ALGORITMI - INDICE

ALGORITMI

1 - COS'E' UN "ALGORITMO"

a) Compra una confezione di filtri da camomilla
b) riempi d'acqua una pentola
c) metti la pentola sul fornello
d) accendi il fornello
e) attendi finché l'acqua bolle
f) spegni il fornello
g) metti un filtro in una tazza
h) riempi la tazza con l'acqua della pentola
i) schiaccia leggermente col cucchiaio il filtro e mescola finché l'acqua ha assunto un colore giallo uniforme
j) togli il filtro dalla tazza
k) aggiungi un cucchiaino di zucchero e assaggia finché il liquido è dolce al punto giusto
l) hai terminato

Quello che abbiamo scritto è un **algoritmo**:
una *sequenza di istruzioni*, finalizzata a risolvere il *problema* della *preparazione di una camomilla*.

a) Leggi il valore di un numero naturale n
b) se n<2 allora
  scrivi: "Il numero deve essere maggiore o uguale a 2: ridammelo";
  leggi il valore di n
c) poni k=1
d) ripeti:
  se n è divisibile per k allora poni massimodivisoretrovato=k;
  incrementa di un'unità il valore di k
finché k=n
e) se massimodivisoretrovato=1 allora scrivi "Il numero dato è primo"
  altrimenti scrivi "Il numero dato non è primo"
f) hai terminato

Quello che abbiamo scritto è un altro **algoritmo**:
una *sequenza di istruzioni*, finalizzata a risolvere il *problema* di *stabilire se un dato intero è primo*.

COS'È, DUNQUE, IN GENERALE, UN ALGORITMO?

E' UNA SEQUENZA DI ISTRUZIONI, DI "PASSI",
IL CUI SCOPO È LA RISOLUZIONE DI UN PROBLEMA.

Le ISTRUZIONI di un algoritmo devono essere

- **PRIVE DI AMBIGUITÀ nel loro CONTENUTO:**
  ogni istruzione deve essere interpretabile dall'esecutore in modo univoco;
- **PRIVE DI AMBIGUITÀ nel loro ORDINE:**
  per ciascuna istruzione, deve essere chiaro qual è l'istruzione da eseguire dopo;
- **CONCRETAMENTE ESEGUIBILI;**
- **COMPRENSIBILI DA PARTE DELL'ESECUTORE, sia questo un essere umano o una macchina:**
  se un'istruzione non possedesse questo requisito,
  occorrerebbe spezzarla in un gruppo di istruzioni più semplici;
- **IN NUMERO FINITO, ed ESEGUIBILI IN UN TEMPO FINITO.**

E il problema risolto deve essere di carattere GENERALE:
ad esempio, non sarebbe un algoritmo una sequenza di istruzioni che fosse capace esclusivamente di stabilire se è o non è primo un numero particolare, che so … il 41.

2 - ALGORITMI E PROGRAMMI

Abbiamo detto che le istruzioni di un algoritmo devono essere concretamente eseguibili e comprensibili da parte dell'esecutore. Ora, ai giorni nostri, l'esecutore per eccellenza di algoritmi, anche molto complessi, è il *computer*, che è precisissimo e rapidissimo nell'eseguire, senza stancarsi e senza protestare, sequenze di istruzioni che gli vengono assegnate nell'ordine corretto da un "programmatore" umano, il quale tenga conto delle limitate capacità che il computer possiede.
Esso non è in grado di pensare, o di prendere autonomamente qualsivoglia decisione: dipende in tutto e per tutto dall'uomo che, attraverso un "programma", gli ordina, istante per istante, COSA fare.

Essenzialmente, un computer sa soltanto:

- **ASSOCIARE**
  A UNA LOCAZIONE DI MEMORIA
  UN NOME
  che la identifichi, come potrebbe essere *n*, o *num*, o *massimodivisoretrovato*, ecc.

Ad esempio:

Nella parte preliminare di un programma scritto nel linguaggio di programmazione **PASCAL**, *dichiarazioni* come
VAR num: integer oppure VAR cognome: string[25]
ordinano al computer di riservare locazioni, nella RAM, destinate a contenere un numero intero o, rispettivamente, una "stringa" (sequenza di caratteri) con non più di 25 caratteri, associando a queste "scatolette" i nomi *num*; *cognome*

- **LEGGERE** un'informazione
  che venga battuta sulla tastiera
  (= ricevere un *input* dalla *tastiera*),
  memorizzando questo input
  in una locazione nella RAM
  (oppure, interpretare opportunamente
  come comandi i movimenti
  e i "clic" o "doppio clic" fatti col *mouse*)

Nel linguaggio di programmazione PASCAL,
l'istruzione **READ**(num)
ordina al computer,
quando l'utente avrà
digitato sulla tastiera un numero e premuto il tasto *Invio*,
di memorizzare quel numero nella locazione
della memoria centrale (RAM),
alla quale è stato associato il nome *num*

- **SCRIVERE** SUL MONITOR,
  o MANDARE IN STAMPA,
  o SALVARE su di una memoria di massa,
  un'informazione
  o un'insieme di informazioni

In linguaggio PASCAL, l'istruzione **WRITE**(num)
ordina al computer di scrivere sul monitor
il valore che in quel momento ha la variabile *num*,
ossia il contenuto che c'è in quel momento
nella locazione di memoria, nella "scatoletta",
alla quale è stato associato il nome *num*

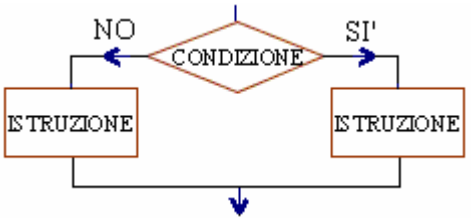
- **ESEGUIRE CALCOLI**
  E CONFRONTI FRA NUMERI

In linguaggio PASCAL,
la **moltiplicazione** viene indicata con un asterisco *,
la **divisione** con /,
la **divisione intera** con DIV e il suo **resto** con MOD,
"**maggiore o uguale**", "**minore o uguale**" si scrivono <=, >=,
"**diverso da**" si scrive <>

- **MODIFICARE IL CONTENUTO**
  DI UNA LOCAZIONE DI MEMORIA

In linguaggio PASCAL, l'istruzione di "**ASSEGNAZIONE**"
c **:=** a+b (occhio: il simbolo grafico non è =, è **:=**)
ordina al computer di eseguire la somma dei contenuti
delle locazioni di memoria associate ai nomi *a*, *b*
e di depositare il risultato nella locazione di memoria *c*
(assegnare il valore ottenuto alla variabile *c*)

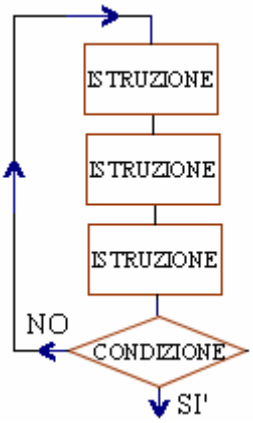
- controllare
  SE è verificata una certa condizione;
  IN CASO AFFERMATIVO
  eseguire una data istruzione
  o blocco di istruzioni,
  ALTRIMENTI
  eseguire un'altra istruzione o blocco



In linguaggio PASCAL, l'istruzione
(o meglio, l' "istruzione complessa" o "struttura")
IF a<b **THEN** max := b **ELSE** max := a
ordina al computer di controllare se a<b
(cioè, se il contenuto della casella *a* nella RAM
ha un valore numerico minore del contenuto della casella *b*);
in caso affermativo,
il computer dovrà eseguire l'istruzione max := b
(cioè, dovrà assegnare alla variabile *max* il valore *b*
ossia dovrà porre nella "scatoletta" *max* in RAM
lo stesso valore che c'è nella "scatoletta" *b*),
in caso negativo dovrà eseguire l'istruzione max := a

- **RIPETERE**
  un'istruzione,
  o un gruppo
  di istruzioni,
  FINCHÉ
  sarà verificata
  una certa condizione

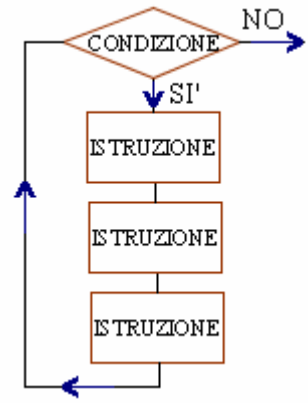
  (CICLO A **CONTROLLO FINALE**)



oppure

FINTANTOCHE'
è verificata
una certa condizione,
RIPETERE
un'istruzione, o un
gruppo di istruzioni

(CICLO A **CONTROLLO INIZIALE**)



In linguaggio PASCAL, un "pezzo" di programma potrebbe essere il seguente:

```
readln(a);
k:=0;
REPEAT
        k:=k+1;
        mult:=a*k;
        writeln(mult);
UNTIL k = 5
```

Questo programma prevede innanzitutto la *lettura* del numero intero *a*
(readln è una *variante* di read, e ordina al computer di *andare a capo*,
sul monitor, dopo che l'utente ha inserito il valore di *a*)
e l' "*inizializzazione*" della variabile *k* al valore 0.
Dopodiché, l'istruzione (o meglio, l' "istruzione complessa" o "struttura")

REPEAT … UNTIL …

ordina al computer di
RIPETERE il blocco delle tre istruzioni

I) poni nella scatoletta *k* il valore che c'era prima, aumentato di 1
(incrementa di 1 il valore della variabile *k*:
" **:=** " non indica uguaglianza, indica **assegnazione**!)

