Appunti di INFORMATICA

Origini dell'informatica

L'informatica è …	L'informatica non è …
La scienza che studia i metodi per risolvere i problemi	Lo studio dei calcolatori
La scienza del ragionamento da tradurre in programmi	L'uso del computer
La logica applicata ai computer	Usare INTERNET
	La conoscenza di molti linguaggi
	La conoscenza di particolari programmi
	La capacità di installare programmi

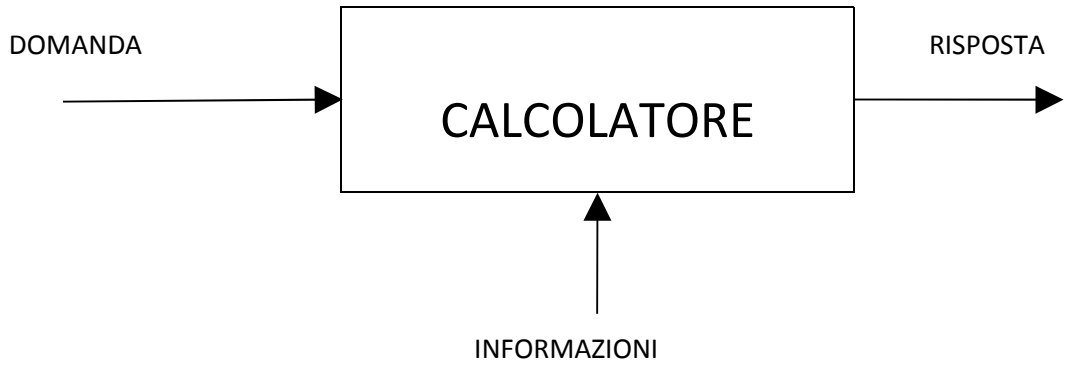
Alla base dell' informatica non c'è il computer ma il **pensiero logico** , il computer per l'informatica è uno strumento come il telescopio per l'astronomia .

Che cosa significa INFORMATICA

INFORMATICA = INFORMAZIONE + AUTOMATICA

(nei paesi anglosassoni Computer Science)

L'informazione è un dato in grado di aumentare la conoscenza di una realtà



Per informatica si intende l'insieme delle scienze e delle tecnologie che riguardano i concetti di :

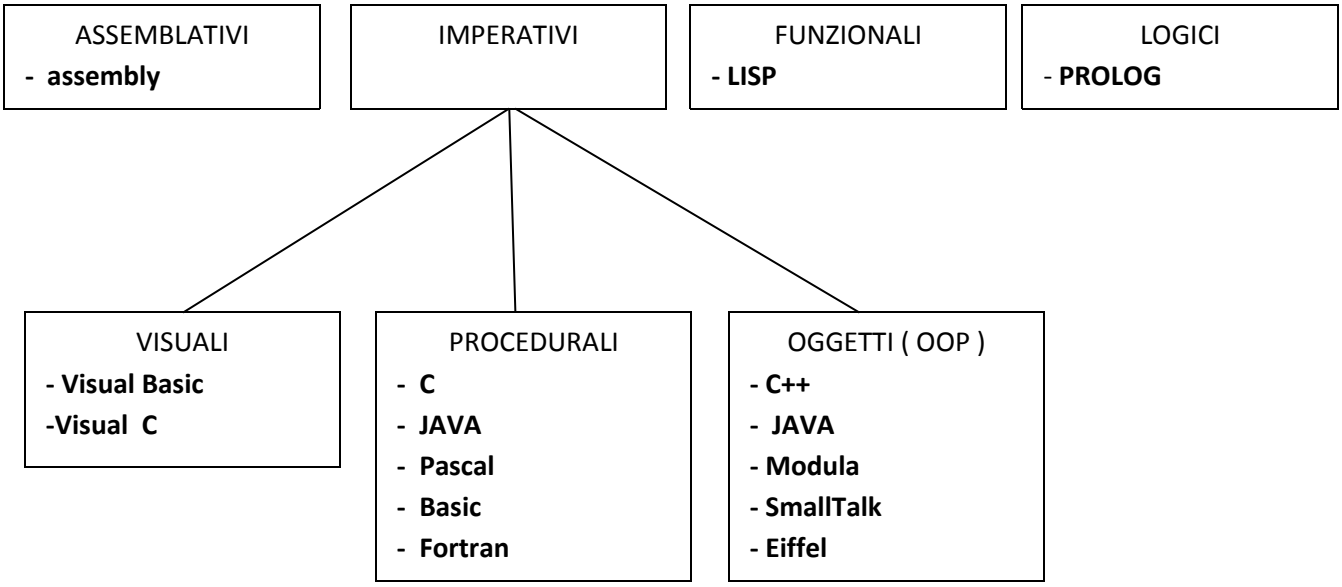
- Elaborazione
- Comunicazione
- Trasmissione

delle informazioni.

ORIGINI della INFORMATICA		
TECNOLOGIA	MATEMATICA	LINGUAGGI
Abaco e calculi	Algoritmo	Linguaggio naturale
Automi meccanici	Algebra di Boole	Linguaggio macchina
Telai	Assioma di induzione (Peano)	Linguaggio alto livello
Orologi	Church , Turing , Godel	Linguaggio visuale (icone)
Calcolatrici meccaniche		

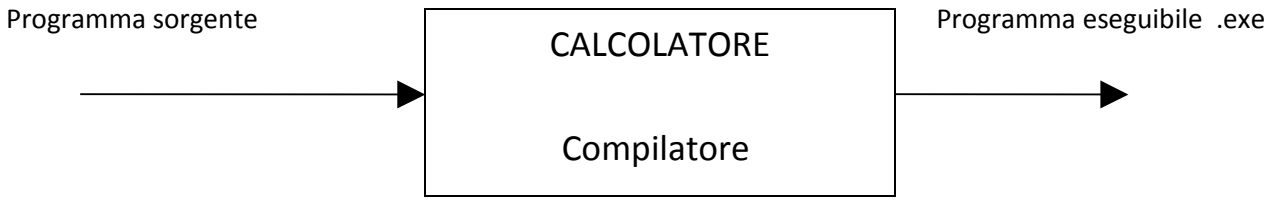
Linguaggi ad alto Livello	
Fortran	Formula Translation : capostipite dei linguaggi ad alto livello
Cobol	Ambito commerciale (simile pascal)
Basic , Pascal , PL1	Evoluzione del Fortran
Ada	Evoluzione del Pascal , con interessanti novità
Lisp	Programmazione Funzionale
Prolog	Programmazione Logica : molto interessante
C , C++ , Java	Linguaggi attuali : sia imperativi che ad oggetti

Classificazione dei linguaggi



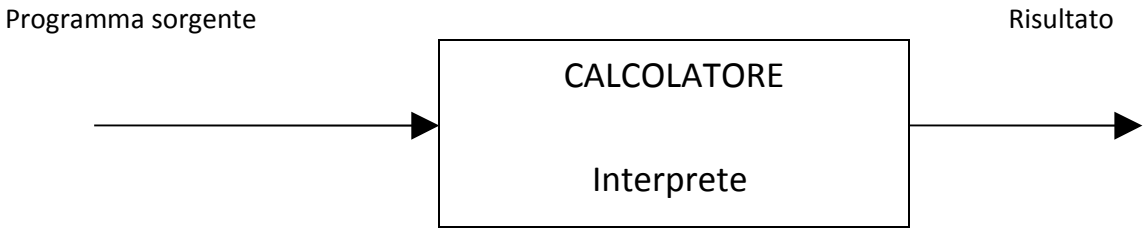
Differenze tra linguaggio COMPILATO ed INTERPRETATO

I linguaggi di alto livello si dividono in linguaggi compilati e linguaggi interpretati , nei linguaggi compilati si usa un programma , il COMPILATORE , per tradurre integralmente un programma sorgente in uno eseguibile dal computer .



- Scarsa portabilità
- Maggiore velocità di esecuzione

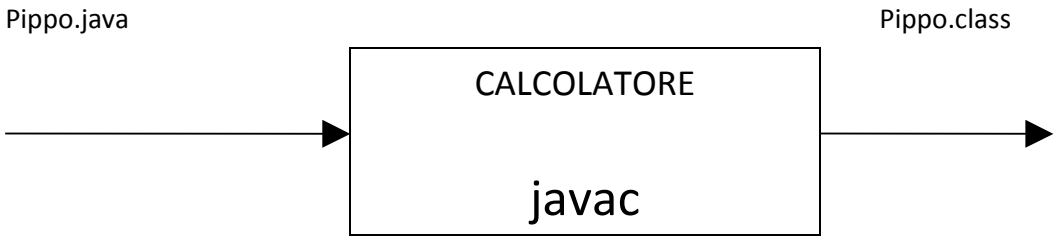
Nei linguaggi interpretati nel computer deve essere presente un INTERPRETE del linguaggio (macchina virtuale) che legge ed interpreta (traduce in codice eseguibile dal computer) le istruzioni .



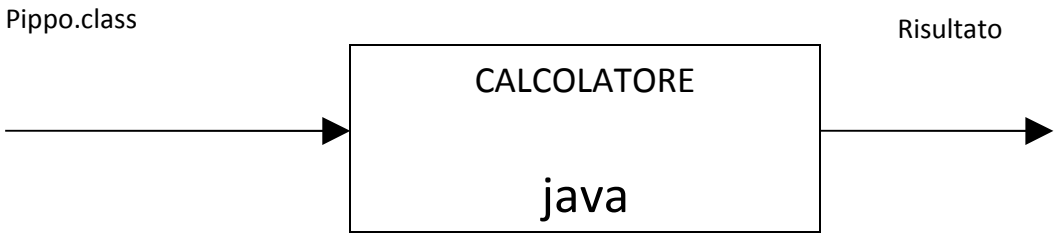
- Più lento
- Portabilità maggiore

ATTENZIONE : JAVA è un linguaggio prima compilato e poi interpretato

1. La compilazione di un file java



2. L'esecuzione del file , che viene interpretato dalla macchina virtuale

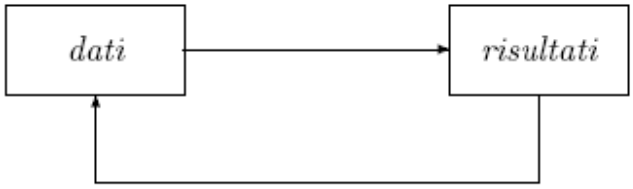


La cosa risulta automatica utilizzando un editor , ad esempio jcreator .

Problemi

Un problema `e un quesito cui si vuole dare risposta o un obiettivo che si intende perseguire.

Risolvere un problema significa ricercare e descrivere un procedimento risolutivo che, eseguito, consente, a partire da delle informazioni iniziali (***dati***), di ottenere delle informazioni finali (***risultati***) soddisfacenti ad un ***criterio di verifica***, secondo lo schema descritto dalla seguente figura.



criterio di verifica

Si afferma che un problema è ***ben formulato*** o ***formalmente definito*** se soddisfa alle seguenti specifiche:

1. Il problema è formulato in una forma comprensibile dal solutore
2. I dati sui quali si deve cercare la soluzione sono completi e coerenti
3. È univoco il criterio di verifica della soluzione

Determinare il barimetro di un poliangolo.

Questo è un chiaro esempio di problema mal formulato poichè non viene rispettata la condizione a) in quanto nella formulazione del problema compaiono dei termini che non hanno alcun significato.

Determinare l'area di un trapezio rettangolo sapendo che le basi maggiore e minore misurano rispettivamente 18 *e* 12 *unità.*

In questo caso non è soddisfatta la condizione b).

FORMALIZZAZIONE DEI PROBLEMI

Il processo di *formalizzazione* o *modellizzazione* di un problema consiste nel tradurre un problema reale in una sua formulazione alternativa, alquanto semplificata ma equivalente per quanto riguarda la tematica di interesse, in modo tale che esso possa essere risolto automaticamente. Applicando l'inverso di questa trasformazione si tradurranno poi immediatamente i risultati nell'ambito originale del problema. Questi artifici sono spesso utilizzati in Matematica (risoluzione delle equazioni differenziali mediante le trasformate di Laplace, analisi di un segnale in frequenza, calcoli nell'ambito della geometria analitica ed in molte altre situazioni). Vengono qui di seguito presentati degli esempi che illustrano questo artificio.

Esempio 11. Consideriamo il seguente problema:

> *Si ha un bersaglio delimitato da* 3 *cerchi concentrici. Il cerchio più interno vale* 19 *punti, mentre le due corone esterne valgono rispettivamente* 13 *e* 7 *punti. Il raggio del cerchio interno e lo spessore delle corone sono di* 5 *cm. Decidere se, ed eventualmente come, è possibile totalizzare* 100 *punti, in più tiri.*

Filtrando i dettagli superflui, tale problema può essere formalizzato come segue:

> *Risolvere l'equazione diofantea* $19x + 13y + 7z = 100$.

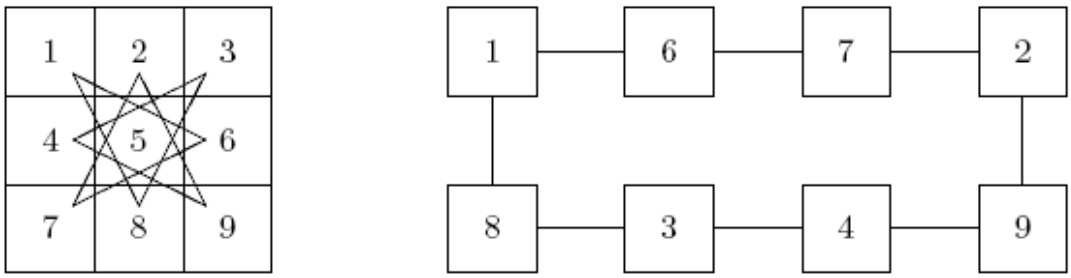
TRASFORMAZIONE DEI PROBLEMI

Non sempre l'ambito nel quale sorgono e vengono formulati i problemi è il contesto più idoneo nel quale ricercarne e descriverne la soluzione. Risulta spesso comodo eseguire una trasformazione in un altro ambito nel quale risulta più facile risolvere il problema. Si parla di *passaggio dallo spazio dei problemi allo spazio delle soluzioni.*

Esempio 12. Consideriamo la seguente situazione:

> *È data una scacchiera* 3×3 *sulla quale sono disposti tre cavalli bianchi e tre cavalli neri, posti ai lati opposti della scacchiera. Il problema consiste nello scambiare fra loro i cavalli bianchi con quelli neri, nel minor numero possibile di mosse, alternando le mosse del bianco e del nero e rispettando le modalità di movimento dei cavalli secondo le regole del gioco degli scacchi.*

Operando direttamente sulla scacchiera ci si convince che il problema non è di immediata soluzione. Il problema si sbroglia rappresentando la scacchiera tramite un grafo; ogni nodo del grafo rappresenta una casella della scacchiera; due nodi del grafo vengono congiunti se si può passare da uno all'altro mediante una mossa del cavallo. Convenendo di etichettare ciascuna casella con un numero naturale da 1 a 9 come indicato in figura, si ottiene la seguente rappresentazione e successiva trasformazione:



Esercizi

Per i problemi che seguono si individuino i dati ed i risultati e si dica se il problema è ben formulato o meno . Se ben formulato descrrivere un procedimento risolutivo del problema .

