



**Lezioni di laboratorio
di programmazione e calcolo**

INDICE

Premessa		pag. VII
1	Introduzione al corso	"
2	Algoritmi e Macchina di Von Neumann	1
2.1	Il concetto di algoritmo	37
2.2	La Macchina di Von Neumann	38
"		55
3	Rappresentazione dei dati e delle istruzioni	71
3.1	Il tipo alfano numerico	"
3.2	Il tipo intero	79
3.3	Il tipo reale	85
3.4	Il tipo logico	98
3.5	La rappresentazione delle istruzioni	135
"		151
4	La progettazione degli algoritmi: componenti di base e metodologie di sviluppo	179
4.1	Variabili e costanti	"
4.2	Strutture di controllo	180
4.3	Variabili strutturate (array)	200
4.4	Le procedure	236
"		265
5	La complessità computazionale	307
6	Il linguaggio Fortran 77	351
7	I sistemi operativi	417

Premessa

In questo volume sono raccolte le copie dei lucidi che sono stati realizzati e utilizzati, a partire dalla metà degli anni '80, per le lezioni del corso di Calcolo Numerico e Programmazione I del Corso di Laurea in Matematica dell'Università di Napoli "Federico II", tenuto dal prof. Almerico Murli.

Tali lucidi sono stati in seguito acquisiti come principale strumento didattico in numerosi altri corsi. In particolare si ricordano il corso di Analisi Numerica del Corso di Laurea in Matematica della Seconda Università di Napoli e i corsi di Laboratorio di Programmazione e Calcolo e di Calcolo Numerico e Programmazione dei Corsi di Laurea in Chimica e in Chimica Industriale dell'Università di Napoli "Federico II".

giuliano.lacetti@dma.unina.it
tel. 081/675618

INTRODUZIONE AL CORSO

- Prolusione

OBIETTIVO DEL CORSO

Risolvere un problema
(di tipo matematico)
col calcolatore



Fornire idee di base,
metodi, algoritmi e strumenti software
per la risoluzione di problemi reali

CALCOLO SCIENTIFICO

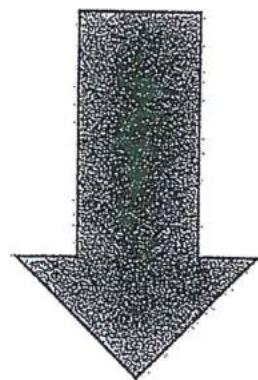
insieme di teorie, metodologie
e tecniche necessarie per risolvere
problemi della scienza
e della tecnica mediante calcolatore



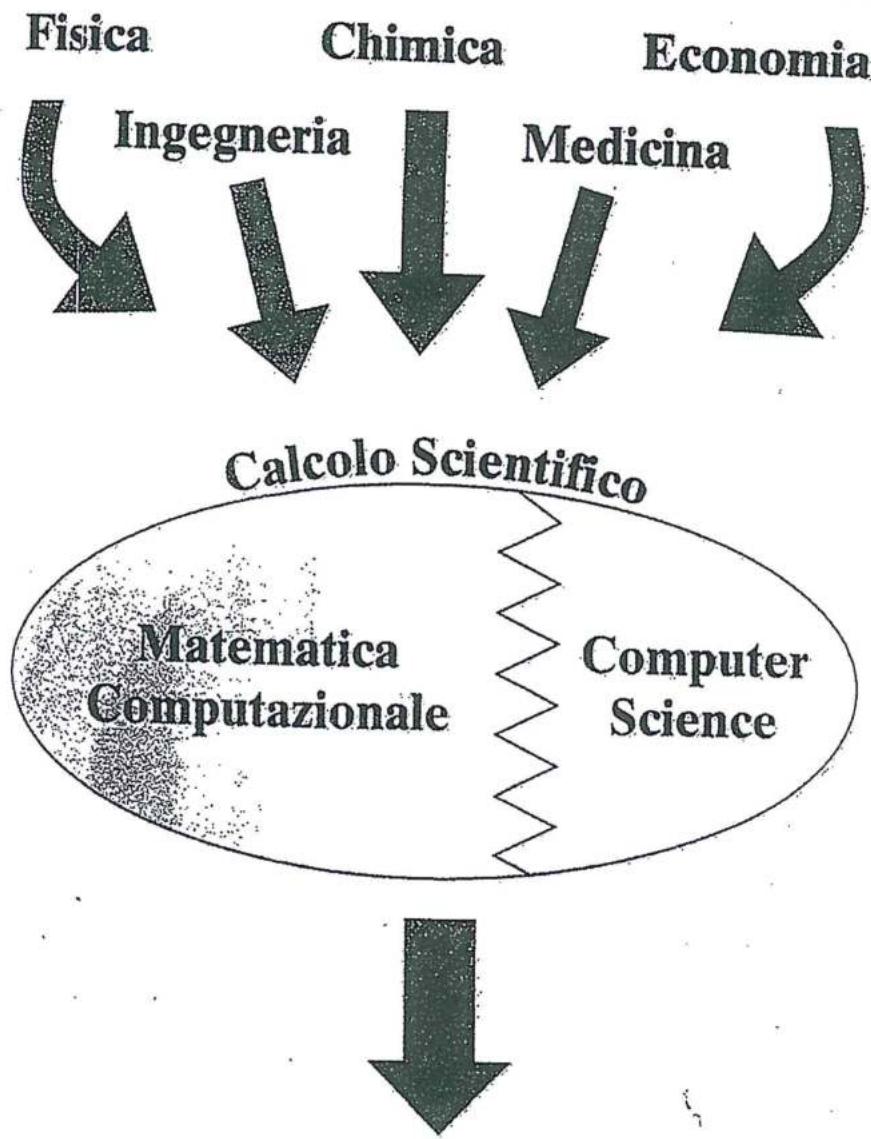
approccio computazionale

CALCOLO NUMERICO (Matematica Computazionale)

**studio, sviluppo, analisi
dei metodi computazionali
e delle tecniche implementative
per la risoluzione dei problemi
della scienza e della tecnica**

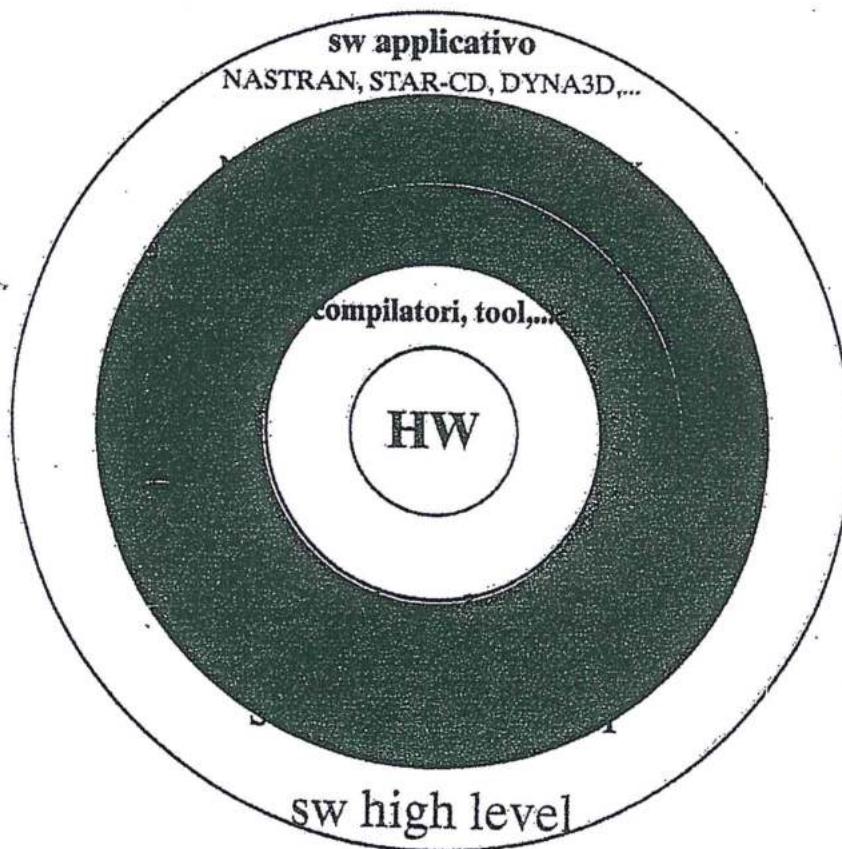
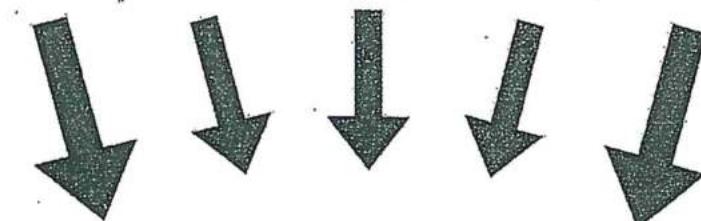


ruolo predominante del calcolo numerico nel calcolo scientifico

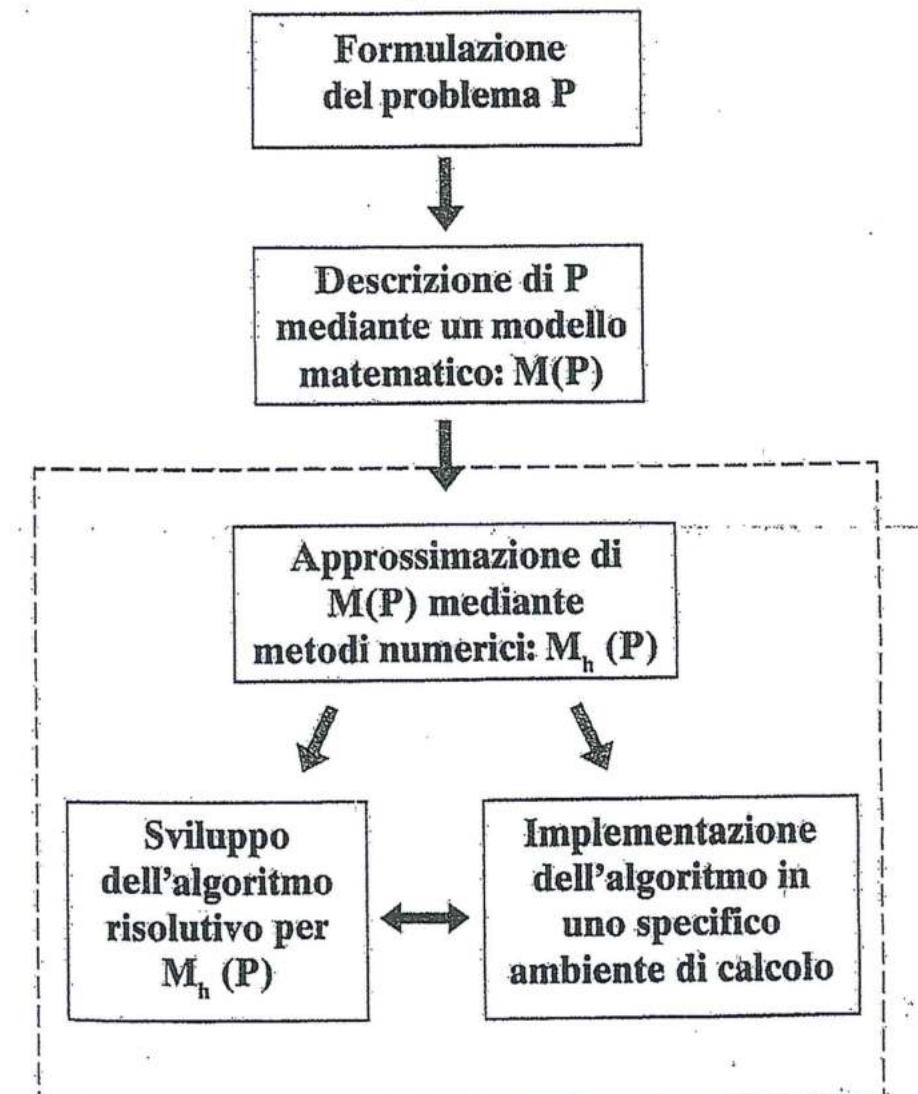


Problemi del mondo reale

Fisica, Chimica, Biologia, Economia, Medicina,...

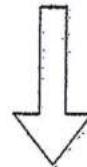


Fasi del processo di risoluzione di un problema



Possibili fonti di errore

Problema reale

P

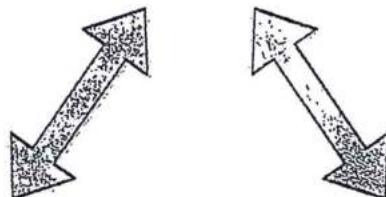
Errori causati da semplificazioni nella formulazione di $M(P)$

Modello matematico

 $M(P)$ 

errori di truncamento analitico

Modello numerico

 $M_h(P)$ 

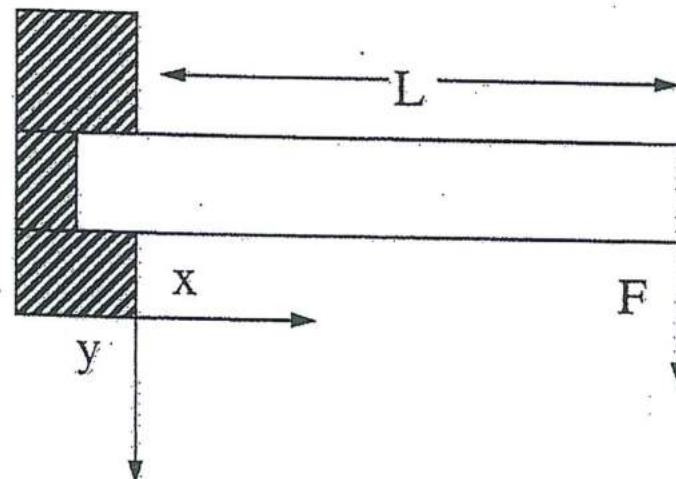
errori di roundoff

Algoritmo \leftrightarrow Programma

Dal Problema al Modello

Esempio 1:

Deflessione elastica di una sbarra di lunghezza L con una sola estremità libera, sottoposta ad un carico F applicato alla estremità libera.



x misura la distanza lungo l'asse della sbarra rispetto all'origine del riferimento coincidente con l'estremità fissa.

y misura la deflessione della sbarra con direzione positiva verso il basso.

La deflessione elastica y soddisfa l'equazione differenziale ordinaria:

$M(P)$:

$$\frac{y''}{[1 + (y')^2]^{3/2}} = \frac{F(L-x)}{E \cdot I}$$

dove:

E costante dipendente dal materiale
I momento di inerzia

con le condizioni:

$$\begin{aligned}y(0) &= 0 \\y'(0) &= 0\end{aligned}$$

(essendo la sbarra fissa alla estremità corrispondente a $x=0$, sia la deflessione y sia la sua pendenza y' sono nulle nell'origine del riferimento).

Modello semplificato $M'(P)$

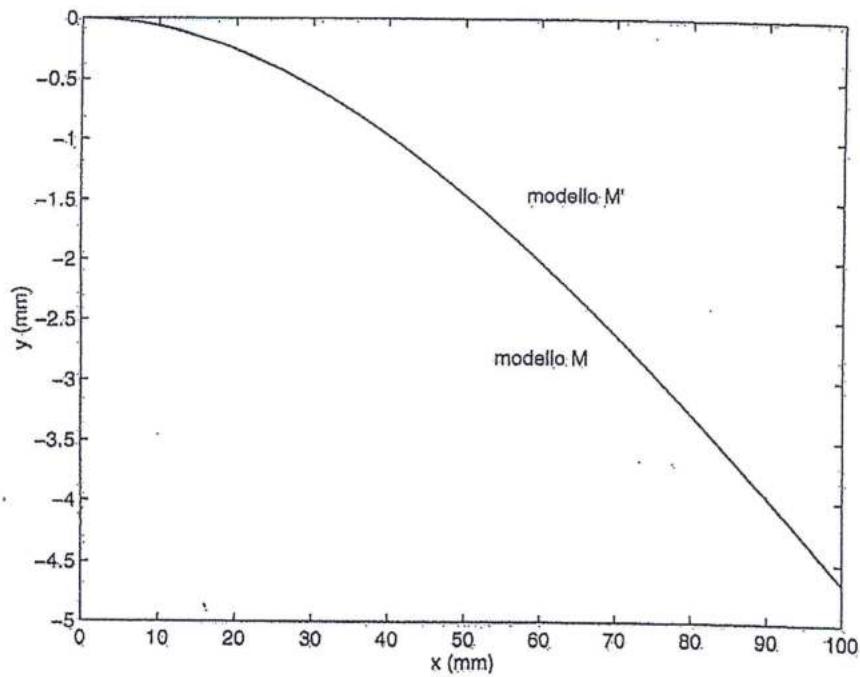
In molte applicazioni la pendenza y' della deflessione è così piccola da aversi $(y')^2 \ll 1$, pertanto si ha:

$M'(P)$:

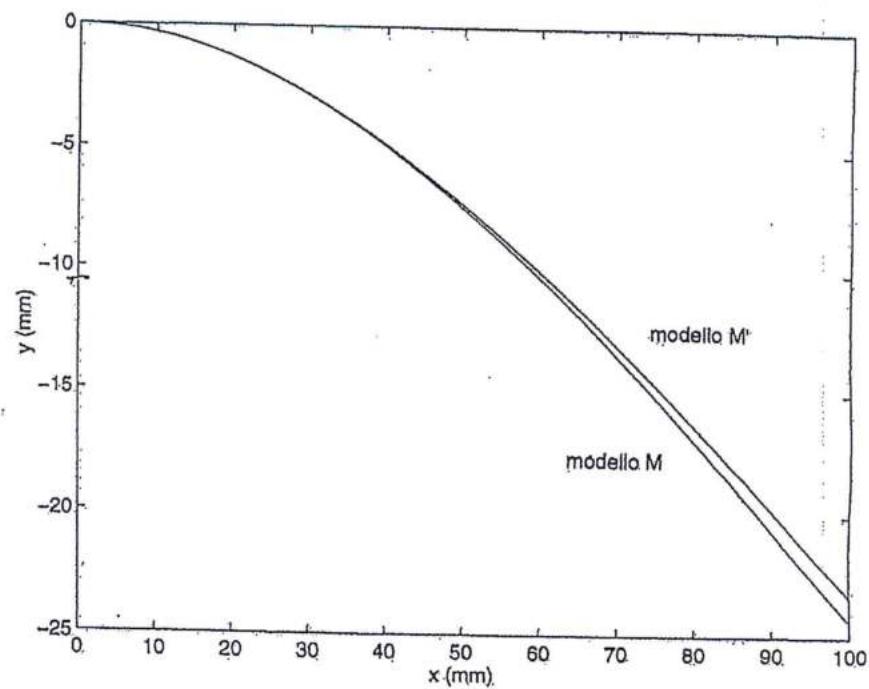
$$y'' = \frac{F(L-x)}{E \cdot I}$$

$$\begin{cases} y(0) = 0 \\ y'(0) = 0 \end{cases}$$

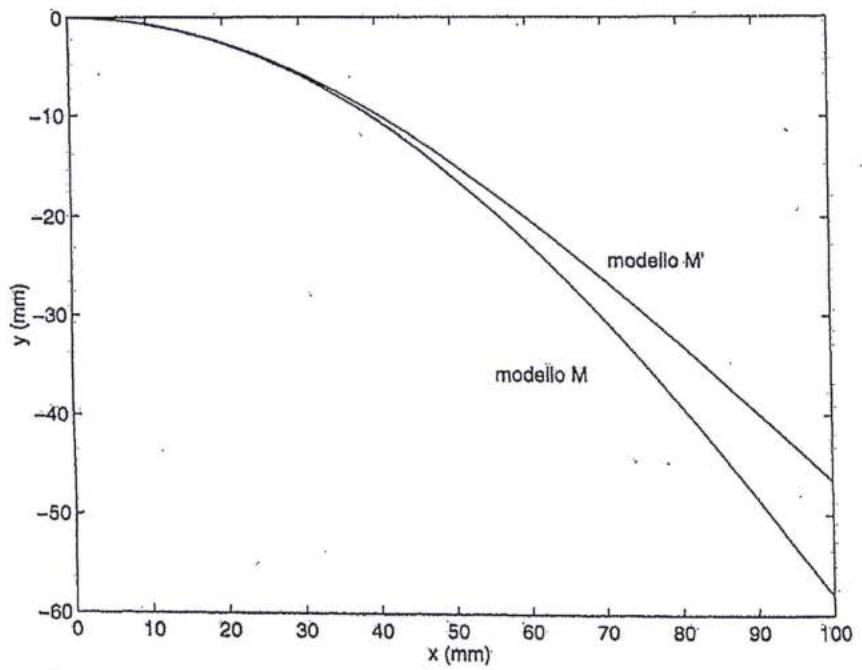
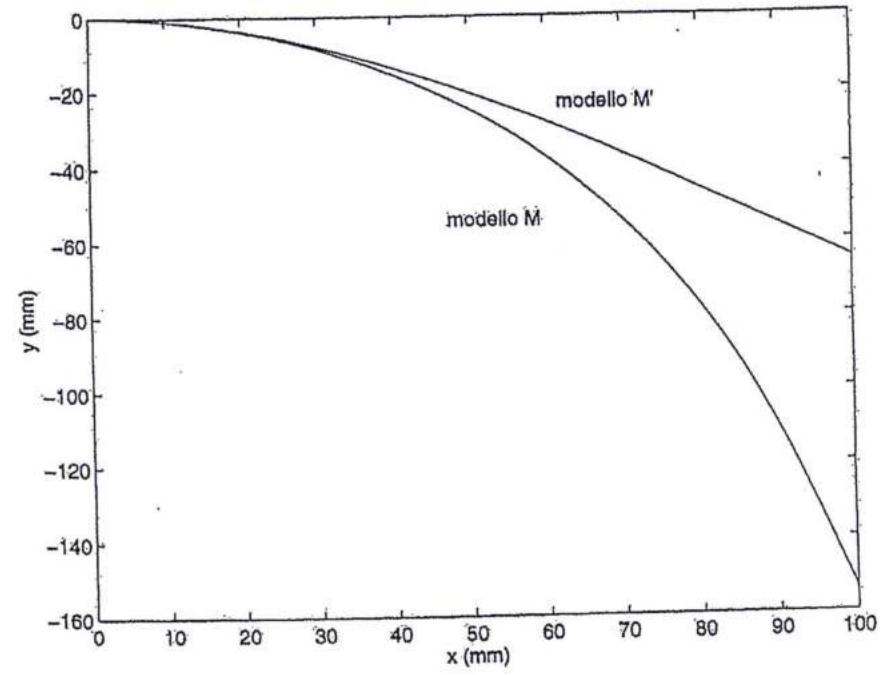
Tale modello è valido per "piccoli" valori della forza F .



$$F = 1$$



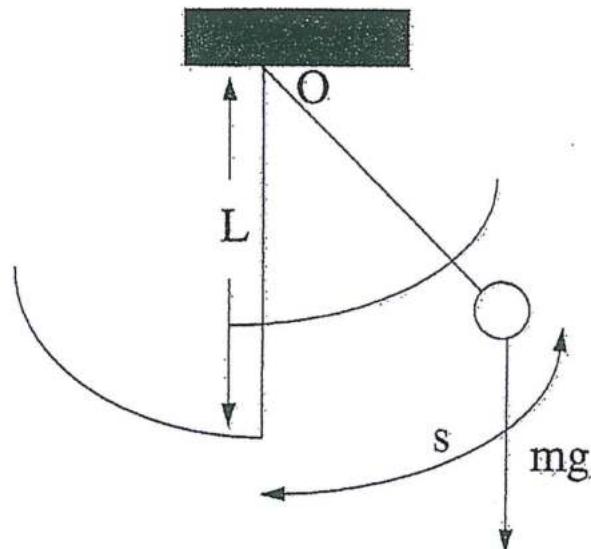
$$F = 5$$

 $F = 10$  $F = 15$

Dal Problema al Modello

Esempio 2:

Moto di un pendolo semplice costituito da una massa m sospesa ad un punto fisso O mediante un filo di lunghezza L .



s misura la lunghezza dell'arco descritto dal pendolo a partire dalla posizione di equilibrio (punto più basso della circonferenza di centro O e raggio L).

g è l'accelerazione di gravità.

La lunghezza dell'arco s soddisfa l'equazione differenziale:

$M(P)$:

$$\frac{d^2s}{dt^2} = -g \sin \frac{s}{L}$$

Se s è piccolo rispetto a L :

$$\sin \frac{s}{L} \approx \frac{s}{L}$$

e quindi $M'(P)$ (modello semplificato) è

$$\frac{d^2s}{dt^2} = -\omega^2 s \quad \left(\omega^2 = \frac{g}{L} \right)$$

Il moto è *un moto armonico semplice* la cui soluzione è nota ed è:

$$s = s_0 + \cos(\omega t + \delta)$$

con:

s_0 spostamento massimo subito dal pendolo;

δ spostamento angolare all'istante $t = 0$

Dal Modello Matematico al Metodo Numerico

Esempio 2 (cont.):

$M_h(P)$

$$s \approx \bar{s} = s_0 + 1 - \frac{(\omega t + \delta)^2}{2} + \frac{(\omega t + \delta)^4}{4!} - \dots$$

$$\dots + (-1)^n \frac{(\omega t + \delta)^{2n}}{(2n)!}$$

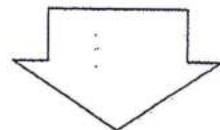
la differenza tra la soluzione s di $M^*(P)$ e la soluzione \bar{s} di $M_h(P)$ è l'errore di troncamento analitico

Dal Modello Matematico al Metodo Numerico

Esempio 3:

$M(P)$:

$$\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$



$M_h(P)$:

$$\cos x \approx 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \dots + \frac{(-1)^n x^{2n}}{(2n)!}$$

Le operazioni matematiche basate sul concetto di infinito sono sostituite da processi matematici costituiti da un numero finito di operazioni aritmetiche.

Dal Modello Matematico al Metodo Numerico

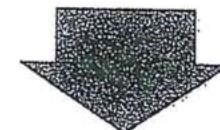
Esempio 4:

$M(P)$:

$$y' = \frac{dy}{dx} = \lim_{h \rightarrow 0} \frac{y(x+h) - y(x)}{h}$$

$M_h(P)$:

$$y_h = \frac{y(x+h) - y(x)}{h}$$



errore di troncamento analitico:

$$y' - y_h$$

Dal Metodo Numerico all'Algoritmo

rappresentazione dei dati e esecuzione delle operazioni aritmetiche in un sistema aritmetico a precisione finita (cioè con un numero finito di cifre).



ERRORE DI ROUNDOFF

Esempio 2 (cont.):

L'algoritmo per il calcolo della soluzione \bar{s} consiste nella somma, in un sistema aritmetico a precisione finita, di un numero finito di termini dello sviluppo in serie del coseno.

$$t = 1 \text{ (sec)}, \quad \omega = 3 \text{ (rad/sec)}, \quad \delta = 0,$$

$$s_0 = 1:$$

eseguiamo la somma dei primi 7 termini supponendo di poter rappresentare al più 7 cifre di un numero reale.

$$\begin{aligned} &+ 1. \\ &- 1. \\ &+ 4.5 \\ &- 3.375 \\ &+ 1.0125 \\ &- .1627232 \\ &+ .0162723 \\ &.0011094 = 1 - 0.9899397 = .0100603 \\ &\text{(valore esatto di } \bar{s} = 1 - 0.989939630681818 \dots = \\ &\quad = 0.01006036931818199 \dots) \end{aligned}$$

Esempio 5:

calcolo di $e^{-5.5}$ con un errore di troncamento analitico $R_n \leq 10^{-7}$.

$$\forall x \in [-5.5, 0],$$

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

con:

$$|R_n(x)| = \left| e^x - \sum_{i=0}^n \frac{x^i}{i!} \right| \leq \frac{e^0 |-5.5|^{n+1}}{(n+1)!}$$

$$\frac{|-5.5|^{25+1}}{(25+1)!} < 10^{-7} < \frac{|-5.5|^{24+1}}{(24+1)!}$$

bisogna sommare i primi 25 termini della serie

Esempio 5 (cont.):

Eseguiamo la somma dei primi 25 termini dello sviluppo in serie di $e^{-5.5}$ supponendo di poter utilizzare solo 5 cifre:

+ 1.0000
 - 5.5000
 + 15.125
 - 27.370
 + 38.129
 - 41.942
 + 38.446
 - 30.208
 + 20.768
 - 12.692
 + 6.9803

= 0.0026363 INVECE DI 0.0040867....

ERRORE DI ROUNDOFF

COLETTA MARVO

Stabilità di un Algoritmo

Esempio 6: calcoliamo

$$x_n = a \cdot x_{n-1}$$

con

$$x_0 = \pi = 3.1415926 \dots, \quad a = 10$$

Se si utilizzano 7 cifre per la rappresentazione di π :

$$x_0 = \pi \approx 3.141593 = \pi + \varepsilon = \tilde{x}_0$$

si ha:

$$\tilde{x}_1 = 10\tilde{x}_0 = 10(\pi + \varepsilon) = 10\pi + 10\varepsilon = 31.415926\dots + 10\varepsilon$$

$$\begin{aligned} \tilde{x}_2 &= 10\tilde{x}_1 = 10(\pi_1 + 10\varepsilon) = 10\pi_1 + 10^2\varepsilon = \\ &= 314.1597\dots + 10^2\varepsilon \end{aligned}$$

$$\begin{aligned} \tilde{x}_n &= 10\tilde{x}_{n-1} = 10(\pi_{n-1} + 10^{n-1}\varepsilon) = \\ &= 10\pi_{n-1} + 10^n\varepsilon = x_n + 10^n\varepsilon \end{aligned}$$

$$x_n - \tilde{x}_n = 10^n \cdot \varepsilon$$



l'errore sul dato iniziale $x_0 = \pi = 3.1415926 \dots$ si è amplificato su \tilde{x}_n di un fattore pari a 10^n



ALGORITMO INSTABILE

In generale, nel calcolo di

$$x_n = a \cdot x_{n-1}$$

Se $\tilde{x}_0 = x_0 + \varepsilon$ si ha:

$$\tilde{x}_1 = a \cdot \tilde{x}_0 = a(x_0 + \varepsilon) = ax_0 + ae = x_1 + ae$$

$$\tilde{x}_2 = a \cdot \tilde{x}_1 = a(x_1 + ae) = ax_1 + a^2\varepsilon = x_2 + a^2\varepsilon$$

.....

$$\tilde{x}_k = a \cdot \tilde{x}_{k-1} = a(x_{k-1} + a^{k-1}\varepsilon) = ax_{k-1} + a^k\varepsilon = x_k + a^k\varepsilon$$



$a > 1$ l'errore su x_0 viene amplificato
ALGORITMO INSTABILE

$a < 1$ l'errore su x_0 non viene amplificato
ALGORITMO STABILE

Malcondizionamento di un problema Matematico

Esempio 7:

Problema P:

$$\begin{cases} 2.10x + 3.50y = 8 \\ 4.190x + 7y = 15 \end{cases}$$



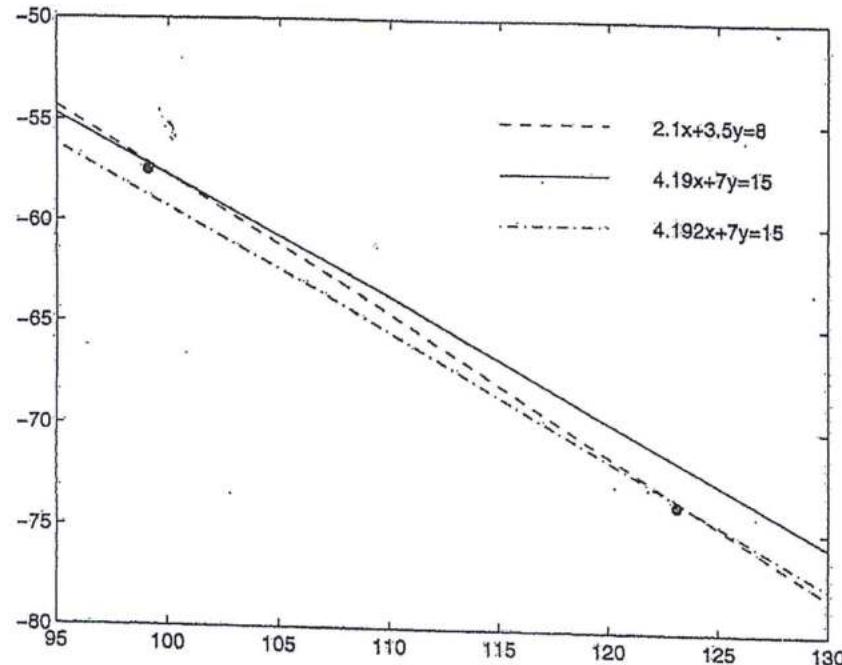
soluzione s: (100, -57.714)

Problema P':

$$\begin{cases} 2.10x + 3.50y = 8 \\ 4.192x + 7y = 15 \end{cases}$$



soluzione s': (125, -72.714)



Esempio 8:

Problema P: risoluzione di:

$$(x-2)^4 = 0$$



soluzione s: 2 (molteplicità 4)

Problema P': risoluzione di:

$$(x-2)^4 = 10^{-8}$$



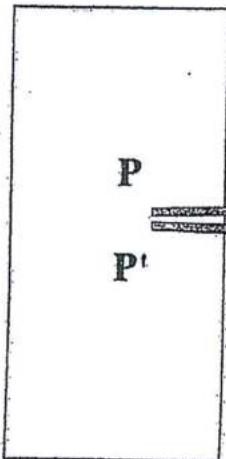
soluzione s': 1.99, 2.01, 2+i0.01, 2-i0.01

Problemi matematici molto sensibili
alle perturbazioni sui dati

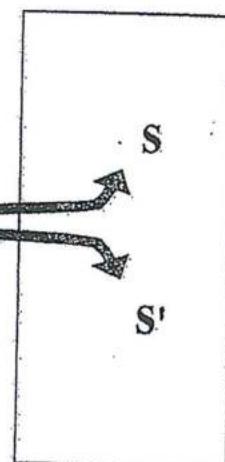


PROBLEMI MALCONDIZIONATI

problemi



soluzioni



Efficienza di un Algoritmo

Problema:

risoluzione di un sistema
di N equazioni lineari in N incognite

Algoritmo



CRAMER
 $(N+1)!$

GAUSS
 $N^3/3 + N^2 - N/3$

moltiplicazioni



Esempio 9:

se il tempo per eseguire una operazione aritmetica è

$$t_M = 10^{-7} \text{ sec.}$$

N	CRAMER	GAUSS
12	10 min.	$7 \cdot 10^{-5}$ sec.
13	2 ore	$9 \cdot 10^{-5}$ sec.
14	1 giorno	$1 \cdot 10^{-4}$ sec.
20	10^5 anni	$3 \cdot 10^{-4}$ sec.
40	10^{35} anni	$1 \cdot 10^{-3}$ sec.
50	10^{50} anni	$4 \cdot 10^{-2}$ sec.

It can be argued that the "mission" of numerical analysis is to provide the scientific community with effective software tools.

[*La "missione" dell'analisi numerica è fornire alla comunità scientifica efficaci strumenti software.*]

... good software development demands a mathematical understanding of the problem to be solved, a flair for algorithmic expression, and an appreciation for finite precision arithmetic.

[*... lo sviluppo di un buon software richiede la conoscenza matematica del problema da risolvere, una mentalità algoritmica e una comprensione dell'aritmetica a precisione finita.*]

[G. Golub, 1989]

What is numerical analysis?...

I propose the following definition with which to enter the new century:

"Numerical analysis is the study of algorithms for the problems of continuous mathematics"...

The pivotal word is "algorithm".

[*Che cos'è l'analisi numerica?*

Io propongo la seguente definizione con cui entrare nel nuovo secolo:

"L'analisi numerica è lo studio degli algoritmi per i problemi della matematica continua".

La parola centrale è "algoritmo".]

[Trefethen, 1992]

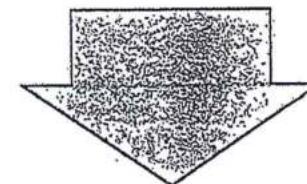
ALGORITMI E MACCHINA DI VON NEUMANN

- Il concetto di Algoritmo
- La Macchina di Von Neumann

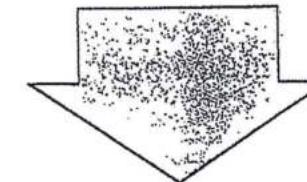
Risoluzione del Problema

- Precisa formulazione del problema
- Ricerca dell'Algoritmo, cioè di
 - un procedimento non ambiguo,
formato da
numero finito di azioni
sufficientemente semplici
che risolva il problema dato

Dati del Problema



ALGORITMO



RISULTATO

Esempio :

sostituzione della ruota di un'auto

istruzioni:

solleva l'auto
svita i bulloni
togli la ruota
metti la ruota di scorta
avvita i bulloni
abbassa l'auto

Un algoritmo viene descritto mediante un

LINGUAGGIO

- Flow chart (diagramma di flusso)
- Tipo Pascal, tipo Matlab, ...
-
-

Convenzioni linguistiche del

FLOW CHART

inizio

denota l'inizio dell'algoritmo



fine

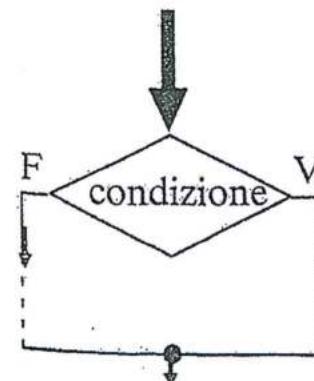
denota la fine dell'algoritmo

Convenzioni linguistiche del

FLOW CHART

istruzione

denota una istruzione



denota che l'esecuzione
dipende dal valore della
condizione (vero o falso)

Flow chart dell'algoritmo per il cambio della ruota



La non ambiguità di una istruzione dipende dalle

**CARATTERISTICHE
DELL'ESECUTORE**

l'istruzione

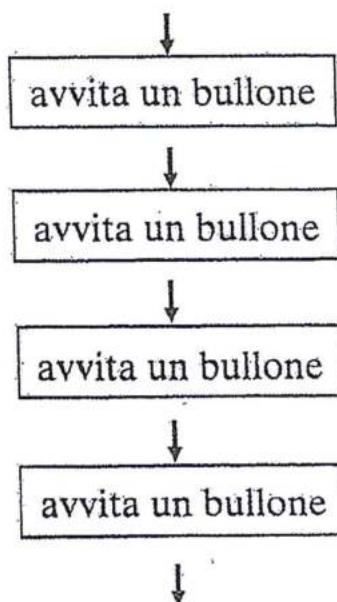
avvita i bulloni

è sufficientemente specificata?

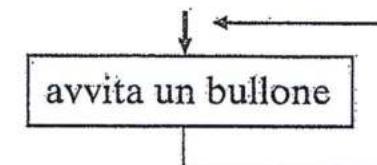
l'istruzione

avvita i bulloni

può essere sostituita dalla sequenza
di istruzioni più semplici

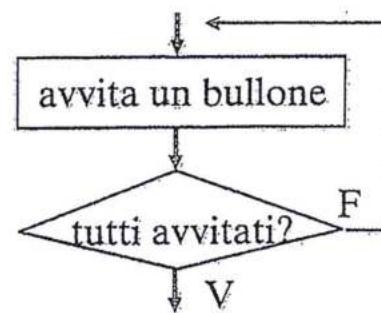


Tale sequenza si può rappresentare
con la struttura di tipo:



ma in questa forma
il ciclo è senza fine!