II) poni nella scatoletta *mult* il valore a*k (l'asterisco sta per *moltiplicazione*)

III) scrivi sul monitor il contenuto che ha, in quel momento, la scatoletta *mult*
(writeln è una variante di write, e ordina al computer
di andare a capo dopo che ha scritto sul monitor)

FINCHE' sarà verificata la condizione k = 5;
quando questa condizione si verificherà, il computer dovrà
uscire dal ciclo ed eseguire le istruzioni successive del programma.

L'esito sarà quello di mandare in output, sul monitor, i primi 5 multipli di *a*.

Ad esempio, se il valore della variabile *a* è 9,
il contenuto delle tre "scatolette" *a*, *k* e *mult*, nella RAM,
si evolverà, per effetto del programma, nel modo seguente:

	k:=0	k:=k+1 mult:=a*k	k:=k+1 mult:=a*k	k:=k+1 mult:=a*k	k:=k+1 mult:=a*k	k:=k+1 mult:=a*k
a	9	9	9	9	9	9
k	0	1	2	3	4	5
mult		9	18	27	36	45

Sempre in linguaggio PASCAL,
la ripetizione ("iterazione") di un'istruzione o di un blocco di istruzioni
può anche essere comandata da una WHILE … DO …,
come nel pezzo di programma che segue (dove *n* è un intero):

```
readln(n);
WHILE n>=0 DO
      begin
          writeln(n);
          n:=n – 1;
      end;
```

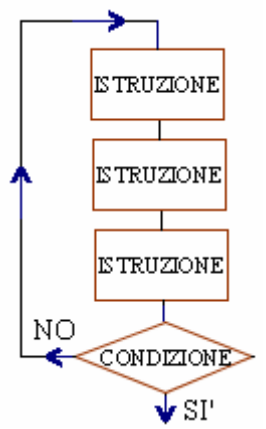
Dopo la lettura, da tastiera, del valore di *n*,
l'istruzione (o meglio, l' "istruzione complessa" o "struttura")

WHILE … DO …

ordina al computer di:
scrivere sul monitor il valore di *n* (e andare a capo);
diminuire di 1 il valore della variabile *n*;
e CONTINUARE A ESEGUIRE (DO) questa coppia di istruzioni
FINTANTOCHE' (WHILE) *n* si mantiene ≥ 0 (>= in PASCAL sta per $\geq$);
quando questa condizione non sarà più verificata, uscire dal ciclo.

L'effetto sarà quello di mandare in output,
sul monitor, un "conto alla rovescia" da *n* fino a 0.

3 - DIAGRAMMI DI FLUSSO; PROGRAMMAZIONE STRUTTURATA



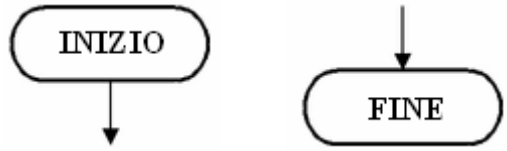
Lo schema qui a sinistra,
che abbiamo utilizzato per descrivere il "funzionamento"
di una REPEAT … UNTIL … del linguaggio Pascal,
è un esempio di
DIAGRAMMA DI FLUSSO.

Per "diagramma di flusso"
(o "diagramma a blocchi"; in lingua Inglese: "flow chart")
si intende dunque
la raffigurazione di un algoritmo
attraverso blocchi, collegati da linee orientate.

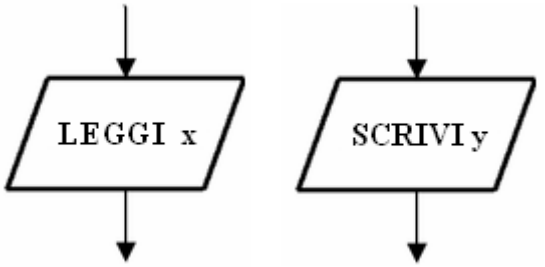
Ciascun blocco contiene una istruzione, oppure una condizione.

Per la **forma dei "blocchi"** si sono affermate certe **convenzioni**.
Precisamente:

blocco **iniziale** e **finale:**
ovale



blocco di **I/O**
(**Input/Output,**
Lettura/Scrittura):
parallelogrammo



blocco di **elaborazione:**
rettangolo

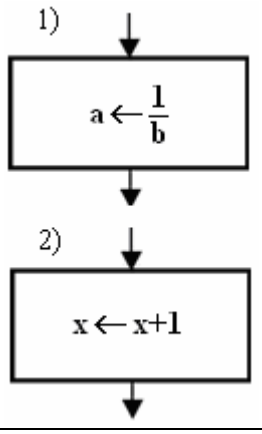
Molto importante
la cosiddetta **assegnazione**:
noi indicheremo
l'assegnazione
col simbolo ←
(vedi NOTA qui a destra)

Esempi:

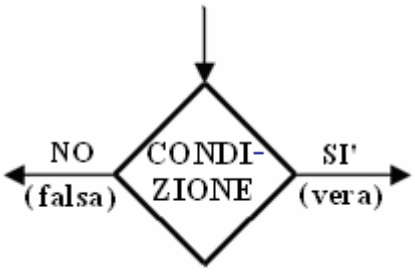
1) assegna alla variabile *a*
il valore uguale al reciproco
del valore della variabile *b*
(poni nella "scatoletta" *a*
il reciproco del contenuto
della "scatoletta" *b*)

2) assegna alla variabile *x*
quello che era
il suo valore precedente,
aumentato di 1
(incrementa di 1
il valore della variabile *x*)

NOTA - Poi, nei vari
linguaggi di programmazione,
quella freccia ←
diventerà un simbolo
specifico per *quel* linguaggio:
ad esempio, in PASCAL è :=

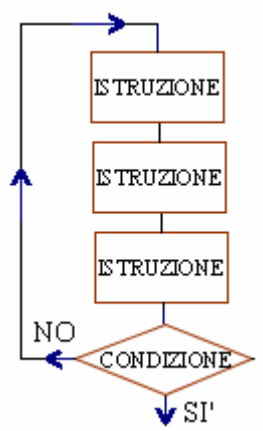


blocco di **controllo**
o di **selezione:**
rombo



E' opportuno organizzare il diagramma di flusso per la risoluzione di un dato problema in modo che, se c'è una "freccia che ritorna indietro", ossia: se c'è una *ripetizione*, una ITERAZIONE di istruzioni, questa iterazione sia governata da una delle due strutture seguenti:

CICLO A **CONTROLLO FINALE**



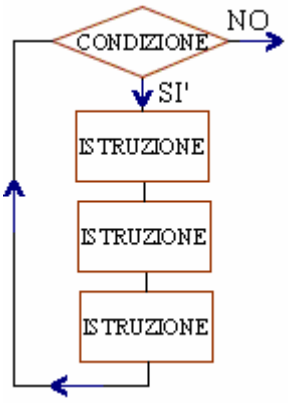
Per evitare cicli infiniti,
fra le "istruzioni"
ce ne dev'essere anche una
che modifica una variabile
presente nella "condizione".

Questa struttura
non va bene
in quei casi nei quali
il blocco di istruzioni
potrebbe
non dover essere eseguito
nemmeno una volta.

In linguaggio PASCAL: REPEAT … UNTIL …
(e si ESCE dal ciclo quando la condizione è VERA).