1. Dato il perimetro e l'area di un triangolo , determinare le misure dei lati
2. Dato il perimetro e l'area di un rettangolo , determinare la misura del lato ad esso equivalente
3. Dato il perimetro e l'area di un rettangolo , determinare la misura della diagonale
4. Dato il perimetro di un rettangolo , determinarne l'area
5. Dato il perimetro e l'area di un rettangolo , detrminare la misura dei lati
6. Noti i cateti di un triangolo rettangolo calcolre l'altezza relativa alla ipotenusa

ALGORITMO

Chiariamoci le idee

Problema è un qualsiasi quesito , di natura diversa , a cui dare soluzione.

Algoritmo è la sequenza di azioni da fare o istruzioni da impartire a qualcuno per risolvere il problema dato. (**procedimento risolutivo**)

Programma è la traduzione in un linguaggio formalizzato delle azioni definite dall'algoritmo.(**visione statica**)

Processo è l'insieme di attività o azioni svolte dal microprocessore quando si manda in esecuzione il programma (**visione dinamica**)

Strategia Operativa

Analisi del problema , ovvero individuare l'obiettivo da conseguire ed i dati iniziali

Soluzione , ovvero individuare le azioni da svolgere sui dati iniziali per ottenere il risultato desiderato.

Astrazione : Capacità di generalizzare la soluzione

Verifica , ovvero valutazione del risultato ottenuto

ESEMPIO

Prendiamo a pretesto il noto problema del contadino che deve attraversare un fiume trasportando un lupo una pecora e un cavolo . Proviamo ad individuare un algoritmo , ovvero scriviamo a parole quello che deve fare il contadino .

Trasporta pecora altra sponda
Torna indietro
Trasporta cavolo altra sponda
Carica pecora
Torna indietro
Scarica pecora

Trasporta lupo altra sponda
Torna indietro
Trasporta pecora altra sponda

Altro problema

Disponendo di una bilancia a due piatti in grado di valutare se i due pesi sono uguali o diversi individuare quale tra tre monete è quella falsa sapendo che la moneta falsa ha un peso diverso rispetto a quelle buone.

Algoritmo risolutivo

Peso M1 e M2 (peso due monete)
Se sono uguali M3 è falsa
Altrimenti lascio M1 e sostituisco M2 con M3
Se sono uguali M2 è falsa
Altrimenti M1 è falsa.
(Sono sufficienti due pesate per risolvere)

Ancora più complicato

Ora le monete sono quattro , di cui una sola falsa.

Algoritmo risolutivo

Peso M1 e M2
Se sono uguali sono entrambe buone (caso 1)
Altrimenti una delle due è falsa.(caso 2)
Sostituisco M2 con M3
caso 1 : Se sono uguali M4 è falsa
Altrimenti M3 falsa
caso 2 : Se sono uguali M2 è falsa
Altrimenti M1 falsa

Calcolo del MCD

Dati due numeri naturali non entrambi nulli trovare il loro MCD (Massimo Comune Divisore)

ESEMPIO : A=25 B=15 MCD = 5

Algoritmo di Euclide

a) MCD tra a e 0 è a
b) MCD tra a e b è uguale a MCD tra b ed il resto della divisione tra a e b

Proviamo l'algoritmo con a = 25 e b = 15 .

MCD (25,15) → MCD(15,10) → MCD(10,5) →MCD(5,0) → 5

Soluzione linguaggio naturale

Per calcolare il MCD tra a e b :

Copia il valore di a in x e quello di b in y .

Se y=0 allora x è il MCD cercato

altrimenti poni in y il resto della divisione tra x e y

e sostituisci il contenuto di x con quello di y

e ripeti il procedimento.

Soluzione pseudo codice

X← a , Y ← b

Finchè Y ≠ 0

R ← resto (X : Y)

X← Y , Y← R

Fine finche

MCD = X

Soluzione java (programma)

```
public  int  MCD ( int a , int b)  {
   int x = a ;  int y = b ; int r = 0 ;
    while ( y != 0 ) {
                       r = x%y ;
                       x= y ;
                       y= r ;
                      }
 return  x ;
}
```

Esempi iin pseudocodice

Su un gocciolatoio sono disposti dei piatti piani e delle fondine in modo da occupare tutte le posizioni. Si vuole sistemare ordinatamente i piatti in modo da avere tutti i piatti piani sulla sinistra e le fondine sulla destra. Si suppone di poter eseguire uno scambio fra due piatti, prendendo un piatto alla volta in ciascuna mano e senza la possibilità di appoggiare i piatti fuori dal gocciolatoio.

Algoritmo 9 - Algoritmo di ordinamento dei piatti

```
1: Considera la fila dei piatti da ordinare;
2: poni le mani alle estremità della fila dei piatti;
3: while le mani non si sono incontrate do
4:    while la mano sinistra è su un piatto piano do
5:       avanza mano verso destra;
6:    end while
7:    while la mano destra è su un piatto fondo do
8:       avanza mano verso sinistra;
9:    end while
10:   if i piatti che hai in mano sono fuori posto then
11:      scambiali
12:   end if
13:   sposta la mano sinistra di un posto a destra;
14:   sposta la mano destra di un posto a sinistra;
15: end while
16: la fila di piatti è ordinata.
```

Si ha un mucchio di sfere di acciaio uguali fra loro, un numero sufficientemente grande di pesi di k unità ed una bilancia a due piatti. Determinare quanto pesa una sfera (si supponga che il numero delle sfere e dei pesi a disposizione sia sufficientemente grande e che il peso delle sfere sia commensurabile con k).

Algoritmo 10 - Algoritmo di pesatura di una sfera

```
1: metti un peso sul piatto P;
2: metti una sfera sul piatto S;
3: while i piatti della bilancia non sono in equilibrio do
4:    if il piatto P è più alto (pesa meno) del piatto S then
5:       aggiungi un peso sul piatto P;
6:    else
7:       aggiungi una sfera sul piatto S;
8:    end if
9: end while
10: sia m il numero dei pesi sul piatto P;
11: sia n il numero delle sfere sul piatto S;
12: una sfera pesa m * k/n unità.
```

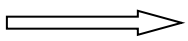
Esercizi per casa

Di ogni problema , si provi ad individuare l'algoritmo risolutivo.

La stesura dell'algoritmo deve avvenire definendo la sequenza di semplici azioni da svolgere.

1. Dato un numero intero dire se è pari o dispari.
2. Dati tre numeri interi dire quale sia il maggiore.
3. Dati tre numeri interi dire se possono essere i lati di un triangolo.
4. Dati due numeri calcolare il loro Massimo Comune Divisore. (MCD)
5. Dati due numeri calcolare il loro Minimo Comune Multiplo. (mcm)
6. Dato un numero intero dire se è un numero primo.
7. Dati due intervalli chiusi di numeri interi dire se si intersecano.
   Esempio : [2, 6] e [4, 9] → [4,6]
8. Data una bilancia a due piatti , in grado di valutare se i pesi sono uguali o diversi , e tre monete di cui una è falsa , individuare la moneta falsa sapendo che ha un peso diverso da quelle buone.
9. Come esercizio precedente ma con quattro monete di cui una è falsa.
10. Come esercizio precedente ma con dodici monete di cui una falsa.
11. Come calcolare le radici della equazione di secondo grado ax^2+bx+c.
12. Dato un numero intero n calcolare la somma di tutti i numeri compresi tra 1 e n .
13. Dato un numero intero calcolarne il fattoriale. Si ricorda che il fattoriale di un numero è ottenuto moltiplicandolo per il prodotto di tutti i numeri interi che lo precedono . Esempio : fatt(4)=4*3*2*1.
14. Dato un numero intero calcolarne il valore della serie di Fibonacci. Si ricorda che la serie di Fibonacci è prodotta in questo modo :
    Fib(0)=0 , Fib(1)=1 e Fib(n) = Fib(n-1)+Fib(n-2)
15. Dire se due regine poste su una scacchiera si trovano sotto presa , ovvero se sono sulla stessa riga o sulla stessa colonna o sulla stessa diagonale.
16. Data una sequenza di quattro numeri interi (a,b,c,d) e un operatore di ordinamento , sort(x , y) , che inverte la posizione dei due numeri se y < x , si trovi il modo di ordinare in modo crescente i quattro numeri della sequenza.

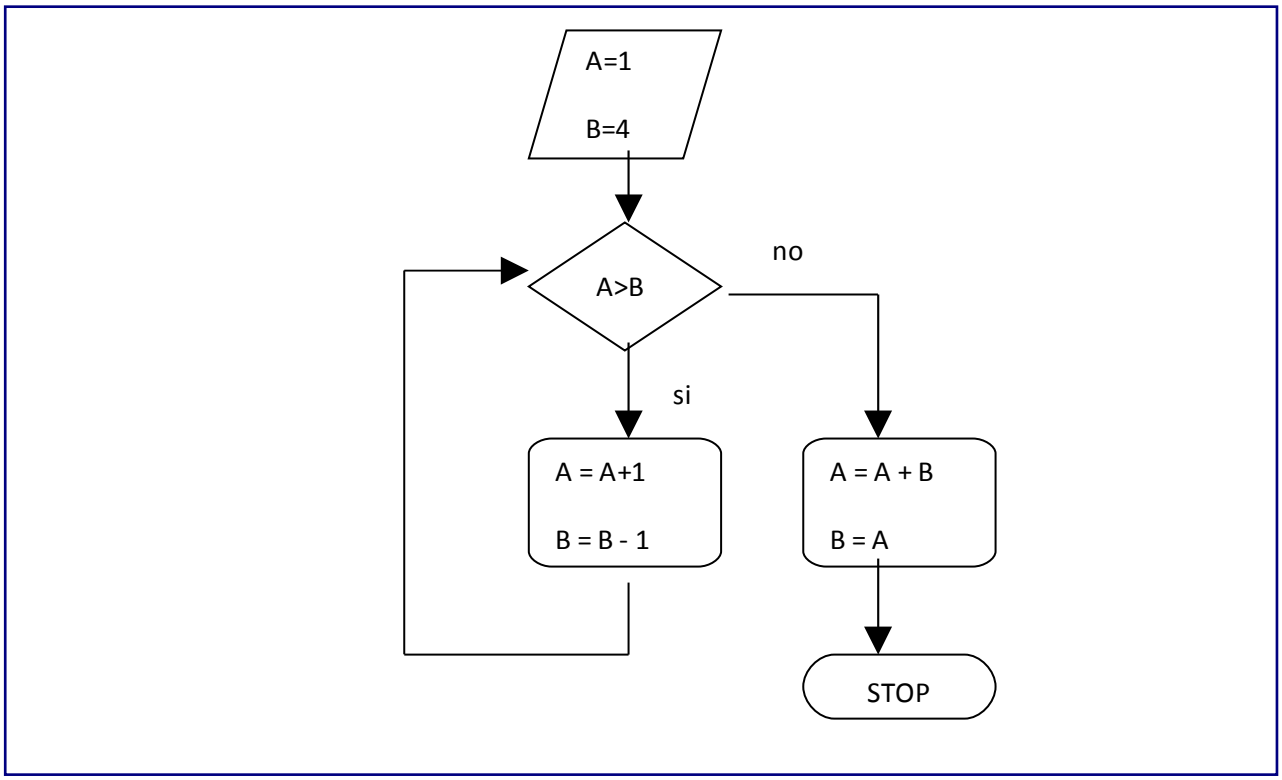
pos	A	B	C	D
num	3	5	4	7



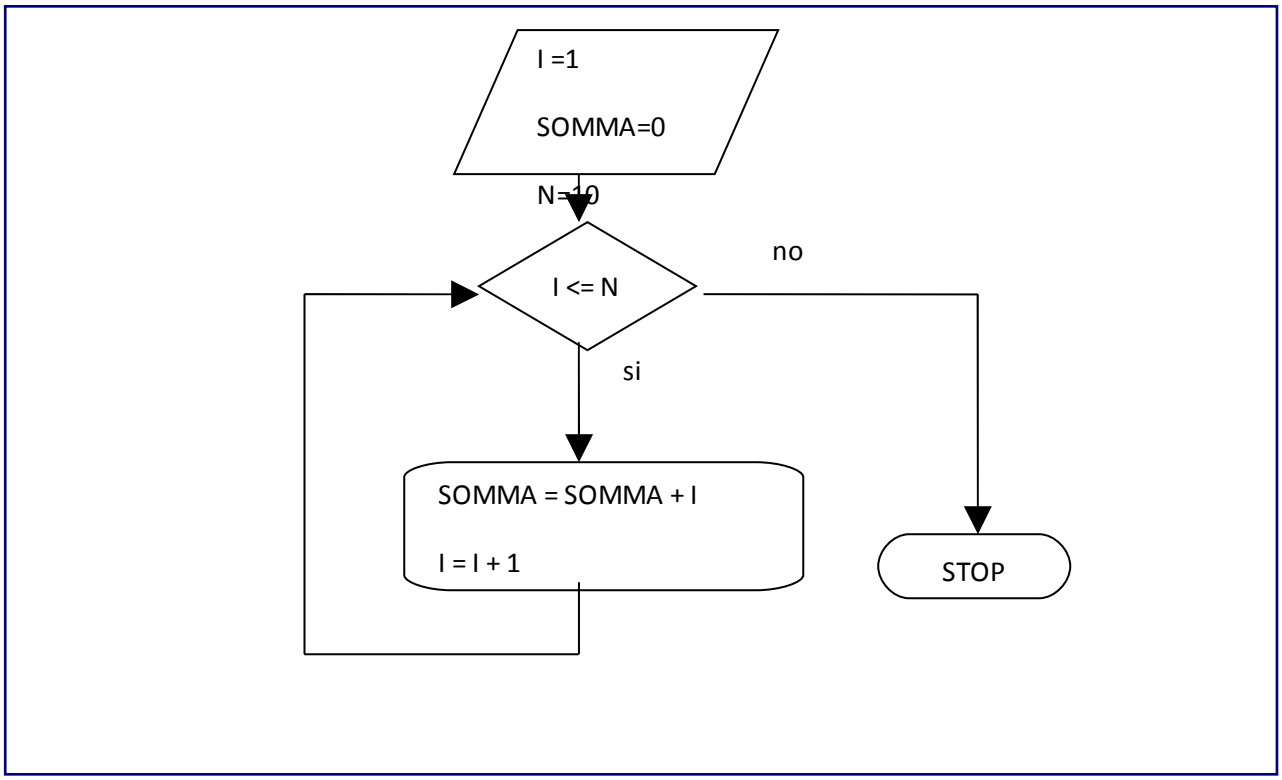
sort (b,c)

pos	A	B	C	D
num	3	4	5	7

Diagrammi di Flusso



Quanto valgono le variabili A e B al termine del flusso ?



Quanto valgono le variabili I , SOMMA e N al termine del flusso ?

Quale dei problemi precedenti risolve il seguente flusso ?