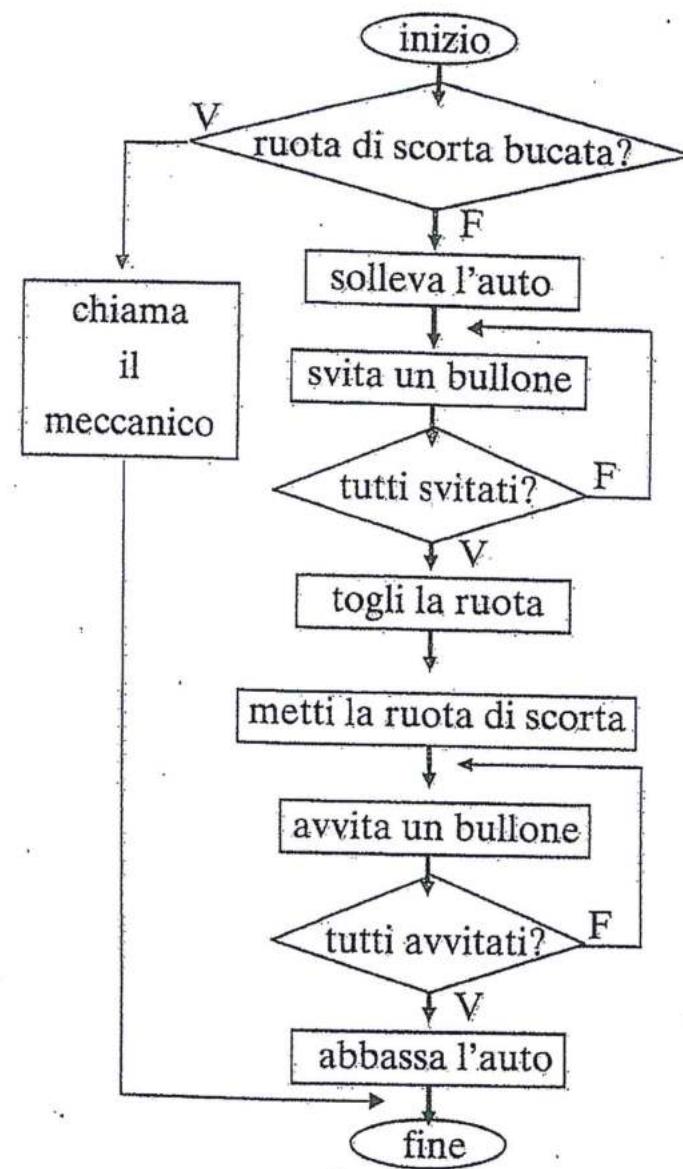
SOLUZIONE



struttura di

TIPO ITERATIVO

algoritmo per il cambio della ruota



Esempio :

preparazione di una torta

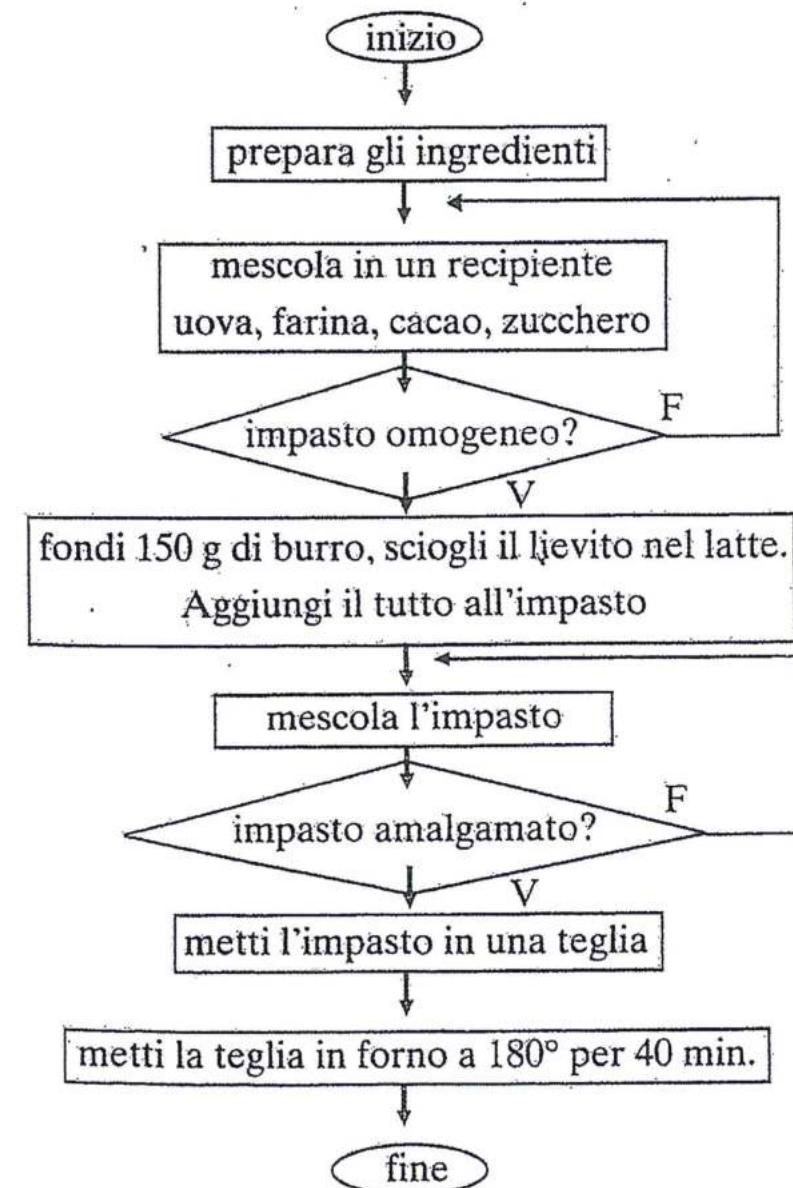
ingredienti: dati del problema

0,25 l latte, 300 g farina, 3 uova, 200 g zucchero,
100 g cacao, 160 g burro, 20 g lievito.

la ricetta: istruzioni

Mescolare farina, uova, zucchero e cacao fino ad ottenere un impasto omogeneo. Aggiungere 150 g di burro fuso e il lievito sciolto nel latte e mescolare fino ad amalgamare il tutto.

Mettere il composto in una teglia imburrata (10 g di burro) e far cuocere in forno per 40 min. alla temperatura di 180 gradi.



Definizione di Algoritmo

Un algoritmo è un procedimento per la risoluzione di una classe di problemi, costituito da un insieme finito di direttive non ambigue che specificano una sequenza finita di operazioni da eseguire su un insieme finito di dati

ETIMOLOGIA: da "Al Khuwarizmi", nome del matematico persiano del IX sec. che descrisse gli algoritmi per le operazioni aritmetiche sui numeri decimali.

Il concetto di

NON AMBIGUITÀ

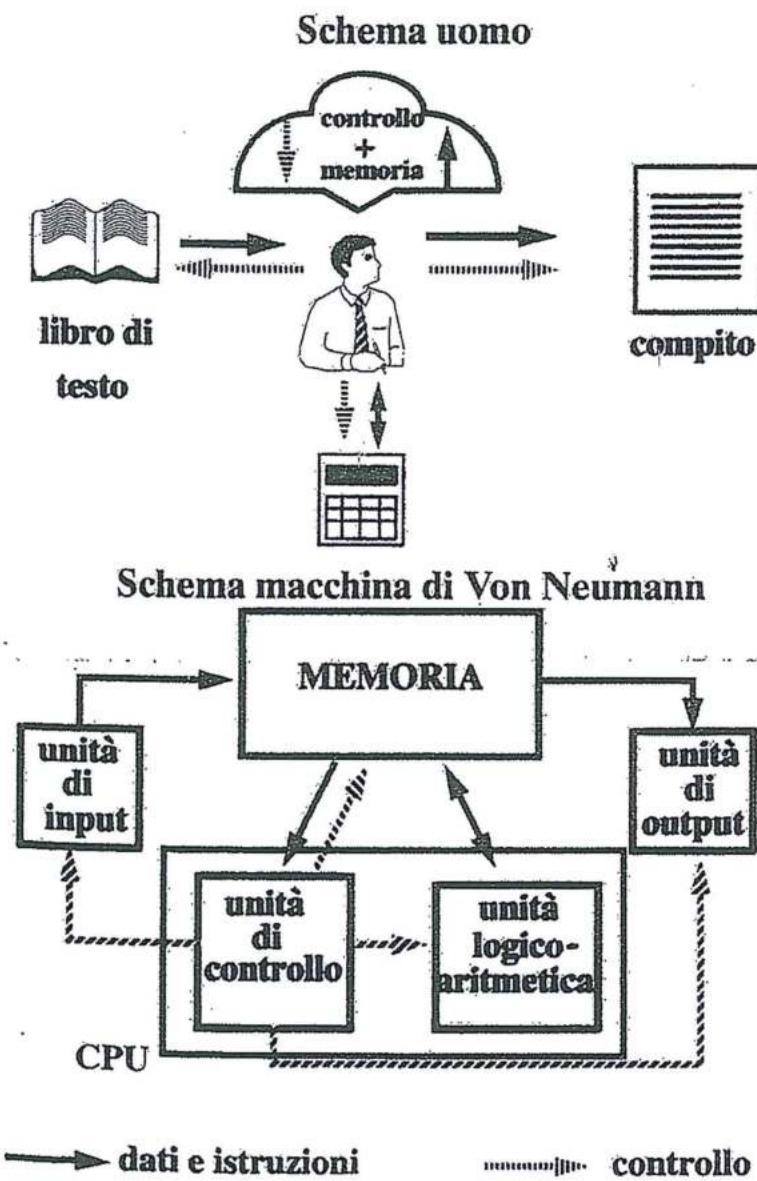
impone che siano note a priori le capacità logiche ed operative dell'esecutore dell'algoritmo

INOLTRE

un algoritmo definisce non solo il flusso delle operazioni da compiere, ma anche i dati su cui tali operazioni vanno eseguite

Un esecutore di algoritmi deve essere quindi in grado di:

- immagazzinare dati (di input intermedi e di output) e istruzioni
- effettuare operazioni aritmetiche e logiche su tali dati
- controllare il flusso delle operazioni



I componenti fondamentali di un calcolatore sono:

- Memoria
- Unità di controllo
- Unità logico-aritmetica
- Unità di input e output

La memoria

è il supporto fisico
che permette di immagazzinare
informazioni (istruzioni e dati)

La memoria è organizzata come
una lista sequenziale di

LOCAZIONI (O CELLE)

Ogni locazione è costituita
da una sequenza finita
di componenti elementari (BIT)
ciascuno dei quali
può rappresentare una cifra binaria
(0 oppure 1)

esempio: locazione a 8 bit



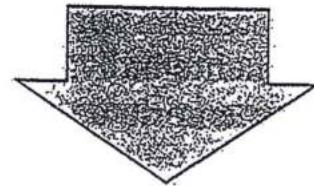
Le locazioni sono
univocamente individuate
in memoria da

INDIRIZZI

schema della memoria

1	0	0	0	1	0	1	0
1	1	0	1	0	1	1	0
0	0	0	1	1	1	0	1
0	0	1	0	0	0	1	1
1	1	0	1	1	0	0	1
1	1	1	0	1	0	0	1

0010
0011
0100
0101
0110
0111



LOCAZIONE

100

minima quantità di bit
indirizzabile

in anni recenti molti costruttori di calcolatori hanno standardizzato locazioni a 8 bit chiamate byte
(1 byte = 8 bit)

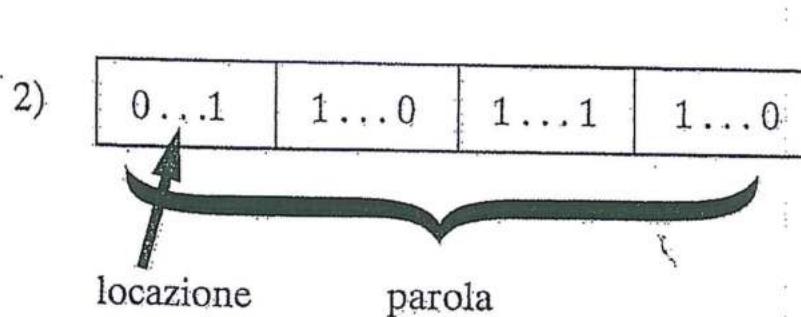
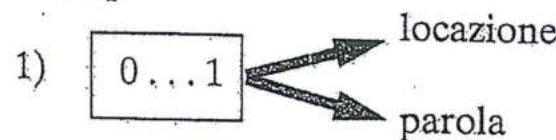
Le locazioni possono essere
raggruppate in
parole (o word)

parola

11

minimo numero di bit
necessari per memorizzare un
dato o una istruzione

Esempi:



Operazioni sulla memoria

lettura

(prelevare il contenuto di una o più locazioni di memoria)

scrittura

(definire il contenuto di una o più locazioni di memoria)

Parametri della memoria

N: capacità

(numero di parole)

W: ampiezza delle parole

(generalmente misurata in byte)

C: tempo di accesso

(tempo per prelevare un dato da una locazione)

Il costo della memoria è proporzionale a:

$$\frac{WN}{C}$$

Caratteristiche della memoria di alcuni calcolatori

calcolatore	parola	capacità (MB)
Sun Sparc	32 bit	64-512
PC 80836-486	32 bit	4-32
PC Pentium	32 bit	64-512
Intel i860	32 bit	8-16
ws HP9000	32 bit	32-400
ws IBM RS 6000	32 bit	128-1024
ws IBM Power2	64 bit	128-768
Digital 5900	32 bit	32-256
ws DEC alpha	64 bit	128-1024

1 Mbyte = 10^6 bytes

Unità di controllo:

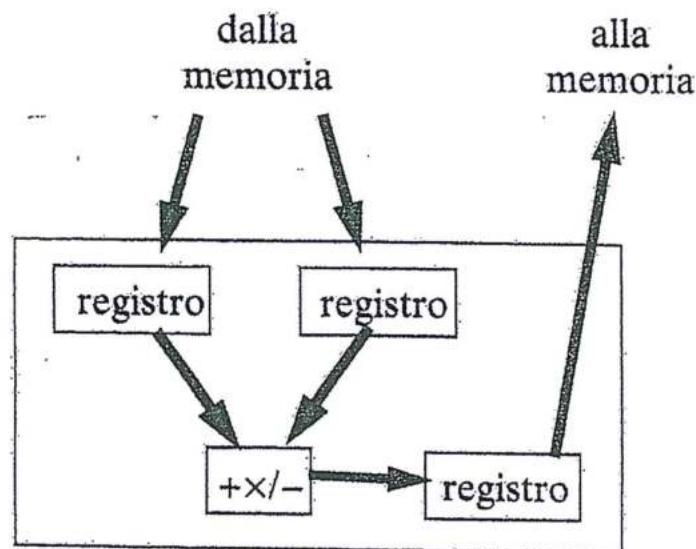
è il coordinatore di tutte le attività del calcolatore

- preleva dalla memoria le istruzioni
- le interpreta
- trasferisce i dati dalla memoria a speciali aree di memoria presenti nell'unità logico-aritmetica (registri)
- determina la successiva istruzione da eseguire

Unità logico-aritmetica

è l'esecutore delle operazioni
sui dati, sia aritmetiche,
sia di confronto

- gli operandi sono memorizzati nei *registri*



unità logico-aritmetica

unità di controllo

+

unità logico-aritmetica

=

CPU

(Central Processing Unit)

Caratteristica della CPU:

velocità operativa

(tempo di esecuzione di una istruzione)

misure della velocità operativa:

- tempo di clock (in MHz)
valori tipici sono 66, 100, 133, 200, 233, 300, 400
- MIPS
Milioni di Istruzioni Per Secondo
- MFLOPS
Milioni di operazioni FLoating-Point al Secondo

Unità di Input-Output

è l'interfaccia del calcolatore con il mondo esterno

- consente l'interazione con l'uomo (video, tastiera, stampante, mouse, ...)
- consente l'accesso alle reti telematiche (Ethernet, Internet, ...)

RAPPRESENTAZIONE DEI DATI E DELLE ISTRUZIONI

- tipo alfanumerico
- tipo intero
- tipo reale
- tipo logico
- rappresentazione delle istruzioni

Rappresentazione delle informazioni elementari:

- rappresentazione dei DATI
- rappresentazione delle ISTRUZIONI

dati + istruzioni = algoritmo.

In un linguaggio naturale, un'informazione è rappresentata da una

SUCCESSIONE DI CARATTERI
scelta da un

ALFABETO \mathcal{A}

Nella lingua italiana \mathcal{A} comprende:

{A, B, C, ..., a, b, c, ...,
0, 1, 2, 3, ..., 9,
, ;, ! \$ % ... }

L'applicazione che trasforma un'informazione in una sequenza di caratteri è detta

CODICE

In un calcolatore le informazioni sono codificate utilizzando un alfabeto \mathcal{A} che comprende solo i simboli 0 e 1

$$\mathcal{A} = \{0,1\}$$

(alfabeto binario)

Esempi di codice

carattere	codice BCD	codice EBCDIC	codice ASCII-8	codice Fieldata
0	00 1010	1111 0000	0011 0000	11 0000
1	00 0001	1111 0001	0011 0001	11 0001
2	00 0010	1111 0010	0011 0010	11 0010
3	00 0011	1111 0011	0011 0011	11 0011
4	00 0100	1111 0100	0011 0100	11 0100
5	00 0110	1111 0101	0011 0101	11 0101
6	00 0110	1111 0110	0011 0110	11 0110
7	00 0111	1111 0111	0011 0111	11 0111
8	00 1000	1111 1000	0011 1000	11 1000
9	00 1001	1111 1001	0011 1001	00 1001
A	11 0001	1100 0001	0100 0001	00 0110
B	11 0010	1100 0010	0100 0010	00 0111
C	11 0011	1100 0011	0100 0011	00 1000
D	11 0100	1100 0100	0100 0100	00 1001
E	11 0101	1100 0101	0100 0101	00 1010
F	11 0110	1100 0110	0100 1000	00 1100
G	11 0111	1100 0111	0100 0111	00 1101
H	11 1000	1100 1000	0100 1000	00 1101
I	11 1001	1100 1001	0100 1001	00 1110
J	10 0001	1101 0001	0100 1010	00 1111
K	10 0010	1101 0010	0100 1011	01 0000
L	10 0011	1101 0011	0100 1100	01 0001
.....
.....

Codice ASCII a 8 bit

0010 0000	(spazio)	0100 0000	@	0110 0000
0010 0010	!	0100 0001	A	0110 0001 a
0010 0010	"	0100 0010	B	0110 0010 b
0010 0011	#	0100 0011	C	0110 0011 c
0010 0100	\$	0100 0100	D	0110 0100 d
0010 0101	%	0100 0101	E	0110 0101 e
0010 0110	'	0100 0110	F	0110 0110 f
0010 0111	(0100 0111	G	0110 0111 g
0010 1000)	0100 1000	H	0110 1000 h
0010 1001	*	0100 1001	I	0110 1001 i
0010 1010	+	0100 1010	J	0110 1010 j
0010 1011	,	0100 1011	K	0110 1011 k
0010 1100	-	0100 1100	L	0110 1100 l
0010 1101	.	0100 1101	M	0110 1101 m
0010 1110	.	0100 1110	N	0110 1110 n
0010 1111	/	0100 1111	O	0110 1111 o
0011 0000	0	0101 0000	P	0111 0000 p
0011 0001	1	0101 0001	Q	0111 0001 q
0011 0010	2	0101 0010	R	0111 0010 r
0011 0011	3	0101 0011	S	0111 0011 s
0011 0100	4	0101 0100	T	0111 0100 t
0011 0101	5	0101 0101	U	0111 0101 u
0011 0110	6	0101 0110	V	0111 0110 v
0011 0111	7	0101 0111	W	0111 0111 w
0011 1000	8	0101 1000	X	0111 1000 x
0011 1001	9	0101 1001	Y	0111 1001 y
0011 1010	:	0101 1010	Z	0111 1010 z
0011 1011	:	0101 1011	[0111 1011 [
0011 1100	<	0101 1100	\	0111 1100 \
0011 1101	=	0101 1101]	0111 1101]
0011 1110	>	0101 1110	\$	0111 1110 ~
0011 1111	?	0101 1111	-	0111 1111 (cancella)

Per rappresentare le informazioni in un calcolatore, si può passare attraverso un codice intermedio, oppure codificarle direttamente

Informazioni

informazioni espresse
in linguaggio naturale

informazioni espresse
in linguaggio binario



Per tipo di un dato si intende l'insieme dei valori che esso può assumere

Tipi di dati:

- alfanumerico
- intero
- reale
- logico

Il tipo alfanumerico

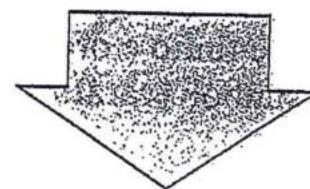
Il tipo alfanumerico indica un insieme finito e ordinato di simboli (caratteri)

Il tipo alfanumerico comprende:

- le lettere dell'alfabeto (A, B, C, ..., a, b, c, ...)
- le cifre decimali (0, 1, 2, ...)
- i simboli speciali (!, @, #, \$, ...)

Esempio 1:

Codice ASCII (8 bit)
lunghezza parola = 8 bit

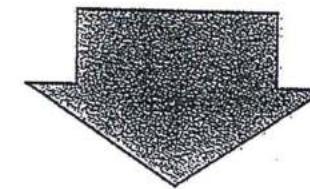


in ogni parola può essere
memorizzato un solo carattere

N	1	0	1	0	1	1	1	0
A	1	0	1	0	0	0	0	1
P	1	0	1	1	0	0	0	0
O	1	0	1	0	1	1	1	1
L	1	0	1	0	1	1	0	0
I	1	0	1	0	1	0	0	1

Esempio 2:

Codice ASCII (8 bit)
lunghezza parola = 32 bit



in ogni parola possono essere
memorizzati 4 caratteri

1 0 1 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 1 1 1

 N A P O

1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1

 L I

Esempio 3:

Codice ASCII (8 bit)
lunghezza parola = 64 bit

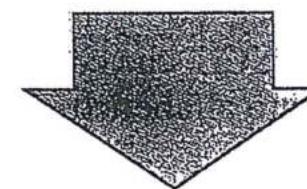


in ogni parola possono essere
memorizzati 8 caratteri

10101110010000110110000101011110101100101011001
N A P O L I

Esempio 4:

Codice ASCII (8 bit)
lunghezza parola = 64 bit



in ogni parola possono essere
memorizzati 8 caratteri

01001100101000010010000001010011010001010010100010001001000001
L A S E D I A
↑
spazio
bianco

anche lo "spazio bianco"
ha una sua codifica ASCII!

Sul tipo ALFANUMERICO sono definite le operazioni di

- Confronto (secondo l'ordinamento del codice ASCII, che conserva l'ordinamento alfabetico)

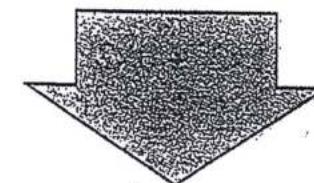
- Concatenazione: //
esempio:
mare // moto = maremoto

OSSERVAZIONE

Sull'insieme {0,1,2, ...9} non sono definite le operazioni aritmetiche

Il tipo intero

Il tipo intero indica l'insieme dei numeri interi rappresentabili nella memoria del calcolatore



In un calcolatore un dato di tipo intero è rappresentato col sistema posizionale in base $\beta=2$

Esempio 1:

Se 2 è la base del sistema:

$$\begin{aligned} 13_{10} &= 1101_2 \\ &= \\ 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 & \end{aligned}$$

IN GENERALE,

in una base $\beta > 1$:

$$\forall K \in N, \text{ se } K = c_n c_{n-1} \dots c_0,$$

allora

$$K = c_n \beta^n + \dots + c_0 \beta^0$$

con $0 \leq c_i < \beta$

Esempio 2:

In una parola di 8 bit,
se si utilizza il primo bit per
il segno, si ha:

$$2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$$

0	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

$$1101001_2 = 105_{10}$$

segno

$$2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$$

1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

$$-1000000_2 = -64_{10}$$

Esempio 3:

$L = \text{lunghezza della parola} = 8$

2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	1

$$1111111_2 = 127_{10} = 2^{8-1} - 1$$

massimo intero rappresentabile

2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1

$$-1111111_2 = -127_{10} = -(2^{8-1} - 1)$$

minimo intero rappresentabile

Esempio 4:

$L = \text{lunghezza della parola} = 16$

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$11111111111111_2 = 32767_{10} = 2^{16-1} - 1$$

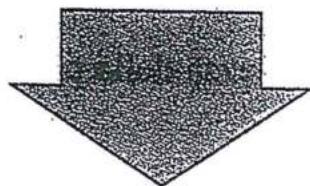
massimo intero rappresentabile

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$-11111111111111_2 = -32767_{10} = -(2^{16-1} - 1)$$

minimo intero rappresentabile

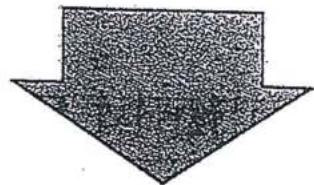
Lunghezza finita della parola



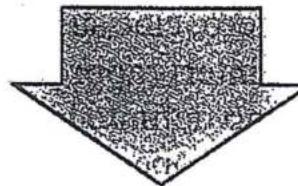
esistono

$\text{MAX} = 2^{L-1} - 1$ massimo intero
rappresentabile

$-\text{MAX} = -(2^{L-1} - 1)$ minimo intero
rappresentabile



il tipo intero specifica un
insieme finito
di numeri consecutivi



un numero intero N
è rappresentabile se

$$-\text{MAX} \leq N \leq \text{MAX}$$

$-\text{MAX}, \dots, -3, -2, -1, 0, 1, 2, 3, \dots, \text{MAX}$



Sul tipo intero, cioè sull'insieme dei numeri interi rappresentabili, sono definite le seguenti operazioni:

- Addizione
- Sottrazione
- Moltiplicazione
- Divisione

Esempio 5:

Se MAX = 100,

$$60 + 50 = ?$$

Risultato indefinito

(non rappresentabile)

Se si indicano con

$+$

$-$

$*$

$/$

le quattro operazioni sul dato di tipo intero, e se gli operandi X e Y sono rappresentabili, cioè:

$|X| \leq MAX$

$|Y| \leq MAX$

allora

$$X \# Y = \begin{cases} X \# Y & \text{se } |X \# Y| \leq MAX \\ \text{indefinito} & \text{se } |X \# Y| > MAX \end{cases}$$

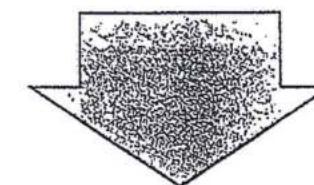
Esempio 6:

Se $MAX = 100$,

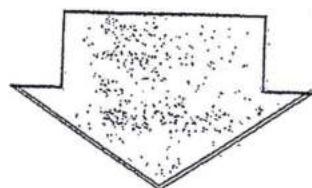
$$\underbrace{(60 + 50) + (-40)}_{\text{indefinito}} = ?$$

mentre

$$60 + (50 - 40) = 60 + 10 = 70$$



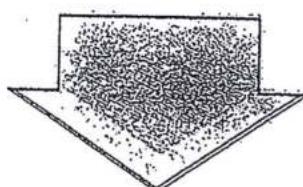
alcune proprietà
dell'aritmetica tradizionale
non valgono!

Esempio 7: $\beta=2, L=32$ 

$$MAX = 2^{31} - 1 \approx 10^9$$

Se $N = 13$,

$$N! = 6227020800 > MAX$$



$N!$ non è rappresentabile per
 $N > 12$

La situazione eccezionale

$$|X \# Y| > MAX$$

è detta

OVERFLOW

Se $L=32$, il calcolo di $N!$ provoca
overflow per $N > 12$

Il tipo reale

Esempio 1:

$$\begin{array}{ll} \text{velocità della luce} & \approx 3000000000 \text{ cm/s} \\ \text{raggio atomico} & \approx 0.0000001 \text{ cm} \end{array}$$

rappresentazione POCO EFFICIENTE



rappresentazione FLOATING-POINT
(floating-point = punto che si sposta)

$$\begin{array}{ll} \text{velocità della luce} & \approx 0.30 \times 10^{11} \text{ cm/s} \\ \text{raggio atomico} & \approx 0.1 \times 10^{-7} \text{ cm} \end{array}$$

rappresentazione PIÙ EFFICIENTE

Ogni numero reale $x \in \mathcal{R}$ può essere rappresentato in forma *floating-point*:

$$x = f \times \beta^e$$

con

$$f \in \mathcal{R}$$

$$\beta \in N \quad \beta > 1 \quad (\text{base})$$

$$e \in Z \quad (\text{esponente})$$

ESEMPI

$$12345 = 0.0012345 \times 10^7$$

$$\pi = 3.141592 \dots = 0.3141592 \dots \times 10^1$$

$$e = 2.718281 \dots = 27.18281 \dots \times 10^{-1}$$

$\mathcal{R} \equiv$ insieme dei numeri reali

$Z \equiv$ insieme dei numeri interi

$N \equiv$ insieme dei numeri naturali

Esempio 2:

$$\begin{aligned}x &= 35.16904 \dots = \\&= 3.516904 \dots \times 10^1 = \\&= 0.3516904 \dots \times 10^2 = \dots\end{aligned}$$

La rappresentazione floating-point
non è unica:

$$x = f \times \beta^e = (f \times \beta^{e-k}) \beta^k, \forall k \in \mathbb{Z}$$

Esempio 3:

$$\beta = 10$$

$$\begin{aligned}12.36 &= 0.1236 \times 10^2 \equiv (1236,2) \\0.0025 &= 0.25 \times 10^{-2} \equiv (25,-2) \\-1.23 &= -0.123 \times 10^1 \equiv (-123,1)\end{aligned}$$

la prima cifra di f dopo il punto
è **diversa da 0**, cioè

$$\frac{1}{\beta} \leq |f| < 1$$



La rappresentazione è unica

Posto $f = 0.m$
e fissato β , il numero x è
univocamente determinato
dalla coppia:

$$(m, e) \quad m = \text{mantissa}$$

$$e = \text{esponente}$$

OSSERVAZIONE:

in questa rappresentazione si assume
che m comprende il segno di f

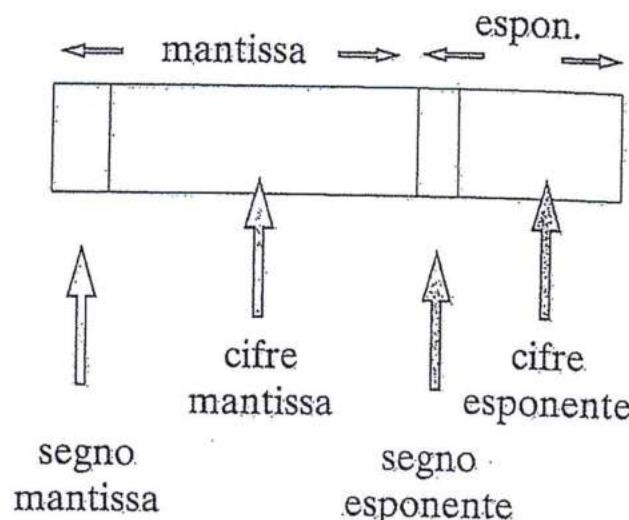
con

$$\forall x \in \mathcal{R} - \{0\}$$

$$x = 0.m \times \beta^e \equiv (m, e)$$

$$\beta^{-1} \leq |0.m| < 1, \quad e \in \mathbb{Z}$$

RAPPRESENTAZIONE FLOATING-POINT NORMALIZZATA

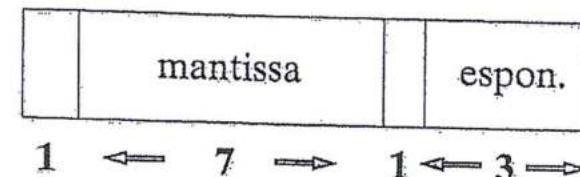
PAROLA

4 informazioni da rappresentare:

- segno di m
- cifre di m
- segno di e
- cifre di e

Esempio 4:

$$\beta = 2 \quad \text{lungh. parola } L = 12$$



numero

$$(11.75)_{10} = (1011.11)_2$$

$$= 0.101111 \times 2^{100}$$

rappresentazione

0	1011110	0	100
---	---------	---	-----

$$(-6.625)_{10} = (-110.101)_2$$

$$= -0.110101 \times 2^{11}$$

1	1101110	0	011
---	---------	---	-----

$$(5.59375)_{10} = (101.10011)_2$$

$$= 0.10110011 \times 2^{11}$$

?
numero cifre
mantissa > 7

$$(128)_{10} = (10000000)_2$$

$$= 0.1 \times 2^{1000}$$

?
numero cifre
esponente > 3

in un calcolatore lo spazio disponibile per memorizzare m ed e dipende dalla lunghezza della parola



limitazioni

nella rappresentazione dei numeri reali

prima limitazione

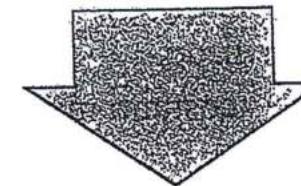
precisione finita

=
numero finito t di cifre
per rappresentare la mantissa

$$m = (d_1 d_2 \dots d_t)_{\beta}$$

$0 < d_1 \leq (\beta - 1)$ (rappr. normalizzata)

$0 \leq d_i \leq \beta - 1, \quad i = 2, \dots, t$

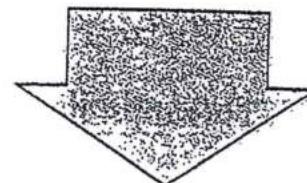


tutti i numeri reali che hanno
più di t cifre
non possono essere
rappresentati esattamente

seconda limitazione

range limitato

numero finito di cifre
per rappresentare l'esponente



$$E_{min} \leq e \leq E_{max}$$

E_{min}, E_{max} costanti macchina

Esempio 5:

$$\beta = 10, \quad t = 3, \quad E_{min} = -9, \quad E_{max} = 9$$



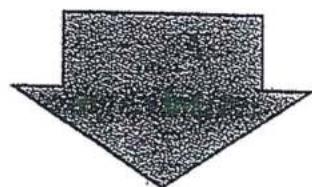
$$r_{max} = 0.999 \times 10^9$$

massimo numero reale
positivo rappresentabile

$$r_{min} = 0.100 \times 10^{-9}$$

minimo numero reale
positivo rappresentabile

$$E_{min} \leq e \leq E_{max}$$



ESISTONO

$$r_{max} = 0.dd \dots d \times \beta^{E_{max}} \quad (d = \beta - 1)$$

max numero reale positivo rappresentabile

$$r_{min} = 0.10 \dots 0 \times \beta^{E_{min}}$$

min numero reale positivo rappresentabile

L'insieme finito $F \subset \mathcal{R}$ dei numeri reali rappresentabili esattamente in un calcolatore è detto *insieme dei numeri macchina*

Il **tipo di dato reale** è costituito dall'insieme dei numeri macchina

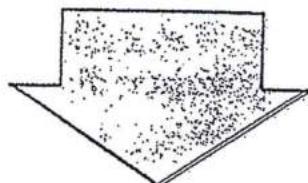
2/11/2000

$F =$ sistema aritmetico
floating-point
a precisione finita

Parametri che caratterizzano F :

$\beta =$ base di numerazione
 $t =$ precisione

$E_{min} =$ minimo esponente
 $E_{max} =$ massimo esponente



$$F \equiv (\beta, t, E_{min}, E_{max})$$

$x \in F$

$x \neq 0$

$$\begin{aligned} x &= \pm 0.(d_1 d_2 \dots d_t) \times \beta^e = \\ &= \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_t}{\beta^t} \right) \times \beta^e \end{aligned}$$

con

$$0 < d_1 \leq \beta - 1$$

$$0 \leq d_i \leq \beta - 1 \quad i = 2, \dots, t$$

$$E_{min} \leq e \leq E_{max}$$

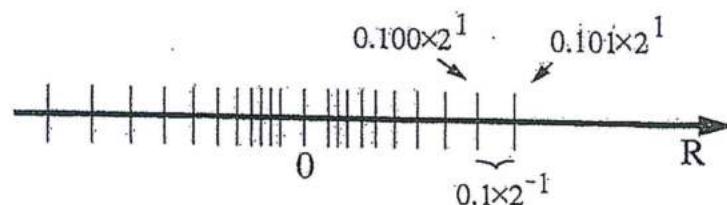
$x = 0$ ha una rappresentazione speciale, per es.:

$$x = 0.\underbrace{00 \dots 0}_t \times \beta^0$$

Esempio 6:

$$\beta = 2, \quad t = 3, \quad E_{\min} = -1, \quad E_{\max} = 2$$

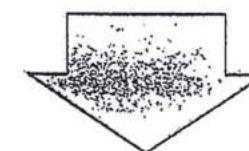
$$F = \left\{ \begin{array}{l} 0 \pm 0.100 \times 2^{-1} \pm 0.100 \times 2^0 \pm 0.100 \times 2^1 \pm 0.100 \times 2^2 \\ \pm 0.101 \times 2^{-1} \pm 0.101 \times 2^0 \pm 0.101 \times 2^1 \pm 0.101 \times 2^2 \\ \pm 0.110 \times 2^{-1} \pm 0.110 \times 2^0 \pm 0.110 \times 2^1 \pm 0.110 \times 2^2 \\ \pm 0.111 \times 2^{-1} \pm 0.111 \times 2^0 \pm 0.111 \times 2^1 \pm 0.111 \times 2^2 \end{array} \right\}$$



I numeri appartenenti a F non sono tutti uniformemente spaziati; essi sono tali solo tra successive potenze di β

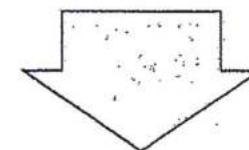
Quanti sono gli elementi di F ?

1) l'esponente di un generico elemento di F è un numero intero compreso tra E_{\min} ed E_{\max}



2) per ogni fissato esponente sono possibili

- $\beta-1$ combinazioni di $\beta-1$ cifre su 1 posto (il primo)
- β^{t-1} combinazioni di β cifre su $t-1$ posti



aggiungendo lo 0 e tenendo conto del segno, F contiene

$$2(\beta-1)\beta^{t-1}(E_{\max} - E_{\min} + 1) + 1$$

numeri

- F ha un elemento r_{max} di massimo modulo ed un elemento r_{min} di minimo modulo
- F contiene lo 0
- F è costituito da un numero finito di elementi

PROBLEMA

$$x = 0.m \times \beta^e$$

$$x \in \mathcal{R} \quad \text{MA} \quad x \notin F$$

CIOÈ

• $e > E_{max}$ oppure $e < E_{min}$

oppure

• $E_{min} \leq e \leq E_{max}$ ma # cifre di $m > t$

NON
RAPP.

NON
RAPP.

È possibile rappresentare x
mediante un numero macchina?

