



OBIETTIVI:

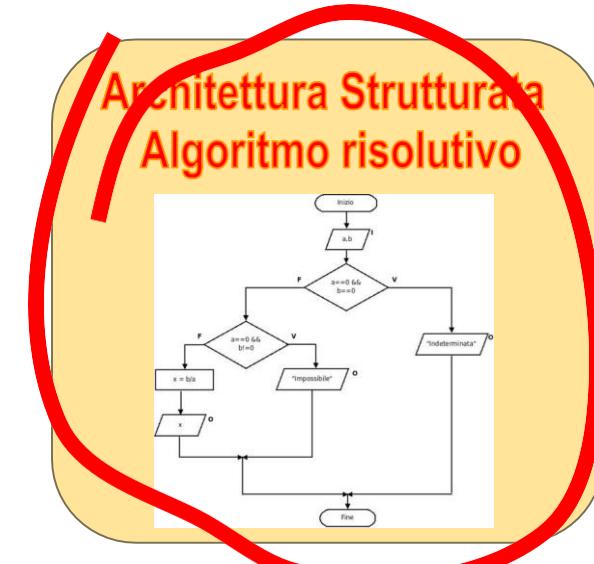
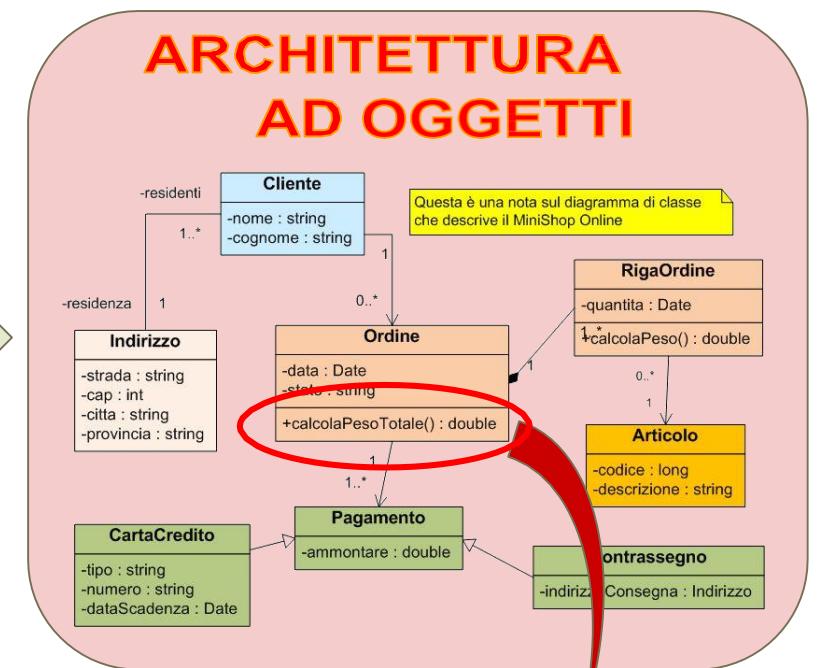
1. Sviluppare la capacità di definire l'algoritmo risolutivo per un problema elaborativo riconducibile ad un comportamento di una Classe (metodo);
2. Formalizzazione dell'algoritmo mediante la notazione grafica standard denominata FLOW CHART, applicando I principi fondamentali della programmazione strutturata (modularità, black box, ecc);
3. Traduzione del Flow Chart in Pseudocodifica.

•CU2 : “Logica Algoritmica”

• Progettazione di una applicazione

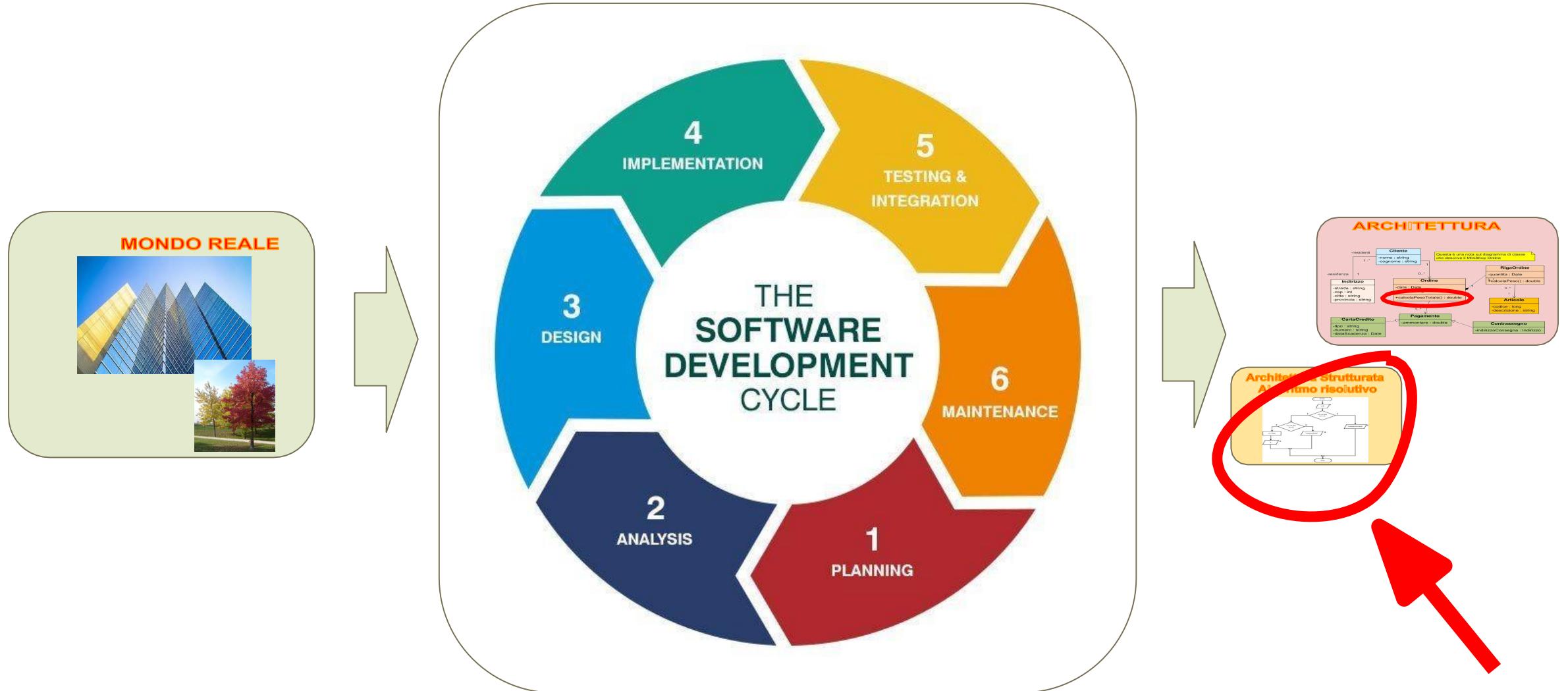


Analisi e Progettazione del MacroProblema (Scomposizione in CLASSI)



Analisi e Progettazione di un MicroProblema

• Ciclo di vita del software



• Differenza tra programmazione strutturata e ad oggetti

In definitiva la scelta è tra due soli paradigmi:

- la programmazione strutturata**
(storicamente più vecchio e più adatto a problemi semplici)
- la programmazione object oriented**
(più recente ed adatto alla progettazione di sistemi complessi che risolvono problemi altrettanto complessi)



Andiamo quindi a trattare i seguenti argomenti:

- algoritmo
- strutture dati
- strutture di controllo del flusso



nella programmazione strutturata la soluzione ad un problema è l'**algoritmo**.



