Modelli e algoritmi

OBIETTIVI

- approfondire il concetto di modello
- acquisire il concetto di algoritmo
- saper costruire semplici algoritmi per la risoluzione dei problemi

1 Dal problema al modello

Gli esercizi di questo paragrafo sono a pag. 28

Ogni giorno dobbiamo affrontare e risolvere problemi che, per giungere a una soluzione, necessitano di un'analisi molto attenta della realtà. A volte si può essere tentati di procedere per via empirica, basandosi sulle proprie esperienze, ma il modo più corretto per affrontare la risoluzione di un problema, come abbiamo già messo in evidenza nel capitolo relativo alle equazioni, è quello di procedere con metodo, cercando di individuare con precisione l'obiettivo che si vuole raggiungere e le relazioni tra gli oggetti che sono coinvolti dal problema.
Facciamo qualche esempio molto banale, ma significativo.

1 Dobbiamo sapere quanto nastro decorato di un certo tipo dobbiamo comperare per bordare una tovaglietta rettangolare di 35cm per 1 metro.

2 Dobbiamo individuare il percorso più conveniente per spostarci da Milano a Napoli.

3 Dobbiamo analizzare i costi per la parziale ristrutturazione di un appartamento; in particolare dobbiamo sapere quanto costa sostituire i vecchi pavimenti in piastrelle con una pavimentazione in parquet.

Per risolvere il problema 1 basta semplicemente calcolare il perimetro di un rettangolo avente i lati lunghi 35cm e 1m e per farlo dobbiamo:

- trasformare prima di tutto le due misure in modo che siano espresse secondo la stessa unità, per esempio il metro: $35\text{cm} = 0{,}35\text{m}$
- calcolare il perimetro $= 2(0{,}35 + 1) = 2{,}70\text{m}$

Dobbiamo quindi comperare come minimo 2,70 metri di nastro.

Risolvere il problema 2 è meno immediato perché la risposta è comunque legata al concetto di *convenienza*; conveniente significa impiegare meno tempo, spendere meno denaro, o viaggiare comodi e arrivare senza essere stanchi? Qualunque sia la risposta, in ogni caso dobbiamo procurarci, a seconda dei casi, un orario ferroviario, l'orario dei voli Milano-Napoli, una cartina stradale, i costi dell'autostrada e quanto carburante dovrebbe servire per fare il viaggio.

Il terzo problema è molto chiaro nella sua formulazione, quello che ci serve è avere una piantina dell'appartamento, con le dimensioni in scala delle pavi-

mentazioni da rifare in modo da poter calcolare la superficie, conoscere il costo della demolizione e smaltimento del vecchio pavimento e il costo al metro quadrato del parquet.

Astrazione e formalizzazione

In questi tre esempi, abbiamo cercato di concentrare la nostra attenzione sulle informazioni essenziali alla risoluzione del problema, e ci siamo costruiti un modello semplificato della situazione reale:

1. un rettangolo per rappresentare la tovaglia e una formula matematica per il calcolo del perimetro
2. un orario ferroviario e dei voli aerei, una cartina stradale
3. una piantina dell'appartamento e le formule di matematica per calcolare le aree.

In ciascun modello compaiono solo le informazioni che servono alla risoluzione del problema e niente altro: le informazioni superflue devono essere eliminate se non si vogliono correre rischi di confusione e di complicazione della situazione.
Per esempio è del tutto inutile sapere se la tovaglia è bianca o di un altro colore, se è di lino o di cotone; non serve sapere se l'appartamento da ristrutturare è in centro città o in periferia.
Il modello deve essere dunque il frutto di un **processo di astrazione** (dal latino *abs-trahere*, tirare fuori) che consiste nell'estrarre dal complesso delle informazioni riguardanti una particolare realtà, solo quelle che servono alla risoluzione del problema, sostituendo contemporaneamente gli oggetti reali con dei simboli più semplici.
Di solito un processo di astrazione porta alla creazione di un insieme di simboli e di concetti astratti che hanno lo scopo di rendere più semplice e schematica una situazione concreta che, per sua natura, è più complessa: la realtà di un appartamento è molto più complicata di una semplice piantina e di una formula per calcolare un'area.
Quando si cerca, attraverso un processo di astrazione, di costruire un modello semplice che descriva la realtà attraverso simboli opportunamente scelti, si dice che si è di fronte ad un processo di **formalizzazione**.

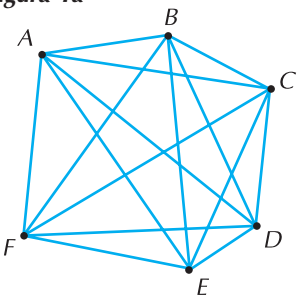
La generalità di un modello

I modelli, proprio perché sono astratti e formalizzati, e quindi non sono legati alla natura degli oggetti reali, sono generali, cioè non risolvono un solo problema specifico, ma tutti i problemi di una certa classe.
Prendiamo per esempio la legge fisica nota come *legge di gravitazione universale* che esprime la forza F con cui due corpi di massa rispettivamente m_1 e m_2 che si trovano a distanza d uno dall'altro si attraggono reciprocamente:

$$F = G \cdot \frac{m_1 \cdot m_2}{d^2}$$

Questa formula consente di calcolare la forza di attrazione tra Terra e Luna, così come quella tra Terra e Sole, oppure tra due galassie o tra due mele poste su un tavolo.
Un altro esempio. Supponiamo di dover organizzare un torneo di scacchi al quale si sono iscritte un certo numero di persone; dopo la prima selezione ne sono rimaste solo 6 e, per determinare il vincitore, ogni concorrente deve incontrare una volta tutti gli altri. Quante partite devono essere programmate?
Un possibile modello del problema è quello raffigurato in ***figura 1a*** dove ogni concorrente è individuato da una lettera; ciascuna connessione indica la par-

Figura 1a

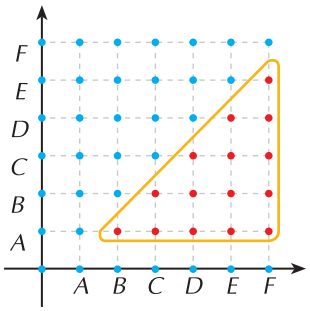


tita che si deve disputare tra i concorrenti; contando il numero di connessioni si ha la risposta al problema.
Un altro possibile modello sfrutta il prodotto cartesiano tra insiemi (***figura 1b***); basta contare il numero di elementi del prodotto, togliere gli elementi che si trovano sulla diagonale (un concorrente non può giocare contro sé stesso) e dividere il risultato per 2 (la partita *A* con *B* è la stessa di *B* con *A*).
Ma questi stessi due modelli potrebbero andare bene per rappresentare altri problemi:

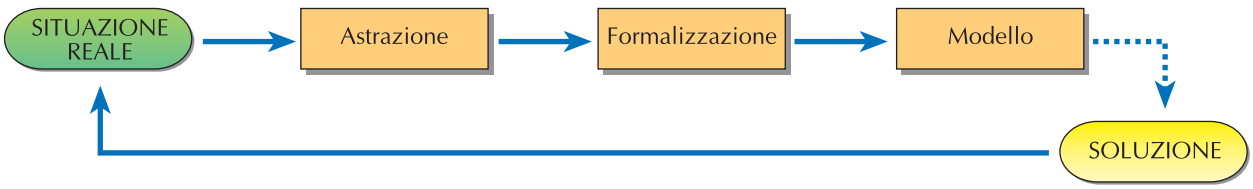
- trovare il numero delle diagonali di un poligono di 6 lati
- trovare quanti sono i segmenti che congiungono a due a due 6 punti distinti
- trovare in quanti modi si possono scambiare una stretta di mano 6 amici.

Tutti questi problemi coincidono con quello più generale di trovare il numero di interazioni a due a due tra 6 elementi di un insieme.