In linguaggio C: DO … WHILE …
(occhio, però:
qui si ESCE dal ciclo quando la condizione è FALSA)

CICLO A **CONTROLLO INIZIALE**



Per evitare cicli infiniti,
fra le "istruzioni"
ce ne dev'essere anche una
che modifica una variabile
presente nella "condizione".

Questa struttura
si adatta anche
a quei casi nei quali
il blocco di istruzioni
potrebbe
non dover essere eseguito
nemmeno una volta.

In linguaggio PASCAL: WHILE … DO …
(e si ESCE dal ciclo quando la condizione è FALSA).

In linguaggio C: WHILE …
(e si ESCE dal ciclo quando la condizione è FALSA)

Conviene invece evitare l'inserimento nel diagramma di flusso di frecce che facciano "salti incondizionati" da un punto all'altro (istruzioni del tipo "go to").

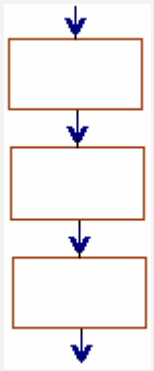
Tale uso disordinato di "salti" (chiamato "**spaghetti programming**") è infatti considerato *controproducente*,

- sia ai fini del controllo della correttezza dell'algoritmo,
- sia per una eventuale successiva traduzione del diagramma in un programma per computer.

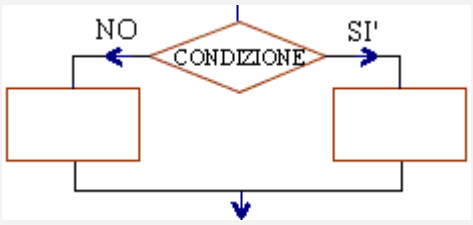
D'altra parte un famoso enunciato, il **teorema di Bohm-Jacopini** (1966), afferma che

qualunque algoritmo si può realizzare con le sole tre strutture di

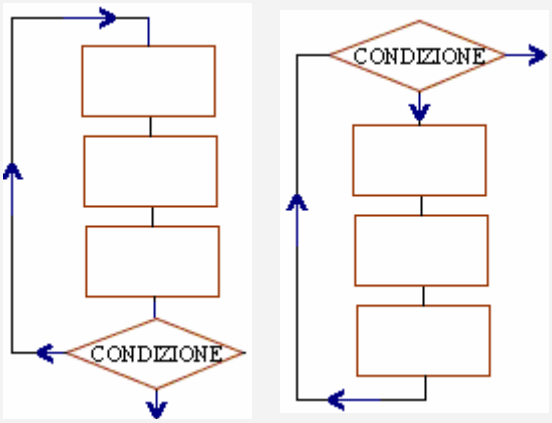
a) SEQUENZA



b) SELEZIONE



c) ITERAZIONE



(nei rettangoli possiamo trovare singole istruzioni oppure altre strutture degli stessi 3 tipi)

E una "buona" programmazione,
che eviti i "salti incondizionati"
e sia invece basata sul teorema di Bohm-Jacopini
ovvero sulle tre strutture
di *sequenza*, *selezione* e *iterazione*
(ciascuna con una sola uscita,
con possibilità di "nidificarle" una dentro l'altra
ma NON di disporle in modo che si "accavallino"),
è chiamata "**programmazione STRUTTURATA**".

La programmazione strutturata
rende più ordinata
la stesura di un programma,
e ne rende più semplici
la lettura e il controllo di correttezza,
nonché
l'eventuale completamento
o utilizzo col ruolo di "sottoprogramma"
(metodi di lavoro *top-down* o *bottom-up*).

4 - ESEMPI; LO PSEUDOCODICE (O "LINGUAGGIO DI PROGETTO")

Qui di seguito daremo alcuni esempi di algoritmi costruiti secondo i principi della "programmazione strutturata".

Accanto a ciascun diagramma di flusso scriveremo anche la corrispondente versione in
pseudocodice (o **pseudolinguaggio**, o **linguaggio di progetto**):
si tratta di una specie di **"linguaggio di programmazione semplificato", non standard**
(noi ne diamo infatti una *nostra* versione, non necessariamente coincidente con altre)
a partire dal quale è poi facile tradurre l'algoritmo in un vero e proprio linguaggio comprensibile dal computer, come ad esempio il linguaggio PASCAL o il linguaggio C o un qualsiasi altro.

… Anche se poi, traducendo lo *pseudocodice* in vero e proprio linguaggio, occorre tenere conto

- delle rigide regole sintattiche di quel linguaggio
- delle risorse proprie del linguaggio adottato (tanto per fare un esempio,
  la RIPETI … FINCHE' … , con la quale si ha uscita dal ciclo quando la condizione è verificata,
  va realizzata, in linguaggio C, attraverso le parole chiave DO … WHILE …
  ma tenendo conto che questa struttura del linguaggio C prevede si esca dal ciclo
  quando la condizione diventa FALSA, e *non* quando diventa *vera*!
  (… Ovviamente, basterà sostituire alla condizione in gioco la sua contraria per sistemare le cose …)
- dell'eventuale presenza, in quel linguaggio, di comode varianti. Tanto per fare due esempi:
  a) nel linguaggio PASCAL accanto alle due strutture iterative con controllo finale e con controllo iniziale, abbiamo pure una struttura iterativa "enumerativa" FOR … DO … che ordina al computer di ripetere l'istruzione, o il blocco di istruzioni, per un numero prefissato di volte
  b) sempre in PASCAL, abbiamo anche una struttura di "selezione multipla" (la CASE … OF …)

- ESEMPIO 1

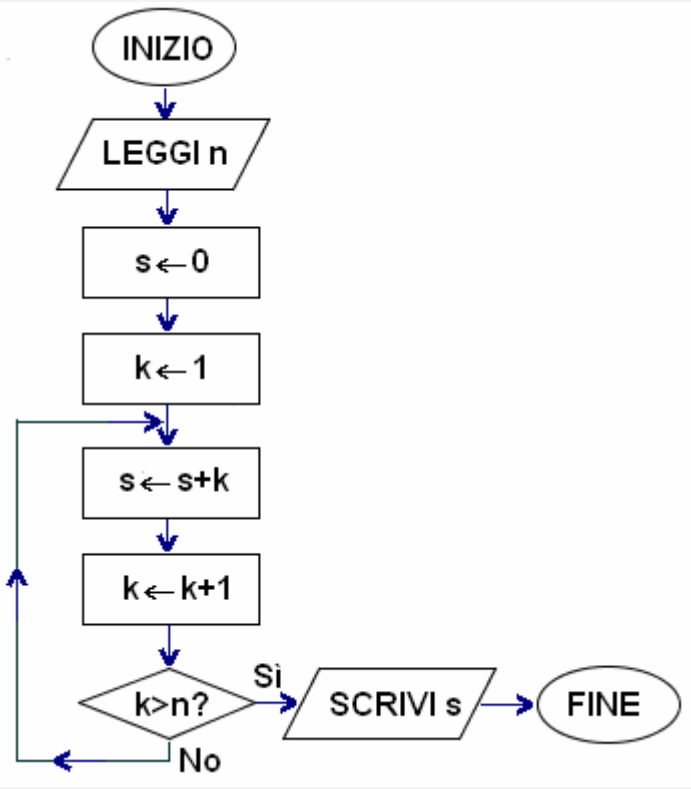
  Il seguente algoritmo serve per
  sommare i primi n interi positivi, ossia eseguire la somma $1+2+3+...+n$, con n letto in ingresso (facciamo finta di non sapere che per questo calcolo c'è l'apposita "formula di Gauss").

  Abbiamo utilizzato la "**freccia a sinistra**" $\leftarrow$ per indicare l'**istruzione di assegnazione**:
  ad esempio, $\mathbf{s \leftarrow 0}$ significa "assegna alla variabile s il valore 0" (metti nella "scatoletta" s il valore 0);
  $\mathbf{k \leftarrow k+1}$ significa "assegna alla variabile k il valore che essa aveva precedentemente, aumentato di 1" (incrementa di 1 il valore della variabile k).

  In linguaggio Pascal, l'assegnazione viene indicata con **:=** *e in linguaggio C con* **=** *senza i "puntini"*

  Osserviamo che la variabile s fa da "**accumulatore**": essa ha il ruolo di "somma parziale",
  e via via, a forza di "accumulare" contributi, si porta a diventare la somma di tutti quanti gli addendi.

DIAGRAMMA DI FLUSSO



LINGUAGGIO DI PROGETTO

INIZIO

LEGGI n

s ← 0

k ← 1

RIPETI
{ s ← s+k

k ← k + 1 }

FINCHE' k>n
SCRIVI s
FINE

La **coppia di GRAFFE** ci serve per evidenziare che quelle istruzioni "fanno blocco", "vanno insieme".

