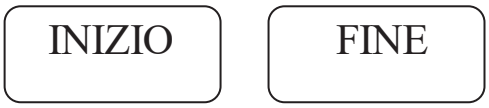
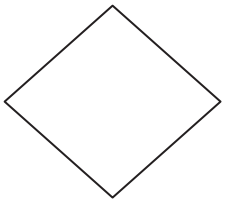
RAPPRESENTAZIONE DI ALGORITMI MEDIANTE DIAGRAMMI DI FLUSSO (FLOW-CHART) O DIAGRAMMI A BLOCCHI



Inizio e fine esecuzione

Operazione di input/output

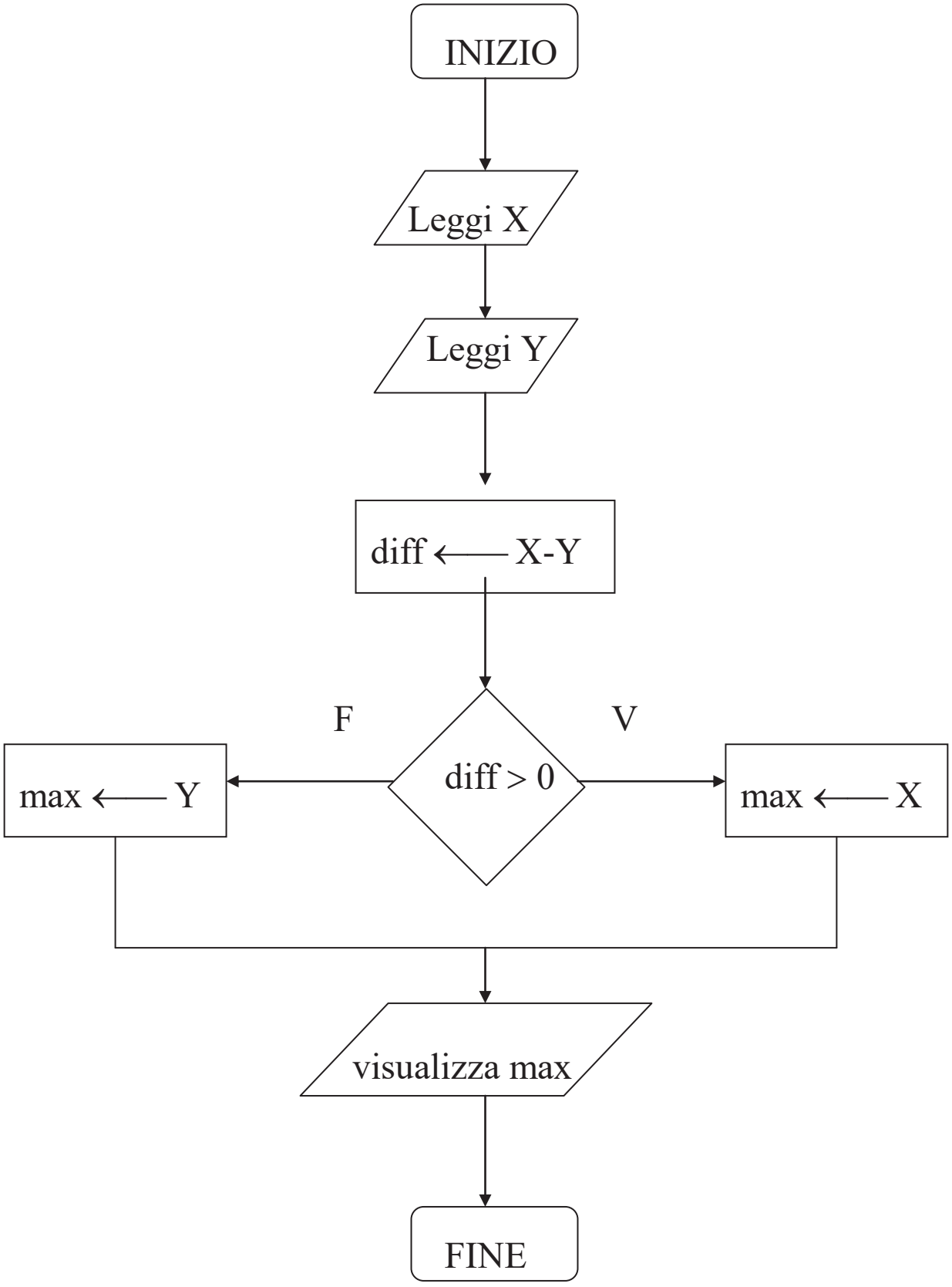
Operazioni da compiere in sequenza



Operazione di test

Le frecce indicano l'ordine di esecuzione delle istruzioni

FLOW CHART DEL PROBLEMA 1



Esercizio 1: calcolare il valore assoluto di un numero assegnato

Il problema può dunque essere formulato matematicamente:

$$|X| = \begin{cases} X & \text{se } X \geq 0 \\ -X & \text{se } X < 0 \end{cases}$$