Esempi di pseudocodice (linguaggio di comodo)

1. Nella ipotesi che a = 8 , b = 2 , c = 12 , quale valore verrà stampato utilizzando il seguente pseudocodice

   se a > b **allora** massimo = a
   altrimenti massimo = b
   se c > massimo **allora** massimo = c
   stampa massimo

2. Nella ipotesi che i = 1 , a = -4 , n = 6 , quale sequenza di stampa verrà visualizzata.
3.
   Finché (i < n) **and** (i > a) **or** (i=2) **fai**
   i = i + 2
   n = n – 1
   a = a + 2
   stampa a
   stampa i
   stampa n
   fine fai

4. Si supponga di disporre della funzione **resto (a , b)** in grado di valutare il resto della operazione di divisione a : b .
   esempi :

   a = resto (9 , 4) (a = 1)
   b = resto (11, 3) (b = 2)
   Nella ipotesi che i = 0 quale sequenza verrà visualizzata
   Finché (i < 13) **fai**
   r = **resto** (i , 3)
   se r=0 **stampa** i
   i = i +1
   fine fai

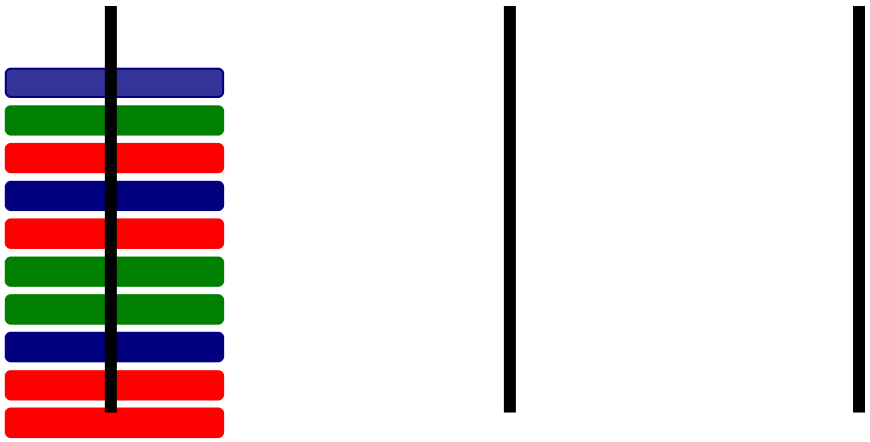
Quale problema risolve il programma in pseudocodice ?

Esercizi di LOGICA

La Pila A contiene dischi di tre colori : Verde , Rosso, Blu.

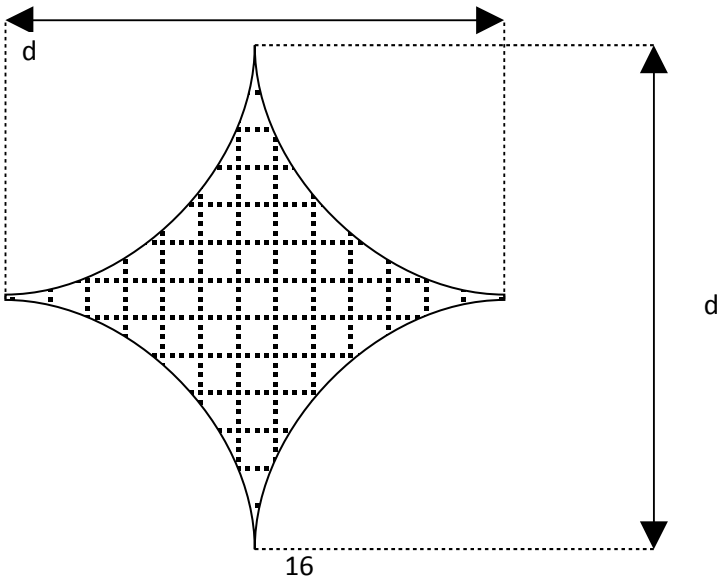
Le Pila B e C sono inizialmente vuote.

Spostando un solo disco alla volta , da una pila ad un'altra pila , fare in modo che la Pila A contenga solo dischi verdi la Pila B solo dischi rossi e la Pila C solo dischi blu.



PILA A **PILA B** **PILA C**

- Disponete di un cerino , di una miccia lunga 130 centimetri e di una forbice . Vi è noto che la miccia brucia in 24 minuti . Vi chiudono in una stanza , senza orologio , e vi chiedono di uscirne dopo 21 minuti . Come operate per risolvere il problema ?
- Calcolare l' area del disegno sapendo che la distanza d = 1 metro



PROBLEMI INTRATTABILI e IRRISOLTI

Un problema è INTRATTABILE se la sua complessità computazionale lo rende di fatto non risolvibile , è IRRISOLTO se non esiste o non si è ancora individuato un algoritmo di risoluzione .

ESEMPI

- Calcolare le radici della eseguente equazione diafontea :

  $3x^2y + 5y^2 - 7xy + x - 3 = 0$ (IRRISOLTO)

- Problema del commesso viaggiatore (INTRATTABILE)
- Problema dei colori (IRRISOLTO)
- Dato un numero a 1000000000000 cifre verificare se è primo (INTRATTABILE)

COMPLESSITA' COMPUTAZIONALE

La complessità computazionale di un algoritmo è la quantità di operazioni (e di tempo) necessarie per produrre la soluzione . Indicando con n la dimensione dei dati da trattare si hanno le seguenti complessità :

SIMBOLO	COMPLESSITA'
O (n)	LINEARE
O ($lg_2 n$)	LOGARITMICA
O (n^2)	QUADRATICA
O (2^n)	ESPONENZIALE
O (n!)	FATTORIALE

ESEMPI di COMPLESSITA'

- Trovare il massimo tra n numeri -> $O(n)$
- Cercare un numero tra n numeri ordinati → $O(\lg_2 n)$
- La torre di Hanoi → $O(2^n)$
- Trovare un numero in una matrice n x n → $O(n^2)$
- Commesso viaggiatore → $O(n!)$

La complessità computazionale è una misura di EFFICIENZA dell'algoritmo

+ efficiente (veloce) - efficiente (lento)

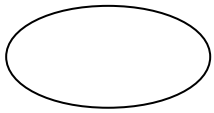
$$O(\lg_2 n) > O(n) > O(n^2) > O(2^n) > O(n!)$$

MODELLI GRAFICI

Sono modelli utilizzati in ambito INFORMATICO che consentono una rappresentazione grafica di un algoritmo

- FLOW CHART
- DIAGRAMMA di TRACCIA
- DIAGRAMMA delle CLASSI
- DIAGRAMMA di SEQUENZA
- MODELLO ER

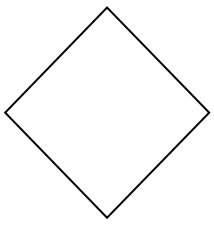
Il Flow Chart (o diagramma di flusso) è un modello che utilizza dei simboli grafici di tipo geometrico , i simboli sono :



OVALE : indica inizio o fine del diagramma

TRAPEZIO : indica operazioni di I/O (lettura o scrittura)

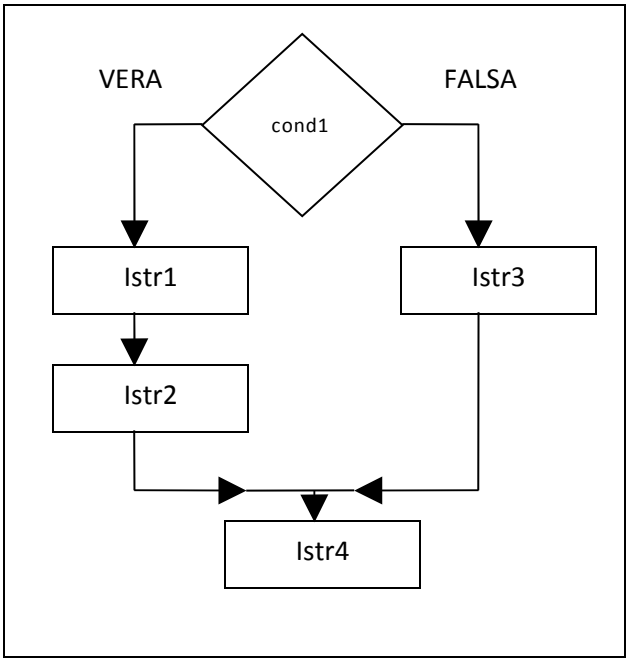
RETTANGOLO : indica operazioni di assegnamento



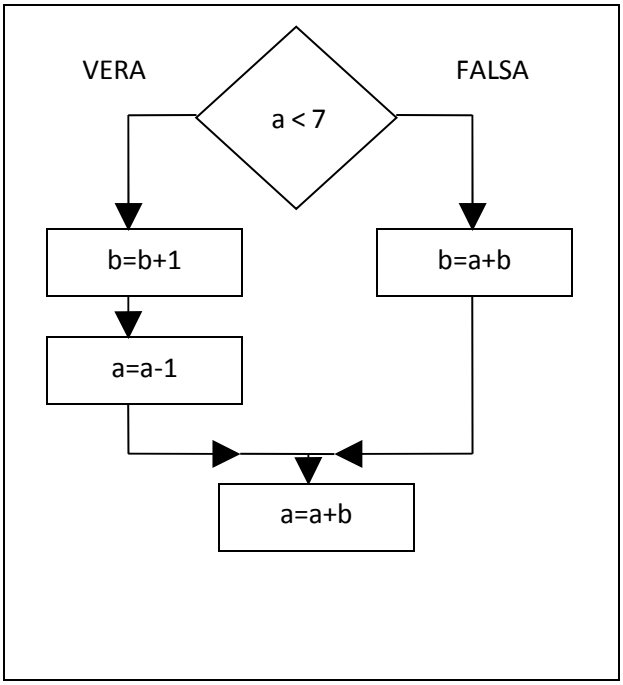
ROMBO : indica operazioni di verifica di condizione

Facciamo subito un esempio per capire come usare e leggere i flow chart

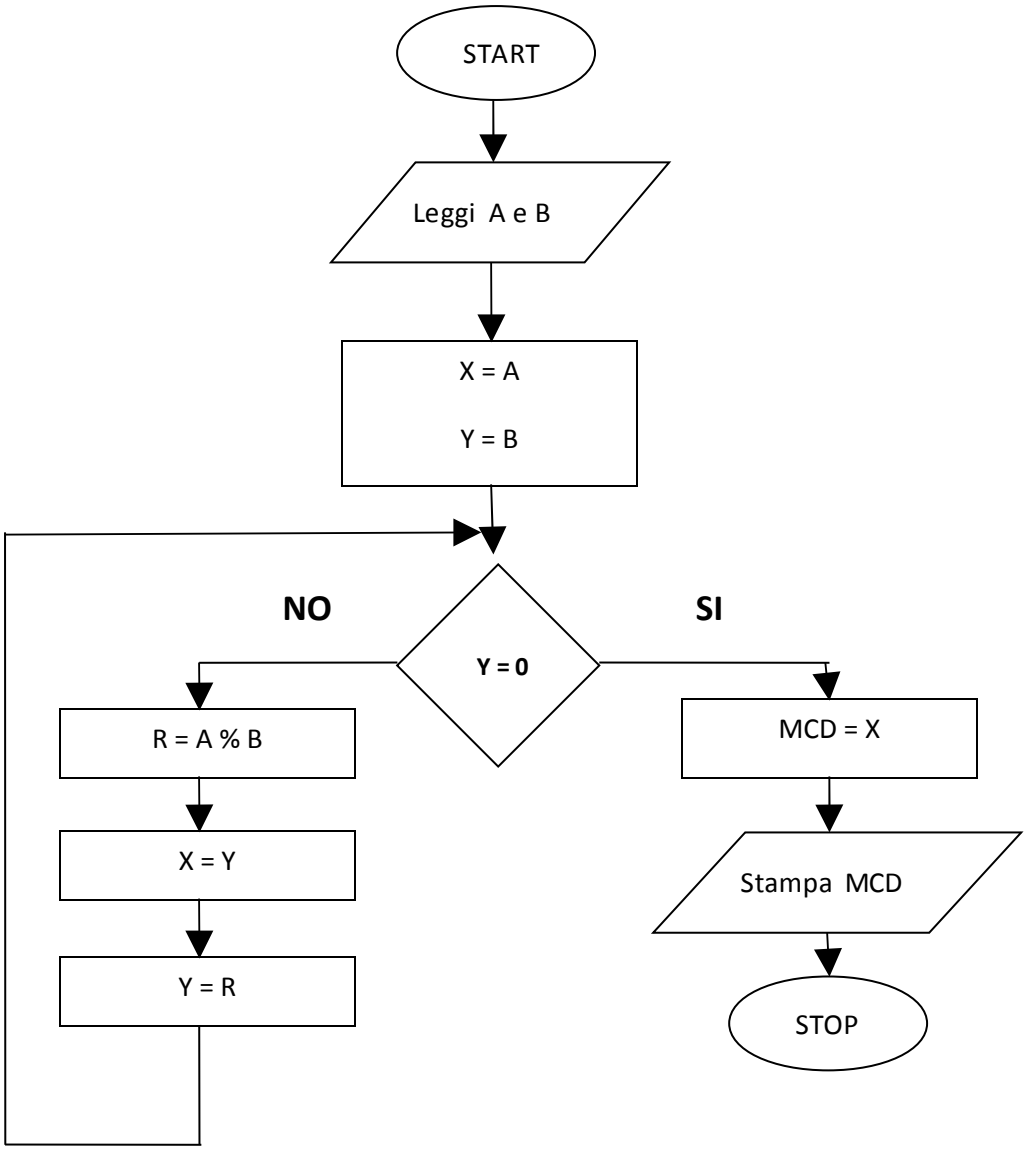
```
Se ( cond1 ) {
                Istr1
                Istr2
              }
Altrimenti   {
                Istr3
              }
Istr4
```




```
if ( a < 7 ) {
                b = b+1
                a = a -1
            }
else        {
                b = a +b
            }
a = a +b
```



Vediamo un esempio noto , ovvero il calcolo del MCD .



Molto più utile il Diagramma di traccia , da usare per analizzare gli errori di programmazione , usatelo per analizzare (debug) i vostri programmi .

Consente di rappresentare l'evoluzione dinamica del programma , ovvero i valori assunti dalle variabili quando il programma va in esecuzione .

Il diagramma di traccia è una tabella in cui sono presenti le seguenti colonne

- Colonna delle istruzioni
- Una colonna per ogni variabile presente nell'algoritmo
- Una colonna per ogni condizione presente nell'algoritmo
- Una colonna per le operazioni di input ed una per quelle di output

Vediamo un esempio per chiarirci le idee. Calcolo del MCD

Leggi a

Leggi b

X← a

Y ← b

Finchè Y ≠ 0

R ← resto (X : Y)

X← Y , Y← R

Fine finche

MCD = X

Stampa MCD

istruzione	a	b	X	Y	R	Y ≠ 0	MCD	INPUT	OUTPUT
Leggi a	9							9	
Leggi b		4						4	
X← a			9						
Y ← b				4					
Y ≠ 0						Vero			
R ← resto (X : Y)					1				
X← Y			4						
Y← R				1					
Y ≠ 0						Vero			
R ← resto (X : Y)					0				
X← Y			1						
Y← R				0					
Y ≠ 0						Falso			
MCD = X							1		
Stampa MCD									1

Esercizio : Calcolare il diagramma di traccia nel caso in cui A = 24 e B = 9

Introduzione a JAVA

Java è un linguaggio orientato agli oggetti (OOP) di tipo general purpose ovvero atto allo sviluppo di programmi di ogni genere , con le seguenti caratteristiche :

- Multi piattaforma e Portabilità
- Grafica indipendente dalla piattaforma (awt , swing)
- Linguaggio ad oggetti puro
- Forte controllo dei tipi
- Supporto a : programmazione web
  sicurezza
  robustezza
  internazionalizzazione
- Gestione ottimale del multithread (multitasking)
- Librerie molte ricche

Come si dichiarano i dati

Nella stesura di un programma è necessario utilizzare dei dati , i dati sono definiti variabili o attributi .