Anche nel linguaggio C le graffe vengono utilizzate con questa funzione.

Invece in linguaggio PASCAL al posto delle graffe si impiegano (se necessario) due indicatori di "inizio blocco" e "fine blocco": BEGIN e END.

- ESEMPIO 2

L'algoritmo che segue legge un intero >2 in ingresso e stabilisce se è o non è un numero primo.

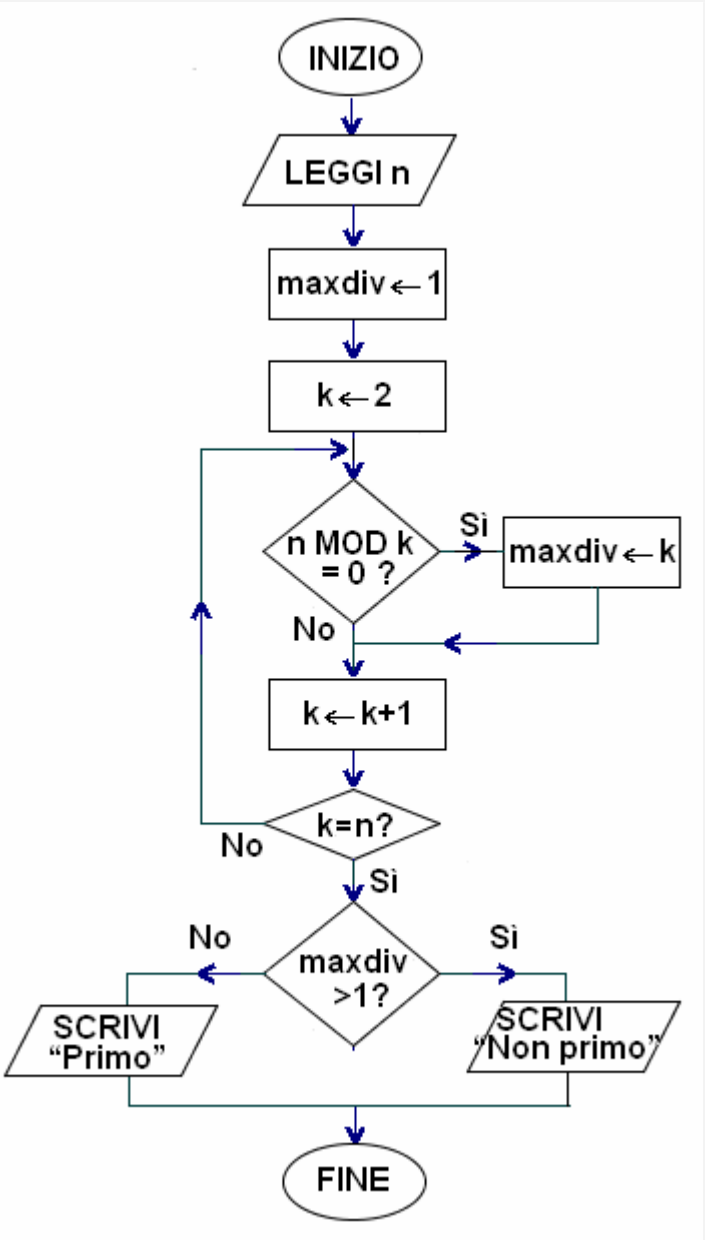
Ricordiamo che
l'operatore MOD dà il resto della divisione intera (mentre DIV ne dà il quoziente intero);
ad esempio, 47 MOD 5 = 2 perché il resto della divisione intera 47:5 è 2 (mentre 47 DIV 5 = 9).
Se a, b sono due interi, a è divisibile per b se e solo se a MOD b = 0.

La variabile maxdiv immagazzina il "massimo fra i divisori trovati fino a quel momento"
e diventerà alla fine il massimo fra i divisori di n, inferiori ad n.
Se, al termine dell'elaborazione, nella "scatoletta" maxdiv ci sarà un numero >1,
vorrà dire che è stato trovato un divisore diverso sia dall'unità che dal numero n stesso,
per cui se ne concluderà che n non è primo.

Il diagramma di flusso contiene due strutture di selezione:
- la prima offre una sola alternativa (SE … ALLORA … senza "altrimenti"),
- la seconda ne offre classicamente due (SE … ALLORA … ALTRIMENTI …)

Controlla tu stesso che con questo algoritmo,
se il numero n dato in ingresso *non* fosse >2,
si rimarrebbe intrappolati in un *ciclo senza fine*.

DIAGRAMMA DI FLUSSO



LINGUAGGIO DI PROGETTO

INIZIO

LEGGI n

maxdiv ← 1

k ← 2

RIPETI

{ SE n MOD k = 0 ALLORA maxdiv ← k

k ← k+1 }

FINCHE' k = n

SE maxdiv > 1 ALLORA
SCRIVI "Non primo"
ALTRIMENTI
SCRIVI "Primo"

FINE

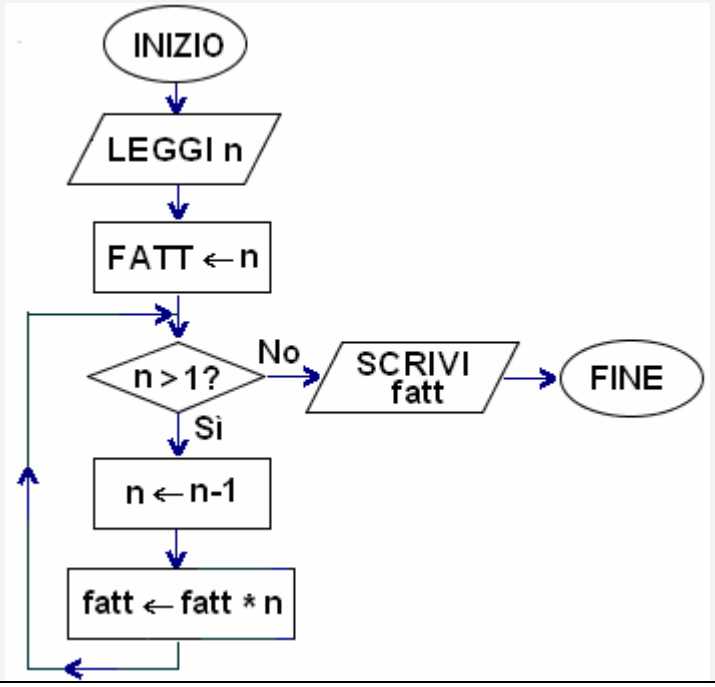
❑ ESEMPIO 3

Il seguente algoritmo calcola il **fattoriale** $\mathbf{n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 2 \cdot 1}$ di un intero $n \geq 1$ letto in ingresso.

n, se è maggiore di 1, viene fatto decrescere sempre di un'unità a partire dal valore iniziale,
e una variabile fatt, inizializzata ad n, viene moltiplicata per ogni nuovo valore di n
generando così, via via, il fattoriale desiderato.

Affinché il procedimento funzionasse anche con n = 1,
valore per il quale il decremento di n NON va effettuato nemmeno una volta,
in questo algoritmo abbiamo utilizzato la STRUTTURA ITERATIVA A CONTROLLO INIZIALE,
che in linguaggio di progetto abbiamo tradotto con FINTANTOCHE' … ESEGUI …

DIAGRAMMA DI FLUSSO



LINGUAGGIO DI PROGETTO

INIZIO

LEGGI n

fatt ← n

FINTANTOCHE' n > 1 ESEGUI

{ n ← n – 1

fatt ← fatt * n }

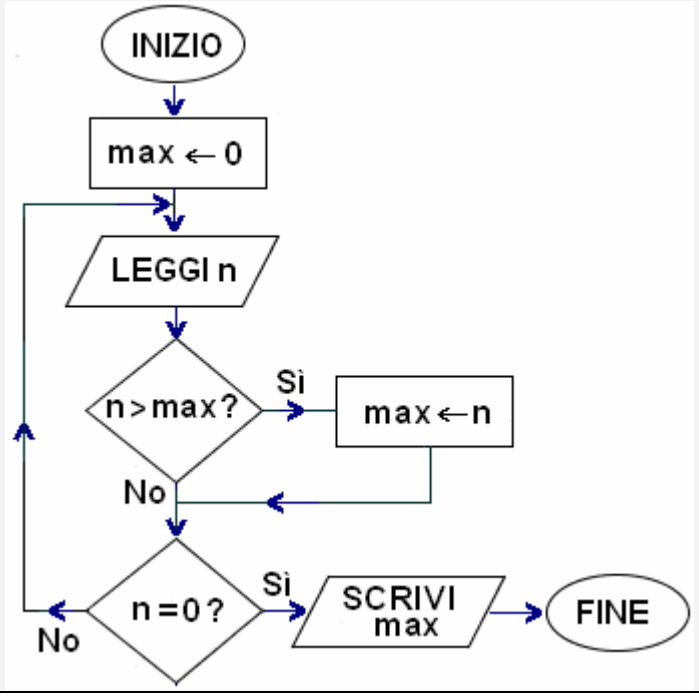
SCRIVI fatt
FINE

Nel nostro linguaggio di progetto la moltiplicazione è indicata con l'asterisco *, come nei linguaggi Pascal e C.