Figura 1b



Servirsi di un modello, e in questo volume lo abbiamo già fatto molte volte anche se in alcuni casi in modo inconsapevole, è sicuramente utile per affrontare la risoluzione di un problema, sia perché ne dà una rappresentazione chiara, sintetica e priva di informazioni superflue, sia perché è un pratico strumento di comunicazione di informazioni.
Quello che abbiamo detto finora si può sintetizzare nel seguente schema:



dove la connessione tra *modello* e *soluzione* è stata lasciata volutamente tratteggiata a significare che l'aver costruito un modello rappresenta sicuramente un passo avanti nella risoluzione di un problema, ma è solo il primo passo, perché bisogna pensare a tutta la serie di operazioni da compiere per arrivare alla soluzione; in altre parole, aver disegnato un rettangolo non significa aver calcolato il suo perimetro o la sua area.
Nei prossimi paragrafi vedremo come completare questo percorso.

2 Il concetto di algoritmo

Gli esercizi di questo paragrafo sono a pag. 28

La formalizzazione di un problema è essenziale per poter trasferire il processo risolutivo ad una macchina che sia poi in grado di risolverlo in modo automatico senza che sia l'uomo a dover intervenire.
A tutti noi sembra semplicissimo ricercare in rete argomenti che ci interessano mediante un motore di ricerca: basta digitare una parola chiave e vediamo comparire una serie di siti che hanno a che vedere con l'argomento che abbiamo digitato; il computer si arrangia da solo a trovarli, senza che l'utente debba intervenire. Questo significa che qualcuno ha scritto una serie di istruzioni per la macchina, che, acquisite certe informazioni dall'utente (le parole chiave), le hanno spiegato come eseguire la ricerca e come proporre i risultati.

*Un **dato** in informatica è un qualsiasi elemento di informazione; è costituito da simboli che devono essere elaborati.*

Il trasferimento della risoluzione di un problema a un computer è indicato comunemente con il termine **implementazione**. Il computer, acquisiti i dati di in-

gresso, o dati di **input**, li elabora e produce i risultati richiesti o dati di **output**.

L'elaborazione dei dati di input per ottenere quelli di output porta a scrivere una procedura codificata che deve essere puntualmente eseguita per rispondere in modo appropriato alle richieste.

La successione ordinata e finita di tutte le operazioni da compiere per raggiungere il risultato richiesto da un problema si dice **algoritmo**.

ESEMPI

1. Spostarsi da una città *A* ad una città *B*.

Obiettivo: determinare il percorso più conveniente per andare da *A* a *B*.

Il problema è solo in apparenza banale perché, per poter dare una risposta, dobbiamo fare un'analisi completa del problema, che comprende l'individuazione dei percorsi stradali che collegano le due località, i mezzi di trasporto disponibili, i tempi necessari per il viaggio, i costi.
Ci accorgiamo così che la domanda che ci siamo posti non è ben formulata perché non specifica rispetto a che cosa dobbiamo calcolare la convenienza. Riformuliamola allora in questo modo.

Obiettivo: determinare il percorso più conveniente dal punto di vista economico per andare da *A* a *B*.

Modalità di risoluzione: trovare i costi di

a. viaggio in aereo;

b. viaggio in treno;

c. viaggio in auto.

Ci servono allora i seguenti dati.

Dati di input:

- costo del viaggio aereo nella tariffa più economica: € 115
- costo del biglietto ferroviario in seconda classe: € 68
- numero di chilometri che separano le due località supponendo di scegliere un percorso autostradale: 585km, costo dei pedaggi autostradali: € 30, costo al chilometro dell'auto che intendiamo usare: € 0,2.

Possiamo adesso determinare i risultati.

Processo risolutivo:

a. costo del viaggio aereo: € 115

b. costo del viaggio in treno: € 68

c. costo del viaggio in auto: € $(30 + 585 \cdot 0{,}2) =$ € 147

Sorge adesso un altro problema: quante persone devono compiere il viaggio?
La domanda non è banale perché un'auto può trasportare 4 o 5 persone e in questo caso i costi del viaggio in auto vengono suddivisi tra le persone che vi partecipano. Dobbiamo allora riformulare la domanda iniziale.

Obiettivo: determinare il percorso più conveniente dal punto di vista economico per andare da *A* a *B* se i viaggiatori sono due oppure tre.

Troviamo i nuovi risultati e riassumiamoli in una tabella:

Mezzo di trasporto	Costo (per due persone in €)	Costo (per tre persone in €)
Aereo	$115 \cdot 2 = 230$	$115 \cdot 3 = 345$
Treno	$68 \cdot 2 = 136$	$68 \cdot 3 = 204$
Auto	147	147

Dopo aver confrontato i risultati ottenuti, possiamo comunicare il risultato della nostra elaborazione.

Dati di output: per due persone: viaggio in treno
per tre persone: viaggio in auto.

2. *Rappresentare la situazione in Matematica delle classi prime di una scuola al termine dello scrutinio del primo quadrimestre.*

Supponiamo per semplicità che ci siano solo tre sezioni di classi prime.

Obiettivo: determinare l'andamento medio di ciascuna classe in Matematica, mettendo anche in evidenza, per ciascuna classe, il numero di studenti insufficienti e il numero di quelli particolarmente preparati (voto maggiore o uguale a 8).

Modalità di risoluzione: reperire le votazioni in Matematica dai prospetti degli scrutini del primo quadrimestre delle classi prime.

Dati di input: i voti in matematica di ciascuno studente.

Sezione A: 7, 6, 6, 5, 4, 6, 8, 7, 4, 8, 6, 6, 7, 9, 5, 6, 7, 8, 8, 7, 6, 6

Sezione B: 5, 5, 8, 7, 6, 6, 6, 8, 7, 9, 4, 5, 4, 3, 7, 6, 6, 8, 7, 5

Sezione C: 6, 6, 6, 7, 8, 7, 6, 5, 5, 6, 7, 4, 8, 8, 9, 7, 8, 6, 5, 8

Processo risolutivo:

- per il calcolo della media: somma dei voti di ciascun studente diviso il numero di studenti

Sezione A: $\dfrac{7+6+6+5+4+6+8+7+4+8+6+6+7+9+5+6+7+8+8+7+6+6}{22} = 6{,}45$

Sezione B: $\dfrac{5+5+8+7+6+6+6+8+7+9+4+5+4+3+7+6+6+8+7+5}{20} = 6{,}1$

Sezione C: $\dfrac{6+6+6+7+8+7+6+5+5+6+7+4+8+8+9+7+8+6+5+8}{20} = 6{,}6$

- conteggio delle insufficienze:
  Sezione A: 4 Sezione B: 7 Sezione C: 4
- conteggio dei voti maggiori o uguali a 8:
  Sezione A: 5 Sezione B: 4 Sezione C: 6

Dati di ouput: i risultati possono essere restituiti in una tabella come quella che segue:

	Sezione A	Sezione B	Sezione C
Voto medio	6,45	6,1	6,6
N. studenti insufficienti	4	7	4
N. studenti eccellenti	5	4	6

3 LA PROGETTAZIONE E LA RAPPRESENTAZIONE DEGLI ALGORITMI

Gli esercizi di questo paragrafo sono a pag. 32

3.1 Il risolutore e l'esecutore

Per progettare un algoritmo efficace per la risoluzione di un problema, si deve anche tener conto delle caratteristiche di chi deve poi materialmente eseguirlo; in altre parole, il **risolutore**, cioè colui che progetta il percorso di risoluzione del problema e costruisce successivamente l'algoritmo, deve conoscere le caratteristiche dell'**esecutore**, cioè di colui che deve materialmente eseguire l'algoritmo.

Risolutore: *chi progetta e costruisce l'algoritmo.*
Esecutore: *che esegue l'algoritmo.*

Supponiamo, per esempio, di dover progettare la risoluzione del seguente problema.

Trovare quante mattonelle quadrate di 33cm di lato servono per ricoprire una stanza rettangolare di 5m per 8m.

Se il risolutore fosse un ragazzo di seconda media, sarebbe sufficiente fornire queste istruzioni:

1. calcolare l'area della stanza in metri quadrati
2. calcolare l'area di una mattonella in metri quadrati
3. dividere l'area della stanza per l'area di una mattonella
4. comunicare il numero di mattonelle che servono.