La dichiarazione di una variabile ha la seguente forma :

```
Tipo      nomeVariabile ;
Tipo      nomeVar1 , nomeVar2 ;
```

ESEMPI :

```
int    voto ;
float  lato, altezza ;
int    numero = 4 ;    // dichiarazione ed inizializzazione
```

TIPI STANDARD in JAVA

Primitiva	*Dimensione*	*Val. minimo*	*Val.Massimo*
boolean	1-bit	-	-
char	16-bit	Unicode 0	Unicode $2^{16} - 1$
byte	8-bit	-128	+127
short	16-bit	-2^{15}	$+2^{15} - 1$
int	32-bit	-2^{31}	$+2^{31} - 1$
long	64-bit	-2^{63}	$+2^{63} - 1$
float	32-bit	IEEE754	IEEE754
double	64 -bit	IEEE754	IEEE754
void	-	-	-

Un dato può essere come una costante , ovvero non modificabile ; anteponendo il modificatore final , ovvero : final int PIPPO = 12 ;

GLI OPERATORI in JAVA

Per modificare i dati sono necessari gli operatori , vediamo i più significativi :

Operatori Aritmetici Binari		
Operatore	**Utilizzo**	**Descrizione**
+	res=sx + dx	res = somma algebrica di dx ed sx
-	res= sx - dx	res = sottrazione algebrica di dx da sx
*	res= sx * dx	res = moltiplicazione algebrica tra sx e dx
/	res= sx / dx	res = divisione algebrica di sx con dx
%	res= sx % dx	res = resto della divisione tra sx e dx

Operatori Relazionali		
Operatore	**Utilizzo**	**Descrizione**
>	res=sx > dx	res = true se e solo se sx è maggiore di dx
>=	res= sx >= dx	res = true se e solo se sx è maggiore o uguale di dx
<	res= sx < dx	res = true se e solo se sx è minore di dx
<=	res= sx <= dx	res = true se e solo se sx è minore o uguale di dx
!=	res= sx != dx	res = true se e solo se sx è diverso da dx

Le tabelle di VERITA'

AND (&&)		
sx	**dx**	**res**
true	true	true
true	false	false
false	true	false
false	false	false

OR (||)		
sx	**dx**	**res**
true	true	true
true	false	true
false	true	true
false	false	false

NOT (!)	
sx	**res**
true	false
false	true

XOR (^)		
sx	**dx**	**res**
true	true	false
true	false	true
false	true	true
false	false	false

Operatori Condizionali		
Operatore	**Utilizzo**	**Descrizione**
&&	res=sx && dx	AND : res = true se e solo se sx vale true e dx vale true, false altrimenti.
||	res= sx || dx	OR : res = true se e solo se almeno uno tra sx e dx vale true, false altrimenti.
!	res= ! sx	NOT : res = true se e solo se sx vale false, false altrimenti.
^	res= sx ^ dx	XOR : res = true se e solo se uno solo dei due operandi vale true, false altrimenti.

L' Operatore di assegnamento

Dopo aver svolto delle operazioni si desidera assegnare il risultato ad una variabile , la cosa è possibile utilizzando l'operatore = , che assegna alla variabile il risultato .

ESEMPIO :

```
int a = 2 ;
int  somma = 3 + a ;
```

STRUTTURA DI UN PROGRAMMA IN JAVA

```
/**
 * Primo esempio di programma in java
 * autore : S. Cecchin
 * data: 10/10/2009
*/

public class Prova {
  public static void main(String[] args){
    System.out.println("Ciao Mondo" );
  }
}
```

Istruzioni di controllo

Sono le istruzioni che consentono di controllare e condizionare l'esecuzione del programma .

Istruzioni per il controllo di flusso	
Istruzione	**Descrizione**
if	Esegue o no un blocco di codice a seconda del valore restituito da una espressione booleana.
if-else	Esegue permette di selezionare tra due blocchi di codice quello da eseguire a seconda del valore restituito da una espressione booleana.
switch	Utile in tutti quei casi in cui sia necessario decidere tra opzioni multiple prese in base al controllo di una sola variabile.
for	Esegue ripetutamente un blocco di codice.
while	Esegue ripetutamente un blocco di codice controllando il valore di una espressione booleana.
do-while	Esegue ripetutamente un blocco di codice controllando il valore di una espressione booleana.

L'istruzione if semplice

Consente di valutare una condizione o più condizioni definite da una espressione logica e di eseguire un blocco di istruzioni se la condizione è soddisfatta.

```
if ( condizione ) {
                    istr 1;
                    istr2;
                   ……..
                  }
```

L'istruzione if else

Consente di valutare una condizione o più condizioni definite da una espressione logica e di eseguire un blocco di istruzioni o un blocco di istruzioni alternative in funzione della condizione . Se vera si svolge il blocco dopo if , se falsa si esegue il blocco dell'else.

```
if ( condizione ) {
                    istr 1;
                    istr2;
                  }
```

```
else            {
                 istr3;
                }
```

L'istruzione if nidificata

Viene utilizzata quando le condizioni da verificare sono molte e per ogni situazione si vuole attivare una diversa strategia .

```
if( condizione1 ) {
                   blocco1 ;
                   }
 else if ( condizione2 ) {
                            blocco2;
                           }
   else if ( condizione3 ) {
                            blocco3;
                            }
        else               {
                            blocco4
                            }
```

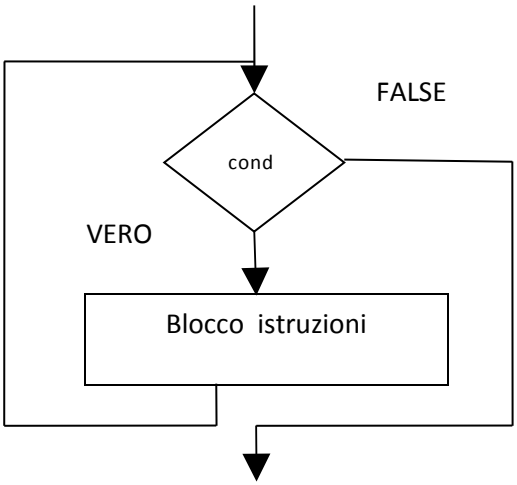
L'istruzione switch

Si usa quando si deve valutare il valore di una variabile che può assumere una serie prestabilita e nota di valori .

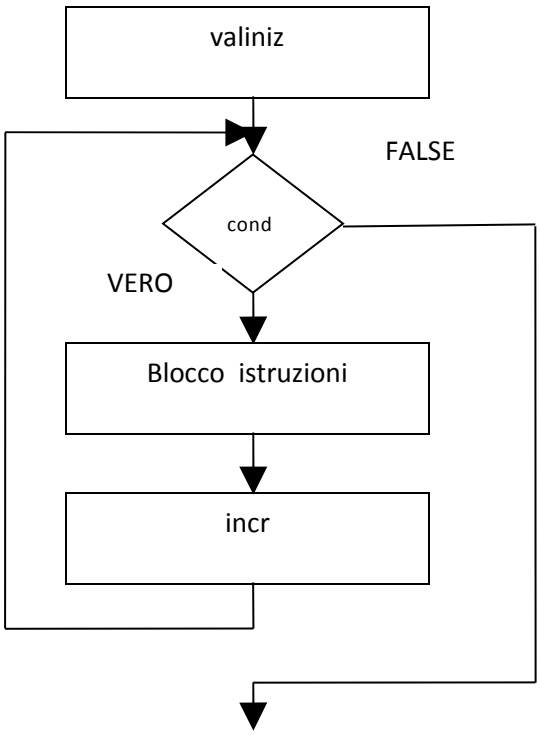
```
switch ( variabile ) {
  case   valore1 :  blocco1 ;
                    break ;
  case   valore2 :  blocco2;
                    break;
     …
     …

  default :         bloccoN ;
}
```

Cicli in java

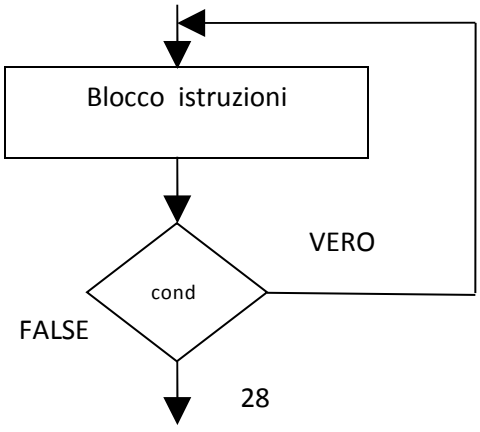
Ciclo while : **while** (condizione) { blocco di istruzioni }



Ciclo for : **for** (valiniz ; cond ; incr) { blocco di istruzioni }



Ciclo do while : **do** { blocco di istruzioni } **while** (condizione)



Furbate nei cicli

1. for(; ;) → ciclo infinito (loop)
2. while(true) → idem
3. while (condizione) → equivale a for (; condizione ;)

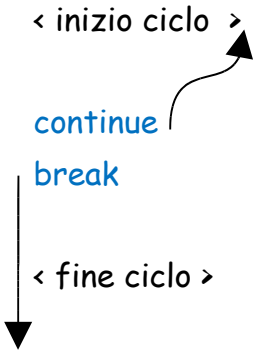
Traduzione di for in while

for (valiniziale ; condizione ; incremento) { …. }

```
valiniziale ;
while( condizione ) {
                        ....
                        incremento
                        }
```

Due istruzioni particolari da usare nei cicli

- break , consente di uscire dal ciclo
- continue , si passa all'inizio del ciclo successivo



Grammatiche

• Un **linguaggio di programmazione** è una notazione comprensibile ad un calcolatore per rappresentare un algoritmo. Un linguaggio è formata da :

- Alfabeto , insiemo finito e non vuoto di simboli (a, b , 3 ,4 ,? >...)
- Lessico , insieme di stringhe (parole) formate con elementi dell'alfabeto (casa , for , +23,3)
- Frase , insieme di parole appartenenti ad un definito lessico posti in sequenza secondo una opportuna sintassi (o grammatica) (if (a>9) ..)

• Un **programma** è un algoritmo scritto in un linguaggio di programmazione
Usando il concetto di frase, sintassi e semantica illustrate in precedenza .

• Un **programma** è una **frase** scritta in un linguaggio di programmazione ed analizzabile dal punto di vista

- LESSICALE : si verifica che ogni parola della frase sia corretta
- SINTATTICO: si verifica la forma linguistica in cui è scritto
- SEMANTICO : si valuta il significato della frase (che sia di senso compiuto)

ESEMPIO :

- Mario è andato a squola (errore lessicale)
- Io ho andato (corretto lessicalmente , errore sintattico)
- La penna sta mangiando (corretto lessicalmente e sintatticamente , errore semantico)

• La **sintassi** specifica come **costruire un programma**

• La **semantica** definisce il **significato del programma**

• Per i linguaggi di programmazione la grammatica deve essere espressa in modo formale, non ambiguo. Gli strumenti più usati sono

1. **notazione Backus-Naur (Backus-Naur Form, BNF)**
2. **diagrammi sintattici**

Introduzione alla BNF

Esempio di grammatica BNF per descrivere i numeri reali, quali 3.14

<numero-reale> ::= <sequenza-cifre> . <sequenza-cifre>
<sequenza-cifre> ::= <cifra> | <cifra> <sequenza-cifre>
<cifra> ::= 0|1|2|3|4|5|6|7|8|9

• Simboli primitivi o simboli terminali : . 0 1 2 3 4 5 6 7 8 9
• Tramite < ... > si denota un costrutto o simbolo non terminale
• Il simbolo ::= significa "è" mentre il simbolo | significa "oppure", quindi

– <cifra> è 0 oppure 1 oppure 2 ...
– <sequenza-cifre> è una cifra oppure una cifra seguita da una sequenza di cifre
– <numero-reale> è una sequenza di cifre seguita da . e da una sequenza di cifre

• Per distinguere le diverse occorrenze dello stesso costrutto si usa il pedice
<numero-reale> ::= <sequenza-cifre>$_1$. <sequenza-cifre>$_2$

Definizione di GRAMMATICA con BNF

Con la notazione BNF una grammatica è definita dai seguenti quattro elementi

- **Vt**: alfabeto di **simboli terminali** denotati con `simbolo terminale`
- **Vn**: alfabeto di **simboli non terminali** denotati con <simbolo non terminale>
- **P**: insieme di **regole di produzione**

<simbolo-non-terminale> ::= stringa$_1$ | stringa$_2$ | ... | stringa$_n$
dove stringa$_i$ denota una qualsiasi sequenza finita di simboli terminali e non terminali

- **S**: un simbolo non terminale detto **scopo della grammatica**

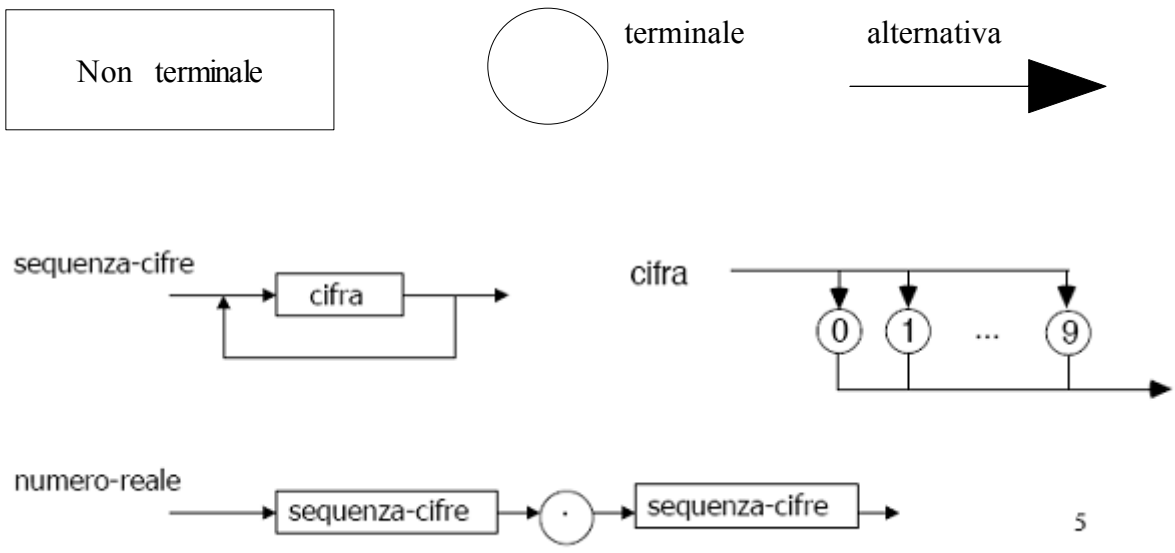
Esempio di grammatica con BNF

- **Vt** = {.,0,1,2,3,4,5,6,7,8,9}
- **Vn** = {<numero-reale>, <sequenza-cifre>, <cifra>}
- **P** = {