Un **algoritmo** fa uso di due classi di strumenti:

- le **strutture dati**
- le **strutture di controllo del flusso**



• Logica Algoritmica: il problema

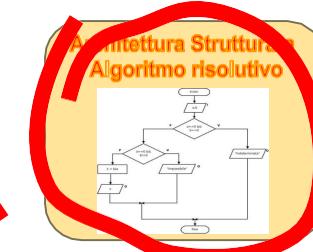
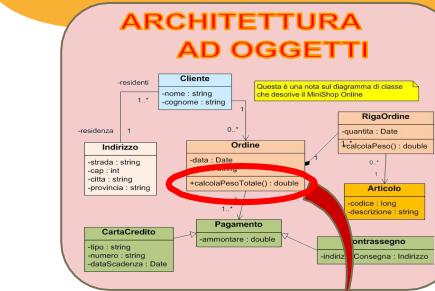
- Con la parola **problema** si intende un compito che si vuole far risolvere automaticamente a un calcolatore
- per risolvere un problema bisogna svolgere le seguenti attivita:
 - comprendere il problema
 - definire un procedimento risolutivo (algoritmo) per il problema
 - implementare l'algoritmo in un linguaggio di Programmazione
- Dato un problema relativo all'elaborazione di un'informazione bisogna individuare un opportuno metodo di elaborazione che trasformi i dati iniziali nei corrispondenti risultati desiderati.

Calcolatore = Esecutore di Azioni Elementari

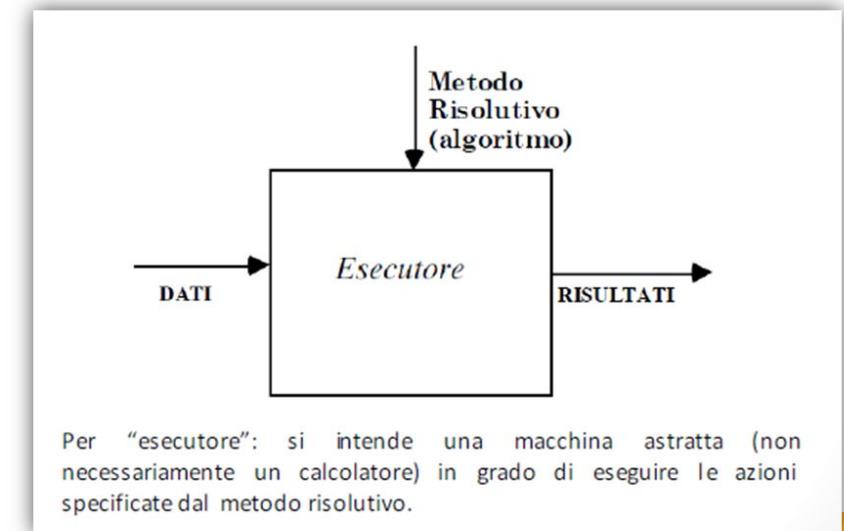
- Affinché la risoluzione di un problema possa essere realizzata attraverso l'uso del calcolatore, tale processo deve poter essere definito come una **sequenza di azioni elementari**.

ESEMPI:

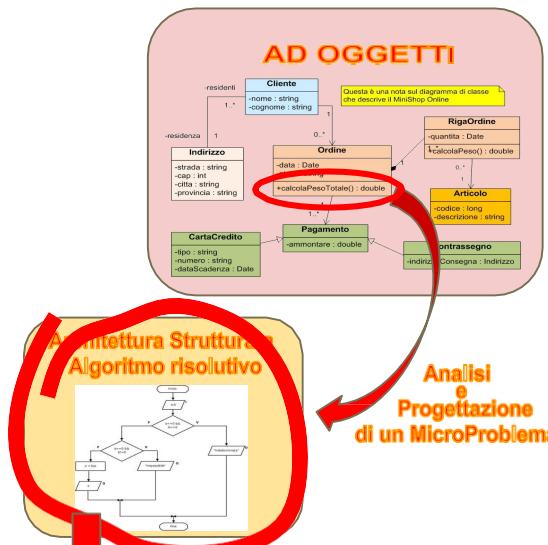
1. Una ricetta di cucina
2. Istruzioni di funzionamento di un elettrodomestico
3. Istruzioni per installare un programma



Analisi
e
Progettazione
di un MicroProblema



• Logica Algoritmica: definizione e proprietà



Analisi e Progettazione di un MicroProblema

Un algoritmo è una sequenza ordinata e finita di passi elementari che risolvono una classe di problemi.

ALGORITMO

Un **passo elementare** non è ulteriormente scomponibile: è un comando chiaro e inequivocabile.

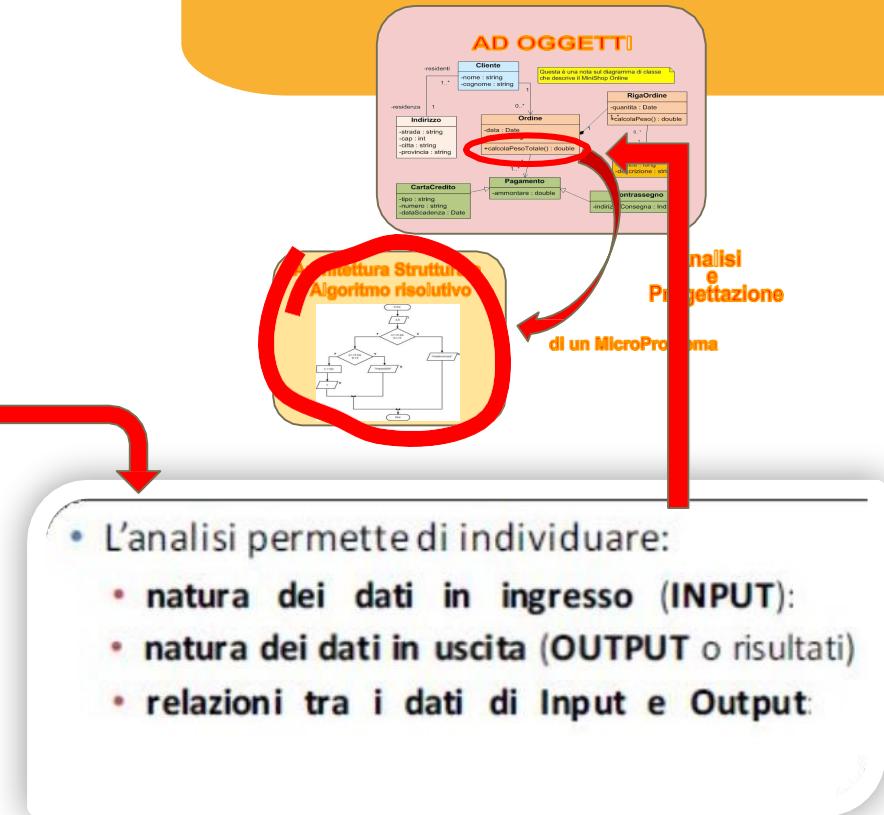
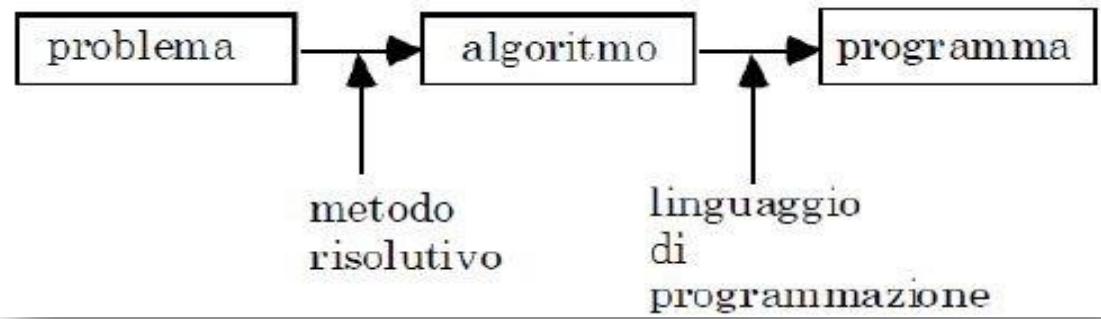
1. passo elementare
2. passo elementare
3. passo elementare
4. passo elementare
5. passo elementare
6. passo elementare
7. passo elementare
8. passo elementare
9. passo elementare
10. passo elementare
11. passo elementare
12. passo elementare
13. passo elementare
14. passo elementare

Un algoritmo raggiunge il suo obiettivo raccogliendo, manipolando e presentando delle **informazioni**.