Esempio 7:

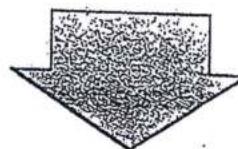
$$\beta = 10, \quad t = 2, \quad E_{min} = -1, \quad E_{max} = 2$$

$$y = 1200 = 0.12 \times 10^4 = 0.m_1 \times 10^{e_1}$$

$$z = 0.00032 = 0.32 \times 10^{-3} = 0.m_2 \times 10^{e_2}$$

$$e_1 > E_{max}$$

$$e_2 < E_{min}$$



y e z non sono numeri macchina e

**non sono rappresentabili
mediante numeri macchina**

$$\beta = 10, \quad t = 2, \quad E_{min} = -1, \quad E_{max} = 2$$

Insieme dei numeri rappresentabili
mediante numeri macchina:

$$I = (-10^2, -10^{-2}] \cup \{0\} \cup [10^{-2}, 10^2]$$

\uparrow \uparrow
 -0.10×10^{-1} 0.10×10^{-1}

In generale

$$I = (-\beta^{E_{max}}, -\beta^{E_{min}-1}] \cup \{0\} \cup [\beta^{E_{min}-1}, \beta^{E_{max}})$$

\uparrow \uparrow
 $-0.10 \dots 0 \times \beta^{E_{min}}$ $0.10 \dots 0 \times \beta^{E_{min}}$
 t t

INSIEME DI RAPPRESENTABILITÀ

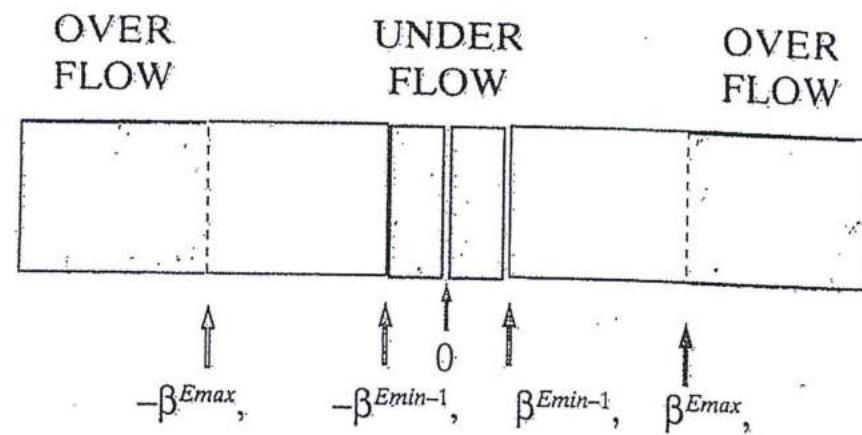
Situazioni eccezionali

Se $e > E_{max}$ si dice che si è verificato un

OVERFLOW

Se $e < E_{min}$ si dice che si è verificato un

UNDERFLOW



Soltanamente in un calcolatore

- Se $|x| \geq \beta^{E_{max}}$ (**OVERFLOW**), si ha un valore indefinito nella rappresentazione floating-point di x
- Se $|x| < \beta^{E_{min}}$ (**UNDERFLOW**), in genere si associa a x il numero 0

Esempio 8:

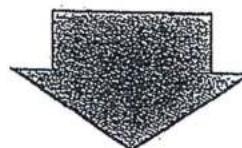
$$\beta = 10, \quad t = 2, \quad E_{\min} = -1, \quad E_{\max} = 2$$

$$x = 1.25 = 0.125 \times 10^1 = f \times 10^e$$

$$E_{\min} \leq e \leq E_{\max}$$

MA

cifre di $f > t$



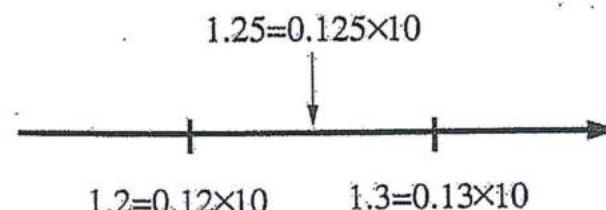
x non è un numero macchina
(non è rappresentabile esattamente in F)
ma può essere rappresentato
con un numero macchina

Idea naturale:

sostituire a x il numero macchina
“più vicino”

$$fl(1.25) = 1.2 \quad \text{oppure} \quad fl(1.25) = 1.3$$

$$fl(1.25) \neq 1.25$$



$fl(x)$ = numero macchina che rappresenta x
(rappresentazione floating-point di x in F)

Metodi utilizzati per passare
da un numero reale x
al numero macchina $fl(x)$:

- TRONCAMENTO
- ARROTONDAMENTO

Esempio 9:

Troncamento ($\beta = 10, t = 3$)

$$fl(12.3) = 0.123 \times 10^2 \quad x = fl(x)$$

$$fl(12.34) = 0.123 \times 10^2 \quad x > fl(x)$$

$$fl(12.39) = 0.123 \times 10^2 \quad x > fl(x)$$

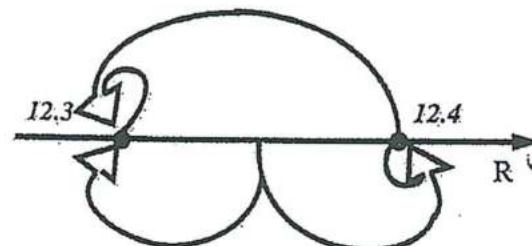
Arrotondamento ($\beta = 10, t = 3$)

$$fl(12.3) = 0.123 \times 10^2 \quad x = fl(x)$$

$$fl(12.34) = 0.123 \times 10^2 \quad x > fl(x)$$

$$fl(12.39) = 0.124 \times 10^2 \quad x < fl(x)$$

troncamento



arrotondamento

$$x = f \times \beta^e \quad (x \neq 0)$$

$$f = \pm 0.d_1 d_2 \dots d_t d_t + 1 \dots, \frac{1}{\beta} \leq |f| < 1$$

- TRONCAMENTO:

$$fl(x) = f' \times \beta^e, \quad f' = \pm 0.d_1 d_2 \dots d_t$$

- ARROTONDAMENTO:

$$fl(x) = f'' \times \beta^e,$$

$$f'' = \begin{cases} \pm 0.d_1 d_2 \dots d_t & d_{t+1} < \frac{\beta}{2} \\ \pm 0.d_1 d_2 \dots d_t + \beta^{-t} & d_{t+1} \geq \frac{\beta}{2} \end{cases}$$

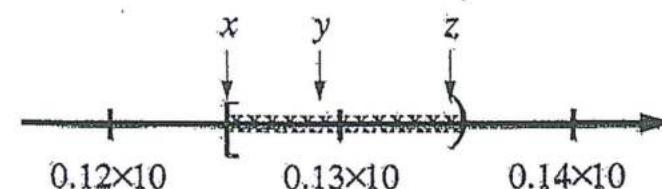
Soltamente si utilizza l'arrotondamento.

Esempio 10:

$$\beta = 10, \quad t = 2, \quad E_{min} = -1, \quad E_{max} = 2$$

arrotondamento

$$x = 1.25, \quad y = 1.29, \quad z = 1.34$$



$$fl(x) = fl(y) = fl(z) = 0.13 \times 10 = 1.3$$

1.3 rappresenta l'intervallo [1.25, 1.35)

In generale

un numero macchina rappresenta
un intervallo di numeri reali

Sull'insieme dei numeri reali rappresentabili,
sono definite le operazioni
aritmetiche floating-point (f.p.)

- addizione f.p. \oplus
- sottrazione f.p. \ominus
- moltiplicazione f.p. \otimes
- divisione f.p. \oslash

Esempio 11:

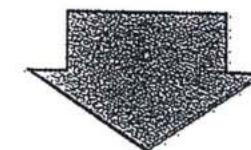
$$\beta = 10, \quad t = 3, \quad E_{min} = -9, \quad E_{max} = 9$$

$$x = 0.257 \times 10^8$$

$$y = 0.332 \times 10^{-4}$$

$$x \oslash y = 0.257 \times 10^8 \oslash 0.332 \times 10^{-4} = 0.774 \times 10^{12}$$

$$12 > 9 = E_{max}$$



$$x \oslash y$$

non rappresentabile

il risultato è indefinito

$$r = x \oplus y$$

il risultato è definito solo se:

- gli operandi x e y sono rappresentabili
- il risultato r è rappresentabile

$$(\# = +, -, \times, /)$$

IN CONCLUSIONE

un sistema aritmetico
floating-point a precisione finita
è costituito da:

- un insieme finito di numeri reali
- un criterio di rappresentazione di tali numeri (rappresentazione floating-point a precisione finita)
- un insieme di operazioni definite su tali numeri (operazioni aritmetiche floating-point)

sistema aritmetico standard IEEE

Sistema aritmetico floating-point a precisione finita, definito nel 1982, con l'obiettivo di rendere uniformi le prestazioni dei programmi su tutti i calcolatori

singola precisione

$$\beta = 2, \quad t = 23, \quad E_{\min} = -127, \quad E_{\max} = 128$$

doppia precisione

$$\beta = 2, \quad t = 52, \quad E_{\min} = -1023, \quad E_{\max} = 1024$$

Nell'aritmetica standard IEEE i numeri rappresentabili hanno esponente e strettamente compreso tra E_{\min} e E_{\max}

$$E_{\min} < e < E_{\max}$$

Inoltre i numeri rappresentabili hanno mantissa m che utilizza la rappresentazione del bit隐式, in cui il bit iniziale non è esplicitamente rappresentato in quanto è sempre uguale a 1 se il numero è normalizzato



se t bit sono usati per la mantissa,
la precisione è $t+1$

Nell'aritmetica standard IEEE
gli esponenti E_{min} e E_{max} sono
utilizzati per gestire 'situazioni eccezionali'

numeri non normalizzati $e = E_{min}$ $m \neq 0$
(es. $0.000321 \times 10^{E_{min}}$)

forme indeterminate $e = E_{max}$ $m = 0$
(es. $0/0$)

infinito con segno $e = E_{max}$ $m \neq 0$

zero $e = E_{min}$ $m = 0$

L'infinito e le forme indeterminate sono
rappresentati convenzionalmente con i simboli
Inf e **NaN** (Not a Number)

23/10

Il tipo logico

Il tipo logico indica
l'insieme costituito da:

{vero, falso}

In memoria ciascuno dei due valori logici può essere rappresentato da un solo bit

In generale,
questa informazione è contenuta in
una intera locazione di memoria,
non essendo di solito possibile
accedere all'informazione contenuta
nel singolo bit

Esempio 1:

vero

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

falso

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Sul tipo logico sono definite
le seguenti operazioni:

- NEGAZIONE (not, \neg)
- CONGIUNZIONE (and, \wedge)
- DISGIUNZIONE (or, \vee)

NEGAZIONE

Esempio 2:

p: $12 > 2$ (vero)

not p: $12 \leq 2$ (falso)

q: Roma è in Francia (falso)

not q: Roma non è in Francia (vero)

CONGIUNZIONE

Esempio 3:

p: $2 < 3$ (vero)

q: $5 < 3$ (falso)

p and q: $2 < 3$ e $5 < 3$ (falso)

p: Napoli è in Italia (vero)

q: la Terra ruota (vero)

p and q: Napoli è in Italia e la Terra ruota (vero)

DISGIUNZIONE

Esempio 4:

p: $2 < 3$ (vero)

q: $5 < 3$ (falso)

p or q: $2 < 3$ oppure $5 < 3$ (vero).

p: 1 è una cifra binaria (vero)

q: 1 è una cifra decimale (vero)

p or q: 1 è una cifra binaria oppure 1 è una cifra decimale (vero)

p: $2 + 1 = 5$ (falso)

q: $3 - 2 = 4$ (falso)

p or q: $2 + 1 = 5$ oppure $3 - 2 = 4$ (falso)

Le operazioni sui dati logici si esprimono mediante le tavole della verità.

negazione

<i>p</i>	not <i>p</i>
vero	falso
falso	vero

congiunzione e disgiunzione

<i>p</i>	<i>q</i>	<i>p and q</i>	<i>p or q</i>
vero	vero	vero	vero
vero	falso	falso	vero
falso	vero	falso	vero
falso	falso	falso	falso

In un calcolatore come è effettuata la somma di due interi?

Esempio: $18 + 7 = 25$

Nel sistema di numerazione binario i due interi hanno rappresentazione:

$$18_{10} = 10010_2$$

$$7_{10} = 00111_2$$

La somma è eseguita bit a bit secondo lo schema:

<i>x</i>	<i>y</i>	<i>sum</i>
0	0	00
0	1	01
1	0	01
1	1	10

Analogamente alle operazioni nel sistema decimale, il bit più a sinistra di sum è il riporto per l'addizione della successiva coppia di bit.

La somma è

$$\begin{array}{r} 10010 + \\ 00111 = \\ \hline 11001_2 \quad (25_{10}) \end{array}$$

Le operazioni logiche elementari (and, or, not) hanno 1 o 2 bit di input e 1 bit di output

Rappresentazione convenzionale:

$1 \equiv$ vero, $0 \equiv$ falso



$p = x \text{ and } y$



$p = x \text{ or } y$



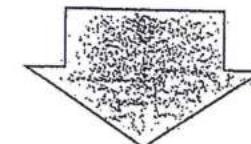
$p = \text{not } y$

sum è composto da due bit (r, s):

$s =$ somma

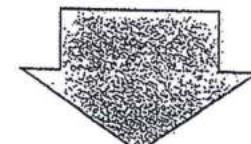
$r =$ riporto per la somma successiva

Il bit più a sinistra di sum è uguale a 1 se x e y sono entrambi uguali a 1



$r = x \text{ and } y$

Il bit più a destra di sum è uguale a 1 se x e y , ma non entrambi, sono uguali a 1 (or esclusivo)



$s = (x \text{ and } (\text{not } y)) \text{ or } ((\text{not } x) \text{ and } y)$

L'operazione di
or esclusivo (XOR)
utilizzata per il calcolo di s
ha la seguente tavola della verità:

XOR	0	1
0	0	1
1	1	0

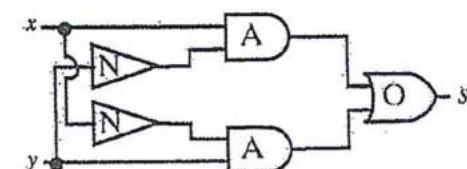
ed è indicata convenzionalmente con

$$\begin{array}{c} x \\ \oplus \\ y \end{array} \rightarrow p \quad p = x \text{ xor } y$$

In sintesi si ha che il bit più a sinistra
si ottiene con:

$$\begin{array}{c} x \\ \oplus \\ y \end{array} \rightarrow r$$

mentre quello più a destra si ottiene
con:



$$\begin{array}{c} x \\ \oplus \\ y \end{array} \rightarrow s$$

SEMIADDIZIONATORE

Un addizionatore di stringhe bit, per il calcolo di ogni bit della somma,

- richiede 3 bit di input (2 addendi x e y e 1 riporto r_{prec})
- produce 2 bit di output (1 somma s 1 riporto r)

secondo la seguente tavola della verità

x	y	r_{prec}	r	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

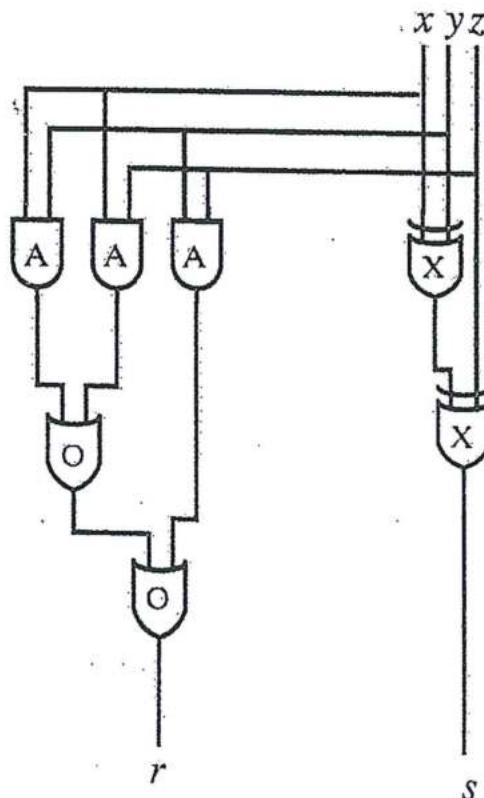
Dalla precedente tavola
si nota che:

- il bit della somma s è il risultato di due operazioni di or esclusivo.

$$s = x \text{ xor } y \text{ xor } z$$

- il bit del riporto r è 1 se almeno 2 bit valgono 1, cioè:

$$r = (x \text{ and } y) \text{ or } (x \text{ and } z) \text{ or } (y \text{ and } z)$$



ADDIZIONATORE

La rappresentazione delle istruzioni

Rappresentazione delle informazioni elementari:

- rappresentazione dei DATI
- rappresentazione delle ISTRUZIONI

istruzioni + dati = algoritmo

Linguaggio direttamente comprensibile
da un calcolatore

=
LINGUAGGIO MACCHINA

L'alfabeto del linguaggio macchina
è costituito da due soli simboli:

0,1

Un algoritmo descritto in linguaggio
macchina prende il nome di

**PROGRAMMA
IN LINGUAGGIO MACCHINA**

Esempio

esecuzione dell'operazione

$$2 \times 3 + 7$$

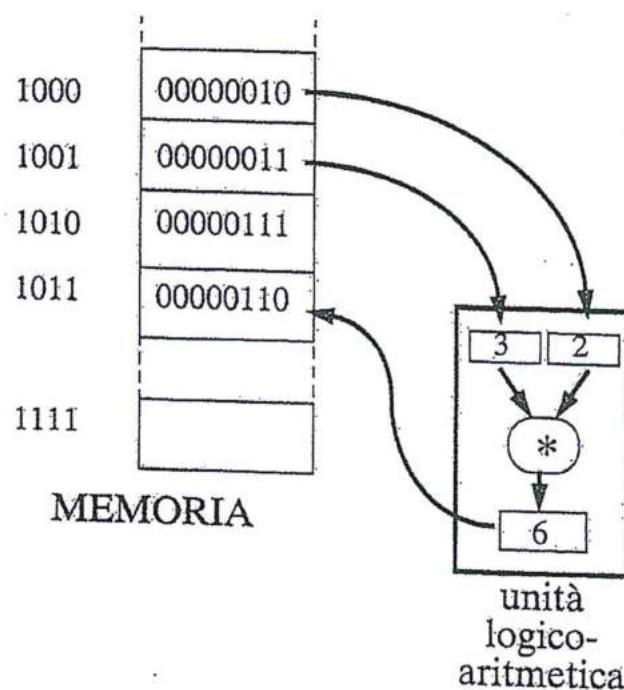
memorizzazione dei dati

indirizzi		
1000	0 0 0 0 0 1 0	2
1001	0 0 0 0 0 1 1	3
1010	0 0 0 0 1 1 1	7
	⋮	
1111		



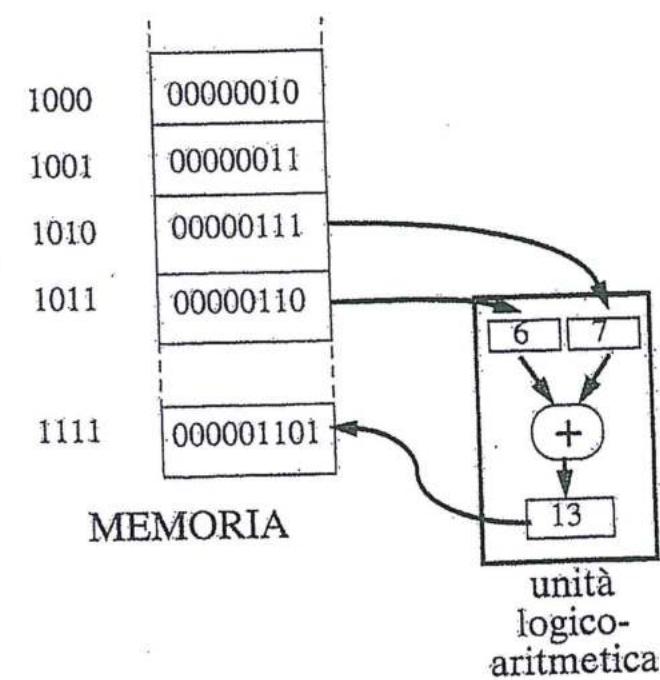
I FASE:

calcolo di 2×3 e memorizzazione del risultato (6) nella locazione di memoria con indirizzo 1011



II FASE:

calcolo di $6+7$ e memorizzazione del risultato (13) nella locazione di memoria con indirizzo 1111



le istruzioni che specificano le varie fasi del calcolo devono quindi contenere informazioni relative a

- quale operazione eseguire
- le locazioni di memoria che contengono gli operandi

Esempio 1: istruzione a 3 indirizzi

CODICE OPERATIVO	INDIRIZZO OPERANDO 1	INDIRIZZO OPERANDO 2	INDIRIZZO RISULTATO
------------------	----------------------	----------------------	---------------------

Esecuzione di $2 \times 3 + 7$

$$2 \times 3 = 6$$

moltiplicaz. indirizzo di 2 indirizzo di 3 indirizzo di 6

0111	1000	1001	1011
------	------	------	------

$$6 + 7 = 13$$

addizione indirizzo di 6 indirizzo di 7 indirizzo di 13

1000	1011	1010	1111
------	------	------	------

Esempio 2: istruzione a 2 indirizzi

CODICE OPERATIVO	INDIRIZZO OPERANDO 1	INDIRIZZO OPERANDO 2
------------------	----------------------	----------------------

Il risultato dell'operazione è memorizzato nella locazione di memoria dell'operando 1

Esecuzione di $2 \times 3 + 7$

$$2 \times 3 = 6$$

moltiplicaz.	indirizzo di 2	indirizzo di 3
0111	1000	1001

$$6 + 7 = 13$$

addizione	indirizzo di 6	indirizzo di 7
1000	1000	1010
0001		

Il risultato dell'operazione (13) è memorizzato nella locazione di indirizzo 1000

Esempio 3: istruzione a 1 indirizzo

CODICE OPERATIVO	INDIRIZZO OPERANDO
------------------	--------------------

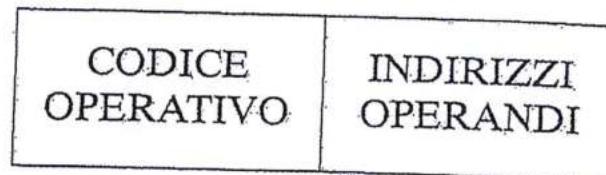
Il primo operando è in una locazione di memoria prestabilita detta *registro*

Anche il risultato è memorizzato nel registro

Esecuzione di $2 \times 3 + 7$

CO10	1110 1000	indirizzo di 2
copia nel registro	0011 1001	indirizzo di 3
moltiplica conten. registro	0001 1010	indirizzo di 7
somma conten. registro	1000 1111	indirizzo di 13
copia dal registro		

L'istruzione in linguaggio macchina ha un formato fissato:



Codice operativo:

specifica quale operazione deve essere eseguita

Indirizzi degli operandi:

indirizzi delle locazioni di memoria contenenti gli operandi

Esempio 4:

somma dei primi 10 numeri interi in un calcolatore con istruzioni ad 1 indirizzo

TABELLA CODICI OPERATIVI

COD. OP.	SIGNIFICATO
1110	copia nella loc. di indirizzo specific. il conten. del registro (copia in)
1100	copia nel registro il conten. della loc. di memoria di indir. specific. (copia da)
1000	addiziona il conten. del registro al conten. della loc. di mem. di indir. specificato
0001	azzerza il conten. della loc. di mem. di indirizzo specific.
0011	incrementa di 1 il conten. della loc. di mem. di indirizzo specific.
1010	confronta il contenuto del registro con quello della loc. di mem. specific.; se il primo è maggiore del secondo salta l'istruz. che segue
0101	vai all'istruzione contenuta nella loc. di indirizzo specific.
1111	termina l'esecuzione

Programma in linguaggio macchina
per la somma dei primi 10 numeri interi

MEMORIA

azzerà
copia in
azzerà
incrementa
copia in
copia da
somma
copia in
copia da
incrementa
copia in
copia da
confronta
vai a
stop

REGISTRO	
0001	000000
1110	010001
0001	000000
0011	000000
1110	010010
1100	010001
1000	010010
1110	010001
1100	010010
0011	000000
1110	010010
1100	010010
1010	010000
0101	000000
1111	000000

indirizzo
000000
000001
000010
000011
000100
000101
000110
000111
001000
001001
001010
001011
001100
001101
001110
001111

Limiti della programmazione in linguaggio macchina

- necessità di tener conto degli indirizzi numerici
- necessità di gestire i tipi di dati in modo diretto
- scarsa potenza del linguaggio
- difficile leggibilità del programma
- “portabilità” del programma solo su macchine simili

Esempio 1 bis: istruzione a 3 indirizziEsecuzione di $2 \times 3 + 7$

$$2 \times 3 = 6$$

moltiplicaz. indirizzo di 2 indirizzo di 3 indirizzo di 6

0111	1000	1001	1011
MPY	A	B	P

$$6 + 7 = 13$$

addizione indirizzo di 6 indirizzo di 7 indirizzo di 13

1000	1011	1010	1111
ADD	P	C	S

In modo più semplice:

MPY A B P**ADD P C S****Esempio 2 bis:** istruzione a 2 indirizzi

$$\begin{array}{ll} \text{MPY A B} & 2 \times 3 = 6 \\ \text{ADD A C} & 6 + 7 = 13 \end{array}$$

Esempio 3 bis: istruzione ad 1 indirizzo

$$\begin{array}{ll} \text{LOAD A} & \\ \text{MPY B} & 2 \times 3 = 6 \\ \text{ADD C} & 6 + 7 = 13 \\ \text{STORE S} & \end{array}$$

LOAD A: carica nel registro il contenuto della locazione di memoria associata ad A

STORE S: copia il contenuto del registro nella locazione di memoria associata ad R

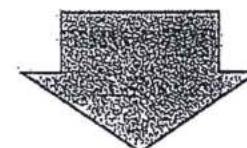
Un modo più semplice
per scrivere un'istruzione:

- un nome mnemonico al posto del codice operativo
- un nome simbolico al posto dell'indirizzo dell'operando

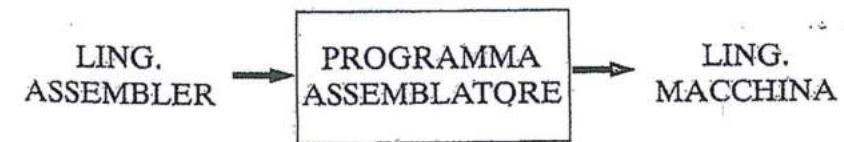


LINGUAGGIO ASSEMBLER

Affinché una sequenza di istruzioni scritta in linguaggio assembler sia eseguita da un calcolatore è necessaria la presenza di un programma, scritto in linguaggio macchina, che traduca il linguaggio assembler in linguaggio macchina



PROGRAMMA ASSEMBLATORE



calcolatore + programma assemblatore
=
esempio di *macchina virtuale*

MACCHINA FISICA:
calcolatore che mette a disposizione
solo le potenzialità del suo *hardware* e del suo
linguaggio macchina

MACCHINA VIRTUALE:
macchina fisica
+
insieme di programmi (*software*)
che sfruttano le potenzialità
della macchina fisica

macchina fisica
+
programma per la somma
di un insieme di numeri
= *macchina virtuale* che sa calcolare
la somma di un insieme di numeri

macchina fisica
+
programma per la ricerca di
un nome in un elenco
= *macchina virtuale* che sa ricercare
un nome in un elenco

Linguaggio di programmazione ad alto livello

- più potente del linguaggio macchina
- più vicino al linguaggio naturale
- formalmente definito

FINE 24/10

Linguaggi più diffusi:

- Fortran
- C
- C++
- Pascal
- Basic
-
- Oracle
- Progress
-
- Matlab
-
- Java
-

L'uso di un linguaggio di programmazione ad alto livello è reso possibile dai

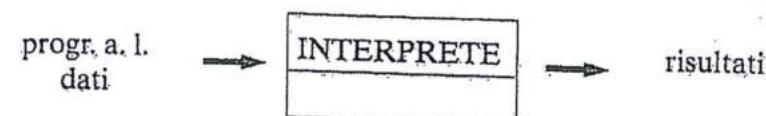
PROGRAMMI TRADUTTORI

che traducono un programma scritto in un linguaggio ad alto livello in un programma scritto in linguaggio macchina.

Due tipi di programmi traduttori

- COMPILATORI
- INTERPRETI

INTERPRETE

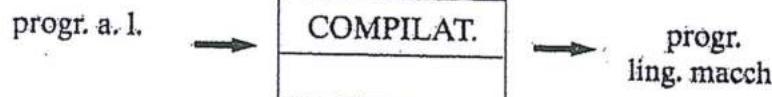


L'interprete

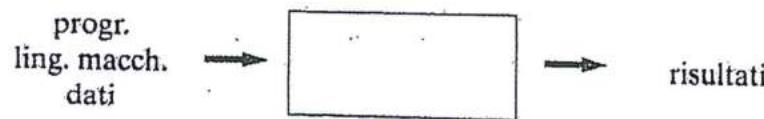
- prende *in input* una istruzione di un programma in linguaggio a. l. ed i dati ad essa relativi
- fornisce *in output* direttamente il risultato dell'esecuzione dell'istruzione

COMPILATORE

FASE 1



FASE 2



FASE 1

Il compilatore:

- prende *in input* un programma in linguaggio a. l.
- fornisce *in output* un programma in linguaggio macchina

FASE 2

Il programma in linguaggio macchina, a cui sono forniti *in input* i dati, viene eseguito e fornisce *in output* i risultati

CODICE SORGENTE:
programma in linguaggio a. l.
input del compilatore

CODICE OGGETTO:
programma in linguaggio macchina,
output del compilatore,
traduzione del codice sorgente

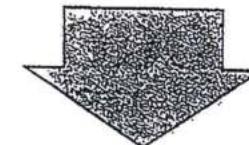
compilatore o interprete:
vantaggi e svantaggi

interprete:

Il programma in linguaggio a. l. è costantemente sotto il controllo dell'interprete

compilatore:

le fasi di compilazione ed esecuzione sono logicamente e temporalmente distinte



compilatore: più efficiente
interprete: più flessibile

La necessità di una fase di traduzione automatica crea il problema di definire:

- sotto quali condizioni una frase appartiene al linguaggio (*definizione sintattica*)
- il significato di ciascuna frase del linguaggio (*definizione semantica*) e delle interrelazioni tra le frasi del linguaggio

LA PROGETTAZIONE DEGLI ALGORITMI: COMPONENTI DI BASE E METODOLOGIE DI SVILUPPO

- variabili e costanti
- strutture di controllo
- variabili strutturate
- procedure



Variabili e costanti

Esempio 1:

Calcolo della circonferenza di un cerchio

raggio = 1

$$\text{circonferenza} = 2 \cdot \pi \cdot 1$$

raggio = 1.5

$$\text{circonferenza} = 2 \cdot \pi \cdot 1.5$$

raggio = 3

$$\text{circonferenza} = 2 \cdot \pi \cdot 3$$

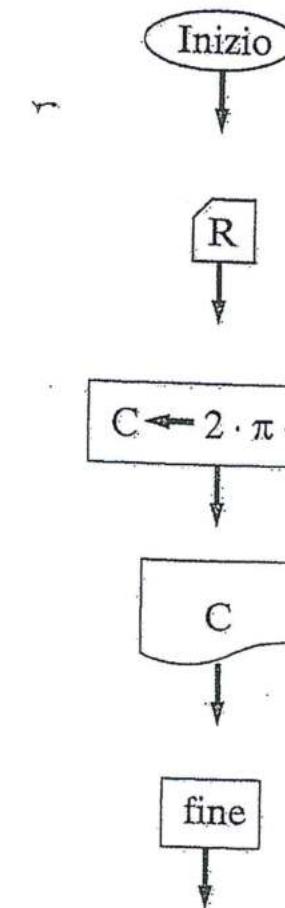


2π uguale per tutti i cerchi

raggio, circonferenza variano al variare del cerchio

ALGORITMO (prima versione)

FLOWCHART



PASCAL-LIKE

begin circonf

read R

$$C = 2 \cdot \pi \cdot R$$

print C

end circonf

L'algoritmo specifica
un procedimento generale per il calcolo
della lunghezza di una circonferenza

$2 \cdot \pi$ è un valore costante;

il valore del raggio R non è specificato;
al suo posto si utilizza un
nome che denota un oggetto variabile
(analogamente per C)



$2 \cdot \pi = COSTANTE$
R, C = VARIABILI

una variabile è un nome a cui si
associa un valore appartenente
ad un insieme prefissato

In un linguaggio di programmazione
una variabile è il nome simbolico
dell'indirizzo di una
locazione di memoria

Esempio:

<u>variabile</u>	<u>locazione di memoria</u>	<u>indirizzo</u>							
R	<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>								0011

Nell'algoritmo precedente
la variabile R indica un
generico numero reale

Il tipo di una variabile è l'insieme
dei valori che essa può assumere
(es. numeri reali, numeri interi, ...)



R = variabile di tipo reale

DICHIARAZIONE DI UNA VARIABILE

La dichiarazione di una variabile è la
specificazione del suo tipo
(reale, intero, alfanumerico, logico, ...)

Esempio:

var: R:	real	(R: variabile di tipo reale)
var: I:	real	(I: variabile di tipo reale)
var: NOME:	character	(NOME: variabile di tipo alfanumerico)
var: L:	logical	(L: variabile di tipo logico)



Esempio:

R

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

R: integer

R = +80

rappr. binaria del numero
intero 80 (primo bit = segno)

R: character

R = P

rappresentazione del dato
alfanumerico P (codice ASCII)

R: real

R = 0.625

rappr. f.p. norm. in base 2
del numero reale 0.625
(bit 1 = segno mant.,
bit 2-5 = cifre mant.,
bit 6 = segno espon.,
bit 7-8 = cifre espon.)

La dichiarazione di una variabile
consente di interpretare il contenuto
della corrispondente locazione di
memoria in base al tipo della variabile.

Algoritmo per il calcolo della circonferenza (seconda versione)

begin circonf

var: R, C: real

read R

C = 2 · π · R

dichiarazione
delle variabili R e C
di tipo reale

print C

end circonf

Nell'algoritmo precedente
il dato $2 \cdot \pi$ è costante

La costante identifica la locazione
di memoria a cui essa è associata,
**mediante l'indicazione esplicita
del suo contenuto**

RAPPRESENTAZIONE DELLE COSTANTI

**Una costante denota esplicitamente
un dato di un certo tipo**

COSTANTI ALFANUMERICHE

una sequenza di caratteri alfanumerici racchiusa tra
apici

'Napoli'
'telefono'
'12345'

COSTANTI LOGICHE

.TRUE. (VERO)

.FALSE. (FALSO)

COSTANTI INTERE

una sequenza di cifre decimali precedute eventualmente dal segno

1234
+27
-99012

COSTANTI REALI

una sequenza di cifre decimali preceduta eventualmente dal segno e contenente necessariamente il punto decimale

4.25 6.
+7.1 -2349.333

oppure

notazione esponenziale (floating-point), con le convenzioni precedenti per la mantissa

95.6e4	-0.13e2	$(95.6 \cdot 10^4, -0.13 \cdot 10^2)$
4.e-3		$4 \cdot 10^{-3}$

$$95.6 \text{e}4 = 95.6 \cdot 10^4 = 956,000$$

$$4 \cdot 10^{-3} = 4 \cdot 10^{-3} = \frac{4}{1000}$$

Algoritmo per il calcolo della circonferenza (versione finale)

begin circonf

var: R, C: real

read R

$C = 2 \times 3.141592 \times R$ ←

dichiarazione
esplicita del
valore di π

print C

end circonf

DEFINIZIONE DI UNA VARIABILE

La definizione di una variabile è l'assegnazione di un valore alla variabile

Esempio:

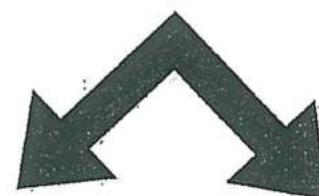
```
begin area_triangolo
    var: altezza, base: real
```

```
read altezza
    base = 1.3
```

.....

```
end area_triangolo
```

L'assegnazione di un valore ad una variabile dichiarata di un certo tipo può essere fatta mediante le operazioni di:



assegnazione

base = 1.3

lettura

read altezza

Esempio:

```

begin area_triangolo
    var: altezza, base, area: real

    read altezza
    base = 5.

    area = base × altezza/2.

    print area
end area_triangolo

```

In una istruzione di assegnazione **prima** si valuta l'espressione al secondo membro e **poi** si assegna il valore alla variabile al primo membro.

La valutazione dell'espressione consiste **prima** nella sostituzione a ciascuna variabile del proprio valore e **poi** nella esecuzione delle operazioni specificate.

Esempio:

```

begin area_triangolo
    var: altezza, base, area: real

    read altezza
    base = 5.

    area = base × altezza

    area = area/2.

    print area
end area_triangolo

```

L'ultima assegnazione ha senso perché si valuta **prima** il secondo membro, e **poi** si assegna il risultato alla variabile area.

L'ultima assegnazione fa perdere il valore precedente della variabile **area** (l'operazione di assegnazione è **distruttiva**)

Esempio:

```
begin perimetro_triangolo
    var: lato1, lato2, lato3, somma; real
    read lato1
    lato2 = 5.

    somma = lato1+lato2+lato3

end perimetro_triangolo
```

ERRATO !!!!

la variabile **lato3** non è stata definita

tutte le variabili
devono essere definite
prima di essere utilizzate

Esempio:

```
begin area_rettangolo
    var: area: logical
    var: base, altezza; real
    altezza = 2.5
    base = 5.1

    area = base × altezza

end area_triangolo
```

ERRATO !!!!

Il valore dell'espressione deve
essere dello stesso tipo di quello
della variabile a cui è assegnato

L'assegnazione di un valore ad una variabile è **ERRATA** se:

- l'espressione non può essere valutata perché vi compaiono variabili indefinite
- il valore dell'espressione è di tipo diverso da quello della variabile a cui è assegnato.

Nel linguaggio della Computer Science

il termine variabile indica un oggetto il cui valore, se non viene effettuata alcuna operazione su tale oggetto, rimane costante nel tempo.

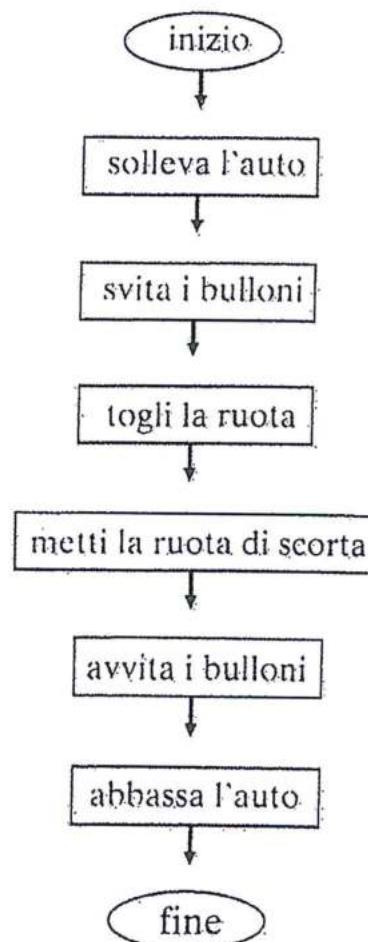
Nel linguaggio della Matematica

una variabile non rappresenta uno specifico valore costante nel tempo
ma il generico elemento di un certo insieme

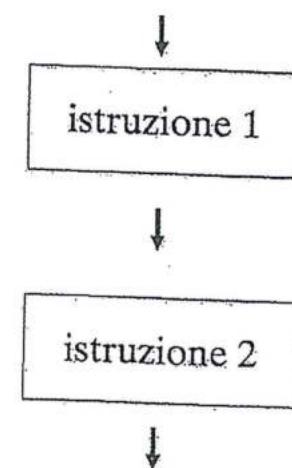
(ES.: $\forall x \in \mathbb{R}, x^2 \geq 0$ indica un generico numero reale.)