<numero-reale> ::= <sequenza-cifre> . <sequenza-cifre>,
<sequenza-cifre> ::= <cifra> | <cifra> <sequenza-cifre> ,
<cifra>::= `0 |1|2|3|4|5|6|7|8|9`
}

- **S** = {<numero-reale>}

Un esempio di **diagramma sintattico:**



BNF estesa

X ::= [a]b equivale a X ::= b|ab

X ::= $\{a\}_n$ b equivale a X ::= b|ab|aab|… ripetendo a fino a n volte

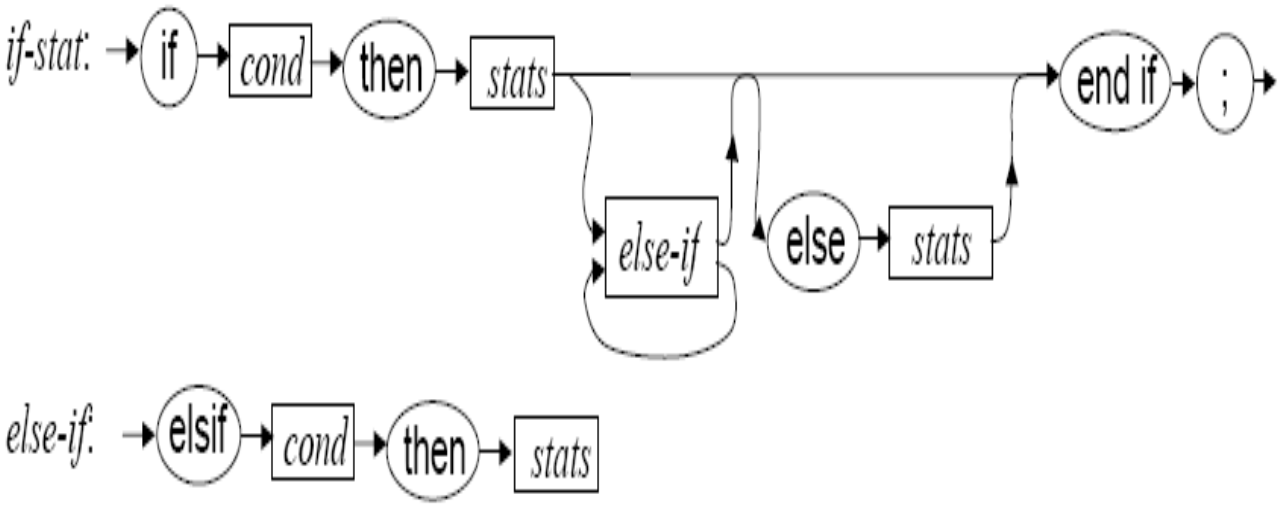
X ::= {a}b equivale a X ::= b|ab|aab|… ripetendo a un numero di volte indefinito

Equivale ad avere nella grammatica la produzione
X ::= b | aX (ricorsiva)

Diagrammi sintattici

Istruzione di Selezione, ovvero if then … else if …

if-stat → if *cond* then *stats* { *else-if* } [else *stats*] end if ;
else-if → elsif *cond* then *stats*



Esempio di BNF: interi con segno

- **Usiamo la BNF estesa per esprimere la grammatica dei numeri interi con segno**
- **Considerare il caso particolare del numero zero**
- **Vt** = {0,1,2,3,4,5,6,7,8,9}
- **Vn** = {<intero>, <intero-senza-segno>, <sequenza-cifre>,<cifra-non-nulla>,<cifra>}
- **P** = {

<intero> ::= [+ | -] <intero-senza-segno>,
<intero-senza-segno> ::= <cifra> | <cifra-non-nulla> <sequenza-cifre> ,
<sequenza-cifre> ::= <cifra> | <cifra> <sequenza-cifre>,
<cifra> ::= <cifra-non-nulla> | 0,
<cifra-non-nulla>::= 1|2|3|4|5|6|7|8|9
}

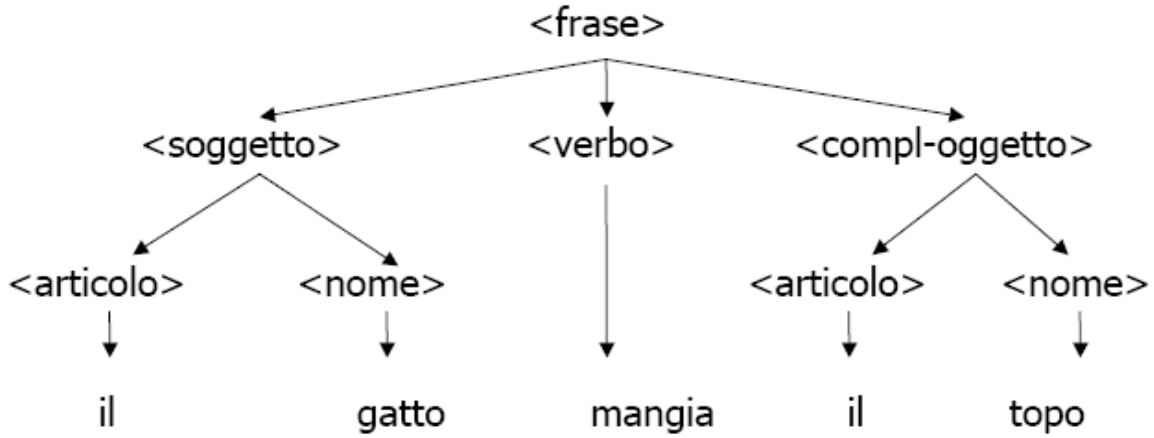
- **S** = {< intero >}

Scopo di una Grammatica

- **Generazione di frasi (programma)**: dallo scopo applicare ripetutamente le produzioni fino ad ottenere una stringa composta di soli simboli terminali
- **Convalida di frasi (compilatore)**: data una stringa composta di soli terminali, verificare se c'è una sequenza di produzioni che la genera a partire dallo scopo
- Il processo di derivazione può essere descritto tramite un **albero sintattico**

Esempio di albero sintattico

- **Vt** = {il, lo, gatto, topo, sasso, mangia, beve }
- **Vn** = {<frase>, <soggetto>, <verbo>, <compl-oggetto>, <articolo>}
- **P** = { <frase> ::=<soggetto><verbo><compl-oggetto>
  <soggetto> ::=<articolo><nome>
  <articolo> ::= il | lo
  <nome> ::= gatto | topo | sasso
  <verbo> ::= mangia | beve
  <compl-oggetto> ::=<articolo><nome> }
- **S** = {<frase>}

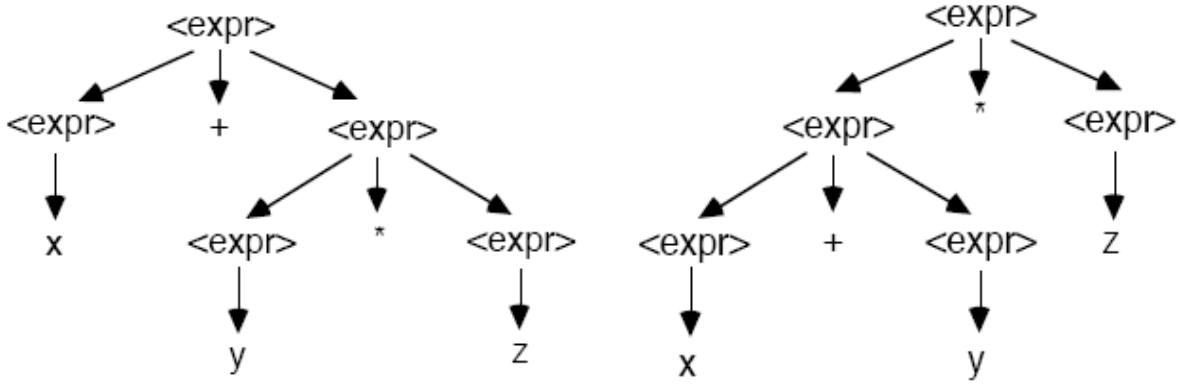


Grammatica ambigua

<expr> ::= x | y | z | <expr> | <expr> + <expr> | <expr> * <expr>

Esempio di Grammatica ambigua: genera lo stesso elemento con due alberi sintattici:

- la grammatica di un linguaggio di programmazione **non deve essere ambigua**



Linguaggio

- un linguaggio può essere generato da più di una grammatica
- Alcuni **aspetti contestuali** non sono rappresentabili in notazione BNF o diagrammi sintattici

– Ad esempio, non è possibile rappresentare tramite la notazione BNF il fatto che non ci possono essere due variabili con lo stesso nome, cioè che una variabile deve essere dichiarata una (ed una sola) sola volta

- ANALIZZATORI (operano in successione quando si compila)

– **analisi lessicale**: controlla che i simboli utilizzati appartengano all'alfabeto
– **analisi grammaticale**: verifica il rispetto delle regole grammaticali
– **analisi sintattica contestuale**: verifica restrizioni di tipo contestuale (unica dichiarazione degli identificatori, tipi di dati …)

ESEMPI sulle grammatiche

Data la seguente GRAMMATICA

Vt = (a, b, c) *terminali*

Vn = (A, B , C) *non terminali*

P = (cB ::= Bc , bB ::= bb) *produzioni*

S ::= aSBc | abc *scopo*

4. Verificare se la parola **aabbcc** appartiene al vocabolario
5. Verificare se la parola **abbc** appartiena al vocabolario

Soluzione :

S→aSBc→a abc Bc→ aab cB c →aabBcc → aa bB cc → aa bb cc (si)

S→aSBC→aabcBc → stop (no)

Costruire la GRAMMATICA

Definire una grammatica in grado di produrre le parole casa e ciao nel suo vocabolario (la soluzione non è unica) ,si utilizzi :

Vt = (a, i, o, s, c)

Soluzione :

Vn = (X)

S ::= X

P =(X::=a | aX , X::= iX, X::=o , X::= sX , X::= cX)

1) X→ cX → caX→ casX → casa

2) X→cX→ciX→ciaX→ciao

Costruire la GRAMMATICA

Costruire la grammatica dei numeri pari .

Soluzione

a. Vt = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
b. Vn = { *num_pari* , *num*}
c. P = {

   <num_pari>::= 0 | 2 | 4 | 6 | 8 |<num><num_pari>

   <num> ::= 0 | 1 | 2 | … | 8 | 9 | <num> 0 | <num>1|… |<num> 9

   }
d. S = num_pari

Verifica

- 64932 è ottenibile?
  - <num_pari>::= <num><num_pari> ::=<num>2 ::= <num>32 ::= <num>932 ::=<num>4932 ::= 64932

ESERCIZI per casa

1. Scrivere una grammatica che generi tutte e sole le stringhe che iniziano per a e finiscono per c . (ovvero ac , aaabc , abcbc ..)

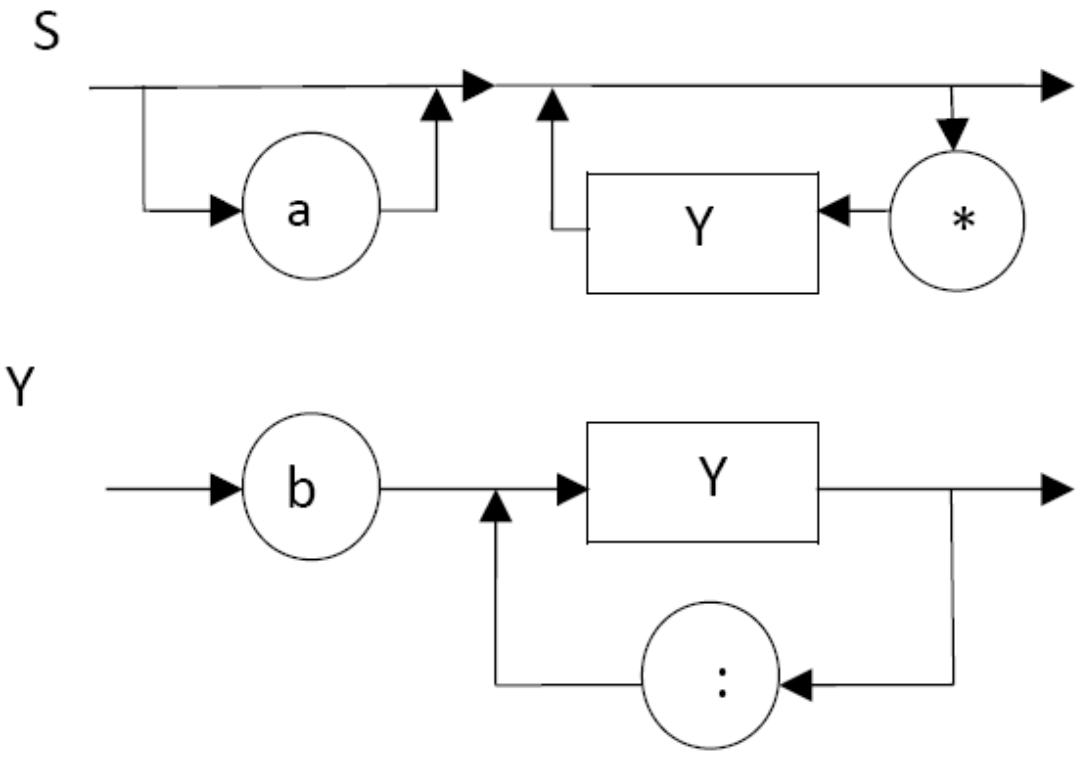
   V_t = (a , b , c)

2. Scrivere una grammatica che generi tutte e sole le stringhe del tipo :

   ac , aacc , aaabccc (con uguale numero di a e di c)

   V_t = (a , b , c)

3. Dato il seguente diagramma sintattico , con S come scopo :



DOMANDE

- Dire quali sono i simboli terminale e quelli non terminali
- Dire quali delle seguenti stringhe appartengono alla grammatica

  b , a * b , a:b , a*ba:a , a
- Tradurre i diagrammi in BNF , ovvero definire la grammatica

PROLOG e LISP

Affrontiamo ora due particolari linguaggi di programmazione :

PROLOG → PROgrammazione LOGica

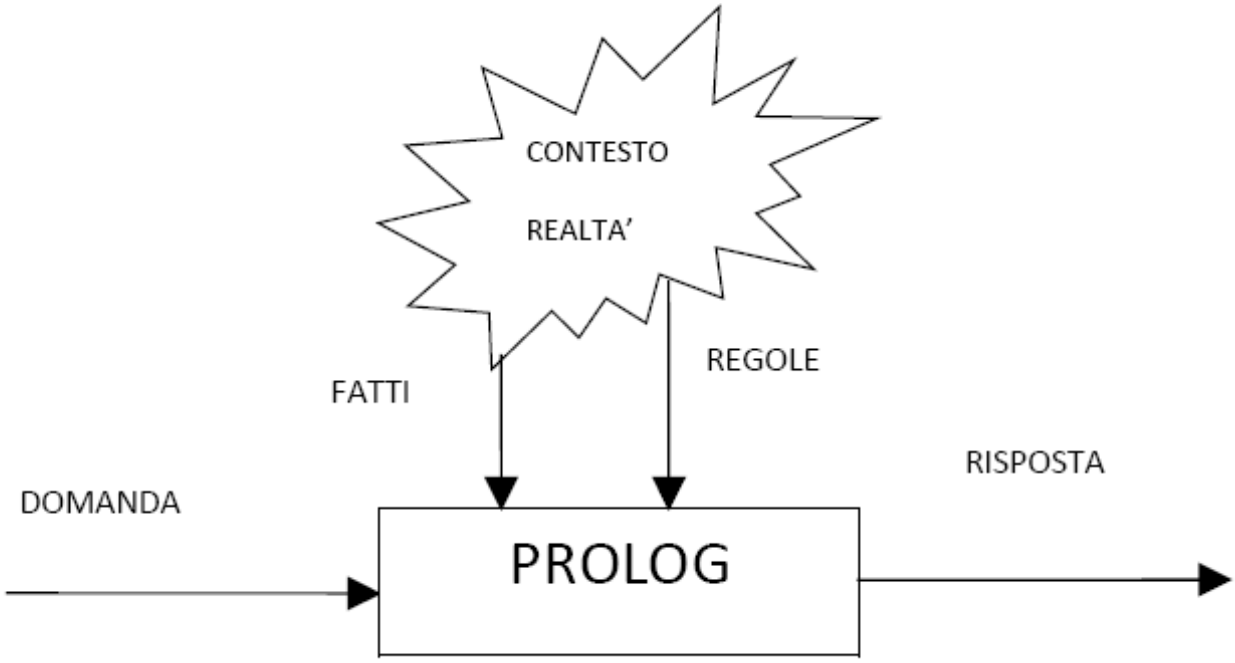
LISP → LISt Programm

Sono linguaggi che richiedono un atteggiamento mentale diverso ma risultano performanti se applicati in alcuni ambiti , ad esempio in quello della intelligenza artificiale.