1. **realizzabilità**: ogni operazione prevista dall'algoritmo deve essere eseguibile con le risorse a disposizione;
2. **non ambiguità**: ogni azione deve essere univocamente interpretabile dall'esecutore, cioè descritta in modo preciso; il risultato non deve cambiare al variare dell'esecutore dell'algoritmo;
3. **finitezza**: il numero totale di azioni da eseguire è finito e le operazioni da esse specificate devono essere eseguito un numero finito di volte.

• Logica Algoritmica: dal problema al programma

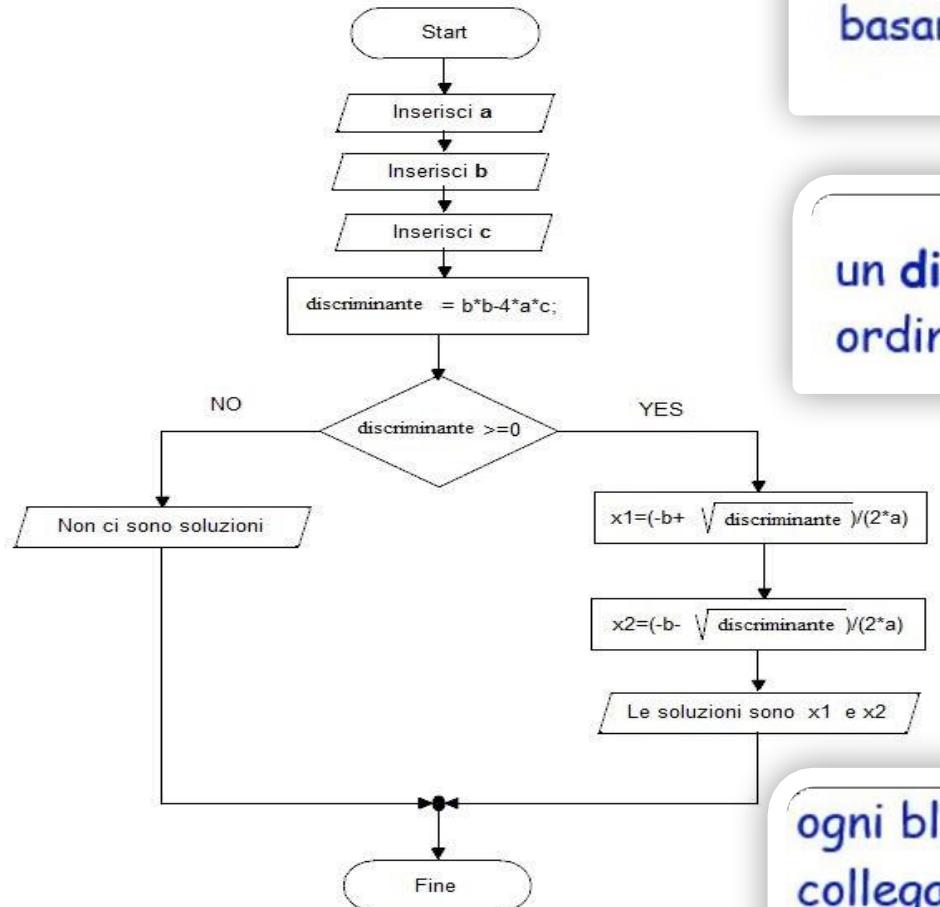
- Dato un problema, la sua soluzione può essere ottenuta mediante l'uso del calcolatore, compiendo i seguenti passi:
 - individuazione di un **metodo risolutivo**
 - scomposizione del procedimento in un insieme ordinato di azioni → **ALGORITMO**
 - rappresentazione dei dati e dell'algoritmo attraverso un formalismo comprensibile dall'elaboratore:
linguaggio di programmazione → **PROGRAMMA**



Classe FIGURA GEOMETRICA
Metodo CALCOLA AREA
Input del metodo: Base, Altezza
Output del metodo: Area
Relazione tra Input e Output: Formula Base X Altezza

• Logica Algoritmica: il Flow Chart

Flow-chart del programma
Risoluzione equazione di 2^a grado nella forma



I 'diagrammi di flusso' permettono di descrivere gli algoritmi basandosi su simboli grafici contenenti le operazioni da eseguire.

un **diagramma di flusso** descrive le azioni da eseguire ed il loro ordine di esecuzione.

ogni **azione** corrisponde un simbolo grafico (**blocco**) diverso.

ogni blocco ha un ramo in ingresso ed uno o piu` rami in uscita; collegando tra loro i vari blocchi attraverso i rami, si ottiene un diagramma di flusso

• Logica Algoritmica: altri blocchi grafici del Flow Chart



Blocco di elaborazione generica



Blocco di selezione



Blocco di ingresso / uscita dei dati



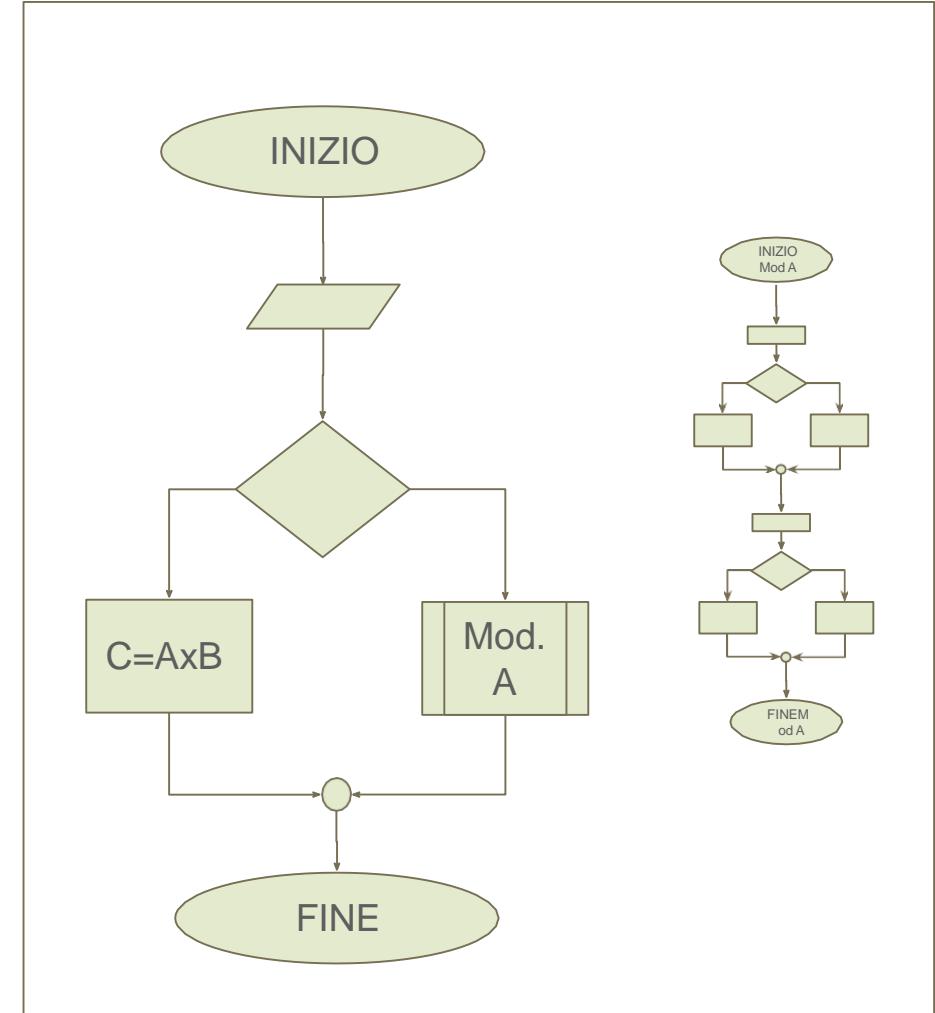
INIZIO e FINE



Sottoprogramma



Blocco connettore



• Logica Algoritmica: un esempio

Problema: forniti in ingresso al programma la ~~b~~ e l'altezza di un triangolo, calcolarne l'area

Analisi:



Dati in input: la lunghezza della base e la lunghezza dell'altezza del triangolo; il vincolo di integrità da porre è accettare in ingresso solo valori > 0