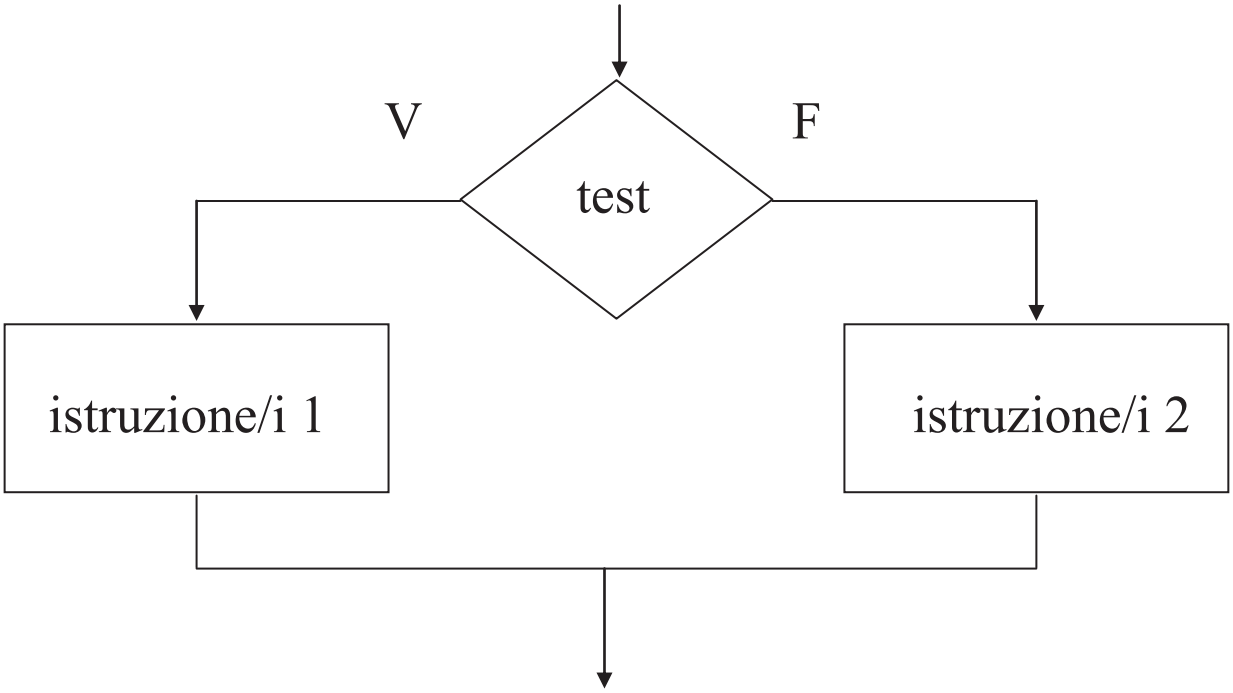
Supponiamo che l'esecutore sia in grado di verificare il segno di un numero e sia in grado di calcolare i prodotti.

Passo	**Istruzione**
P1:	Leggi X
P2:	Se X > 0 allora
	val_assoluto ⟵ X
	altrimenti
	val_assoluto⟵ X*(–1)
P3:	Visualizza val_assoluto