❑ ESEMPIO 4

Si leggono in input più numeri positivi, anche eventualmente tantissimi, e se ne stabilisce il massimo.
La fine della sequenza numerica in ingresso viene segnalata comunicando un "**numero sentinella**": lo 0.

DIAGRAMMA DI FLUSSO



LINGUAGGIO DI PROGETTO

INIZIO

max ← 0

RIPETI

{ LEGGI n

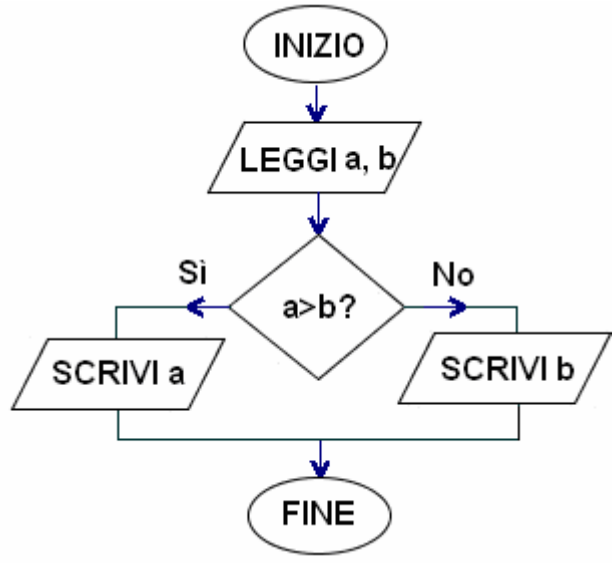
SE n > max ALLORA max ← n }

FINCHE' n=0
SCRIVI max
FINE

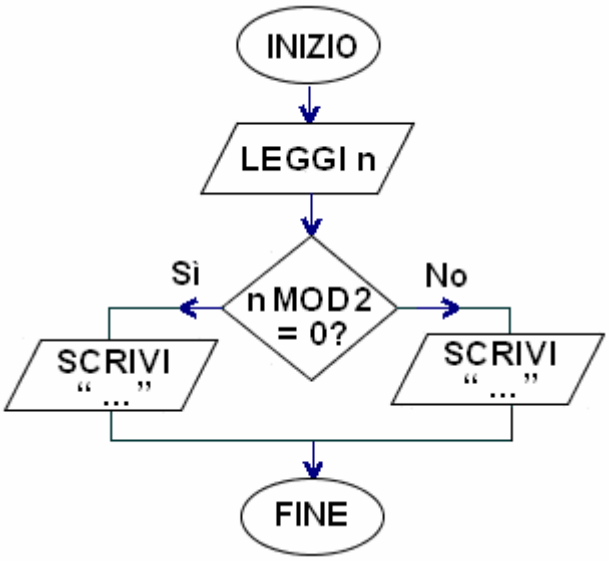
5 - ESERCIZI (le risposte sono alle pagg. 14 e 15)

Nei seguenti esercizi ti converrà preparare su di un foglio di carta tante caselle quante sono le variabili in gioco, per segnare, istruzione dopo istruzione, come cambia (ammesso che cambi) il contenuto di ciascuna casella.
In tal modo, avrai fatto la cosiddetta "TRACCIA" (inglese: trace) del procedimento.

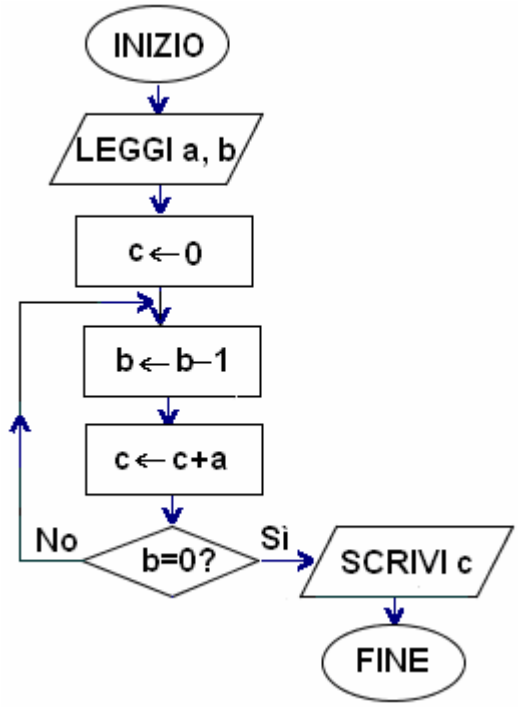
1) Cosa fa il seguente algoritmo?
(a, b sono due numeri qualsiasi)



2) Cosa metteresti fra virgolette?
(n è un numero naturale)



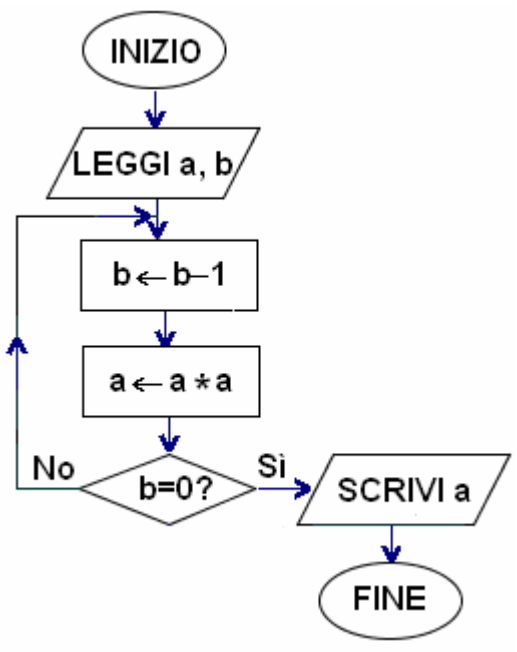
3) Cosa fa il seguente algoritmo?
(a è un numero naturale, b un numero naturale >0)



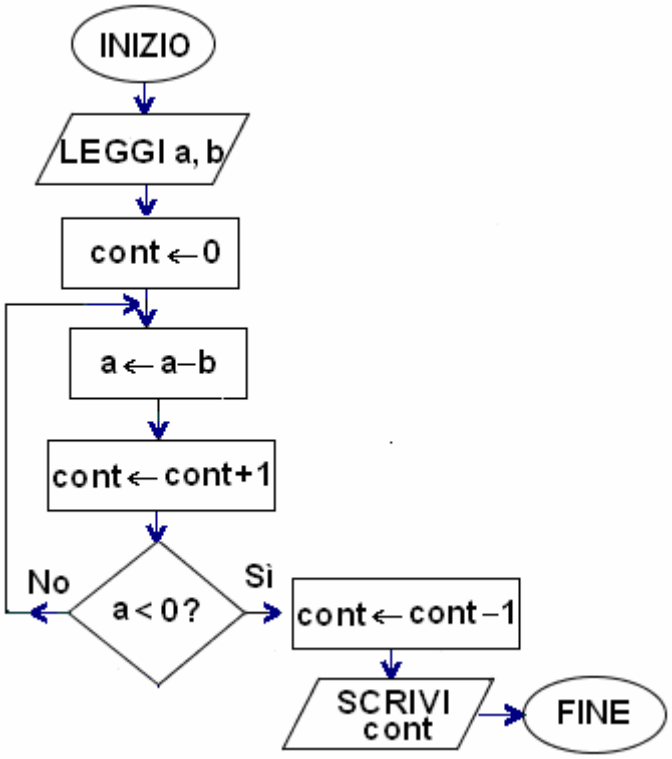
4) L'algoritmo che segue
(dove a, b sono due interi positivi)
è un po' strambo.