Dati in output: l'area del triangolo

Relazione tra I/O: l'area del triangolo si ottiene moltiplicando la base per l'altezza e dividendo per 2 $A = (b * h) / 2$

Algoritmo



1. Inizio
2. Ricevi in ingresso da tastiera la lunghezza della base del triangolo (con il vincolo che sia > 0)
3. Ricevi in ingresso da tastiera la lunghezza dell'altezza del triangolo (con vincolo che sia > 0)
4. Calcolare l'area del triangolo moltiplicando la lunghezza della base per la lunghezza dell'altezza e dividendo il prodotto ottenuto per 2
5. Stampa a video l'area del triangolo calcolata
6. Fine

• Logica Algoritmica: classi di strumenti dell'algoritmo

Strutture Dati

Il **dato** viene tenuto in una **struttura dati** della quale conosciamo il **tipo**, il **nome** ed il suo **indirizzo** in memoria centrale o di memoria di massa

DATO

- variabili
- costanti

STRUTTURE DATI SEMPLICI

- vettori
- record

STRUTTURE DATI COMPLESSE

- file

S.D. SU MEMORIA DI MASSA

- puntatori

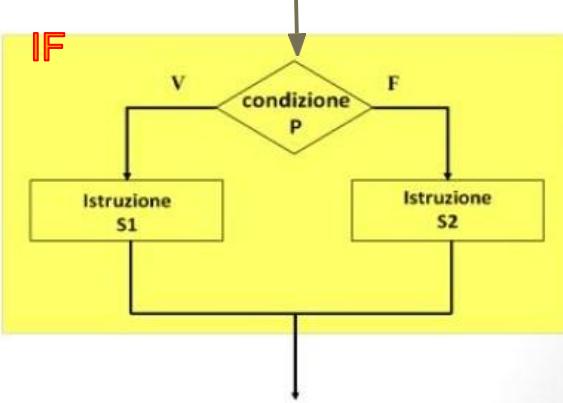
STRUTTURE DATI DINAMICHE

Strutture di controllo del Flusso

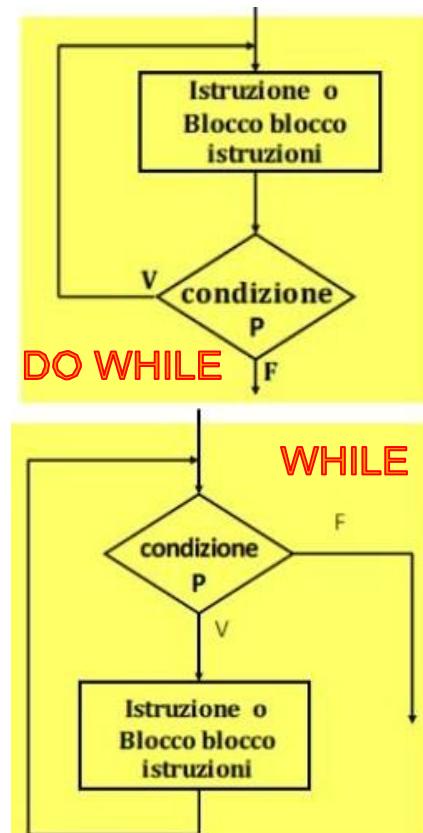
SEQUENZA



SCELTA



ITERAZIONE

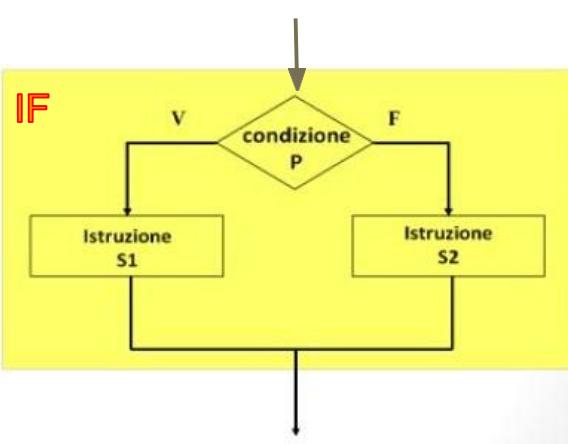


- Ogni problema è riconducibile alle 3 strutture fondamentali di flusso (Teorema di Bohm-Jacopini)
 - Ogni struttura deve avere 1 entrata 1 uscita
- ATTENZIONE**
- ricorda di incrementare l'indice nelle **ITERAZIONI**
 - ricorda di inizializzare le variabili

- Logica Algoritmica: trasposizione in pseudocodice

Notazione grafica
FlowChart

SCELTA



Sequenza di istruzioni
in pseudocodice

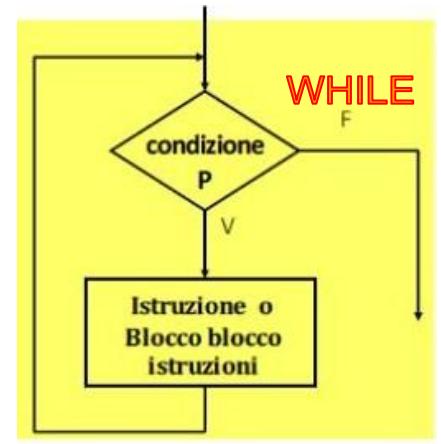
```

0001 - IF punteggio >= 90 THEN
0002 -     Stampa(A)
0003 - ELSE
0004 -     Stampa(B)
  
```

ITERAZIONE



ITERAZIONE



```

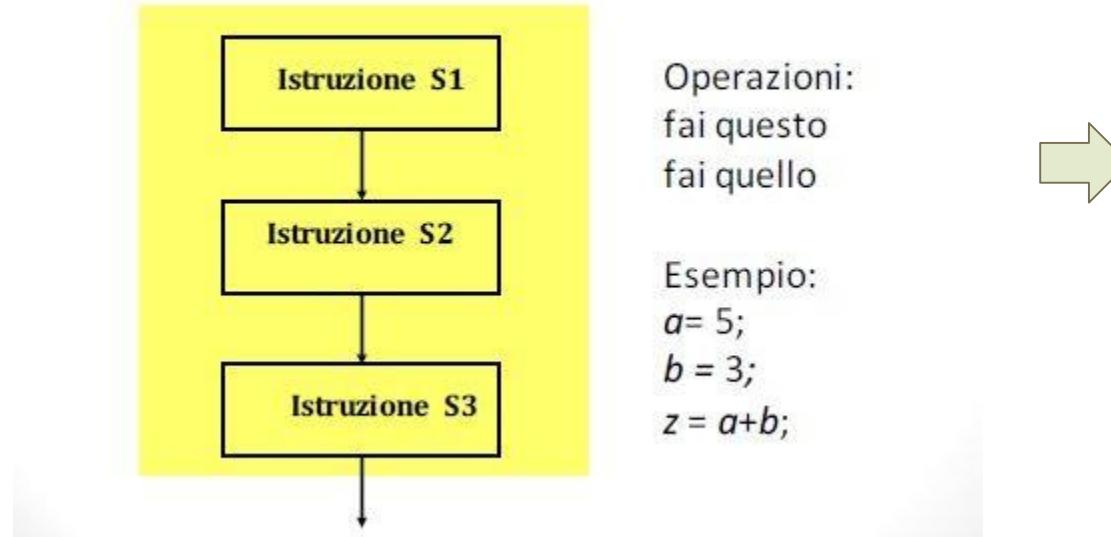
0001 - DO {
0002 -     Stampa(A)
0003 -     i++
0004 - } WHILE (i < 5)
0005 - Stampa(B)
  
