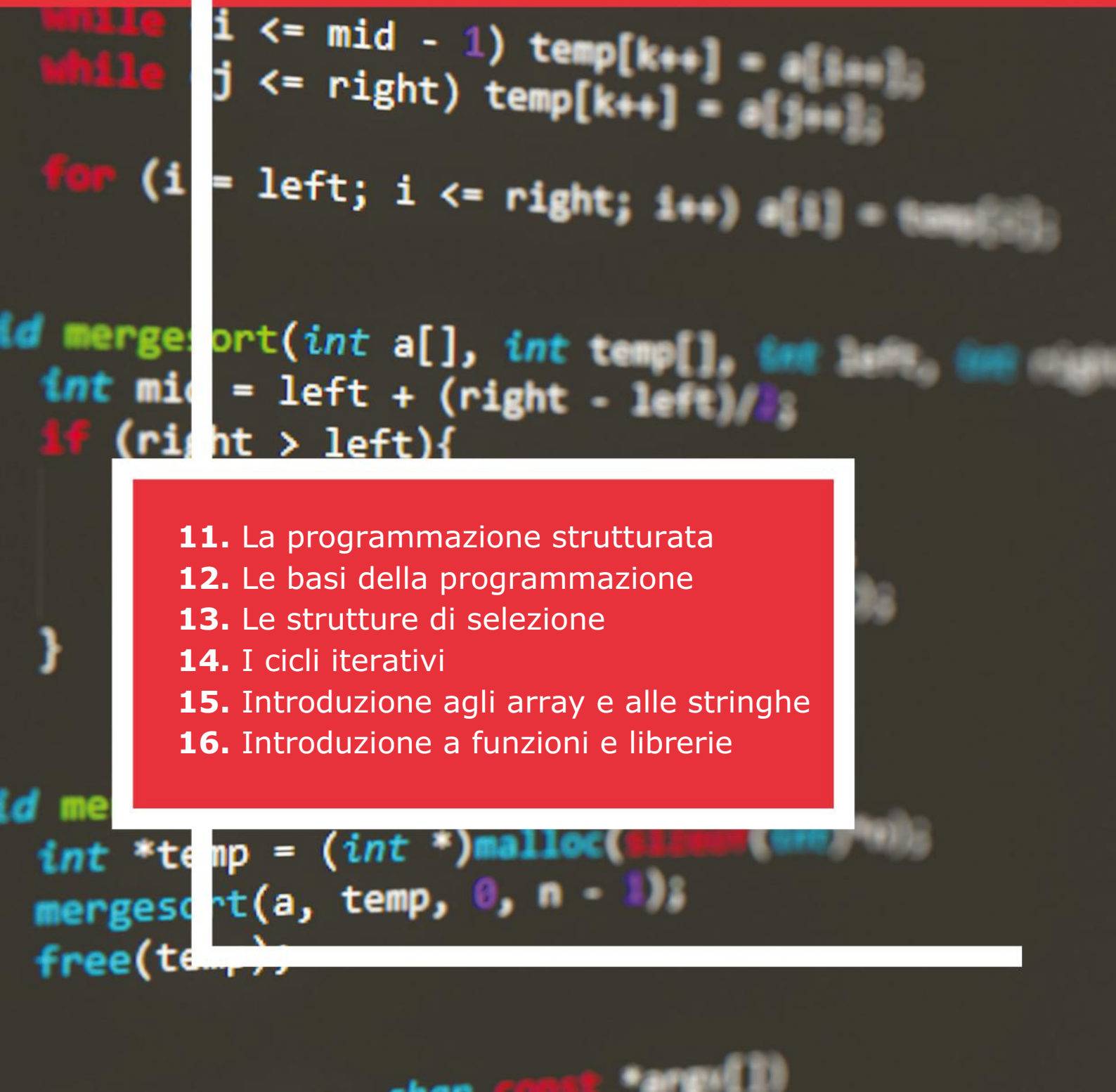
Iniziare a programmare



11. La programmazione strutturata
12. Le basi della programmazione
13. Le strutture di selezione
14. I cicli iterativi
15. Introduzione agli array e alle stringhe
16. Introduzione a funzioni e librerie

CAPITOLO

11 La programmazione strutturata

Per risolvere problemi con l'aiuto del calcolatore, si scrivono **programmi**; ogni programma dunque è la traduzione di un **algoritmo** – cioè di una serie di precise istruzioni elementari – in un linguaggio che la macchina possa comprendere ed eseguire.

Uno strumento utile per **progettare** gli algoritmi è la loro rappresentazione come schemi grafici mediante i **diagrammi di flusso**, in inglese *flowchart*.

11.1 Gli algoritmi e i diagrammi di flusso

Il **diagramma di flusso** è un modo per rappresentare in forma grafica un algoritmo.

I diagrammi di flusso sono chiamati anche *diagrammi a blocchi*, perché ogni istruzione dell'algoritmo è rappresentata dentro un blocco grafico, la cui forma è determinata dal tipo di istruzione, come riassunto dalla **tabella 11.1**.

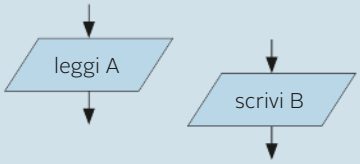
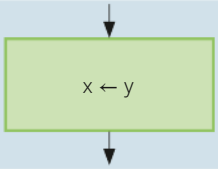
I blocchi sono collegati da linee che terminano con una **freccia** e indicano il **flusso del controllo** dell'algoritmo, cioè il percorso da seguire nell'esecuzione delle istruzioni.

Con questi pochi blocchi elementari, opportunamente collegati da frecce, si può rappresentare qualsiasi algoritmo.

Leggere e disegnare i diagrammi di flusso

Un diagramma di flusso si legge partendo dal blocco iniziale e seguendo la freccia fino al blocco successivo. Si effettua l'operazione descritta nel blocco, poi si segue la freccia che ne esce, fino al blocco successivo. E così via, fino a giungere al blocco finale.

Tabella 11.1 I blocchi grafici usati nei diagrammi di flusso.

forma	aspetto grafico	funzione
OVALE **blocchi di inizio e fine algoritmo** (in inglese *start/begin* ed *end*)		indicano il punto di partenza e la conclusione dell'algoritmo
PARALLELOGRAMMA **blocco di lettura/scrittura** (in inglese *I/O*, cioè *input/output*)		indica l'acquisizione di dati in ingresso o la comunicazione di dati in uscita
RETTANGOLO **blocco di azione/elaborazione** (in inglese *operation/process*)		indica un'operazione da effettuare sulle variabili o sui dati
ROMBO **blocco di controllo/test** (in inglese *decision*)		indica un controllo da effettuare e la direzione in cui procedere a seconda del risultato

La **figura 11.1** per esempio mostra il diagramma di flusso per questo semplice algoritmo per la messa in carica dello smartphone:

1. Controlla quanta carica ha ancora tuo telefono.
2. Se la carica della batteria supera il 20%, non fare nulla.
3. Altrimenti, cioè se la carica è inferiore al 20%, collega il caricabatteria.

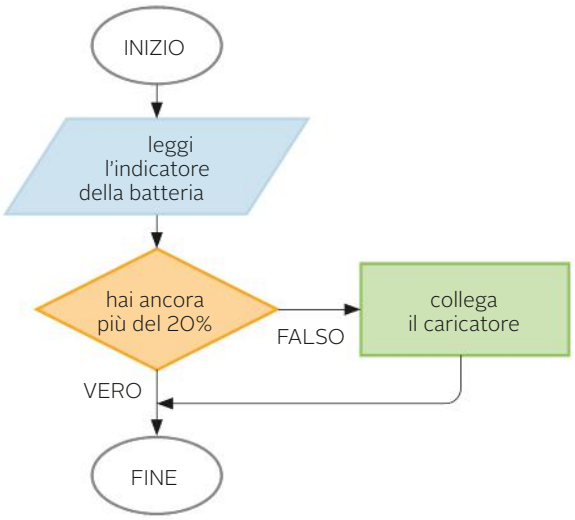




Figura 11.1 Un diagramma di flusso per la messa in carica del telefonino.

Nel diagramma di flusso dal blocco iniziale si passa a un blocco di lettura; il dato di input per l'algoritmo è il valore mostrato dall'indicatore della batteria.

Da qui poi si passa a un blocco di controllo, dove la condizione da verificare è la percentuale di carica ancora disponibile: se la carica supera il 20%, non si fa nulla e l'algoritmo termina passando al blocco finale; in caso contrario, prima si passa all'azione di collegare il caricatore, quindi al blocco finale.

UN ESERCIZIO AL VOLO

Che cosa contengono i blocchi di controllo?

.....

Un semplice algoritmo matematico

Il diagramma di flusso della **figura 11.2** rappresenta un semplice esempio di algoritmo matematico. Un programma basato su questo algoritmo somma due numeri dati come input e mostra il risultato, con questa sequenza di azioni:

- il programma inizia;
- il programma legge i due numeri;
- chiama i due numeri rispettivamente A e B;
- somma i valori di A e di B e chiama il risultato SOMMA;
- produce come output il valore di SOMMA;
- il programma termina.

Nel linguaggio informatico A, B e SOMMA sono *variabili* e il simbolo della freccia verso sinistra (←) indica l'*assegnazione* del valore a una variabile.

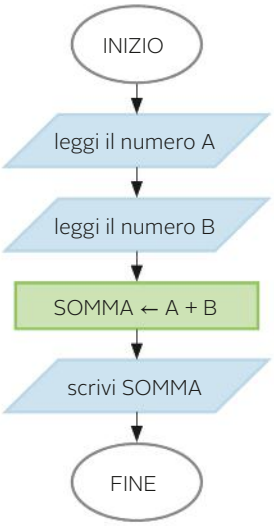


Figura 11.2 Il diagramma di flusso per l'algoritmo che calcola e mostra la somma di due numeri dati come input.

Le regole per i diagrammi di flusso

Nel disegnare un diagramma di flusso bisogna sempre rispettare alcune regole:

- ogni blocco di lettura/scrittura e di azione/elaborazione deve avere una sola freccia entrante e una sola freccia uscente;
- ogni blocco di controllo deve avere una sola freccia entrante e due frecce uscenti;
- ogni freccia deve entrare in un blocco oppure innestarsi in un'altra freccia;
- ci deve essere una freccia che arriva al blocco di fine algoritmo.

Flowgorithm, un approccio grafico al *coding*

L'applicazione **Flowgorithm** per Windows è un freeware che permette di **disegnare diagrammi di flusso** e di **eseguire gli algoritmi** corrispondenti.

Flowgorithm è prodotto in California dalla Sacramento State University e si scarica dal sito **flowgorithm.org**. Un software simile per Linux e macOS è **diagrams.net**.

Dopo l'installazione, all'apertura il programma mostra soltanto i blocchi di **Inizio** e **Fine** di un algoritmo ancora da costruire, collegati da una freccia.

Per inserire nuovi blocchi nel diagramma, si avvicina il puntatore del mouse alla freccia che collega i blocchi: quando questa diventa arancione, si fa clic e appare una finestra con i diversi blocchi disponibili (**figura 11.3**), che si selezionano con un clic.

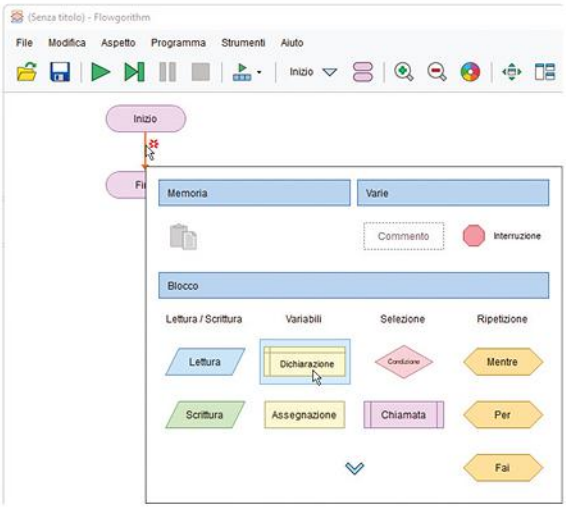


Figura 11.3 L'interfaccia di Flowgorithm.

UN ESERCIZIO AL VOLO

Come è chiamato in Flowgorithm il blocco di controllo/test?

.....

Lo stesso meccanismo permette poi di inserire ulteriori blocchi; le frecce di collegamento vengono aggiunte in modo automatico dal programma.

Costruisci per esempio il diagramma di flusso della precedente **figura 11.2**. Nota che per prima cosa dovrai **dichiarare le variabili** che intendi usare.

Quando hai completato il diagramma, puoi fare clic sul pulsante **Esegui** per eseguire l'algoritmo. Apparirà allora la nuova finestra **Esecuzione** che vedi in basso a destra nella **figura 11.4**: digita in basso il primo valore e fai clic su **Invio**, poi ripeti per il secondo valore; il programma allora mostrerà (con il colore dei blocchi di scrittura) la somma dei due dati di input.

Con questa stessa procedura puoi rappresentare graficamente e poi eseguire algoritmi di qualsiasi tipo.

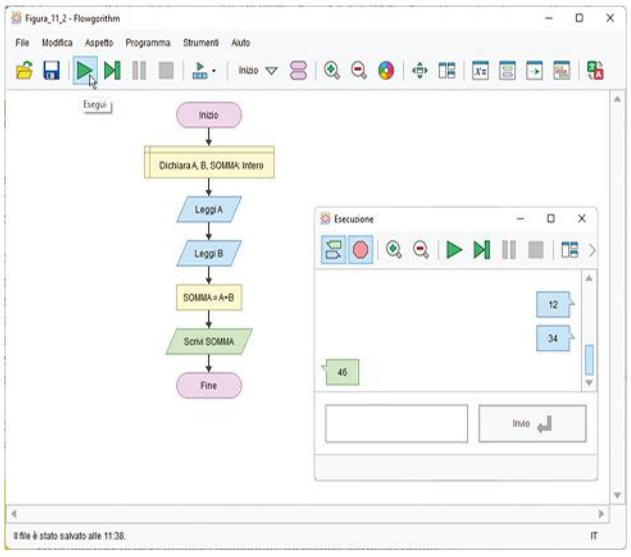


Figura 11.4 Il diagramma di flusso per la somma di due numeri e la sua esecuzione in Flowgorithm.