Se il risolutore fosse una macchina, le istruzioni dovrebbero essere molto più dettagliate:

1. moltiplicare la misura del lato della mattonella per se stessa
2. dividere il risultato per 10000 (trasformazione di cm^2 in m^2)
3. moltiplicare le misure dei due lati della stanza
4. dividere il risultato ottenuto al punto **3.** per il risultato ottenuto al punto **2.**
5. arrotondare all'intero superiore il risultato ottenuto
6. comunicare il risultato del punto 5.

Se un ragazzo sa come calcolare un'area e intuisce che il numero di mattonelle deve essere intero, ad una macchina bisogna spiegare tutto.
Nella scrittura di un algoritmo eseguibile da un computer dobbiamo quindi tenere presenti due fattori importanti:

- le abilità che possiamo attribuire a un computer
- le caratteristiche e le possibilità del linguaggio che intendiamo usare per scrivere gli algoritmi.

Possiamo attribuire a un computer le seguenti abilità:

- eseguire le quattro operazioni fondamentali e l'estrazione di radice quadrata
- svolgere operazioni di confronto
- trasferire dati da una zona all'altra della memoria
- acquisire dati dall'ambiente esterno
- comunicare dati all'ambiente esterno.

Non dobbiamo invece pensare che un computer sia in grado di prendere iniziative; una macchina non può decidere da sola quello che deve fare perché

non è intelligente, è solo un ottimo esecutore, rapido, sicuro, affidabile, ma non autonomo.

3.2 La rappresentazione degli algoritmi e la pseudocodifica

Le informazioni fornite a un computer possono essere dati (per esempio le misure dei lati di un rettangolo) oppure istruzioni che operano sui dati (per esempio eseguire la moltiplicazione tra le misure dei due lati).
Tutti i dati, sia di input che di output, per poter essere riconosciuti dal computer, devono avere un nome che li identifica (nel nome non ci devono essere spazi vuoti) e, per questioni di semplicità e chiarezza, è bene attribuire a un dato un nome che in qualche modo lo identifichi per quello che rappresenta.
Inoltre si deve anche specificare la tipologia di ciascun dato; questo perché un computer memorizza in modo differente numeri di tipi intero e numeri di tipo reale, caratteri alfanumerici. Per esempio, i lati di un rettangolo potrebbero chiamarsi *Lato1* e *Lato2* ed essere dichiarati di tipo reale; il codice fiscale di una persona potrebbe chiamarsi CF ed essere di tipo alfanumerico.

Riassumendo possiamo dire che nella scrittura di un algoritmo dobbiamo specificare:

- i nomi dei dati di input e di output e la loro tipologia
- le istruzioni per trasformare i dati al fine di ottenere i risultati desiderati.

Proviamo allora a costruire un algoritmo eseguibile da un computer per risolvere il precedente problema delle mattonelle.

Un carattere alfanumerico è una cifra da 0 a 9, un carattere alfabetico, un simbolo speciale come +, &, una parentesi e così via. Una serie di caratteri alfanumerici costituisce una ***stringa****. Sono per esempio delle stringhe:*

- *il codice fiscale di una persona*
- *la targa di un'auto*

Algoritmo Mattonelle

Oggetti usati
    Lato1, Lato2, LatoMattonella: numeri reali
    AreaRettangolo, AreaQuadrato: numeri reali
    NumeroMattonelle: numero intero

Processo risolutivo
inizio
    acquisisci il valore di Lato1
    acquisisci il valore di Lato2
    acquisisci il valore di LatoMattonella
    calcola AreaRettangolo = Lato1 × Lato2
    calcola AreaQuadrato = LatoMattonella × LatoMattonella
    calcola NumeroMattonelle = divisione intera di (AreaRettangolo : AreaQuadrato)
    aumenta di 1 NumeroMattonelle
    comunica NumeroMattonelle
fine.

Analizziamo la struttura di questo algoritmo. In esso notiamo tre parti fondamentali:

- la **riga di intestazione** che indica il nome dell'algoritmo
- la **sezione dichiarativa** nella quale si elencano i nomi e la tipologia degli oggetti usati nell'algoritmo
- la **sezione esecutiva** nella quale si elencano le istruzioni che il computer deve eseguire per produrre i dati di output; tali istruzioni sono racchiuse dalle parole *inizio* e *fine*.

Analizziamo più in dettaglio ciascuna di queste parti.

- **Riga di intestazione**

  È opportuno che il titolo sia significativo, vale a dire che deve ricordare l'obiettivo del problema in modo che sia intuibile lo scopo dell'algoritmo; per esempio, il nome adatto ad un algoritmo che calcola il prezzo scontato di una merce può essere Sconto.

- **Sezione dichiarativa**

  In questa sezione sono dichiarati i nomi degli oggetti usati dal problema e la loro tipologia.
  In un algoritmo ci possono essere degli oggetti che non devono cambiare mai il loro valore durante tutta l'elaborazione o da un'elaborazione all'altra, come per esempio la costante π che vale sempre e comunque 3,14 (valore approssimato).
  Altri oggetti possono invece cambiare valore sia durante l'esecuzione dell'algoritmo che da un'esecuzione all'altra; per esempio, lo stesso algoritmo potrebbe essere usato per una sala lunga 30m e larga 25m e per mattonelle di 20cm di lato. Tutti gli oggetti dichiarati come numeri reali possono quindi assumere valore diverso da un'esecuzione dell'algoritmo all'altra.

Un oggetto che assume sempre lo stesso valore durante l'esecuzione di un algoritmo o da un'elaborazione all'altra si dice **costante**.
Un oggetto che può cambiare il suo valore si definisce **variabile**.

In genere, tutti gli oggetti sui quali si compiono delle operazioni sono delle variabili.

- **Sezione esecutiva**

  Questa sezione contiene l'elenco dettagliato delle azioni che l'esecutore deve compiere per raggiungere l'obiettivo del problema; esse vengono impartite mediante ordini del tipo: acquisisci un dato, calcola un prodotto, comunica un risultato, e rappresentano quindi le **istruzioni** per l'esecutore.
  Tutte le istruzioni di questa sezione sono racchiuse tra le parole **inizio** e **fine**.
  In questa sezione si possono inserire anche dei commenti racchiudendoli in una coppia di parentesi graffe; i commenti servono ad annotare osservazioni o spiegazioni sulla procedura seguita.

Vediamo adesso le regole fondamentali per eseguire la **dichiarazione delle costanti e delle variabili**.

Per **dichiarare una costante** si deve:

- scrivere la parola *costanti*
- scrivere il nome seguito dal simbolo di uguaglianza e dal valore della costante.

Per esempio:

costanti
Pigreco = 3,14
VelocitàLuce = 300000

Per **dichiarare una variabile** si deve:

- scrivere la parola *variabili*
- scrivere i nomi delle variabili dello stesso tipo separati da una virgola, mettere il simbolo " : ", indicare il tipo della variabile.
  Per esempio:

variabili
Lato1, Lato2, Area : numeri reali
Indirizzo: stringa
NumeroCivico: numero intero

Le regole fondamentali per **scrivere le istruzioni della sezione esecutiva** sono le seguenti.

- Per indicare l'acquisizione di un dato si usa il comando *leggi* seguito dal nome del dato racchiuso da parentesi tonde.
  Per esempio:

leggi (Lato1)
leggi (Lato2)
leggi (Raggio)

Si può anche scrivere una sola istruzione leggi seguita dall'elenco dei nomi separati da una virgola:

leggi (Lato1, Lato2, Raggio)

- Per indicare la **comunicazione di un dato** si usa il comando *scrivi* seguito dal nome del dato racchiuso tra parentesi tonde; la sintassi di questo comando è analoga a quella del comando leggi.
  Per esempio:

scrivi (Area)
scrivi (Misura)

oppure in una sola istruzione

scrivi (Area, Misura)

- Per attribuire il risultato di un calcolo ad una variabile si usa l'istruzione di **assegnamento** che ha la seguente sintassi:

VariabileRisultato = calcolo

Il risultato di un calcolo deve sempre essere attribuito ad una variabile; per esempio:

$A = B + C$

significa che il valore calcolato di $B + C$ viene assegnato alla variabile di nome A.