La PROGRAMMAZIONE LOGICA (PROLOG)

E' una programmazione DICHIARATIVA .

Il programma consiste in una serie di fatti (conoscenze) e di regole (conseguenze logiche) , dopo aver fornito regole e fatti si interroga il programma per avere delle risposte.



Ogni sistema intelligente si basa sulla capacità di :

- utilizzare le conoscenze (premesse note)
- operare secondo logica deduttiva (inferenza logica)

Un programma PROLOG è formato da :

- conoscenze → predicati con argomenti **ama (piero , lucia)**
- regole induttive → **studente(X) :- uomo(X) , studia(X)**
-

ATTENZIONE : in minuscolo gli argomenti costanti (piero)

in maiuscolo le incognite (X , COSA)

CARATTERI PARTICOLARI : _ → qualsiasi , X \= Y , X == Y

Come si scrive un programma prolog

- **Serve la capacità di scomporre il problema individuando gli elementi base della realtà che si vuole descrivere**
- **Risolvere i casi semplici con regole semplici e banali**
- **Definire le regole generali che descrivono il problema**
- **Con regole logiche sfruttando le regole generali precedenti costruire le soluzioni dei casi meno semplici**
- **Sfruttare la propria LOGICA DEDUTTIVA**

ESEMPI di PROLOG

La realtà che vogliamo descrivere è la seguente :

- **Socrate è un uomo**
- **Cecchin è un dio**
- **Tutti gli uomini sono mortali**
- **Il figlio di un uomo è un uomo**
- **Un dio è un figo**

PROGRAMMA

uomo(socrate). // fatto (a)

dio(cecchin). // fatto (b)

mortale(X):-uomo(X). // regola (c)

uomo(figlio(Y)):-uomo(Y). // regola (d)

figo(X):-dio(X). // regola (e)

Interroghiamo il programma con le seguenti domande :

? mortale(figlio(figlio(socrate))) (è mortale il nipote di socrate)

Risposta : uomo(socrate) → (d) uomo(figlio(socrate)) → (d) uomo(figlio(figlio(socrate)))

→ (c) mortale(figlio(figlio(socrate)))

Siamo partiti da un fatto (non confutabili) e per INFERENZA LOGICA si è dimostrato che è VERO .

? figo(CHI) (chi è figo)

Risposta : dio(cecchin) → (e) figo(cecchin) :- dio(cecchin)

→ CHI = cecchin

Proviamo ora a descrivere una realtà matematica , ovvero l'insieme dei numeri naturali e le operazionei di somma e prodotto sui naturali . Facciamo una premessa , l' insieme dei numeri naturali N e un' opportuna successione di simboli secondo la seguente tabella :

```
0 →  0
1 →  s(0)
2 →  s(s(0))
3 →  s(s(s(0)))
  ...
n →   s(s(..s(0)))..)
```

Definiamo i numeri naturali :

a) naturale(0).

b) naturale(s(N)) :- naturale(N).

Le operazioni

c) somma(0,X,X). → 0+X = X

d) somma(s(X),Y,S(Z)) :- somma(X,Y,Z).

e) prodotto(0,Y,0). → 0*Y= 0

f) prodotto(s(X),Y,Z) :- ptodotto(X,Y,W) , somma(Y,W,Z).

L'unica cosa da chiarire è la regola (f) , vediamo di spiegarla :

X * Y = W

Y + W = Z → Y + (X * Y) = Z → Y + XY = Z → Y (1 + X) = Z → Y + S(X) = Z

Interroghiamo il programma con le seguenti domande :

? naturale(X). (quali sono i naturali)

X = 0

ancora ? s

X= s(0)

ancora ? s

X=s(s(0))

. . . . .

? somma(s(0), s(s(0)),X) (quanto fa 1 + 2)

X= s(s(s(0)))

ancora ? s

NO

? somma(X, s(0),s(s(s(0)))). (quanto fa 3-1)

X= s(s(0))

? somma(X , s(s(0)),s(0)) (quanto fa 1-2)

NO (non esisono naturali negativi)

? somma(X, Y , s(s(0))) (quali sono coppie X Y che sommati danno 2)

X=0 , Y = s(s(0)).

Ancora ? S

X= s(0) , Y= s(0)

Ancora ? S

X = s(s(0)) , Y = 0

Ancora ? S

NO

? prodotto(s(s(0)) , s(s(s(0)))) (quanto fa 2 per 3)

? prodotto (X , Y , s(s(s(s(s(o))))))) , somma(X,Y , s(s(s(s(0))))))

Esistono due naturali X e Y la cui somma è 5 ed il cui prodotto è 6 .

? prodotto(s(s(0)) , X , H) , somma(H , Y , s(s(s(s(0)))))) , somma(s(0) , Y , X)

Risolve il sistema $2X + Y = 5$

$X - Y = 1$

ESEMPI da studiare

1) Prendiamo in esame la realtà residenza delle persone

```
abita (marco,roma).
abita (giovanni,milano).
abita (piero,parigi).
abita (tiziana,roma).
capitale (roma,italia).
capitale (parigi,francia).
```

Per rispondere alla domanda chi abita in una capitale? Bisogna aggiungere

```
abita_in_capitale(marco).
abita_in_capitale(piero).
abita_in_capitale(tiziana).
```

Alternativa (migliore)

```
abita_in_capitale(X):- abita (X,Y) capitale(Y ,_).
```

"_" sta, in questo caso, per "stato qualsiasi"

2) Secondo esempio , realtà orario scolastico

FATTI
insegna(ricci,inglese,classe_2A).
insegna(giorgi,matematica,classe_1B).
insegna(rippi,italiano,classe_1B).
frequenta(giorgio,classe_1B).
frequenta(maria,classe_2A).
frequenta(mirco,classe_1B).

REGOLE
insegnante_di(Professore,Ragazzo):- insegna(Professore,_,Classe), frequenta(Ragazzo,Classe).

Se faccio questa domanda cosa ottengo :
insegnante_di(ricci,Ragazzo).
insegnante_di(Professore,giorgio).
insegnante_di(Professore,Ragazzo).

Sapreste costruire una REGOLA per conoscere i compagni di classe
compagno_classe (X,Y):- frequenta(X,C), frequenta(Y,C).

Sapreste costruire una REGOLA per conoscere tutti gli insegnanti di una classe
insegnante_classe (X , C):- insegna(X,_,C);

Sapreste costruire una REGOLA per conoscere tutti gli insegnanti di una materia
insegnante_materia (X , M):- insegna(X,M,_);

3) Dato il seguente programma prolog

```
ladro(tony).
possiede(mara,denaro).
possiede(mara,libro).
piace(tony,X):-possiede(X,denaro).
puo_rubare(X,Y):-ladro(X), piace(X,Y).
```

La seguente interrogazione cosa significa e cosa torna, spiegare perché

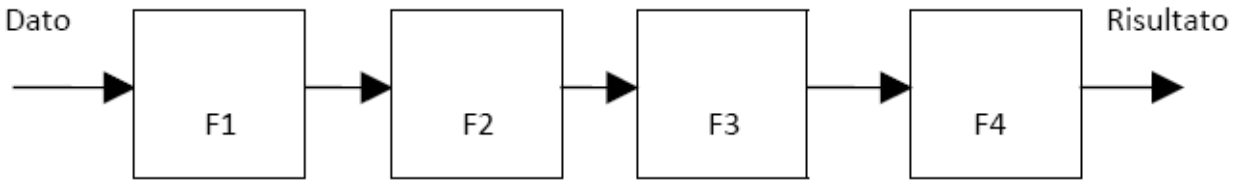
```
puo_rubare(tony,X)
```

4) Dato il seguente programma prolog

```
p(a,b).
p(b,c).
q(X):-p(X,Y).
q(f(X)):-q(X).

Quali sono le risposte alle interrogazioni
     ? p(a,c).         R( no )
     ? p(a,X),p(X,c).  R( X=b)
     ? p(X,Y),q(X).    R(X=a,Y=b ; X=b,Y=c)
     ? q(f(X)).        R( X=a ,f(a),f(f(a)) X=b ,f(b),f(f(b)) )
```

La PROGRAMMAZIONE FUNZIONALE (LISP)

Si basa sul concetto matematico di funzione e sfrutta la composizione di più funzioni per produrre la soluzione . La programmazione funzionale risulta elegante nella forma , sintetica in scrittura e chiara .



Una trattazione completa di questo linguaggio richiederebbe lo studio delle liste e delle principali operazioni su di esse . Limitiamoci ad alcuni esempi in cui si sfrutta la ricorsione come strumento di soluzione dei problemi .

Esempio di programmazione FUNZIONALE

Proviamo a descrivere la funzione RESTO .

a) RESTO (X , Y) → X se X < Y
b) RESTO (X , Y) → RESTO (X – Y , Y) se X > =Y

Vediamo se funziona :
RESTO (3 , 5) → (a) = 3 , infatti 3 / 5 fa 0 con resto di 3
RESTO (10 , 4) → (b) RESTO (6 , 4) → (b) RESTO (2 , 4) → (a) = 2

Proviamo a descrivere la funzione MCD .

a) MCD (X , 0) → X
b) MCD (X; Y) → MCD (Y , RESTO (X, Y)) se Y \= 0

Si osservi che la soluzione proposta sfrutta la definizione della funzione RESTO data precedentemente .

Vediamo con un esempio come funziona : MCD (45 , 30)
MCD (45 , 30) → MCD (30 , RESTO(45,30)) → MCD (30 , RESTO(15,30)) → MCD (30 , 15) → MCD(15 , RESTO(30 , 15)) → MCD (15 , RESTO(15 , 15)) → MCD (15 , RESTO(0 , 15)) → MCD(15 , 0) → 15

Vediamo degli esempi

```
; Esempio: fattoriale
(defun factor (x)
                    (if (= x 0) 1 (* x (factor (- x 1)))))

;Per interrogare (factor 4)

; Esempio: calcolo successione di Fibonacci
(defun fib (n)
      (if (<= n 1) 1 (+ (fib (- n 1)) (fib (- n 2)))))

;Per interrogare (fib 5)

; Esempio: massimo comune divisore
(defun resto (x y)
                    (if (< x y) x (resto (- x y) y)))

(defun mcd (x y)
                    (if (= y 0) x ( mcd y (resto  x y))))

;Per interrogare (  mcd ( 45,  30  ) )

; Verifica se un numero è primo
  (defun resto (x y)
                    (if (< x y) x (resto (- x y) y)))

  (defun primo (n r)
    (if(>= r n) true ( if (= (resto n r) 0 ) false (primo n (+ r 1)))))

; Per interrogare ( primo 13 2 )
```

LA RICORSIONE

Quando la soluzione di un problema si ottiene ricorrendo al problema stesso , ovvero qundo un algoritmo (metodo) fa riferimento a se stesso .
In altre parole all'interno dell'algoritmo (metodo) c'è una chiamata a se stesso.

- Ci sono problemi naturalmente ricorsivi
- Ogni problema ricorsivo è anche iterativo , non viceversa

Vediamo un esempio per chiarirci le idee.
Calcolare la potenza a^n , con a \= 0 .

SOLUZIONE ITERATIVA

$a^n = 1$ se $n = 0$
$a^n = a\ a\ a\ a\ a\ ..$ se $n \neq 0$

```
potenza ( a , n ) {
  int ris = 1;
  if( n = = 0 )  ris ;
   else for(int i = 1 ; i <= n ; i++ )
          ris = ris * a ;
 return ris;
 }
```

SOLUZIONE RICORSIVA

$a^n = 1$ se $n = 0$
$a^n = a^{n-1}\ a$ se $n \neq 0$

```
potenza ( a , n ) {
  int ris = 1 ;
  if( n = = 0 )  ris ;
   else ris = a * potenza(a , n-1) ;
 return ris ;
 }
```

La soluzione ricorsiva risulta più sintetica e chiara concettualmente , richiede sempre due cose :

a) La legge (o funzione) di chiusura o terminazione , ovvero $a^n = 1$ se $n = 0$
b) La funzione ricorsiva che porta verso la chiusura , ovvero $a^n = a\ a^{n-1}$ se $n \neq 0$

Vediamo in pratica come funziona la soluzione ricorsiva con l' esempio 2^3 , a = 2 n = 3 .