Flowgorithm consente anche di vedere il **codice sorgente** che può esprimere l'**algoritmo** in esame nei diversi **linguaggi di programmazione**.

Fai clic sull'icona **Codifica** in alto a destra nella barra dei comandi: si aprirà una finestra con la codifica dell'algoritmo in uno **pseudo-codice** che sta a metà strada tra il linguaggio naturale e il linguaggio formale usato dai programmatori.

Se ora fai clic sull'icona **Linguaggio**, puoi scegliere tra decine di linguaggi di programmazione diversi. In questo modo potrai iniziare a prendere familiarità con i costrutti che caratterizzano i vari linguaggi e confrontarli tra loro (**figura 11.5**).

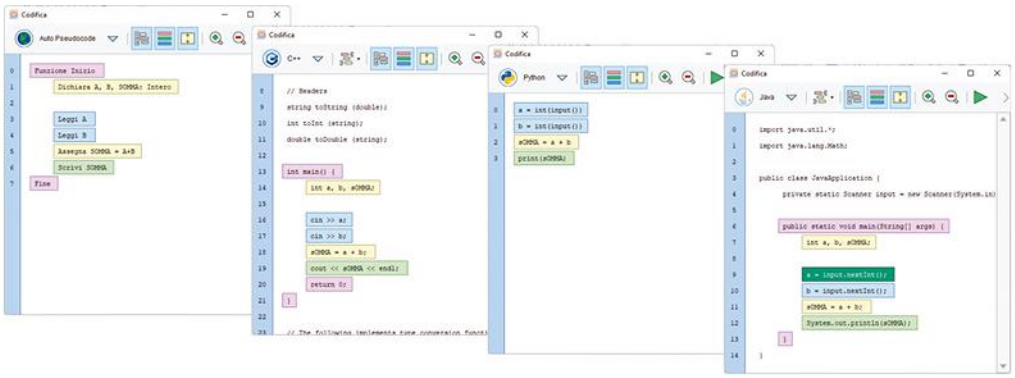


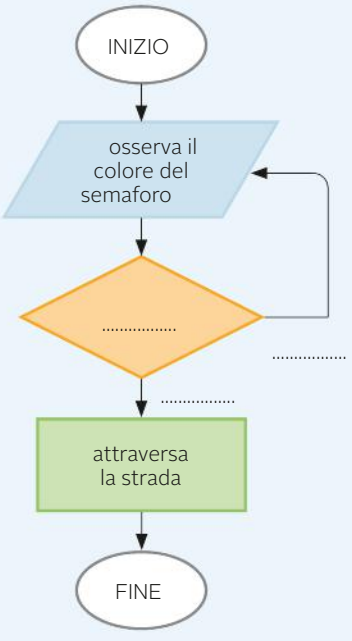
Figura 11.5 L'algoritmo per la somma di due numeri espresso in diversi linguaggi di programmazione.

RIPASSA I CONCETTI-CHIAVE

- Ogni **algoritmo** si può rappresentare in forma di **diagramma di flusso** usando **blocchi grafici** collegati da **frecce** che indicano la sequenza in cui eseguire le istruzioni.
- I diagrammi di flusso aiutano a visualizzare gli algoritmi evidenziando i **collegamenti** tra le istruzioni e quindi il **flusso del controllo** durante l'esecuzione.
- L'applicazione **Flowgorithm** permette di disegnare diagrammi di flusso, eseguire gli algoritmi e vedere come si esprimono nei diversi linguaggi di programmazione.

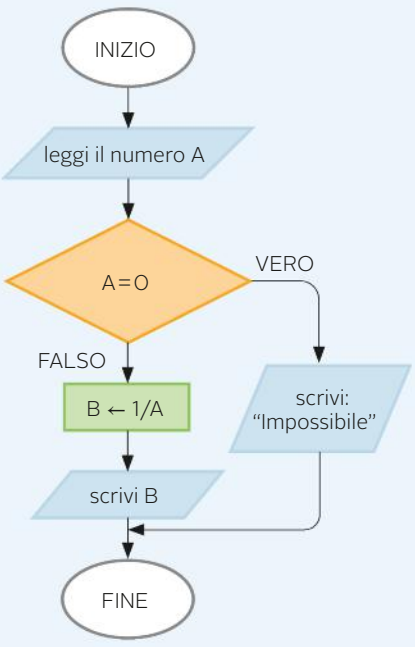
ESERCIZI

1. Completa questo diagramma di flusso che descrive un algoritmo per l'attraversamento pedonale a un semaforo.



2. Spiega che cosa fa questo algoritmo e perché il diagramma di flusso deve contenere un blocco di test.

.....
.....
.....
.....
.....
.....
.....
.....



11.2 Gli elementi di base dei programmi

Nel 1966 Corrado Böhm (**figura 11.6**), professore di informatica all'Università di Roma, insieme al suo studente Giuseppe Jacopini dimostrò che gli algoritmi hanno una proprietà generale molto importante: si possono sempre esprimere usando soltanto tre strutture fondamentali, che sono la *sequenza*, la *selezione* (o *struttura condizionale*) e il *ciclo* (o *struttura iterativa*).

Questo celebre **teorema di Böhm-Jacopini** è il fondamento su cui si basa la **programmazione strutturata**, il *paradigma* che caratterizza tutti i principali linguaggi di programmazione odierni.

Il termine «paradigma» qui indica l'approccio concettuale usato come riferimento dai programmatori quando progettano e scrivono un programma per il computer.



Figura 11.6 Corrado Böhm con i suoi studi ha dato importanti contributi allo sviluppo dell'informatica teorica.

La **programmazione strutturata** permette di limitare la complessità del flusso di controllo dei programmi, grazie all'uso di tre sole strutture di base:

- la **sequenza** di istruzioni;
- la **selezione** o **struttura condizionale**;
- il **ciclo** o **struttura iterativa**.

Nel seguito del capitolo esamineremo in dettaglio queste tre strutture.

Le istruzioni in sequenza

La **struttura sequenziale** è la più semplice: le istruzioni sono elencate in sequenza e vanno eseguite una dopo l'altra nell'ordine in cui le ha scritte il programmatore.

Le istruzioni possono essere di tre tipi:

- **lettura** (o ingresso, o input): consente di leggere il valore di una variabile da un'unità di ingresso (per esempio la tastiera) e trasferisce tale valore nella parte di memoria RAM che contiene i dati del programma;
- **scrittura** (o uscita, o output): operazione inversa alla lettura, serve per far apparire sullo schermo (o per stampare su carta, o per registrare in un file) il contenuto di una variabile;
- **assegnazione**: fa assumere a una variabile un valore inserito come input, copiato da un'altra variabile oppure ottenuto calcolando una data espressione.

Vediamo un esempio molto semplice di algoritmo con struttura sequenziale: dato il valore del diametro **D** di un cerchio, vogliamo trovare la lunghezza **L** della circonferenza. L'algoritmo è rappresentato dal diagramma di flusso della **figura 11.7**.

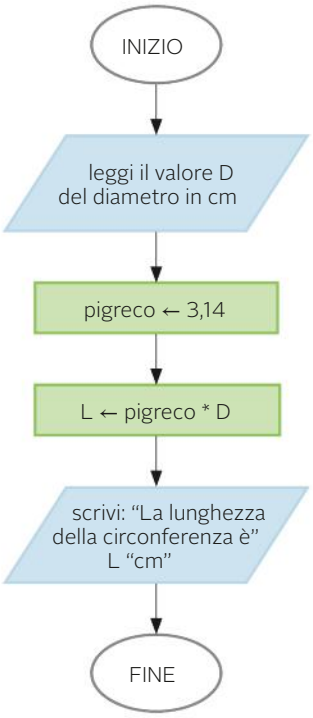


Figura 11.7 Il diagramma di flusso per il calcolo della lunghezza della circonferenza **L**, dato il diametro **D**.

ALGORITMO PER CALCOLARE LA LUNGHEZZA DELLA CIRCONFERENZA

1. Leggi il numero di input e assegna il suo valore a **D**.
2. Assegna a **pigreco** il valore 3,14.
3. Assegna a **L** il valore trovato calcolando **pigreco** * **D**.
4. Scrivi come output un messaggio che contiene il valore finale di **L**.

Nella formula abbiamo usato la notazione standard in informatica per la moltiplicazione, che si indica con l'asterisco * (per la divisione si usa invece la barra /).

Nell'algoritmo e nel diagramma di flusso intervengono due *variabili*, **D** e **L**, e una *costante*, **pigreco**.

Come abbiamo già visto, il simbolo della freccia verso sinistra (←) indica l'*assegnazione* del valore a queste entità.

I concetti di «variabile», «costante» e «assegnazione» sono molto importanti nell'informatica ed è fondamentale capire bene il loro significato, che ora andremo ad approfondire.

Le variabili e le costanti

Gli algoritmi servono per risolvere classi generali di problemi. Perciò, come avviene anche nelle formule matematiche, negli algoritmi appaiono **variabili**, ossia simboli che rappresentano dati il cui valore non è specificato *a priori*.

L'algoritmo della precedente **figura 11.7**, per esempio, non è scritto per calcolare la lunghezza della circonferenza per un particolare valore del diametro: funziona per *qualsiasi* valore della variabile diametro **D**.

In informatica una **variabile** è uno **spazio di memoria** a cui è associato un **nome**: registra un **valore** che può cambiare durante l'esecuzione di un programma.

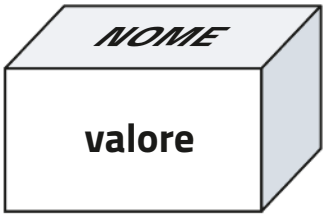


Figura 11.8 Una variabile è come una scatola contenente un dato che può cambiare nel tempo.

Nelle applicazioni informatiche a ogni variabile viene *allocato* (cioè riservato) uno **spazio di memoria**, nella RAM del computer, destinato a contenere il dato che costituisce il valore della variabile. Una variabile dunque è come una scatola su cui è scritto il nome della variabile, mentre il valore è il contenuto della scatola (**figura 11.8**).

Nei programmi per computer ciò che importa di una variabile non è il nome, bensì il **valore** che essa ha in ogni dato momento durante l'esecuzione.

All'inizio di ogni algoritmo bisogna **inizializzare** le variabili, cioè far sì che abbiano un valore (introdotto come dato di input, oppure assegnato con una specifica istruzione).

Il valore di una variabile può poi cambiare nel corso dell'algoritmo.

In informatica una **costante** invece è uno spazio di memoria il cui valore, una volta inizializzato, non cambia mai.

Nell'algoritmo della circonferenza, per esempio, **pigreco** è una costante: il contenuto del suo spazio di memoria non sarà mai modificato durante l'esecuzione del programma.

Ci sono buone ragioni per usare negli algoritmi costanti simboliche, invece di scrivere i loro valori numerici espliciti. Per esempio, nel calcolo della lunghezza della circonferenza si potrebbe semplicemente scrivere la formula come **L** ← 3,14 * **D**. Ma introdurre la costante **pigreco** è più elegante dal punto di vista formale, e inoltre può rivelarsi utile per alcune ragioni importanti:

- se si vuole estendere l'algoritmo ad altri calcoli che coinvolgono π (come trovare l'area del cerchio o il volume della sfera di diametro **D**), si evita di dover riscrivere ogni volta il numero, riducendo così il rischio di commettere errori;
- se si decide poi di usare un valore diverso per la costante (per esempio 3,141592 invece di 3,14, così da ottenere un risultato più preciso), basta cambiarlo in una sola istruzione, quella che assegna il valore alla costante **pigreco**.

UN ESERCIZIO AL VOLO

Qual è la differenza tra una variabile e una costante?

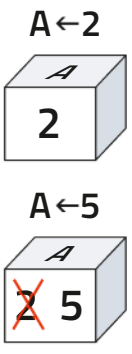
.....

.....

L'assegnazione dei valori

Il simbolo della freccia verso sinistra (←) indica l'operazione di **assegnazione**: alla variabile a sinistra della freccia si assegna il valore indicato a destra della freccia.

Assegnare un valore a una variabile significa **sovrascrivere** con il nuovo valore quello che era precedentemente contenuto nello spazio di memoria della variabile.



Nei linguaggi di programmazione l'assegnazione di solito è rappresentata con il segno di uguaglianza, =. Non si tratta però dell'uguaglianza a cui siamo abituati nelle formule algebriche: l'istruzione **A = B** non significa «le variabili **A** e **B** hanno lo stesso valore», bensì «assegniamo ad **A** lo stesso valore che ha **B**».

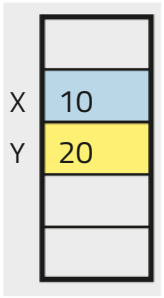
Per esempio, un'istruzione come **A = A+1** in termini algebrici è priva di senso: non è possibile che un numero sia uguale allo stesso numero aumentato di un'unità. Nel contesto della programmazione, invece, l'istruzione significa «cambiamo il valore di **A**, aumentandolo di un'unità».

Per evitare possibili confusioni, negli algoritmi useremo sempre la freccia ← di assegnazione e non il segno «uguale», scrivendo l'istruzione nella forma **A ← A+1**.