Non è possibile invece scrivere

$B + C$

perché in questo caso l'esecutore non saprebbe dove mettere il risultato dell'espressione $B + C$.

Riscriviamo allora l'algoritmo *Mattonelle* usando queste convenzioni.

```
Algoritmo Mattonelle                                   riga di intestazione

Variabili                                              sezione dichiarativa
    Lato1, Lato2, LatoMattonella: numeri reali
    Arearettangolo, AreaQuadrato: numeri reali
    NumeroMattonelle: numero intero

inizio                                                 sezione esecutiva
    leggi (Lato1)
    leggi (Lato2)
    leggi (LatoMattonella)
    AreaRettangolo = Lato1 × Lato2
    AreaQuadrato = LatoMattonella × LatoMattonella
    NumeroMattonelle = AreaRettangolo div AreaQuadrato
{l'operatore DIV esegue la divisione intera tra due numeri}   riga di commento
    NumeroMattonelle = NumeroMattonelle + 1
    scrivi (NumeroMattonelle)
fine.
```

Nomi come AreaRettangolo, AREARETTANGOLO e arearettangolo rappresentano la stessa variabile.

Notiamo che nello scrivere questo algoritmo si sono usate le regole dell'**indentazione**, convenendo di scrivere più a destra alcune righe in modo da mettere in evidenza gruppi di istruzioni o l'appartenenza a diverse sezioni.
Abbiamo poi usato alcune lettere in carattere maiuscolo per le variabili e le costanti, ma questa è solo una convenzione di scrittura utile alla chiarezza di lettura dell'algoritmo; non esiste infatti differenza tra i caratteri maiuscoli e minuscoli.
È poi utile inserire delle **righe di commento** che possono far comprendere meglio lo sviluppo dell'algoritmo; i commenti vengono scritti all'interno di una coppia di parentesi graffe e vengono ignorate dal computer in fase di esecuzione.

Questa modalità di scrittura di un algoritmo prende il nome di **pseudocodifica**. Si tratta di una via di mezzo tra il linguaggio comune e il linguaggio di programmazione; un algoritmo scritto in pseudocodifica non è infatti direttamente comprensibile da un computer e necessita di una successiva traduzione nel linguaggio scelto (Visual Basic, Pascal, Delphi o altro).
L'uso della pseudocodifica è però importante perché rappresenta la struttura dell'algoritmo; conviene quindi sempre conservare il materiale ad essa relativo per eventuali traduzioni in linguaggi di programmazione diversi.

Vediamo qualche altro esempio.

ESEMPI

1. Calcolare la media tra tre numeri interi assegnati.

Dati di input: i tre numeri interi che indichiamo con A, B, C.

Dati di output: la media che indichiamo con M e che deve essere un numero reale.

Processo risolutivo: la formula per il calcolo della media $M = \frac{A + B + C}{3}$

Nella costruzione dell'algoritmo dobbiamo quindi prevedere:

- l'acquisizione dei dati
- il calcolo della media
- la comunicazione dei risultati.

Pseudocodifica dell'algoritmo.

```
Algoritmo Media
variabili
   A, B, C : numeri interi
   M : numero reale
inizio
   leggi (A, B, C)
   M = (A + B + C) / 3
   scrivi (M)
fine.
```

2. Calcolare il prezzo scontato di una merce.

Dati di input: il costo della merce, che indichiamo con Costo, il tasso di sconto in forma percentuale, che indichiamo con Tasso, entrambi numeri reali.

Dati di output: il prezzo scontato, che indichiamo con Prezzo e che deve essere un numero reale.

Processo risolutivo: la formula per il calcolo del prezzo scontato $\text{Prezzo} = \text{Costo} - \frac{\text{Costo} \cdot \text{Tasso}}{100}$

Nella costruzione dell'algoritmo dobbiamo quindi prevedere:

- l'acquisizione dei dati
- il calcolo del prezzo scontato
- la comunicazione del risultato.

Pseudocodifica dell'algoritmo.

```
Algoritmo Sconto
variabili
   Costo, Tasso, Prezzo: numeri reali
inizio
   leggi (Costo, Tasso)
   Prezzo = Costo − (Costo · Tasso) / 100
   scrivi (Prezzo)
fine.
```

APPROFONDIMENTO

Il diagramma di flusso

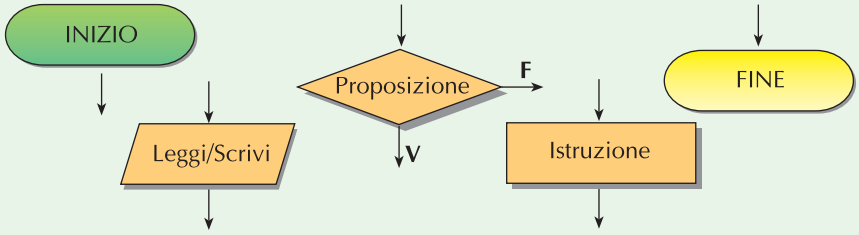
L'algoritmo di risoluzione di un problema può anche essere rappresentato mediante schemi grafici detti **diagrammi di flusso** o **flow-chart** che corrispondono sostanzialmente alla sezione esecutiva della pseudocodifica. Un flow-chart usa i seguenti simboli grafici (***figura 2*** di pagina seguente):

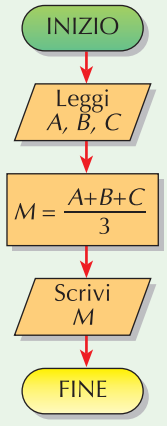
- un ovale per indicare l'inizio e la fine dell'algoritmo

- un parallelogramma per indicare la lettura e scrittura dei dati
- un rettangolo per le istruzioni di elaborazione dei dati
- un rombo per indicare la scelta fra due o più alternative possibili.

Le istruzioni corrispondenti ai simboli si scrivono al loro interno.

Figura 2





Per esempio, il flow-chart a lato descrive l'algoritmo della media fra tre numeri già rappresentato nell'esempio 1 con la pseudocodifica.

Come si nota, il flow-chart rappresenta solo la parte esecutiva, l'algoritmo non ha un nome e non vengono neppure dichiarate in modo esplicito le variabili, anche se esse si deducono dalle istruzioni scritte all'interno dei simboli.

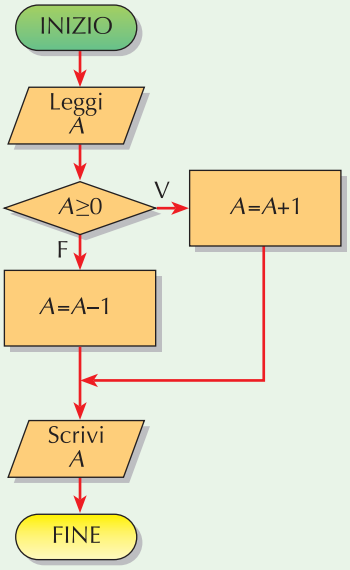
Consideriamo adesso il seguente problema: acquisito un numero intero, calcolare il suo successivo se è positivo o nullo, il suo precedente se è negativo.

Dati di input: il numero intero che indichiamo con A.

Dati di output: successivo o il precedente di A.

Processo risolutivo: se A è maggiore o uguale a zero calcolare $A + 1$, altrimenti calcolare $A - 1$.

Il flow-chart corrispondente è a lato.



Oggi i flow-chart vengono usati quasi esclusivamente per rappresentare a scopo esplicativo parti limitate di programma e questo è anche l'uso che ne faremo nei prossimi paragrafi.