```

```

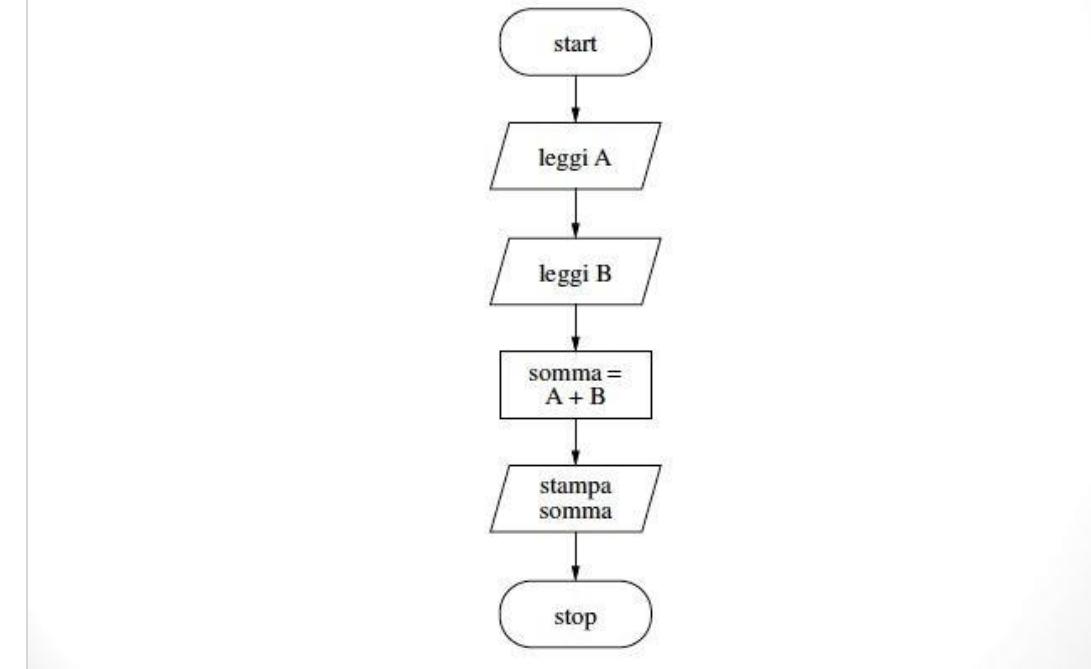
0001 - WHILE (i < 5) {
0002 -     Stampa(A)
0003 -     i++
0004 - }
0005 - Stampa(B)
  
```

• Logica Algoritmica: Sequenza

Le istruzioni sono una di seguito alla altra



Esempio: somma di due numeri

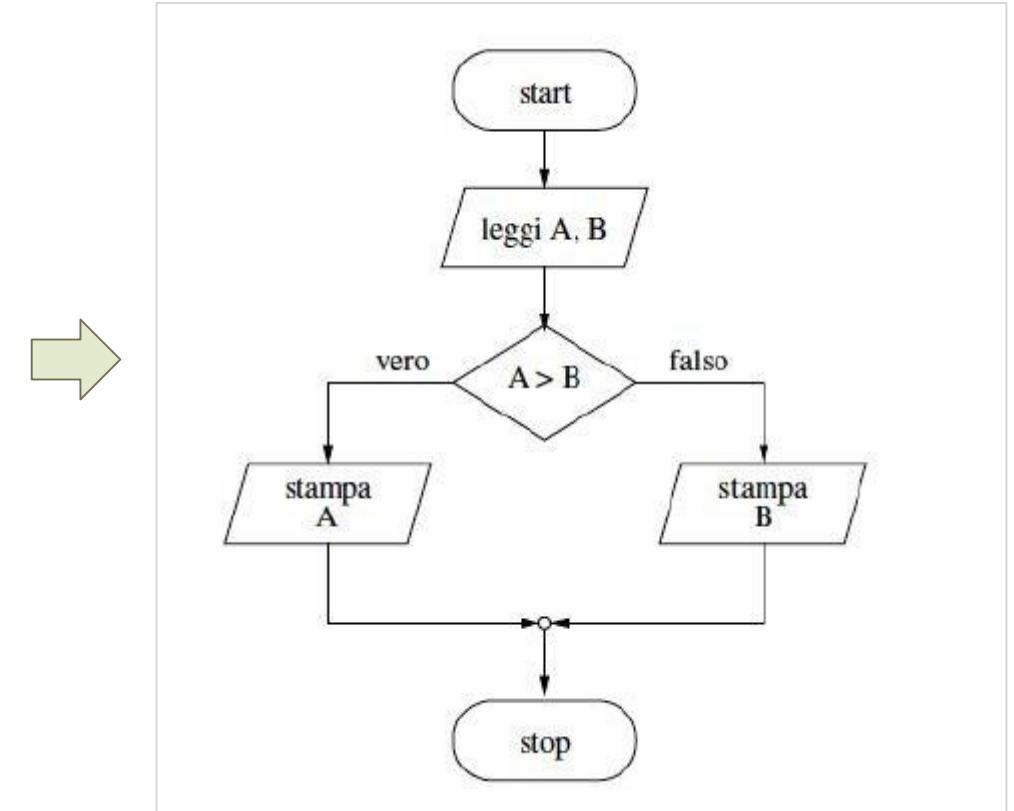
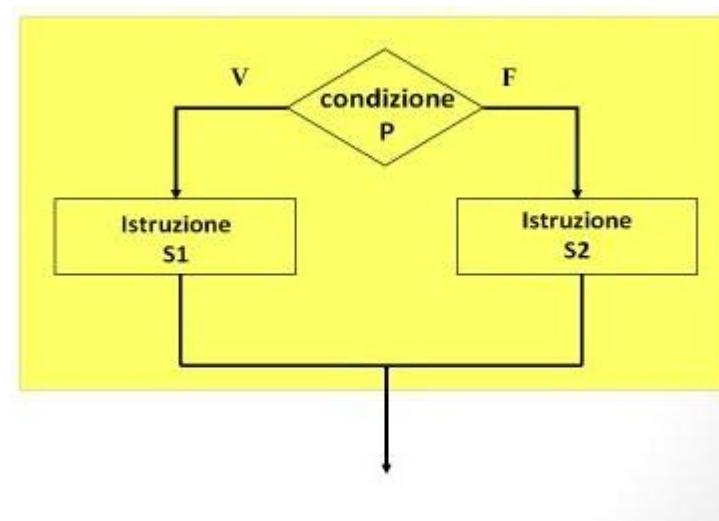


• Logica Algoritmica: Selezione o Scelta

**Se si verifica la Condizione P verrà eseguita l'istruzione S1
altrimenti verrà eseguita l'istruzione S2**

Di seguito, in ogni caso, verranno eseguiti i blocchi successivi del Flow Chart

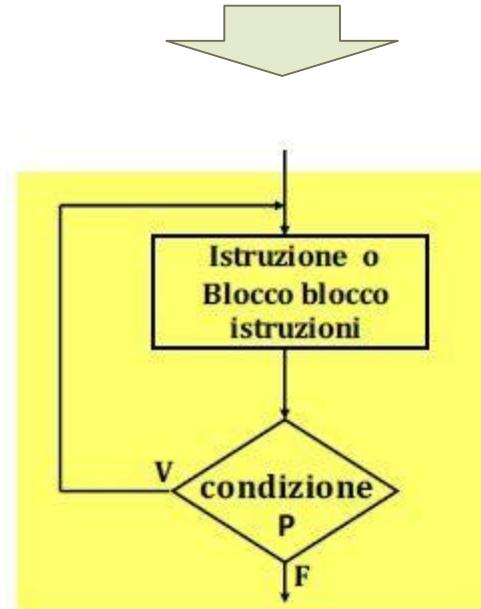
La condizione P verrà indicata tramite una espressione complessa e segue i criteri dell'Algebra di Boole



• Logica Algoritmica: Iterazione

Nel DO WHILE

- verrà eseguito almeno una volta il blocco di istruzioni.
- Successivamente verrà valutata, secondo la logica di Boole, se la condizione sia Vera o Falsa.
- Nel caso in cui la condizione dovesse risultare Vera, il ciclo verrà ripetuto

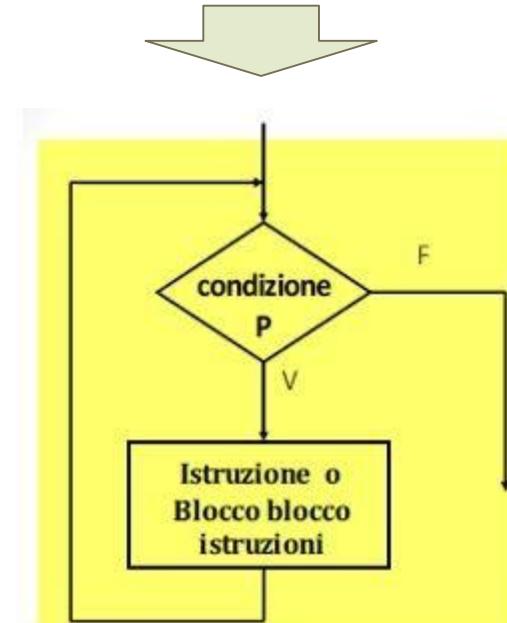


Nel WHILE

- verrà innanzitutto valutata, secondo la logica di Boole, se la condizione sia Vera o Falsa.
- Il blocco di istruzioni verrà eseguito solo se la condizione sarà Vera e ripetuto nuovamente il controllo della condizione,
- Il Ciclo si interromperà non appena la condizione diventerà falsa.