ISTRUZIONI CONDIZIONALI

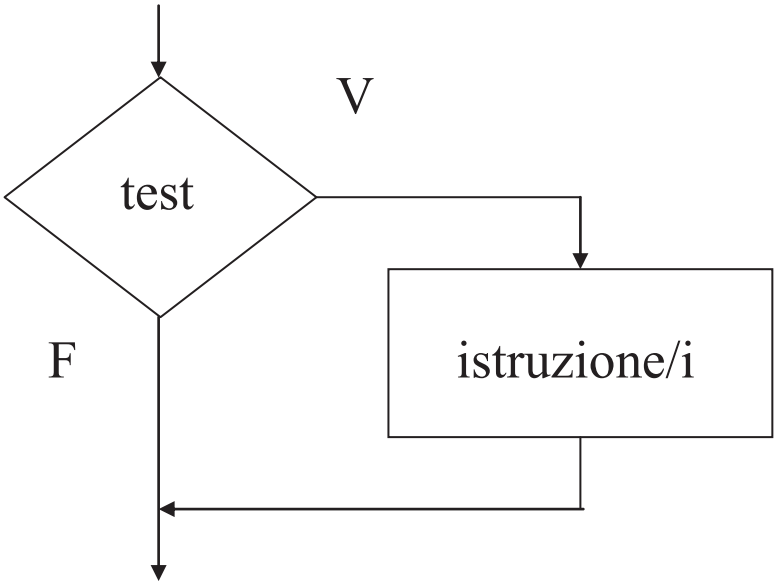
Se test è vero allora
    istruzione/i 1
altrimenti
    istruzione/i 2

If condizione *Then*
    istruzione/i
Else
    istruzione/i
End If



Se test è vero allora
    istruzione/i

If condizione *Then*
    istruzione/i
End If



Esercizio 2: riscrivere l'algoritmo per il calcolo del valore assoluto supponendo di disporre soltanto dell'istruzione condizionale If…Then…

Passo	**Istruzione**
P1:	Leggi X
P2:	val_assoluto ⟵ X
P3:	Se X < 0 allora
	val_assoluto ⟵ val_assoluto*(−1)
P4:	Visualizza val_assoluto

oppure, risparmiando una variabile:

Passo	**Istruzione**
P1:	Leggi X
P2:	Se X < 0 allora
	X⟵X*(−1)
P3:	Visualizza X

SUDDIVISIONE IN SOTTOPROBLEMI

L'algoritmo è costituito da una sequenza di *sottoproblemi elementari* (le istruzioni) che devono essere *comprensibili* e *direttamente eseguibili* dall'esecutore.

Problemi molto complessi possono richiedere per la loro soluzione una sequenza di sottoproblemi elementari particolarmente elevata e tali da rendere il relativo algoritmo poco maneggevole, per cui si potrebbero avere difficoltà

- nel seguire il ragionamento
- nel trovare gli eventuali errori logici
- nell'apportare modifiche in un momento successivo alla stesura dell'algoritmo stesso.

In questi casi è opportuno semplificare l'algoritmo arrestando la scomposizione del problema in *sottoproblemi terminali* anche non elementari (cioè non direttamente eseguibili) ma dei quali si conosce la soluzione.

Algoritmi	Sottoproblemi terminali elementari	Istruzioni direttamente eseguibili
	Sottoproblemi terminali non elementari	Per essere eseguibili vanno scomposti in sottoproblemi terminali elementari

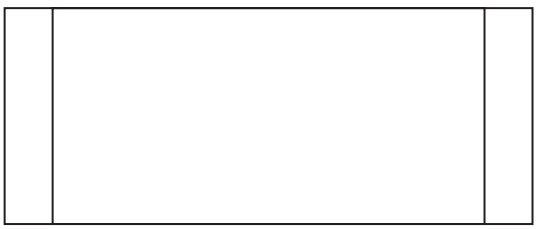
Problema 2: determinare il massimo tra tre numeri reali X, Y, Z

Per risolvere questo problema si può utilizzare l'algoritmo per il sottoproblema elementare di ricerca del massimo tra due numeri.

P1: Leggi X
P2: Leggi Y
P3: Leggi Z
P4: Ricerca il massimo tra X ed Y e metti il risultato in M
P5: Ricerca il massimo tra M e Z e metti il risultato in maximum
P6: Scrivi maximum