4 Le strutture di controllo

Gli esercizi di questo paragrafo sono a pag. 33

Un algoritmo ben costruito è più facile da scrivere, da comprendere, da modificare; per questo le sue istruzioni sono organizzate in tre schemi fondamentali:

- la **sequenza**, vale a dire istruzioni che devono essere eseguite una dopo l'altra nell'ordine in cui sono scritte
- la **selezione**, vale a dire istruzioni da eseguire in alternativa a seconda del verificarsi o meno di alcune condizioni

- l'**iterazione**, vale a dire istruzioni che devono essere ripetute un certo numero di volte oppure fino a quando non si verifica una determinata condizione.

Un enunciato importante, che è noto come **Teorema di Böhm-Jacopini**, afferma che:

> qualsiasi algoritmo può essere scritto utilizzando solo le tre strutture di base di sequenza, selezione, iterazione.

Questi tre modelli organizzativi fondamentali si chiamano **strutture di controllo**, perché con esse si può controllare il percorso risolutivo del problema per ottenere i risultati desiderati.
Analizziamo dunque queste strutture una alla volta.

4.1 La sequenza

È la più semplice tra le strutture di controllo e indica l'ordine in cui devono essere eseguite le istruzioni. Negli esempi 1 e 2 precedenti le istruzioni racchiuse tra le parole *inizio* e *fine* erano entrambe sequenze di istruzioni.
In pseudocodifica la sequenza viene quindi rappresentata in questo modo, usando l'indentazione per raggruppare le istruzioni:

```
inizio
   istruzione 1
   istruzione 2
   .............
   .............
   istruzione n
fine.
```

4.2 La selezione binaria

La seconda struttura logica fondamentale è la **selezione binaria** che consente di fare una scelta tra due possibili alternative. Per esempio:

- **se** l'indicatore della benzina della mia auto lampeggia, **allora** mi fermo a un distributore per il rifornimento, **altrimenti** proseguo
- **se** nel compito di inglese prendo un voto maggiore o uguale a 6, **allora** sono sufficiente, **altrimenti** sono insufficiente.

In pseudocodifica un'istruzione di selezione assume questa forma:

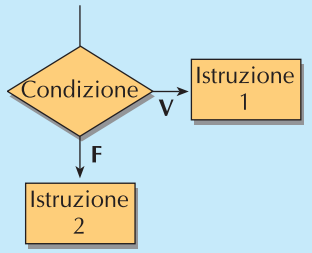
```
se proposizione condizionale
   allora istruzione 1
   altrimenti istruzione 2
fine se
```

In alcuni linguaggi di programmazione le parole *fine se* che servono a chiudere l'istruzione non sono necessarie; nella traduzione in codice si dovrà tener conto della sintassi specifica di questa istruzione.

L'istruzione viene eseguita in questo modo:

- si analizza la *proposizione condizionale* che può essere vera (V) o falsa (F)
- se essa è V, viene eseguita l'*istruzione 1* che si trova dopo la parola *allora*
- se essa è F, viene eseguita l'*istruzione 2* che si trova dopo la parola *altrimenti*.

In un flow-chart il simbolo di selezione binaria è il rombo:



Per esempio:

- l'istruzione:

  se N $>$ 3 allora A $=$ N $+$ 6 altrimenti A $=$ N $-$ 1

  viene eseguita in questo modo a seconda del valore di N:

  - per N $=$ 5 la proposizione N $>$ 3 è V, viene quindi eseguita l'istruzione A $=$ N $+$ 6 la variabile A assume valore 11
  - per N $=$ $-$ 2 la proposizione N $>$ 3 è F, viene quindi eseguita l'istruzione A $=$ N -1 e la variabile A assume valore -3.

- se vogliamo progettare un algoritmo che, letti due numeri, li scrive in ordine crescente, dobbiamo usare l'istruzione:

  se A $>$ B allora scrivi (B, A) altrimenti scrivi (A, B).

Le istruzioni che seguono la parola *allora* o la parola *altrimenti* possono anche essere più di una; è bene in questi casi usare l'indentazione per mettere in evidenza i due gruppi:

se proposizione condizionale
  allora
    istruzione 1
    istruzione 2
    .....
    .....
    istruzione n
  altrimenti
    istruzione 1'
    istruzione 2'
    .....
    .....
    istruzione n'
fine se

Occorre specificare poi che la seconda parte dell'istruzione, che comprende la parola *altrimenti* e il relativo gruppo di istruzioni, non è obbligatoria e può essere omessa quando non ci sono istruzioni da eseguire se la proposizione condizionale risulta falsa.

Vediamo alcuni esempi di algoritmi nei quali si deve usare la struttura di selezione.

ESEMPI

1. Dati due numeri non nulli, stabilire se sono concordi o discordi.

Dati di input: i due numeri, che indichiamo con A e B.

Dati di output: la stringa *'I numeri sono concordi'* oppure *'I numeri sono discordi'*.

Processo risolutivo: due numeri sono concordi se hanno lo stesso segno, sono discordi se hanno segno opposto; si può allora calcolare il loro prodotto e, se questo è positivo, concludere che i due numeri sono concordi, se è negativo concludere che sono discordi.

Variabili utilizzate: oltre ad A e B si deve usare una terza variabile reale P alla quale assegnare il valore del prodotto.

Algoritmo Concordi-Discordi

```
variabili
   A, B, P: numeri reali
inizio
   leggi (A,B)
   P = A · B
   se P > 0
      allora scrivi ('I numeri sono concordi')
      altrimenti scrivi ('I numeri sono discordi')
   fine se
fine.
```

2. Calcolare il valore assoluto di un numero.

Dati di input: un numero reale che indichiamo con A.

Dati di output: il valore assoluto del numero.

Processo risolutivo: il valore assoluto di un numero è il numero stesso se questo è positivo, il suo opposto se questo è negativo; di conseguenza, se A è positivo o nullo basta scrivere A, se è negativo si deve scrivere −A.

Variabili utilizzate: è sufficiente la sola variabile A.

Algoritmo Valore assoluto

```
variabili
   A: numero reale
inizio
   leggi (A)
   se A > 0
      allora scrivi (A)
      altrimenti scrivi (−A)
   fine se
fine.
```

APPROFONDIMENTO

La selezione multipla

In molte situazioni è necessario operare delle scelte orientandosi tra più di due possibilità; per esempio nella scelta del percorso universitario o del tipo di investimento per i nostri risparmi.

Consideriamo un insegnante che deve valutare un test di 20 domande attribuendo un punteggio variabile da 3 a 8 in base al numero di risposte esatte.
La situazione potrebbe essere questa:

N. risposte esatte (N)	Punteggio (P)
da 0 a 3	3
da 4 a 6	4
da 7 a 10	5
da 11 a 13	6
da 14 a 17	7
da 18 a 20	8

Se volessimo usare la selezione binaria dovremmo scrivere una serie di istruzioni **se ... allora ... altrimenti** annidate una nell'altra per valutare tutte le possibilità:

se $N \leq 3$ allora $P = 3$ altrimenti
  se $4 \leq N \leq 6$ allora $P = 4$ altrimenti
    se $7 \leq N \leq 10$ allora $P = 5$ altrimenti ..... e così via

Per rendere più agevole la scelta tra diverse possibilità si può usare la struttura di **selezione multipla**, che consente di selezionare gruppi distinti di istruzioni nel caso in cui la variabile in base alla quale avviene la scelta, e che si chiama per questo **selettore**, assuma particolari valori.

In pseudocodifica l'istruzione di selezione multipla ha la seguente sintassi:

nel caso selettore **assume valore**
  espressione 1 : gruppo istruzioni 1
  espressione 2 : gruppo istruzioni 2
    .....
    .....
  espressione n : gruppo istruzioni n
  altrimenti
    istruzione
  fine selezione

Quando la variabile *selettore* assume il valore indicato da *espressione 1* viene eseguito il gruppo *istruzioni 1*, quando assume il valore indicato da *espressione 2* viene eseguito il gruppo *istruzioni 2* e così via; se *selettore* assume un valore non specificato nell'elenco viene eseguita l'istruzione (o il gruppo di istruzioni) dopo la parola *altrimenti*; se questa parte non compare e il selettore assume un valore diverso da quelli in elenco, l'esecutore passa all'istruzione successiva.