Cosa produrrebbe in output,
se l'input fosse a = 10, b = 3?

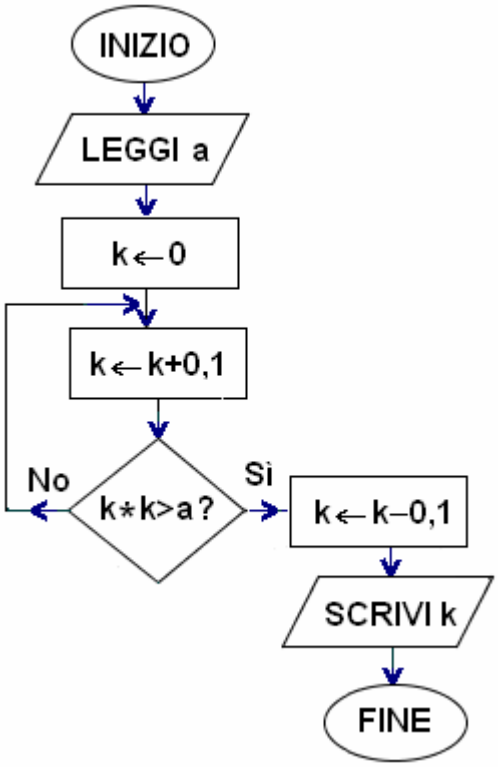
E in generale,
qual è l'espressione algebrica
contenente a, b (un po' strana ...)
che corrisponde all'output?



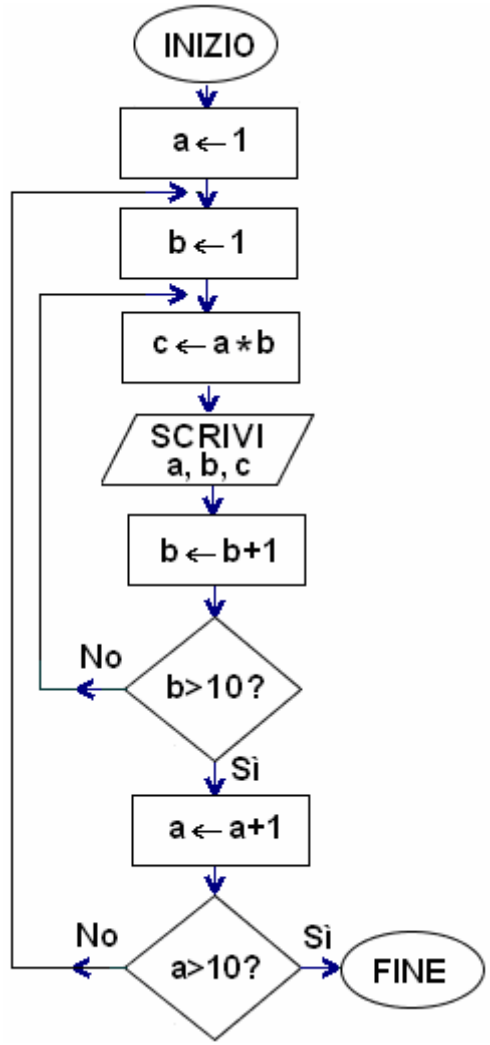
5) Cosa fa il seguente algoritmo? (a, b interi positivi)



6) Cosa fa il seguente algoritmo?
(a è un numero reale positivo)



7) Cosa produce in output l'algoritmo che segue?

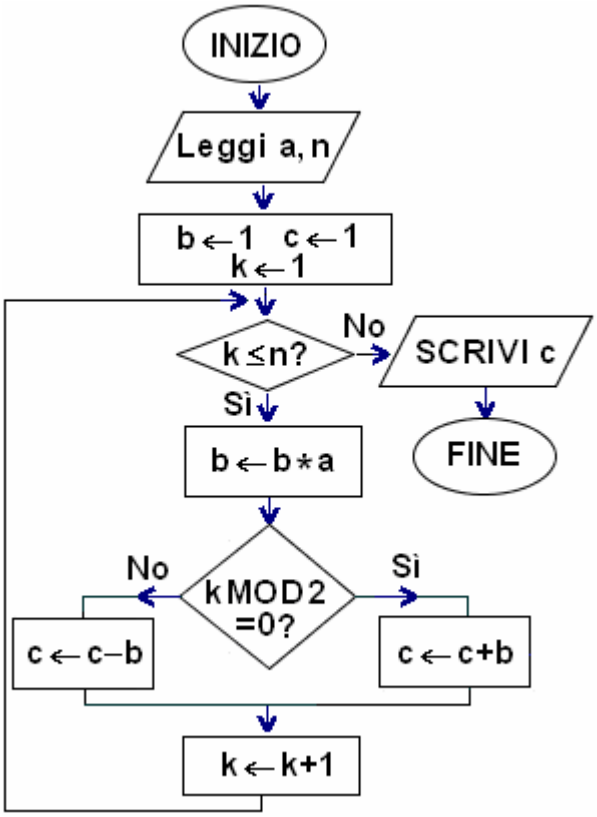


8) L'algoritmo seguente genera in output il numero

$$c = 1 - a + a^2 - a^3 + a^4 - \ldots + (-1)^n a^n$$

(a è un reale positivo, n un intero positivo).

Potrebbe essere simpatico mettersi d'accordo con due compagni/e: scelta una coppia a, n, ad esempio: $a = 5$, $n = 4$, uno farà la parte di b, un altro la parte di c, il terzo quella di k.
Buon divertimento.



9) *Nel seguente algoritmo*
m, n sono due interi positivi.

```
LEGGI m, n
x ← 0
k ← 0
FINTANTOCHE' k<n ESEGUI:
   { k ← k+1
     x ← x+m }
SCRIVI x
```

Qui a fianco, a titolo di esempio,
è riprodotta la "traccia" nel caso m = 7, n = 3

"Traccia" (nel caso m = 7, n = 3)	
m 7	n 3
x ~~0~~ ~~7~~ ~~14~~ 21	k ~~0~~ ~~1~~ ~~2~~ 3

Determina la "traccia" e l'output quando l'input è m = 5, n = 4 .
Quale operazione aritmetica realizza questo algoritmo?

10) L'algoritmo seguente sarebbe finalizzato a scambiare i valori delle due variabili a, b,
cioè a scambiare i contenuti delle due "scatolette" a, b; sennonché … è sbagliato. Non funziona!
Perché?

```
LEGGI a, b
a ← b
b ← a
SCRIVI a, b
```

"Traccia" (nel caso a = 4, b = 5)	
a	b

11) *Nell'algoritmo che segue*
a, b sono due numeri qualsiasi.

```
LEGGI a, b
x ← a
a ← b
b ← x
SCRIVI a, b
```

Qual è l'output per a = 4, b = 5 ?

12) *Nel quesito seguente,*
a, b sono due interi >0.

```
LEGGI a, b
k ← 0
FINTANTOCHE' a ≥ b ESEGUI
  {a ← a–b
   k ← k+1}
SCRIVI k
```

Output per: a = 16, b = 3? a = 12, b = 2?
Quale operazione esegue, in pratica, questo algoritmo?

13) *Nel seguente quesito,*
a è un numero qualsiasi, n è un intero >0

```
LEGGI a, n
b ← 1
k ← 0
RIPETI
  { k ← k+1
    b ← b*a }
FINCHE' k = n
SCRIVI b
```

Qual è l'output di questo algoritmo
nel caso a = 5, n =3 ?
E nel caso a = 2, n = 10 ?
Riusciresti a modificare l'algoritmo
in modo che funzioni anche nel caso n = 0 ?

14) *a, b, c sono 3 numeri qualsiasi*

```
LEGGI a, b, c
SE b>a ALLORA
   {x ← a   a ← b   b ← x }
SE c>b ALLORA
       { SE c>a ALLORA
                {x ← a  y ← b  a ← c  b ← x  c ← y}
            ALTRIMENTI {x ← b  b ← c  c ← x}  }
SCRIVI a, b, c
```

Cosa fa questo algoritmo?

(MOLTO istruttivo fare con attenzione la "traccia"
scegliendo la terna a, b, c in più modi differenti!)

15) *n intero positivo*

```
LEGGI n
k ← 0
RIPETI
     { k ← k+1
       SCRIVI " * "}
FINCHE' k = n
```

Qual è l'effetto di questo algoritmo?