$2^3 \rightarrow$ (b) $2\ 2^2 \rightarrow$ (b) $2\ 2\ 2 \rightarrow$ (b) $2\ 2\ 2\ 2^0 \rightarrow$ (a) $2\ 2\ 2\ 1 = 8$

Temporalmente le cose avvengono seconda la sequenza :

2^3
$2\ \ 2^2$
$2\ \ 2$
$2\ \ 2^0$
1
2
4
8

VANTAGGI

- Algoritmo più facile e naturale
- Algoritmo di facile lettura e comprensione

SVANTAGGI

- Bisogna possedere una logica mentale ricorsiva , non sempre naturale
- L'esecuzione del programma richiede un maggior spazio di memoria (stack overflow)
- L'esecuzione del programma richiede più tempo (più lento)

ESERCIZI sulla RICORSIONE

1. Deteminare la soluzione ricorsiva che consente di calcolare il prodotto di due numeri interi .

2. Deteminare la soluzione ricorsiva che consente di calcolare il fattoriale di un numero.

3. Determinare la soluzione ricorsiva che consente di calcolare la somma degli interi compresi tra N e M (estermi compresi)

4. Determinare la soluzione ricorsiva che consente di invertire una stringa.

5. Determinare la soluzione ricorsiva che consente di calcolare la sequenza di Fibonacci , si ricorda che la sequenza di Fibonacci è la seguente :

   Fib(0) = 1
   Fib(1) = 1
   Fib(2) = Fib(1) + Fib(0)
   …
   Fib(n) = Fib(n-1) + Fib(n-2) // in genere se n> 1

6. Determinare la soluzione ricorsiva che consente di valutare se un numero intero è primo.

```
// Conta le cifre da cui è composto un numero
static long contaCifre(int n) {
                        if(n<10) return 1;
                         else return 1 + contaCifre(n/10) ;
}
// Calcola il Fattoriale di un numero
static long fatt(int n) {
                        if(n==0) return 1;
                         else return n*fatt(n-1);
}

// Calcola il MCD tra due numeri in modo ricorsivo
static int mcdRicorsivo(int n, int m) {
                        if(m == 0) return n;
                         else return mcdRicorsivo(m,n%m);
}

// Valuta se un numero è primo
static boolean primo(int n) {
  return primoAux(n,2);
}

// Funzione ausialiare alla precedente
static boolean primoAux(int n,int a)  {
                        if(n%a==0) return n==a;
                         else return  primoAux(n,a+1);
}

// Calcola il valore di Fibonacci di un numero
static long fib(int n) {
                        if (n==0 || n == 1 ) return 1 ;
                            else return fib(n-1)+fib(n-2) ;
}
// Come il precedente ma usando un algoritmo più efficiente
static long fibVeloce(int n) {
                        if(n==0 || n==1) return 1 ;
                         else return fibAux(0,1,1,n);
}
// Funzione ausiliare alla precedente
static long fibAux(long penultimo,long ultimo,int x, int n) {
                        if (n == x) return penultimo + ultimo;
     else return  fibAux(ultimo,penultimo+ultimo,x+1,n);
}
```

```
// Come calcolare la potenza di un numero
static double potenzaVeloce(double x, int n) {
                        if(n==0) return 1.0 ;
                         else if(n %2 == 0) return quadrato(potenzaVeloce(x,n/2));
    else return x*potenzaVeloce(x,n-1);
}
// Come calcolare il quadrato di un numero
static double quadrato(double x) { return x*x; }
```

La Programmazione Strutturata

Un progrmma si definisce strutturato quando risulta scritto in modo ordinato senza inutili ripetizioni di linee di codice ed è di facile lettura e comprensione . Il programma è suddiviso in procedure (aree autonome di codice) , ed ogni procedura risolve un particolare problema. Le procedure possono ricevere degli attributi e tornare dei dati.

Fase di ANALISI

La cosa più importante nella stesura de programma è la fase di analisi del problema . La gran parte del tempo viene dedicata a questo aspetto.
Una cattiva analisi , o peggio la sua assenza , generano soluzioni inadeguate poco strutturate e con forti limitazioni di sviluppo .
Le metodologie che si possono utilizzare nella fase di analisi sono sostanzialmente tre :

1. TOP DOWN
2. BOTTOM UP
3. MISTO

TOP DOWN , scomposizione del problema in sottoproblemi ed eventuale scomposizione dei sottoproblemi in ulteriori sottoproblemi . La soluzione si costruisce sfruttando le sottosoluzioni prodotte.

BOTTOM UP , parte dal presupposto di riutilizzare il codice già prodotto. Si parte cercando di riconoscere subito i sottoproblemi di cui si dispone di una soluzione e si cerca di costruire la soluzione a sottoproblemi di dimensione maggiore sino a risolvere l'intero problema.

MISTA , si sfruttano contemporaneamente i due metodi .

Alla base della programmazione strutturata c'è il concetto di sottoprogramma che viene concretizzato nella stesura di una procedura o metodo in java .
Un metodo , che è un sottoprogramma , viene scritto una sola volta ma può essere utilizzato più volte dal main o da altri sottoprogrammi.

SINTASSI di un metodo

```
public  static tipo NomeMetodo ( lista ) {
  attributi locali
  corpo
  return  esito  // se non void
}
```

BNF

tipo = void | int | double | float | char | String | array | Object

lista = { attributo , }

attributo = void | tipo variabile

esito = variabile | valoreNumerico

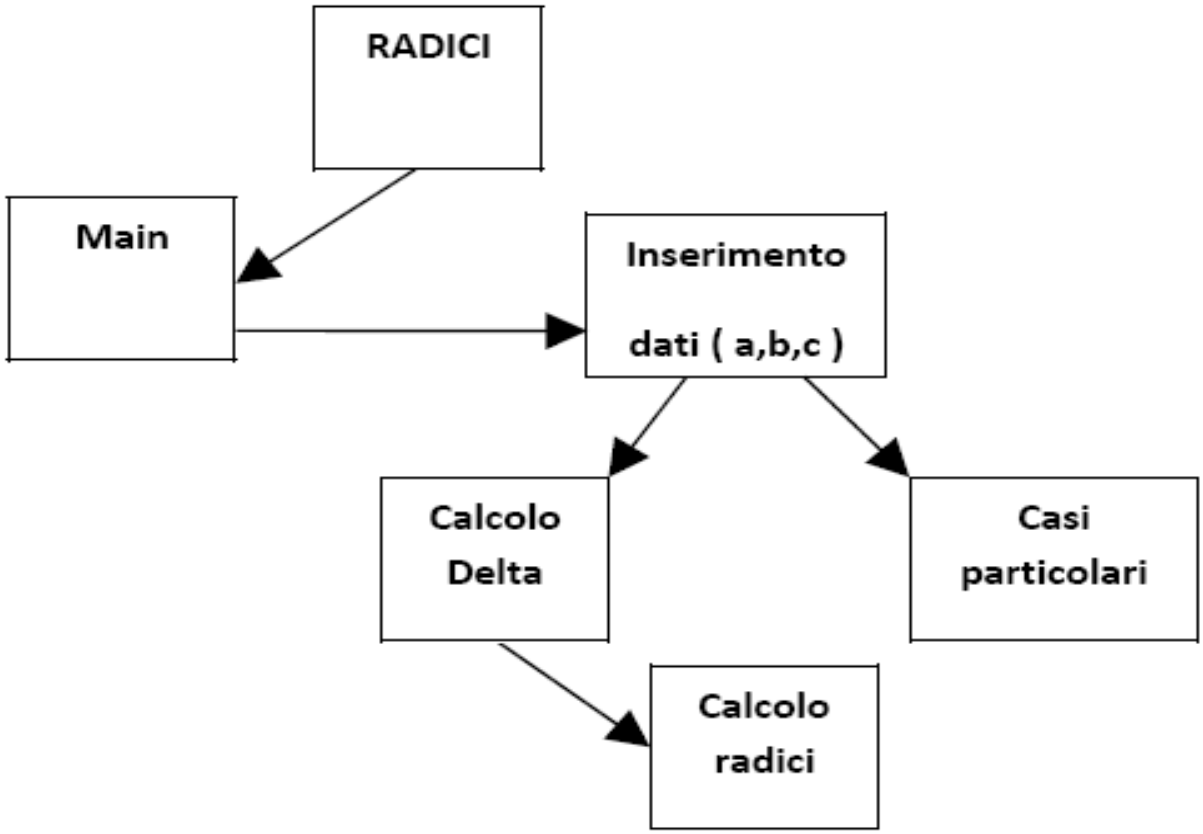
ESEMPIO

```
public static int Somma ( int a , int b) {
int somma = a+b ;
return somma
}
```

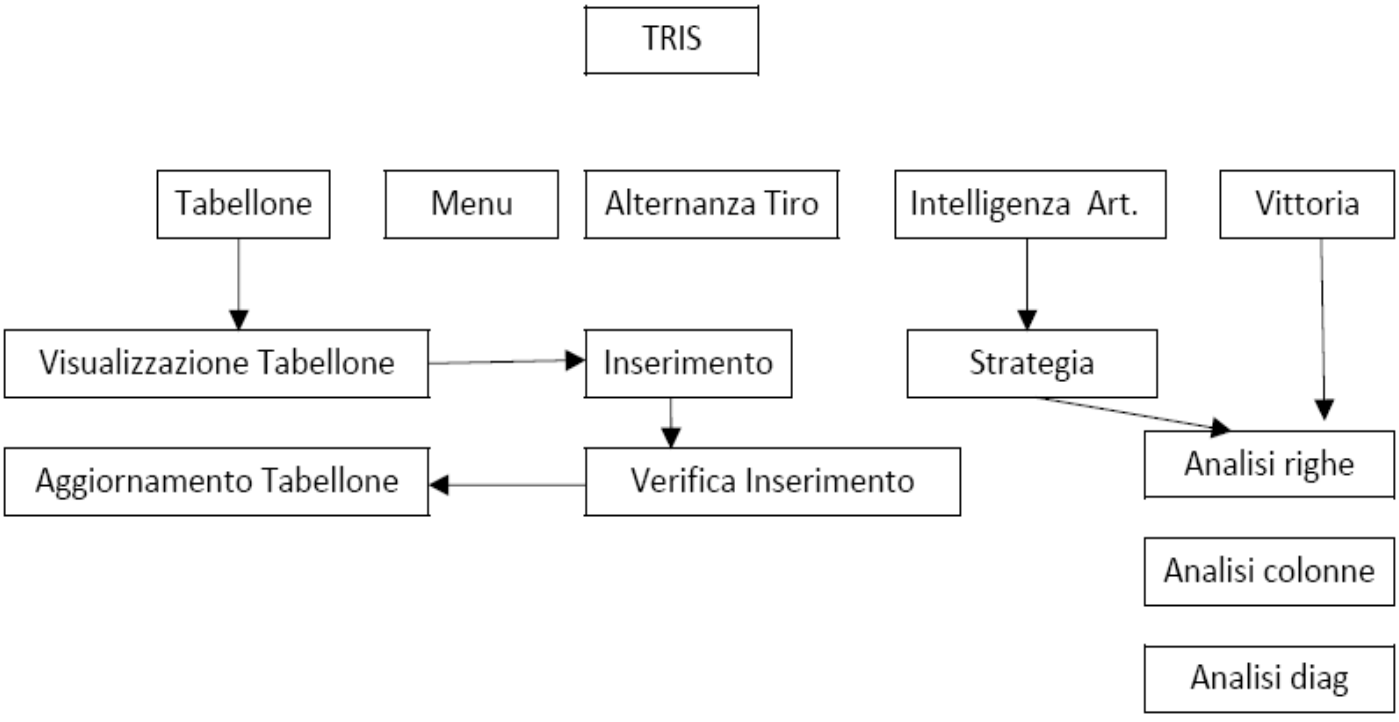
PERCHE' USARE la PROGRAMMAZIONE STRUTTURATA

- Risparmio di codice
- Isola gli ambiti risolutivi
- Favorisce il riutilizzo del codice
- Riduce i tempi di stesura dei programmi
- Consente un risparmio di spazio
- Rende più leggibili i programmi
- Consente più facili modifiche al codice
- Consente una miglior gestione delle variabili

Vediamo un semplice esempio di programmazione strutturata , il calcolo delle radici di una equazione di secondo grado : $ax^2 + bx + c = 0$

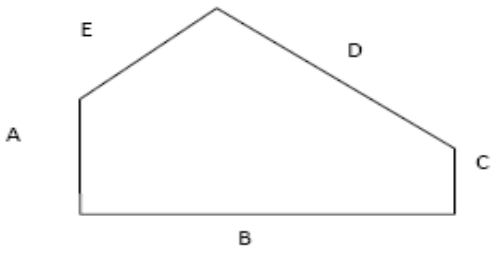


Vediamo un esempio più complesso : il gioco del TRIS . Scomposizione in sottoproblemi



ESERCIZI da SVOLGERE in CLASSE

1. Definire con metodologia TOP DOWN i sottoproblemi che una coppia di sposi con un bimbo piccolo deve risolvere per assistere ad un concerto.
2. Applicando la metodologia BOTTOM UP dire quali problemi si possono risolvere disponendo della soluzione ai seguenti sottoproblemi
   - So cercare una cosa
   - So usare una cosa
3. Utilizzando la metodologia TOP DOWN calcolare l'area della seguente figura



4. Si vuole costruire n programma in grado di svolgere le operazioni di somma e prodotto tra frazioni , ad esempio 2/3 (½ + 2/5) . Quali sottoproblemi bisogna risolvere ?

Esercizi di PROGRAMMAZIONE STRUTTURATA

1. Costruire un programma che consenta di ricevere da tastiera una stringa composta dai soli caratteri a , b ,c ,d . La stringa subisce una traslazione , carattere per carattere , di una posizione , ovvero : a→b , b→c , c→d, d→a . Dopo la traslazione l'intera stringa viene invertita . Implementare i metodi necessari . Si supponga che la stinga cosi ottenuta venga ricevuta da un'altra persona che vuole ricostruire la stringa originale . Implementare i metodi necessari per la ricostruzione .

2. Scrivere un programma in grado di calcolare :
   - L'area di un rettangolo (noti base ed altezza)
   - L'area di un triangolo (noti i tre lati)
   - L'area di un cerchio (noto il raggio)
   - Il volume del cilindro (noto il raggio e l'altezza)
   - Il volume di un cubo (noto il lato)

3. Scrivere un programma in grado di operare sui numeri ,ovvero di :
   - Calcolare il fattoriale di un numero intero (fatt(4) = 4*3*2*1 = 24)
   - Calcolare il numero binario di un numero intero (7→ 111)

4. Scrivere un programma in grado di operare con le stringhe , ovvero di :

- Concatenare due stringhe (al + ba = alba)
- Estrarre la sottostringa (3 caratteri) (classe → cla)
- Cercare una sottostringa in una stringa (cercare st in asta)(boolean)

5. Scrivere un programma che calcoli quanto vi costano o vi rendano NUMGIORNI di interessi in banca sapendo che se siete debitori vi viene applicato un tasso di interesse annuo di TASSODEBITO e se siete in credito vi viene applicato un tasso di interesse annuo pari a TASSOCREDITO.
(ad esempio si può supporre che TASSODEBITO = 12% e TASSOCREDITO = 1%)

6. Costruire una libreria Matematica che contiene una serie di metodi con le seguenti caratteristiche :

- Calcolare il valore assoluto di un numero
- Calcolare la radice quadrata di un numero (la sua parte intera) , ovvero la radice di 38 è 6
- Calcolare l'elevamento a potenza di un numero intero , ad esempio 5^4
- Calcolare la parte intera di un numero , ad esempio 5,67 → 5
- Tradurre in numero intero un numero ricevuto come Stringa , ovvero "23"→ 23

7. Scrivere un programma che sia in grado di calcolare la resistenza equivalente di un circuito elettrico composto da solo bipoli passivi (ovvero con sole resistenze).