Relativamente all'esempio precedente, la variabile N è il selettore e l'istruzione di selezione multipla diventa:

nel caso N assume valore
  da 0 a 3 : $P = 3$
  da 4 a 6 : $P = 4$
  da 7 a 10 : $P = 5$
  da 11 a 13 : $P = 6$
  da 14 a 17 : $P = 7$
  da 18 a 20 : $P = 8$
fine selezione

4.3 L'iterazione

Nella risoluzione di un problema può capitare che alcune operazioni debbano essere eseguite ripetutamente; per esempio:

- un impiegato allo sportello delle Poste che esegue versamenti fa sempre le stesse cose fino a che ci sono clienti in fila:
  - prende il bollettino dal cliente
  - indica il numero di conto, l'importo del bollettino e il nome del versante sul computer e la causale del versamento
  - ritira il denaro e restituisce la ricevuta e l'eventuale resto.
- un atleta che si sta allenando facendo flessioni, esegue per 50 volte le stesse azioni
- un bibliotecario che deve registrare 20 nuovi volumi esegue per 20 volte le stesse operazioni sul computer:
  - attribuisce il codice
  - indica l'autore
  - indica il titolo
  - indica la casa editrice
  - indica l'anno di edizione
  - segna il costo.

Tutte le volte che un determinato gruppo di operazioni devono essere ripetute sempre nello stesso ordine si parla di **iterazione** (o anche *ripetizione* o *ciclo* o *loop*).

L'iterazione enumerativa

La forma più semplice di ripetizione è l'**iterazione enumerativa** che consente di ripetere uno stesso gruppo di istruzioni un numero fissato di volte; la sua sintassi in pseudocodifica è la seguente:

per variabile = valore iniziale **fino a** valore finale
    istruzione 1
    istruzione 2
    .....
    istruzione n
fine ciclo

dove variabile è una sorta di contatore che assume inizialmente il valore specificato da valore iniziale.

Supponiamo, per esempio di chiamare *I* la variabile contatore, che il *valore iniziale* sia 1 e che il *valore finale* sia 3; il ciclo funziona così:

Per I = 1 fino a 3
    istruzione
fine ciclo

- vengono eseguite una prima volta le istruzioni elencate (*I* è uguale a 1)
- al termine del primo ciclo (l'esecutore incontra le parole *fine ciclo*) il contatore viene incrementato automaticamente di una unità (*I* è adesso uguale a 2)
- il valore assunto da *I* viene confrontato con *valore finale* (che è 3) e poiché si verifica che *I* è minore di 3 si esegue una seconda volta il ciclo
- al termine della seconda esecuzione il contatore viene ancora incrementato di una unità (*I* è uguale a 3)
- si esegue di nuovo il confronto di *I* con valore finale e poiché questa volta *I* è uguale a 3, il ciclo viene eseguito un'ultima volta.

Al termine della terza ripetizione si esce dal ciclo e l'esecutore passa all'istruzione successiva.
Vediamo allora alcuni esempi di utilizzo di questa forma iterativa.

ESEMPI

1. Calcolare la potenza *n*-esima di un numero reale, con *n* numero intero positivo.

Dati di input: la base della potenza che indichiamo con A, l'esponente che indichiamo con N.

Dati di output: il valore calcolato della potenza che indichiamo con P.

Processo risolutivo: il calcolo di A^N viene eseguito calcolando il prodotto di N fattori uguali ad A; si può quindi ipotizzare di porre uguale a 1 il valore iniziale di P (elemento neutro del prodotto) e di eseguire il prodotto $P \cdot A$ per N volte, vale a dire di usare un ciclo enumerativo.

Variabili utilizzate: oltre ad A e a P si deve utilizzare una variabile contatore I che assume valore iniziale 1 e valore finale N.

Algoritmo Potenza

```
Variabili
   A, P : numeri reali
   N, I : numeri interi

inizio
   leggi (A,N)
   P = 1
   per I = 1 fino a N
      P = P · A
   fine ciclo
   scrivi ('Valore della potenza: ',P)
fine.
```

Osserviamo l'istruzione di scrittura: in essa abbiamo inserito un messaggio di spiegazione del risultato racchiudendolo tra apici.
Proviamo ad eseguire l'algoritmo come farebbe il computer per renderci conto meglio di quello che accade supponendo di calcolare 2^4 (tra l'altro questo tipo di controllo serve anche ad evidenziare eventuali errori di progettazione):

Fase	A	N	P	I
situazione iniziale (I < N)	2	4	1	1
esecuzione primo ciclo (I < N)	2	4	$1 \cdot 2 = 2$	2
esecuzione secondo ciclo (I < N)	2	4	$2 \cdot 2 = 4$	3
esecuzione terzo ciclo (I = N)	2	4	$4 \cdot 2 = 8$	4
esecuzione quarto e ultimo ciclo	2	4	$8 \cdot 2 = 16$	4

A questo punto, poiché I è uguale al valore finale indicato nel ciclo (cioè N) viene eseguita la prima istruzione dopo l'iterazione, cioè scrivi (P); viene quindi comunicato il valore della potenza.

2. Calcolare la somma di *n* numeri introdotti da tastiera.

Dati di input: il valore di *n*, che indichiamo con N, e i numeri da sommare.

Dati di output: la somma degli *n* numeri, che indichiamo con S.

Processo risolutivo: non è opportuno servirsi di *n* variabili, una per ogni numero dato; conviene quindi usare una sola variabile, che chiamiamo A, alla quale assegnare i numeri da sommare in S, uno alla volta. La procedura di calcolo dovrà quindi essere la seguente:

- inizializzare la variabile S assegnando ad essa il valore zero, elemento neutro della somma
- leggere il primo numero e assegnarlo alla variabile A
- sommare A in S
- ripetere le operazioni di lettura e di somma fino ad esaurimento dei numeri.

Variabili utilizzate: oltre ad A, S e N, dovremo utilizzare la variabile contatore I con valore iniziale 1 e valore finale N.

Algoritmo Somma

```
Variabili
   N, I : numeri interi
   A, S : numeri reali

inizio
   leggi (N)
   S = 0
   per I = 1 fino a N
      leggi (A)
      S = S + A
   fine ciclo
   scrivi ('Valore della somma: ', S)
fine.
```

Simuliamo l'esecuzione dell'algoritmo supponendo che sia $N = 3$ e che i numeri A siano, nell'ordine 10, -15, 36:

Fase	N	A	S	I
situazione iniziale (I < N)	3		0	1
esecuzione del primo ciclo (I < N)	3	10	$0 + 10 = 10$	2
esecuzione del secondo ciclo (I < N)	3	-15	$10 - 15 = -5$	3
esecuzione del terzo e ultimo ciclo (I = N)	3	36	$-5 + 36 = 31$	3

Essendo I uguale a N il controllo delle operazioni passa alla prima istruzione successiva al ciclo che è scrivi (S); viene quindi comunicato il valore di S che è 31.

L'iterazione per falso

Un altro modo di esprimere una forma iterativa è quello di indicare fino a quando un gruppo di istruzioni deve essere ripetuto; la sua sintassi in pseudo-codifica è la seguente:

```
ripeti
   istruzione 1
   istruzione 2
   ..............
   istruzione n
finché proposizione condizionale
```

Il ciclo funziona così:

- si eseguono una prima volta tutte le istruzioni del ciclo
- si analizza la *proposizione condizionale* e:
  - se la proposizione è falsa si ripete il ciclo
  - se la proposizione è vera si esce dal ciclo.

Iterazione per falso *significa che il ciclo viene ripetuto fintanto che la proposizione condizionale risulta falsa.*

A differenza dell'iterazione enumerativa, in quella per falso non si sa a priori quante volte viene ripetuto il ciclo, anche se possiamo dire che viene eseguito almeno una volta, perché la valutazione del valore di verità della proposizione avviene in coda.
Un ciclo di questo tipo può essere usato per il controllo dei dati in ingresso; se, per esempio, riprendiamo l'algoritmo *Concordi-Discordi* (esempio 1 del paragrafo 4.2.), possiamo modificare la parte iniziale in modo da essere sicuri che i due numeri introdotti non siano nulli (osserva la parte in grassetto e ricorda che le parti racchiuse da parentesi graffe sono commenti e non vengono considerati dall'esecutore).