16) *a, b, c sono 3 numeri qualsiasi*

```
LEGGI a, b, c
SE a<=b ALLORA
   { SE a<=c  ALLORA x ← a  ALTRIMENTI x ← c }
ALTRIMENTI
   { SE b<=c  ALLORA x ← b  ALTRIMENTI x ← c }
SCRIVI x
```

Qual è l'effetto di questo algoritmo?

17) *Nel seguente quesito, n è un intero >1.*

```
LEGGI n
cont ← 0
RIPETI
    { cont ← cont+1
      SE n MOD 2 = 0  ALLORA  n ← n DIV 2
                      ALTRIMENTI  n ← 3n+1 }
FINCHE' n = 1
SCRIVI cont
```

Output per n = 5? E per n = 6? E per n = 7?

Verifica che per n = 27 il valore finale di cont è 111.

Fra gli interi n da 2 a 20, esiste un valore di n per cui l'esecuzione del procedimento non ha mai termine?

18) *Nel seguente quesito, n è un intero positivo.*

```
LEGGI a
RIPETI
    { b ← a MOD 10
      a ← a DIV 10
      SCRIVI b }
FINCHE' a = 0
```

Prova ad eseguire l'algoritmo con a = 2345. In generale, che relazione hanno in numeri scritti in output, col numero letto in input?

19) *Nel seguente quesito, n è un intero positivo.*

```
LEGGI n
s ← 0
k ← 0
RIPETI
    { k ← k+1
      s ← s+2k–1 }
FINCHE' k = n
SCRIVI s
```

Stabilisci cosa scrive questo algoritmo quando il numero n in ingresso è 5. E quando n = 8?

20) *Nel seguente quesito, a è un intero positivo.*

```
LEGGI a
cont ← 0
FINTANTOCHE' a<>1 ESEGUI
    { a ← a DIV 2
      cont ← cont+1 }
SCRIVI cont
```

Stabilisci cosa scrive questo algoritmo quando il numero a in ingresso è 33.
E quando a è inizialmente uguale a: 1? 1024?
Se si desse in ingresso a = 0, che succederebbe?

a) Scrivi un diagramma di flusso per un algoritmo che risolva i seguenti problemi;
b) traduci anche il diagramma in linguaggio di progetto

21) Leggere in ingresso un numero non prefissato di addendi, e scrivere la somma di tutti questi addendi.
Si intende che la sequenza di addendi sia terminata quando in input viene fornito il "numero sentinella" 0.

22) Leggere in ingresso un intero *n*, poi una sequenza di *n* numeri; eseguire il prodotto di questi.

23) Leggere in input una sequenza di numeri positivi qualsiasi, e scriverne il minimo e il massimo.
Servirsi dello 0 come "sentinella" per indicare la fine della sequenza.
OSSERVAZIONE
Non è facile come potrebbe sembrare, scrivere correttamente questo algoritmo: occorre infatti evitare che la "sentinella" 0, la quale NON fa parte dell'insieme dei numeri, vada a comprometterne gli esiti ...

24) Si chiama "successione di Fibonacci" la sequenza di numeri 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... così costruita:
- i primi due numeri della sequenza sono 0 e 1 rispettivamente;
- a partire dal 3°, ciascun termine della sequenza è ottenuto sommando i due termini che lo precedono.

Scrivi un algoritmo che, letto in ingresso n (≥ 3), scriva i primi n termini della successione di Fibonacci.

25) Vengono forniti in input più numeri; la sequenza terminerà quando verrà introdotto il "numero sentinella" 0.
Si desidera fornire in uscita la media aritmetica dei numeri dati.

26) Letto in ingresso un intero *n*, scriverne i divisori e contarli
(s'intende che l'algoritmo tenga conto anche dei divisori "impropri", che sono il numero stesso e l'unità).

27) Completa l'algoritmo precedente, in modo che fornisca in uscita anche la scritta
"Il numero è primo" oppure "Il numero non è primo", a seconda dei casi.
E' chiaro che la prima circostanza si verifica quando i divisori sono soltanto 2 (l'unità e il numero).

28) Vengono forniti in input più voti (dall'1 al 10); si vogliono contare quelli sufficienti, ossia non inferiori a 6.

29) Scrivi un algoritmo che, letti in ingresso due numeri interi *a*, *b*, ne calcoli il minimo comune multiplo.
Ci sono diversi modi alternativi per impostare l'algoritmo;
ad esempio, si può calcolare la successione dei multipli di *a* (a, $2a$, $3a$, $4a$, ...), arrestandosi quando si perviene ad un numero che risulta multiplo anche di *b* (cioè, che risulta divisibile per *b*).
E i successivi multipli di *a* possono essere generati
I) per somma ($mult \leftarrow mult + a$)
II) oppure tramite moltiplicazione per un intero ($mult \leftarrow a * k$, con k che assume i valori 1, 2, 3, …)

30) Un intero si dice "perfetto" se è uguale alla somma dei suoi divisori, inclusa l'unità ma escluso il num. stesso: ad esempio 28 è perfetto, perché ha come divisori (a parte sé stesso) $1, 2, 4, 7, 14$, e $1+2+4+7+14=28$.
Letto in input un intero *n*, stabilire se *n* è un numero "perfetto".
NOTA - Ricordiamo che *b* è divisore di *a* se e solo se a MOD $b = 0$ (MOD dà il *resto* della *divisione intera*)

31) Letto in ingresso un intero positivo *n*, stabilire se si tratta di un numero primo. Questa volta, però, è richiesto di *ridurre* il numero di passi che l'algoritmo dovrà compiere prima di fornire la risposta.
A tale scopo, non si stanno a individuare e contare *tutti* i divisori di *n*;
si prova invece a vedere se *n* è divisibile per 2, poi per 3, ecc., fermandosi quando
a) si trova un divisore
b) oppure il numero che è "candidato" ad essere un divisore di *n* ha già superato la radice quadrata di *n* sqrt(*n*) (infatti si dimostra che, se in tal caso non sono ancora stati trovati divisori propri, *n* non potrà avere alcun divisore proprio; i divisori propri sono quelli diversi dall'unità e dal numero stesso).

NOTA - In un blocco romboidale, la *condizione* può benissimo contenere connettivi logici come E, O

32) Letti in ingresso due interi positivi *a*, *b*, si determinano il quoziente intero e il resto della divisione intera *a*:*b* sottraendo dal numero *a* il numero *b* più volte.

33) Una marionetta puntiforme M si trova inizialmente alla distanza di *x* metri da una parete.
Improvvisamente, comincia a muoversi a scatti, facendo uno scatto al secondo, e ad ogni scatto percorrendo, in direzione della parete, uno spazio uguale alla META' della distanza che ancora la separa dalla parete.
Dopo quanti secondi la distanza diventerà minore di 1 milionesimo di millimetro?
Scrivi un algoritmo che risponda a questa domanda.

34) Scrivi un algoritmo che, letti in ingresso due numeri interi non nulli *a*, *b*, ne calcoli il massimo comun divisore con l'efficiente e ingegnoso "algoritmo di Euclide".

L'Algoritmo di Euclide per il calcolo del Massimo Comun Divisore di due interi

Siano a, b i due interi di cui vogliamo determinare il M.C.D.
Calcoliamo il resto r della divisione intera $a:b$,
e a questo punto sostituiamo la coppia (a,b) con la coppia (b, r):
$a \leftarrow b,\ b \leftarrow r$.

Si può dimostrare che il M.C.D. della "nuova" coppia coincide col M.C.D. della "vecchia".
Iteriamo (= ripetiamo) il procedimento per la nuova coppia;
prima o poi, poiché in questo modo il "b", evolvendosi, diventa sempre più piccolo,
si arriverà ad ottenere $b = 0$.
Ma a quel punto, essendo M.C.D. $(a, 0) = a$, il valore che avrà a in quel momento sarà il M.C.D. cercato.

Esempio:

$a = 108,\ b = 84$
$108 : 84 = 1$ *col resto di* $24 \rightarrow r = 24$
$a = \cancel{108}\,84,\ b = \cancel{84}\,24$
$84 : 24 = 3$ *col resto di* $12 \rightarrow r = 12$
$a = \cancel{84}\,24,\ b = \cancel{24}\,12$
$24 : 12 = 2$ *con resto* $0 \rightarrow r = 0$
$a = \cancel{24}\,12,\ b = \cancel{12}\,0$
Essendo $b = 0$, *il procedimento termina*: $M.C.D. = a = 12$

Schematicamente:

$a = 108,$	$b = 84$	$r = 24$
$a = 84,$	$b = 24$	$r = 12$
$a = 24,$	$b = 12$	$r = 0$
$a = 12,$	$\boxed{b = 0}$	*STOP*

M.C.D. $= 12$

35) Viene fornita in ingresso una somma di denaro in euro, con eventualmente decimi e centesimi (es.: 8,47).
L'algoritmo dovrà calcolare il minimo numero di monete necessarie per totalizzare quella cifra, se si ha a disposizione un numero a piacere di pezzi da: 2 euro, 1 euro; 50, 20, 10, 5, 2 e 1 centesimo.