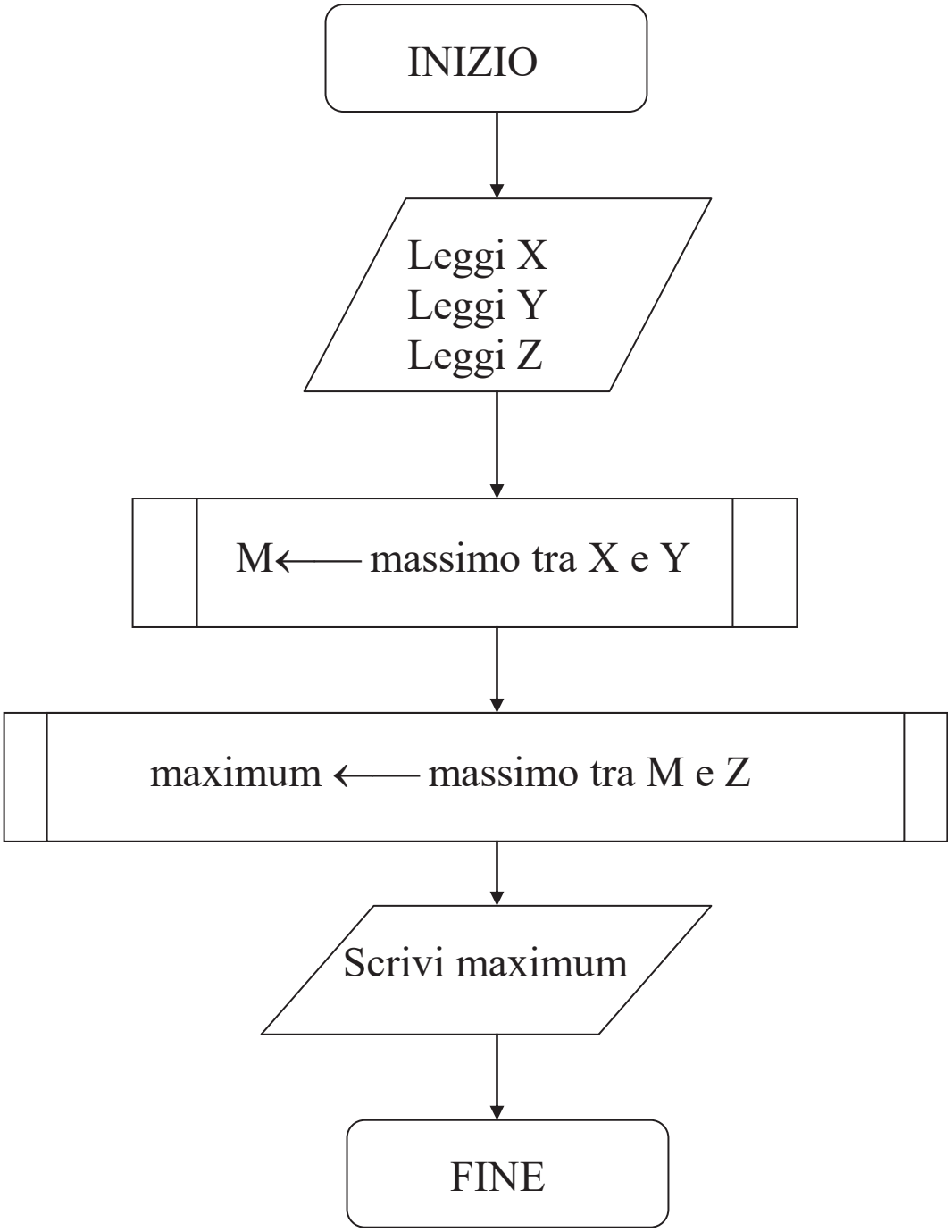
I passi P4 e P5 corrispondono alla risoluzione di sottoproblemi terminali non elementari. Pertanto, affinché questo algoritmo sia eseguibile dall'esecutore, al posto di P4 e P5 si devono considerare i passi degli algoritmi che consentono di risolvere tali sottoproblemi.

Quando un algoritmo viene tradotto in un programma per venir eseguito da un calcolatore, i sottoproblemi terminali elementari corrispondono alle istruzioni del programma, mentre ai sottoproblemi terminali non elementari corrispondono *sottoprogrammi*, a loro volta costituiti da istruzioni o sottoprogrammi.