Le variabili e la loro posizione in memoria

Per programmare non occorre sapere dove siano fisicamente posizionati i dati nel computer; è importante però capire la relazione che esiste tra una varabile e il suo *indirizzo*, cioè la sua posizione nella memoria elettronica.

Supponiamo per esempio di avere nel programma due variabili **X** e **Y** che contengono rispettivamente i valori 10 e 20, come schematizzato qui a fianco. Vogliamo scambiare il loro contenuto, cioè far sì che il valore che si trova in memoria all'indirizzo di **X** passi all'indirizzo di **Y**, e viceversa (*swapping* in inglese).



Per ottenere questo risultato useremo l'istruzione di assegnazione, ma dobbiamo fare attenzione a non «perdere per strada» nessun valore.

Non possiamo per esempio iniziare con l'istruzione **Y** ← **X**: se lo facessimo, copieremmo nell'indirizzo di **Y** il valore di **X**, come desiderato, ma perderemmo al contempo il vecchio valore di **Y**, che verrebbe sovrascritto.

Il procedimento da seguire è simile a quello per scambiare il contenuto di due bottiglie, una piena d'acqua e una di vino: occorre una terza bottiglia in cui travasare il liquido.

Per il nostro problema occorre utilizzare un terzo indirizzo nella RAM, quello della variabile **Z**, come schematizzato nella **figura 11.9**:

1. per prima cosa si copia in **Z** il valore di **X**;
2. poi si copia in **X** il valore di **Y**;
3. infine si copia in **Y** il valore di **Z**.

La sequenza delle istruzioni di assegnazione sarà dunque quella mostrata in basso nella figura: prima **Z**←**X**, poi **X**←**Y**, infine **Y**←**Z**.

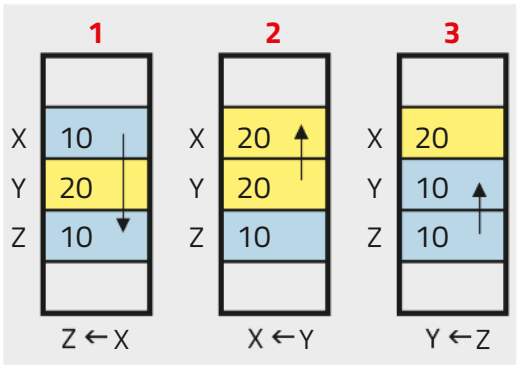


Figura 11.9 Lo scambio dei contenuti di due variabili nella memoria del computer.

RIPASSA I CONCETTI-CHIAVE

- La **programmazione strutturata**, basata sul teorema di Böhm-Jacopini, usa soltanto tre strutture: la **sequenza di istruzioni**, la **selezione** (o **struttura condizionale**) e il **ciclo** (o **struttura iterativa**).
- Negli algoritmi alle **variabili** è associato un valore che può cambiare durante l'esecuzione dell'algoritmo; le **costanti** invece hanno un valore immutabile.
- Lo **spazio di memoria** riservato a una variabile o a una costante contiene un valore attribuito con un'**assegnazione**, indicata nei diagrammi di flusso dalla freccia ←.

ESERCIZI

1. Se esegui il programma della **figura 11.7** assegnando a **D** il valore di input 1, che cosa conterrà alla fine lo spazio di memoria RAM della variabile **L**?

2. Se nella **figura 11.7** invece del diametro l'input fosse il raggio, assegnato a una variabile **R**, come andrebbe riscritta l'istruzione di assegnazione per **L**?

3. Se nell'esercizio precedente si volesse calcolare anche l'area del cerchio, che cosa andrebbe aggiunto all'algoritmo?

4. Se la costante **C** contiene il valore 15 all'inizio dell'esecuzione di un programma, quale valore conterrà alla fine dell'esecuzione?

5. Se la variabile **V** contiene il valore 15 all'inizio dell'esecuzione, quale valore conterrà alla fine dell'esecuzione?

6. Hai due coppie di variabili A, B e C, D e vuoi scambiare tra loro i valori della prima coppia e poi quelli della seconda coppia. Quante variabili addizionali dovrai introdurre?

11.3 La struttura condizionale o selezione

La **selezione** o **struttura condizionale** permette di eseguire istruzioni diverse a seconda che una **condizione** sia vera o falsa.

Per rappresentare le selezioni, i linguaggi di programmazione usano la parola-chiave **if** («se» in inglese), che può essere seguita da **else** («altrimenti») e da **end if** («fine della struttura if»).

LA STRUTTURA CONDIZIONALE O SELEZIONE

```
if (condizione da verificare)
    {istruzioni da eseguire se la condizione è vera;}
else
    {istruzioni da eseguire se la condizione è falsa;}
end if
```

queste istruzioni (una come minimo) sono sempre presenti

queste istruzioni sono presenti nelle *selezioni a due vie*

Quando si applica una struttura condizionale, si controlla se una certa condizione sia vera oppure falsa e si eseguono istruzioni diverse a seconda del caso che si presenta.

La struttura condizionale più semplice è quella **a una via**, che dice: «se si verifica la condizione, fai questo».

La struttura **a due vie**, del tipo «se... altrimenti...», specifica invece che cosa fare in due casi diversi: prevede cioè due strade alternative, a seconda che la condizione di controllo risulti vera oppure falsa. Vediamo un semplice esempio di uso di questa struttura.

UN ESERCIZIO AL VOLO

Qual è un altro nome per le strutture condizionali?

.....

Trovare il valore assoluto di un numero

Dato un numero X, positivo o negativo, trovare il *valore assoluto* di X (indicato in matematica con |X|) significa prendere il valore numerico di X trascurandone il segno:

$$\textbf{se } X \geq 0\textbf{, allora } |X| = X \quad \text{ma} \quad \textbf{se } X < 0\textbf{, allora } |X| = -X$$

La funzione «valore assoluto» dunque lascia X immutato se X è positivo o nullo, mentre trasforma X nel suo opposto se X è negativo. Per esempio: $|2| = 2$ mentre $|-5| = 5$.

ALGORITMO PER CALCOLARE IL VALORE ASSOLUTO

1. Leggi il numero e assegna il suo valore alla variabile X.
2. Se X è maggiore o uguale a zero, mostra come risultato il valore di X.
3. Altrimenti, cioè se invece X è minore di zero, mostra come risultato il suo opposto, ossia il valore ottenuto cambiando segno a X.

La **figura 11.10** mostra il corrispondente diagramma di flusso.

Il blocco a forma di rombo contiene il test di selezione: se il numero letto è negativo occorre compiere un'azione (cambiare il suo segno), mentre se è positivo o nullo non è necessaria alcuna azione. In entrambi i casi, l'algoritmo termina stampando o mostrando a schermo il valore finale di X.

Va notato che il valore della variabile X può cambiare nel corso dell'algoritmo, se viene eseguita l'istruzione di assegnazione $X \leftarrow -X$.

In tal caso, al termine dell'algoritmo il valore iniziale (negativo) di X non sarà più presente nella memoria RAM del computer: è stato sostituito dal valore (positivo) –X.

Quindi, nel caso di $X < 0$, alla fine del programma nel computer non ci sarà più traccia del valore originario (di ingresso) della variabile X.

Se invece si volesse conservare in memoria tale valore, per esempio perché serve in altre parti del programma, bisognerebbe complicare un poco le istruzioni: dopo il test di selezione si dovrebbe assegnare a una nuova variabile Y il valore X oppure –X (a seconda del risultato del test su X) e produrre alla fine come output il valore della variabile Y.

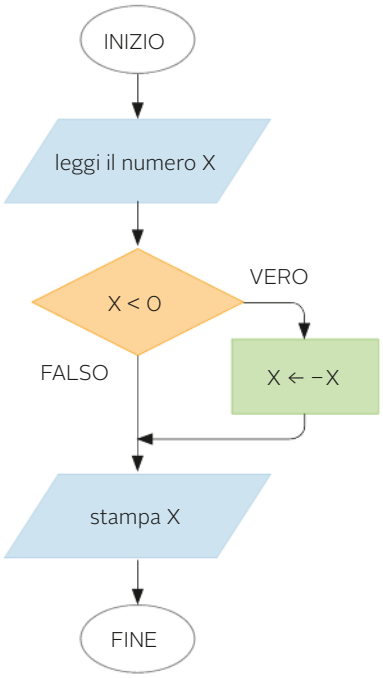


Figura 11.10 Un diagramma di flusso per il calcolo del valore assoluto di un numero dato come input.

Trovare le soluzioni, o radici, di una equazione di secondo grado

Vediamo adesso un caso in cui entrambi i rami di una selezione (vero e falso) richiedono di eseguire azioni.

Per un'equazione di secondo grado che ha la forma $ax^2 + bx + c = 0$, le radici o soluzioni sono date dall'espressione:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Le due radici dell'equazione, x_1 e x_2, si ottengono applicando la formula una volta con il segno «più» e un'altra con il segno «meno», là dove compare il simbolo $\pm$.

L'espressione sotto la radice quadrata, $\Delta = b^2 - 4ac$, si chiama *discriminante* perché, a seconda del suo valore, ci sono due soluzioni, oppure una sola, o nessuna.

Esistono radici reali soltanto se il discriminante $\Delta = b^2 - 4ac$ è maggiore o uguale a zero. Se invece $\Delta < 0$, la radice quadrata non si può calcolare e non ci sono soluzioni reali.

Prima di eseguire il calcolo delle radici, bisogna quindi fare un controllo sul valore del discriminante Δ (questo simbolo è la lettera greca «Delta» maiuscola).

Un possibile algoritmo è descritto dal diagramma di flusso della **figura 11.11**.

ALGORITMO PER TROVARE LE RADICI DI UN'EQUAZIONE DI 2° GRADO

1. Leggi i tre coefficienti a, b e c.
2. Calcola $\Delta = b^2 - 4ac$.
3. Se Δ è maggiore o uguale a zero, calcola con la formula di risoluzione le due radici x_1 e x_2 e producile come output.
4. Altrimenti, cioè se Δ è negativo, scrivi «Non ci sono radici reali».

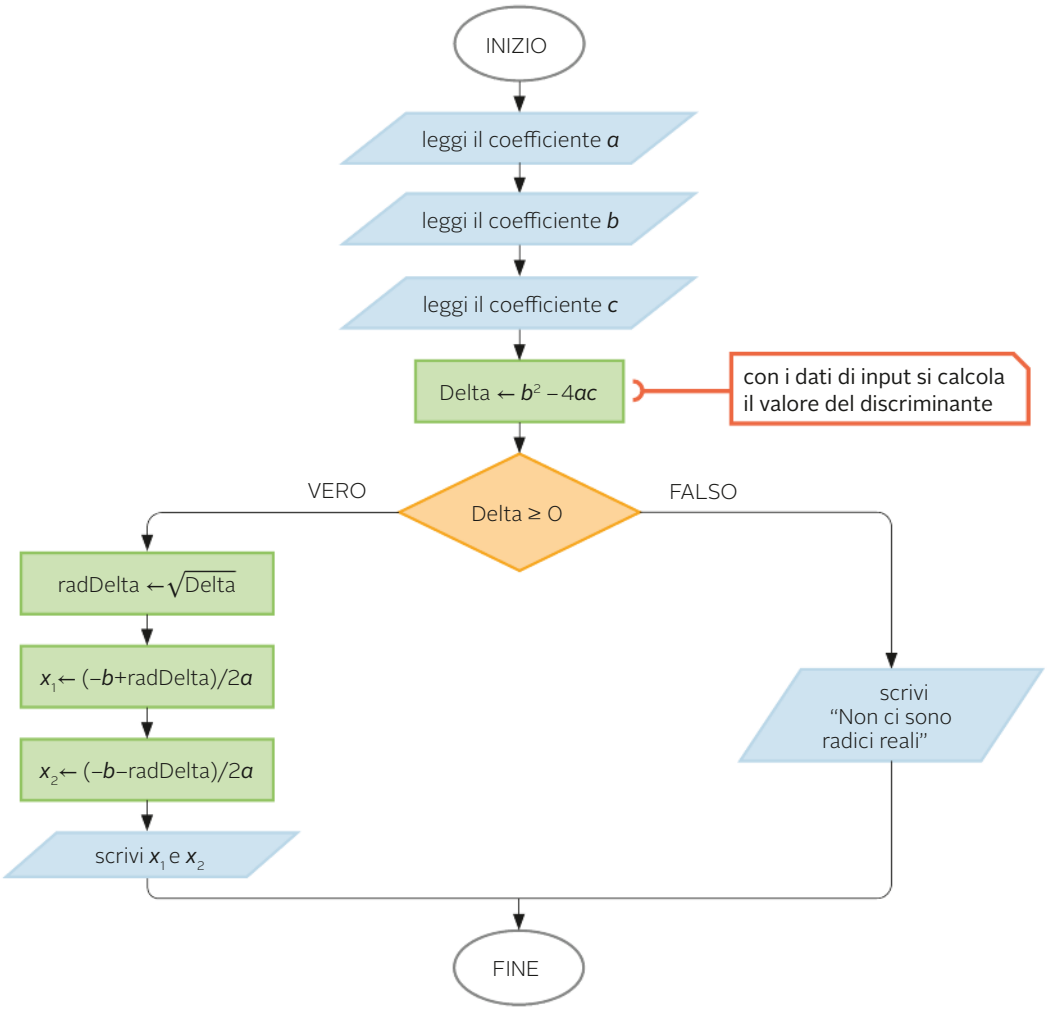


Figura 11.11 Un diagramma di flusso per la risoluzione di un'equazione di secondo grado.

UN ESERCIZIO AL VOLO

A quante vie è la selezione in questo diagramma di flusso?

.....

Le strutture nidificate

Una selezione può contenere, fra le azioni condizionate, ulteriori sotto-selezioni, che allora si dicono **nidificate** (o **annidate**) nella prima.