Algoritmo Concordi-Discordi

```
variabili
   A, B, P: numeri reali

Inizio
   ripeti
      leggi (A)
   finché A < > 0                    {il simbolo < > sta per diverso}
   ripeti
      leggi (B)
   finché B < > 0
   P = A · B
   se P > 0
      allora scrivi ('I numeri sono concordi')
      altrimenti scrivi ('I numeri sono discordi')
   fine se
fine.
```

Vediamo un altro esempio di utilizzo di questa forma iterativa.

ESEMPI

1. Dati due numeri naturali, calcolare il quoziente e il resto della loro divisione intera

Dati di input: i due numeri da dividere che chiamiamo A e B, con $B \neq 0$.

Dati di output: il quoziente intero, che indichiamo con Q, il resto, che indichiamo con R.

Processo risolutivo: ricordiamo che il computer sa eseguire le operazioni fondamentali e che se chiediamo di calcolare A : B viene restituito il valore decimale, eventualmente approssimato, corrispondente alla divisione; per esempio, 23 : 5 dà 4,6 mentre noi vogliamo ottenere 4 come quoziente e 3 come resto. Possiamo allora procedere in questo modo:

- sottraiamo 5 da 23 e continuiamo a sottrarre 5 dalla differenza ottenuta, tenendo conto del numero di

volte che si esegue questa operazione, fino a che troviamo un numero minore di 5:

$\underbrace{23 - 5 = 18}_{1^a \text{ volta}}$ $\underbrace{18 - 5 = 13}_{2^a \text{ volta}}$ $\underbrace{13 - 5 = 8}_{3^a \text{ volta}}$ $\underbrace{8 - 5 = 3}_{4^a \text{ volta}}$

- il numero di volte che viene eseguita la sottrazione corrisponde al quoziente intero, l'ultima differenza ottenuta corrisponde al resto: $Q = 4$, $R = 3$

Variabili utilizzate: A, B rappresentano il dividendo e il divisore (stiamo eseguendo A : B), Q è la variabile che conta il numero di sottrazioni eseguite, R la differenza trovata ogni volta; la variabile Q all'inizio della procedura deve essere uguale a 0, mentre la variabile R deve essere uguale ad A.
Teniamo poi presente che la lettura della variabile B deve essere ripetuta fino a che il valore immesso sia diverso da zero.

Algoritmo Divisione-Resto

```
Variabili
   A, B, Q, R : numeri interi

inizio
   leggi (A)
   ripeti
      leggi (B)
   finché B <> 0
   Q = 0                          {inizializzazione di Q}
   R = A                          {inizializzazione di R}
   ripeti
      R = R − B                   {esecuzione della differenza}
      Q = Q + 1                   {incremento del contatore}
   finché R > B
   scrivi ('Valore del quoziente: ', Q)
   scrivi (Valore del resto: ', R)
fine.
```

In realtà esistono due operatori supportati dalla maggior parte dei linguaggi di programmazione che consentono di trovare il quoziente intero ed il resto di una divisione tra numeri interi:

- la funzione **div** restituisce il quoziente intero
- la funzione **mod** restituisce il resto.

L'algoritmo precedente diventerebbe in questo caso:

Algoritmo Divisione-Resto

```
Variabili
   A, B, Q, R : numeri interi

inizio
   leggi (A,B)
   Q = A div B
   R = A mod B
   scrivi ('Valore del quoziente: ', Q)
   scrivi ('Valore del resto: ', R)
fine.
```

L'iterazione per vero

Questa terza forma iterativa è molto simile all'iterazione per falso; anche in questo caso, infatti, il gruppo di istruzioni del ciclo viene eseguito fino alla verifica di una particolare condizione. La sua sintassi in pseudocodifica è la seguente:

mentre proposizione condizionale **esegui**
  istruzione 1
  istruzione 2
  .....
  .....
  istruzione n
fine ciclo

Il ciclo funziona così:

- si analizza la proposizione condizionale e:
  - se la proposizione è vera si esegue il ciclo
  - se la proposizione è falsa si salta il gruppo di istruzioni e si esce dal ciclo.

Iterazione per vero *significa che il ciclo viene ripetuto mentre la proposizione condizionale si mantiene vera.*

Osserviamo che, a differenza dell'iterazione per falso che viene eseguita almeno una volta, quella per vero potrebbe non venire mai eseguita nel caso in cui la proposizione condizionale fosse subito falsa.
Vediamo un esempio.

ESEMPI

1. Progettare un algoritmo che calcoli l'importo della spesa al supermercato.

Dati di input: i prezzi di ogni prodotto, che indichiamo con *P*.

Dati di output: l'importo della spesa, che indichiamo con *S*.

Processo risolutivo: la cassiera di un supermercato fa scorrere l'oggetto acquistato sul lettore di codici a barre per immettere il prezzo (operazione corrispondente alla lettura di un dato); il prezzo letto deve essere sommato ai precedenti e l'operazione termina quando non ci sono più oggetti acquistati.

Possiamo simulare la condizione di fine spesa immettendo come ultimo prezzo il valore zero; il ciclo di lettura e somma continua quindi fino a che viene letto il valore zero.

Variabili utilizzate: P, S.

Algoritmo Spesa

```
Variabili
   P, S : numeri reali

inizio
   S = 0                                  {inizializzazione di S}
   leggi (P)                              {lettura del primo prezzo}
   mentre P < > 0 esegui
      S = S + P
      leggi (P)                           {lettura dei prezzi successivi}
   fine ciclo
   scrivi ('Importo della spesa: ',S)
fine.
```

Riepilogo

Per concludere, riassumiamo le caratteristiche fondamentali delle tre forme iterative.

- L'**iterazione enumerativa** si usa quando si sa a priori quante volte il ciclo di istruzioni deve essere eseguito; la variabile contatore è gestita automaticamente dall'esecutore e si incrementa di una unità ad ogni ripetizione del ciclo.
- Nell'**iterazione per falso** le istruzioni del ciclo vengono eseguite ripetutamente fino a che la proposizione condizionale si mantiene falsa, si esce dal loop quando la proposizione diventa vera; inoltre, poiché la condizione di uscita è in coda, il gruppo di istruzioni viene eseguito almeno una volta.
- Nell'**iterazione per vero** le istruzioni del ciclo vengono eseguite ripetutamente fino a che la proposizione condizionale si mantiene vera, si esce dal loop quando la proposizione diventa falsa; inoltre, poiché la condizione di uscita è in testa, il gruppo di istruzioni può anche non venire mai eseguito se la proposizione è subito falsa.

Le ultime due forme si usano prevalentemente quando non si conosce il numero *n* di iterazioni; esse possono tuttavia essere usate anche quando *n* è noto, ma la variabile contatore deve essere gestita dall'algoritmo.

APPROFONDIMENTO

Le condizioni composte

Può capitare che in una selezione binaria o in una delle proposizioni condizionali nell'ambito di un ciclo, la condizione che si deve analizzare per decidere quale azione deve essere compiuta sia una **condizione composta**, cioè una proposizione che è la negazione di un'altra oppure che è formata da due o più condizioni semplici unite tra loro dalle particelle "e" oppure "o". Per esempio:

- se A **non** è minore di 10
- se $A > 0$ **e** $A < 3$
- se $A < 2$ **o** $A > 4$

Queste particelle prendono il nome di **connettivi** ed hanno questo significato:

- **negazione:** se una proposizione è preceduta da "*non*" cambia il suo valore di verità, cioè se è Vera diventa Falsa e viceversa

  "3 è un numero dispari" è Vera, "3 **non è** un numero dispari" è Falsa

- **congiunzione:** se due proposizioni sono unite dal connettivo "*e*", la condizione è vera solo se sono vere entrambe le proposizioni che la formano

  "3 è un numero dispari" (V) e "4 è un numero pari" (V) è Vera

  "7 è un numero primo" (V) e "10 è divisibile per 3" (F) è Falsa

- **disgiunzione:** se due proposizioni sono unite dal connettivo "*o*", la condizione è vera se almeno una delle due proposizioni che la formano è vera

  "12 è divisibile per 4" (V) o "4 è un numero primo" (F) è Vera

  "7 è minore di 5" (F) o "−2 è maggiore di 1" (F) è Falsa.