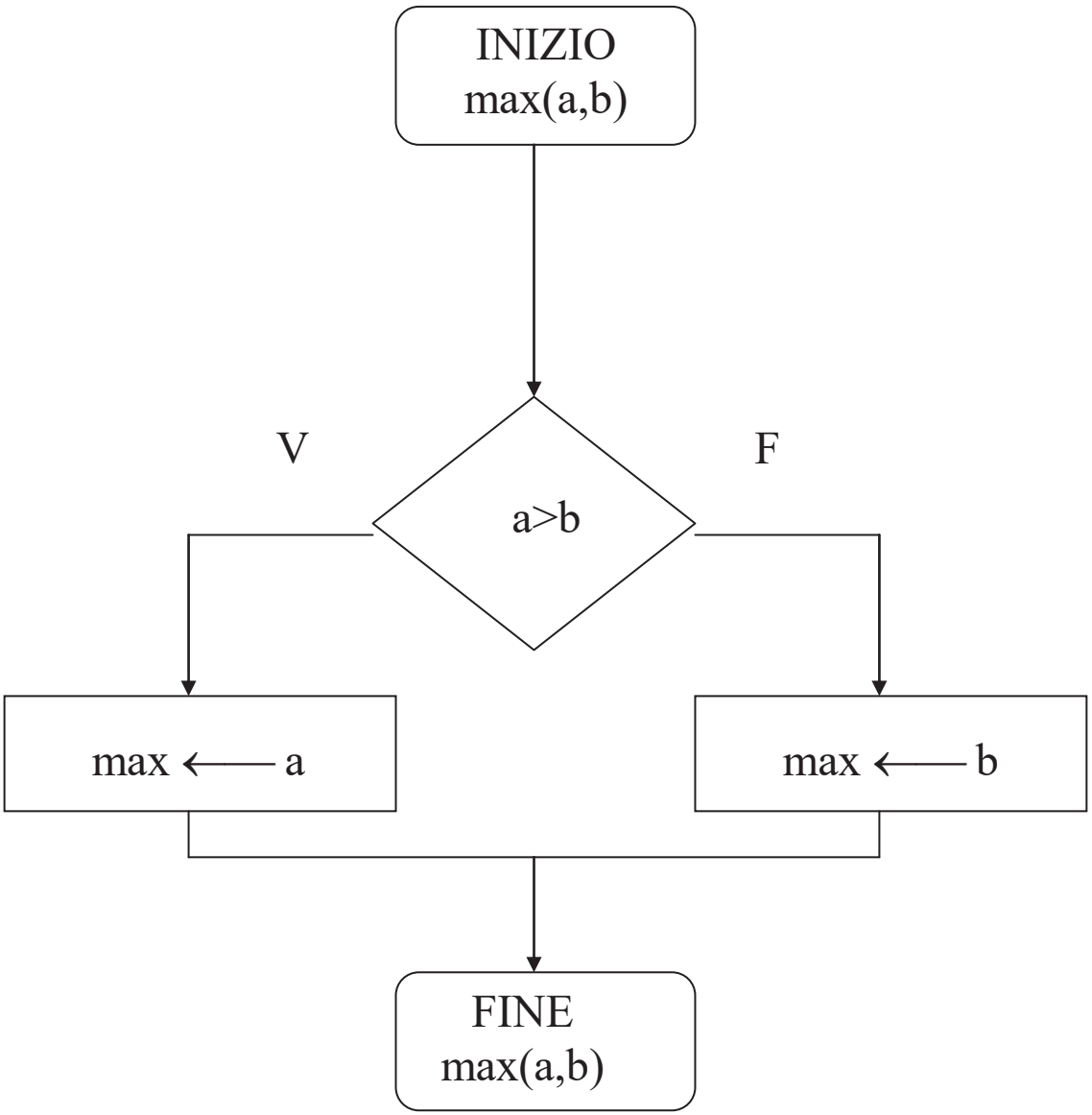
I sottoprogrammi vengono rappresentati in un flow-chart da un blocco di questo tipo.

Il diagramma a blocchi del Problema 2 è quindi il seguente:

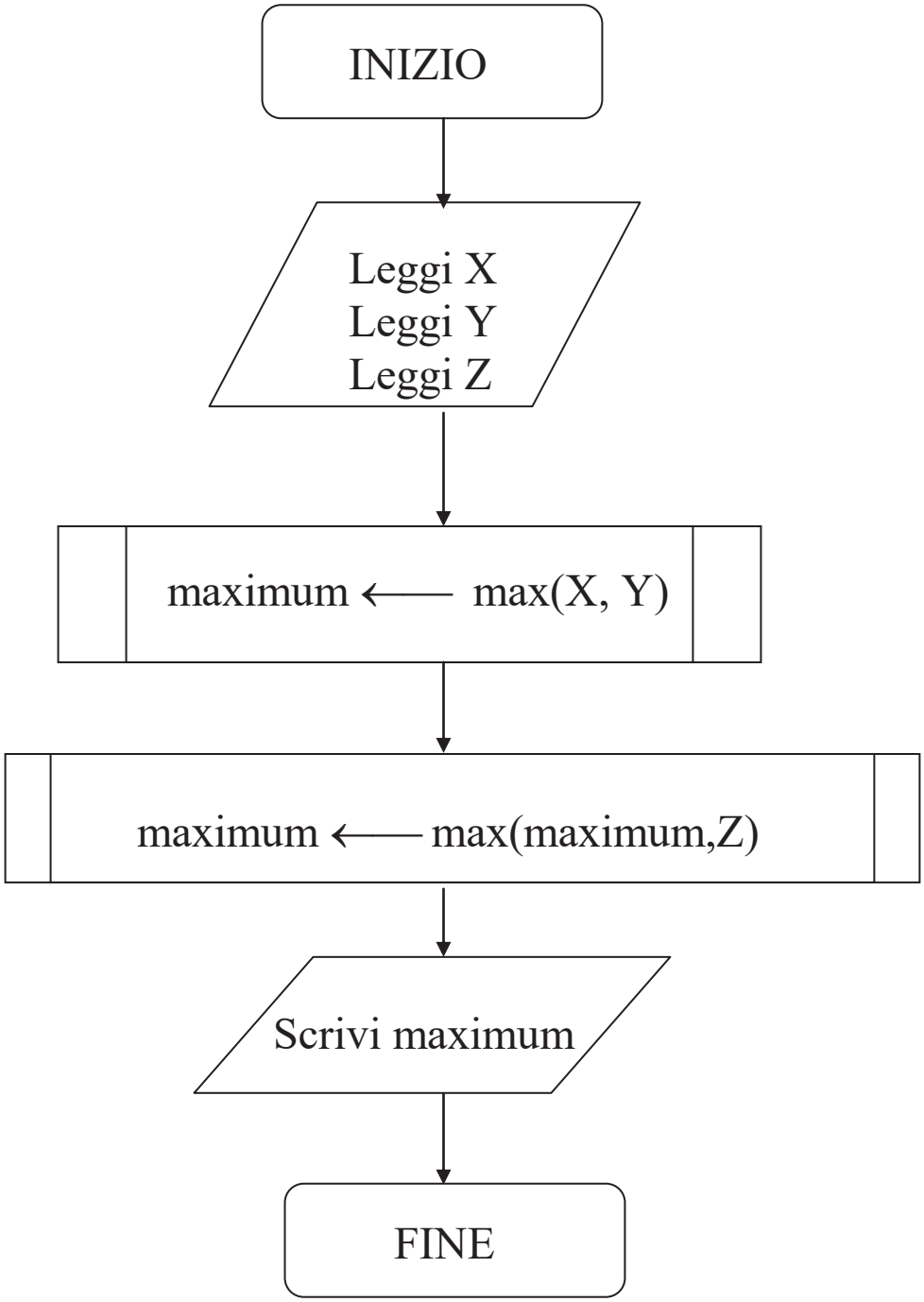


Il problema della ricerca del massimo tra due numeri può venire trattato mediante una *funzione*, max(*a*,*b*), in cui *a* e *b* sono detti *parametri*.

Il diagramma a blocchi della funzione max sarà il seguente:



Utilizzando questa funzione il diagramma di flusso relativo al Problema 2 diviene:



In generale i sottoprogrammi possono essere costituiti da

- *funzioni*, se restituiscono un risultato direttamente utilizzabile in un'espressione
- *procedure* o *subroutine*, se non restituiscono un risultato direttamente utilizzabile in un'espressione

ALCUNE OSSERVAZIONI

- Le istruzioni che compongono un sottoprogramma vengono scritte *una volta soltanto*.
- Un programma può richiamare un sottoprogramma *più volte* ed ogni volta l'esecutore esegue tutte le istruzioni che compongono il sottoprogramma stesso.
- Ad ogni chiamata di un sottoprogramma è possibile passare allo stesso dati diversi su cui operare attraverso i *parametri*.
- Definendo un sottoprogramma (cioè scrivendo le istruzioni dell'algoritmo corrispondente) è come se si dotasse l'esecutore della capacità di risolvere, come sottoproblemi elementari, dei problemi più complessi.
- I linguaggi di programmazione evoluti dispongono abitualmente di *librerie di funzioni* che consentono di trattare come sottoproblemi elementari dei problemi che non lo sono affatto (ad esempio il calcolo della radice quadrata di un numero)

In generale, con problemi complessi, è indispensabile individuare sottoproblemi più semplici, da risolvere separatamente.