In tal caso occorre però evitare attentamente, mentre si scrive il programma, di «incrociare» tra loro le strutture condizionali in modo incoerente dal punto di vista logico, come esemplificato a destra nella **figura 11.12**.

A seconda del linguaggio di programmazione usato, questo errore di logica porterebbe a un programma che non è eseguibile oppure, se lo è, produce risultati non corretti.

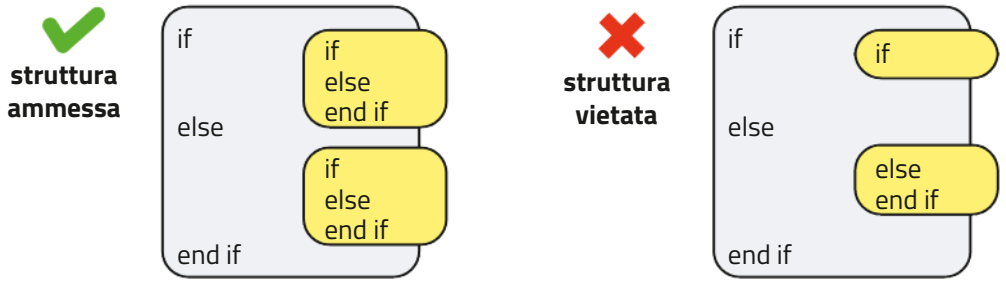


Figura 11.12 Selezioni nidificate ammesse e vietate.

Vediamo un esempio di algoritmo che contiene una struttura nidificata.

Nel caso dell'algoritmo per risolvere un'equazione di secondo grado, può essere utile introdurre una selezione nidificata come mostrato nella **figura 11.13**.

Qui nella parte inferiore del diagramma di flusso abbiamo esplicitato tutti i casi possibili per il discriminante.

Ora il caso di Δ positivo è trattato in modo distinto dal caso Δ = 0, in cui le due radici coincidono e i calcoli si possono semplificare.

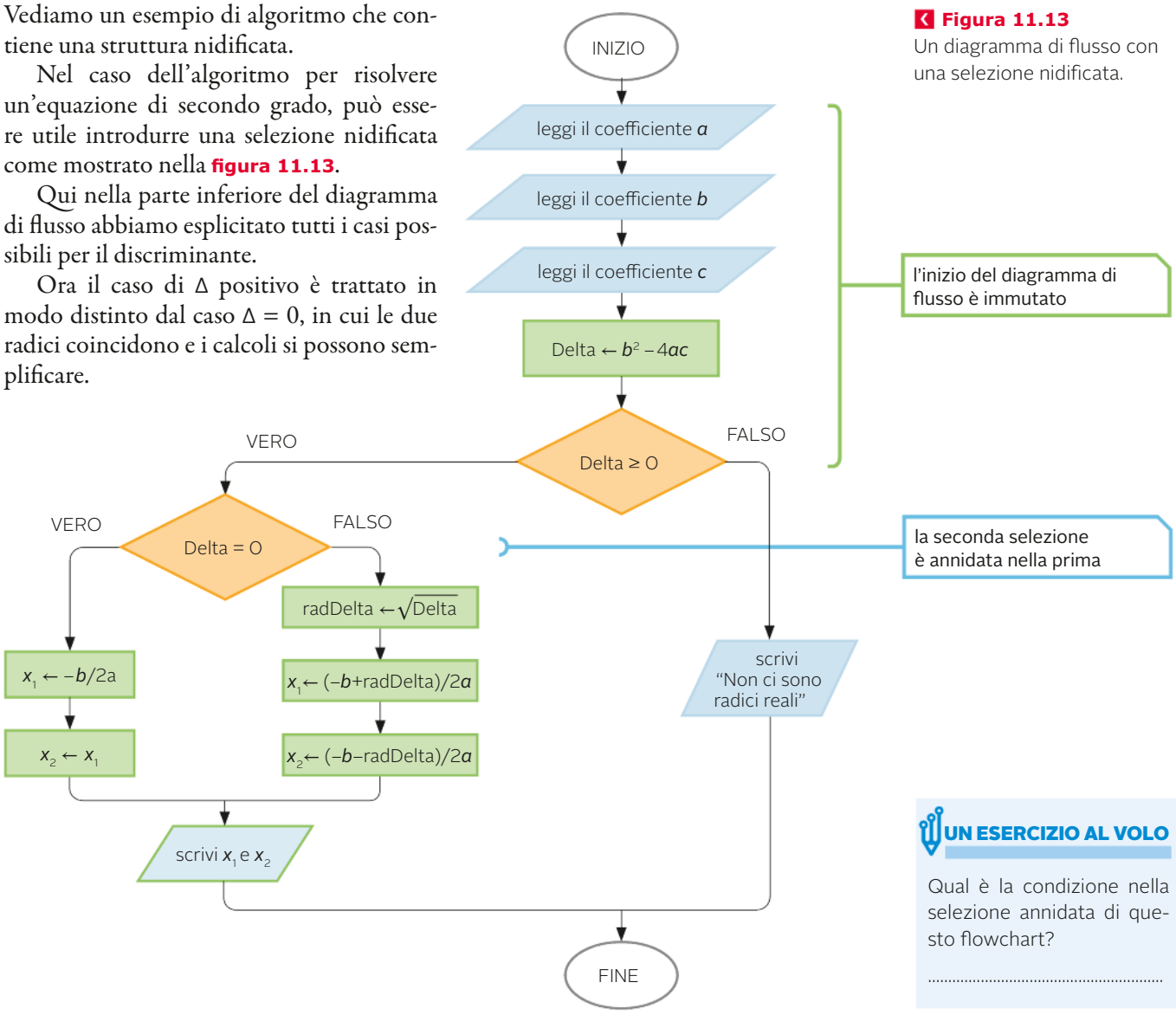


Figura 11.13 Un diagramma di flusso con una selezione nidificata.

UN ESERCIZIO AL VOLO

Qual è la condizione nella selezione annidata di questo flowchart?

.....

Gli operatori relazionali e gli operatori logici

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

Per esempio, la condizione «A > B» scritta in un blocco a forma di rombo significa «se è vero che il valore della variabile A è maggiore di quello della variabile B».

I predicati si esprimono usando gli **operatori relazionali**, che sono:

=	uguale
≠	diverso
>	maggiore
<	minore
≥	maggiore o uguale
≤	minore o uguale

UN ESERCIZIO AL VOLO

Come si scrive la condizione «A diverso da B»?

.....

Nei predicati spesso è utile usare anche gli **operatori** (o **connettivi**) **logici AND**, **OR** e **NOT**, che consentono di controllare simultaneamente più condizioni.

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

Se chiamiamo C_1 e C_2 due condizioni, ciascuna delle quali può essere vera (V) oppure falsa (F), allora la *congiunzione logica* **AND** e la *disgiunzione logica* **OR** sono definite dalle seguenti **tabelle di verità**:

C_1	C_2	C_1 AND C_2
V	V	V
V	F	F
F	V	F
F	F	F

C_1	C_2	C_1 OR C_2
V	V	V
V	F	V
F	V	V
F	F	F

Dunque la condizione composta C_1 **AND** C_2 è vera soltanto se sono vere *sia* C_1 *sia* C_2. Invece, perché C_1 **OR** C_2 sia vera, basta che sia vera *almeno una* tra le condizioni C_1 e C_2.

Come esempio concreto, considera questo algoritmo per decidere che fare quando ricevi una chiamata telefonica:

Rispondi se chi chiama è un amico AND è pomeriggio.



In base a questa regola di comportamento, risponderai alla chiamata soltanto se arriva da un amico di pomeriggio. Se un amico ti chiama di mattina o di sera non risponderai, e anche di pomeriggio non risponderai, se chi chiama non è un amico.

Se invece la regola fosse:

Rispondi se chi chiama è un amico OR è pomeriggio.

allora risponderesti alla chiamata in molti più casi: ogni volta che ti chiama un amico (in qualsiasi momento del giorno) e sempre al pomeriggio (chiunque sia a chiamare).

Negli algoritmi può accadere di controllare simultaneamente anche più di due condizioni. In tal caso:

- se si usa la congiunzione logica **AND**, il predicato composto:

  condizione1 AND condizione2 AND condizione3 AND ...

  è VERO soltanto se *tutte* le singole condizioni sono soddisfatte, mentre è FALSO se *anche una sola* tra le condizioni non è soddisfatta;

- se si usa la disgiunzione logica **OR**, il predicato composto:

  condizione1 OR condizione2 OR condizione3 OR ...

  è VERO se *almeno una* tra le condizioni è soddisfatta, mentre è FALSO soltanto se *nessuna* tra le singole condizioni è soddisfatta.

L'operatore logico NOT nega la condizione a cui è applicato. Così, per esempio, questa regola per il telefono:

Rispondi se NOT è mattina.

significa che risponderai a tutte le chiamate, tranne a quelle che arrivano al mattino.

L'operatore NOT, applicato a una condizione C, è definito da questa tabella di verità:

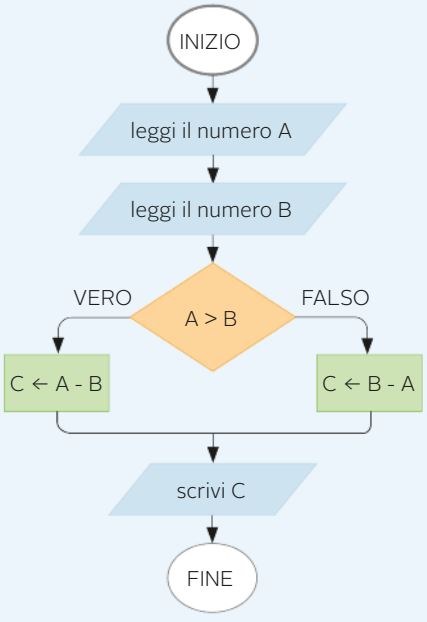
C	NOT C
V	F
F	V

RIPASSA I CONCETTI-CHIAVE

- La **struttura condizionale** o **selezione** prevede l'esecuzione di istruzioni diverse, a seconda che una condizione di controllo risulti vera oppure falsa.
- Nei linguaggi di programmazione la **selezione a due vie** si rappresenta con le istruzioni **if** (seguita dalle istruzioni da eseguire se la condizione è vera) ed **else** (seguita dalle istruzioni da eseguire se la condizione è falsa).
- I **predicati**, cioè le condizioni di cui si controlla il valore di verità, si esprimono usando gli **operatori relazionali** «uguale a» (=), «diverso da» (≠), «minore di» (<), «maggiore di» (>), «minore o uguale a» (≤) e «maggiore o uguale a» (≥).
- Un predicato può anche combinare tra loro **più condizioni diverse** mediante gli **operatori logici AND**, **OR** e **NOT**.

ESERCIZI

1. Spiega a che cosa serve l'algoritmo descritto da questo diagramma di flusso.



2. Che cos'altro andrebbe cambiato nel diagramma di flusso dell'esercizio precedente, se nel blocco di controllo si scegliesse di scrivere la condizione «A<B» invece di «A>B»?

3. Disegna su un foglio il diagramma di flusso per un algoritmo che, dato un numero qualsiasi, dica se il numero è positivo o negativo. Per quale ragione è utile che quest'algoritmo contenga una selezione nidificata?

4. Data una variabile numerica **NUM**, scrivi una condizione che permetta di controllare se il suo valore è compreso tra 3 e 7. Usa un operatore logico.

5. Data una variabile testuale **giorno**, che può assumere qualsiasi valore da **lunedì** a **domenica**, scrivi una condizione che permetta di controllare se il valore di **giorno** corrisponde al fine settimana. Usa un operatore logico.

11.4 Le strutture iterative

Si usa una **struttura iterativa**, o **ciclo iterativo**, quando un algoritmo richiede che certi gruppi di istruzioni siano eseguiti ripetutamente, per un numero di volte predefinito oppure fino al verificarsi di una data condizione.

Iterare significa ripetere, perciò l'esecuzione ripetuta di uno stesso gruppo di istruzioni si chiama **iterazione**, o anche **ciclo**, perché la ripetizione avviene in modo ciclico.

Ora vedremo in quali modi questa struttura si può realizzare seguendo le regole della programmazione strutturata.

Il ciclo di tipo *for*

Il **ciclo di tipo for** si usa quando il numero delle iterazioni è prestabilito.

La struttura generale di questo tipo di ciclo si può riassumere così:

LA STRUTTURA DEL CICLO DI TIPO `for`

```
for (condizione da verificare)
    {istruzioni da eseguire se la condizione è vera;}
    poi torna all'istruzione for per l'iterazione successiva
end for
```

Qui **end for** indica il termine del ciclo: quando la condizione risulta falsa, non si itera più e si procede con le istruzioni successive del programma.

La condizione del ciclo dipende da una variabile intera detta **contatore**, perché conta il numero delle iterazioni eseguite.

Di solito questa variabile è chiamata **i** e il suo valore viene incrementato di un'unità a ogni esecuzione del ciclo.

Va notato che nel ciclo **for** il controllo della condizione da soddisfare *precede* il gruppo delle istruzioni da iterare. Può quindi accadere che le istruzioni previste nel ciclo non siano mai eseguite.

Consideriamo come esempio il problema di calcolare, dato un qualsiasi valore di **n**, la somma dei primi **n** numeri interi positivi, ossia il valore della seguente espressione:

$$1 + 2 + 3 + \ldots + (n-1) + n.$$

Si tratta dunque di ripetere **n** volte un'operazione di addizione.