Strutture di controllo

Flow chart dell'algoritmo per il cambio della ruota



le istruzioni del flow chart



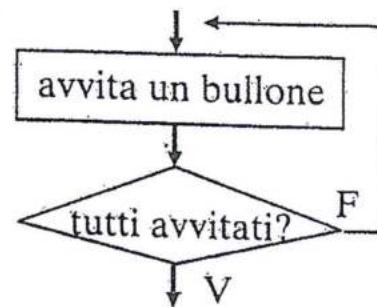
costituiscono una

**SEQUENZA
DI ISTRUZIONI**

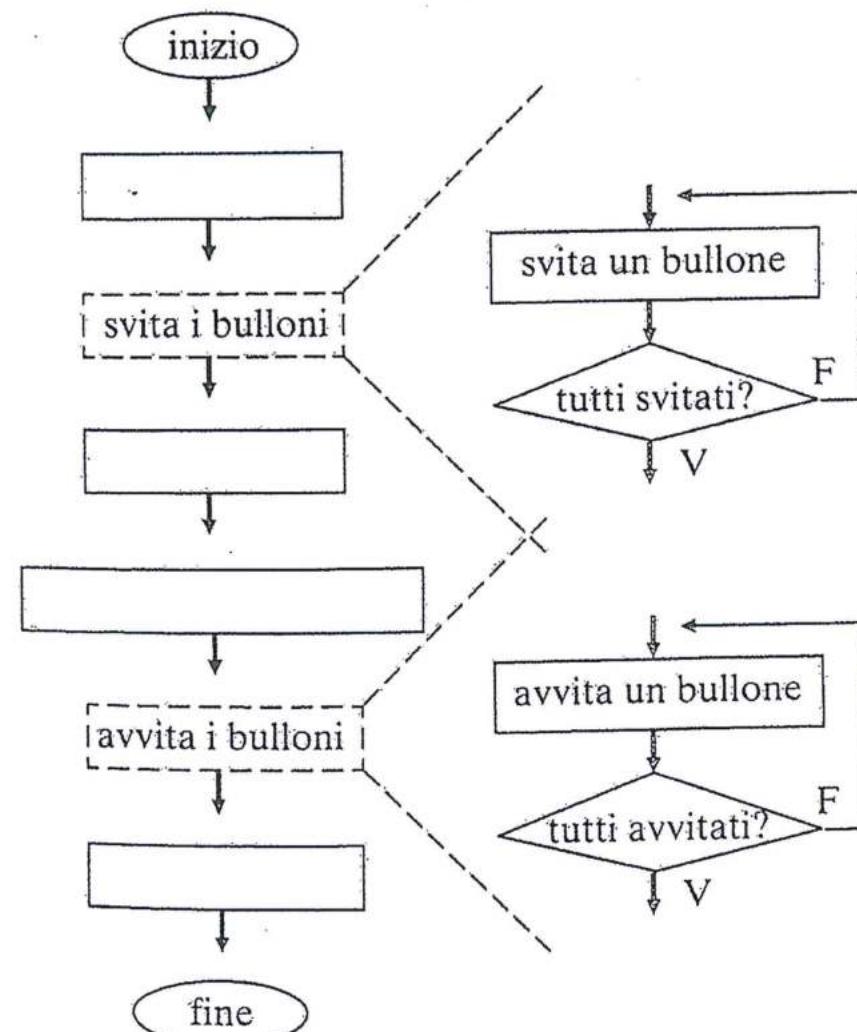
l'istruzione

avvita i bulloni

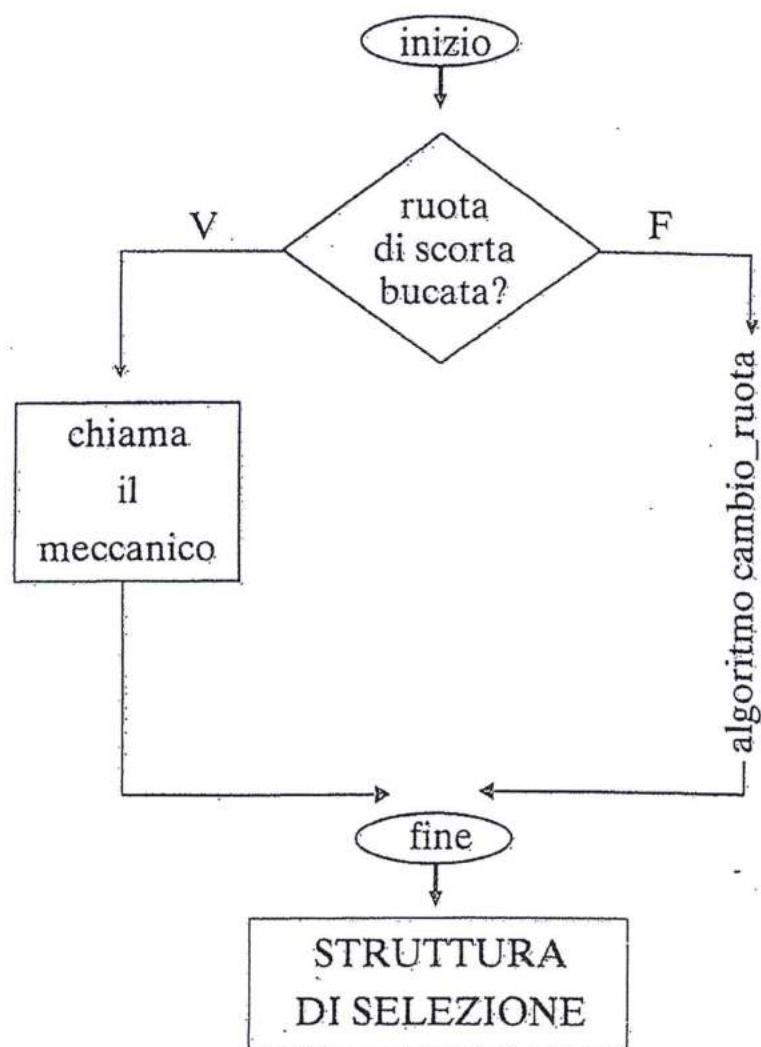
può essere sostituita da



**STRUTTURA
DI ITERAZIONE**



Che succede se la ruota di scorta è bucata?



Le strutture di controllo (o costrutti di controllo)

determinano l'ordine con cui devono essere eseguite le istruzioni

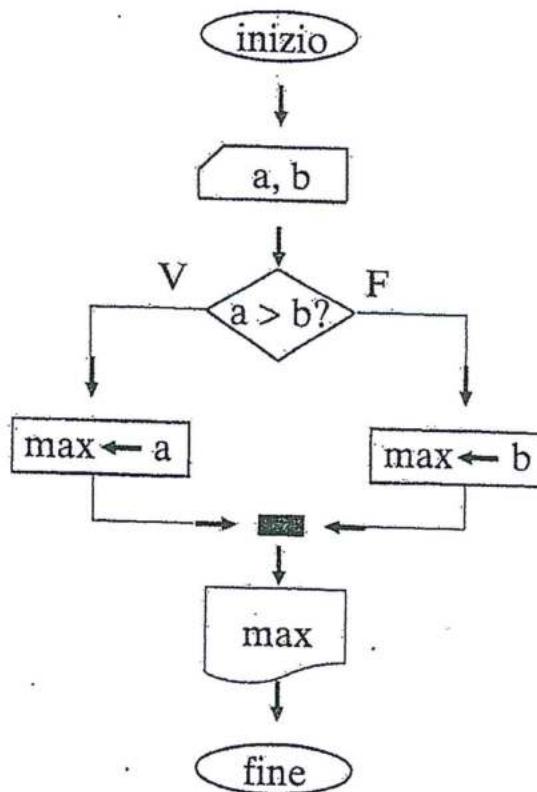
- sono indipendenti dalla natura delle istruzioni
- sono strumenti logici universali utilizzabili in qualunque problema

Analisi delle strutture di controllo

Esempio:

massimo tra 2 numeri

dati di input: i due numeri (a,b)
dati di output: il massimo (max)



PASCAL LIKE

```
begin massimo  
var: a, b, max: real
```

```
read a, b
```

```
if (a > b) then
```

```
    max = a
```

```
else
```

```
    max = b
```

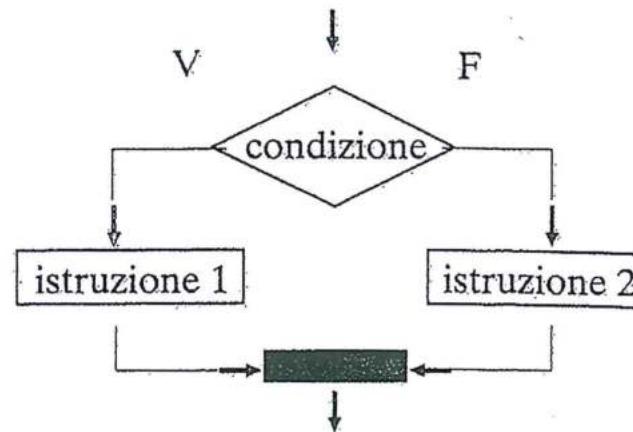
```
endif
```

```
print max
```

```
end massimo
```

struttura
di
selezione

La struttura di selezione
del linguaggio del flow chart:



nel linguaggio Pascal-like
si traduce con

if (condizione) then

 istruzione 1

else

 istruzione 2

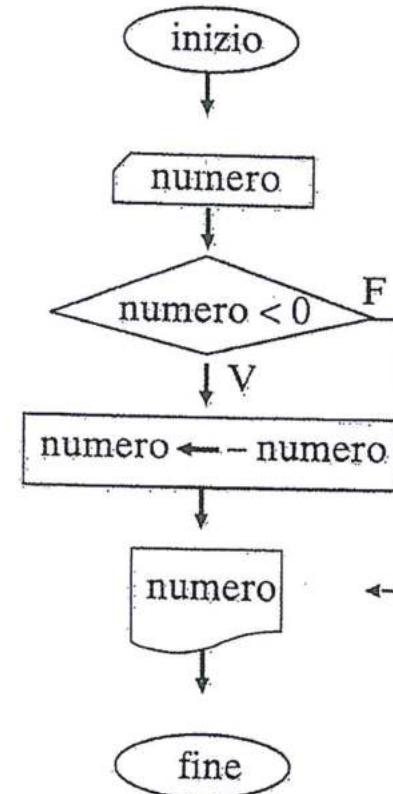
endif

Esempio:

valore assoluto di un numero reale

dati di input: numero

dati di output: valore assoluto del numero



PASCAL-LIKE

```

begin valore_assoluto
  var: num: real

  read num

  if (num < 0) then
    num = - num

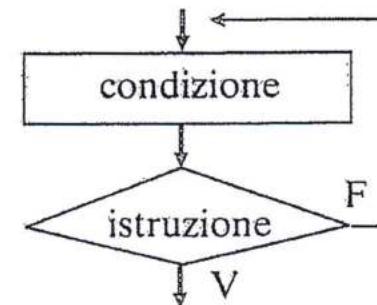
  endif

  print num

end valore_assoluto

```

la struttura di selezione
del linguaggio del flow chart:



nel linguaggio Pascal-like
si traduce con

```

if (condizione) then
  istruzione
endif

```

Esempio:

somma di N numeri

dati di input: valore di N,
N numeri

dati di output: somma degli N
numeri

dati di input: 4
5 7 10 -2

dati di output: 20

inizio:

sum = 0

passo 1:

read numero

sum = sum+numero

passo 2:

read numero

sum = sum + numero

passo 3:

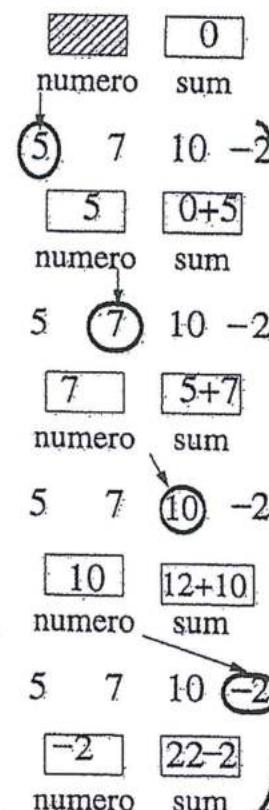
read numero

sum = sum + numero

passo 4:

read numero

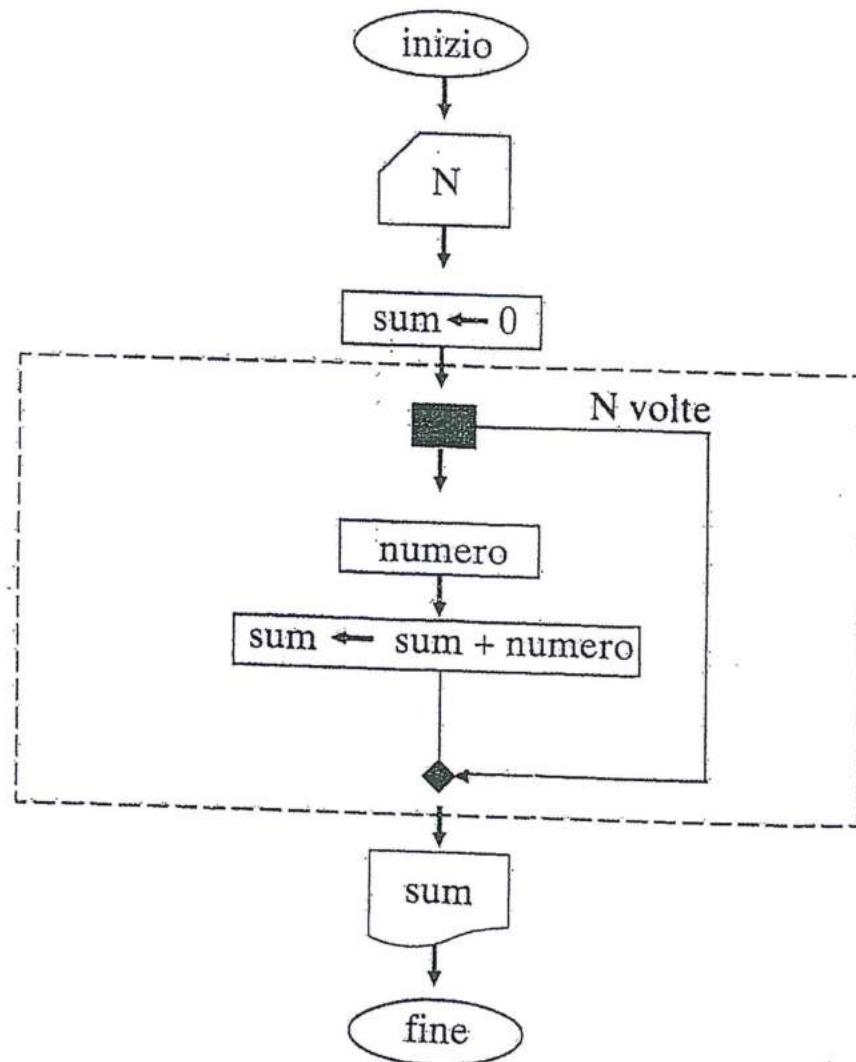
sum = sum + numero



4 volte
(# dati)

20
sum

FLOW CHART



PASCAL LIKE

```
begin somma  
var: N, i: integer  
var: sum, numero: real
```

```
read N  
sum = 0.
```

```
for i = 1, N do  
    read numero  
    sum = sum + numero
```

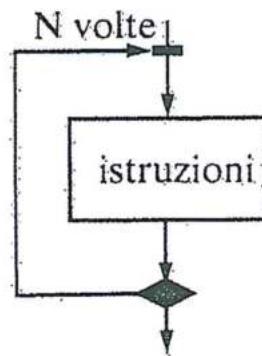
```
endfor
```

```
print num
```

```
end somma
```

struttura
di iterazione

la struttura di iterazione del linguaggio del flow chart:



si traduce nel linguaggio Pascal-like con

```

for i = 1, N do
  istruzioni
endfor
  
```

Le strutture di controllo possono essere
innestate l'una nell'altra

```

for i = ...
  for j = ...
    for k = ...
      if (condizione) then
        istruzioni
      endif
    endfor
  endfor
endfor
  
```

Esempio:

calcolare il massimo
di N numeri

dati di input: N, N numeri

dati di output: massimo degli N
numeri

dati di input: 4

5 7 10 -2

dati di output: 10

inizio:

read numero
massimo:=numero

passo 1:

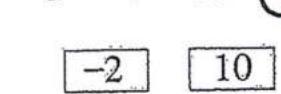
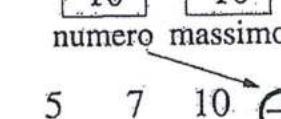
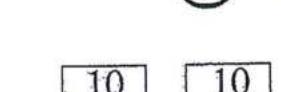
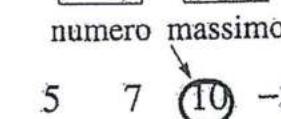
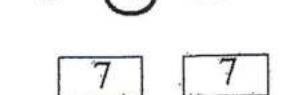
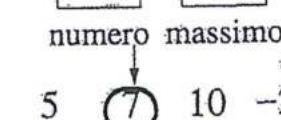
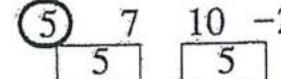
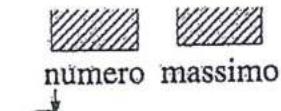
read numero
if (massimo<numero)
massimo:=numero

passo 2:

read numero
if (massimo<numero)
massimo:=numero

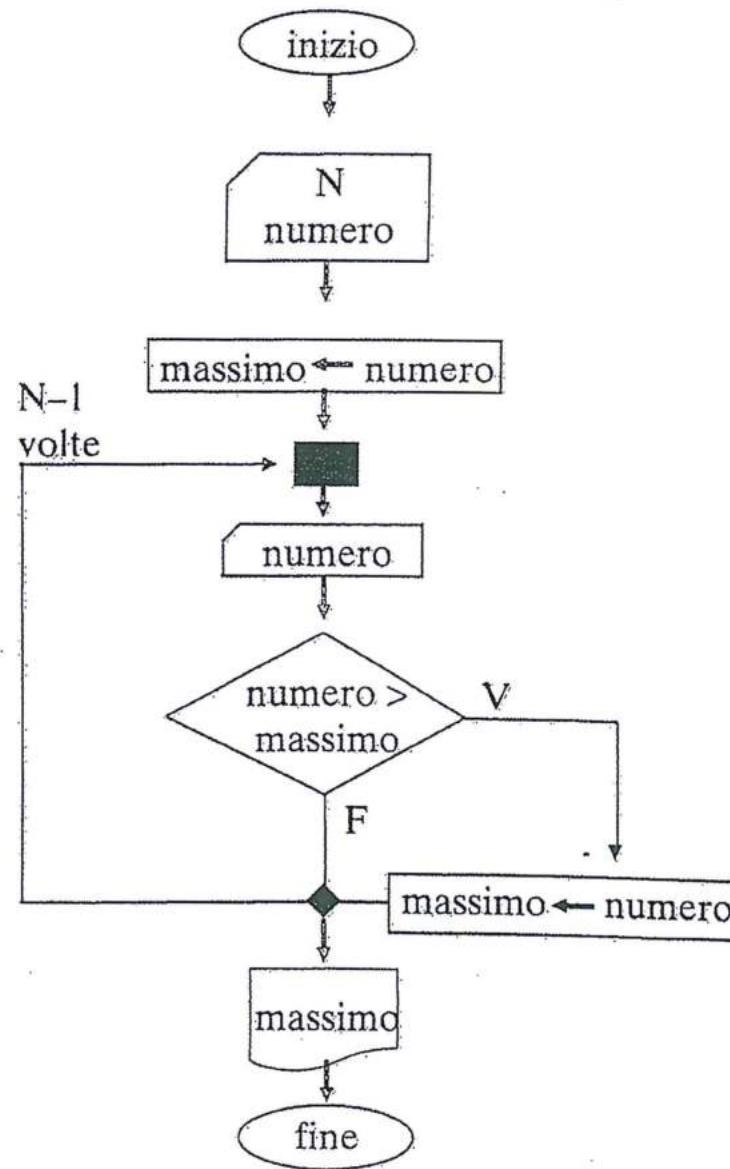
passo 3:

read numero
if (massimo<numero)
massimo:=numero



10
massimo

4 volte
(# dati)

FLOW CHARTPASCAL LIKE

```

begin max_n_numeri
var: numero, massimo: real
var: i, N: integer

read N, numero
massimo = numero

for i = 1, N-1 do
    read numero
    if (numero > massimo) then
        massimo = numero
    endif
endfor

print massimo

end max_n_numeri

```

Esempio:

Calcolo del prodotto
di un insieme di N numeri

dati di input: N, N numeri

dati di output: prodotto degli N
numeri

dati di input: 4
5 2 0 1

dati di output: 0

inizio : ⑤ 2 0 1

read numero
prodotto:=numero

numero prodotto

numero prodotto

passo 1: 5 ② 0 1

read numero
prodotto:=prodotto*numero

numero prodotto

passo 2: 5 2 ① 1

read numero
prodotto:=prodotto*numero

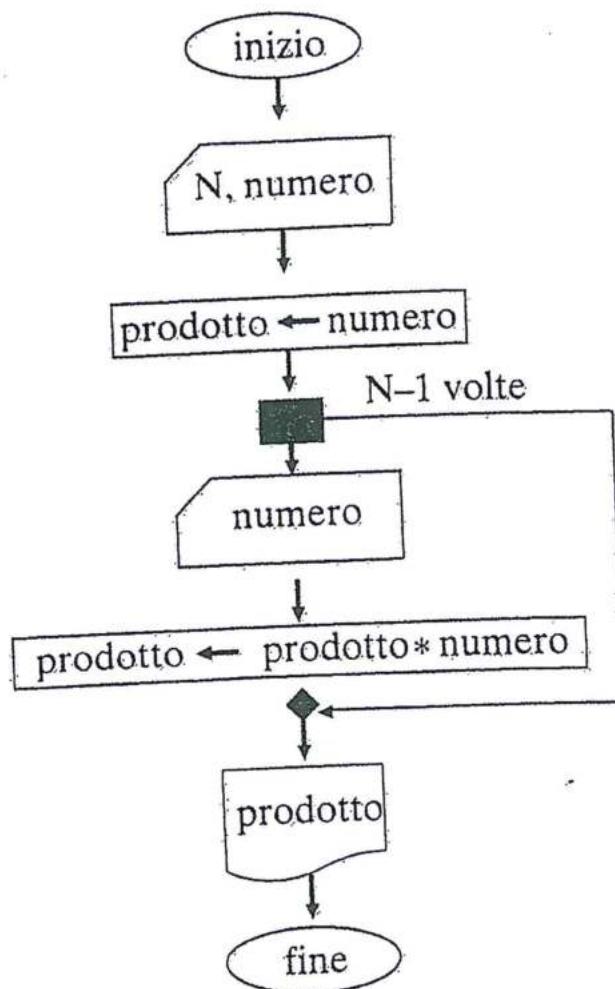
numero prodotto

passo 3: 5 2 ① 1

read numero
prodotto:=prodotto*numero

numero prodotto

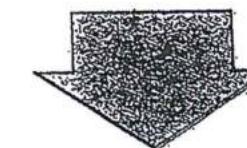
prodotto

FLOW CHART**PROBLEMA**

Se un numero è nullo,
il prodotto è nullo



Prevedere il caso in cui
un dato sia nullo



Si arresta il calcolo quando
sono stati moltiplicati tutti i numeri
oppure
quando un dato è nullo

inizio : ⑤ 2 0 1
read numero
prodotto=numero
 $i \leftarrow 1$

■ ■ ■
i numero prodotto

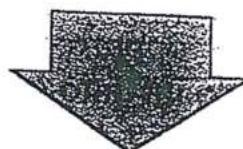
passo 1 : 5 ② 0 1
read numero
prodotto=prodotto*numero
 $i \leftarrow i+1$

1 5 5
i numero prodotto

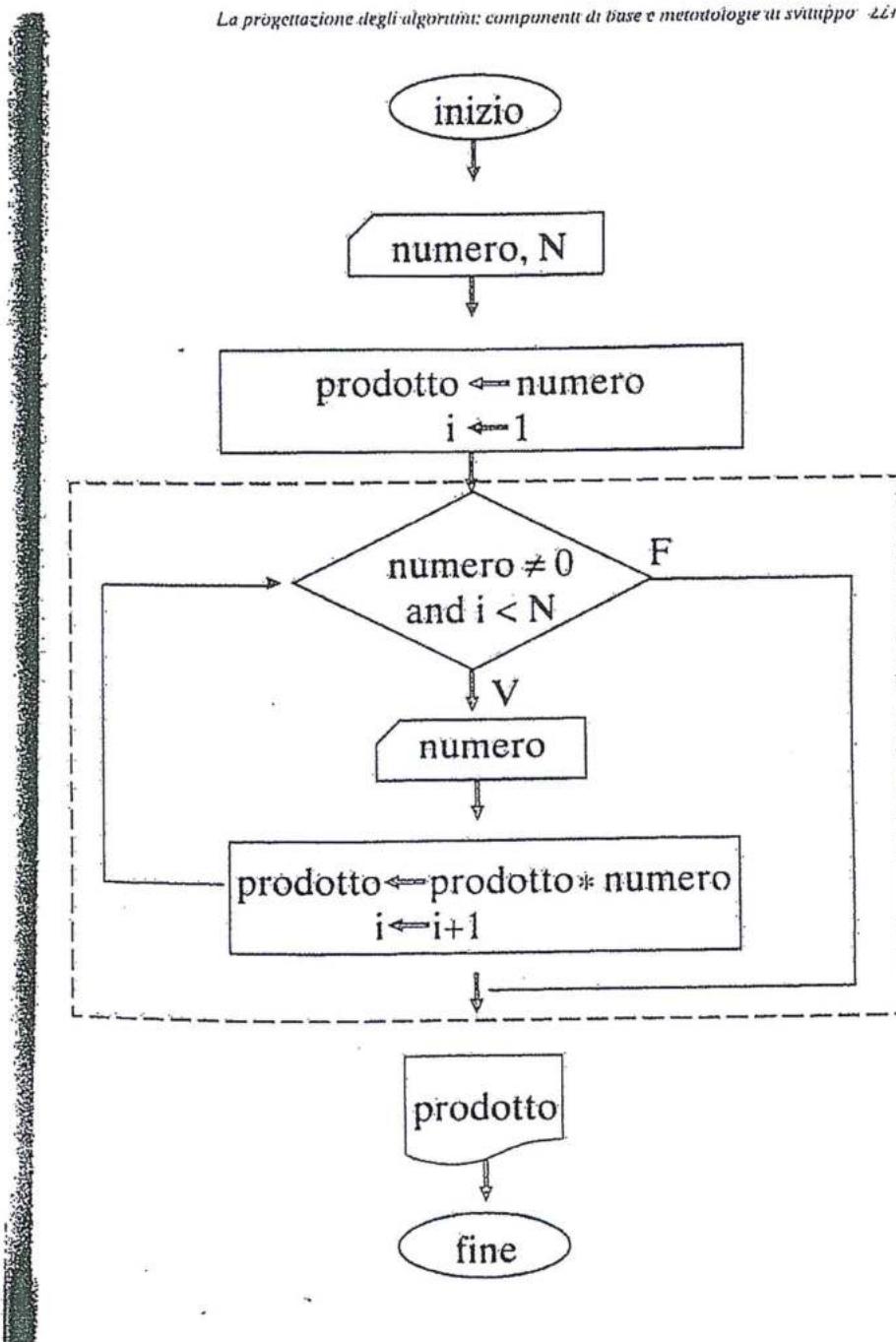
passo 2 : 5 2 ① 1
read numero
prodotto=prodotto*numero
 $i \leftarrow i+1$

2 2 10
i numero prodotto

3 0 0
i numero prodotto



STOP



PASCAL-LIKE

```

begin prod_numeri
  var: N, i: INT.INTEGER
  var: numero, prodotto: real

  read numero, N
  prodotto := numero
  i := 1

  while (numero ≠ 0 and i < N) do
    read numero
    prodotto := prodotto * numero
    i := i + 1

  endwhile
endwhile
  print prodotto
end prod_numeri

```

*Quando questa cond. è falsa,
il programma passa
alle istruzioni
dopo endwhile*

*Quello che si fa con il ciclo
for se può fare anche con
while*

Esempio:

ricerca di un elemento dato
in un insieme di N numeri

dati di input: N, N numeri
numero da cercare

dati di output: informazione che indica
se il num. è stato trovato

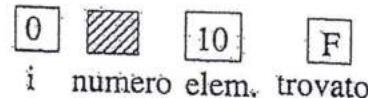
"posizione" del numero

dati di input: 4
2 1 10 5
10

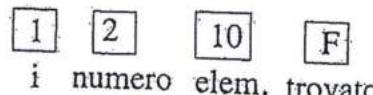
dati di output: true., 3

~~inizio~~

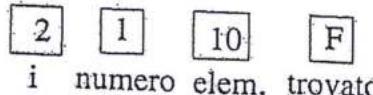
inizio:



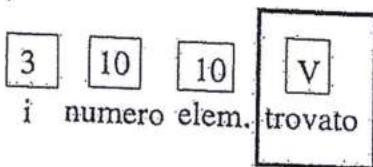
passo 1: ② 1 10 5
read numero
numero = elemento ?
i=i+1



passo 2: 2 ① 10 5
read numero
numero = elemento ?
i=i+1

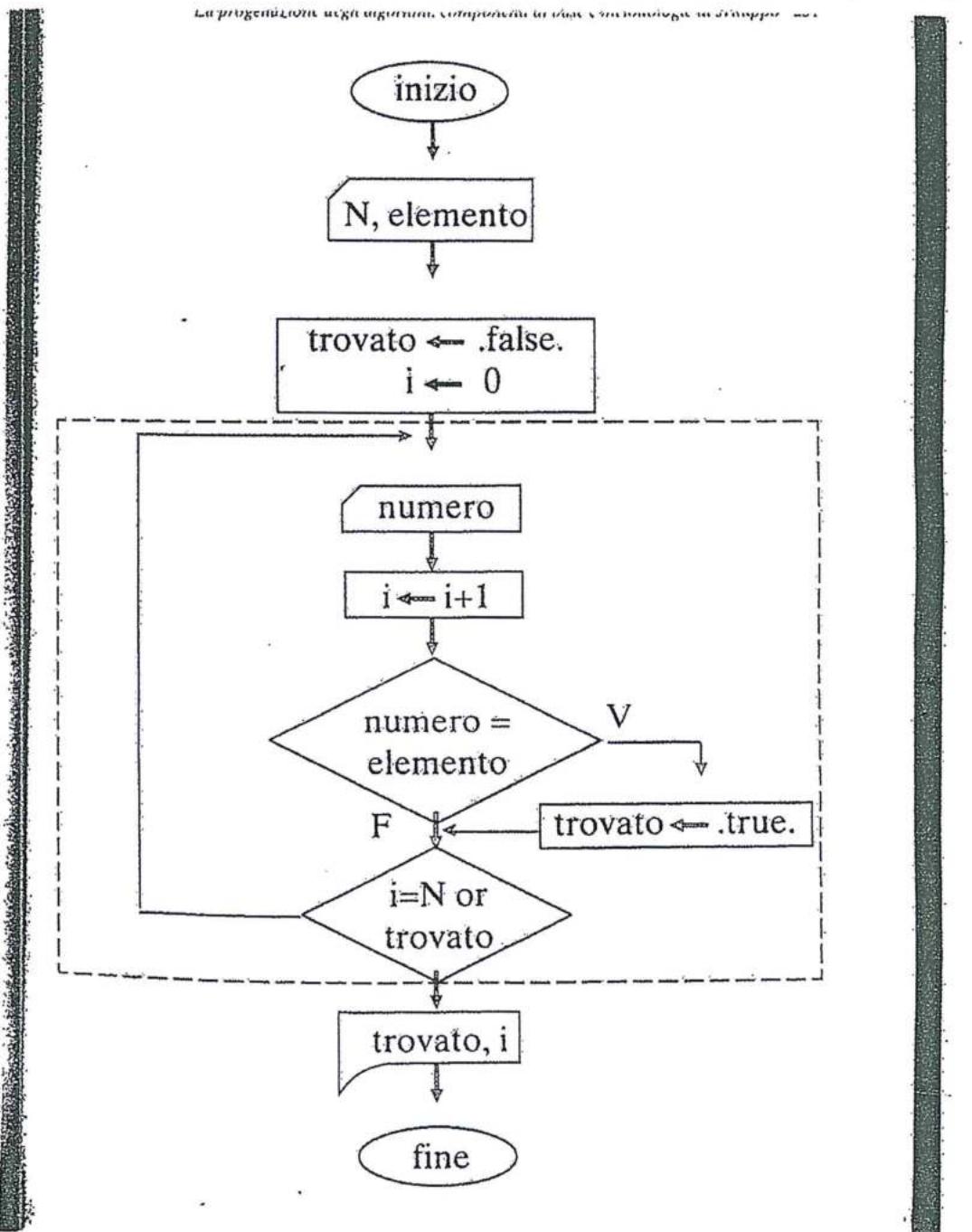


passo 3: 2 1 ⑩ 5
read numero
numero = elemento ?
i=i+1



STOP !

La programmazione degli algoritmi comprende le due sottostages di sviluppo ...



PASCAL LIKE

E' veniente = in corso di doppioni

begin ricerca

var: N, numero, elemento, i: integer
var: trovato: logical

read N, elemento
trovato: = .false. *inizializzazione*
i: = 0

repeat

read numero 2, 1, 10
 i: = i+1 1, 2, 3
 if (elemento = numero) then 10, 10, SI
 trovato: = .true.
 endif

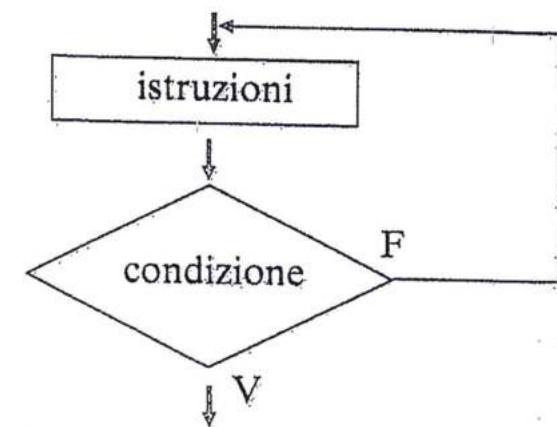
until (i=N or trovato)

print trovato, i
 1-4
 2-4
 false
 true

end ricerca

*In while può non essere eseguito
il repeat until si verifica almeno 1 volta*

la struttura di iterazione del linguaggio del flow chart:



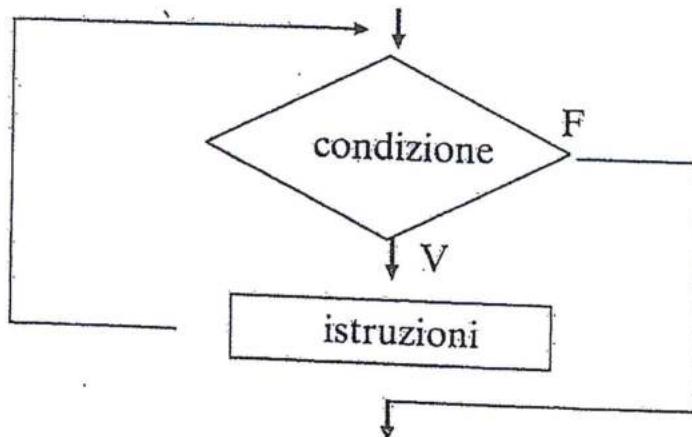
si traduce nel linguaggio Pascal-like con

repeat

 istruzioni

until (condizione)

la struttura di iterazione del linguaggio del flow chart:



si traduce nel linguaggio Pascal-like con

while (condizione) do

 istruzioni

endwhile

Differenze fra le tre strutture di iterazione

- Il for richiede che sia **noto a priori il numero di iterazioni da effettuare**
- while e repeat non richiedono tale informazione.
- nel while se la condizione è falsa non si esegue nessuna istruzione del ciclo
- nel repeat le istruzioni del ciclo sono eseguite almeno una volta



Variabili strutturate
(array)

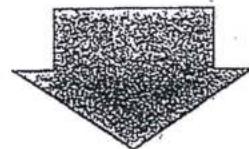
successione (di numeri reali):

$$i \in \mathcal{N} \rightarrow a_i \in \mathbb{R}$$

vettore (di numeri reali):

$$i \in \mathcal{I} \rightarrow a_i \in \mathbb{R}$$

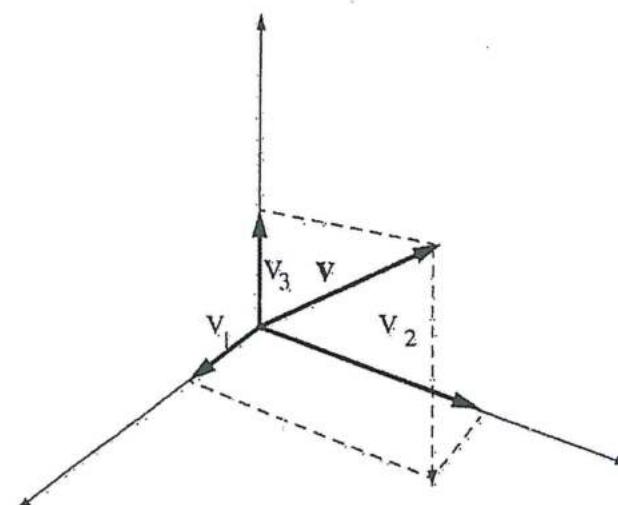
con $\mathcal{I} \subset \mathcal{N}$ sottoinsieme finito



ogni componente del vettore è
univocamente determinata
dall'indice i (= posizione $i-ma$)

Esempio: $a_1, a_2, a_3, \dots, a_{10}$

Nello spazio...



ogni componente del vettore v
è univocamente determinata
dalla proiezione del vettore
lungo una direzione cartesiana

IN GENERALE

NOMI COLLETTIVI

Necessità di individuare con un nome **non un singolo valore**, ma **un insieme finito di valori**, e di fare riferimento ad ogni singolo valore dell'insieme

ESEMPI DI NOMI COLLETTIVI

elenco telefonico

primo abbonato
secondo abbonato
.....

fila di utenti ad uno sportello

primo utente
secondo utente
.....

mazzo di carte

asso di cuori
due di cuori
.....