Nel linguaggio informatico gli operatori corrispondenti sono:

- per la negazione: **not**
- per la congiunzione: **and**
- per la disgiunzione: **or**

Con queste notazioni, le tre precedenti condizioni si scrivono in questo modo e hanno il significato indicato a lato:

- se **not** $(A < 10)$ — significa che la proposizione è vera solo se A assume valori maggiori o uguali a 10
- se $A > 0$ **and** $A < 3$ — significa che la proposizione è vera solo se A assume valori che sono contemporaneamente maggiori di 0 e minori di 3, cioè valori compresi tra 0 e 3
- se $A < 2$ **or** $A > 4$ — significa che la proposizione è vera sia che A assuma valori minori di 2, sia che assuma valori maggiori di 4.

5 FUNZIONI CALCOLABILI

Gli esercizi di questo paragrafo sono a pag. 40

Le funzioni che man mano abbiamo imparato a conoscere non ci hanno mai dato problemi di calcolabilità; data per esempio la funzione $y = 3x - 1$, sappiamo che ad ogni valore reale di x resta associato un solo valore reale di y e della coppia (x, y) si dice che soddisfa la relazione.
Altre funzioni che apparentemente sembrano "meno matematiche" sono ugualmente calcolabili; per esempio la funzione che ad ogni proposizione associa il proprio valore di verità è una funzione in generale calcolabile; se associamo il valore Vero al numero 1 e il valore Falso al numero 0 possiamo dire ad esempio che:

- $f(14 > 3) = 1$ — perché "$14 > 3$" è una proposizione Vera
- $f(10 < -4) = 0$ — perché "$10 < -4$" è una proposizione Falsa

Ma se consideriamo per esempio la proposizione

x : *questa frase è falsa*

di $f(x)$ non possiamo affermare che è uguale a 1, ma nemmeno che è uguale a 0, perché non possiamo dire né che sia vera, né che sia falsa.
In questo caso la funzione $f(x)$ è una funzione non calcolabile.
Precisiamo meglio che cosa intendiamo con il termine *calcolabile*.

Diciamo che una funzione $f(x)$ è **calcolabile** se, per ogni valore della variabile indipendente x, esiste un algoritmo mediante il quale, con un numero finito di passi, è possibile calcolare $f(x)$.

Tutte le funzioni di cui ci siamo occupati finora sono funzioni calcolabili in base a questa definizione.
Osserva gli esempi che seguono.

ESEMPI

1. Costruiamo l'algoritmo che, ad ogni valore intero di x variabile in un intervallo precisato, associa il valore $f(x)$ dato dall'equazione $f(x) = x^2 - 3x + 4$.

Dati di input: l'estremo inferiore INF e l'estremo superiore SUP dell'intervallo in cui può variare x
il valore di x che indichiamo con la lettera A

Dati di output: il valore di $f(x)$ che indichiamo con F.

Ecco l'algoritmo in linguaggio di pseudocodifica:

```
Algoritmo Funzione
Variabili
    INF, SUP, A: numeri interi
    F: numero reale
inizio
    leggi (INF,SUP)
    per I = INF fino a SUP
        F = A × A
        F = F - 3 × A
        F = F + 4
    fine ciclo
    scrivi (F)
fine.
```

2. Una successione di numeri interi si ottiene considerando due numeri assegnati e calcolando i successivi come somma dei due precedenti; per esempio, se i primi due numeri sono 3 e 4, la successione è

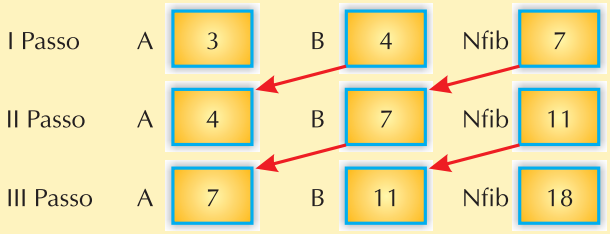
3 4 $3 + 4 = \mathbf{7}$ $4 + 7 = \mathbf{11}$ $7 + 11 = \mathbf{18}$ $18 + 11 = \mathbf{29}$ ...

Successioni numeriche di questo tipo si dicono successioni di Fibonacci.
Costruiamo un algoritmo che generi i numeri di una successione di Fibonacci, partendo da due numeri assegnati, fino a che si trova un elemento che sia maggiore di 200.

Dati di input: i due numeri iniziali, indichiamoli con A e B.

Dati di output: la successione di Fibonacci.

Processo risolutivo: osserviamo innanzi tutto che, poiché ogni numero si ottiene sommando i due precedenti, è necessario tenere sempre traccia degli ultimi due numeri della successione. Posto quindi il primo numero in A ed il secondo in B, il terzo numero, che memorizziamo in una variabile che chiamiamo Nfib, si trova calcolando A + B.



A questo punto il numero in A non serve più e conviene quindi trasferire B in A e Nfib in B in modo da ripetere l'operazione A + B per ottenere il numero successivo.
L'operazione di trasferimento e di somma deve essere ripetuta fino a che Nfib diventa maggiore di 200 e quindi, per risolvere il problema, dovremo usare una delle forme iterative; poiché non sappiamo quante volte si deve eseguire il ciclo, non possiamo usare l'iterazione enumerativa e scegliamo quindi l'iterazione per falso.

Variabili utilizzate: A, B, *Nfib.*

Algoritmo Fibonacci

```
Variabili
   A, B, Nfib : numeri interi

inizio
   leggi (A, B)
   scrivi (A, B)                    {scrittura dei primi due termini della successione}
   ripeti
      Nfib = A + B
      scrivi (Nfib)
      A = B                         {trasferimento di B in A}
      B = Nfib                      {trasferimento di Nfib in B}
   finché Nfib > 200
fine.
```

Concetti chiave e regole

Modelli e algoritmi

Un modello è una rappresentazione schematica e semplificata della realtà che mette in evidenza solo ciò che è importante ai fini della risoluzione di un problema.
Il processo risolutivo prevede la costruzione di un **algoritmo**, nel quale vengono indicate, in modo dettagliato e puntuale, tutte le operazioni che devono essere eseguite per raggiungere l'obiettivo.
Perché un algoritmo sia efficace, è necessario conoscere le abilità dell'esecutore e il linguaggio con cui poter comunicare con esso. Se l'esecutore è un computer, nella stesura dell'algoritmo occorre indicare:

- quali sono i dati di input e di output e qual è la loro tipologia
- quali sono le istruzioni da eseguire sui dati di input per ottenere quelli di output.

La struttura di un algoritmo

Un algoritmo in linguaggio di pseudocodifica è composto da tre parti:

- la **riga di intestazione** con il nome dell'algoritmo
- le **sezione di dichiarazione** nella quale si dichiarano le costanti e le variabili usate e la loro tipologia
  la **sezione esecutiva** con l'elenco delle istruzioni racchiuse dalle parole *inizio* e *fine*.

Le istruzioni fondamentali per scrivere un algoritmo sono:

- l'istruzione di lettura per l'acquisizione del valore di un dato:
  leggi (dato)
- l'istruzione di scrittura per la comunicazione del valore di un dato:
  scrivi (dato)
- l'istruzione di assegnamento per assegnare un valore o un'espressione a una variabile:
  A = "espressione"

Le strutture di controllo

Qualunque algoritmo può essere costruito usando le seguenti strutture di controllo:

- la **sequenza** che indica le istruzioni da eseguire nell'ordine in cui sono indicate
- la **selezione** che permette di scegliere tra due alternative possibili a seconda che si verifichi o meno una certa condizione
- l'**iterazione** che permette di eseguire più volte un determinato gruppo di istruzioni.