Un possibile algoritmo è dato dal diagramma di flusso della **figura 11.14**, che si può descrivere così:

ALGORITMO PER SOMMARE I PRIMI n INTERI

1. Assegna come input il valore della variabile **n**.
2. Inizia azzerando la variabile **somma** e parti dal valore 1 per il contatore **i**.
3. Se il valore del contatore **i** è minore o uguale a **n**, aggiungi il suo valore a **somma**, poi incrementa di un'unità il valore di **i** e ripeti il test di controllo.
4. Altrimenti, cioè se il contatore **i** è maggiore di **n**, produci come output il valore di **somma**.

Merita sottolineare alcuni aspetti di quest'algoritmo:

- il numero **n** delle iterazioni da fare è noto fin dall'inizio, cioè dal momento della lettura dell'input;
- il valore di **i** viene aumentato a ogni iterazione, perciò prima o poi **i** supererà **n**; quindi il ciclo avrà sicuramente un termine;
- nel caso particolare in cui l'input **n** vale 0, la condizione sarà falsa fin dall'inizio e le istruzioni del ciclo non verranno mai eseguite.

UN ESERCIZIO AL VOLO

In quale caso si può usare il ciclo di tipo **`for`**?

.....

.....

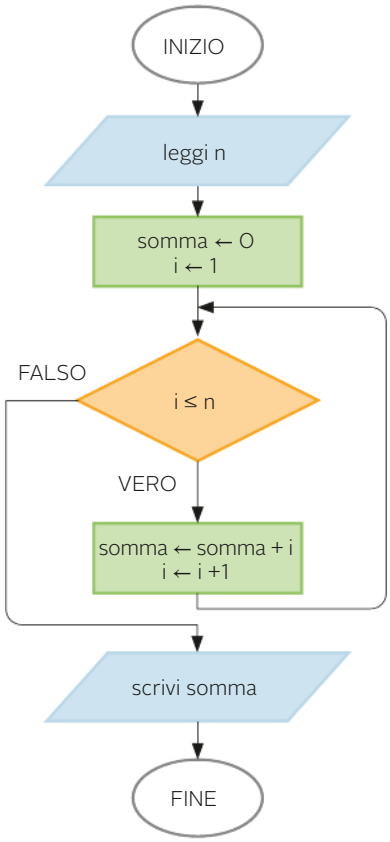


Figura 11.14 Un diagramma di flusso basato sul ciclo di tipo **for**.

I cicli di tipo *while* e *do*

I cicli di tipo **while** e i cicli di tipo **do** si usano per codificare algoritmi contenenti iterazioni quando non si conosce *a priori* il numero dei cicli da effettuare, perché quel numero dipende dal verificarsi o meno di una certa condizione.

La differenza principale fra i due costrutti sta nella posizione della condizione per la permanenza all'interno del ciclo, come mostrano gli schemi della **figura 11.15**.

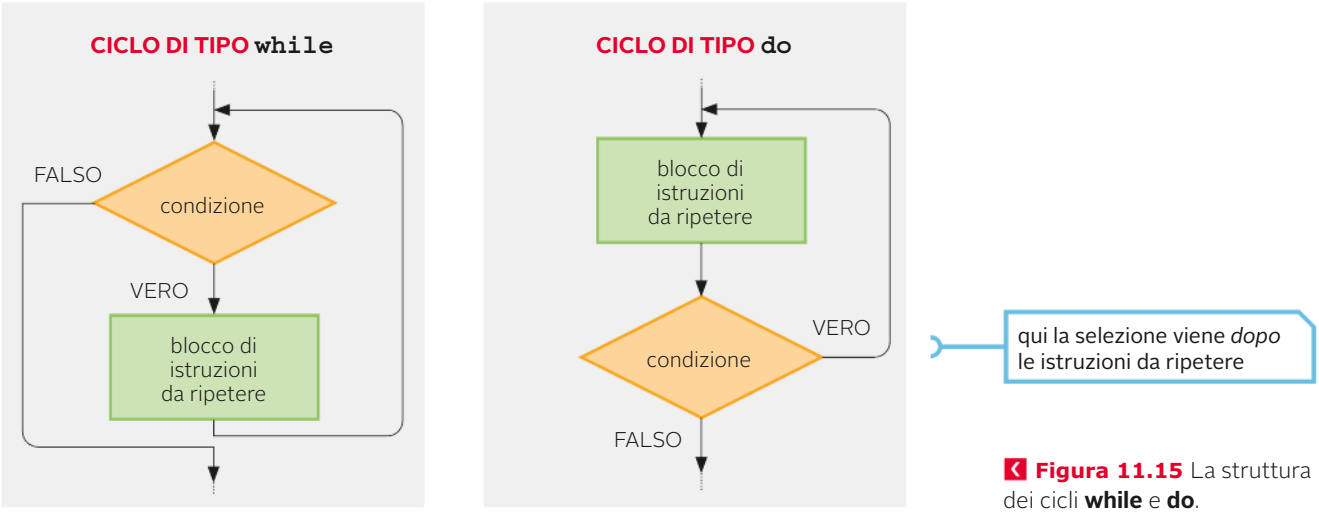


Figura 11.15 La struttura dei cicli **while** e **do**.

```
while (condizione da verificare)
   {istruzioni da eseguire
    finché la condizione è vera;}
end while
```

```
do
   {istruzioni da eseguire;}
while (condizione da verificare)
se la condizione è vera, torna a do
```

- Nel ciclo di tipo **while** la condizione *precede* le istruzioni da ripetere, come nel ciclo di tipo **for**; è possibile che il test dia subito come risultato **falso**, nel qual caso le istruzioni del ciclo sono trascurate e il programma prosegue con le istruzioni successive.
- Nel ciclo di tipo **do** la condizione è controllata invece *alla fine* del blocco di istruzioni da ripetere; tali istruzioni verranno dunque eseguite almeno una volta.

I cicli di tipo **while** e di tipo **do** sono equivalenti tra loro all'atto pratico.

UN ESERCIZIO AL VOLO

Quale ciclo si può usare se il numero delle iterazioni non è prestabilito?

.....

Vediamo ora un esempio concreto di applicazione dei due costrutti.

Supponiamo di voler rispondere a questa domanda: se sommiamo i numeri della sequenza degli interi positivi (così: 1 + 2 + 3 + ...), a quale numero dovremo arrivare perché la somma ottenuta superi il valore 100?

La soluzione di questo problema richiede un algoritmo di tipo iterativo.

Per un ciclo di tipo **while**, l'algoritmo può avere la forma seguente:

ALGORITMO BASATO SUL CICLO `while`

1. Inizializza con il valore 0 la variabile numero intero **i** e la variabile somma **s**.
2. Controlla se il valore della somma **s** è minore del valore-soglia 100.
3. Se ciò risulta vero, aumenta il valore di **i** di un'unità (cioè passa al numero intero successivo) e aggiungilo alla somma **s**.
4. Altrimenti, cioè se invece risulta che **s** è maggiore o uguale a 100, esci dal ciclo e produci come output il valore di **i**.

Otterremo così come risultato l'ultimo valore del contatore **i** che è stato utilizzato nel ciclo. Questo valore è proprio la risposta alla domanda che ci eravamo posti: infatti alla fine **i** è il numero degli interi che sono stati sommati per superare 100.

Per un ciclo di tipo **do** un algoritmo che risolve lo stesso problema è invece il seguente:

ALGORITMO BASATO SUL CICLO do

1. Inizializza con il valore 0 la variabile numero intero **i** e la variabile somma **s**.
2. Inizio del ciclo: aumenta il valore di **i** di un'unità (cioè passa all'intero successivo) e aggiungilo a **s**.
3. Controlla se il valore della somma **s** è minore di 100.
4. Se **s < 100**, torna all'inizio ciclo, cioè al punto **2**; altrimenti esci dal ciclo e produci come output il valore di **i**.

Ai due algoritmi appena visti corrispondono il diagrammi di flusso della **figura 11.16**:

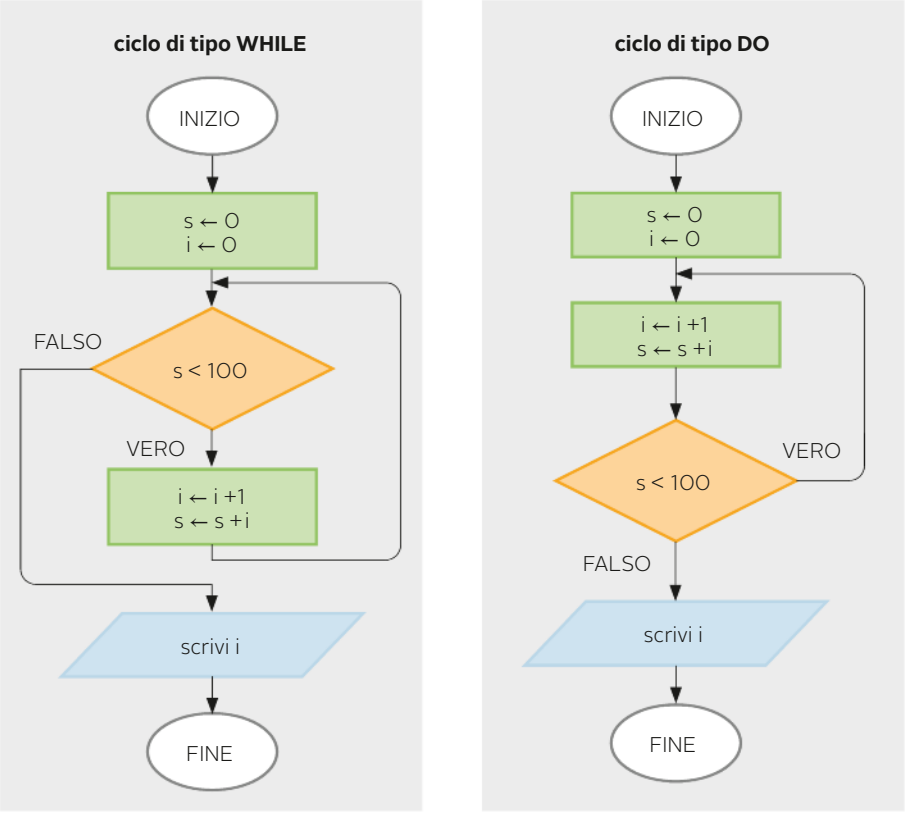


Figura 11.16 Diagrammi di flusso basato sul ciclo **while** e sul ciclo **do** per l'algoritmo che calcola per quale valore di **i** si supera il valore 100.

I due algoritmi che abbiamo scritto sono equivalenti tra loro, ed entrambi daranno come output il valore 14 (infatti la somma degli interi da 1 a 14 vale 105).

L'algoritmo scritto per il valore-soglia 100 si può facilmente generalizzare: basta leggere all'inizio come input un numero intero qualsiasi, assegnarlo a una variabile **soglia** e sostituire poi nel test la condizione **s < soglia**.

Per riassumere tutto ciò che abbiamo visto sui cicli, quando si vuole progettare una struttura iterativa in un algoritmo, occorre stabilire:

- quali azioni debbano essere inserite prima dell'avvio del ciclo;
- quale tipo di ciclo sia più adatto per l'algoritmo in esame;
- quali siano le azioni da ripetere a ogni iterazione;
- quale sia la condizione che pone termine all'iterazione;
- quali azioni vadano eseguite dopo la fine del ciclo.

Come negli esempi che abbiamo visto, è molto comune avere nei cicli una variabile che aumenta di un'unità a ogni iterazione; perciò i linguaggi di programmazione di solito hanno istruzioni concise per farlo.

Nei linguaggi C e C++, per esempio, basta scrivere **i++** per incrementare automaticamente di un'unità una variabile intera **i** che funge da contatore.

UN ESERCIZIO AL VOLO

Come si può scrivere più concisamente in C/C++ l'assegnazione **i ← i +1**?

.....

Evitare l'istruzione GO TO

Nei primi linguaggi di programmazione, quando si voleva ritornare all'inizio di un blocco di istruzioni oppure uscirne, era comune l'uso dell'istruzione **go to**.

Questa istruzione, detta anche *salto incondizionato*, rimanda a un'altra istruzione del programma senza che debba essere soddisfatta alcuna condizione.

L'istruzione è permessa anche da alcuni linguaggi odierni, ma **il suo uso è fortemente sconsigliato** nell'ambito della programmazione strutturata.

L'istruzione **go to**, infatti, rende meno chiara la struttura dei programmi, in cui diventa più difficile seguire il flusso di controllo; aumenta, in altre parole, il rischio di produrre quello che in inglese è chiamato *spaghetti code*, cioè un codice aggrovigliato come gli spaghetti in un piatto di pastasciutta.

Inoltre l'uso del **go to** facilita la creazione di situazioni indesiderate, in cui per esempio un blocco di istruzioni si ripete all'infinito, oppure il processo di esecuzione del programma termina bruscamente in modo imprevisto.

RIPASSA I CONCETTI-CHIAVE

- L'**iterazione** o **ciclo** è la ripetizione di uno stesso gruppo di istruzioni con dati diversi all'interno di un algoritmo.
- Quando si vuole iterare per un numero prestabilito di volte, si usa il **ciclo di tipo for**, in cui una variabile intera (contatore) viene incrementata di un'unità a ogni iterazione.
- Quando invece il numero dei cicli da effettuare non è prevedibile *a priori*, si usa il **ciclo di tipo while** (in cui la selezione precede il blocco di istruzioni da ripetere) oppure il **ciclo di tipo do** (in cui la selezione è alla fine).

ESERCIZI

1. Supponi di voler calcolare, dato **n**, il prodotto dei primi **n** numeri interi positivi. Nell'algoritmo puoi usare un ciclo di tipo **for**? Motiva la tua risposta.

.....

.....

2. L'algoritmo descritto da questo diagramma di flusso calcola, con un ciclo **while**, la parte intera del rapporto tra due numeri interi **m** e **n** dati come input.