La VISIBILITA' degli attributi (o variabili)

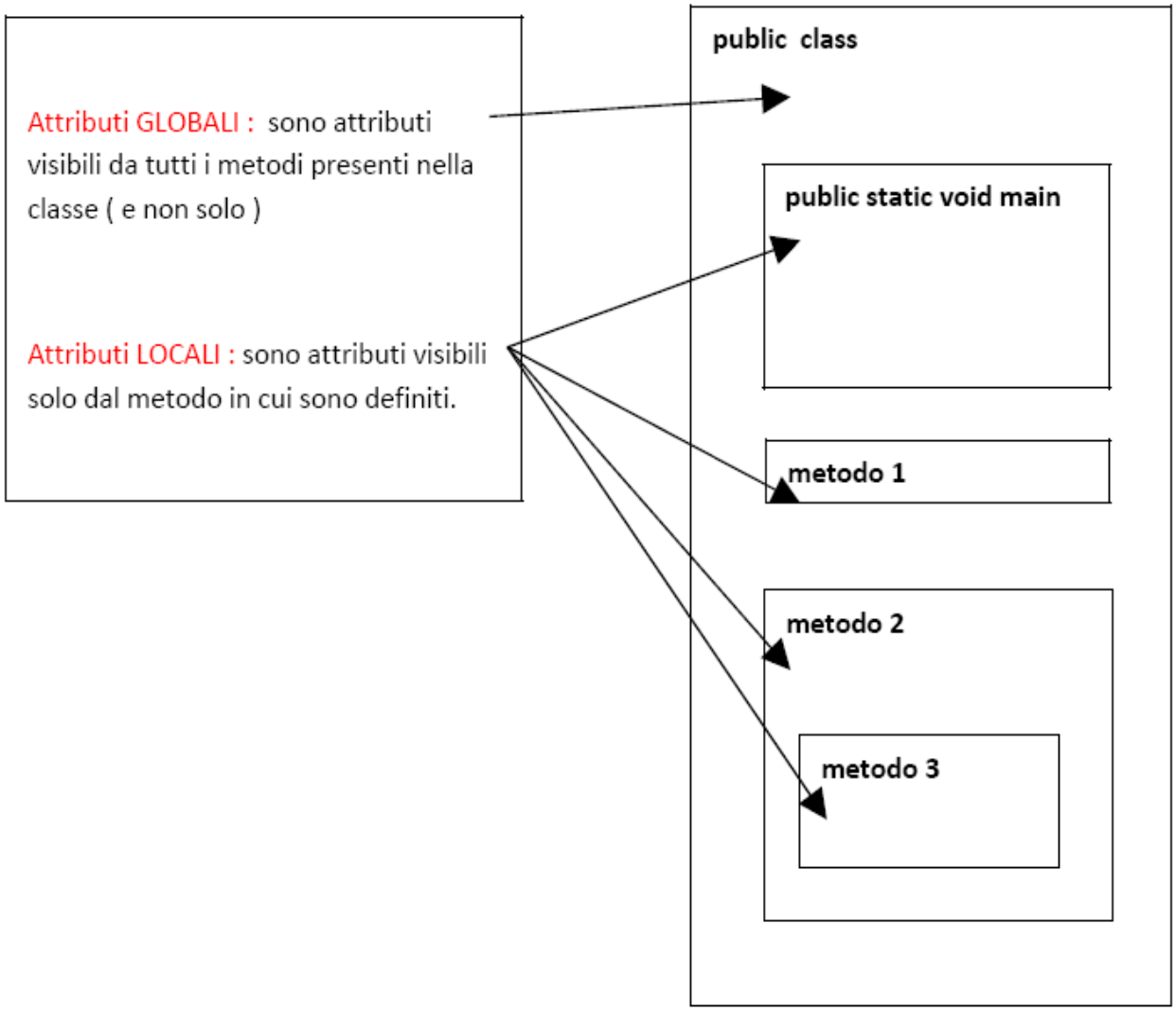
Iniziamo con il dare la definizione di ambiente , si definisce ambiente l'insieme di istruzioni comprese tra due parentesi graffe , ovvero :

```
{
  AMBIENTE
}
```

Quindi sono ambienti :

- una classe
- il main
- ogni metodo
- Le istruzioni for , while …

ATTRIBUTI GLOBALI e LOCALI



Gli Attributi GLOBALI

- Vengono inizializzati una sola volta
- Restano allocate per l'intera durata del programma (Area Dati)

Gli Attributi LOCALI

- Vengono inizializzati ogni volta che si esegue il metodo o ambito
- Restano allocati solo per la durata del metodo (Area Stack)

REGOLE di VISIBILITA'

1. Un attributo è visibile nell'ambito in cui è definito e negli ambiti che gli sono interni
2. Se esistono due o più attributi con lo stesso nome si fa riferimento a quello più vicino (più locale)

Vettori e Matrici

Diamo per prima cosa la definizione di array (vettore) e di matrice .
Definizione : Struttura in grado di contenere più oggetti dello stesso tipo (omogenei)

•A una dimensione (vettore)

[2 , 4 , 6]

•A più dimensioni (matrice)

| 2 5 6 |

| 3 6 7 |

| 2 0 2 |

Per poter usare un array si deve :

1.Dichiarare l' array

2.Definire la sua dimensione

3.Riempirlo di valori

La dichiarazione consiste nella definizione del tipo di oggetti che contiene

ESEMPI

```
int[] i   // vettore di interi
String[] nomi   // vettore di stringhe
float[] numeri   // vettore di float
Studente[] classe  // vettore di oggetti , prossimo anno
```

Definire la dimensione significa definire la quantità di oggetti che contiene (al massimo)

ESEMPI

```
i =  new int [4]
nomi  = new String [10]
numeri = new float[6]
```

Un vettore può essere dichiarto e la sua dimensione può essere definita in modo sintetico con una sola istruzione

```
int i [] = new int[4]
String nome[] = new String[10]
```

Si può dichiarare e definire dimensione e valori contenuti con una sola istruzione

```
int[]  i = { 1 , 4 , -3 }
```

Equivale a :

```
int[]  i
i = new int [3]
i[0] =  1
i[1] =  4
i[2] = -3
```

Chiarimenti importanti

```
int[]  v  =   new int [3]        v →  { null , null  ,  null }
int[]  v  =   { 3 ,  -5  ,  7 }    v →  { 3  , -5  ,  7 }
```

Gli indici per identificare i singoli elementi dell'array partono dal valore 0 ed arrivano al valore numElementi – 1 , ovvero un vettore di dimensione 5 avrà un indice che si muove da 0 a 4 .

Convenzione usate per gli indici :

v[i] v[i] [j] v[i] [j] [k]

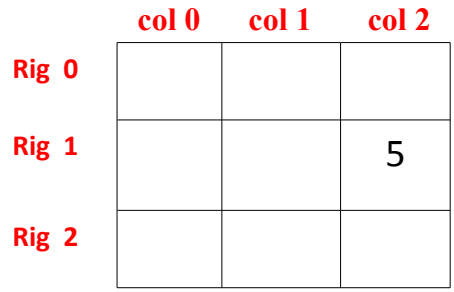
LA MATRICE

E' un vettore bidimensionale

```
int scacchiera[] []
scacchiera = new int [3] [3]
```

ATTENZIONE

scacchiera [RIGA] [COLONNA]



schacchiera [1] [2] = 5

VEDIAMO UN ESEMPIO in JAVA

```
/*
 *  Definizione di un vettore , acquisizione della sua dimensione
 *  e caricamento dei valori da tastiera, sua visualizzazione.
 **/
class CreaInserisci {
   static InOut tastiera = new InOut();
  public static void main(String[] args) {
    // definizione del vettore e della sua dimensione
    int[] vett;
    int dim;
    // acquisizione della dimensione
    dim=dimensioneVett();
    // carico i valori
    vett=carica(dim);
    // visualizzo il vettore
    visualizza(vett);
  }
  static int dimensioneVett(){
          System.out.println("Inserire dimensione vettore :");
          return tastiera.leggiInt();
  }
```

```
 static int[] carica(int quanti){
        int[] a = new int[quanti];
        for(int i=0;i<quanti;i++) {
                        System.out.println("Inserire valore " + i + " : ");
            a[i]= tastiera.leggiInt();
        }
        System.out.println("Fine Inserimento");
   return a;
 }

 static void visualizza(int[] a) {
   System.out.print("VETTORE : ");
   for(int i=0; i<a.length; i++)
          System.out.print(a[i]+" ");
   System.out.println(" ");
 }
}
```

VEDIAMO UN ESEMPIO in JAVA (matrice)

```
 /*
  * Esercizio che cre una matrice 3X3 di numeri interi in grado di calcolare
  * la somma delle singole righe e delle singole colonne
  **/
public class Matrice  {
        static final int dim = 3;
        static int[][] mat  = new int[dim][dim];  // la matrice

  public static void main(String[] args){
   crea();
   visualizza();
   sommaRighe();
   sommaColonne();
```

```
  sommaDiagonali();
 }
 public static void crea(){
        for(int i = 0 ; i < dim ; i++)
         for(int j = 0 ; j < dim ; j++)
          mat[i][j] = (int)(Math.random()*100);
 }
 public static void visualizza(){
        for(int i = 0 ; i < dim ; i++){
         for(int j = 0 ; j < dim ; j++)
          System.out.print(" "+mat[i][j]);
        System.out.println(" ");
        }
 }
 public static void sommaRighe(){
        for(int i = 0 ; i < dim ; i++){
         int somma = 0;
         for(int j = 0 ; j < dim ; j++)
          somma+=mat[i][j];
        System.out.println("Riga "+i+" = "+ somma);
        }
 }
 public static void sommaColonne(){
        for(int j = 0 ; j < dim ; j++){
         int somma = 0;
         for(int i = 0 ; i < dim ; i++)
          somma+=mat[i][j];
        System.out.println("Colonna "+j+" = "+ somma);
        }
 }
}
```

Le Operazioni sui VETTORI e sulle MATRICI

Le operazioni che si svolgono sui vettori sono :

- di inserimento dati
- di visualizzazione dei dati
- l'operazione di ricerca di un dato
- di ordinamento dei dati

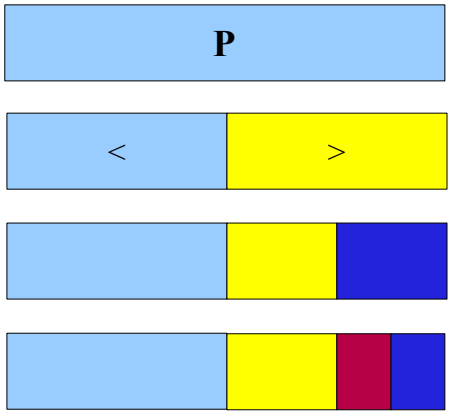
Operazioni poco utilizzate sui vettori

- inversione dei dati
- shiftamento dei dati

Sulle matrici ci si limita alla sola operazione di inserimento , visualizzazione e ricerca.
Per quanto riguarda le operazioni di inserimento e visualizzazione si faccia riferimento agli esempi di codice precedenti . Si ricorda che le due operazioni sono molti simili , differiscono per una sola linea di codice , e che utilizzano dei cicli for nidificati , tanti quanti la dimensione della struttura .

L'operazione di ricerca consiste nel valutare la presenza di un valore all'interno del vettore o della matrice . Ad esempio valutare la presenza del numero 3 all'interno di un vettore di interi .

Caso1 Il vettore non è ordinato e si desidera arrestare la ricerca al primo elemento trovato. La ricerca torna la posizione (indice) in cui si è trovato l'elemento , -1 se la ricerca non ha avuto successo .
Caso2 come il caso precedente ma si vogliono tornare tutte le posizioni e non arrestare la ricerca al primo elemento . Si deve utilizzare un vettore in cui inserire tutti gli indici ttrovati , quindi si torna il vettore degli indici .
Caso3 Il vettore è ordinato e si desidera arrestare la ricerca al primo elemento . Si utilizza l'algoritmo di Ricerca Binaria o Dicotomica.
Ricerca Binaria : Si confronta il valore centrale del vettore con quello da cercare se il confronto ha successo si termina la ricerca altrimenti se valoreRicerca > pernoCentrale si considera la metà superiore del vettore altrimenti la metà inferiore del vettore e si ripete il procedimento finchè il vettore non degenera in un solo elemento.



Analisi della complessità

La ricerca può essere sequenziale, ovvero si passano in rassegna tutti gli elementi del vettore dal primo all'ultimo . Il numero di confronti a cui siamo costretti varia , ovvero :

Caso	Numero Confronti
Ottimo (primo)	1
Medio	n/2
Pessimo(ultimo)	n

Se si opera su un vettore ordinato si può utilizzare l'algoritmo di Ricerca Binaria , suddividendo ogni volta in due il vettore si riduce la complessità , ovvero :

Caso	Numero Confronti
Ottimo (primo)	1
Medio	Tra 2 e Lg_2 n+1
Pessimo(ultima divisione)	Lg_2 n+1

Se si opera un confronto tra i due metodi nella ipotesi che n = 15 si vede che :

Caso Medio Ricerca Sequenziale → **n/2 = 7**
Caso Pessimo Ricerca Binaria → **Lg_2 n+1 = Lg_2 16 = 4**

Come si può vedere dall'esempio la possibilità di operare su un vettore ordinato riduce di molto la complessità di calcolo delle operazioni di ricerca . Quindi è il caso di vedere come si possa ordina un vettore .

Algoritmi di ORDINAMENTO

L'ordinamento di un vettore può essere crescente o decrescente. Tutti gli algoritmi di ordinamento consentono di ordinare un vettore ma con efficienze diverse .

+ EFFICIENTE → - COMPLESSO → + VELOCE

Algoritmi di orinamento

1. Per minimi o massimi successivi o per selezione
2. Per Inserimento
3. A Bolla (BubbleSort) monodirezionale o bidirezionale
4. Quick Sort (ricorsivo)

Gli algoritmi sono in ordine cresce nte di efficienza .

Per minimi o massimi successivi o per selezione : Si scorre il vettore trovando il valore minore (o maggiore) e lo si pone in prima posizione , si scorre il vettore partendo dalla seconda posizione trovando il valore minore (o maggiore) e lo si pone in seconda posizione ….. si prosegue finchè si sono ordinati tutti gli elementi del vettore , semplice ma poco efficiente.

Per Inserimento : Si opera con lo stesso criterio con cui si ordinano le carte da gioco , più efficiente ma obbliga a shiftamenti .

A Bolla (BubbleSort) : La bolla è una coppia di valori contigui (vicini) , si considera la bolla composta dal primo e secondo elemento invertendoli se non sono ordinati , quindi si considera la bolla tra il secondo ed il terzo elemento invertendoli se non sono ordinati , si opera in questo modo finchè si consira la bolla composta dal penultimo ed ultimo elemento . Se nel corso dell'ultimo scorrimento dell'intero vettore non si sono verificati scambi tra gli elementi delle bolle il vettore è ordinato altrimenti si opera un ulteriore scorrimento . Obbliga a molti scambi ma è efficiente.

Quick Sort (ricorsivo) : Fondamentalmente si opera una suddivisione in due sottovettori e si itera il procedimento di suddivisione finchè non si formano vettori di un solo elemento . Molto veloce ed efficiente ma complesso da impementare in termini di codice .