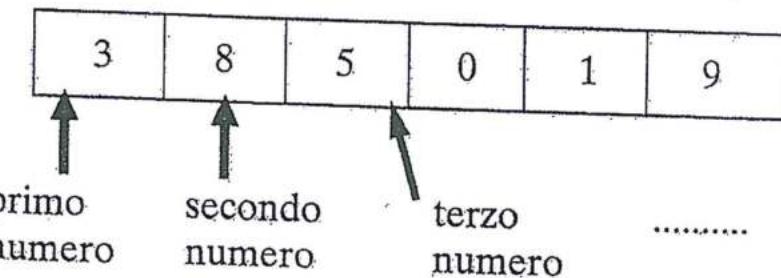
nel linguaggio algoritmico
un nome collettivo prende il nome di

VARIABILE STRUTTURATA

**nome a cui è associato un
insieme di valori
univocamente determinati
da un criterio prestabilito.**

L'ARRAY

Nella variabile strutturata **array**,
il criterio con cui sono
organizzate le componenti,
è la relazione d'ordine stretto,
definita nell'insieme degli indici,
ciascuno dei quali indica la posizione
occupata dalla corrispondente componente



ESEMPI

TAB = variabile strutturata
alla quale è associata la seguente sequenza di
6 interi

3	8	5	0	1	9
---	---	---	---	---	---

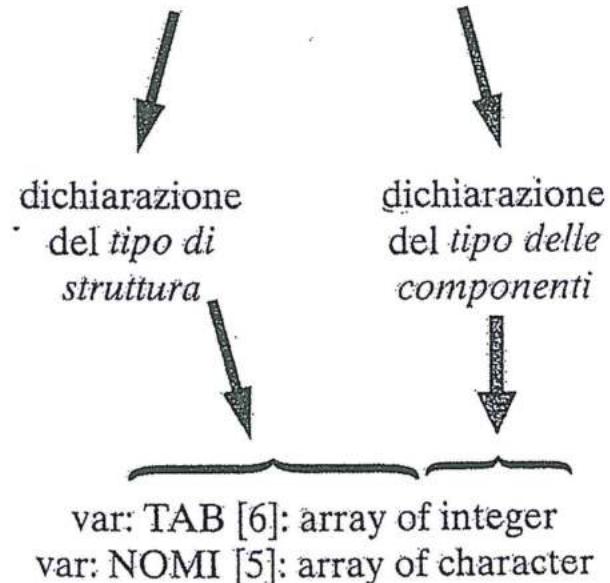
NOMI = variabile strutturata
alla quale è associata la seguente sequenza di
5 dati alfanumerici

MARIO	LAURA	PIPO	GINO	ELSA
-------	-------	------	------	------

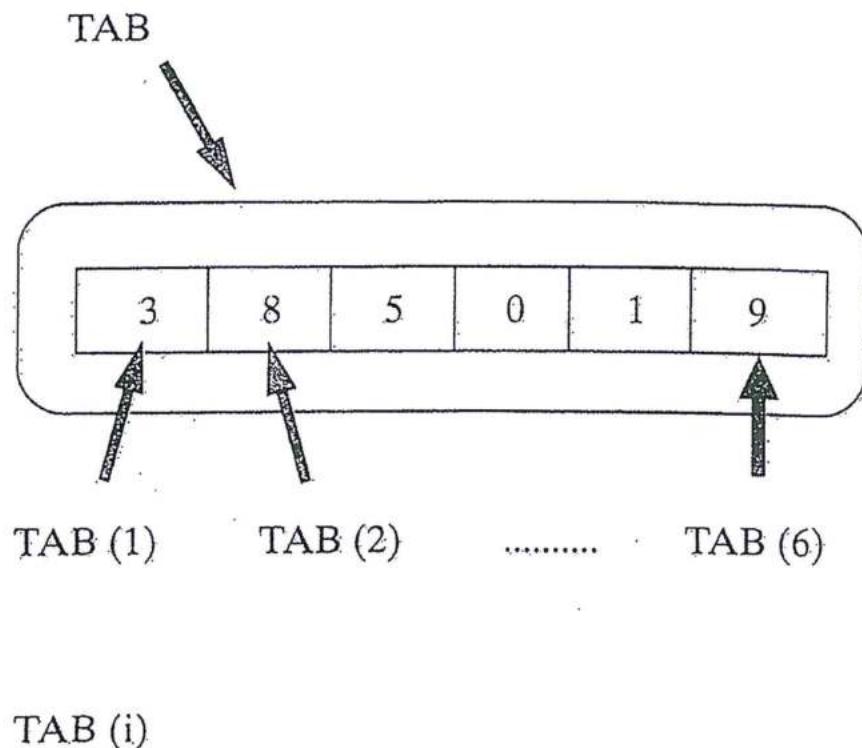
OSSERVAZIONE:

tutte le componenti di un array
sono dello stesso tipo

DICHIARAZIONE DI UN ARRAY



con la dichiarazione viene
fissata la dimensione massima
dell'array



identifica l' i -ma componente di TAB

TAB (i) è una variabile (scalare)

Esempio:

TAB [6]; array of integer

3	8	5	0	1	9
---	---	---	---	---	---

3	8	22	0	1	9
---	---	----	---	---	---

$i=3$

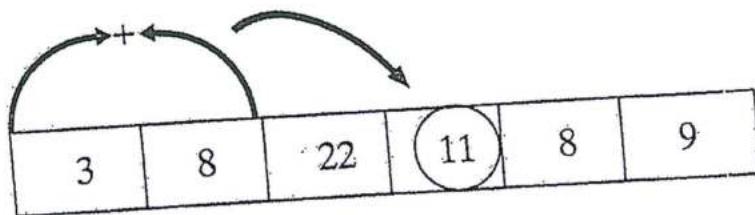
TAB (3) = 22

3	8	22	0	8	9
---	---	----	---	---	---

$i=5$
TAB (2) TAB (i) = TAB (2)

3	8	22	1	8	9
---	---	----	---	---	---

```
for i = 1,4 do
    read TAB(i)
endfor
```



$i = 1, j = 2, k = 4$
 $TAB(k) = TAB(i) + TAB(j)$

0	0	0	11	8	9
---	---	---	----	---	---

```
for i = 1,3 do
    TAB(i) = 0
endfor
```

3	8	22	11	8	9
---	---	----	----	---	---

$i = 6$
 $TAB(i) = 'luisa'$

ERRATO!

il valore da assegnare deve essere
dello stesso tipo di quello della
variabile a cui è assegnato

3	8	22	11	8	9
---	---	----	----	---	---

i=10
 TAB(i) = 20

ERRATO!

non si può fare riferimento
 ad una componente che
 non è stata dichiarata

Esempio:

calcolo della media delle
 temperature (in gradi C)
 di una settimana

13.5	15.4	16.2	14.1	12.8	10.1	11.0
------	------	------	------	------	------	------

var: TEMP[7]: array of real

lunedì	13.5
martedì	15.4
mercoledì	16.2
giovedì	14.1
venerdì	12.8
sabato	10.1
domenica	11.0

```
begin settimana
```

```
var: TEMP[7]: array of real
```

```
var: media: real
```

```
var: i: integer
```

dichiarazione

```
for i = 1,7 do
```

```
    read TEMP(i)
```

```
endfor
```

lettura

```
media = 0.
```

```
for i = 1,7 do
```

```
    media = media + TEMP(i)
```

```
endfor
```

```
media = media/7.
```

```
print media
```

```
end settimana
```

PROBLEMA

come calcolare la temperatura
media di due settimane?

prima settimana

13.2	15.1	16.9	14.4	12.2	15.5	11.6
------	------	------	------	------	------	------

var: temp1[7]: array of real

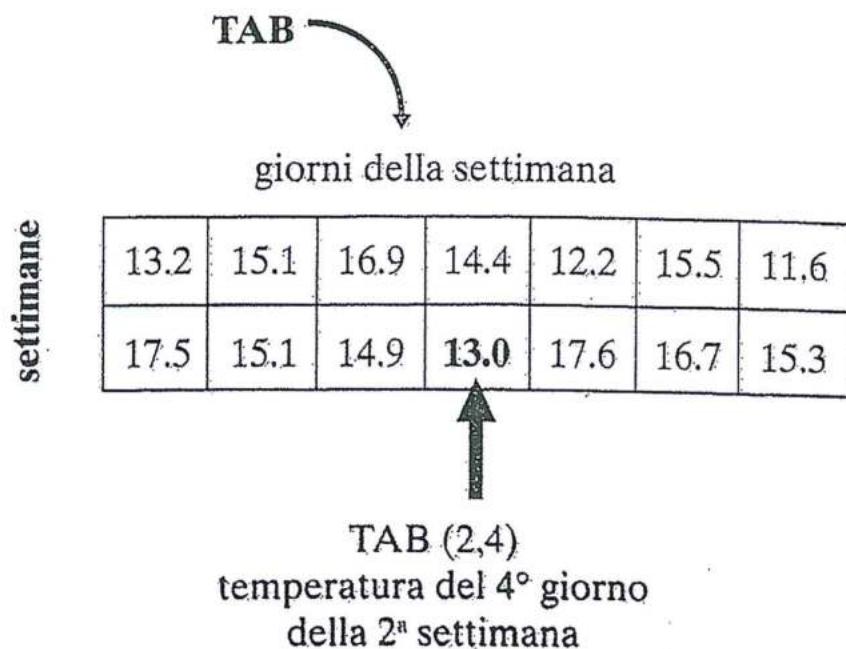
seconda settimana

17.5	15.1	14.9	13.0	17.6	16.7	15.3
------	------	------	------	------	------	------

var: temp2[7]: array of real

scelta più naturale:

definire ***una tabella (o matrice)***
di temperature TAB del tipo



DICHIARAZIONE

var: TAB [2, 7]: array of real

tale dichiarazione definisce
un array con

- 2 righe (numero di settimane)
- 7 colonne (giorni della settimana)

13.2	15.1	16.9	14.4	12.2	15.5	11.6
17.5	15.1	14.9	13.0	17.6	16.7	15.3

array bidimensionale

TAB (i, j) identifica l'elemento della i-ma riga (i-ma settimana) e della j-ma colonna (j-mo giorno)

13.2	15.1	16.9	14.4	12.2	15.5	11.6
17.5	15.1	14.9	13.0	17.6	16.7	15.3

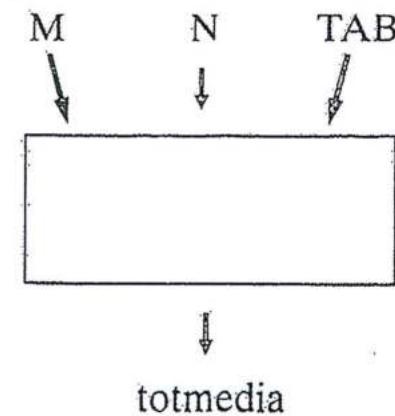
TAB (2,1) TAB (1,4) TAB (2,7)

Esempio:

costruzione di un algoritmo per
il calcolo della temperatura
media di M=2 settimane
(ogni settimana è composta di N=7 giorni)

13.2	15.1	16.9	14.4	12.2	15.5	11.6
17.5	15.1	14.9	13.0	17.6	16.7	15.3

var; TAB [2, 7]: array di reali :

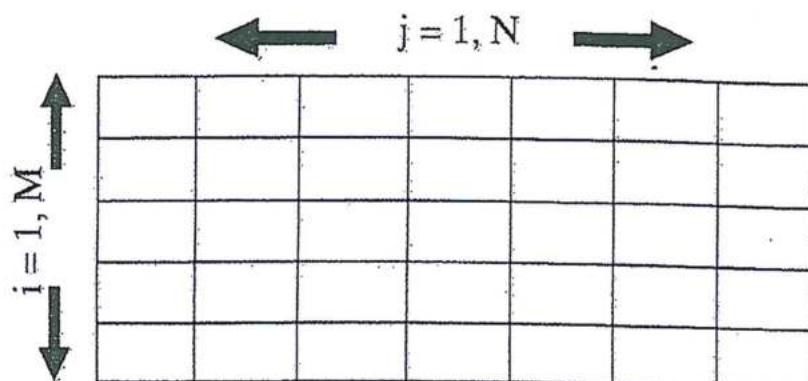


```

begin temp_media
var: TAB[5 ; 7]: array di reali
var: totmedia: real
var: M, N, i, j: intero

read M
N = 7
lettura dell'array TAB
totmedia = 0.
for i = 1, M do
    for j = 1, N do
        totmedia = totmedia + TAB (i,j)
    endfor
endfor
totmedia = totmedia / (M*N)
print totmedia
end temp_media

```



Caratteristiche di un array:

- dimensione fissata al momento della dichiarazione
- componenti dello stesso tipo
- ogni componente viene univocamente individuata tramite l'indice (posizione)
- le componenti sono memorizzate in parole consecutive

PROBLEMA

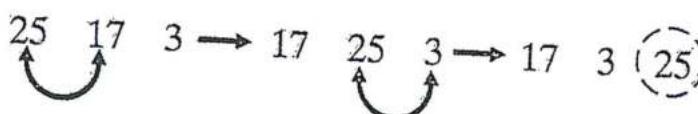
ordinamento crescente di 3 numeri:

25 17 3

I passo: 25 17 3

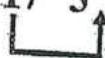


idea



in due scambi 25 è stato portato
al suo posto

II passo: 17 3 25



idea

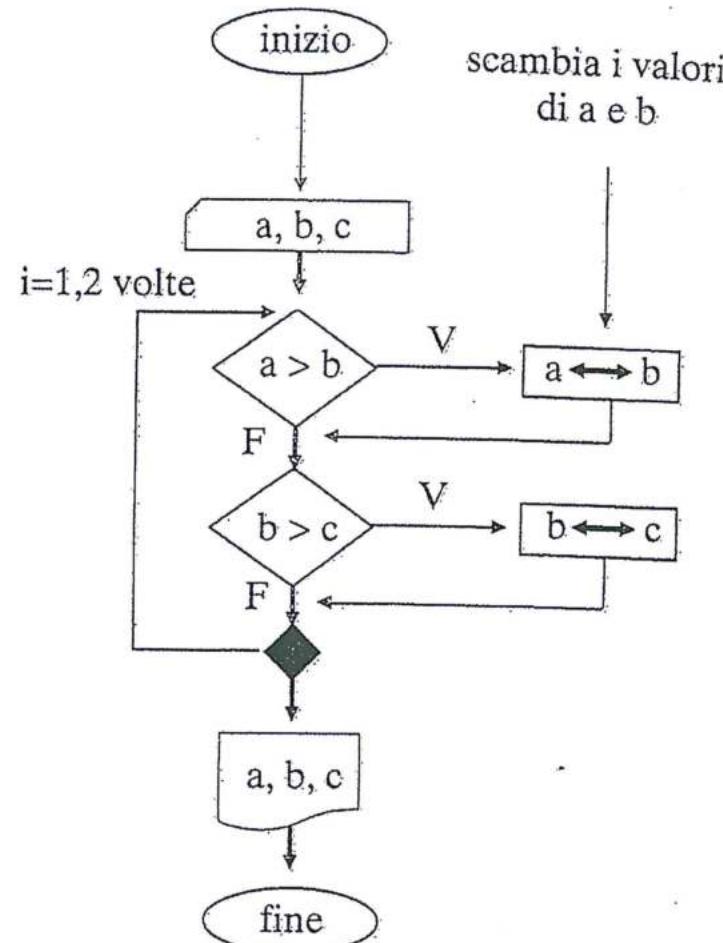


in uno scambio 17 è stato portato
al suo posto
ordinamento completato!

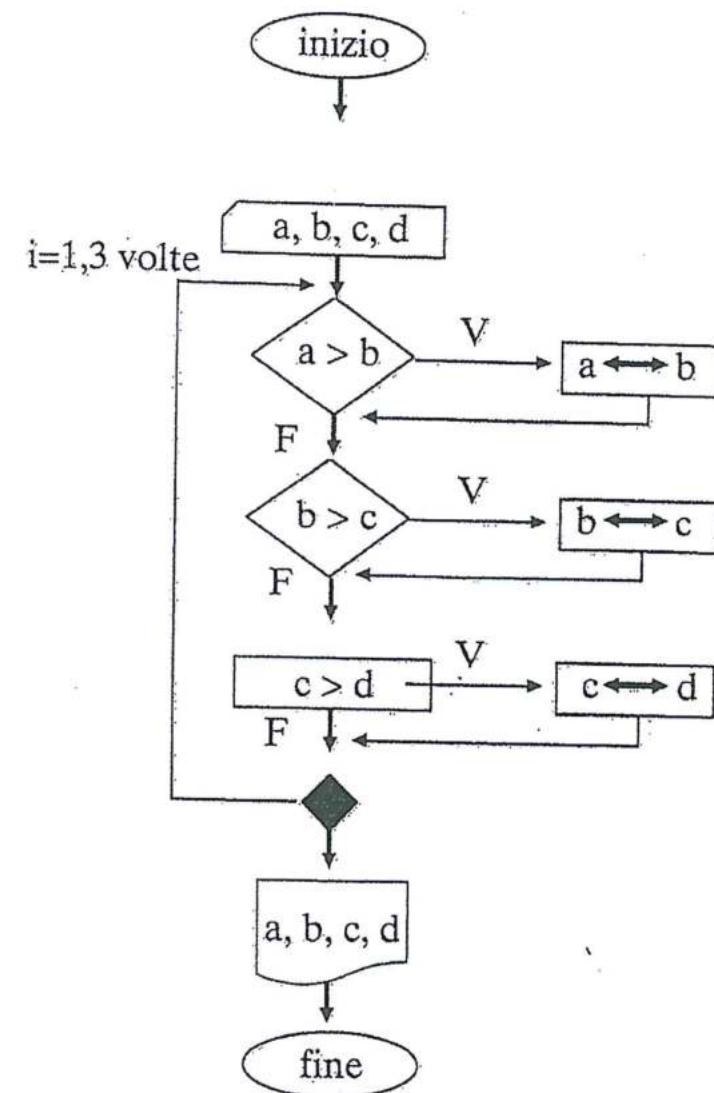
STRATEGIA

si scambiano di posto due elementi
adiacenti che non sono in ordine,
in modo da spostare
il massimo verso destra

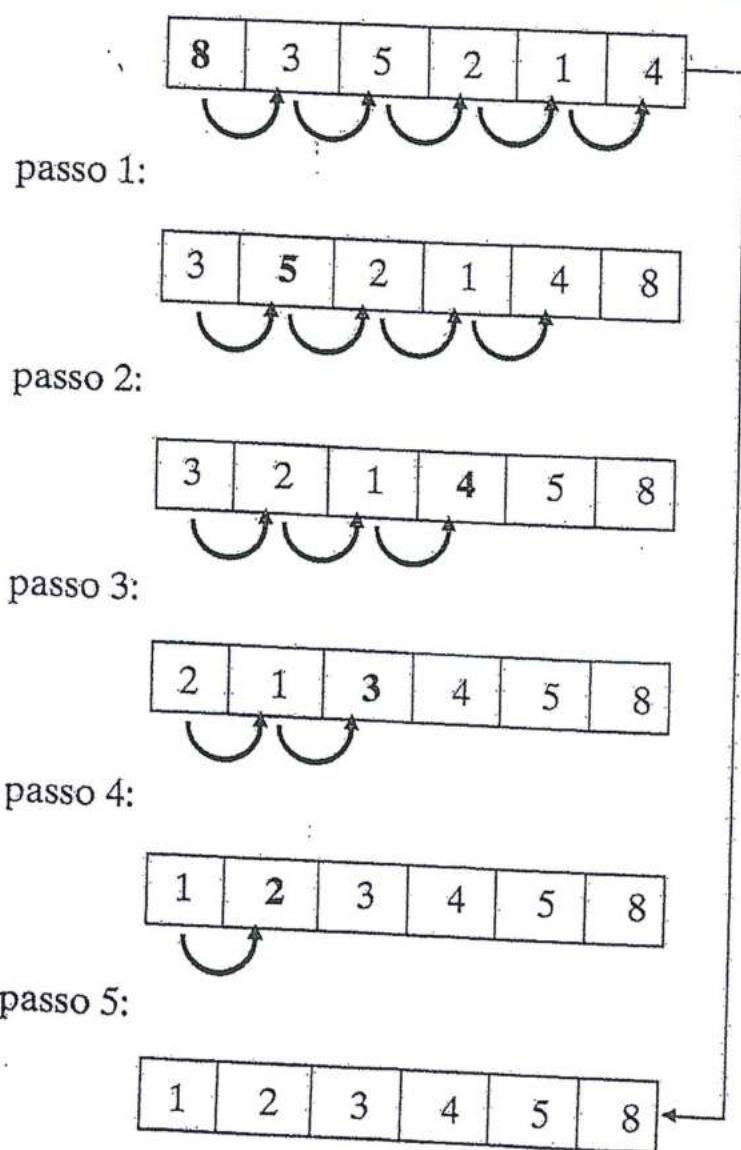
si ripete fino al completo ordinamento
(ORDINAMENTO PER SCAMBI)



analogamente con 4 numeri

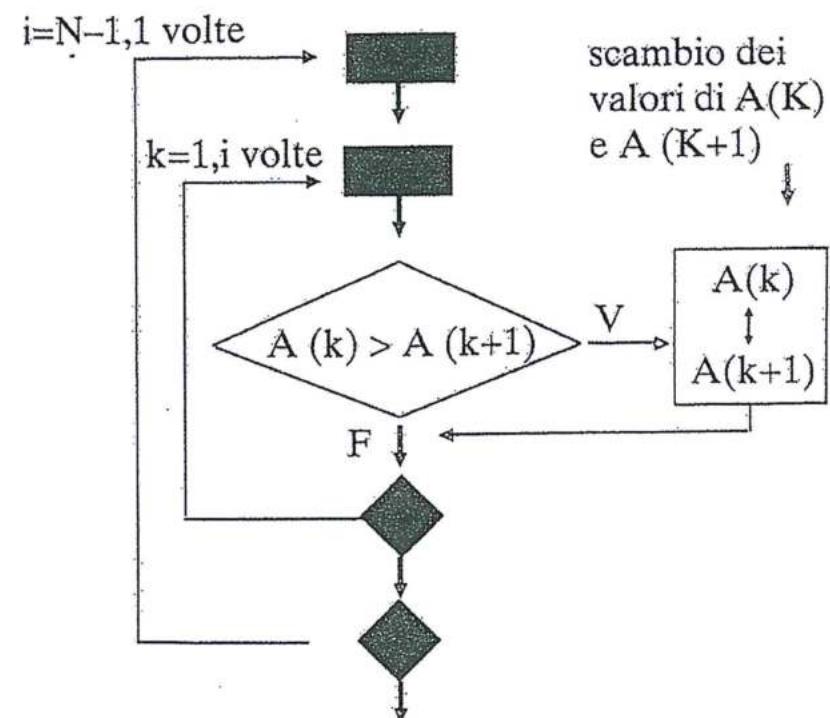


ORDINAMENTO DI N NUMERI



l'algoritmo per l'ordinamento di N numeri è il seguente

var: A[100]: array di interi



In Pascal-like

```
begin ordina
var: A[100]: array of integer
var: N, i, k: integer
var: t: real
```

```
read N
```

```
for i=1, N do
  read A(i)
endfor
```

lettura
di un
array

```
for i = N-1, 1, -1 do
  for k = 1, i do
    if (A(k) > A(k+1)) then
      t = A(k)
      A(k) = A(k+1)
      A(k+1) = t
    endif
  endfor
endfor
```

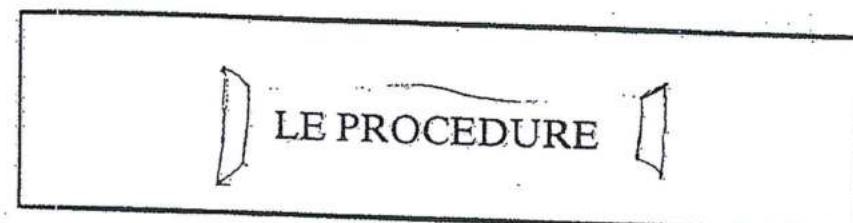
N-1 passi

scambi

```
for i = 1, N do
  print A(i)
endfor
```

stampa
di un
array

```
end ordina
```



Flow chart dell'algoritmo per il cambio della ruota



l'istruzione

avvita un bullone

può essere decomposta come

incastra la chiave esagonale sul bullone

gira la chiave di 90° in senso orario

il bullone
è stretto?

V

F

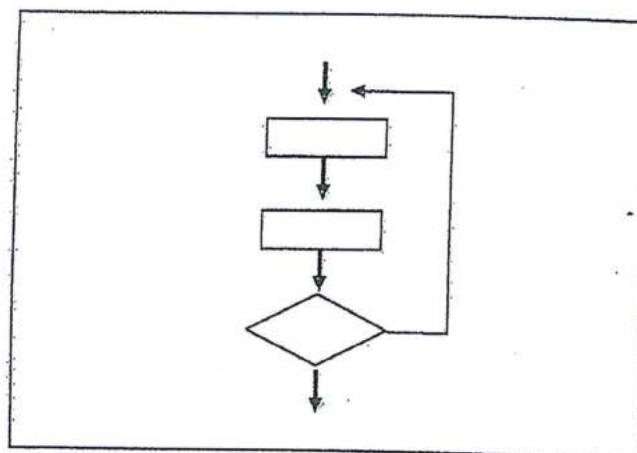
l'istruzione

avvia un bullone

può essere vista come un

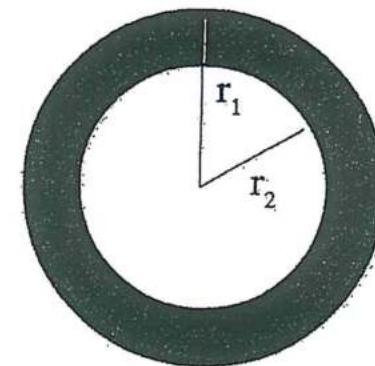
BLOCCO DI ISTRUZIONI

al cui interno vengono specificate
operazioni più semplici



PROBLEMA 1

calcolo dell'area della corona circolare



corona

$$= \frac{\text{area del cerchio di raggio } r_1}{\text{area del cerchio di raggio } r_2}$$

un algoritmo può essere:

```
begin corona_circolare
```

```
    var: r1, r2, area1, area2, corona, pigreco: real  
    read r1, r2  
    pigreco = 3.1415926
```

```
    area1 = pigreco × r1 × r1
```

```
    area2 = pigreco × r2 × r2
```

```
    corona = area1 - area2  
    print corona
```

```
end corona_circolare
```

un algoritmo può essere:

```
begin corona_circolare
```

```
    var: r1, r2, area1, area2, corona, pigreco: real  
    read r1, r2  
    pigreco = 3.1415926
```

{area1 = area del cerchio di raggio r1}

```
    area1 = pigreco × r1 × r1
```

{area2 = area del cerchio di raggio r2}

```
    area2 = pigreco × r2 × r2
```

```
    corona = area1 - area2  
    print corona  
end corona_circolare
```

PROBLEMA

nel calcolo della
corona circolare
è possibile 'richiamare'
l'algoritmo
per il calcolo dell'area del cerchio
?

algoritmo
che calcola l'area del cerchio:

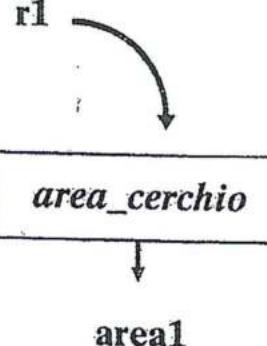
begin area_cerchio

var: r, area, pigreco; real
read r
pigreco = 3.1415926
area = pigreco × r × r
print area

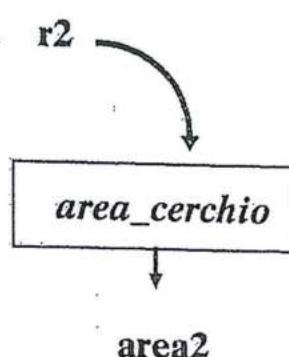
end area_cerchio

l'algoritmo ***area_cerchio*** è in grado di calcolare le aree dei due cerchi

I chiamata:

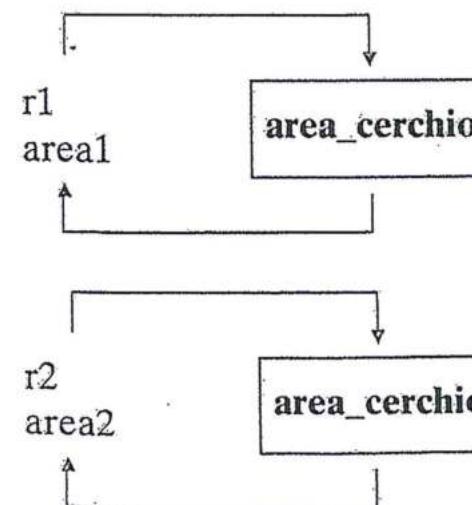


II chiamata:



begin corona_circolare

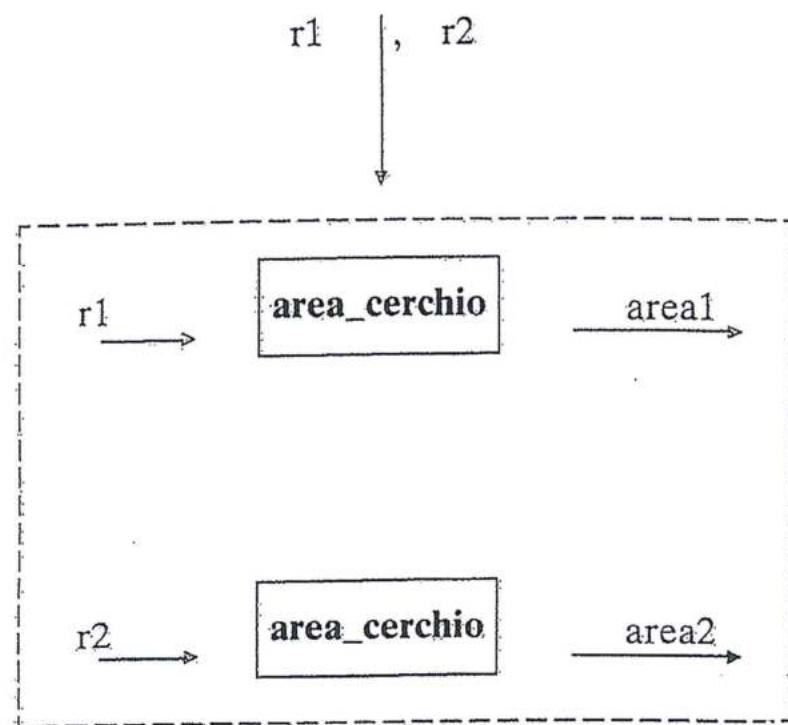
var: r1, r2, area1, area2, corona: real
read r1, r2



corona = area1 - area2
print corona

end corona_circolare

IDEA



PROBLEMA 2:

dati due array
a, b
di dimensione n

calcolare:

il massimo di ogni array
(max a, max b)

il massimo tra
max a, max b
(max)

begin massimo

var: a[100], b[100]: array of real

var: n, i: integer

var: maxa, maxb, max: real

read n

for i = 1,n do

 read a(i), b(i)

endfor

{calcola massimo di a (maxa)}

{calcola massimo di b (maxb)}

{calcola massimo tra i massimi (max)}

print max

end massimo

begin massimo

var: a[100], b[100]: array of real

var: n, i: integer

var: maxa, maxb, max: real

read n

for i = 1,n do

 read a(i), b(i)

endfor

{calcola massimo di a}

maxa = a(1)

for i = 2, n do

 if (maxa < a(i)) then

 maxa = a(i)

 endif

endfor

{calcola massimo di b}

maxb = b(1)

for i = 2, n do

 if (maxb < b(i)) then

 maxb = b(i)

 endif

endfor

{calcola massimo tra i massimi}

max = maxa

if (maxa < maxb) then

 max = maxb

endif

print max

end massimo

begin massimo

var: a[100], b[100]: array of real

var: n, i; integer

var: maxa, maxb, max: real

read n

for i = 1, n do

 read a(i), b(i)

endfor

{calcola massimo di a}

 maxa = a(1)

 for i = 2, n do

 if (maxa < a(i)) then

 maxa = a(i)

 endif

 endfor

{calcola massimo di b}

 maxb = b(1)

 for i = 2, n do

 if (maxb < b(i)) then

 maxb = b(i)

 endif

 endfor

{calcola massimo tra i massimi}

 max = maxa

 if (maxa < maxb) then

 max = maxb

 endif

print max

end massimo

• {calcola massimo di a}

```
maxa = a(1)
for i = 2, n do
    if (maxa < a(i)) then
        maxa = a(i)
    endif
endfor
```

• {calcola massimo di b}

```
maxb = b(1)
for i = 2, n do
    if (maxb < b(i)) then
        maxb = b(i)
    endif
endfor
```

• {calcola massimo tra i massimi}

```
max = maxa
if (max < maxb) then
    max = maxb
endif
```

massimo_vett

massimo_vett

massimo

begin massimo

var: a[100], b[100]: array of real

var: n, i: integer

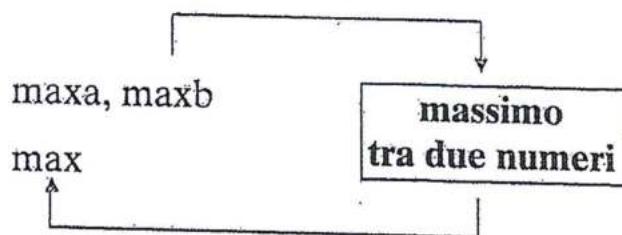
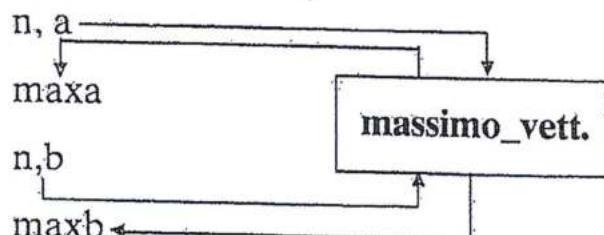
var: maxa, maxb, max: real

read n

for i = 1,n do

 read a(i), b(i)

endfor



end massimo

PROBLEMA 3:

dati tre vettori

a, b, c

di dimensione n

calcolare:

il massimo di ogni vettore
(maxa, maxb, maxc)

il massimo tra
maxa, maxb, maxc
(max)

begin massimo

var: a[100], b[100], c[100]: array of real

var: n, i: integer

var: maxa, maxb, maxc, max: real

read n

for i = 1,n do

 read a(i), b(i), c(i)

endfor

{calcola massimo di a (maxa)}

{calcola massimo di b (maxb)}

{calcola massimo di c (maxc)}

{calcola massimo tra i massimi (max)}

print max

end massimo

begin massimo

var: a[100], b[100], c[100]: array of real

var: n, i: integer

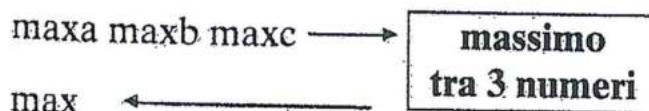
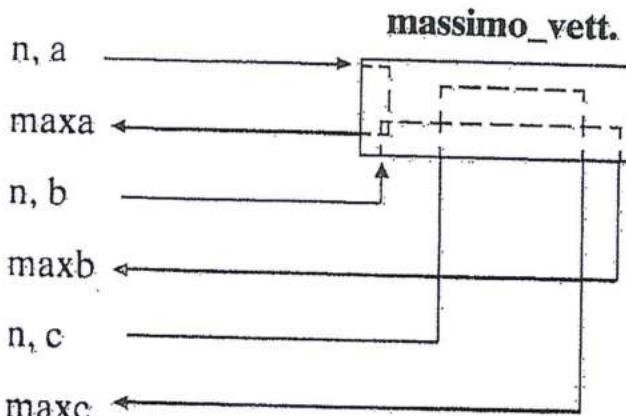
var: maxa, maxb, maxc, max: real

read n

for i = 1,n do

 read a(i), b(i), c(i)

endfor



print max

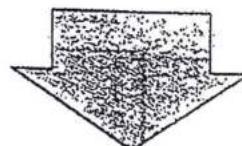
end massimo

OSSERVAZIONE

massimo tra maxa, maxb, maxc



massimo di un vettore di dim. 3



**richiamare l'algoritmo
massimo_vett
che determina il
massimo in un vettore**

begin massimo

var: a[100], b[100], c[100], massimi [3]:

array of real

var: n, i: integer

var: maxa, maxb, maxc, max: real

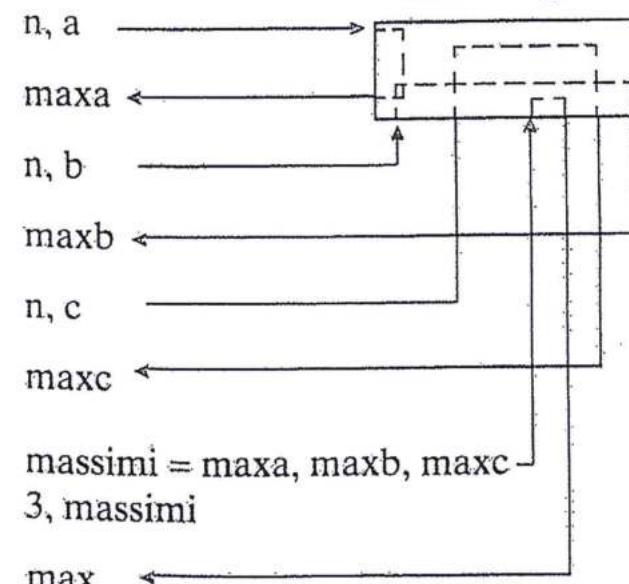
read n

for i = 1,n do

 read a(i), b(i), c(i)

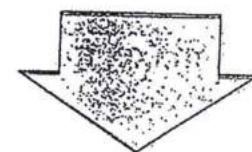
endfor

massimo_vett.



print max

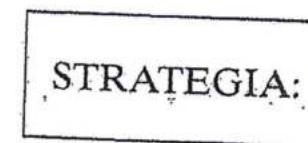
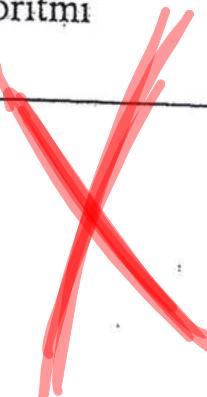
Gli esempi precedenti mostrano che parti di un algoritmo possono essere affidate ad opportuni "programmi"



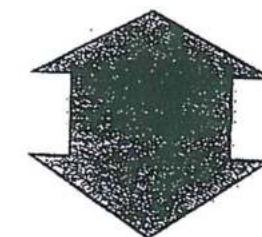
PROCEDURE

PROBLEMA

stabilire delle regole per la "comunicazione" tra i due algoritmi



- 1) scrivere un **algoritmo generale**
- 2) assegnare un nome a tale algoritmo
- 3) fare riferimento a tale algoritmo (attraverso il nome) quando serve

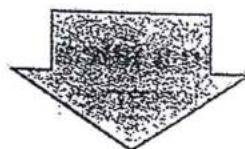


scrivere una PROCEDURA

Esempio: calcolo della corona circolare

area_cerchio
è il

NOME DELLA PROCEDURA



USO DELLA PROCEDURA
attraverso il nome

algoritmo per l'area della corona circolare mediante procedure

begin corona_circolare

var:r1, r2, area1, area2, corona: real

read r1, r2

area_cerchio (in: r1; out: area1)

area_cerchio (in: r2; out: area2)

corona = area1 - area2

print corona

end corona_circolare

begin massimo

```

var: a[100], b[100], c[100], massimi [3]:
      array of real
var: n, i: integer
var: maxa, maxb, maxc, max: real
read n
for i = 1,n do
    read a(i), b(i), c(i)
endfor

{calcola massimo di a}
massimo_vett (in: n, a; out: maxa)
massimi (1) := maxa;
{calcola massimo di b}
massimo_vett (in: n, b; out: maxb)
massimi (2) := maxb;
{calcola massimo di c}
massimo_vett (in: n, c; out: maxc)
massimi (3) := maxc;
{calcola massimo tra i massimi}
massimo_vett (in: 3, massimi; out: max)

```

print max

end massimo

l'istruzione

area_cerchio (in: r1; out: area1)

nell'algoritmo chiamante provoca
l'esecuzione della procedura

dall'algoritmo
chiamante

all'algoritmo
chiamante

procedure area_cerchio (in: raggio; out: area)

```

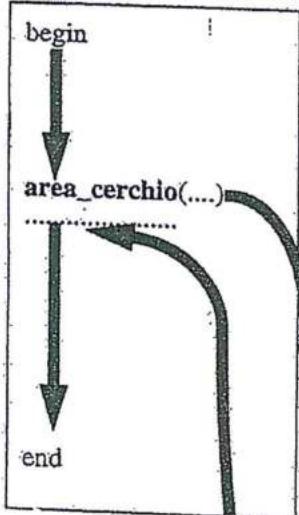
var: raggio, area, pigreco: real
pigreco = 3.1415926
area = pigreco * raggio * raggio

```

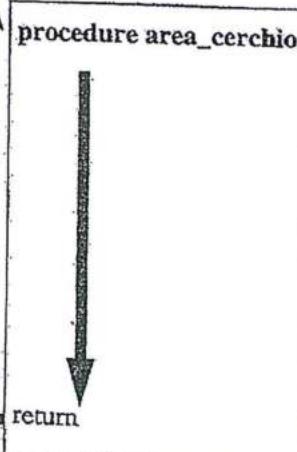
return
end

l'istruzione **return** rimanda il controllo
all'algoritmo chiamante

algoritmo chiamante



flusso delle istruzioni

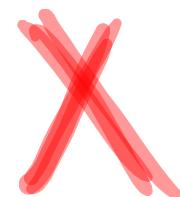


procedura

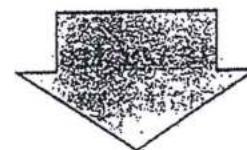
area_cerchio (in: r1; out:area1)
parametri attuali

procedure area_cerchio (in: raggio; out:area)
parametri formali

- La corrispondenza tra parametri attuali e parametri formali è per posizione
- È necessaria la corrispondenza del tipo.
- I parametri costituiscono un canale di comunicazione tra l'algoritmo principale e la procedura.