36) L'offerta promozionale di un negozio prevede il 10% di sconto su tutti gli acquisti, e addirittura, per spese superiori ai 100 euro, il normale 10% fino a 100 euro, e il 25% di sconto sulla cifra spesa in più.
Quanto pagherà, in seguito all'offerta, una persona che ha comprato merce per *x* euro?
Scrivi un algoritmo che risponda a questa domanda.

37) Ci sono 8 palline tutte identiche all'apparenza, ma mentre 7 di esse hanno pure ugual peso, una invece ha peso minore delle altre.
Trova un algoritmo per stabilire quale sia la pallina anomala, tramite 2 pesate con una bilancia a due piatti.

38) Ci sono 9 palline tutte identiche all'apparenza, ma mentre 8 di esse hanno pure ugual peso, una invece ha peso minore delle altre.
Trova un algoritmo per stabilire quale sia la pallina anomala, tramite 2 pesate con una bilancia a due piatti.

RISPOSTE

1) Scrive il massimo fra due numeri letti in ingresso

2) "n è pari" (dalla parte del "Sì"), "n è dispari" (dalla parte del "No")

3) Moltiplica a per b col metodo delle "somme ripetute"

4) 100 000 000 (cento milioni). L'espressione è a^{2^b} (a elevato all'esponente 2^b)

5) Conta "quante volte b sta in a", ossia determina il quoziente della divisione intera a:b.
In pratica, l'algoritmo equivale all'operazione a DIV b.

6) Approssima per difetto, con 1 cifra dopo la virgola, la radice quadrata di a.

7) La tabellina del 10 (tutte le coppie di interi da 1 a 10, con accanto il risultato della moltiplicazione)

8) Con $a = 5$, $n = 4$, l'output è 521.

9) L'output in questo caso è 20. Esegue la moltiplicazione m*n col metodo delle somme ripetute.

10) E' sbagliato perché con l'istruzione $a \leftarrow b$ il valore originario di a *va perso*,
in quanto nella "scatoletta" che lo conteneva esso viene "sovrascritto", viene rimpiazzato dal valore di b.
Quando poi viene eseguita l'istruzione successiva $b \leftarrow a$,
nella "scatoletta" b viene messo il valore "*attuale*" di a,
ossia il valore presente nella "scatoletta" a *al momento in cui l'istruzione viene eseguita*.
E tale valore è, come abbiamo visto, quello che era il valore iniziale di b.
In definitiva, se inizialmente era $a = 4$, $b = 5$,
alla fine, con questo algoritmo, è $a = 5$, $b = 5$.

11) L'output è
5 4.
In pratica, questo algoritmo scambia i contenuti delle "scatolette" a, b;
scambia i valori delle due variabili a, b.
C'è voluta una variabile ausiliaria; altrimenti (vedi esercizio precedente) non sarebbe stato possibile.

12) 5; 6.
Calcola il quoziente intero della divisione intera a:b.
L'algoritmo equivale all'operazione a DIV b.

13) 125; 1024.
E' chiaro che l'algoritmo calcola la potenza avente base a ed esponente n.
Un modo per modificarlo affinché funzioni correttamente anche con n = 0 potrebbe consistere,
ad esempio, nel sostituire la RIPETI … FINCHE' … con una FINTANTOCHE' … ESEGUI … :

```
LEGGI a, n
b ← 1
k ← 0
FINTANTOCHE' k<n ESEGUI
  { k ← k+1
    b ← b*a }
SCRIVI b
```

14) Scrive in ordine *decrescente* i 3 numeri letti

15) L'effetto è di scrivere tanti asterischi quant'è il numero letto in ingresso

16) Scrive il minimo fra tre numeri letti in ingresso

17) 5; 8; 16
No, non esiste.
Anzi: fino ad oggi TUTTI gli interi n coi quali si è provato hanno condotto a sequenze terminanti con 1.
E sono stati testati tutti i numeri interi da 2 fino a valori enormi (dell'ordine dei miliardi di miliardi).
Si è quindi portati a credere che *per qualsiasi* intero n il procedimento abbia termine
(ossia, porti, dopo un numero più o meno elevato di passi, a ottenere 1).
Tuttavia, questa congettura (nota come Congettura di Collatz, o Congettura del 3n+1, o con altri nomi)
alla data attuale non è stata ancora dimostrata.

18) Ne rappresentano le cifre, scritte in ordine invertito

19) 25; 64. In generale, si ottiene il quadrato di n, attraverso la somma dei primi n interi dispari

20) 5; 0; 10; con a=0 si otterrebbe un ciclo infinito, l'esecuzione non avrebbe mai termine.
In generale, con a>0, si ottiene in output l'esponente della potenza di 2 più vicina, per difetto, ad a.

22) Ad esempio:

```
INIZIO
LEGGI n
p ← 1
k ← 1
RIPETI
    {LEGGI a
     p ← p*a
     k ← k+1}
FINCHE' k>n
SCRIVI p
FINE
```

23) Ad esempio:

```
INIZIO
LEGGI a
min ← a
max ← a
RIPETI
  { SE a<min ALLORA min ← a
    ALTRIMENTI
    SE a>max ALLORA max ← a
    LEGGI a }
FINCHE' a=0
SCRIVI min, max
FINE
```

oppure:

```
INIZIO
LEGGI a
min ← a
max ← a
RIPETI
  {  LEGGI a
     SE a<>0 ALLORA
      { SE a<min ALLORA min ← a
        ALTRIMENTI
        SE a>max  ALLORA max ← a } }
FINCHE' a=0
SCRIVI min, max
FINE
```

NOTA1:
<> significa "diverso da"

NOTA 2:
l'algoritmo funzionerebbe anche se non si scrivesse ALTRIMENTI"

24) Ad esempio:

```
INIZIO
LEGGI n
a ← 0
b ← 1
SCRIVI a, b
i ← 3
RIPETI
    { c ← a+b
      SCRIVI c
      a ← b
      b ← c
      i ← i+1 }
FINCHE' i>n
FINE
```

25) Ad esempio:

```
INIZIO
somma ← 0
cont ← 0
RIPETI
    { LEGGI numero
      cont ← cont+1
      somma ← somma+numero }
FINCHE' numero=0
cont ← cont−1
media ← somma/cont
SCRIVI media
FINE
```

26) Ad esempio:

```
INIZIO
LEGGI n
contdiv ← 0
k ← 1
RIPETI
    {  SE n MOD k = 0
       ALLORA
          { SCRIVI k
            contdiv ← contdiv+1}
       k ← k+1   }
FINCHE' k>n
SCRIVI contdiv
FINE
```

33) Ad esempio:

```
INIZIO
LEGGI x
cont ← 0
FINTANTOCHE' x>=0,000000001
                    ESEGUI
                    { x ← x/2
                      cont ← cont+1 }
SE cont=0 ALLORA
    SCRIVI "la distanza è già inferiore
                a 1 milionesimo di mm"
ALTRIMENTI
    SCRIVI cont
FINE
```

34) **ALGORITMO DI EUCLIDE**

```
INIZIO
LEGGI a, b
RIPETI
    { r ← a MOD b
      a ← b
      b ← r }
FINCHE' b=0
SCRIVI a
FINE
```

37)

```
INIZIO
Pesa due gruppi di 3 palline ciascuno
SE la bilancia resta in equilibrio, ALLORA
                { pesa le due palline rimanenti;
                  prendi la più leggera }
ALTRIMENTI
   { prendi due palline dal gruppo meno pesante;
     SE la bilancia resta in equilibrio,
     ALLORA prendi la pallina rimanente
     ALTRIMENTI prendi la più leggera }
FINE
```

CHALLENGE
Un problema veramente TREMENDO è quello "delle 12 palline":

"*Ci sono 12 palline tutte identiche in apparenza;*
ma mentre 11 di esse hanno pure ugual peso,
una invece ha peso diverso dalle altre;
non si sa se questa pallina è più pesante o più leggera.
Trova un algoritmo per stabilire quale sia la pallina anomala, e se sia più pesante o più leggera delle altre, tramite 3 pesate con una bilancia a due piatti".