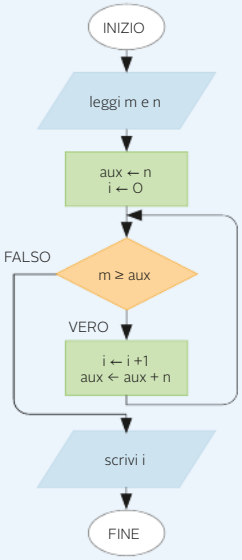
Prova ad applicare il flowchart con diversi valori di input, per capire la logica del suo funzionamento.

Qual è il risultato dell'algoritmo nel caso in cui **n > m**?

.....

Qual è il risultato nel caso in cui **m = n**?

.....



3. Supponi di voler trovare il valore di **n** tale che il prodotto dei primi **n** interi superi un certo valore dato. Nell'algoritmo puoi usare un ciclo di tipo **for**? Motiva la tua risposta.

.....

.....

4. Disegna nel riquadro un diagramma di flusso che equivalga a quello dell'esercizio qui a sinistra, ma che usi un ciclo di tipo **do**.

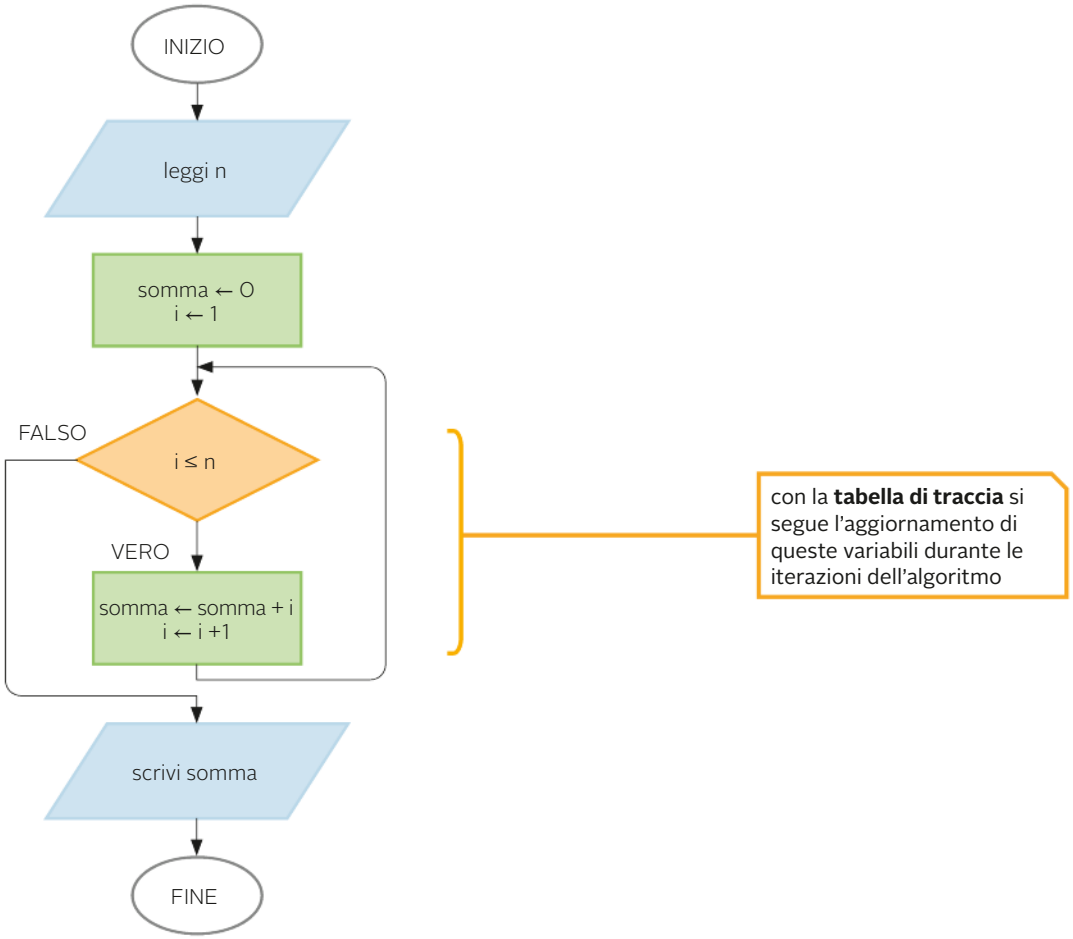
11.5 La tabella di traccia

Quando si formula un algoritmo, prima di iniziare a codificarlo in un linguaggio di programmazione conviene sempre verificare che l'algoritmo sia corretto. Un modo per farlo è costruire la cosiddetta **tabella di traccia**.

> La **tabella di traccia** è una tecnica di verifica puntuale per mettere alla prova, passo per passo, il funzionamento di un algoritmo.

Per spiegare come si costruisce una tabella di traccia, useremo l'algoritmo di somma dei numeri interi che abbiamo visto come esempio di applicazione del ciclo **for**.

Il diagramma di flusso dell'algoritmo è quello già visto nella **figura 11.14**, che riproduciamo per comodità qui sotto.



ALGORITMO PER SOMMARE I PRIMI n INTERI

1. Assegna come input il valore della variabile **n**.
2. Inizia azzerando la variabile **somma** e parti dal valore 1 per il contatore **i**.
3. Se il valore del contatore **i** è minore o uguale a **n**, aggiungi il suo valore a **somma**, poi incrementa di un'unità il valore di **i** e ripeti il test di controllo.
4. Altrimenti, cioè se il contatore **i** è maggiore di **n**, produci come output il valore della variabile **somma**.

Nell'algoritmo intervengono tre variabili:

- **n** è il numero intero massimo a cui arrivare nell'iterazione;
- **i** ha la doppia funzione di contatore e di intero da sommare in ogni ciclo;
- **somma** è il risultato che si accumula progressivamente a ogni ciclo.

Useremo allora una tabella a quattro colonne per tenere traccia, durante il calcolo iterativo, dei valori delle tre variabili e del risultato del test di selezione.

UN ESERCIZIO AL VOLO

Qual è la variabile di input in questo algoritmo?

.....

Consideriamo per esempio il caso in cui il valore di input dato al programma è 3.

Ecco allora come una tabella di traccia permette di seguire l'evoluzione del calcolo basato sull'algoritmo:

1. il valore 3 di input viene assegnato alla variabile **n**, poi **i** è inizializzata con il valore 1; così il primo test di confronto dà come risultato **vero**, perché **i** è minore di **n**; il programma perciò effettua una prima iterazione e aggiunge 1 al valore di **somma**, che era stata inizializzata con 0;

n	i	i ≤ n	somma
3	1	vero	0 + 1 = 1

2. all'inizio del secondo ciclo il valore di **i** è diventato 2; il test è ancora soddisfatto, perciò il programma prosegue effettuando la seconda iterazione e aggiorna la variabile **somma** aggiungendo al suo valore il valore di **i**, cioè 2;

n	i	i ≤ n	somma
3	1	vero	0 + 1 = 1
	2	vero	1 + 2 = 3

3. all'inizio del terzo ciclo il valore di **i** è diventato uguale a quello di **n**; il test è ancora soddisfatto, giacché nella condizione c'è minore o *uguale*, perciò il programma prosegue con la terza iterazione e aggiunge 3 al valore di **somma**;

n	i	i ≤ n	somma
3	1	vero	0 + 1 = 1
	2	vero	1 + 2 = 3
	3	vero	3 + 3 = 6

4. al quarto ciclo invece il valore di **i** supera quello di **n**; il test di confronto adesso dà come risultato **falso** e il programma esce dall'iterazione: il valore di **somma** non subisce altri cambiamenti e viene stampato come output.

n	i	i ≤ n	somma
3	1	vero	0 + 1 = 1
	2	vero	1 + 2 = 3
	3	vero	3 + 3 = 6
	4	falso	6

la condizione ora è **falsa**: il ciclo termina qui

Nelle verifiche con la tabella di traccia è bene usare per prima cosa dati di input qualsiasi, scelti in modo arbitrario, come abbiamo appena fatto.

Per controllare che l'algoritmo funzioni correttamente in tutte le situazioni, però, è anche bene provarlo con dati di input che rappresentino casi-limite significativi.

Nel nostro esempio, un caso-limite è quello in cui il valore di input per **n** è zero. Si tratta di una scelta bizzarra per gli scopi dell'algoritmo; tuttavia è una scelta possibile, ed è bene verificare che l'algoritmo «si comporti bene» anche in questo caso.

Quando l'assegnazione iniziale è **n** ← **0**, la tabella di traccia diventa:

n	i	i ≤ n	somma
0	1	falso	0

Il test di selezione dà subito come risultato **falso** (giacché 1 > 0), perciò il programma non farà nessuna iterazione: il valore della variabile **somma** rimane così uguale a zero, che è un risultato sensato.

UN ESERCIZIO AL VOLO

Quante iterazioni farà il ciclo se il dato di input *n* vale 5?

.....

Verificare il funzionamento dell'algoritmo in casi noti

La **tabella di traccia** permette di controllare il funzionamento di un algoritmo.

Dopo aver fatto i controlli con la tabella di traccia, si può procedere con la scrittura del codice e l'esecuzione del programma.

Spesso accade che vi siano particolari valori dei dati di input per i quali conosciamo già la soluzione del problema. In tal caso è comodo e utile eseguire il programma proprio con quei particolari valori di ingresso, come ulteriore verifica di sicurezza.

Nel caso del nostro algoritmo di somma dei numeri interi, esiste un «trucco» che rende molto semplice calcolare il valore della variabile **somma** per particolari valori di **n**.

La **figura 11.17** mostra, per esempio, come ragionare nel caso in cui **n** vale 10.

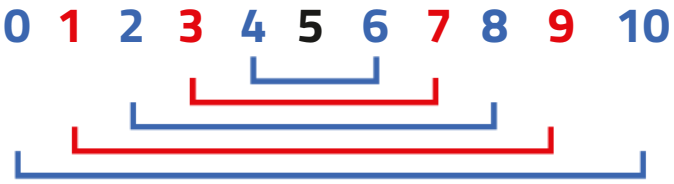


Figura 11.17 Il «trucco» per sommare a mente i primi 10 numeri interi.

Si suddividono in coppie i numeri da 0 a 10, in modo tale che la somma di ciascuna coppia valga 10, in questo modo: 0+10, 1+9, 2+8, 3+7, 4+6.

Si ottengono così 5 coppie (per una somma totale di 50) e va ancora aggiunto il numero 5, che è rimasto in mezzo da solo: per **n** ← **10**, dunque, il valore finale di **somma** deve essere **55**.

Applicando esattamente la stessa logica, è facile convincersi che la somma dei numeri da 1 a 100 vale 5050, quella dei numeri da 1 a 1000 vale 50 500, e così via. Più in generale, si può dimostrare che la somma dei primi n numeri interi è data dalla formula $n(n+1)/2$.

Per verificare il funzionamento del programma, perciò, si potrà metterlo alla prova controllando che per **n** ← **100** l'output **somma** valga 5050, per **n** ← **1000** valga 50 500, e così via.

Per fare un altro esempio, immagina di aver scritto un algoritmo che calcola il valore di una potenza, dati come input i valori della base e dell'esponente.

In questo caso la prima cosa da controllare durante il test è che, per qualsiasi valore della base, il programma dia come risultato 1 quando l'esponente vale 0.

RIPASSA I CONCETTI-CHIAVE

- La **tabella di traccia** è una tecnica per verificare in modo puntuale, passo per passo, la correttezza di un algoritmo.
- Con dati di input scelti a caso si mette alla prova il **funzionamento** di un algoritmo; è sempre bene testarlo anche nei casi-limite.
- Quando la soluzione è già nota per certi valori dei dati di input, è utile provare anche questi valori per testare il corretto funzionamento del programma.

ESERCIZI

1. Costruisci una tabella di traccia per l'algoritmo a sinistra nella **figura 11.16** (ciclo **while**); usa però, come valore-soglia, 20 invece di 100.

2. Ripeti poi l'esercizio precedente per l'algoritmo a destra nella **figura 11.16** (ciclo **do**), sempre con valore-soglia 20.

Informatica e società

AGENDA 2030 EDUCAZIONE CIVICA

10 RIDURRE LE DISUGUAGLIANZE

LA PRIMA PROGRAMMATRICE

Fin dal Seicento gli scienziati si erano resi conto di quanto sarebbe stato vantaggioso poter automatizzare i calcoli aritmetici.

Isaac Newton, per esempio, poté formulare la sua teoria della gravitazione grazie al fatto che prima di lui Johannes Kepler aveva scoperto le tre celebri leggi che regolano il moto dei pianeti.

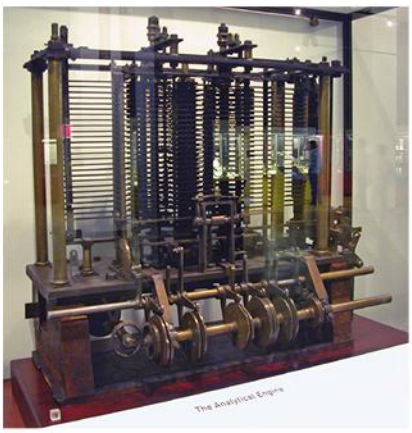
Per elaborare le sue leggi, Kepler aveva dovuto analizzare molte migliaia di dati numerici, raccolti dall'astronomo danese Tycho Brahe in decenni di osservazioni del moto dei pianeti nel cielo notturno.

L'*Analytical Engine* di Babbage

I laboriosi calcoli ripetitivi fatti a mano da Kepler avrebbero richiesto una frazione del tempo se egli avesse avuto a disposizione una macchina calcolatrice automatica.