I valori dei parametri attuali
nell'algoritmo chiamante
sono associati ai parametri formali
all'interno della procedura



- i parametri **attuali di input** devono essere definiti nell'algoritmo chiamante
- i parametri **formali di input** non devono essere definiti nella procedura
- i parametri **formali di output** devono essere definiti nella procedura

ESEMPIO:

calcolo dell'area della
corona circolare

prima *chiamata*:

area1=314.1592

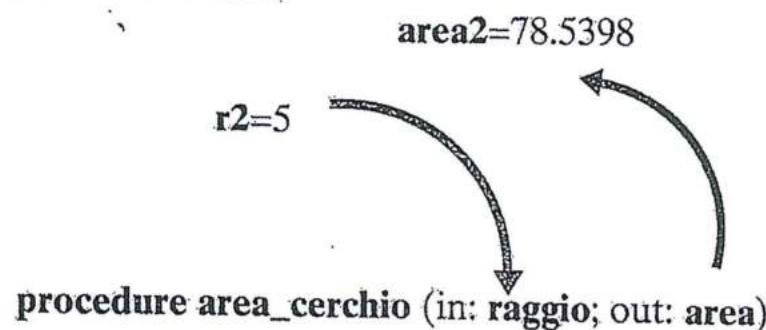


procedure area_cerchio (in: raggio; out: area)

IN MEMORIA

r1	10	← raggio
area1	314.1592	← area
r2		
area2		

seconda chiamata:

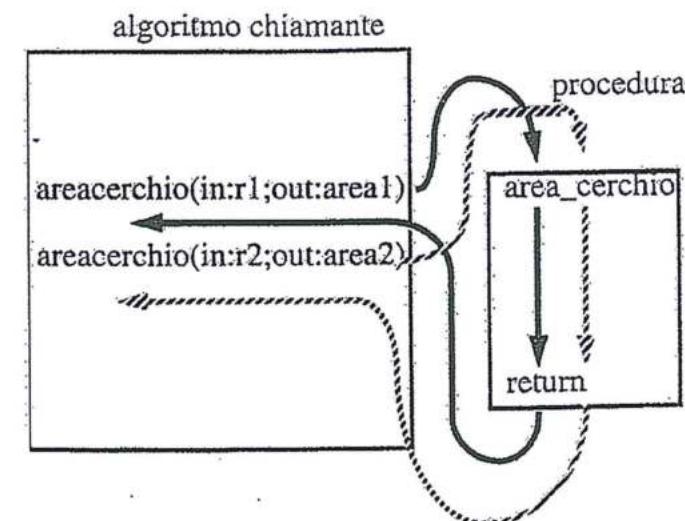


IN MEMORIA

r1	10
area1	314.1592
r2	5
area2	78.5398

raggio

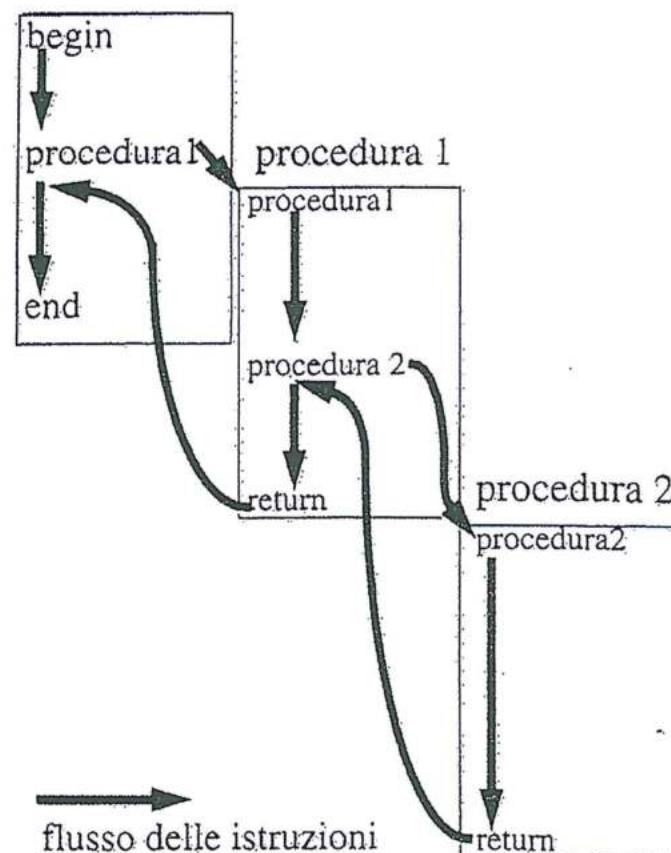
area



SOLID LINE prima chiamata

DASHED LINE seconda chiamata

le procedure possono essere innestate
l'una nell'altra



una **PROCEDURA** si può considerare
un **nome** che denota un **algoritmo**

- possibilità di essere richiamate più volte
- possibilità di innestare più livelli di procedura



le procedure rappresentano
moduli (o scatole nere) per sviluppare
algoritmi complessi

ESEMPIO:

costruzione di una automobile

preparare la carrozzeria

montare il motore

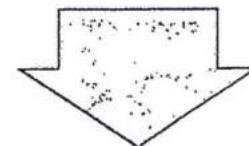
montare il cambio

montare l'impianto elettrico

montare le parti mobili (ruote, sportelli, ...)

montare le rifiniture (sedili, cruscotto, ...)

ogni fase costituisce un problema
a sé stante



ognuna di tali parti può essere affidata
ad una procedura

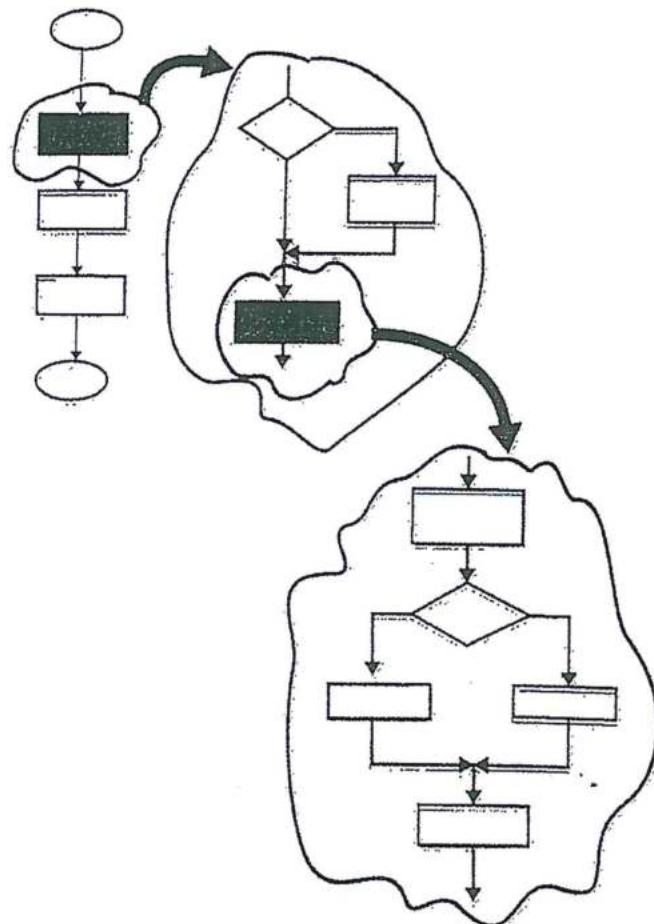
ESEMPIO:

montare il motore

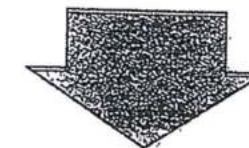


montare i cilindri
montare le valvole di iniezione
montare l'albero di trasmissione
montare l'impianto di raffreddamento
montare lo spinterogeno
montare il motorino di avviamento
.....

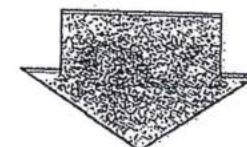
sviluppo MODULARE degli algoritmi



Algoritmo



insieme di procedure connesse
mediante strutture di controllo



ogni procedura può a sua volta
essere composta da più procedure

TECNICA DEI RAFFINAMENTI SUCCESSIVI

- a partire dal problema si scrive un algoritmo usando istruzioni non elementari
- si decompone ciascuna istruzione non elementare fino ad avere solo istruzioni elementari

LA COMPLESSITÀ COMPUTAZIONALE

Esempio:

Risolvere l'equazione

$$7x = 21$$

in un sistema aritmetico floating-point con

$$\beta=10 \quad e \quad t=6$$

algoritmo 1

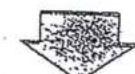
$$x = \frac{21}{7} = 3$$



1 divisione

algoritmo 2

$$x = 21 \times \frac{1}{7} = \\ 21 \times 0.142857 = \\ 0.299997 \times 10.$$



1 molt. + 1 div.



L'algoritmo 1 è più efficiente

Esempio:

Calcolare $f(x) = x^{16}$

algoritmo 1

$$1) \quad x^2 = x \cdot x$$

$$2) \quad x^3 = x^2 \cdot x$$

$$3) \quad x^4 = x^3 \cdot x$$

$$15) \quad x^{16} = x^{15} \cdot x$$

algoritmo 2

$$1) \quad x^2 = x \cdot x$$

$$2) \quad x^4 = x^2 \cdot x^2$$

$$3) \quad x^8 = x^4 \cdot x^4$$

$$4) \quad x^{16} = x^8 \cdot x^8$$



15 moltiplicazioni

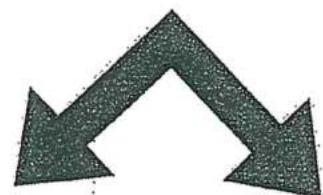


4 moltiplicazioni



L'algoritmo 2 è più efficiente

In generale, calcolare
 $f(x) = x^n$, $n = 2^m$
 richiede

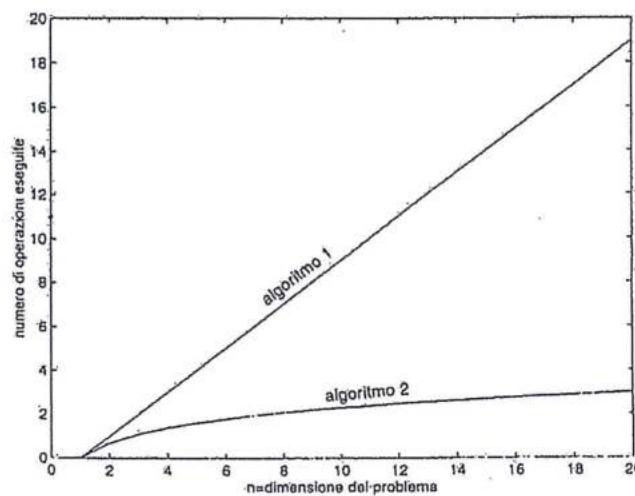


Algoritmo 1

$n-1$ operazioni.
 moltiplicando x per
 se stesso
 ripetutamente

Algoritmo 2

$\log_2 n$ operazioni
 elevando x al
 quadrato
 ripetutamente



confronto tra il numero di operazioni
 eseguite per il calcolo di x^n ,
 al variare di n ,
 dall'algoritmo 1 e dall'algoritmo 2

Problema

Valutare per un fissato valore di x il polinomio,

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

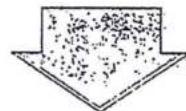
Si può procedere nel seguente modo:

$$p_1 = a_3 \cdot x \cdot x \cdot x \quad 3M$$

$$p_2 = a_2 \cdot x \cdot x \quad 2M$$

$$p_3 = a_1 \cdot x \quad 1M$$

$$p_1 = p_1 + p_2 + p_3 + a_0 \quad 3A$$



6 moltiplicazioni e 3 addizioni

M=1 moltiplicazione o divisione
A=1 addizione o sottrazione

Caso generale

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

Algoritmo 1

```

p:=a0

for i := n, 1, -1 do

    y:=x

    for j := 2, i do

        y:=y*x

    end for

    p:=p+ai*y

end for
```

$$(n + (n-1) + \dots + 1) M = \frac{n(n+1)}{2} M$$

$$\underbrace{(1 + 1 + \dots + 1)}_n A = nA$$

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

un procedimento più efficiente:

$$y_1 = x \cdot x \quad 1M$$

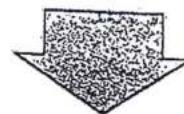
$$y_2 = y_1 \cdot x \quad 1M$$

$$p_1 = a_3 \cdot y_2 \quad 1M$$

$$p_2 = a_2 \cdot y_1 \quad 1M$$

$$p_3 = a_1 \cdot x \quad 1M$$

$$p_4 = p_1 + p_2 + p_3 + a_0 \quad 3A$$



5 moltiplicazioni e 3 addizioni

Caso generale

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

Algoritmo 2

```

p:=a0
y:=x
for j:= 1, n, do
    if (i≠1) then
        y := y*x
    endif
    p:=p+ai*y
end for

```

$$(1 + 2(n-1)) M = (2n - 1) M$$

$$(1 + 1 \dots + 1) A = nA$$

Un procedimento ancora più efficiente:

$$((a_3x + a_2)x + a_1)x + a_0 = a_3x^3 + a_2x^2 + a_1x + a_0$$

$$y_1 = a_3 \cdot x \quad 1M$$

$$y_2 = y_1 \cdot a_2 \quad 1A$$

$$y_3 = y_2 \cdot x \quad 1M$$

$$y_4 = y_3 \cdot a_1 \quad 1A$$

$$y_5 = y_4 \cdot x \quad 1M$$

$$y_6 = y_5 + a_0 \quad 1A$$



3 moltiplicazioni e 3 addizioni

Caso generale

Algoritmo di HORNER

$$p := a_n$$

for i := $n - 1, 0, -1$, do

$$p := p * y + a_i$$

end for

$$nM + nA$$

L'Algoritmo di Horner è il più efficiente

L'efficienza di un algoritmo
dipende dal numero di
operazioni richieste per ottenere
la soluzione del problema

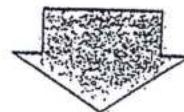
Problema

Memorizzazione di una matrice
“sparsa”

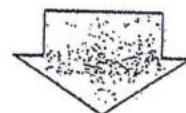
$$A = \begin{pmatrix} 7 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -6 \\ -3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ -1 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 8 & 0 & 3 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

A: matrice 8×8

Memorizzazione mediante array bidimensionale



Sono necessarie $8 \times 8 = 64$ parole



Spreco di memoria

Memorizzazione mediante 3 vettori

- $A' = (7, -1, 2, 8, 1, -6, -3, 2, 4, -1, 7, 9, 8, 3, 2)$
 A' = elementi non nulli di A riga per riga
- $J = (1, 4, 1, 2, 5, 8, 1, 7, 6, 1, 4, 1, 5, 7, 3)$
 J : indice di colonna in A dell'elemento $A'(i)$
- $I = (1, 3, 5, 7, 9, 10, 12, 15, 16)$
 $I(i)$: posizione in A' del primo elemento non nullo della i -ma riga di A ; $I(9) = \#$ elementi di A' + 1



Sono necessarie
15 + 15 + 9 = 39 locazioni



Risparmio di memoria

Un algoritmo che utilizza
il secondo schema
di memorizzazione è più efficiente
di un algoritmo che utilizza il primo

L'efficienza di un algoritmo
dipende dallo spazio richiesto
dai dati su cui opera

Per valutare l'**efficienza**
di un algoritmo si misurano:

- il **numero di operazioni** effettuate
- lo spazio di memoria richiesto dai dati (di input, intermedi e di output)

=

Costo computazionale
dell'algoritmo

In generale
per misurare il **costo computazionale**
di un algoritmo si definiscono

- una funzione
complessità di tempo $T(n)$
- una funzione
complessità di spazio $S(n)$

n = dimensione del problema

$T(n)$ è il numero delle operazioni più significative che si effettuano nell'esecuzione dell'algoritmo

Il tempo di esecuzione di un algoritmo τ , è proporzionale a $T(n)$.

$$\tau = \kappa \cdot T(n) \cdot \mu \quad (k = \text{cost})$$

μ è funzione del periodo di clock

Il periodo di clock dipende dal tempo di risposta dei componenti elementari del calcolatore.

Il tempo di esecuzione di un algoritmo dipende:

- dall'algoritmo (attraverso $T(n)$)
 - dal calcolatore (attraverso μ)

$S(n)$ è il numero delle variabili utilizzate dall'algoritmo

Lo spazio di memoria occupato dai dati di input, intermedi e di output è proporzionale a $S(n)$

Esempio

Valutazione di

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

- Algoritmo 1

$$T(n) = \frac{n(n+1)}{2} M + nA$$

$$S(n) = n + 4$$

- Algoritmo 2

$$T(n) = (2n-1) M + nA$$

$$S(n) = n + 4$$

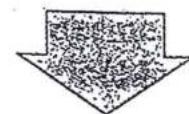
- Algoritmo di Horner

$$T(n) = nM + nA$$

$$S(n) = n + 3$$

Hostrowski (1954) ha dimostrato che
sono necessarie almeno
 n moltiplicazioni e n addizioni
per valutare un polinomio
di grado $n \leq 4$

(In seguito tale risultato è stato esteso
ai polinomi di grado qualsiasi)



L'algoritmo di Horner
è ottimale

In generale

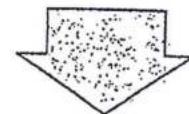
il costo computazionale
di un algoritmo si
valuta al crescere
della dimensione
del problema

Esempio

Algoritmo 1

$$T(n) = \left(\frac{n^2}{2} + \frac{n}{2} \right) M + nA$$

n	$T(n)$
2	$(2+1)M+2A$
10	$(50+5)M+100A$
100	$(5000+50)M+100A$
1000	$(500000+500)M+100A$



al crescere di n

il termine dominante è $\frac{n^2}{2}$

Esempio

Algoritmo di Horner

$$T(n) = nM + nA$$

n	$T(n)$
2	$2M+2A$
10	$10M+10A$
100	$100M+100A$
1000	$1000M+100A$



al crescere di n
il termine dominante è n

In generale si studia
il comportamento delle funzioni
; $T(n)$ e $S(n)$ al crescere di n
(complessità asintotica)

ovvero

si considerano
i termini dominanti per $n \rightarrow \infty$,

(generalmente trascurando
le costanti moltiplicative)

Esempio

Algoritmo 1

$$T(n) = \left(\frac{n^2}{2} + \frac{n}{2} \right) M + nA$$

$\frac{n^2}{2}$ = termine dominante

$$T(n) = O(n^2)$$

n^2 complessità asintotica

Algoritmo di Horner

$$T(n) = nM + nA$$

n = termine dominante



$$T(n) = O(n)$$

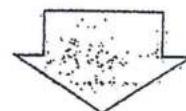
n complessità asintotica

In generale si pone:

$$T(n) = O(f(n)) \quad (f(n) > 0)$$

se

$$\lim_{n \rightarrow +\infty} \frac{T(n)}{f(n)} = cost \neq 0$$



$f(n)$ complessità di tempo asintonica

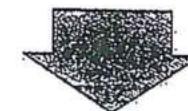
Problema

Determinare un limite superiore
ed inferiore per la
complessità asintotica di un algoritmo

Esempio

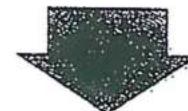
Algoritmo 1

$$T(n) = \left(\frac{n^2}{2} + \frac{n}{2} \right) M + nA$$



$$T(n) \leq n^2 \quad \forall n \geq 2$$

$$(T(2) = 5, T(3) = 9, T(4) = 14)$$



n^2 limite superiore per $T(n)$

In generale
se esistono due costanti $c, n_0 > 0$ tali che:

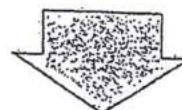
$$T(n) \leq cf(n) \quad \forall n \geq n_0$$

e

$$\forall g(n) : T(n) \leq g(n) \quad \forall n \geq n_g$$

si ha

$$g(n) \geq cf(n) \quad \forall n \geq n_g$$



$cf(n)$ limitazione superiore per $T(n)$

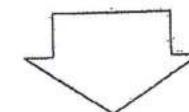
Analogamente, se esistono
due costanti $c, n_0 > 0$ tali che:

$$T(n) \leq ch(n), \quad \forall n \geq n_0$$

$$\forall g(n) : T(n) \geq g(n) \quad \forall n \geq n_g$$

si ha

$$g(n) \leq ch(n), \quad \forall n \geq n_g$$



$ch(n)$ limitazione superiore per $T(n)$

Limitazioni sulle dimensioni dei problemi risolvibili
da algoritmi con alcuni valori tipici di $T(n)$

compl. di tempo	massima dimensione del problema risolvibile in		
$T(n)$	1 sec	1 min	1 ora
$\log_2 n$	2^{10^6}	$2^{6 \times 10^7}$	$2^{3.6 \times 10^9}$
n	10^6	6×10^7	3.6×10^9
$n \log_2 n$	6.3×10^4	2.8×10^6	1.3×10^8
n^2	10^3	7.7×10^3	6×10^4
n^3	10^2	3.9×10^2	1.5×10^3
2^n	20	25	32
$n!$	9	11	12

su un calcolatore con velocità di 1 Mflops
(10^6 operazioni floating-point al sec.)

PROBLEMA:

ordinamento (in senso crescente) di un array

DATI DI INPUT: array

DATI DI OUTPUT: array ordinato

ESEMPIO 1

DATI DI INPUT: (8, 6, 5, 4)

DATI DI OUTPUT: (4, 5, 6, 8)

8	6	5	4
---	---	---	---

vettore di input

passo 1

6	8	5	4
---	---	---	---

confr. scambi

1 1

6	5	8	4
---	---	---	---

1 1

6	5	4	8
---	---	---	---

1 1

passo 2

5	6	4	8
---	---	---	---

1 1

5	4	6	8
---	---	---	---

1 1

passo 3

4	5	6	8
---	---	---	---

1 1

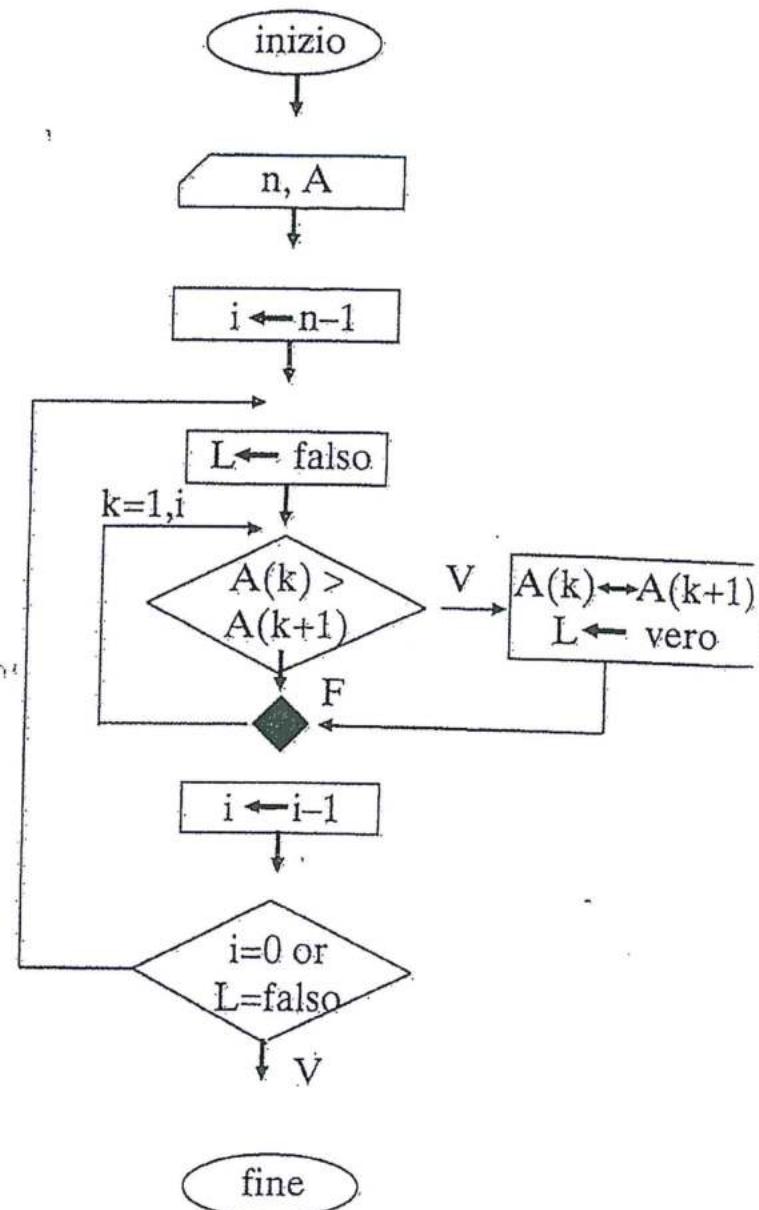
TOT. 6 6

IDEA

ad ogni passo,
spostare il massimo
nell'ultima posizione

STRATEGIA

ad ogni passo, **confrontare** a due
a due coppie consecutive di
elementi e **scambiarli** di posto se
il maggiore dei due si trova
a sinistra



ORDINAMENTO

begin ordina
var: A[100]: array di interi
var: L: logica
var: i, k, t, n: interi
read n
for i := 1, n do
 read A (i)
endfor
i:=n-1
repeat
 L:=false
 for k:=1,i do
 if (A(k) > A(k+1)) then
 t:=A(k)
 A(k):=A(k+1)
 A(k+1):=t
 L:= true
 endif
 endfor
 i:=i-1
until (i=0 or (not L))
end ordina

La complessità di tempo dipende
dal numero di confronti e di scambi

Nel caso di un array di lunghezza n
con gli elementi in ordine
decrescente (caso peggiore)
sono necessari

$$\underbrace{(n-1)}_{\text{passo 1}} + \underbrace{(n-2)}_{\text{passo 2}} + \dots + \underbrace{1}_{\text{passo } n-1} \text{ confronti}$$

e

$$\underbrace{(n-1)}_{\text{passo 1}} + \underbrace{(n-2)}_{\text{passo 2}} + \dots + \underbrace{1}_{\text{passo } n-1} \text{ scambi}$$



$$\frac{n(n-1)}{2} \text{ confronti e scambi}$$

Esempio 2

1	2	3	4
---	---	---	---

vettore di input

passo 1

confr. scambi

1	2	3	4
---	---	---	---

1 0

1	2	3	4
---	---	---	---

1 0

1	2	3	4
---	---	---	---

1 0

TOT. 3 0

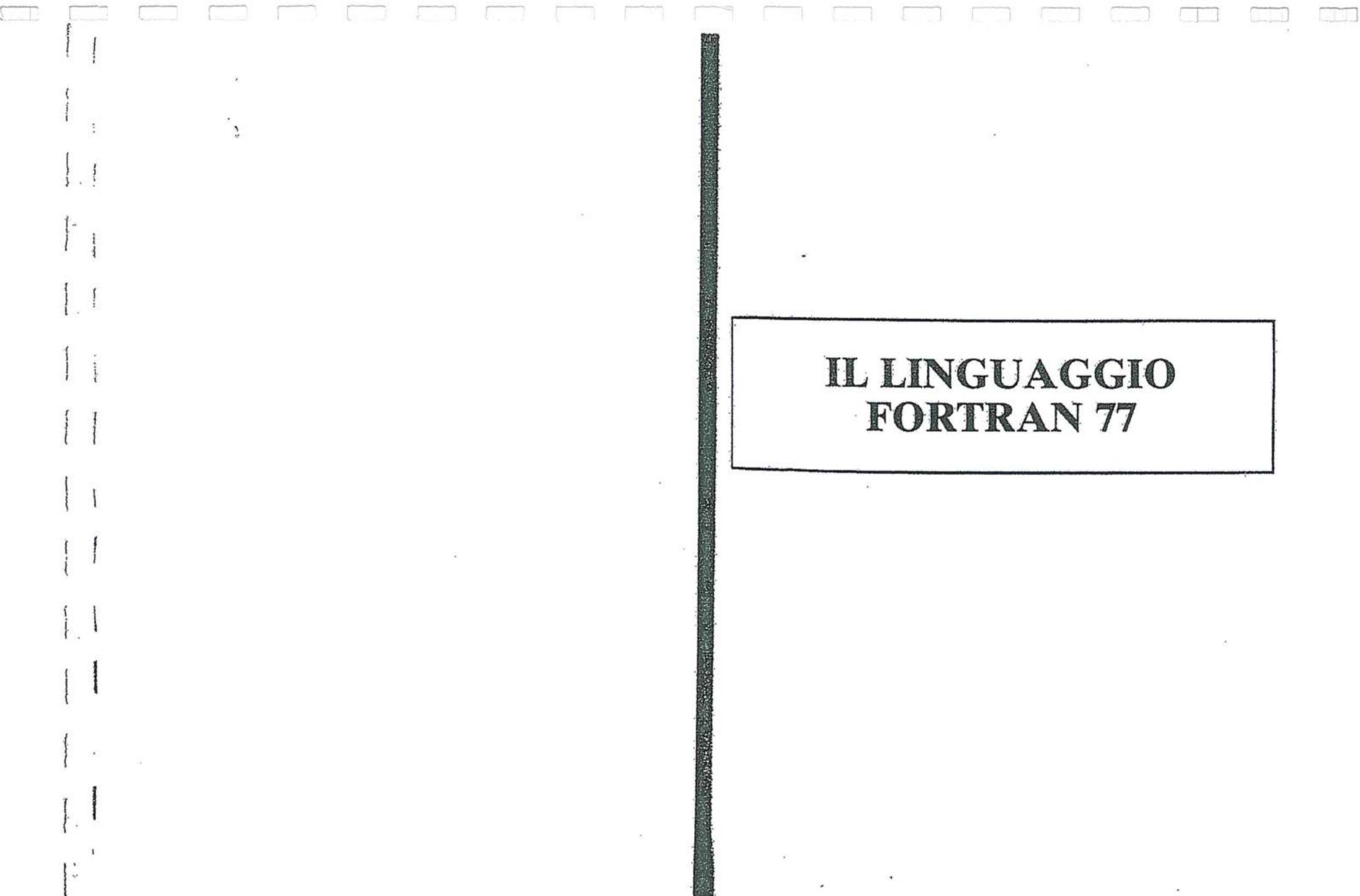
Nel caso di un array di lunghezza n con gli elementi in ordine crescente
(caso migliore)
sono necessari

$(n-1)$ confronti
e
0 scambi

Per alcuni algoritmi la complessità
di tempo dipende dai dati di input
oltre che
dalla dimensione del problema



la complessità di tempo è
compresa fra quella del caso
migliore e quella del caso peggiore



IL LINGUAGGIO FORTRAN 77

FORmula TRANslator
Linguaggio di programmazione
ad alto livello
di tipo *compilativo*.
Versione: ANSI FORTRAN 77
(American National Standard Institute)

1953: definizione di un sistema di programmazione sull'elaboratore IBM-704
1954: IBM Mathematical Formula Translator system: FORTRAN
1960: versioni del compilatore FORTRAN sono disponibili per diversi elaboratori
1962: definizione di uno standard da parte dell'ANSI
1966: ANSI FORTRAN 66
1978: ANSI FORTRAN 77
1990: ANSI FORTRAN 90
2000: previsione della nuova versione standard ANSI

Problema

Calcolo della lunghezza di una circonferenza di raggio r ($l = 6.28 * r$)

Algoritmo (*Pascal-like*)

```

begin circonferenza
{calcolo circonferenza}
var: r, l: real
read r
if (r<=0) then
    print 'errore di input'
else
    l=6.28*r
    print 'lunghezza:', l
endif
end circonferenza

```

Qual è il corrispondente
programma Fortran?

Programma *Fortran*
per il calcolo della lunghezza
di una circonferenza

```

PROGRAM CIRCONFERENZA
C CALCOLO CIRCONFERENZA
REAL R, L
READ*, R
IF (R.LE.0) THEN
    PRINT*, 'ERRORE DI INPUT'
ELSE
    L=6.28*R
    PRINT*, 'LUNGHEZZA=', L
ENDIF
STOP
END

```

Programma *Fortran*
per il calcolo della lunghezza
di una circonferenza

1 67

```
C      PROGRAM CIRCONFERENZA
          CALCOLO CIRCONFERENZA
          REAL R, L
          READ*, R
          IF (R.LE.0) THEN
              PRINT*, 'ERRORE DI INPUT'
          ELSE
              L=6.28*R
              PRINT*, 'LUNGHEZZA=', L
          ENDIF
          STOP
          END
```

Istruzione PROGRAM

PROGRAM CIRCONFERENZA

è l'istruzione che dà il nome
CIRCONFERENZA
al programma

tale istruzione è facoltativa e serve solo a distinguere un programma dall'altro;

quando è presente, deve precedere qualsiasi altra istruzione

Istruzioni STOP ed END

tali istruzioni
indicano la fine del programma.

STOP indica al compilatore la *fine logica* del programma.

END indica al compilatore la *fine fisica* del programma (termine della traduzione in linguaggio macchina).

Dichiarazione delle variabili

Fortran
REAL R, L

Pascal-like
var:r, l: real

Questa istruzione indica al compilatore di riservare due parole di memoria, rispettivamente per le variabili R ed L, di *tipo reale*.

(allocazione in memoria delle variabili R e L)

In FORTRAN sono ammessi i seguenti *identificativi di tipo*:

REAL	reale
INTEGER	intero
LOGICAL	logico
CHARACTER	alfanumerico

Operazioni di I/O

READ*, R

indica che

- si vuole leggere un valore dall'unità standard di input (la tastiera),
- il valore letto è assegnato ad R.

Analogamente

PRINT*, L

indica che

- si vuole stampare un valore sull'unità standard di output (il video),
- il valore da stampare è contenuto nella variabile L.

Istruzione di assegnazione

espressione aritmetica

$$L = \underbrace{6.28}_{\substack{\text{costante di} \\ \text{tipo reale}}} * \underbrace{R}_{\substack{\text{operatore di moltiplicazione}}}$$

Tale istruzione consiste nella

- valutazione dell'espressione aritmetica
- memorizzazione del risultato nella parola di memoria associata alla variabile L

Il risultato della valutazione dell'espressione deve essere dello *stesso tipo* della variabile a cui è assegnato.

Costanti

6.28 è una costante di tipo reale.

Una costante reale è un numero reale rappresentabile in memoria.

Essa si scrive come

una sequenza di cifre decimali preceduta eventualmente dal segno e ***contenente necessariamente il punto decimale***

es.: 5.3 -4.776 +2.

oppure

utilizzando la ***notazione esponenziale***, con le precedenti convenzioni per la mantissa

es.: -88.92e4 5.e-3 0.001e+2

0 è una costante di tipo intero.

Una **costante intera** è un numero intero rappresentabile in memoria.

Essa si scrive come

una sequenza di cifre decimali preceduta eventualmente dal segno e **non contenente il punto decimale**

es.: 1125 -477 +2

.TRUE., .FALSE.
sono costanti logiche.

'area circonferenza'
è una costante alfanumerica

una costante alfanumerica è
una qualsiasi stringa di caratteri
racchiusa tra apici.