ATTENZIONE

Qualora la condizione dovesse essere falsa al primo ciclo, il blocco di istruzioni non verrà mai eseguito



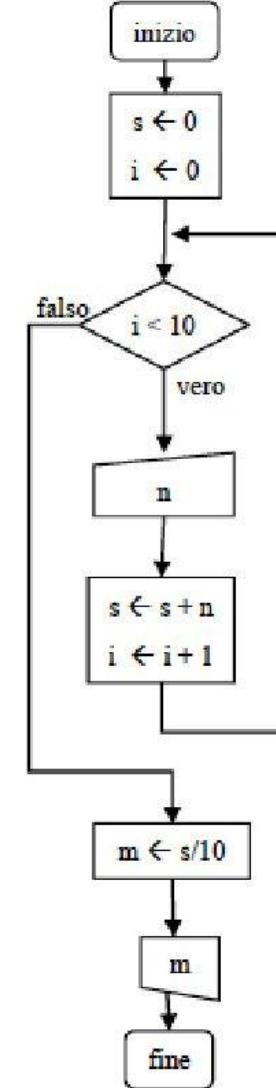
• Logica Algoritmica: un esempio

Problema:

Acquisizione di 10 numeri interi e calcolo della media

Algoritmo

1. Inizio
2. Somma=0
3. i=0
4. Leggi(n)
5. Somma=somma + n
6. i =i+1
7. Se i <10 torna a 4
8. Media=somma/10
9. Stampa(media)
10. Fine



• Logica Algoritmica: struttura Dati, Variabili e Costanti

Variabile: Rappresenta un dato ed è individuata da un nome simbolico cui è assegnato un valore che può cambiare durante l'esecuzione dell'algoritmo.

Costante: è una grandezza nota a priori, il cui valore non cambia durante l'esecuzione.



I **valori** dei dati possono essere:

- **numerici:** interi e reali (es. 10 23,4)
- **logici:** Vero e Falso
- **alfanumerici, o stringhe**
(es. "AAAAA", "C.Colombo")

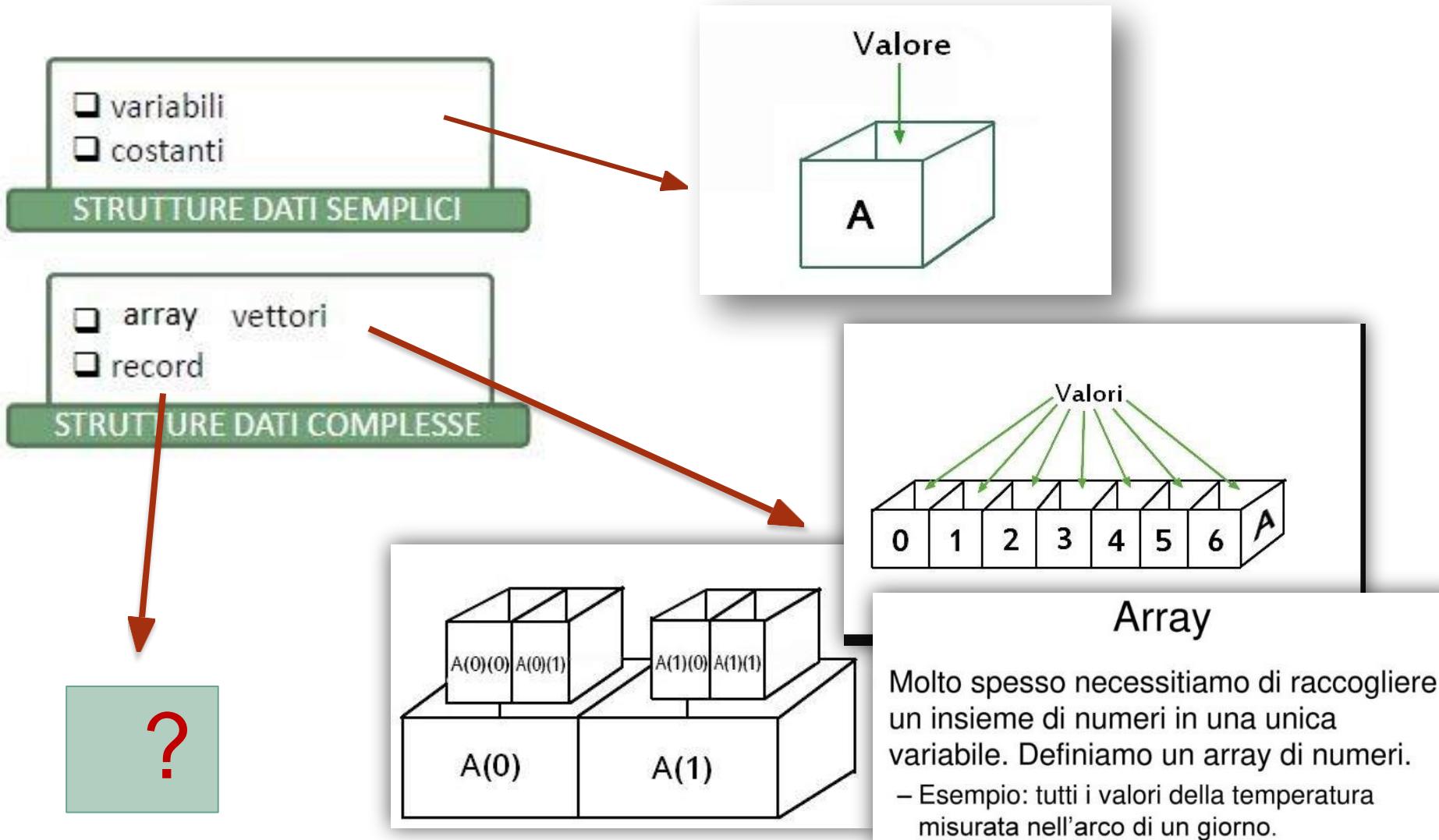
- ◆ Operazioni aritmetiche e assegnamenti di valori a singole variabili
 - Es. $C \leftarrow (A + B)$
- ◆ Condizioni sul valore di singole variabili
 - se $(A > B)$ allora ... altrimenti ...
- ◆ Lettura e scrittura di variabili
 - "Leggi A" oppure "Stampa B"

• Logica Algoritmica: esempio con variabili semplici

Esempi:

- Scambia valori delle variabili
- Stampa dei primi 100 numeri
- Stampa dei primi n numeri
- Fattoriale di n
- Massimo di tre numeri

• Logica Algoritmica: Variabili semplici e strutturate



• Logica Algoritmica: Array e Vettori

L'**ARRAY** è una struttura caratterizzata dal proprio nome, dal tipo di dato (ogni elemento è dello stesso tipo), dal numero di dimensioni (o indici), dal dominio di ogni indice. Ogni elemento dell'array viene individuato specificando il nome dell'array e, tra parentesi, le coordinate di ciascun indice. Il **VETTORE** è un array a una dimensione, la **MATRICE** è un array a due dimensioni.

VETTORE

V	
1	345
2	8200
3	55
4	33
5	120

MATRICE

M	1	2	3
1	ITA	STO	ITA
2	MAT	ING	ITA
3	INF	INF	MAT
4	RAG	INF	RAG
5	TEC	EDF	TEC

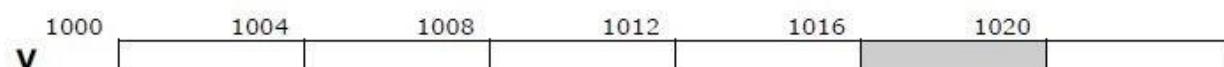
V(2) o **V[2]** individua l'elemento di indice 2 il cui valore è 8200.

M(4,2) o **M[4][2]** individua l'elemento di riga 4 e colonna 2 il cui valore è "INF".

Se l'array ha tre dimensioni si specifica prima la riga, poi la colonna ed infine la profondità.

In generale, per array a **n** dimensioni, si specifica, nell'ordine, la coordinata di ciascuna dimensione.

L'implementazione del vettore in memoria centrale consiste nel memorizzare tutti i suoi elementi in celle contigue. L'indirizzo dell'elemento **i**-esimo è: $V[i] = IB + L * (i-1)$, dove IB è l'indirizzo iniziale e L la lunghezza di ciascun elemento.

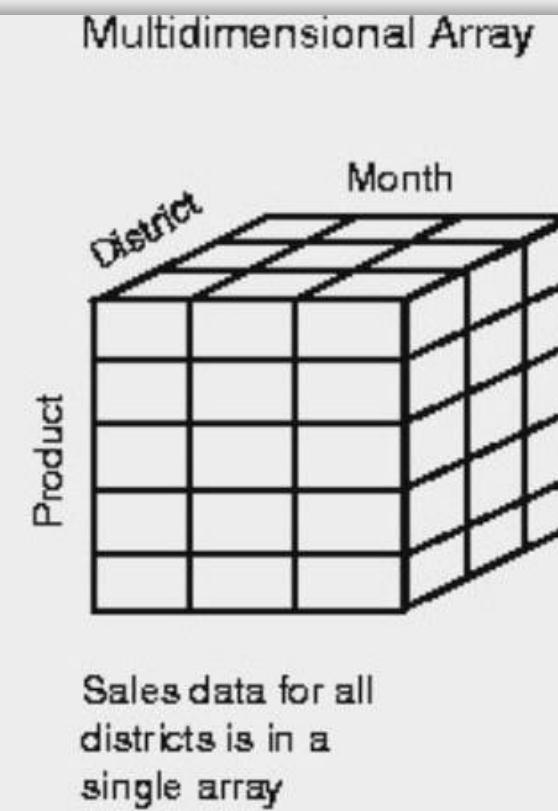
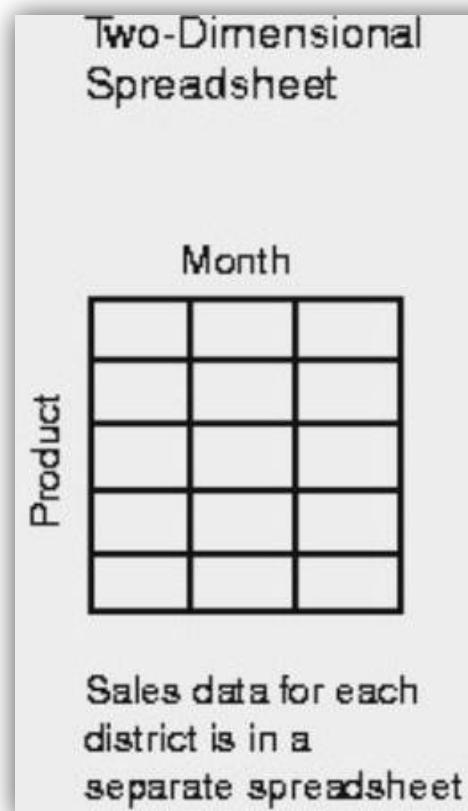


In questo esempio V è un vettore di 6 elementi floating point, ogni elemento occupa 4 byte (L=4), il suo indirizzo iniziale è 1000 (IB=1000).

L'indirizzo del 4° elemento (i=5) è: $V[5] = 1000 + 5 * (5-1) = 1000 + 16 = 1016$

• Logica Algoritmica: Vettori e Array multidimensionali

0	Lunedì
1	Martedì
2	Mercoledì
3	Giovedì
4	Venerdì
5	Sabato
6	Domenica



• Logica Algoritmica: esercizi con Array

Esempi:

- Acquisizione di 10 numeri e calcolo della media
- Acquisizione di 10 numeri, calcolo e visualizzazione del maggiore e del minore
- Acquisizione di 10 ricerca e stampa i numeri dispari
- Scambia elementi dell'array

• Logica Algoritmica: concetto di RECORD IN MEMORIA RAM

Il **RECORD** (o *registrazione*) è una struttura che serve a contenere le informazioni di un oggetto complesso, è costituito da un insieme strutturato di campi e ogni **CAMPO** contiene una informazione. Per ogni campo si stabilisce il nome che lo individua, il tipo di dato che può contenere e la lunghezza in byte (che va specificata solo per i campi alfanumerici, altrimenti è implicita e data dal tipo del campo).

Record ALUNNO

Matr	Nominativo	Nato-il	Nato-a	MF	Tassa	Indirizzo	Citta
num	alfanumerico	data	Alfanum.	alf.	numero	alfanumerico	alfanum.

Nell'esempio precedente il record **ALUNNO** è strutturato nei seguenti campi:

- Matr	numero intero	2 byte	matricola
- Nominativo	alfanumerico	30 byte	cognome e nome
- Nato-il	data	10 byte	data di nascita
- Nato-a	alfanumerico	20 byte	città di nascita
- MF	alfanumerico	1 byte	sesso (M o F)
- Tassa	numero reale	4 byte	tassa di iscrizione
- Indirizzo	alfanumerico	40 byte	indirizzo di residenza
- Citta	alfanumerico	20 byte	città di residenza

		145 byte	(lunghezza totale del record)

La sequenza dei campi che costituiscono il RECORD (record LOGICO) e le loro caratteristiche (nome, tipo, lunghezza) costituiscono il **tracciato record**.

Un campo del record può a sua volta essere un record, un array, un tipo elementare qualsiasi.

I campi del record sono identificati da *nome_record.nome_campo* (esempio: *Alunno.Matr*)

Tutti i linguaggi di programmazione forniscono la variabile strutturata record.

Variabili Strutturate in memoria RAM:

Pippo,
campo1 char(1),
campo2 char(1),
campo3 int

• Logica Algoritmica: Tabelle in memoria RAM e File

ATTENZIONE

**una tabella in RAM è
idonea ad ospitare SOLO
UNA PARTE dei record di
file molto estesi, essendo
la RAM di dimensioni finite**

La **TABELLA** è una struttura costituita da record che può essere definita **array di record** o, più precisamente, **vettore di record**.

La tabella sarà costituita da tante righe quanti sono i record e da tante colonne quanti sono i campi dei record che la costituiscono.

Tabella CLASSE

Matr	Nominativo	Nato-il	Nato-a	MF	Tassa	Indirizzo	Citta
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

Nell'esempio precedente la tabella CLASSE è costituita da 10 righe (10 record ALUNNO) e da 8 colonne (gli 8 campi del record: Matr, Nominativo, Nato-il, Nato-a, MF, Tassa, Indirizzo, Citta).