Questo tipo di macchina fu progettato per la prima volta nel 1834 dal matematico inglese **Charles Babbage**, che non riuscì però a completarne la costruzione.

L'*Analytical Engine* («motore analitico») di Babbage era un dispositivo meccanico alimentato a vapore che comprendeva un'unità logico-aritmetica, un'unità di controllo e una memoria interna, proprio come i computer moderni.



Una ricostruzione dell'*Analytical Engine* di Babbge.

I dati di input erano inseriti mediante schede perforate. Dopo l'elaborazione, per l'output erano previsti una stampante, un plotter per produrre grafici e anche una campanella.

L'intuizione di Ada Lovelace

Babbage mostrò un prototipo della macchina alla giovane **Ada Lovelace**, figlia diciottenne del poeta Lord Byron.

Ada era una ragazza speciale per la sua epoca: appassionata di matematica e di meccanica, non si lasciava intimorire dall'idea che quelle fossero «cose da uomini».

Ada fu affascinata dal progetto e nei decenni successivi, dopo aver approfondito gli studi matematici, collaborò allo sviluppo della macchina di Babbage.

Fu proprio Ada a capire che la macchina poteva avere applicazioni generali, che andavano al di là delle semplici operazioni aritmetiche.

Il linguaggio di programmazione per la macchina di Babbage prevedeva istruzioni simili a quelle dei linguaggi moderni, come strutture condizionali e cicli iterativi.

Ada usò questo linguaggio per scrivere la prima trasposizione informatica di un **algoritmo**, un procedimento generale per calcolare i *numeri di Bernoulli*, una particolare successione di numeri razionali.

Per questa impresa Ada Lovelace è considerata la prima programmatrice della storia. Oggi esiste un linguaggio di programmazione chiamato **Ada** in suo onore.



Un ritratto di Ada Lovelace e il suo algoritmo.

RIFLETTICI SU

1. Per quale ragione prima dell'epoca di Ada Lovelace nessuno aveva pensato di dedicarsi alla programmazione di algoritmi?

.....

.....

.....

2. Quale caratteristica di Ada Lovelace ne ha fatto la candidata ideale per diventare la prima programmatrice della storia?

.....

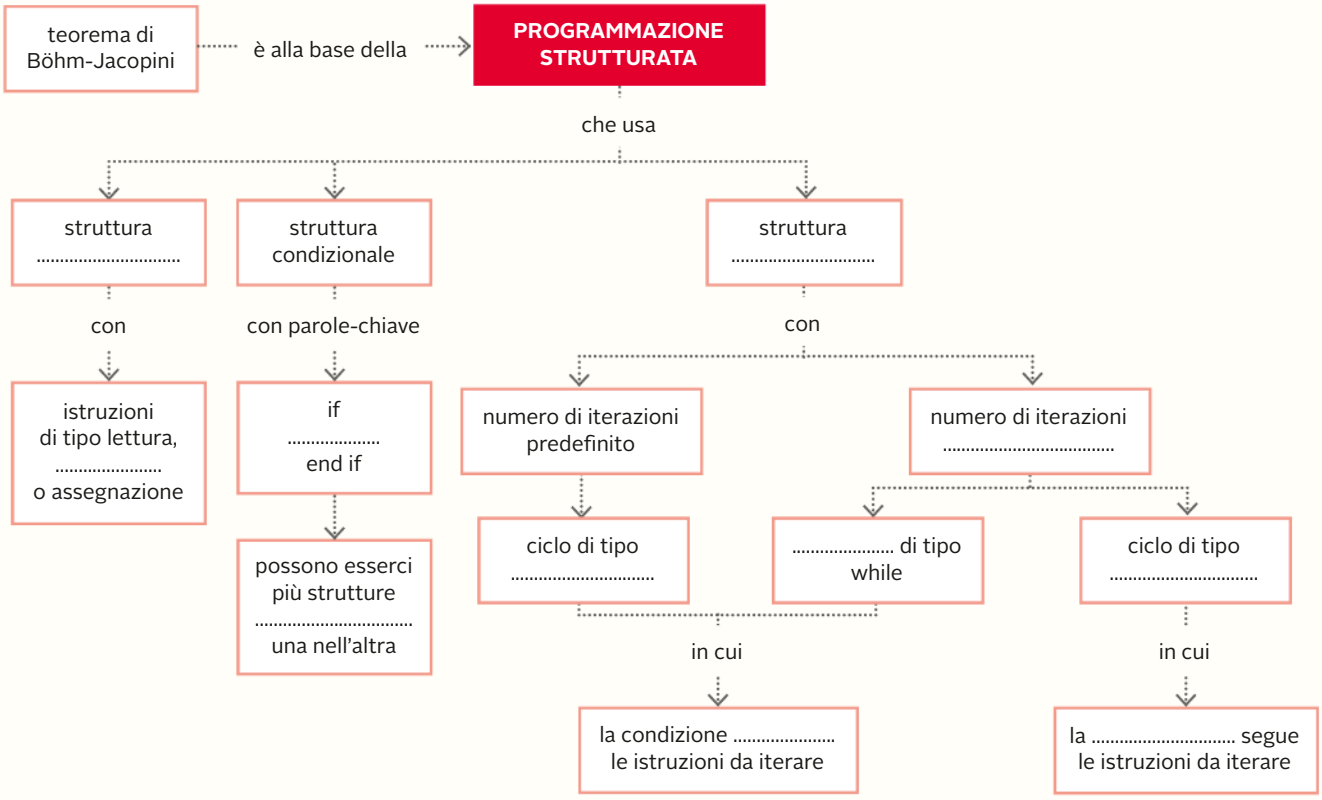
.....

.....

Altri esercizi online su ZTE

ESERCIZI DI RIEPILOGO

1 Completa la mappa con le parole:
annidate • ciclo • condizione • do • else • for • iterativa • precede • scrittura • sequenziale • variabile



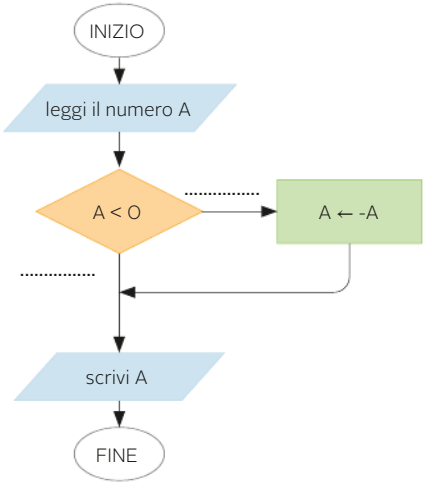
2 **a.** Che cosa corrisponde nel computer a una *variabile* di un programma?

.....

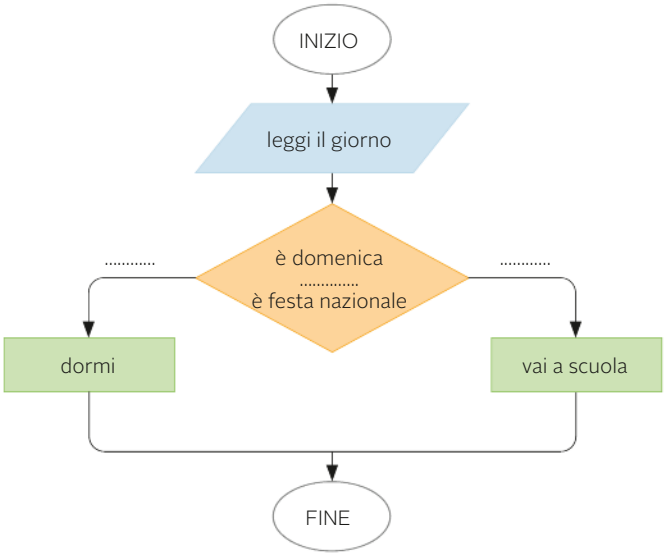
b. Che cosa distingue una *costante* da una variabile?

.....

3 Questo diagramma di flusso per il calcolo del valore assoluto di un numero è incompleto. Completalo aggiungendo i due elementi mancanti vicino al rombo.



4 Completa questo diagramma di flusso per un algoritmo che risolve il problema di che cosa fare al mattino.



Come va modificato l'algoritmo se si ha la fortuna di non andare a scuola il sabato?

.....

.....

.....

5 Immagina che all'interno di un algoritmo si debba assegnare a una variabile il valore 2. Se disegni il diagramma di flusso dell'algoritmo, quale forma dovrà avere il blocco relativo a questo passaggio?

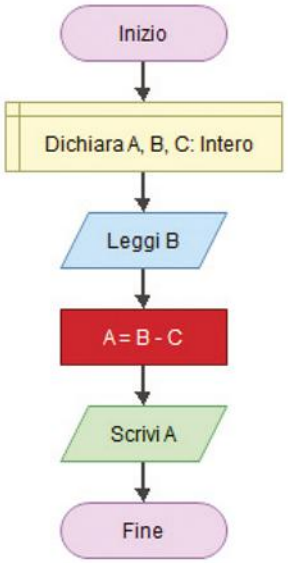
.....

6 Che cosa viene sovrascritto nel computer quando si assegna un valore a una variabile?

.....

.....

7 Quando si esegue in Flowgorithm il semplice algoritmo della figura qui sotto, il programma segnala un errore e colora di rosso il blocco centrale.



Che cosa c'è di sbagliato nell'algoritmo?

.....

.....

.....

8 Come viene rappresentata in Flowgorithm l'operazione di assegnazione del valore a una variabile?

.....

9 Come si dice in inglese «se...altrimenti»?

.....

10 Nei cicli iterativi di solito è presente una variabile intera il cui valore si aggiorna ripetutamente; questa variabile è chiamata *contatore*. Per quale ragione ha questo nome?

.....

11 Quale strumento permette di controllare passo per passo il funzionamento di un algoritmo senza bisogno di eseguirlo al computer?

.....

12 Quali sono i tre costrutti su cui si basa la programmazione strutturata?

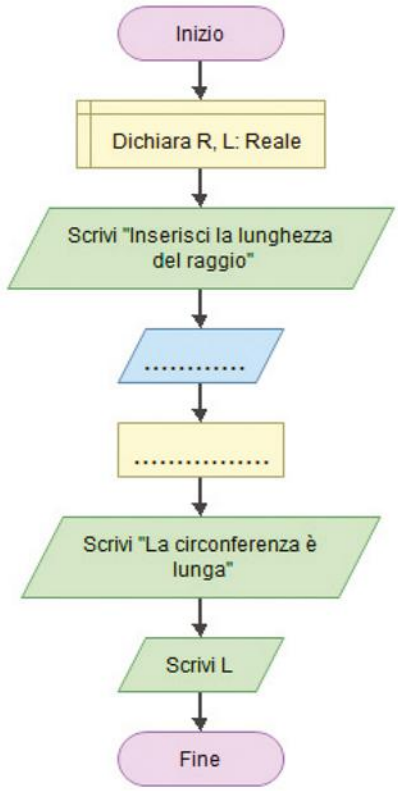
a......

b......

c......

13 Completa il diagramma di flusso di Flowgorithm, che calcola la lunghezza di una circonferenza quando l'utente specifica come input la lunghezza del raggio.

Scrivi ciò che manca al posto dei puntini.



14 Costruisci in Flowgorithm il diagramma di flusso per un algoritmo che calcola l'area di un cerchio che ha il raggio specificato dall'utente (può essere un numero qualsiasi).

Verifica che l'algoritmo produca come risultato 314 quando il valore di input è 10.

Salva il lavoro con il nome **area_cerchio.fprg**.

15 **a.** Quali sono i quadrilateri che hanno quattro lati uguali AND quattro angoli uguali?

.....

b. Quali sono i quadrilateri che hanno quattro lati uguali OR quattro angoli uguali?

.....

16 Completa con due operatori logici la condizione composta che stabilisce quando puoi sicuramente dormire fino a tardi la mattina.

è Natale ..... è Pasqua ..... è agosto

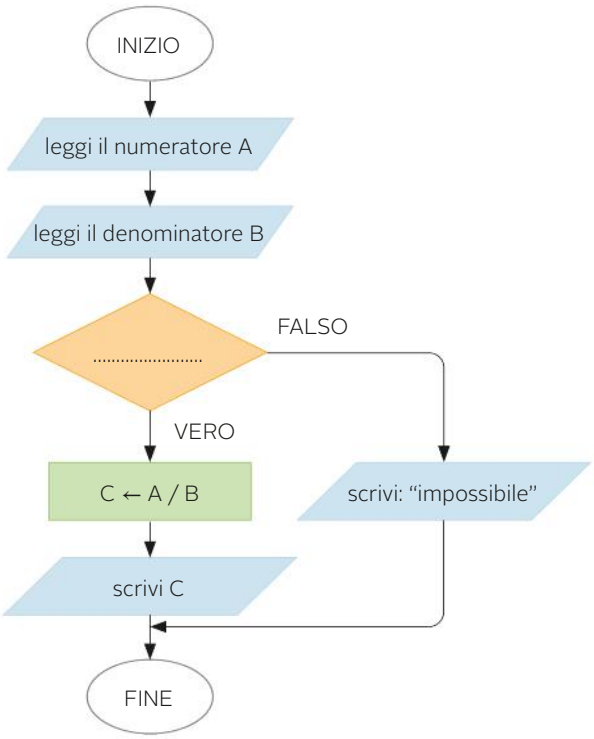
17 Completa con due operatori logici la condizione che stabilisce quando *non puoi* guardare sullo smartphone un video della tua cantante preferita.

c'è lezione ..... (è vacanza ..... la batteria è scarica)

18 Scrivi a che cosa serve l'algoritmo della figura e completa il diagramma di flusso scrivendo la giusta condizione nel blocco di test.