Selezione

Fortran

```
IF (R .LE. 0) THEN
    PRINT*, 'ERRORE DI INPUT'
ELSE
    L=6.28*R
    PRINT*, L
ENDIF
```

Pascal like

$r \leq 0$

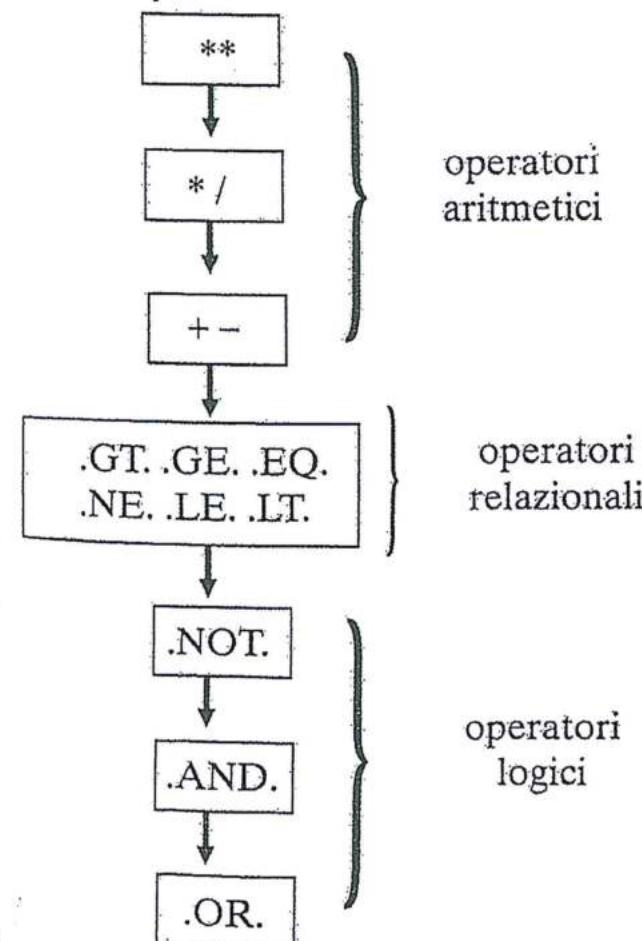
Fortran
R .LE. 0

espressione logica
(restituisce il valore vero o falso)

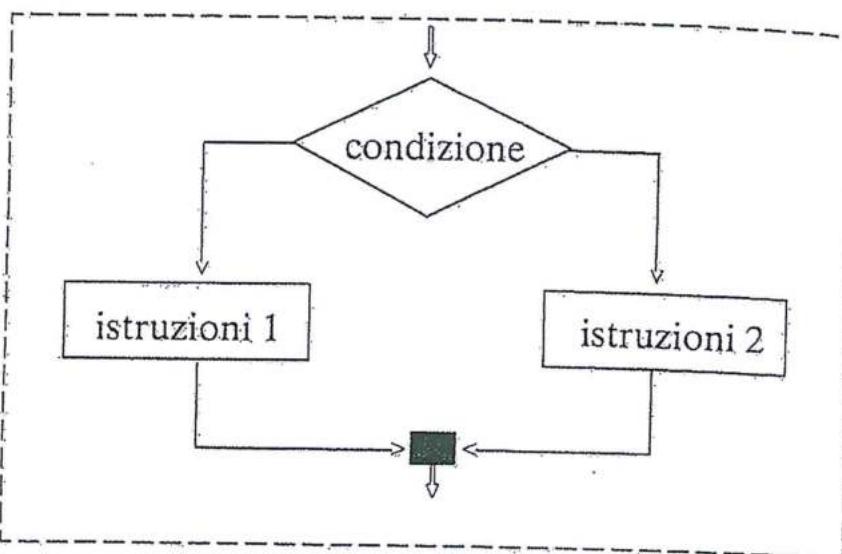
Priorità degli operatori

le espressioni sono valutate da sinistra verso destra

viene rispettata la priorità delle operazioni aritmetiche e delle parentesi



Selezione Flowchart



Pascal-like

```
if (condizione) then
  istruzioni 1
else
  istruzioni 2
endif
```

Fortran

```
IF (condizione)THEN
  istruzioni 1
ELSE
  istruzioni 2
ENDIF
```

Struttura di iterazione for-end-for

Problema:
Somma dei primi 500 numeri naturali

Pascal-like

```
begin s500
var; sum, n: integer
begin
sum=0
for n=1, 500
  sum=sum+n
endfor
print sum
end
end s500.
```

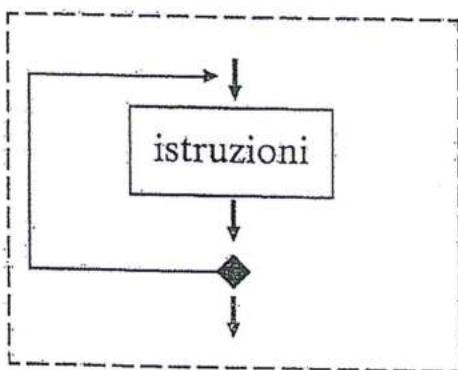
Fortran

1 6.7

```
PROGRAM S500
INTEGER N, SUM
SUM=0
DO 10 N=1, 500
  SUM=SUM+N
CONTINUE
PRINT*, SUM
STOP
END.
```

10

Iterazione for-endfor
Flowchart



Pascal-like

```

for i=1, n step k
  istruzioni
endfor
  
```

Fortran

```

DO <label>, I=1, N, K
  istruzioni
<label> CONTINUE
  
```

Cosa accade se non è noto a priori
il numero degli elementi da sommare?



Struttura di iterazione di tipo

repeat-until

Struttura di iterazione di tipo

while-endwhile

Struttura di iterazione

repeat-until

Pascal-like

```

begin somma
var: sum, n: intero
begin
read n
sum:=0
repeat
sum:=sum+n
until n=0
print sum
end
end s500
  
```

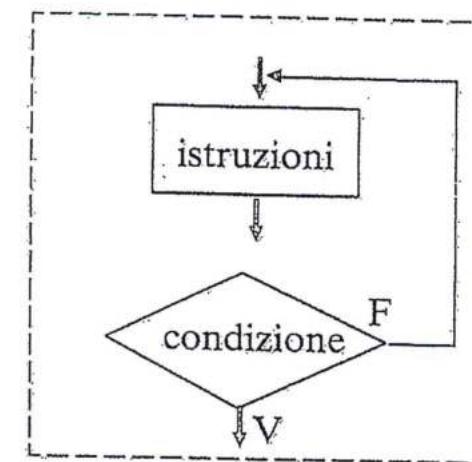
l 67

Fortran

```

PROGRAM SOMMA
INTEGER N, SUM
READ*, N
SUM=0
CONTINUE
SUM=SUM+N
IF (N.NE.0 GOTO 5)
PRINT*, SUM
STOP
END
  
```

5

Iterazione repeat-until
Flowchart*Pascal-like*

```

repeat
  istruzioni
until <condizione>
  
```

Fortran

```

<label> CONTINUE
  istruzioni
IF (.NOT. condizione) GOTO <label>
  
```

Struttura di iterazione

repeat-until

Pascal-like

```
begin somma2
var: sum, n: intero
begin
  read n
  sum:=0
  while n>0
    sum:=sum+n
    read n
  endwhile
  print sum
end
end somma2
```

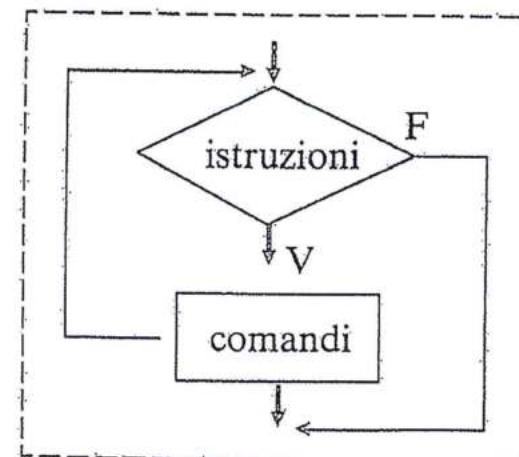
Fortran

1 67

```
PROGRAM SOMMA2
INTEGER N, SUM
READ*, N
SUM=0
IF (N .LE. 0) GOTO 20
SUM=SUM+N
READ*, N
GOTO 10
CONTINUE
PRINT*, SUM
STOP
END
```

Iterazione while-endwhile

Flowchart



Pascal-like

```
while condizione do
  comandi
endwhile
```

Fortran

```
<label1> IF (.NOT. <condizione>) GOTO <label2>
      istruzioni
      GOTO <label1>
<label2> CONTINUE
```

Per tradurre i costrutti di controllo

while ... endwhile

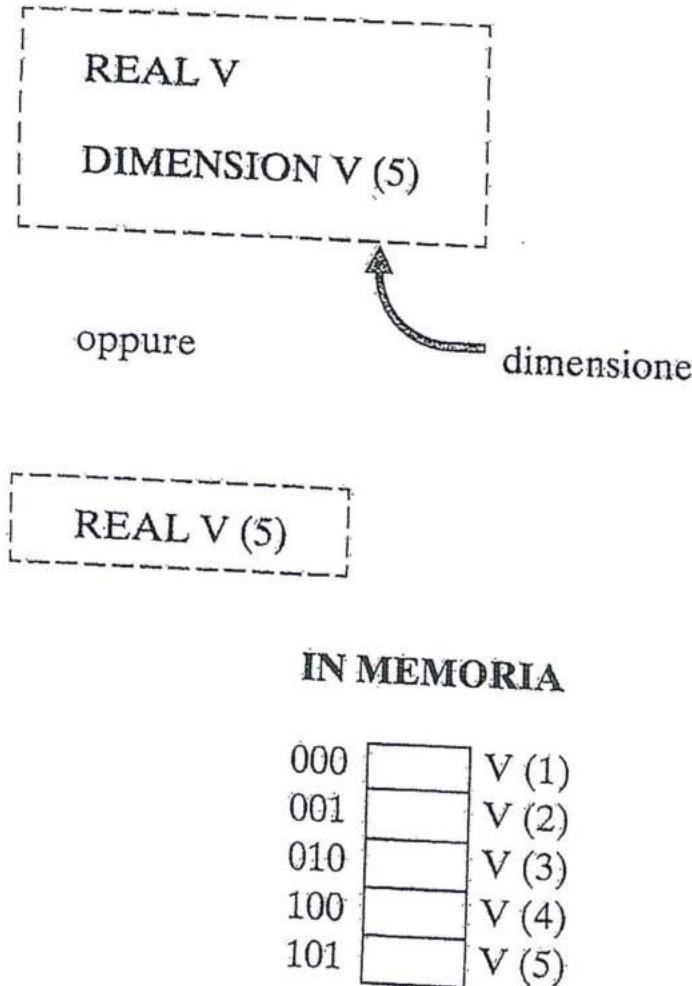
repeat ... until

si combinano opportunamente:

- IF logico
IF (*condizione*) istruzione
- GOTO incondizionato:
GOTO <*label*>

ARRAY
IN
FORTRAN

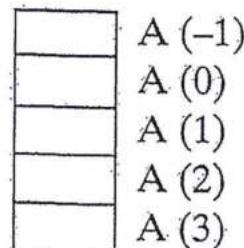
Array monodimensionali
dichiarazione in *Fortran*



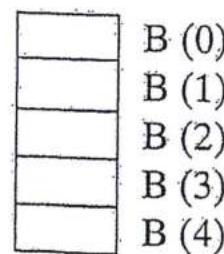
È possibile specificare l'*insieme di variabilità* degli indici

Esempio:

REAL A (-1:3)



REAL B (0:4)



```

REAL A(5)
.....
.....
N = 6
DO 10 I=1, N
    A(I)= 0.      ERRATO!
10 CONTINUE

```

*Non è possibile fare riferimento
ad una componente
che non è stata dichiarata*

I=3	J=2	IPIV (I) = 2	A (IPIV (I)) = 5	A (I+J) = 6.
-----	-----	--------------	------------------	--------------

IPIV

		2		...
--	--	---	--	-----

A

	5			6
--	---	--	--	---

Gli indici possono essere
espressioni intere
contenenti costanti e/o variabili

Array bidimensionali

Esempio:

$$A = \begin{pmatrix} 1.2 & 1 & 0.4 \\ 3.1 & 4.5 & 5.0 \\ 0.01 & 1.15 & 0.0 \end{pmatrix}$$

dichiarazione in *Fortran*

REAL A (3,3)

oppure

REAL A

DIMENSION A (3,3)

IN MEMORIA:

0000	1.2	A (1,1)
0001	3.1	A (2,1)
0010	0.01	A (3,1)
0100	1	A (1,2)
1000	4.5	A (2,2)
1001	1.15	A (3,2)
1010	0.4	A (1,3)
1100	5.0	A (2,3)
1101	0.0	A (3,3)

prima colonna

seconda colonna

terza colonna

REAL A(5,5)

.....

.....

.....

N = 6

DO 20 J=1, N

DO 10 I=1, N

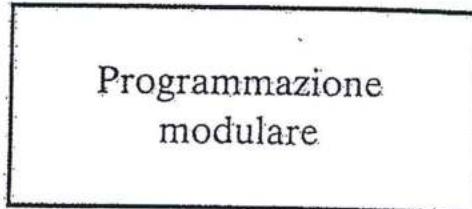
A(I,J)=I+J ERRATO!

10 CONTINUE

20 CONTINUE



*Non è possibile fare riferimento
ad una componente
che non è stata dichiarata*



Pascal-like

procedure



Fortran

sottoprogrammi
di tipo function
e subroutine

Esempio

Pascal-like

```
begin
var: a, b: real
read r1, r2
    circ(r1, a)
    circ(r2, b)
    print a, b
end
```

```
procedure circ (r,l)
var: r,l:real
begin
    l=6.28×r;
    return l
end
```

Sottoprogrammi di tipo
FUNCTION

FORTRAN**Esempio****Programma principale**

```

REAL A, B, CIRC, R1, R2
READ*, R1, R2
A=CIRC(R1)
B=CIRC(R2)
PRINT*, A, B
STOP
END

```

Sottoprogramma di tipo function

```

REAL FUNCTION CIRC (RAG)
REAL RAG
CIRC=6.28*RAG
RETURN
END

```

CHIAMATA ALLA FUNCTION**programma chiamante**

```

.....
.....
A = CIRC (R1)
.....

```

sottoprogramma di tipo function

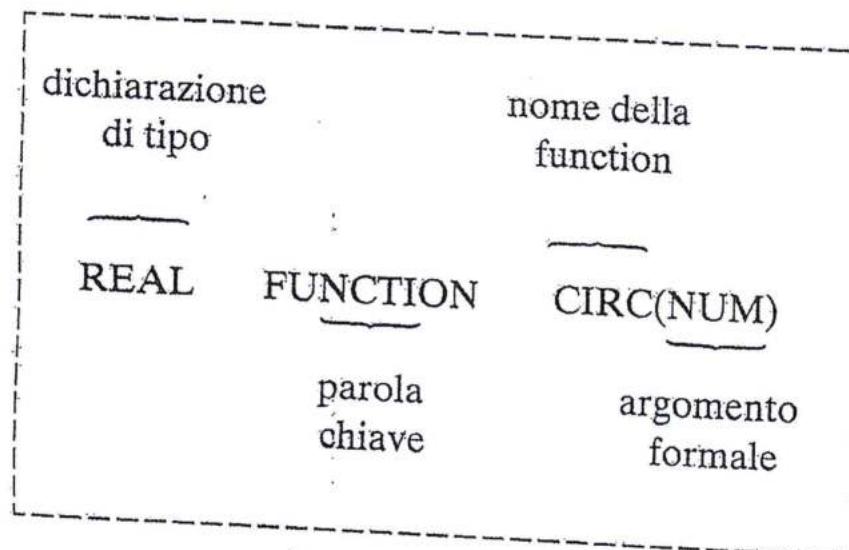
```

REAL FUNCTION CIRC(RAG)
.....
```



Istruzione FUNCTION

Definisce la testata del sottoprogramma di tipo function



REAL FUNCTION CIRC (RAG)

-
-
-

CIRC=6.28*RAG
RETURN

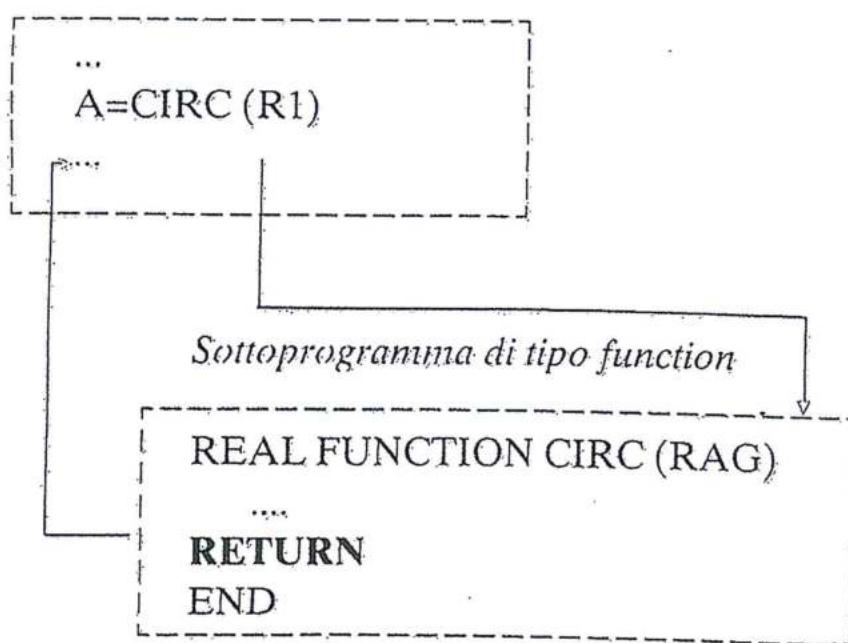
Nel corpo della function:

- il nome della function, CIRC, viene usato come una variabile del tipo dichiarato nella testata della function
- a tale variabile deve essere SEMPRE assegnato un valore
- al termine della function il valore di tale variabile diventa il valore della function

Istruzione RETURN

l'istruzione **RETURN** restituisce il controllo al programma chiamante nel punto in cui è stata chiamata la function

Programma chiamante



SOTTOPROGRAMMI
DI TIPO
SUBROUTINE

Esempio

Programma principale

```
REAL A, B, R1, R2
READ*, R1, R2
  CALL CIRC (R1, A)
  CALL CIRC (R2, B)
  PRINT*, A, B
STOP
END
```

Sottoprogramma di tipo subroutine

```
SUBROUTINE CIRC (RAG, LUNG)
REAL RAG, LUNG
  LUNG=6.28*RAG
RETURN
END
```

CHIAMATA ALLA SUBROUTINE

Programma chiamante

```
...
  CALL CIRC (R1, A)
...
...
```

Sottoprogramma di tipo subroutine

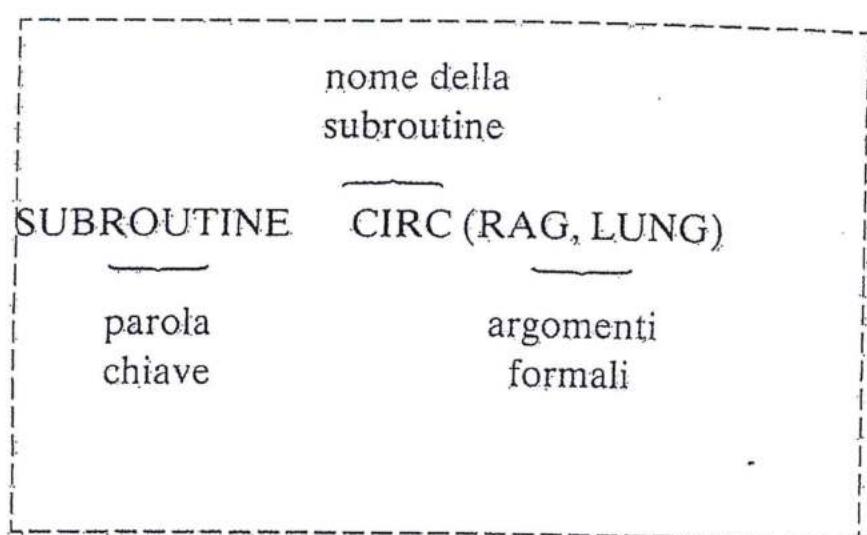
```
SUBROUTINE CIRC(RAG, LUNG)
.....
.....
.....
```



Istruzione SUBROUTINE

Definisce la testata del sottoprogramma di tipo subroutine

Esempio



Istruzione RETURN

l'istruzione **RETURN** restituisce il controllo al programma chiamante per eseguire le istruzioni successive al CALL

Programma chiamante

CALL CIRC (R1, A)
->istruzione successiva

Sottoprogramma di tipo subroutine

SUBROUTINE CIRC (RAG, LUNG)

...
RETURN
END

Esempio

```
SUBROUTINE NUMERO (M, N, X)
REAL X
INTEGER M, N
M=N*INT (X)
RETURN
END
```

Programma chiamante

```
REAL ALPHA
...
CALL NUMERO (L, 2) non corretto!
...
CALL NUMERO (L, 2.5, ALPHA)
           non corretto!
...
CALL NUMERO (K, 2, ALPHA)
           non corretto!
```

La *corrispondenza* tra i *parametri attuali*, che compaiono nella chiamata ad un sottoprogramma, e i *parametri formali*, che compaiono nella testata del sottoprogramma è *per posizione*

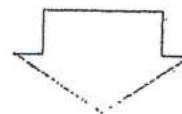
Esempio

```
...  
CALL CIRC (R1, A)  
...
```

```
SUBROUTINE CIRC (RAG, LUNG)
```

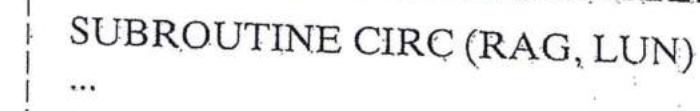
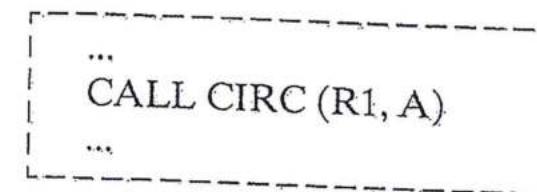
I *parametri attuali* che compaiono nella chiamata ad un sottoprogramma devono *corrispondere in numero e tipo* a quelli *formali* che compaiono nella testata del sottoprogramma

Come avviene il 'collegamento' tra i parametri formali e quelli attuali?

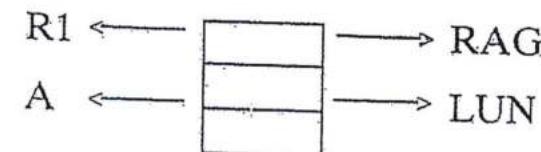


In Fortran la corrispondenza avviene tramite il **passaggio dell'indirizzo** della locazione di memoria del parametro attuale al sottoprogramma chiamato

Il *parametro attuale* e quello *formale* che si corrispondono in una chiamata ad un sottoprogramma puntano alla stessa locazione di memoria.



in memoria



R1 e RAG corrispondono alla stessa locazione di memoria
A e LUN corrispondono alla stessa locazione di memoria

Come avviene la corrispondenza per gli array?

Attraverso il nome dell'array viene passato l'indirizzo della prima componente dell'array.

Esempio

```
REAL BETA (50)
```

```
.....
```

```
.....  
S = SUM (BETA, 50)
```

```
.....
```

```
STOP  
END
```

```
REAL FUNCTION SUM (X, N)
```

```
REAL X(N), SUM
```

```
SUM=0.
```

```
DO 10 I=1, N
```

```
SUM=SUM+X(I)
```

```
CONTINUE
```

```
RETURN
```

```
END
```

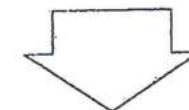
BETA (1) X(1)

BETA (2) X(2)

⋮

BETA (50) X(50)

La *prima* componente del parametro formale *corrisponde* alla locazione di memoria associata al *primo* elemento del parametro attuale



le altre componenti, essendo memorizzate nelle locazioni consecutive, si corrispondono automaticamente

Esempio

```

SUBROUTINE UNIT (A, N)
INTEGER N, A (N, N), I, J
C COSTRUZIONE MATRICE IDENTICA
DO 10 I=1,N
    DO 20 J=1, N
        IF (I.EQ. J) THEN
            A (I, J)=1
        ELSE
            A (I, J)=0
        ENDIF
20     CONTINUE
10     CONTINUE
      RETURN
END

```

Programma chiamante

```

INTEGER   R(3, 3)
CALL UNIT (R, 3)

```

```

PRINT*, R(1, 1), R(1, 2), R(1, 3)
PRINT*, R(2, 1), R(2, 2), R(2, 3)
PRINT*, R(3, 1), R(3, 2), R(3, 3)
STOP

```

In memoria:

R(1,1)
R(2,1)
R(3,1)
R(1,2)
R(2,2)
R(3,2)
R(1,3)
R(3,2)
R(3,3)

Programma chiamante

```

INTEGER R(3,3)
CALL UNIT (R, 3)

PRINT*, R(1, 1), R(1, 2), R(1, 3)
PRINT*, R(2, 1), R(2, 2), R(2, 3)
PRINT*, R(3, 1), R(3, 2), R(3, 3)
STOP

```

In memoria:

R(1,1)
R(2,1)
R(3,1)
R(1,2)
R(2,2)
R(3,2)
R(1,3)
R(3,2)
R(3,3)

	1
	0
	0
	0
	1
	0
	0
	0
	1

esatto!

A(1,1)
A(2,1)
A(3,1)
A(1,2)
A(2,2)
A(3,2)
A(1,3)
A(2,3)
A(3,3)

INTEGER W (3, 3)

```

CALL UNIT (W, 2)

C STAMPA DELLA MATRICE W

PRINT*, W(1,1), W(1,2)
PRINT*, W(2,1), W(2,2)
STOP
END

```

in memoria:

W(1,1)
W(2,1)
W(3,1)
W(1,2)
W(2,2)
W(3,2)
W(1,3)
W(2,3)
W(3,3)

```
INTEGER W (3, 3)
```

```
CALL UNIT (W, 2)
```

```
C STAMPA DELLA MATRICE W
```

```
PRINT*, W(1,1), W(1,2)
```

```
PRINT*, W(2,1), W(2,2)
```

```
STOP
```

```
END
```

in memoria:

W(1,1)	1
W(2,1)	0
W(3,1)	0
W(1,2)	1
W(2,2)	
W(3,2)	
W(1,3)	
W(2,3)	
W(3,3)	

SBAGLIATO!

A(1,1)
A(2,1)
A(1,2)
A(2,2)

```
.....
```

```
CALL UNIT (W, 3, 2)
```

```
.....
```

```
SUBROUTINE UNIT (A, LDA, N)
```

```
INTEGER LDA, N, A (LDA, N), I, J
```

```
C COSTRUZIONE MATRICE IDENTICA
```

```
.....
```

```
RETURN
```

```
END
```

W(1,1)	1	A(1,1)
W(2,1)	0	A(2,1)
W(3,1)		A(3,1)
W(1,2)	0	A(1,2)
W(2,2)	1	A(2,2)
W(3,2)		A(3,2)
W(1,3)		A(1,3)
W(2,3)		A(2,3)
W(3,3)		A(3,3)

ESATTO!

LDA è la
leading dimension
dell'array A



Il suo valore è uguale al
numero di righe
con cui è stata dichiarata
la corrispondente matrice
nel programma chiamante

Linee di commento

Le linee che hanno
i caratteri C o * in prima colonna sono
linee di commento
e sono ignorate in compilazione

1

80

C

CALCOLO CIRCONFERENZA

Un commento occupa almeno
una intera linea

Operazioni di I/O con formato

Nelle istruzioni PRINT e READ si possono specificare:

- le periferiche utilizzate per tali operazioni,
- la modalità di rappresentazione delle informazioni.

Esempio

...
 identificativo
 dell'unità di input

 READ (5, 100) R, NOME

label dell'istruzione
 per il formato

100 FORMAT (I6, 2X, A11)

specifiche di formato

Esempio

REAL R
 ...
 100 READ (5, 100) R
 FORMAT (2X, F8.4)

complessivo
 di spazi riservati
 per il valore di R

cifre
 dopo il punto

Se l'input è

180.1

alla variabile R è assegnato il valore 180.1

Esempio

```
      ...
      WRITE (6,200) RN
200   FORMAT (0, 'RN=', F5.2)
```

se RN contiene il valore 0.713256,
in input si ha

RN = 0.71

Nel Fortran sono presenti dei sottoprogrammi per il calcolo di alcune funzioni matematiche elementari e per la conversione di tipo

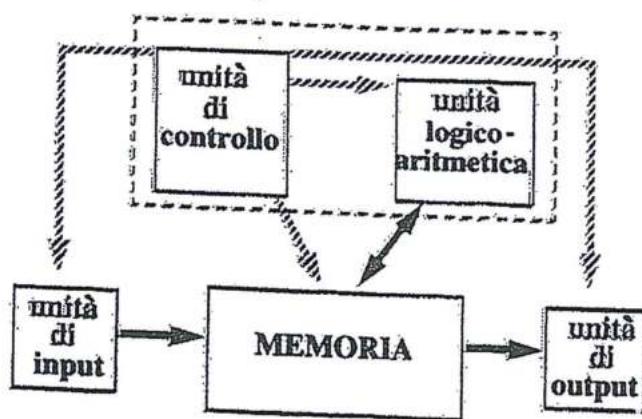
funzione	nome	tipo argomento	tipo del valore
radice quadrata	SQRT	reale	reale
conversione intera	INT	reale	intero
conversione reale	FLOAT	intero	reale
valore assoluto	ABS	reale	reale
	IABS	intero	intero
seno	SIN	reale	reale
coseno	COS	reale	reale
tangente	TAN	reale	reale
esponenziale	EXP	reale	reale
logaritmo naturale	ALOG	reale	reale



I SISTEMI OPERATIVI

L'architettura dei calcolatori attuali
è la stessa dei primi calcolatori
(fine anni '40)

**(MACCHINA DI
VON NEUMANN)**



→ flusso dei dati

flusso delle istruzioni

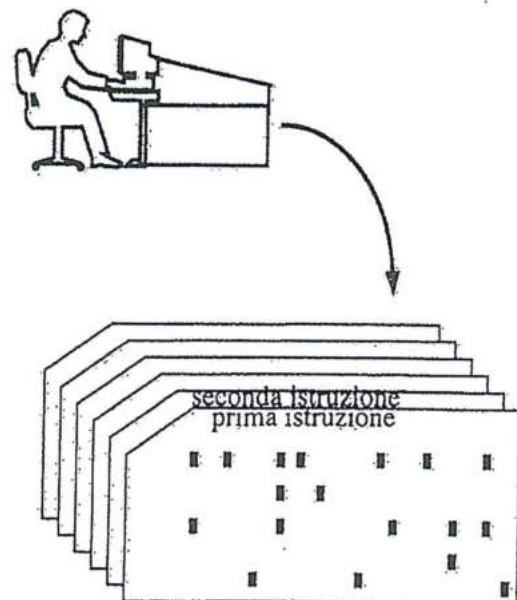
anni '40

Passi per il calcolo della
soluzione di un problema:

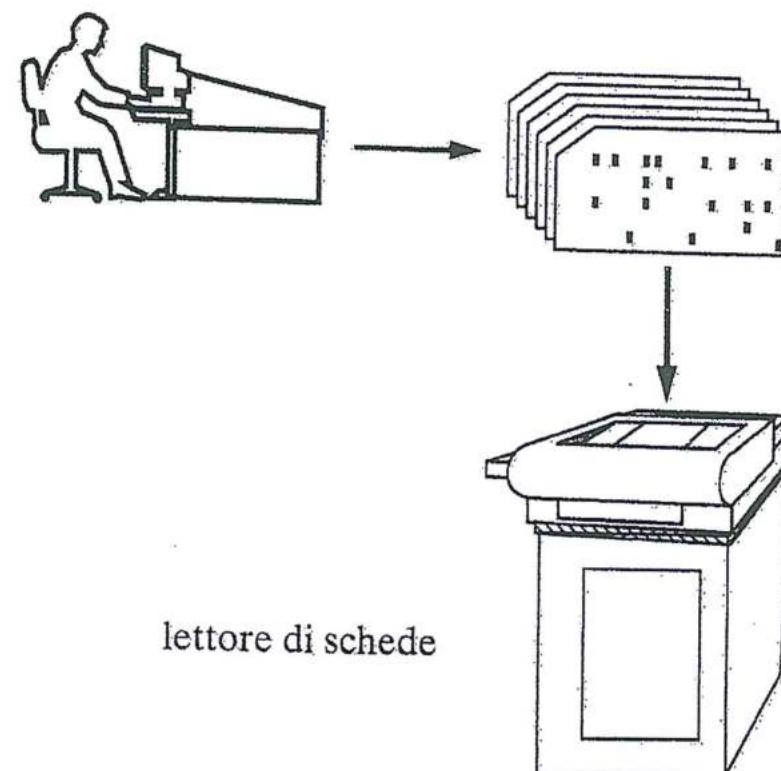
- 1 **preparare** il programma in linguaggio macchina
- 2 **memorizzare** istruzioni e dati
- 3 **fare eseguire** il programma
- 4 **estrarre** il risultato dalla memoria

Passo 1:

scrittura del programma in
linguaggio macchina e sua
trascrizione su **schede perforate**
mediante apposite
macchine perforatrici

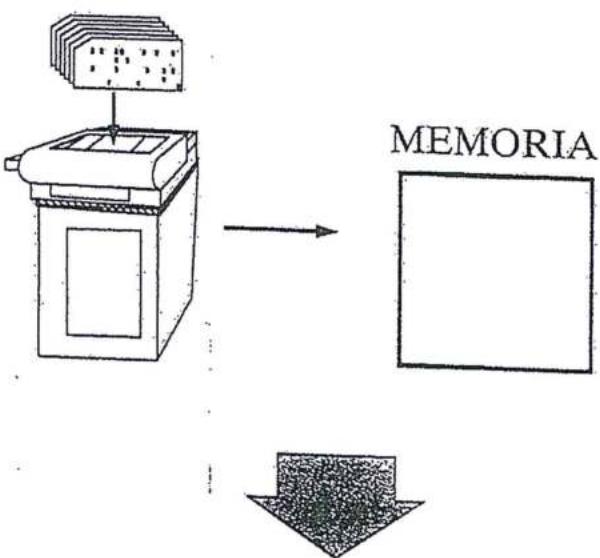


Le schede perforate sono lette
da un **lettore di schede**.



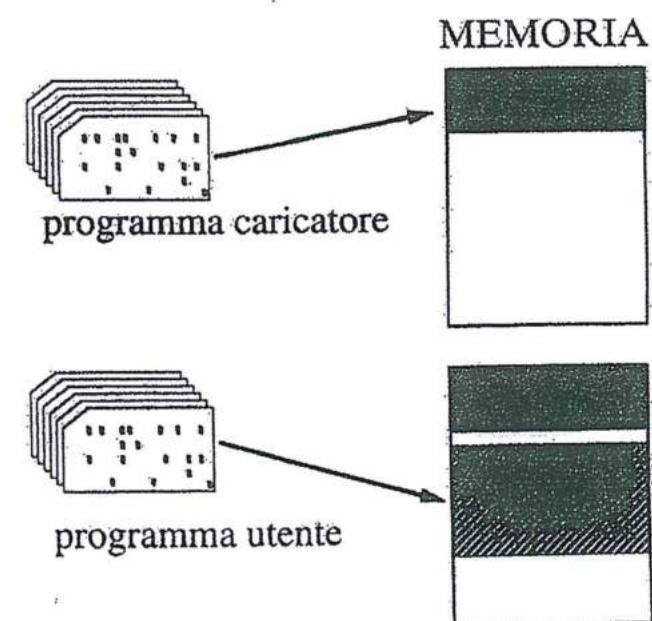
Passo 2:

il programma
deve essere trasferito dal lettore di
schede alla memoria del calcolatore

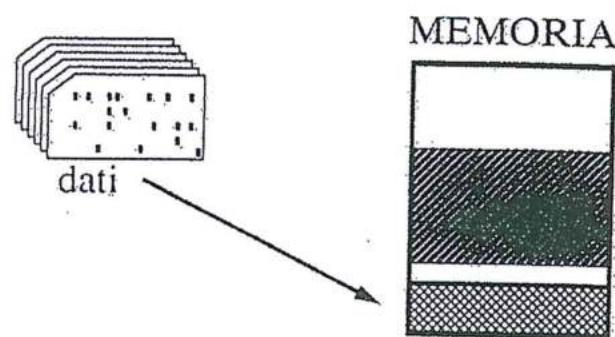


necessità di un

programma caricatore o loader

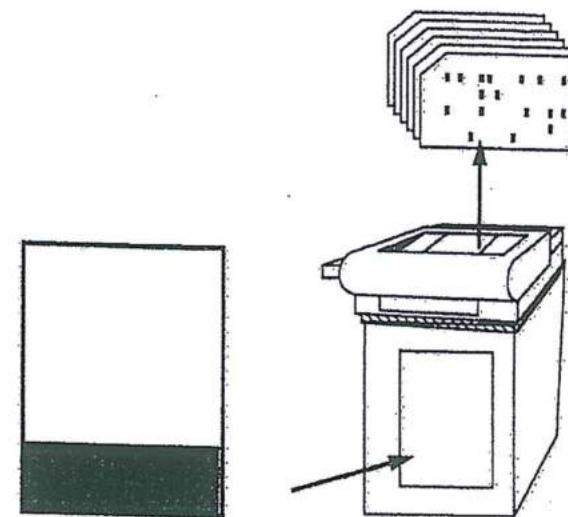


Passo 3:

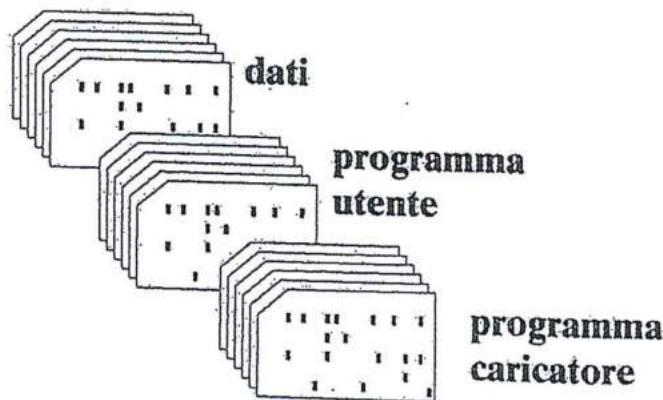


Passo 4:

Il risultato viene trascritto su altre schede (o nastri perforati) mediante un perforatore



Ogni singolo utente,
per poter caricare il proprio programma in
memoria, **deve prima memorizzare il
programma cariatore**



il programma caricatore è
una **utility** comune
a tutti i programmi utente

La presenza di un programma cariatore
disponibile a tutti gli utenti costituirà
il primo embrione di

sistema operativo



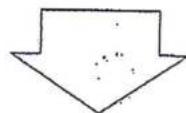
SISTEMA OPERATIVO

programma caricatore

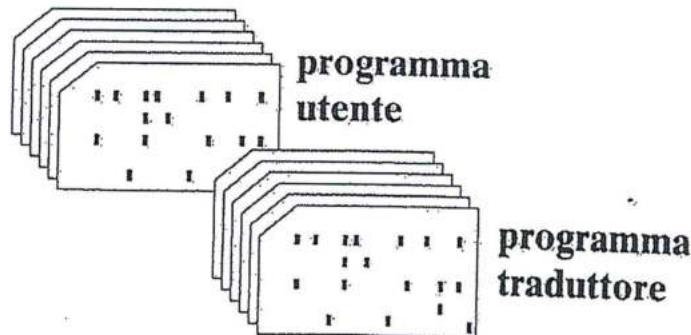
HARDWARE

Fine anni '50

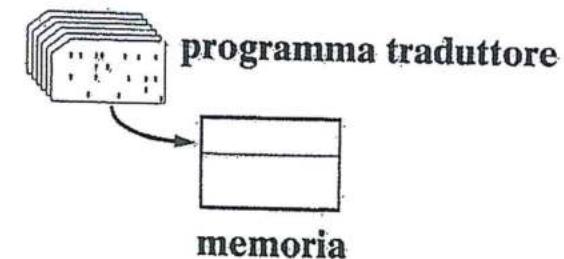
introduzione dei
linguaggi ad alto livello



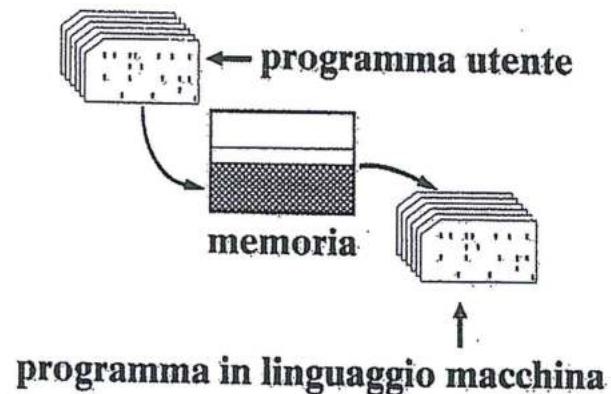
necessità di programmi
traduttori



Fase 1: caricamento del traduttore

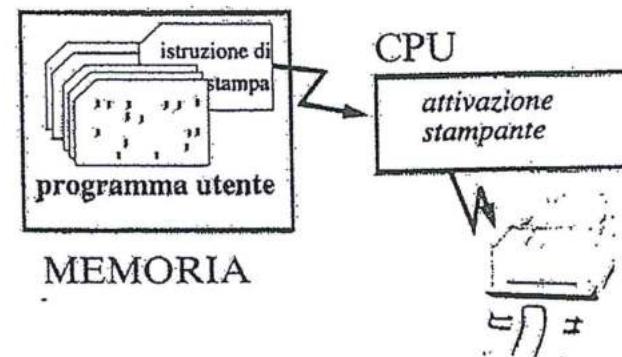
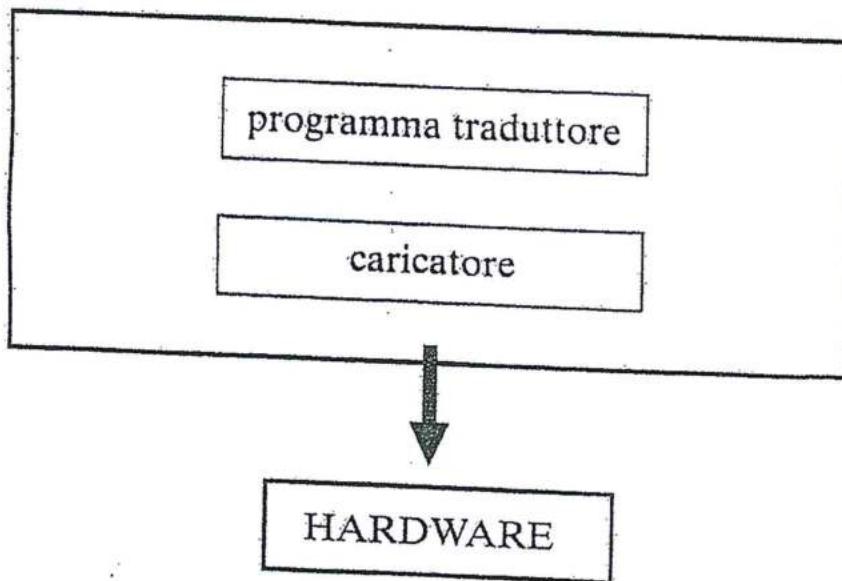


Fase 2: traduzione del programma

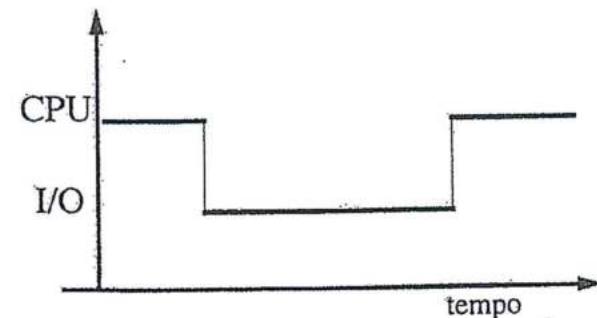




SISTEMA OPERATIVO



Mentre viene eseguita l'istruzione
di stampa, la **CPU** resta inattiva.