- Programmazione *top-down*: si parte da un algoritmo molto generico che descrive globalmente la soluzione del problema per passare via via ad algoritmi sempre più dettagliati che descrivono le singole operazioni particolari (problemi terminali elementari)

- Programmazione *bottom-up*: si parte della realizzazione di algoritmi che risolvono specifici problemi (moduli); questi vengono poi collegati tra loro creando una struttura più complessa fino ad arrivare al programma finale.

La programmazione di tipo top-down consente

- di avere una visione più chiara delle varie parti del programma
- di sviluppare il programma per fasi
- di affidare eventualmente lo sviluppo dei singoli moduli a programmatori diversi
- di utilizzare eventuali moduli già esistenti
- di verificare il programma ad ogni livello di realizzazione (test più semplici ed efficienti)

Problema 3: determinare il massimo di n numeri reali.

Anche in questo caso ci si riconduce alla ricerca del massimo tra due numeri.

Tralasciando per il momento le istruzioni di input, il problema potrebbe essere risolto in questo modo:

P1:	Trovare il massimo tra i primi due numeri
P2:	Trovare il massimo tra il terzo numero ed il risultato del sottoproblema precedente
P3:	Trovare il massimo tra il quarto numero ed il risultato del sottoproblema precedente
	……
Pn-1	Trovare il massimo tra l'n-esimo numero ed il risultato del sottoproblema precedente

Il problema è così risolto poiché ad ogni passo ci si riconduce alla soluzione del Problema 1.

Per ripetere l'esecuzione del calcolo del massimo tra due numeri si dovrà eseguire un ciclo, nel modo seguente:

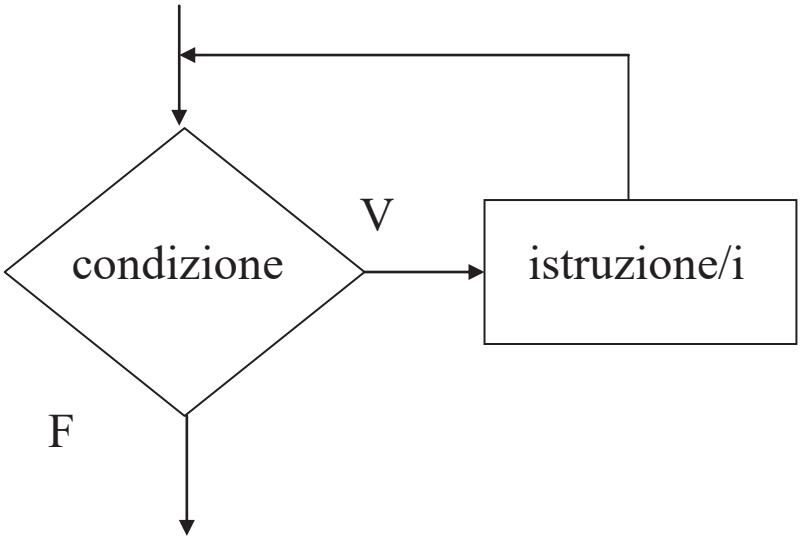
P1:	Trovare il massimo tra i primi due numeri
P2:	Finché ci sono numeri da verificare, ripetere il passo P3
P3:	Trovare il massimo tra il nuovo numero da esaminare ed il più grande trovato in precedenza

La struttura: *Finché condizione ripetere*

istruzione/i

è presente in tutti i linguaggi strutturati e viene chiamata *struttura di tipo while* o *ciclo while*.

- La condizione costituisce un test per il quale si verifica il valore logico (vero o falso).
- Fintantoché il valore logico del predicato è vero si ripete l'esecuzione dell'istruzione.
- Dopo ogni esecuzione dell'istruzione viene rivalutato il test ed il ciclo termina quando tale test risulta falso.
- Una volta che il ciclo sia terminato, vengono eseguite le istruzioni successive nella sequenza.
- Affinché il ciclo abbia termine l'istruzione (o le istruzioni) contenute all'interno del ciclo devono prima o poi fare in modo che la condizione assuma il valore logico falso.



Do While condizione
istruzione/i
Loop

La risoluzione completa del Problema 3 richiede di indicare come effettuare l'input dei dati, in modo che l'algoritmo funzioni per un numero qualsiasi di numeri tra i quali cercare il massimo.