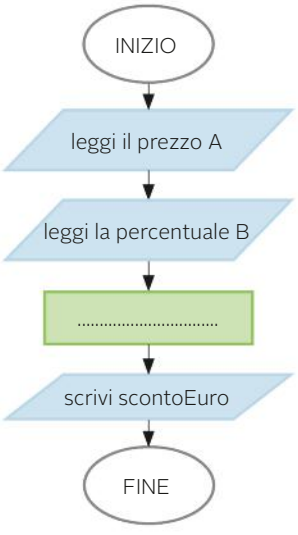
.....

.....



19 Dato un prezzo in euro e data la percentuale di sconto da applicare, vuoi calcolare lo sconto in euro. Per esempio, se il **prezzo** è 100 € e lo **sconto** è del 15%, il programma dovrà restituire come output il valore 15 € per la variabile **scontoEuro**.

Completa il diagramma di flusso dell'algoritmo.



20 Come modificheresti gli ultimi blocchi del precedente diagramma di flusso, se volessi calcolare e mostrare, invece dello sconto, il valore del prezzo scontato?

.....

.....

.....

21 **a.** Qual è la principale differenza tra un ciclo di tipo **do** e un ciclo di tipo **while**?

.....

.....

.....

b. Qual è la principale differenza tra un ciclo di tipo **for** e un ciclo di tipo **while**?

.....

.....

22 Vai a riguardare la tabella di traccia esemplificata nel capitolo per l'algoritmo della **figura 11.14**.

Quale sarebbe il risultato prodotto dall'algoritmo, se il valore di input dato al programma fosse 4?

.....

Quale sarebbe il risultato se il valore di input fosse 5?

.....

Quale sarebbe il risultato se il valore di input fosse 6?

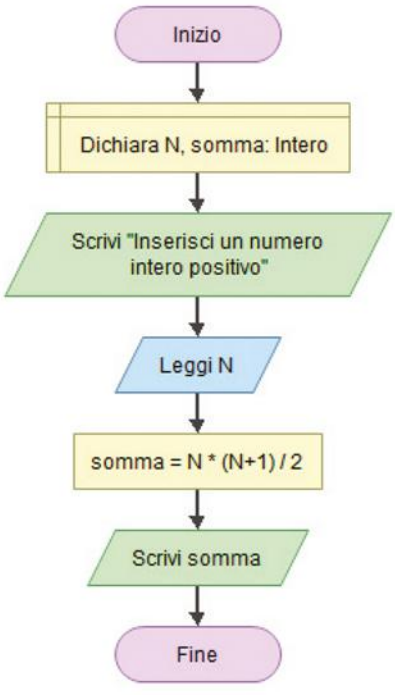
.....

23 A quale algoritmo iterativo equivale l'algoritmo sequenziale di questo diagramma di flusso di Flowgorithm?

.....

.....

.....



24 Costruisci in Flowgorithm un diagramma di flusso per l'algoritmo della **figura 11.14** del capitolo.

Per implementare il ciclo iterativo, scrivi la condizione dentro un blocco di tipo **Mentre**.

Salva il lavoro con il nome **sommainteri_mentre.fprg**.

Poi costruisci anche il diagramma di flusso dell'esercizio precedente e salvalo con il nome **sommainteri_formula.fprg**.

Esegui i due algoritmi e verifica che producano risultati identici.

Ripeti il confronto per almeno tre diversi dati di input.

25 Costruisci in Flowgorithm un diverso flowchart per l'algoritmo della **figura 11.14**: questa volta scrivi la condizione dentro un blocco di tipo **Per**.

Salva il lavoro con il nome **sommainteri_per.fprg**.

Esegui anche questo algoritmo e verifica che produca risultati identici ai due algoritmi dell'esercizio precedente.

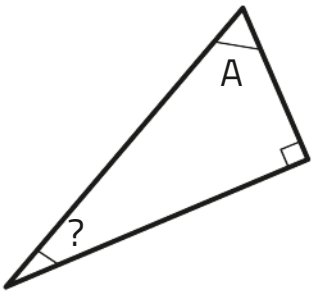
Per quale ragione in questo algoritmo è indifferente usare la struttura iterativa di tipo **Mentre** o quella di tipo **Per**?

.....

.....

.....

26 Dato come input il valore in gradi **A** di uno dei due angoli adiacenti all'ipotenusa in un triangolo rettangolo, vuoi trovare e stampare il valore del terzo angolo (chiamalo **B**).



Scrivi qui sotto una descrizione per punti dell'algoritmo da applicare e disegna su un foglio il corrispondente diagramma di flusso.

.....

.....

.....

.....

.....

27 Disegna in Flowgorithm il diagramma di flusso di un algoritmo che chieda all'utente di digitare come input una cifra binaria e poi analizzi l'input effettivamente inserito: se è corretto, il programma deve mostrare il messaggio «Bene!», altrimenti il messaggio «Input non corretto».

Salva il lavoro con il nome **cifra_binaria.fprg.**

28 Disegna in Flowgorithm il flowchart di un algoritmo che chieda all'utente di inserire un numero intero positivo e produca come output il conto alla rovescia dal numero di input fino a zero.

Salva il lavoro con il nome **countdown.fprg**.

29 Disegna su un foglio il diagramma di flusso di un algoritmo che, dati come input tre numeri qualsiasi, restituisca come output il valore del *più piccolo* tra i tre numeri.

30 Disegna su un foglio il diagramma di flusso di un algoritmo che, dati come input **n** numeri, restituisca come output il loro valore medio.

Supponi che il valore di **n** sia il primo dato di input, seguito poi dagli **n** valori di cui bisogna fare la media.

Nell'algoritmo puoi usare non più di quattro variabili.

31 Disegna su un foglio il diagramma di flusso di un algoritmo che restituisca come output il valore medio di un numero qualsiasi di valori dati come input.

Dopo la lettura di ciascun valore, devi chiedere all'utente del programma di inserirne un altro.

Se il valore inserito è zero, l'algoritmo deve terminare mostrando il valore medio dei dati che l'utente ha inserito fino a quel momento.

32 Fai una ricerca su Internet per scoprire la regola in base a cui certi anni sono *bisestili*, cioè hanno un giorno in più degli altri anni (il 29 febbraio).



Poi scrivi qui sotto un algoritmo che, dato come input un numero intero di 4 cifre che rappresenta un anno, dica se quell'anno è bisestile oppure no.

Supponi di avere a disposizione un operatore matematico % che restituisce il resto della divisione tra due numeri interi (così che, per esempio, 4%2=0 mentre 5%2=1).

.....

.....

.....

.....

.....

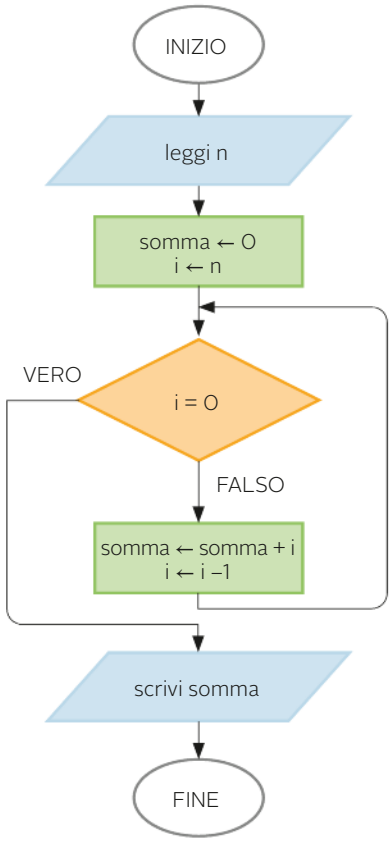
.....

.....

VERIFICA DELLE COMPETENZE

1 LAVORA CON I DATI

Il diagramma di flusso rappresenta un algoritmo (diverso da quello già visto nel capitolo) per la somma degli interi fino a **n**.



Verifica il funzionamento dell'algoritmo, usando la tabella di traccia, nel caso in cui il valore di input per **n** è 5. Per controllare il valore ottenuto, puoi usare il fatto che la somma degli interi fino a **n** vale sempre **n**·(**n**+1)/2. Verifica poi che l'algoritmo funzioni correttamente anche nel caso in cui **n** vale 0.

n	i	i =o	somma

Scrivi tre differenze tra questo algoritmo e quello equivalente visto in precedenza nel capitolo.

1. .....
2. .....
3. .....

2 INFORMATICA E CITTADINANZA

EDUCAZIONE CIVICA

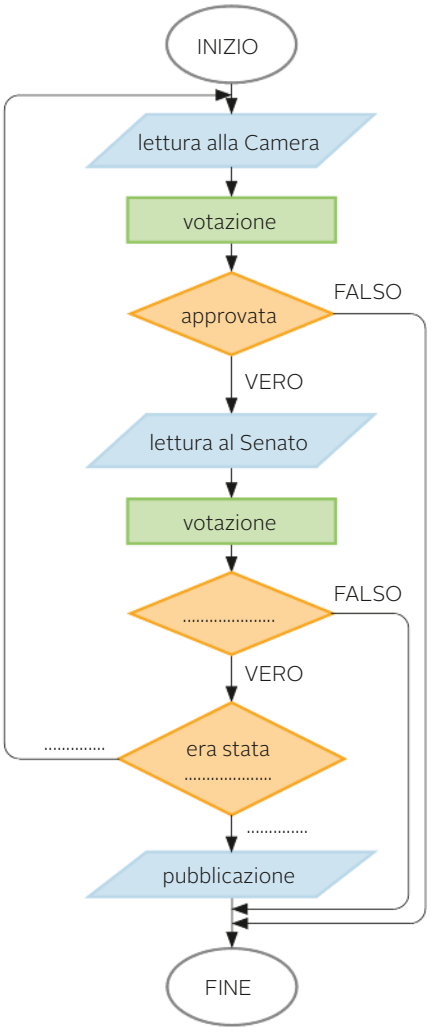
In Italia ogni legge deve essere approvata nel medesimo testo da entrambi i rami del Parlamento, la Camera dei deputati e il Senato della Repubblica.

Se una delle due assemblee vota contro una proposta di legge, cioè non la approva, la proposta è abbandonata e il processo termina. Se il Senato approva una legge modificata rispetto alla versione votata dalla Camera, la legge deve ritornare alla Camera per essere di nuovo approvata con il testo cambiato.

Il processo legislativo perciò è di tipo iterativo, e lo si può rappresentare con un diagramma di flusso.

Completa il diagramma che segue scrivendo nelle opportune posizioni le seguenti parole:

approvata • **modificata** • **VERO** • **FALSO**



In linea di principio il processo descritto dal diagramma di flusso potrebbe non avere mai termine (e quindi, si potrebbe dire, l'algoritmo in astratto è difettoso). Sai dire in quale caso ciò avverrebbe?

.....

.....

.....

CLIL

Learn from the originals

Read it as they wrote it

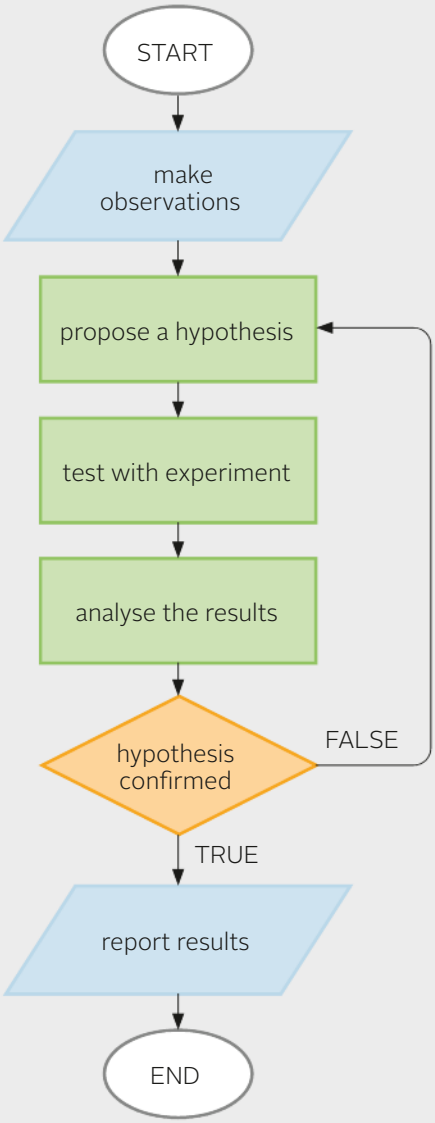
A flowchart for the scientific method

by Anne Helmenstine in www.thoughtco.com/scientific-method-flow-chart-609104

The scientific method is a system of exploring the world around us, asking and answering questions, and making predictions. Scientists use the scientific method because it is objective and based on **evidence**.

A hypothesis is fundamental to the scientific method: it can take the form of an explanation or a prediction.
There are several ways to break down the steps of the scientific method, but it always involves forming a hypothesis, testing the hypothesis, and determining whether or not the hypothesis is correct.
The figure shows the steps of the scientific method in the form of a flowchart.

If a hypothesis is **rejected**, this does not mean the experiment was a **failure**.
In fact, if you proposed a **null hypothesis** (the easiest to test), rejecting the hypothesis may be sufficient to state the results.
Sometimes, if the hypothesis is rejected, you reformulate the hypothesis and then go back to the experimentation stage.
Even when no hypothesis is proven correct, this result may be important and **worth** reporting.



QUESTIONS

1. **Is there a unique way to represent the scientific method with a flowchart?**
2. **What are the input data for all scientific research?**
3. **What is tested in the selection structure of the «scientific algorithm»?**
4. **The procedure in the flowchart could go on forever in an infinite loop. In which case?**

Glossary

evidence	one or more reasons for believing that something is or is not true
to reject	to refuse, to disprove
failure	the opposite of success
null hypothesis	a hypothesis the researcher tries to disprove
worth	important or interesting enough