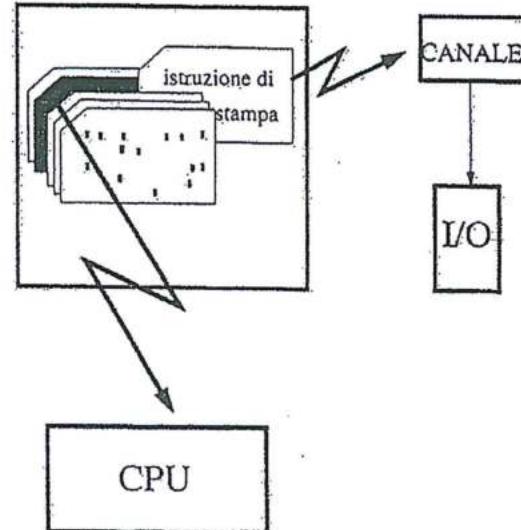
È possibile utilizzare la CPU per eseguire
le istruzioni successive?

I passi 2. (input) e
4. (output), eseguiti
manualmente e meccanicamente,
sono molto più lenti
rispetto al passo 3. (esecuzione)



Non viene sfruttata in pieno la
velocità operativa della CPU e
viene rallentato l'intero processo

MEMORIA



mentre il canale controlla la stampa,
la CPU può prendere in esame
un'altra istruzione

affidare il controllo dell'I/O alla CPU



cattivo utilizzo della CPU



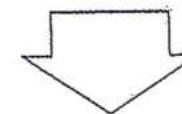
esigenza di avere a disposizione un dispositivo aggiuntivo per controllare esclusivamente le operazioni di I/O

CANALE di I/O

Il canale può essere visto come un calcolatore

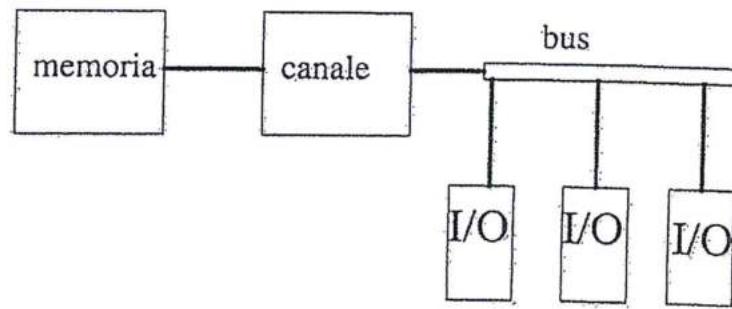
indipendente dalla CPU

che provvede a gestire l'I/O mentre la CPU esegue il programma utente

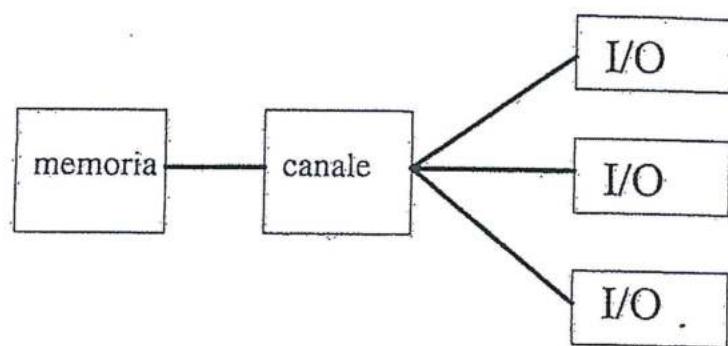


- eventuale conversione di codici (es. da Hollerit ad ASCII)
- controllo sulla quantità di dati da memorizzare
- indirizzamento delle varie unità di I/O

Di solito un canale gestisce più unità di I/O



gestione mediante bus



gestione a stella



sistema operativo

programma per la
gestione dell'I/O

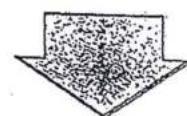
programma per la sincronizzazione
tra CPU e canale

programma caricatore
programma traduttore

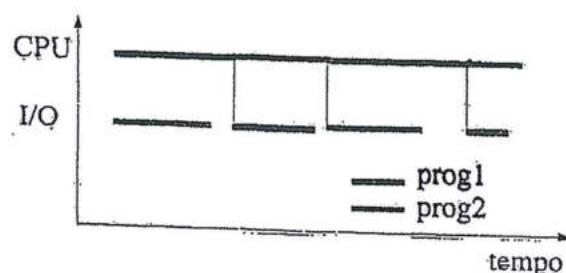
HARDWARE

Anni '60

L'introduzione dei canali di I/O
consente l'uso della CPU da
parte di più programmi



MULTIPROGRAMMAZIONE

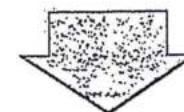


uso efficiente della CPU

MULTIPROGRAMMAZIONE

=
capacità del sistema operativo di
far condividere a più programmi
l'uso della CPU

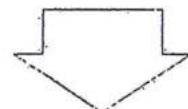
quando un programma effettua una
operazione di I/O, l'uso della CPU
passa ad un altro programma



i programmi in coda attendono
lunghi periodi di tempo
nell'attesa dell'istruzione di I/O
del programma che sta utilizzando la CPU

SOLUZIONE

ad ogni programma viene assegnato
un tempo massimo di utilizzo
della CPU (*time slice*)



TIME SHARING

Ogni programma può utilizzare
la CPU finché non si verifica una delle
seguenti condizioni:

- 1) il programma finisce
- 2) il programma va in errore (es. divisione per
zero, overflow...)
- 3) è richiesta una operazione di I/O
- 4) è terminato il tempo assegnato.

Casi 1 e 2: il programma è eliminato dalla memoria

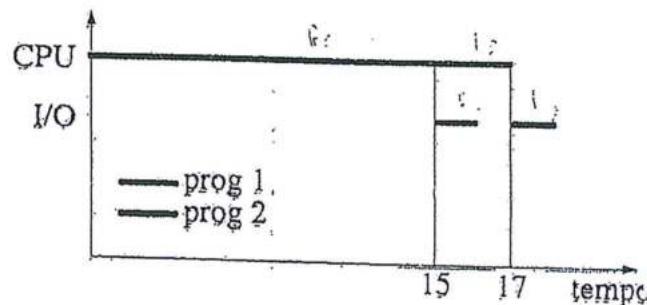
Casi 3 e 4: il programma è sospeso



Esempio: multiprogrammazione senza time sharing

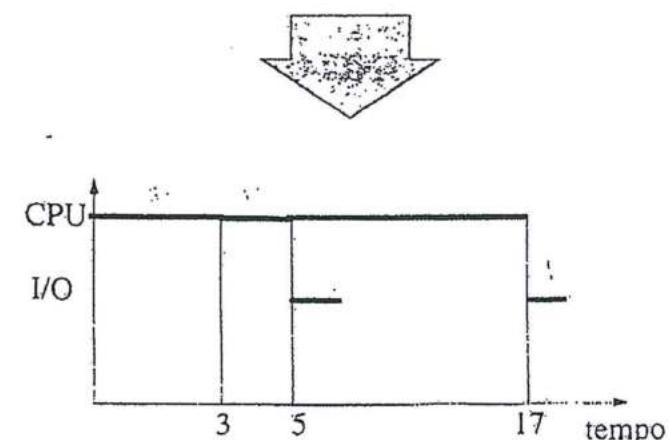
due programmi con le seguenti
caratteristiche sono presenti in memoria:

	tempo CPU	tempo I/O
Prog. 1	15	1
Prog. 2	2	2

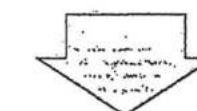


il programma 2 attende 15 unità di tempo
per utilizzare la CPU solo per 2 unità

Esempio (cont.): multiprogrammazione con time sharing (time slice = 3 unità)

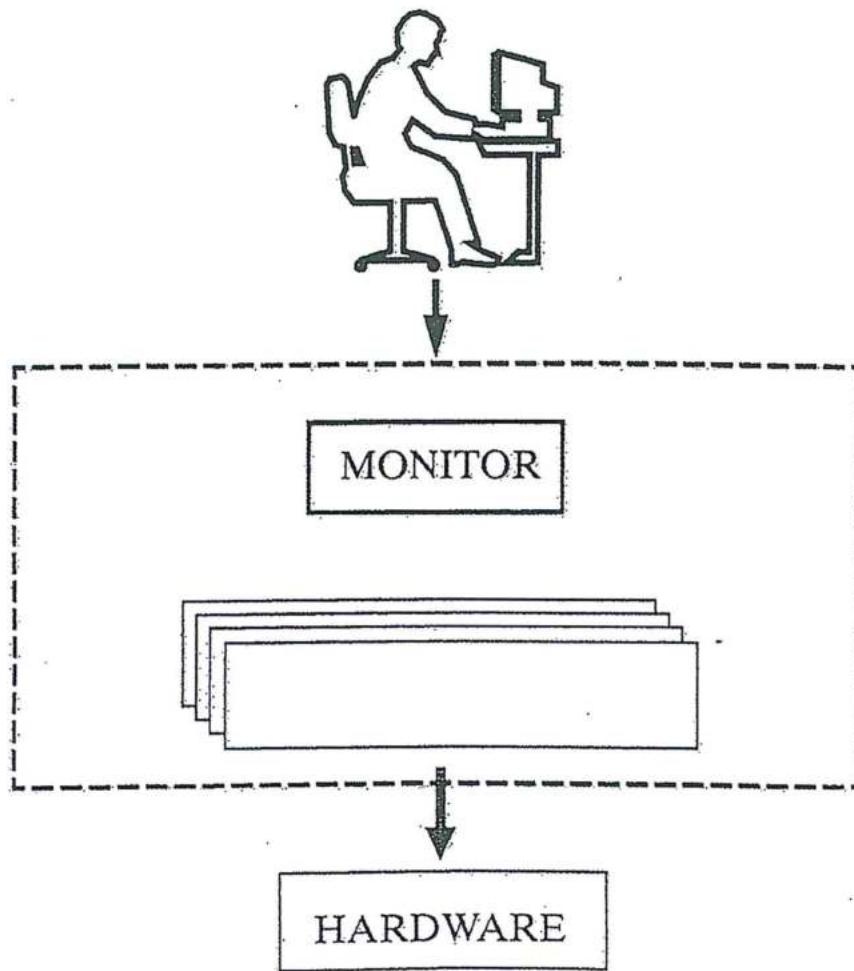


con il time sharing il programma 2 attende
solo 3 unità di tempo prima di utilizzare la
CPU



riduzione dei tempi medi di attesa

La multiprogrammazione e il time sharing comportano la presenza di un programma che gestisce la corretta sospensione e ripresa dei programmi nella CPU (**MONITOR**)

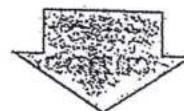


Il MONITOR deve:

- conservare tutte le informazioni relative allo stato del programma da sospendere, necessarie per una corretta ripresa della sua esecuzione
- decidere a quale tra gli altri programmi in attesa passare il controllo
- interrompere l'esecuzione del programma e mandare in esecuzione il nuovo programma

Per decidere quale programma passare in memoria quando diversi programmi sono in attesa, bisogna stabilire un ordine di accesso alla CPU

Un programma è incaricato di stabilire tale ordine



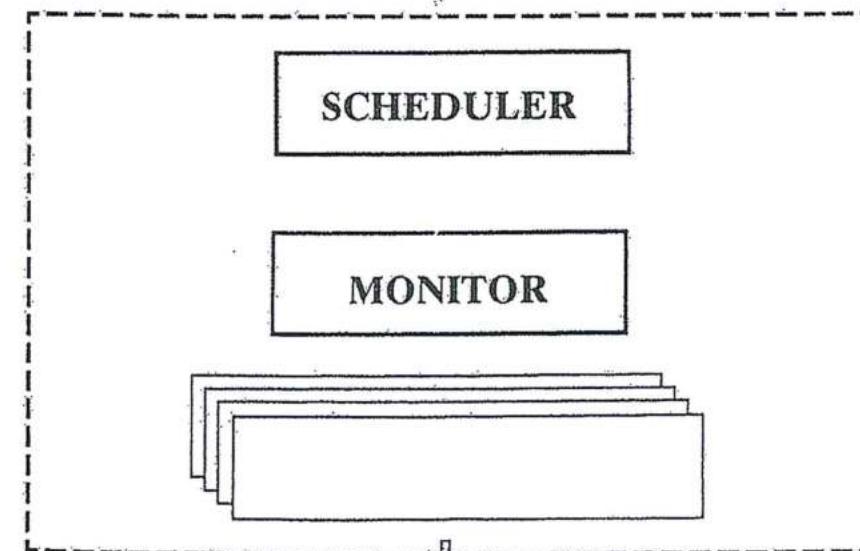
SCHEDULER



SCHEDULER

MONITOR

HARDWARE



Lo *Scheduler* tiene conto di diversi fattori:

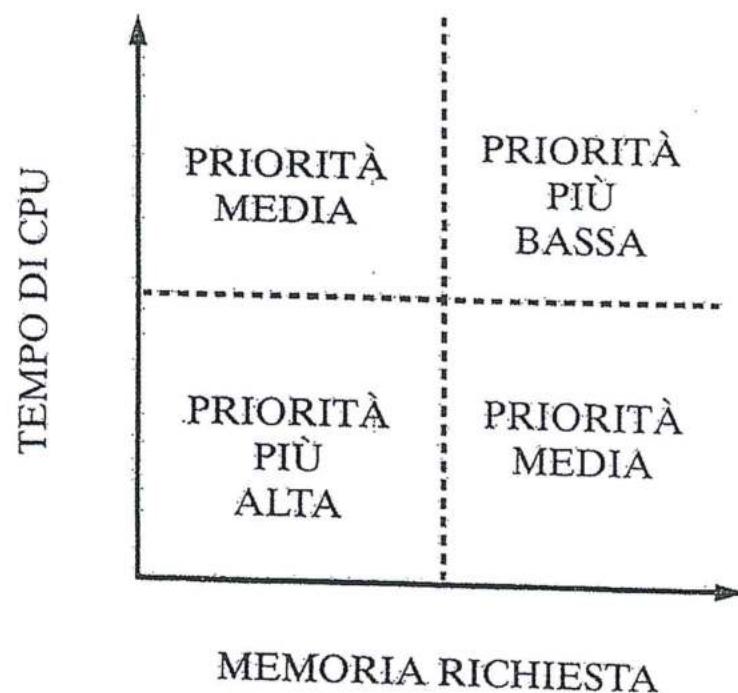
- memoria richiesta dai programmi
- tempo di CPU richiesto dai programmi
- tipo del programma (di utente o di sistema)
- tipo di utente (utente qualunque o gestore del sistema)
- uso recente della CPU

lo *scheduler* assegna ad ogni programma un numero che determina il criterio di precedenza nell'utilizzo della CPU;

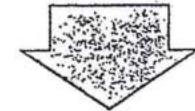
PRIORITÀ

In particolare:

- poca memoria e poco tempo di CPU hanno priorità più alta;
- programmi di sistema hanno priorità più alta rispetto a quelli di utente



Un **SISTEMA OPERATIVO** è
l'insieme dei programmi che
consentono il controllo e la gestione di
tutte le risorse del calcolatore



HARDWARE + SISTEMA OPERATIVO .
(SOFTWARE DI SISTEMA)
= macchina virtuale

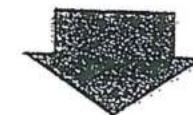
EVOLUZIONE DELLA MEMORIA



limitatezza della memoria



incapacità di memorizzare
grosse quantità
di informazioni

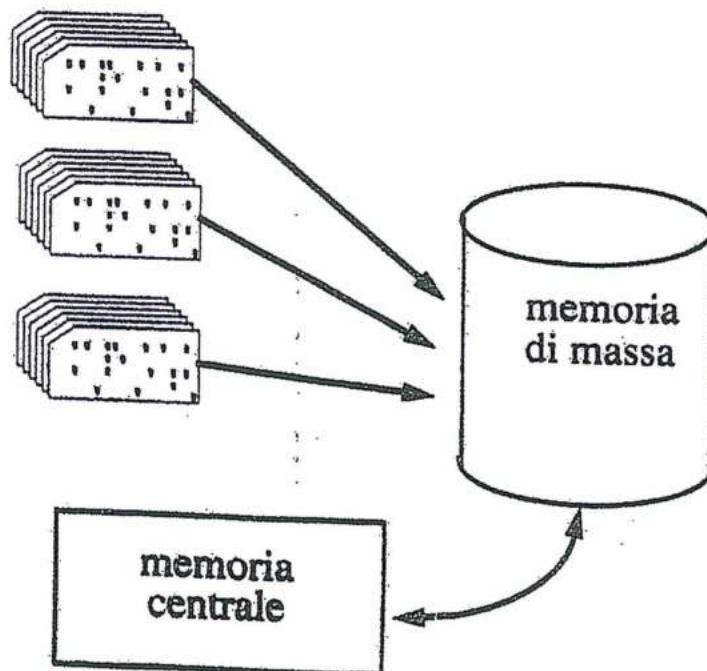


i programmi più
voluminosi (**ad es. i compilatori**),
venivano conservati su schede e caricati
in memoria
solo quando necessario

per automatizzare il caricamento e per conservare in modo efficiente tutti i programmi vennero introdotte le

MEMORIE DI MASSA O AUSILIARIE

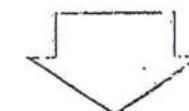
(dischi e nastri magnetici)



MEMORIE DI MASSA

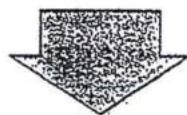
minor costo, maggiore capacità,
ma più lento tempo di accesso

la CPU non può manipolare informazioni direttamente nella memoria di massa. Queste devono essere prima trasferite in *memoria centrale*



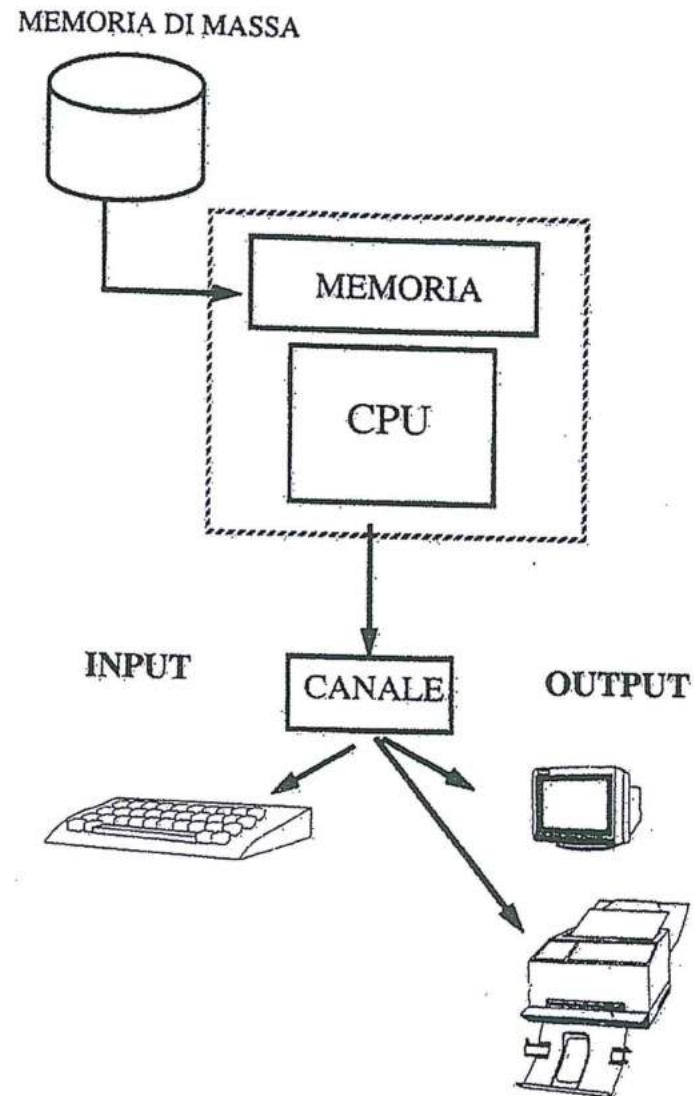
necessità di un programma per automatizzare il trasferimento dei programmi dalla memoria di massa alla memoria centrale

Avvento della memoria di massa

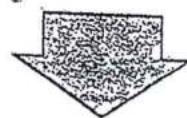


organizzazione dei programmi
del sistema operativo in

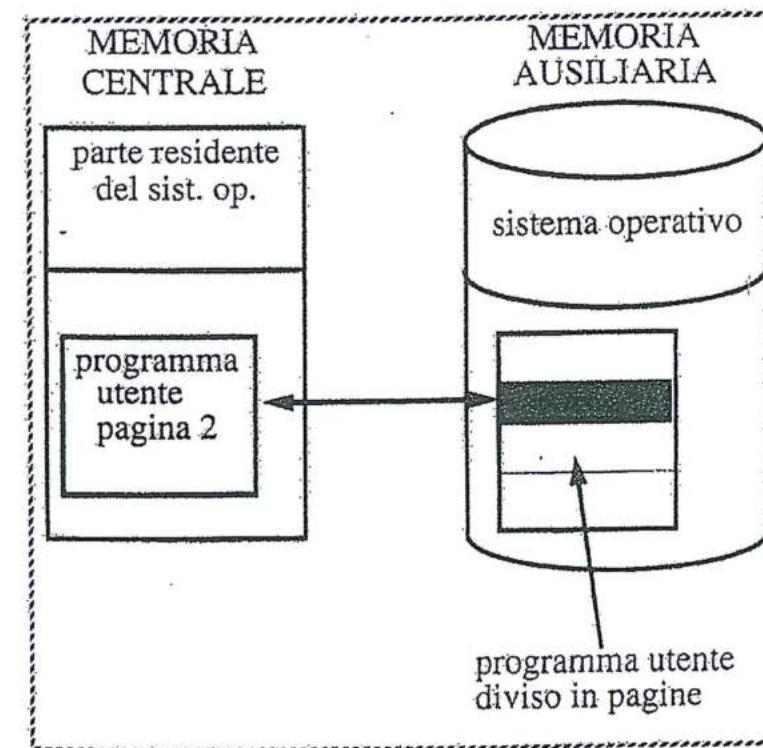
- **parte residente in memoria centrale** (es. trasferimento di programmi dalla memoria ausiliaria memoria centrale, caricatori, gestione dell'I/O,...)
- **parte presente in memoria ausiliaria**, costituita dai programmi non necessari a soddisfare al momento richieste dell'utente (ad es. i programmi traduttori e i programmi utente)



i programmi eseguibili non possono superare le dimensioni della memoria centrale



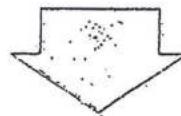
l'utente deve dividere il programma in pezzi (pagine) per tenere in memoria centrale solo l'istruzione in esecuzione ed un certo numero di quelle che seguono



necessità di programmi per la gestione automatica dello scambio di informazioni tra memoria centrale e memoria ausiliaria

MEMORIA VIRTUALE

automatizzazione dello scambio
di informazioni tra memoria
centrale e memoria ausiliaria

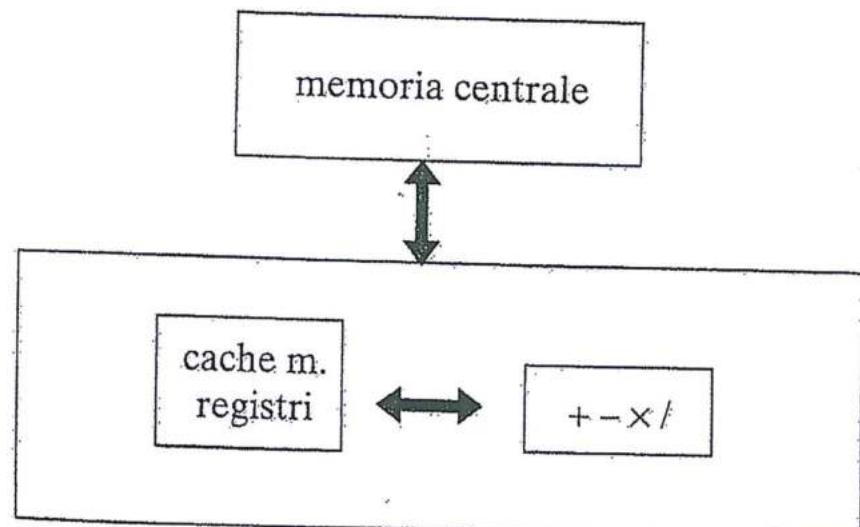


MEMORIA VIRTUALE

recentemente sono state introdotte
alcune aree di memoria molto
piccole e di rapidissimo accesso

(cache memory e registri)

che sono state aggiunte all'unità
aritmetico-logica per memorizzare
i dati più frequentemente utilizzati



unità aritmetico logica

velocità e capacità dei vari
“livelli” di memoria

velocità

10^{-9} sec =
 $1 n$ sec

registri

10^{-8}
sec

cache
memory

10^{-7}
sec

memoria
centrale

10^{-6}
sec

memoria di
massa

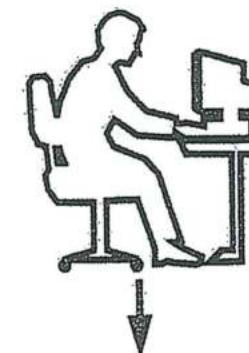
capacità

16-512
bytes

256-1024
Kbytes

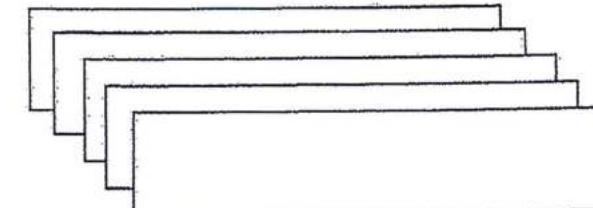
16-512
Mbytes

>1
Gbyte



sistema operativo

programmi per la
gestione delle memorie



HARDWARE

La comunicazione tra calcolatore e l'esterno avviene tramite il
LINGUAGGIO DI COMANDO



Ogni frase del linguaggio di comando è una richiesta al sistema operativo di esecuzione di una particolare operazione

ESEMPI DI COMANDI AL SISTEMA OPERATIVO

UNIX MS/DOS

ls	dir	lista dei file
cp	copy	copia dei file
rm	delete	cancellazione di un file
cat	type	lista il contenuto di un file
f77	fl	compilazione Fortran

ogni richiesta al sistema operativo comporta le seguenti azioni:

- interpretare le frasi del linguaggio di comando
- individuare il programma di sistema che soddisfa la richiesta
- caricare tale programma in memoria centrale
- far eseguire il programma
- restituire il controllo all'utente per rispondere ad altre richieste

Tali azioni sono eseguite da un programma del sistema operativo

L'INTERPRETE DEL LINGUAGGIO DI COMANDO

Recentemente sono state introdotte delle

INTERFACCE GRAFICHE

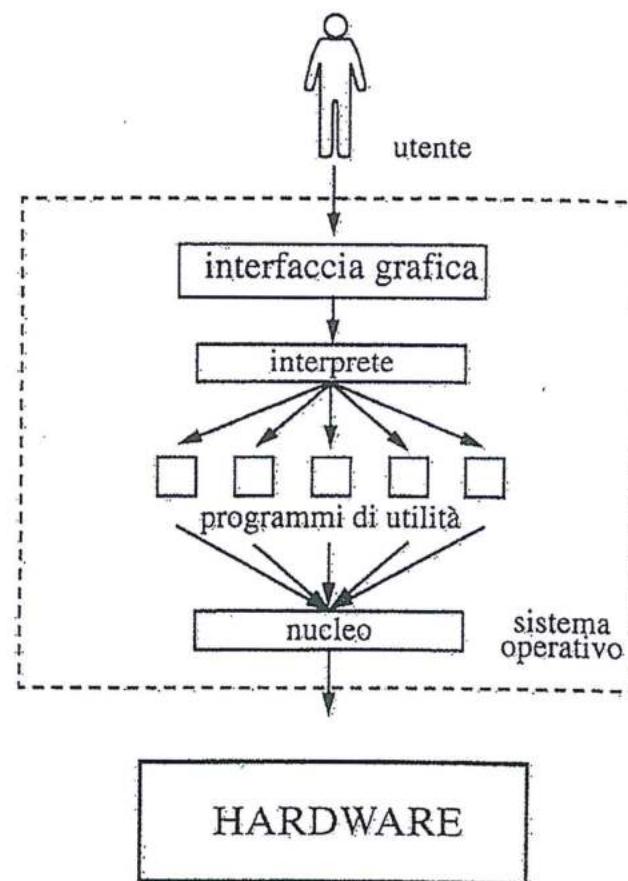
ai comandi del sistema operativo per facilitarne l'uso

Windows 95 - Windows 98

OS/2 - Warp

Xwindows

L'evoluzione dei sistemi operativi ha definito una struttura logica "a livelli"

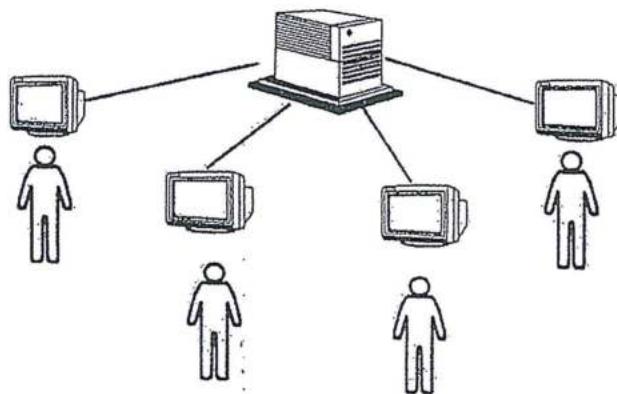


Esistono 2 tipi di sistemi operativi:

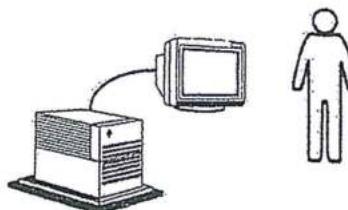
MONOUTENTE: il sistema operativo permette ad un solo utente l'accesso alle risorse della macchina (Es. *MS/DOS*)

MULTIUTENTE: il sistema operativo permette a più utenti l'accesso alle risorse della macchina, mediante terminali remoti (Es. *Unix*)

sistema operativo multiutente (es. UNIX)



sistema operativo monoutente (es. MS/DOS)



- Se il **sistema operativo è monoutente**, l'utente ha immediato accesso al sistema
- se il **sistema operativo è multiutente**, l'utente è obbligato a dichiarare la propria identità prima di qualunque richiesta al sistema operativo

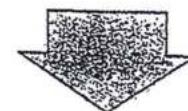
(FASE DI LOGIN)

L'utente ha accesso alla macchina solo se il suo nome è registrato nella memoria del calcolatore

Evoluzione dei sistemi operativi

1945	primo calcolatore elettronico (ENIAC)
1949	primo calcolatore capace di memorizzare programmi (EDSAC)
anni '50	primi sistemi operativi
1957	primo linguaggio ad alto livello (Fortran)
1960-65	introduzione della multiprogrammazione e del time sharing
fine anni '60	introduzione della memoria virtuale
anni '80	interfacce grafiche

L'obiettivo degli **attuali sistemi operativi** è non solo permettere lo sviluppo di programmi, ma creare un ambiente di lavoro, completo e "confortevole"



Disponibilità di strumenti per far fronte alle esigenze di diversi tipi di utente
(ricercatori, segreterie, ...)

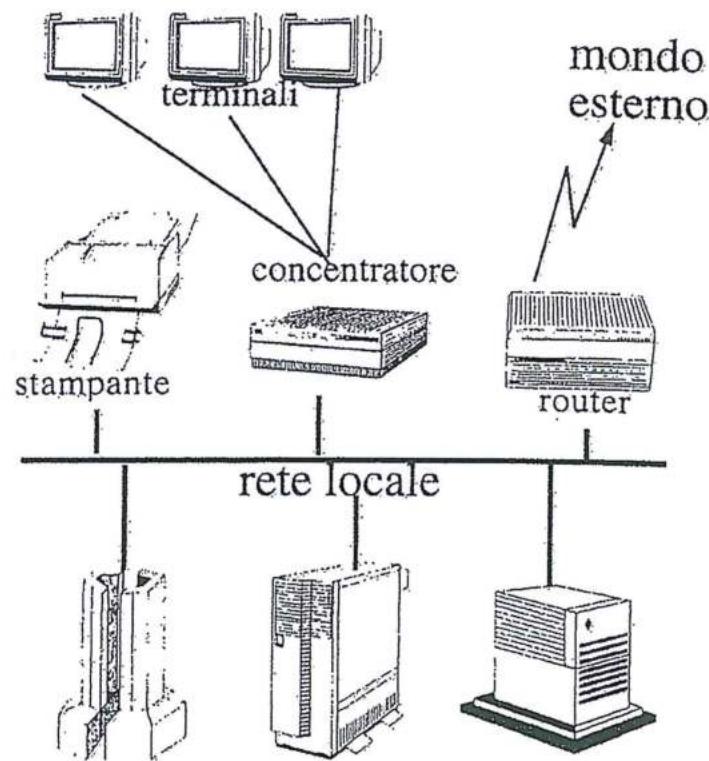
Strumenti per lo sviluppo di software

- compilatori
- programmi per la ricerca degli errori (*debugger*)
- programmi per la valutazione delle prestazioni (*timer*)
- programmi per la scrittura dei programmi di utente (*editor*)
-

Strumenti per l'office automation

- programmi per l'elaborazione di testi
- posta elettronica
- agende elettroniche e calendari elettronici
-

Nei moderni centri di ricerca e università convivono più calcolatori collegati in rete



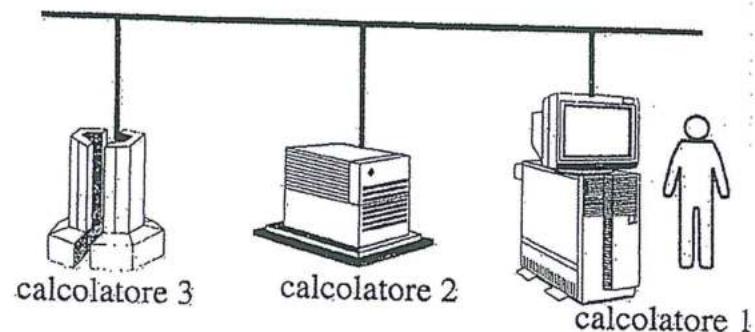
supercalcolatori e workstation

ogni calcolatore è identificato da un nome

In tale ambiente di calcolo ogni utente può accedere a tutte le risorse della rete in maniera trasparente

ESEMPIO:

l'utente collegato con il calcolatore 1 utilizza un programma residente sul calcolatore 2 con dei dati residenti sul calcolatore 3

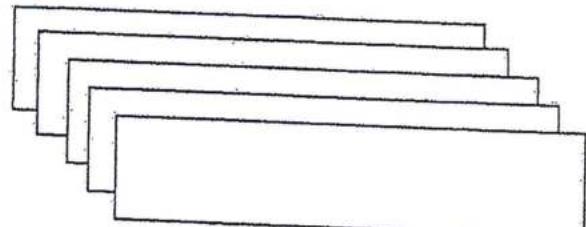


Necessità di programmi per il trasferimento automatico delle informazioni tra i calcolatori della rete



sistema operativo

programmi per il trasferimento delle informazioni



HARDWARE

INTERNET

=
connessione di più livelli di rete

altre reti nazionali

rete nazionale

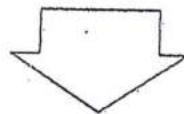
rete cittadina

reti locali

ogni rete è identificata da un nome

ESEMPIO

rete	nome
nazionale	it
cittadina	unina
locale	dma
calcolatore	matna2



matna2.dma.unina.it

identifica lo specifico calcolatore
in tutto il mondo

I sistemi operativi attuali
includono alcuni
strumenti software per
l'utilizzo delle reti
di calcolatori

- trasferimento di file (ftp)
- collegamento remoto (telnet)
- browsers
(Netscape, Mosaic, Explorer...)