1° caso - Si richiede all'utente di indicare quanti sono i dati.

L'algoritmo sarà allora:

Scrivi "Quanti sono i numeri da esaminare?"
Leggi n
If n > 0 Then
  Leggi m
  cont ⟵ 1 cont è un *contatore*.
  Do While cont < n
    Leggi x
    cont ⟵ cont + 1
    m ⟵ max(m,x)
  Loop
  Scrivi "Il massimo è" m
End If

A sua volta, ammettendo ora che l'esecutore "conosca" la relazione d'ordine nell'insieme dei numeri reali, l'algoritmo di determinazione del massimo tra due numeri sarà:

If a > b Then
  max ⟵ a
Else
  max ⟵ b
End If

2° caso – Si chiede all'utente se vuole continuare o meno.

L'algoritmo diventa allora:

Scrivi "Questo programma serve per trovare il massimo in una sequenza di numeri"

Scrivi "Inserisci il primo numero"

Leggi m

Scrivi "Vuoi inserire altri numeri? (S/N)"

Leggi risposta

Do While risposta = "S"

Leggi x

m ⟵ max(m,x)

Scrivi "Vuoi inserire altri numeri ? (S/N)"

Leggi risposta

Loop

Scrivi "Il massimo è" m

PROGRAMMAZIONE STRUTTURATA

L'efficienza di un programma è in genere misurata in termini di

- tempo di esecuzione
- memoria richiesta
- affidabilità
- facilità di manutenzione
- estensibilità

Un programma chiaro e semplice consente

- facilità nella fase di verifica e correzione degli errori
- facilità nello sviluppo, nella manutenzione e nell'aggiornamento anche da parte di persone diverse dall'autore
- riduzione degli errori in fase di realizzazione

Alcune buone regole di programmazione:

- commentare adeguatamente il programma
- scegliere nomi di variabili significativi
- aumentare la modularità
- standardizzazione
- altre semplici regole (indentazione, eliminazione di parti ridondanti, …)

Ma prima di tutto:

SEMPLIFICARE LA STRUTTURA DEL PROGRAMMA

TEOREMA DI BÖHM-JACOPINI (1966)

Qualunque algoritmo descrivibile con diagrammi a blocchi è descrivibile utilizzando soltanto le seguenti *strutture di controllo*:

1) **BLOCCO**: sequenza di istruzione

2) **SELEZIONE**: If ... Then ... Else ...

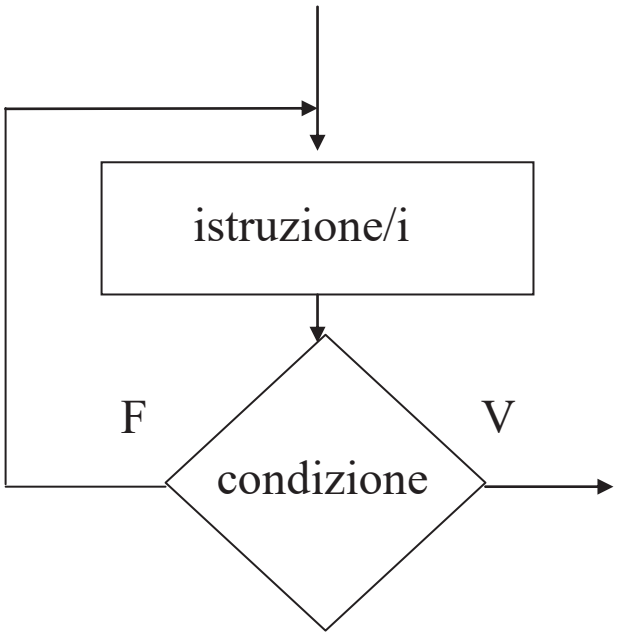
3) **FRASE ITERATIVA**: Do While ...

a condizione di usare, se necessario, variabili ausiliarie e duplicazioni di blocchi.

Questo teorema costituisce la base della:

PROGRAMMAZIONE STRUTTURATA

NB: le strutture precedenti hanno la caratteristica fondamentale di avere *un unico punto di entrata* ed *un unico punto di uscita*.
Oltre alla selezione ed al ciclo while esistono anche altre strutture di controllo, ad esempio



Do
istruzione/i
Until condizione

Il Problema 3 precedente potrebbe essere risolto anche con un algoritmo che non utilizza le sole strutture di controllo viste: