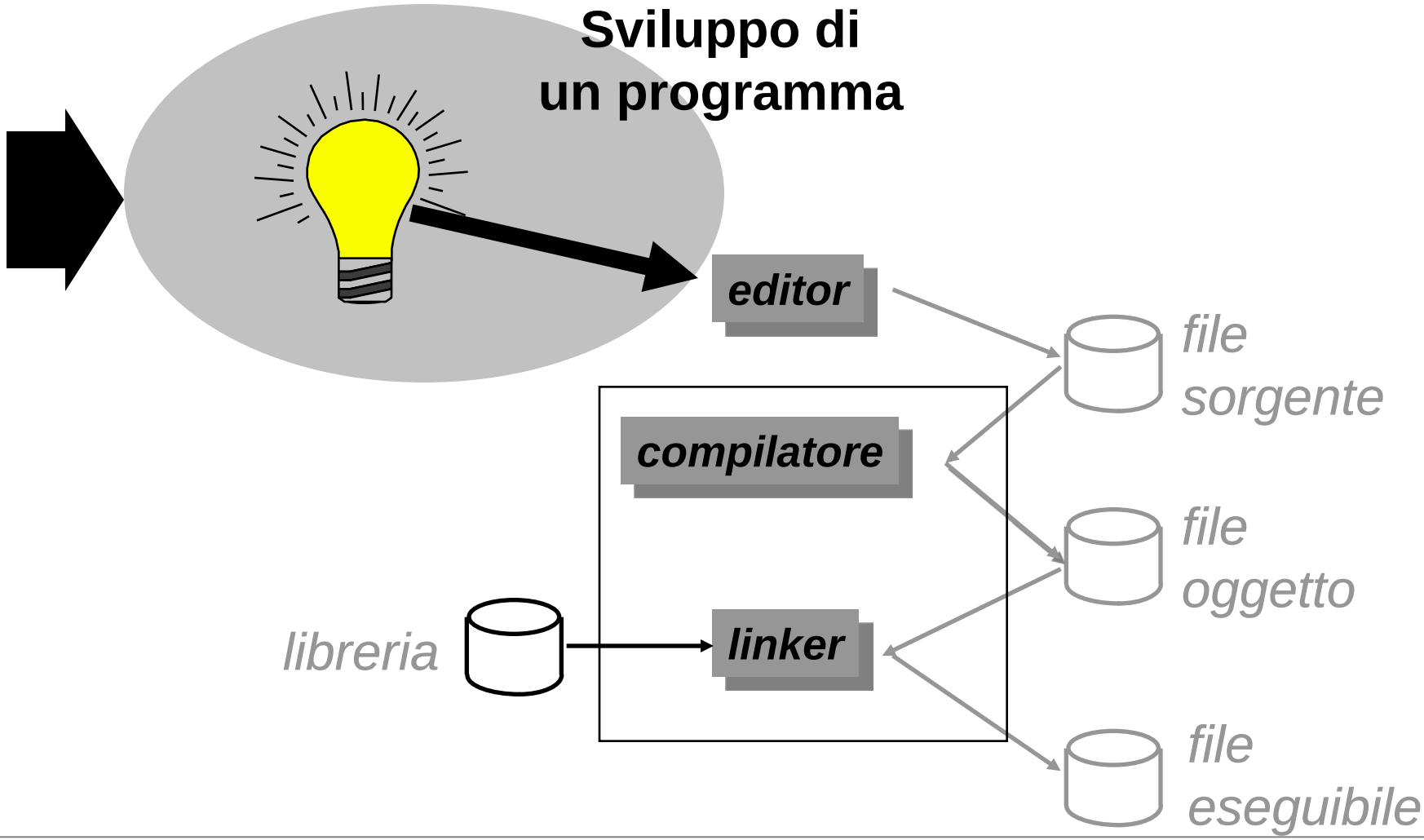
Cosa fa un calcolatore



In particolare

- Le istruzioni e i dati sono numeri binari (pacchetti di bit)
- Linguaggio macchina (elementare)
- Le istruzioni corrispondono ad azioni elementari (somma, differenza, prodotto, divisione e poco più)
- Problemi connessi:
  - Poter interagire con la macchina mediante un linguaggio meno elementare, più vicino agli umani.
  - Soluzione: linguaggi d'alto livello (C ed altri) + traduttori (compilatori)
  - Risolvere problemi complessi (controllo di una centrale atomica, produzione di queste slide, ecc.) mediante azioni elementari
  - Soluzione: tecniche di programmazione, algoritmi

Cosa impariamo in questo corso?



Cosa vuol dire “programmare”?

- La programmazione consiste nella scrittura di un testo, chiamato *programma sorgente*, che descrive in termini di istruzioni note alla macchina, la soluzione del problema in oggetto.
  - Esempio: *ricerca del valor massimo in una serie di numeri.*
- In generale non esiste una sola soluzione ad un certo problema; le soluzioni potrebbero essere numerose.
- La programmazione consiste proprio nel trovare la strada più efficiente che conduce alla soluzione del problema in oggetto.

Cosa vuol dire “programmare”?

- Programmare è un’operazione “creativa”! Si può dire infatti che non esiste un problema uguale a un altro! che non esistono soluzioni universali.
- In generale programmatori diversi scrivono programmi diversi per risolvere lo stesso problema: le soluzioni possono essere ugualmente efficienti!
- Programmare è un’operazione generalmente molto complessa, organizzata per *step successivi.*
- A parte i casi riferentesi ai problemi più banali è completamente inefficiente un approccio “diretto” (scrivere cioè direttamente il programma sorgente definitivo partendo dal testo del problema).

Cosa vuol dire “programmare”?

- Scrivere un programma significa *adattare la logica funzionale della macchina alle esigenze operative di un certo problema, secondo uno svolgimento sequenziale*.

- Le *esigenze operative* del problema scaturiscono dalla logica risolutiva idonea al conseguimento dei risultati voluti.

- La *logica funzionale* di macchina è l'insieme di operazioni elementari che questa è in grado di svolgere.

- Per *svolgimento sequenziale* si intende la successione nel tempo delle operazioni elementari.

Cosa vuol dire “programmare”?

- Per risolvere un problema con un elaboratore elettronico, conviene passare attraverso una serie di tappe intermedie, che condurranno alla fine alla scrittura del programma.

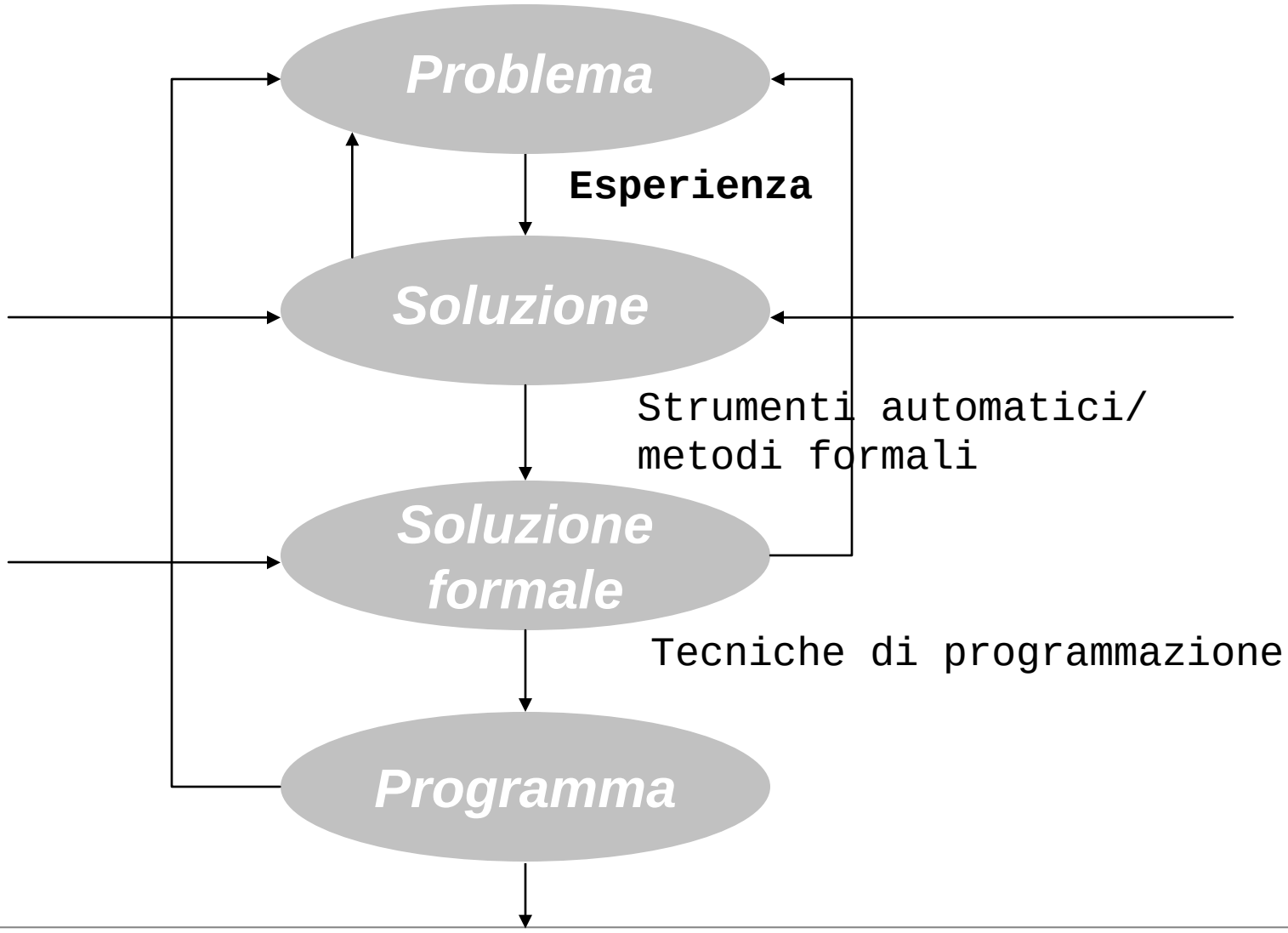
- La prima di queste tappe è la determinazione del meccanismo di soluzione del problema. Questo meccanismo che non ha ancora alcun legame con quello che sarà il programma finale è detto *algoritmo*.

- **Algoritmo** dunque, è una descrizione che specifica una serie di operazioni, eseguendo le quali è possibile risolvere un determinato problema.

Stadi di sviluppo di un programma

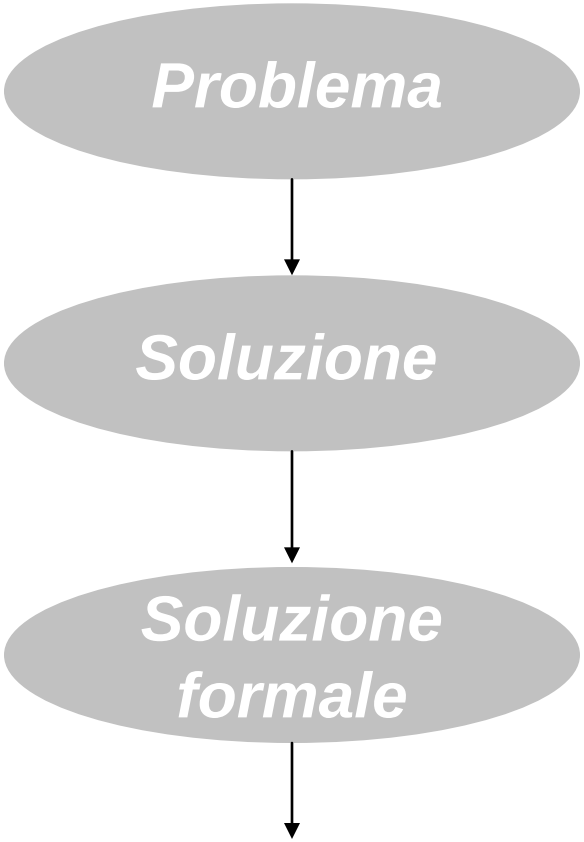
- Si deve risolvere un **problema**
- La prima fase consiste nella ricerca della soluzione migliore
- Successivamente si converte l'idea in una soluzione formale, ovvero in un testo che descrive la soluzione in modo formale (*algoritmo*)
- L'algoritmo viene tradotto in una sequenza di istruzioni comprensibili all'esecutore (in questo caso l'elaboratore elettronico): questa sequenza si chiama *programma*
- Il programma deve essere provato con un insieme significativo di dati per garantire che funzionerà in ogni occasione (qualsiasi siano i dati di input).
- Infine occorre documentare opportunamente il programma a beneficio di chi lo userà ed eventualmente lo modificherà.

Stadi di sviluppo di un programma



Esempio di flusso

- Problema: calcolo del massimo tra due valori A e B
- Soluzione: Il massimo è il più grande tra A e B...
- Soluzione formale:
  1. *max iniziale = 0*
  2. *se A >B allora max = A; stop*
  3. *altrimenti max = B; stop*



Difficoltà

- I punti critici nello sviluppo del programma risiedono essenzialmente:

  1. nello sviluppo di una soluzione informale

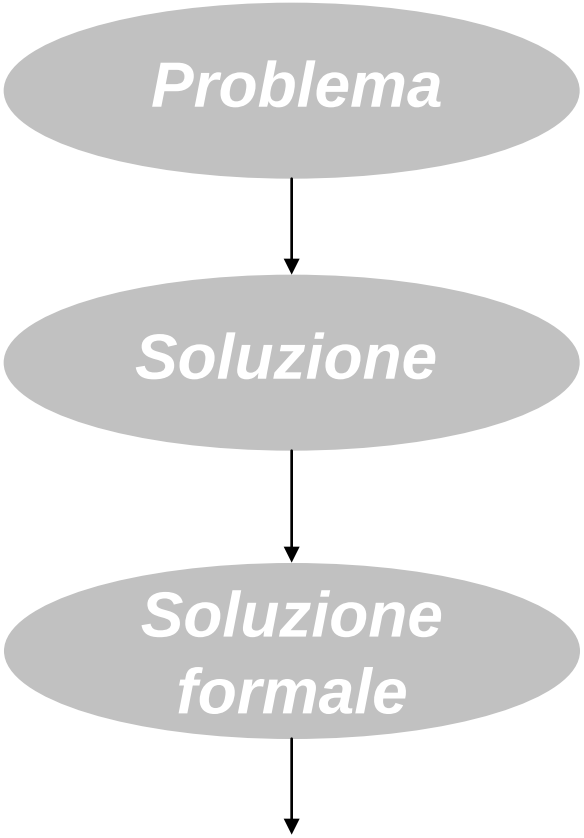
  2. nella formalizzazione della soluzione per rendere più semplice l'operazione di traduzione nelle regole (istruzioni) del linguaggio di programmazione.

- La formalizzazione della soluzione di un problema significa proprio sviluppare un *algoritmo*!

Esempio di flusso

- Problema: calcolo del massimo comun divisore (MCD) fra due valori.
- Soluzione: definizione di MCD!
- Soluzione formale:

  ???



Algoritmo

- **Algoritmo** è una *sequenza di operazioni* atte a risolvere un dato problema.

- L'esperienza quotidiana suggerisce infiniti esempi di algoritmi: oltre a una ricetta da cucina o alle istruzioni per l'uso di un elettrodomestico ne possiamo individuare moltissimi altri come le indicazioni per un lavoro a maglia, le regole per calcolare la somma o la moltiplicazione di due numeri, le regole per guidare un automobile da casa al Politecnico, ecc.

- Le *operazioni* che un algoritmo può prescrivere possono essere di natura assai differente, ma devono possedere, per poter essere considerate tali, delle caratteristiche ben precise.

Algoritmo

- Ogni operazione deve produrre, se eseguita, un *effetto osservabile* che possa essere descritto.

- Ad esempio, l'esecuzione dell'operazione di "*spostare il tavolo*" produce un effetto che può essere descritto specificando la disposizione dei mobili nella stanza prima e dopo lo spostamento.

- Invece, "*pensare* " o "*pensare al numero 5* " non possono essere considerate operazioni poiché non è affatto chiaro come il loro effetto possa essere "osservato" e descritto.

Algoritmo

- Ogni operazione deve produrre lo stesso effetto ogni volta che viene eseguita a partire dalle stesse condizioni iniziali (*determinismo*).

- Così se X vale 5 e Y vale 10, tutte le volte che eseguiamo la somma di X e Y con quei valori iniziali, il risultato dovrà essere sempre 15.

Algoritmo

- L'esecuzione di un algoritmo da parte di un “esecutore” - uomo o macchina - si traduce in una successione di operazioni che vengono effettuate nel tempo evocando un “*processo sequenziale* ”.

- Per “*processo sequenziale* ” s’intende una serie di eventi che occorrono uno dopo l'altro, ognuno con un inizio e una fine bene identificabile.

Algoritmo

- Talvolta il processo è fisso, cioè è sempre lo stesso ad ogni diversa esecuzione.

- Esempio: algoritmo per calcolare l'importo di una fattura:

  - *cerca l'aliquota IVA sulla tabella;*
  - *moltiplica l'importo netto per l'aliquota trovata;*
  - *somma il risultato all'importo netto.*

- Questo algoritmo è composto da tre *istruzioni*, che devono essere eseguite in sequenza.

Algoritmo

- Talvolta lo stesso algoritmo può evocare più processi sequenziali differenti, a seconda delle condizioni iniziali.

- Esempio: il precedente algoritmo, quando è richiesto di dover considerare la possibilità che la merce in esame non sia soggetta a IVA, diventa:

  - *SE la merce da fatturare è soggetta a IVA ALLORA*
  - *Cerca la corretta aliquota IVA sulla tabella e moltiplica l'importo per l'aliquota trovata. Somma il risultato all'importo netto.*
  - *ALTRIMENTI*
  - *Tieni conto solo dell'importo di partenza.*

Algoritmo

- In questo caso, il processo evocato non è fisso, ma dipende dai dati da elaborare, in particolare dal tipo di merce da fatturare: l'algoritmo descrive un insieme costituito da due sequenze di esecuzione diverse.

- Esistono situazioni, ancora più complesse, in cui il numero di sequenze d'esecuzione descritte da uno stesso algoritmo non solo non è noto a priori, ma addirittura può essere infinito.

Algoritmo

- Ad esempio, consideriamo l'algoritmo per effettuare una telefonata:

1. *solleva il ricevitore;*
2. *componi il numero;*
3. *SE qualcuno risponde ALLORA*
   i. *conduci la conversazione*
4. *ALTRIMENTI*
   i. *deponi il ricevitore e*
   ii. *RIPETI L'INTERO PROCEDIMENTO.*

Algoritmo

- L'algoritmo descrive infinite sequenze d'esecuzione, corrispondenti al fatto che la telefonata riesca al 1 , 2 , 3 , ... tentativo.

- Si potrà avere un processo ciclico che potrebbe non avere mai termine se l'interlocutore non risponde mai al telefono!

- Si può dire che *un algoritmo è un testo in grado di descrivere un insieme di sequenze di esecuzione*.

Algoritmo

- Supponiamo di voler specificare un algoritmo per effettuare il prodotto di due numeri interi col metodo delle addizioni successive. L'algoritmo potrebbe essere costituito semplicemente dal seguente testo:
  - *si sommi il moltiplicando a se stesso un numero di volte uguale al valore del moltiplicatore.*

- Specifichiamo con maggiore dettaglio le operazioni richieste evitando i costrutti ambigui. Ad esempio, potremmo scrivere:

  - *Si sommi il moltiplicando a se stesso, e si decrementi di uno il valore del moltiplicatore.*
  - *Si sommi ancora il moltiplicando al valore ottenuto dalla precedente somma, e si decrementi di nuovo il valore ottenuto dalla precedente sottrazione.*
  - *Si ripeta il procedimento fino a che, per decrementi successivi, non si raggiunga il valore uno.*

Algoritmo

- Per evitare ambiguità, si è stati costretti ad usare le espressioni:

  - "*valore ottenuto dalla precedente somma*"
  - "*valore ottenuto dalla precedente sottrazione*"

  per specificare, di volta in volta, di quale valore si sta parlando.

- Per distinguere tali valori si ricorre a *simboli* o *identificatori* per riferirsi senza ambiguità, di volta in volta, al valore desiderato.

Algoritmo

- Ad esempio:

1. *Si chiami M il valore del moltiplicando ed N il valore del moltiplicatore e sia M1 il risultato (inizialmente zero).*
2. *Si ripetano le seguenti operazioni fino a che il valore di N non diventi uguale a 0:*
   i. *si sommi il valore del moltiplicando M al valore di M1 e si chiami il risultato ancora M1;*
   ii. *si sottragga 1 dal valore di N, e si chiami il risultato ancora N.*
3. *Alla fine il valore di M1 è il risultato cercato.*

- I simboli M, N e M1 sono detti *identificatori di variabili*
- M1 è detto *accumulatore di risultato*

Formalizzazione della soluzione

- La differenza tra una soluzione informale e una formale sta nel modo di rappresentare un algoritmo:
  - Informale: *descrizione a parole*
  - Formale: *descrizione in termini di sequenza di operazioni elementari*

- Sono disponibili numerosi strumenti per rappresentare una soluzione in modo formale, i più utilizzati sono:
  - diagrammi di flusso (grafico)
  - pseudo-codice (testo)

Formalizzazione della soluzione

- Pseudocodice
  - Vantaggi
    - Immediato
  - Svantaggi
    - Meno astratto
    - Interpretazione più complicata

- Diagrammi di flusso
  - Vantaggi
    - Più intuitivi perchè grafici
    - Più astratti
  - Svantaggi
    - Richiedono apprendimento della funzione dei vari tipi di blocco

Diagrammi di flusso (flow-chart)

- Sono strumenti grafici che rappresentano l'evoluzione logica della risoluzione del problema

- Sono composti da:

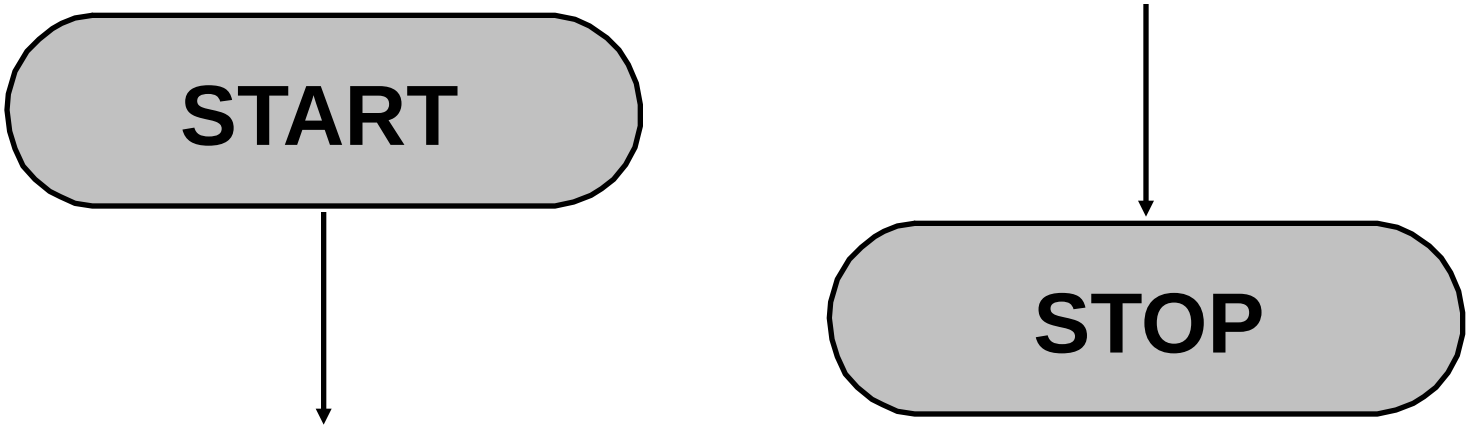
  - blocchi elementari per descrivere azioni e decisioni (esclusivamente di tipo binario)

  - archi orientati per collegare i vari blocchi e per descrivere la sequenza di svolgimento delle azioni

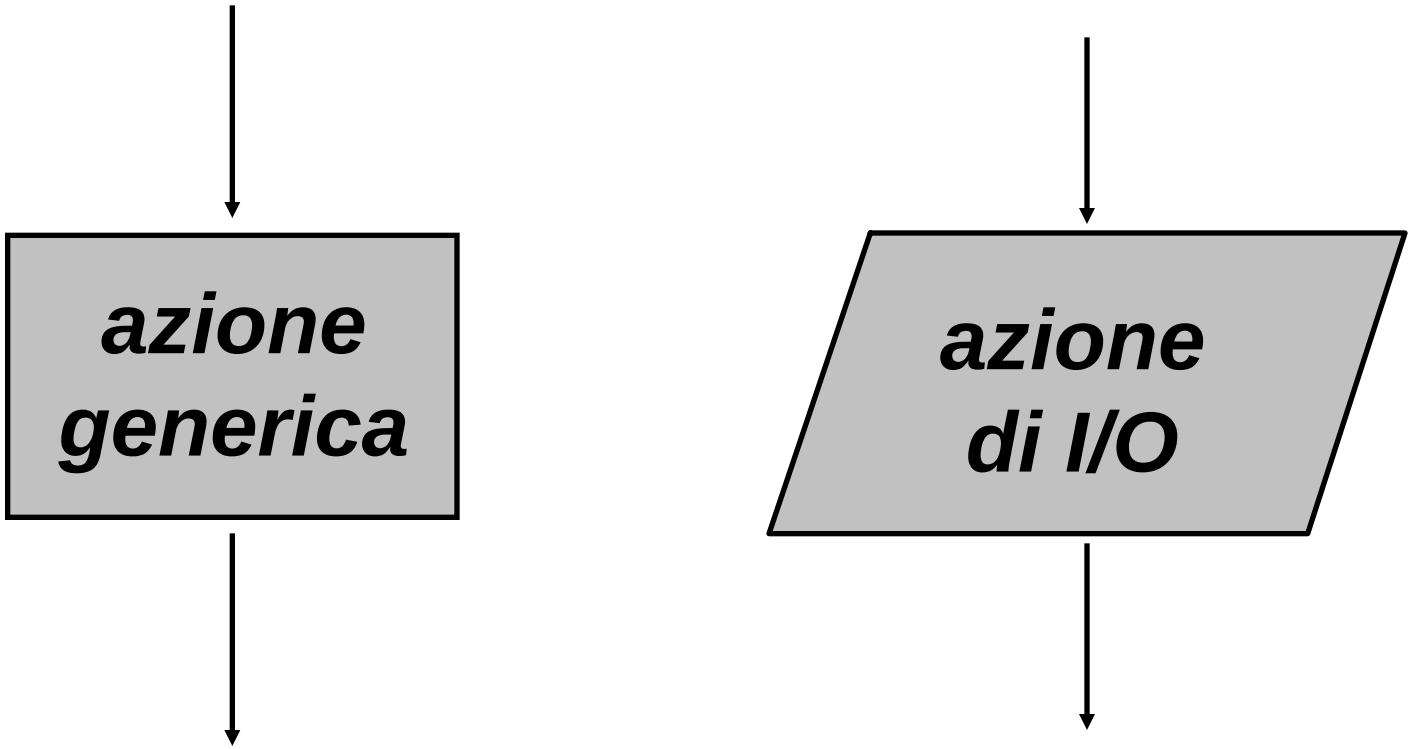
Diagrammi di flusso (flow-chart)

- I blocchi elementari sono:
  - Blocco di *Start*
  - Blocco di *Stop*
  - Blocco di *azione generica*
  - Blocco di *azione di I/O*
  - Blocco di *decisione binaria*
  - Blocco di *connessione*
  - (Simbolo di *richiamo di un blocco)*

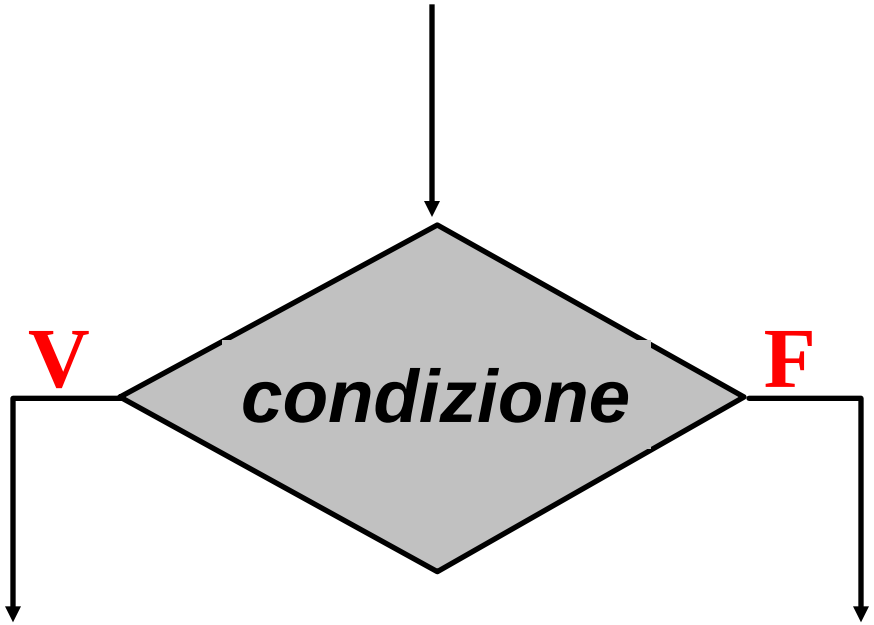
Flow-chart: Blocco di inizio/fine



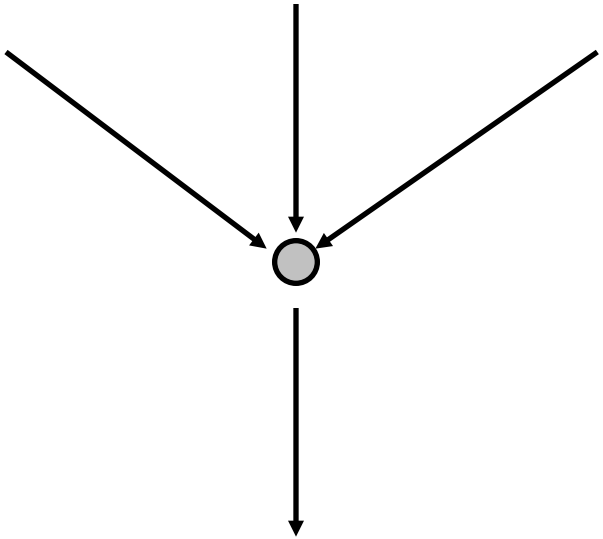
Flow-chart: Blocchi di azione



Flow-chart: Blocco di decisione

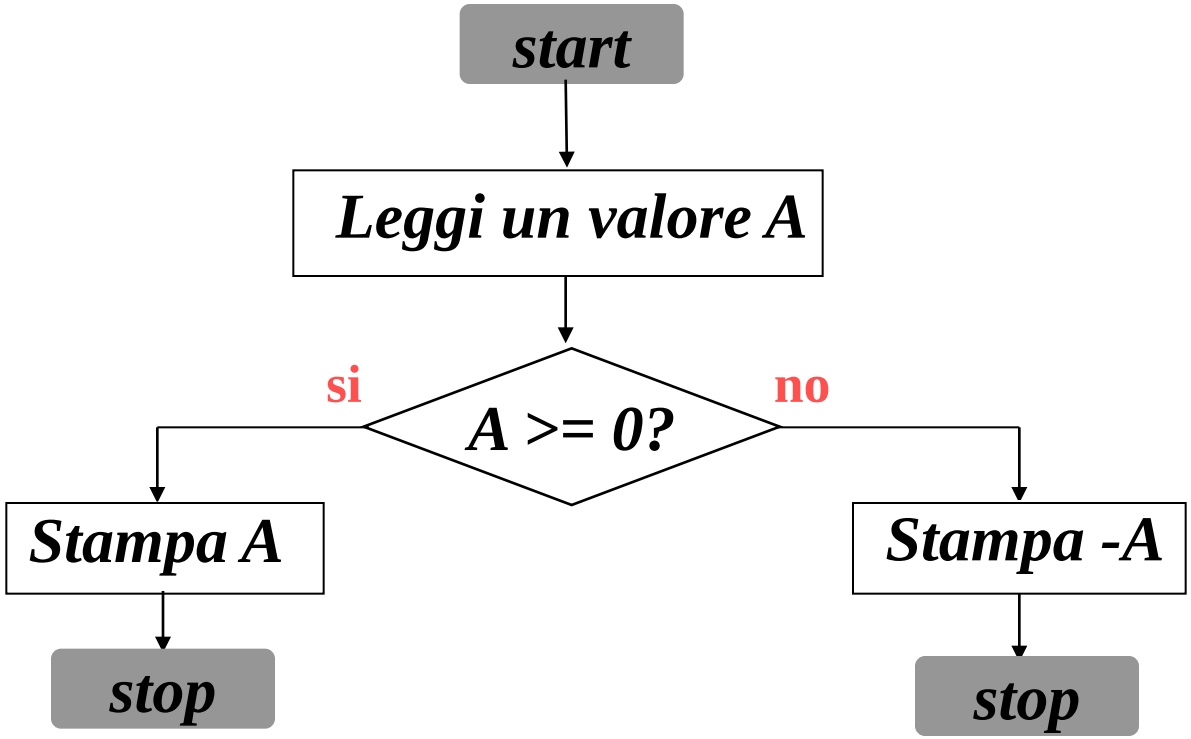


Flow-chart: Connettore



Esempio di flowchart

- Acquisire un valore A e visualizzarne il valore assoluto



Le etichette

- All'interno dei blocchi che costituiscono un flow-chart si indica l'azione che quel blocco rappresenta.
- Come si indica quest'azione?
- Normalmente si tratta di un testo il più sintetico possibile, ricorrendo anche a simboli mutuati altrove
- Ad esempio il simbolo:

A ← m

significa "riponi m nel contenitore etichettato A".

Costruzione di flow-chart

- Per mezzo dei flow-chart si possono rappresentare flussi logici complicati a piacere.

- E' però preferibile scrivere flow-chart *strutturati*, che seguano cioè le regole della programmazione strutturata. Così facendo si ottiene:

  - maggiore formalizzazione dei problemi
  - riusabilità del codice
  - maggiore leggibilità

Flow-chart strutturati

- Definizione formale: un flow-chart si dice strutturato quando è composto da *strutture elementari* indipendenti tra loro.

- Una *struttura elementare* è una composizione particolare di blocchi elementari avente un solo ingresso e una sola uscita.

- Un flow-chart strutturato è sempre riconducibile a un flow-chart elementare costituito da un'unica azione (ovvero un unico blocco).

- Differenza rispetto a un flow-chart non strutturato è dunque l'assenza di *salti incondizionati* all'interno del flusso logico del diagramma.

Diagrammi di flusso strutturati

- Un diagramma di flusso è strutturato se contiene solo un insieme predefinito delle seguenti *strutture elementari*:
  - uno ed *uno solo* blocco **START**
  - uno ed *uno solo* blocco **STOP**
  - blocchi di azione e/o di input-output
  - blocchi di tipo **IF - THEN - (ELSE)**
  - blocchi di tipo **WHILE - DO**
  - blocchi di tipo **REPEAT - UNTIL**

Teorema di Böhm - Jacopini

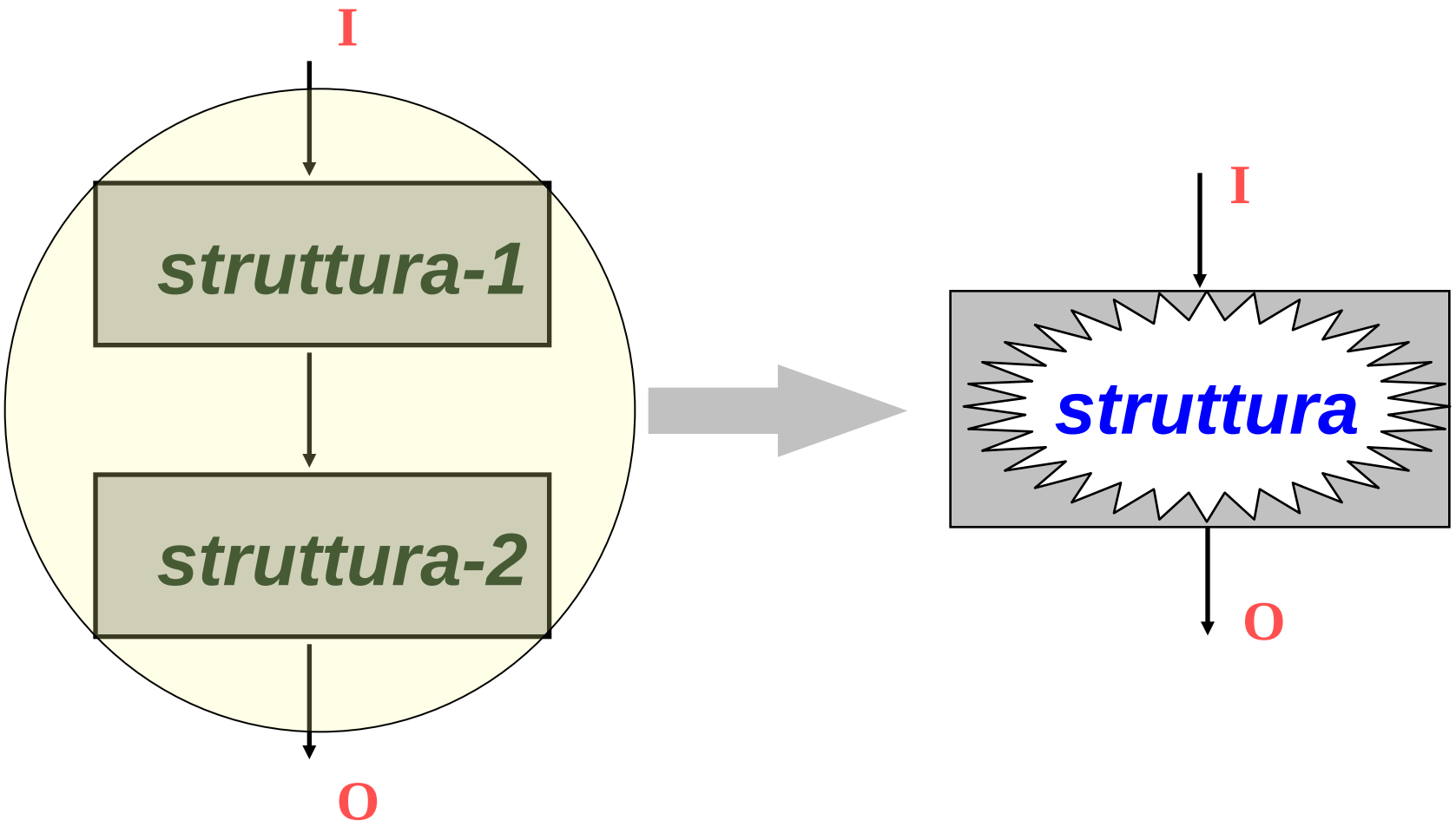
Qualunque diagramma di flusso è sempre trasformabile in un diagramma di flusso strutturato equivalente a quello dato.

- In altre parole, qualunque flusso logico può essere realizzato utilizzando solamente le due strutture di controllo:

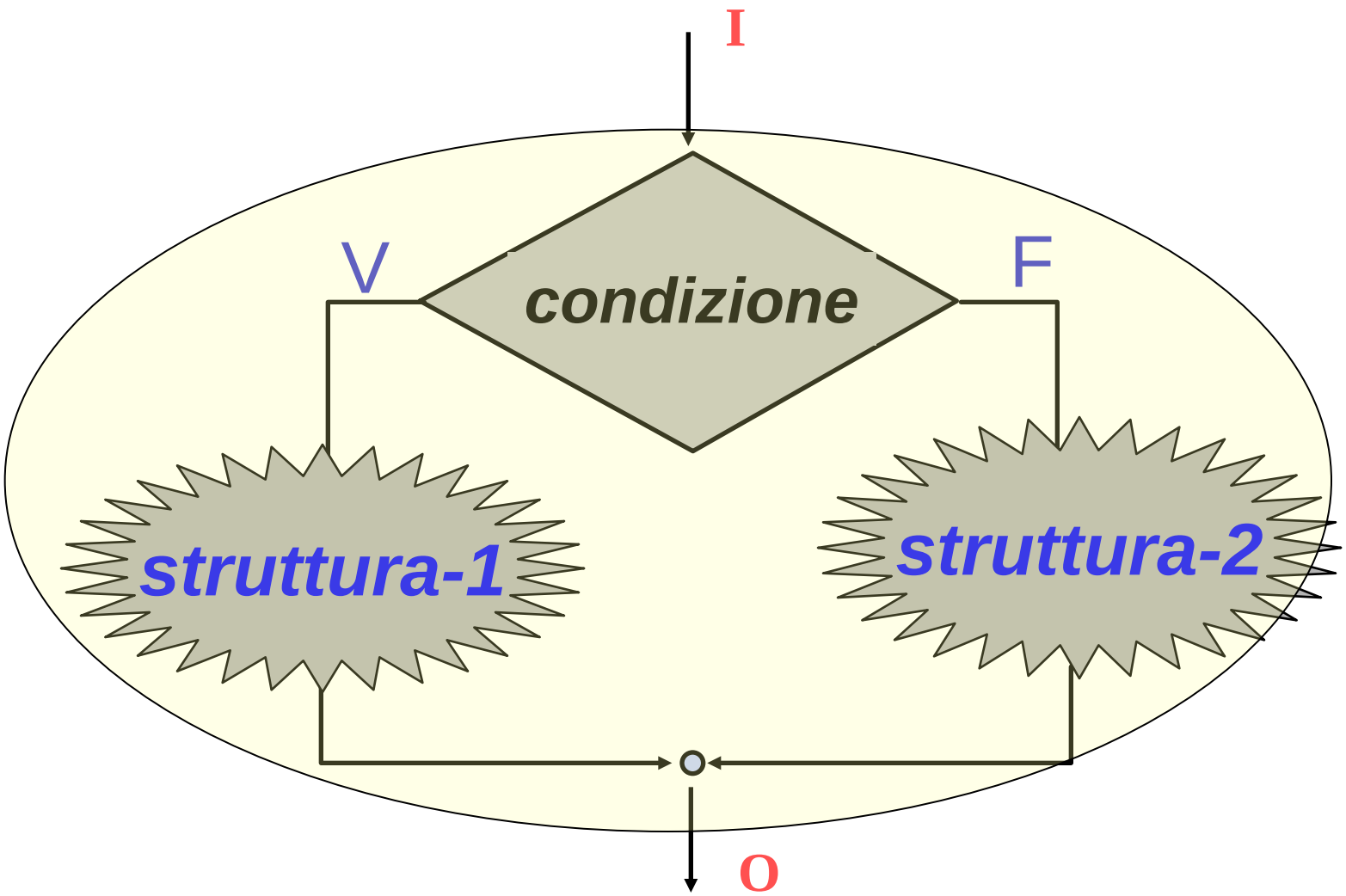
  - meccanismo di decisione binaria (if – then – else)
  - meccanismo di ripetizione (loop) (do – while / repeat – until)

  insieme ai blocchi di azione e di input/output

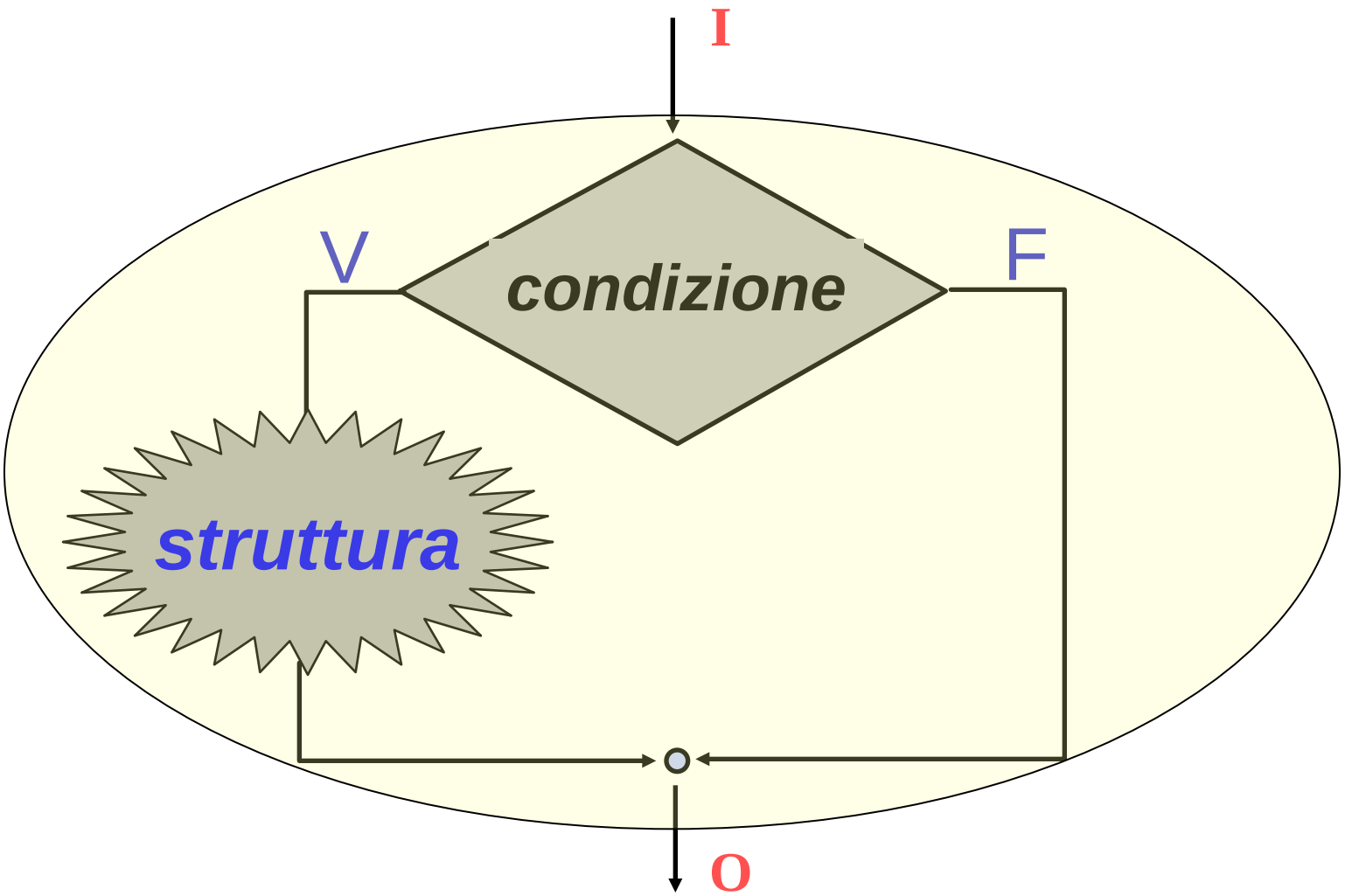
Sequenza



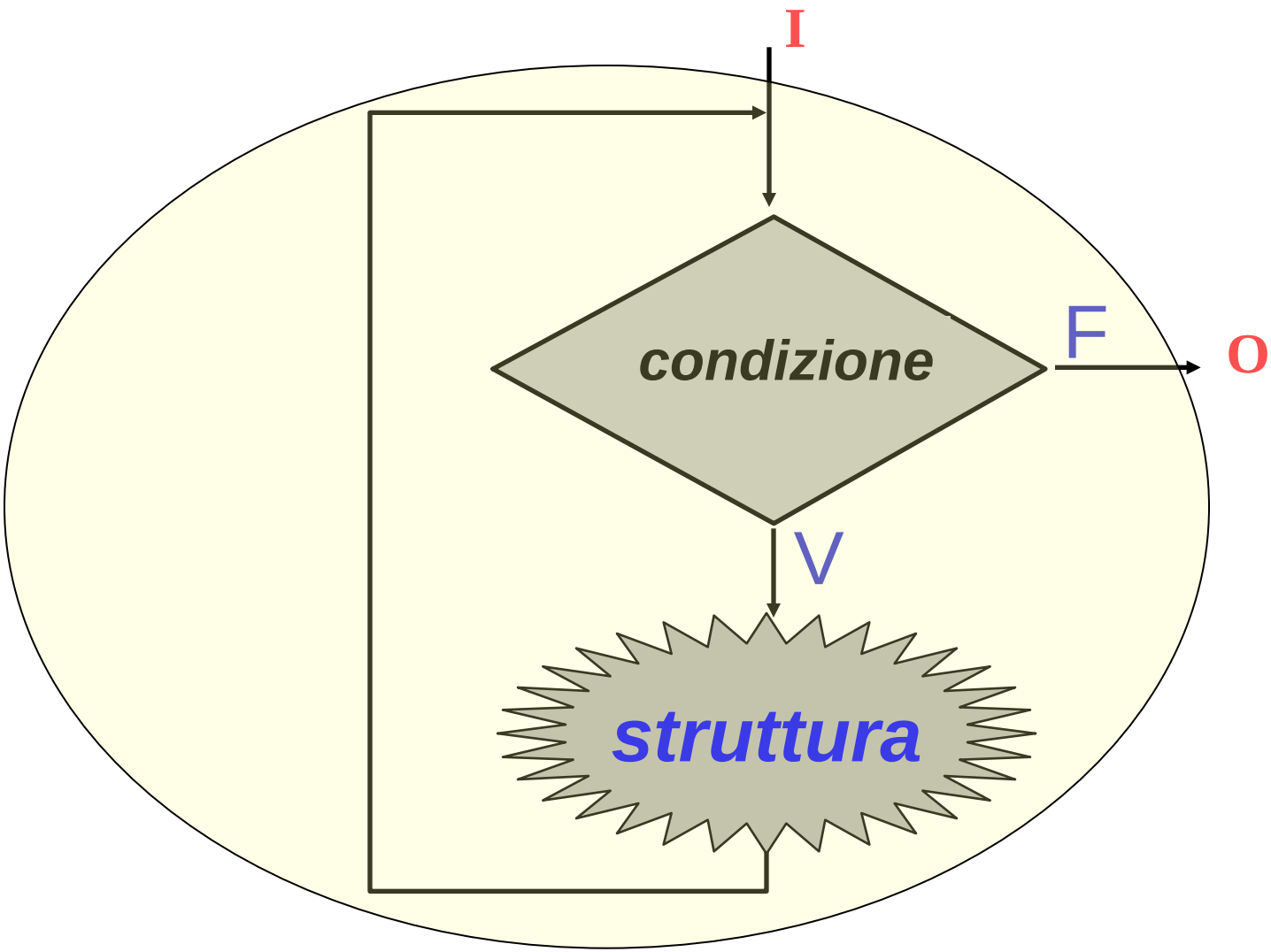
If-Then-Else



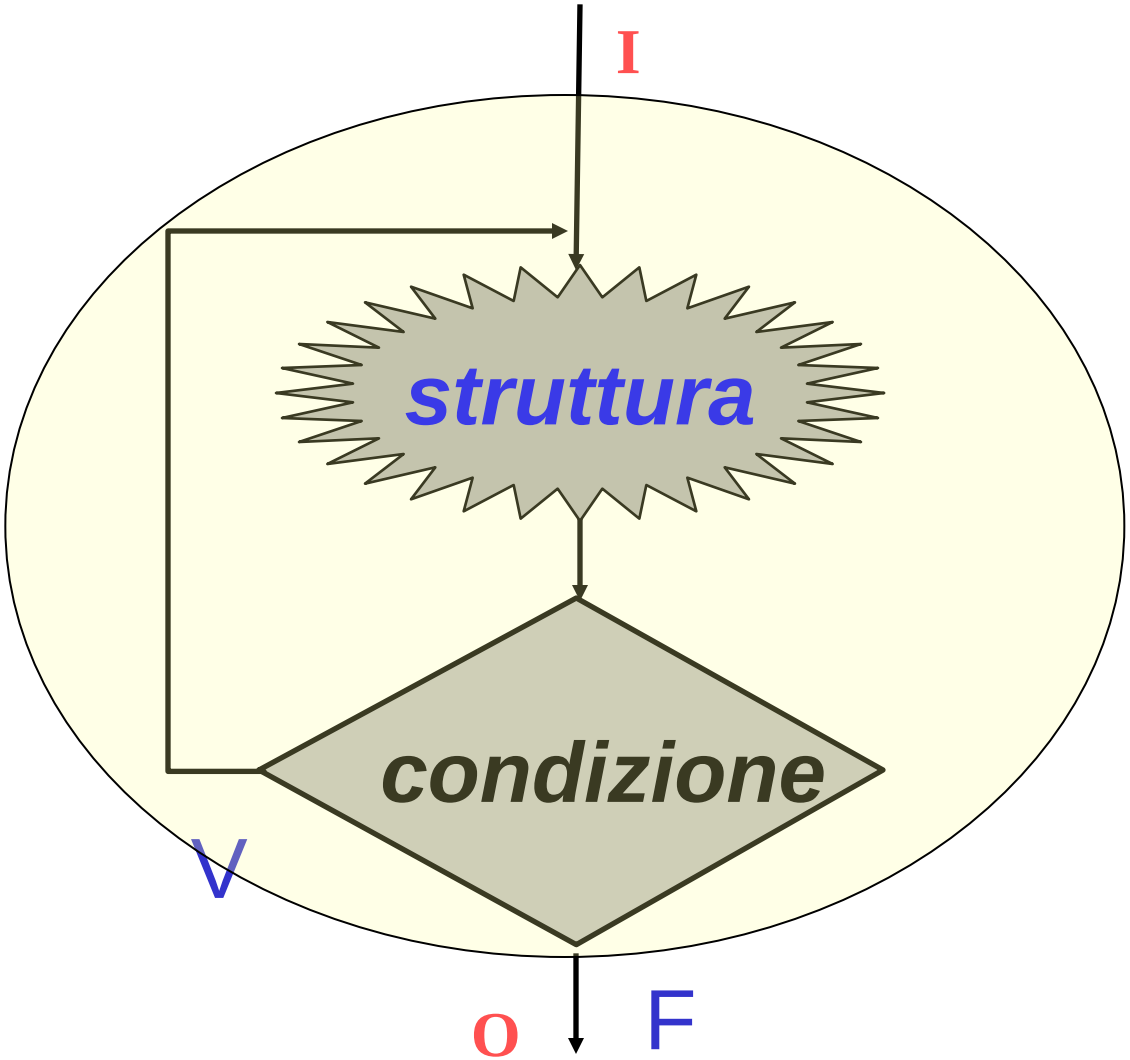
If-Then



While-Do



Repeat-Until



FOR

- Una derivazione interessante del costrutto DO WHILE è il *ciclo a contatore*, che ha la forma:

- *FOR variabile FROM valore_iniz TO valore_fin DO*
  - *sequenza di operazioni;*

- A *variabile* si attribuiscono tutti i valori compresi tra *valore_iniz* e *valore_fin* e, per ogni attribuzione, si esegue la sequenza di *istruzioni*.

- Se all'inizio *valore_iniz* è maggiore di *valore_fin*, la *sequenza di operazioni* non viene eseguita mai.

Algoritmi

Che cosa s'intende comunemente per algoritmo ?

"Un algoritmo è costituito da un insieme finito di passi distinti e non ambigui che, eseguiti a partire da assegnate condizioni iniziali, producono l'output corrispondente e terminano in un tempo finito".

La definizione precedente implica aver identificato uno o più “passi” che si è in grado di eseguire:

occorre disporre di una serie di operazioni (elementari o complesse) e di un “esecutore” capace di realizzare tali operazioni.

La descrizione di un algoritmo dipende dunque da quell'insieme di operazioni.

Per la soluzione degli esercizi proposti si suppone di disporre delle seguenti operazioni:

- somma / sottrazione / prodotto tra numeri interi o numeri reali: il risultato è rispettivamente un numero intero o un reale

- divisione tra numeri reali: il risultato è un numero reale
- divisione tra numeri interi: il risultato è la parte intera del quoziente (*quota*)
- modulo tra numeri interi: il risultato è il resto della divisione tra numeri interi

- confronto per >, ≥, =, <, ≤ tra numeri interi e numeri reali: il risultato è un valore logico, VERO o FALSO

- and ("*e insieme*") / or ("*oppure*") / not (*negazione logica*) tra valori logici: il risultato è ancora di tipo logico

Alcune altre operazioni su dati strutturati (accesso ad elementi di un vettore, ecc.) verranno introdotte in seguito.

Il metodo

I punti salienti del metodo adottato possono essere così riassunti:

- scomposizione dei problemi complessi in problemi più semplici (si suppone che siano noti un certo numero di problemi elementari di cui si conosce la soluzione);

- focalizzazione dell’attenzione: riconoscimento di uno o più “passi” che risultano risolutivi (la soluzione può essere ottenuta, ad esempio, ripetendo più volte lo stesso passo);

- minimizzazione delle differenze tra “descrizione attuale della soluzione” ed “obiettivo”; si noti che ciò implica individuare e tentare di risolvere dapprima il problema – o l’aspetto del problema – più rilevante, rimandando a fasi successive il raffinamento della soluzione;

- utilizzo delle analogie: se il problema che si sta affrontando può essere ricondotto alla *forma* di un problema che si è già risolto, si può riutilizzare lo stesso tipo di soluzione.

Esercizio 1

Realizzare il diagramma di flusso per il calcolo dell'area geometrica di un triangolo.

I. Leggi la *base*
II. Leggi l'*altezza*
III. Calcola *base* ∗ *altezza* / 2, risultato in *area_triangolo*
IV. Fine.

Esercizio 2

Realizzare un algoritmo per il calcolo della tabellina pitagorica di un numero.

Occorre identificare un *passo significativo*:

1 Calcola *numero* ∗ *fattore*, risultato in *prodotto*

Stabilisco *quante volte* devo ripetere l'operazione:

1 Con *fattore* che assume valori da 1 a 10

1.1 Calcola *numero* * *fattore*, risultato in *prodotto*

Completo con i dettagli:

1 Leggi *numero*
2 Con *fattore* che assume valori da 1 a 10
  2.1 Calcola *numero* * *fattore*, risultato in *prodotto*
  2.2 Visualizza *prodotto*
3 Fine

Esercizio 3

Realizzare un algoritmo per il calcolo della tabellina pitagorica dei numeri da 1 a 10.

Occorre identificare un *passo significativo*:

1 Calcola la tabellina di *numero* (operazione già nota)

Stabilisco *quante volte* devo ripetere l'operazione:

1 Con *numero* che assume valori da 1 a 10
1.1 Calcola la tabellina di *numero*

L’algoritmo completo è:

1 Con *numero* che assume valori da 1 a 10
  1.1 Con *fattore* che assume valori da 1 a 10
    1.1.1 Calcola *numero* * *fattore*, risultato in *prodotto*
    1.1.2 Visualizza *prodotto*
2 Fine

Esercizio 4

Trasformare il seguente algoritmo non strutturato in un equivalente algoritmo strutturato:

1 si pone 1 in *num_val* e 0 in *accum*
2 con *indice* che va da 1 a 100 si esegue
  2.1 si legge un dato intero in *varint*
  2.2 se *varint* < 0 si va a 3.
  2.3 si somma *varint* ad *accum*
  2.4 si incrementa *num_val*
3 si decrementa *num_val*
4 si visualizza *accum*
5 fine.

I. L’algoritmo contiene un ciclo a contatore basato su *`indice`*, ma dal ciclo si esce anche se si verifica l’evento “*`varint`* `< 0`”.

Occorre pertanto trasformare il ciclo a contatore in un ciclo basato su evento (ciclo `WHILE` o ciclo `REPEAT-UNTIL`).

Si esce dal ciclo se sono stati trattati tutti i dati (test su *`indice`*) oppure se il dato letto in *`varint`* è negativo.

Nell'adottare un ciclo basato su evento occorre invece determinare la condizione per continuare il ciclo:

si permane nel ciclo finché non sono stati trattati tutti i dati e allo stesso tempo *`varint`* è positivo o nullo.

1 si pone 1 in *num_val* e 0 in *accum*
2 si pone *indice* a 0, *varint* a 0
3 finché (*indice* < 100) e insieme
(*varint* ≥ 0) si esegue
3.1 si legge un dato intero in *varint*
3.2 se *varint* ≥ 0
3.2.1 si somma *varint* ad *accum*
3.2.2 si incrementa *num_val*
3.2.3 si incrementa *indice*
4 si decrementa *num_val*
5 si visualizza *accum*
6 fine.

All’interno del ciclo si è inserito un test su *varint* per ottenere un comportamento equivalente a quello dell’algo-ritmo specificato dal testo dell’esercizio.

Esercizio 5

Realizzare un algoritmo per calcolare la media aritmetica di una sequenza forniti da un dispositivo di input.

Analisi nel dominio della matematica:

- la media si calcola effettuando la somma di tutti i numeri e dividendo per la quantità di numeri;
- la divisione è lecita se il divisore non è nullo.

Identifico un *passo significativo*:

1 Leggi *numero*
2 Calcola *numero* + *accumulatore*, risultato in *accumulatore*
3 Incrementa *quanti_numeri*

Determino quante volte effettuare il *passo significativo*, tengo conto di dover effettuare le inizializzazioni e della condizione per effettuare correttamente la divisione.

L’algoritmo è:

1 Inizializza *accumulatore* e *quanti_numeri* con il valore 0
2 Finché ci sono numeri
  2.1 Leggi *numero*
  2.2 Calcola *numero* + *accumulatore*, risultato in *accumulatore*
  2.3 Incrementa *quanti_numeri*
3 Se *quanti_numeri* > 0
  3.1 Calcola *accumulatore* / *quanti_numeri*, risultato in *media*
4 Altrimenti
  4.1 Visualizza “media non calcolabile”
5 Fine.

Dal problema alla soluzione

- Realizziamo un algoritmo per la ricerca del massimo fra quattro numeri.
- Etichettiamo con *N1*, *N2*, *N3*, *N4* e *max* gli identificatori dei quattro numeri e del massimo che cerchiamo.
- Ipotizziamo dapprima che il numero più grande sia *N1* e lo mettiamo in *max.*
- Confrontiamo ora il contenuto di *max* con *N2* : se *N2* è più grande allora trascriviamo in *max* tale elemento, se invece *N2* è più piccolo non alteriamo il contenuto di *max*.
- Ripetiamo il procedimento con *N3* e *N4*.

Dal problema alla soluzione

- L'algoritmo risultante sarà del tipo:

1. Leggi N1, N2, N3, N4
2. *max* ← N1
3. Se N2 > *max* ALLORA
   - ✓ *max* ← N2
4. Se N3 > max ALLORA
   - ✓ *max* ← N3
5. Se N4 > max ALLORA
   - ✓ *max* ← N4
6. stampa "Il massimo è" *max*

Dal problema alla soluzione

- Si osservi come il precedente non sia un processo ottimizzato in quanto è valido solo per 4 elementi. Se vogliamo trovare il massimo fra 5 elementi occorrerà aggiungere un altro confronto.

- Quando si devono effettuare operazioni su elementi simili, cioè della stessa specie, conviene "strutturare" i dati, ad esempio in un "*vettore* " e utilizzare etichette con indici.

- Ogni *elemento del vettore* si individua mediante un indice che lo distingue dagli altri dello stesso tipo.

- Normalmente si rappresenteranno così: N[1], N[2] o, più genericamente, N[I], dove I funge da etichetta dell'indice.

Dal problema alla soluzione

- Usando questa notazione possiamo modificare il precedente diagramma di flusso.

1. Per I da 1 a 4
   - ❑ leggi N[I]
2. *max* ← N[1]
3. Per I che va da 2 a 4
   - ❑ Se N[I] > *max* ALLORA
     - ➢ *max* ← N[I]
4. stampa “Il massimo è” *max*

- I due algoritmi differiscono tra loro sostanzialmente: il primo procede direttamente dall'inizio alla fine; il secondo itera perché contiene un ciclo o *loop*.

Dal problema alla soluzione

- Meglio ancora generalizzare l'algoritmo rispetto al numero di elementi trattati. Ciò si può realizzare ad esempio indicando il numero di elementi come *parametro (variabile)*;

- Anziché 4 elementi si parlerà genericamente di *num_dati* elementi.

- Il valore di *num_dati* potrà essere stabilito di volta in volta oppure addirittura richiesto all'utente, per cui l'algoritmo diventa:

Dal problema alla soluzione

- 1 Leggi *num_dati*
- 2 I ← 1
- 3 Finché (While) I ≤ *num_dati*
- 3.1 leggi N[I]
- 3.2 incrementa I
- 4 *max* ← N[1]
- 5 I ← 2
- 6 Finché (While) I ≤ *num_dati*
- 6.1 Se (IF) N[I] > *max* ALLORA
- 6.1.1 *max* ← N[I]
- 6.2 incrementa I

7 stampa “Il massimo è” *max*

Il linguaggio di programmazione

Dalla soluzione al programma

- La scrittura del programma vero e proprio è praticamente immediata a partire dalla soluzione formale

- I linguaggi di programmazione forniscono infatti costrutti che realizzano:

  - Operazioni semplici
  - Strutture condizionali
  - Strutture iterative

Elementi del linguaggio

- Essendo il linguaggio un'astrazione, esistono alcuni fondamentali elementi sintattici essenziali per l'uso del linguaggio stesso:

  - parole chiave (*keyword*)
  - dati
  - identificatori
  - istruzioni

- Gli elementi sintattici definiscono la struttura formale di tutti i linguaggi di programmazione

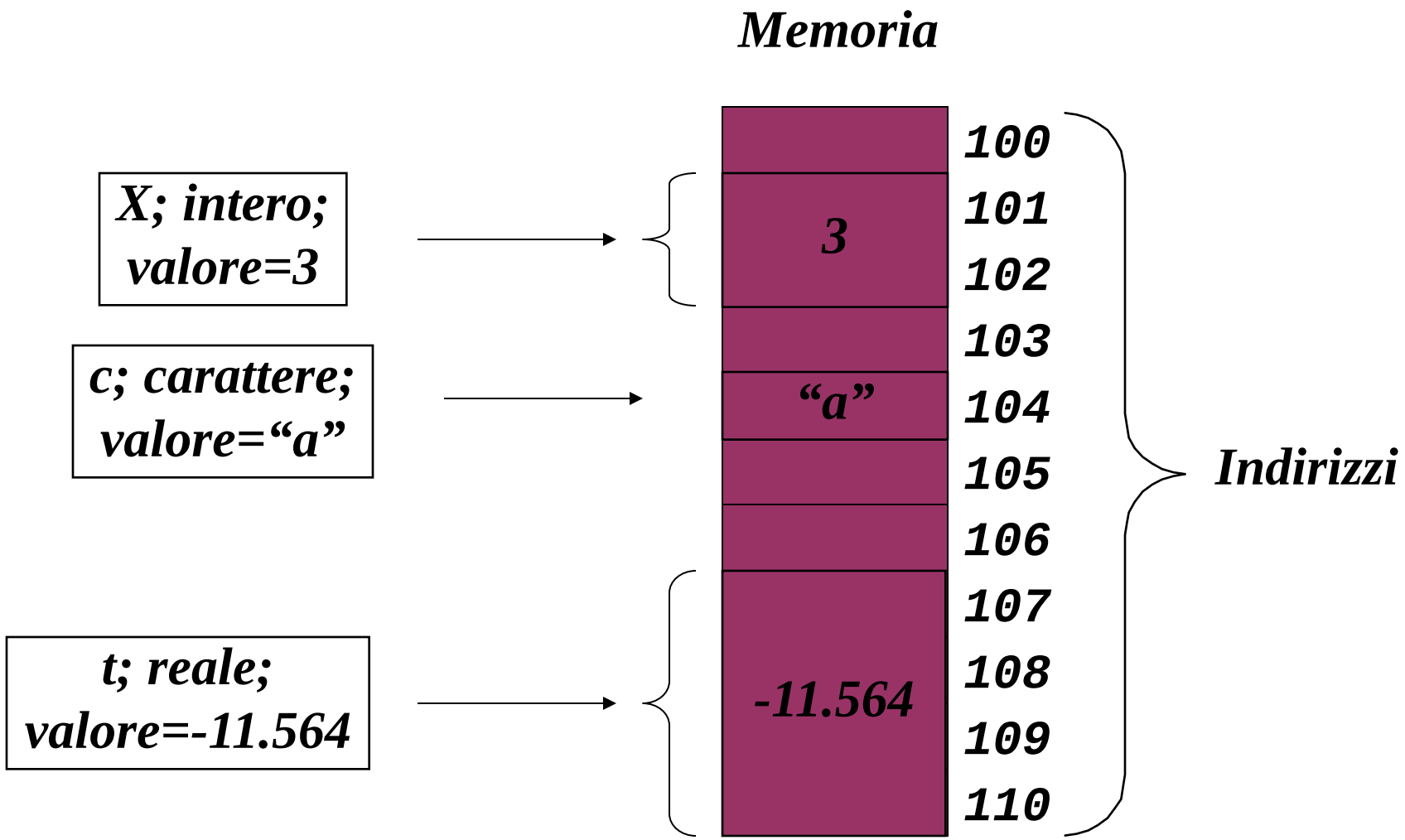
Parole chiave (keyword)

- Sono vocaboli “riservati” al traduttore per riconoscere elementi del linguaggio
  - Per esempio, le istruzioni sono tutte identificate da una keyword. Ad esempio la keyword PRINT in Basic identifica il comando di visualizzazione su schermo
- Non possono essere usate per altri scopi
- Costituiscono i “mattoni” della sintassi del linguaggio

Dati

- Dalla parte del calcolatore:
  - *un dato è un insieme di bit memorizzato in memoria centrale*
- Dalla parte dell'utente:
  - *un dato è una quantità associata ad un certo significato*
- Il linguaggio di programmazione supporta la vista utente
- Un **dato** è individuato da:
  - nome (*identificatore*)
  - rappresentazione (*tipo*)
  - modalità di accesso (*variabile, costante*)

Astrazione dei dati



Identificatore

- Indica il nome di un **dato** (e di altre entità) in un programma
- Permette di battezzare in modo intuitivo i dati
- Esempio:
  - `somma, volume, diametro`, …
- Il nome deve ovviamente essere unico all'interno del suo ambiente di esistenza (spesso questo ambiente coincide con l'intero programma)

Tipo

- Stabilisce il codice col quale i dati sono rappresentati in memoria (numeri interi in *complemento a 2,* numeri frazionari in *floating-point,* caratteri in *ASCII,* ecc.).
- Stabilisce lo *spazio* occupato da un dato nella memoria del calcolatore (numero di celle di memoria).
- Definisce la *capienza* di quel dato (l'intervallo di valori che può assumere).
- Definisce quali *operatori* possono agire su quel dato.

Tipo di accesso

- Indica le modalità di accesso ai dati:
  - Variabili
    - sono modificabili in qualunque punto del programma
  - Costanti
    - sono dati a sola lettura
    - il valore è assegnato una volta per tutte all'inizio del programma e non è modificabile

Istruzioni

- Sono le operazioni che il linguaggio è in grado di eseguire; possono essere suddivise in:

  - *pseudoistruzioni* : non sono vere e proprie istruzioni perché non sono eseguibili. Si tratta di direttive i cui effetti sono visibili durante l'esecuzione delle istruzioni eseguibili

  - *istruzioni elementari* : sono operazioni più o meno riconducibili a operazioni dirette sull'hardware della macchina quali operazioni di input/output, lettura di dati dalla memoria, ecc.

  - *istruzioni di controllo del flusso* : permettono di eseguire strutture complesse quali if-then-else, while-do, repeat-until

Esempio di programma

```
PROGRAM demo;
// definizione di istruzione
CONSTANTS
  pig = 3.14159;
  val = 0.19;
VARIABLES
  x: INTEGER;
  y: REAL;
  c: CHARACTER;
BEGIN PROGRAM
  x = 2;
  IF (y > 2) THEN y = x * pig;
  PRINT x, y;
END PROGRAM
```

pseudoistruzioni

specifica di tipo

istr. elementare

istr. elementare

istr. di controllo del flusso

istruzioni eseguibili

Linguaggio di programmazione

- Imparare un linguaggio significa conoscere
  - le keyword
  - i tipi predefiniti
  - le istruzioni e la loro sintassi

- ATTENZIONE: conoscere un linguaggio non significa saper programmare!

- In questo corso impareremo il linguaggio C

- Imparato un linguaggio è semplice e immediato trasferire i concetti ad altri linguaggi!