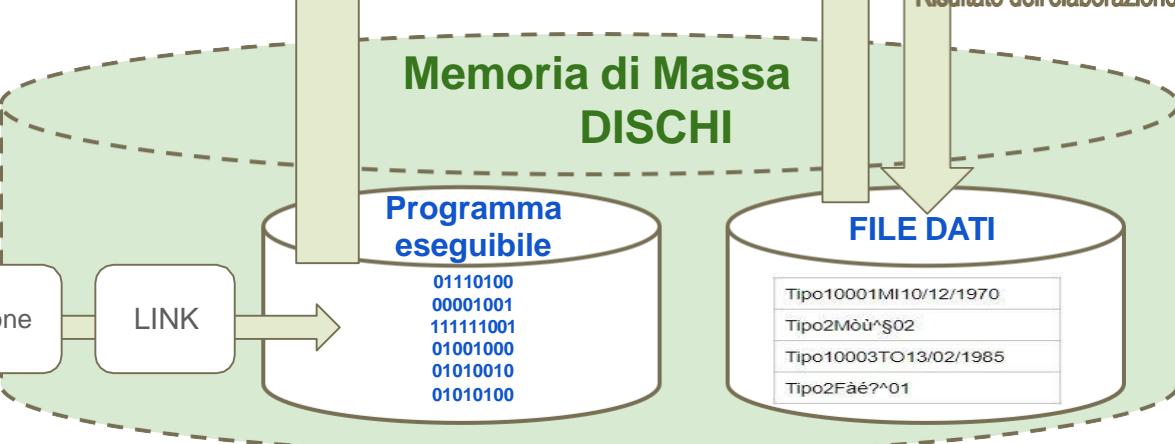
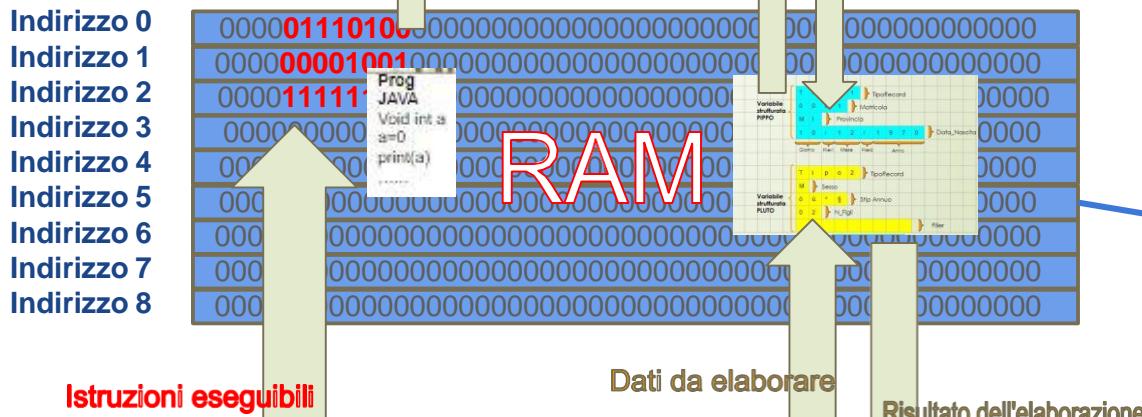
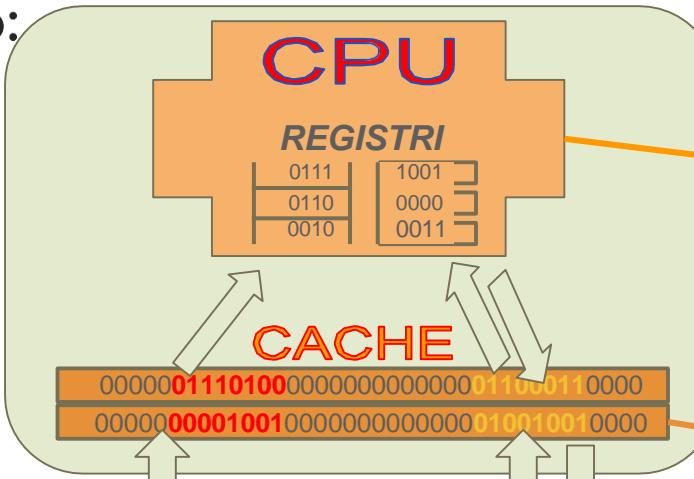
I record sono individuati dalla loro posizione e i campi dal loro nome. Ad esempio il campo **Nato-a** del record numero 5 si identifica con: **CLASSE(5).Nato-a**.

Questo per quanto riguarda le tabelle in memoria centrale.

Se la tabella è memorizzata su un file in memoria di massa, allora è un **Archivio di Dati**, se è memorizzato su un database, allora è una **Tabella di DataBase**. In questo caso per leggere e scrivere i record della tabella occorre utilizzare le istruzioni che regolano l'accesso agli archivi o ai database.

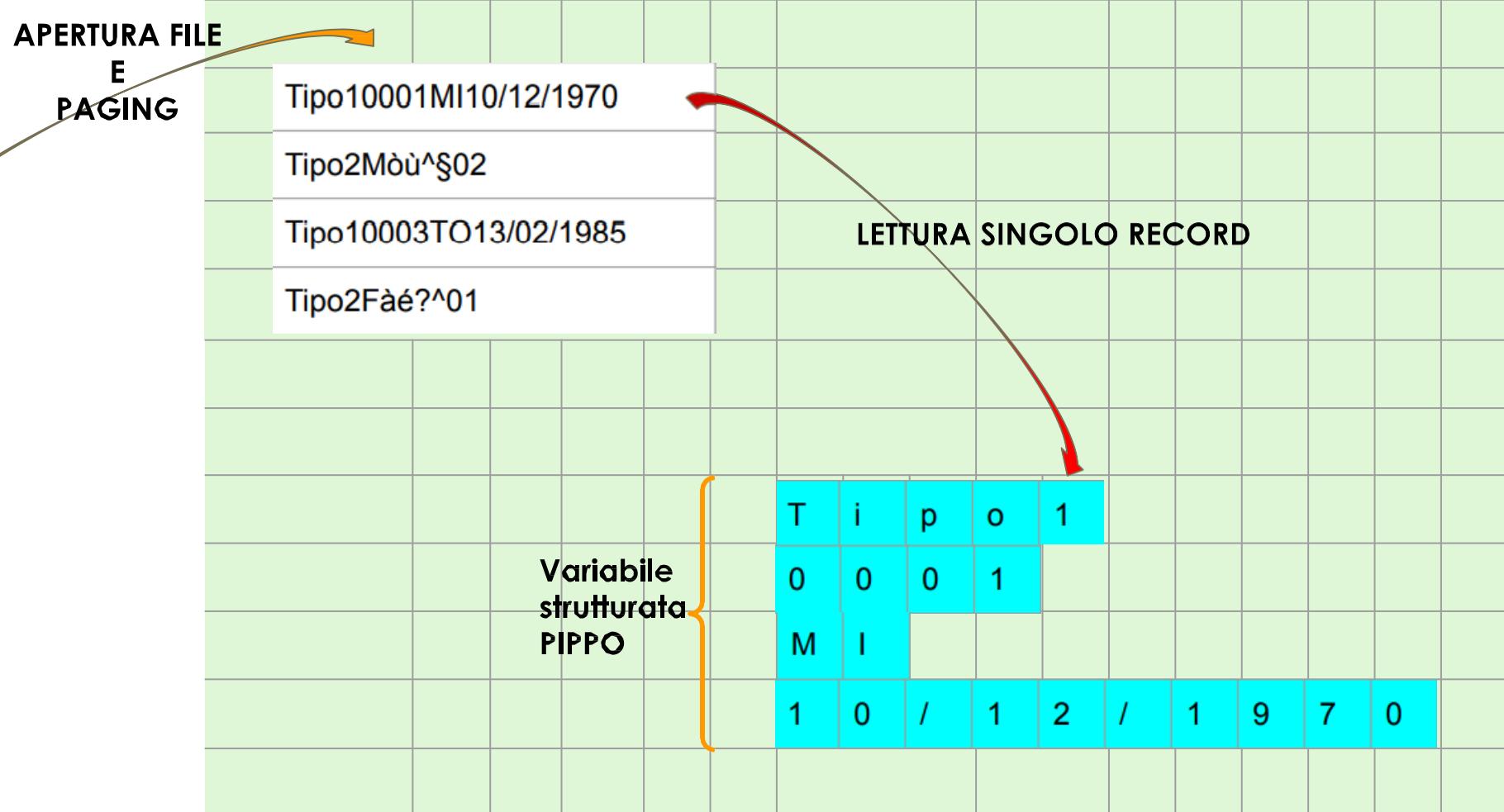
Il calcolatore elettronico:

Flusso Elaborativo



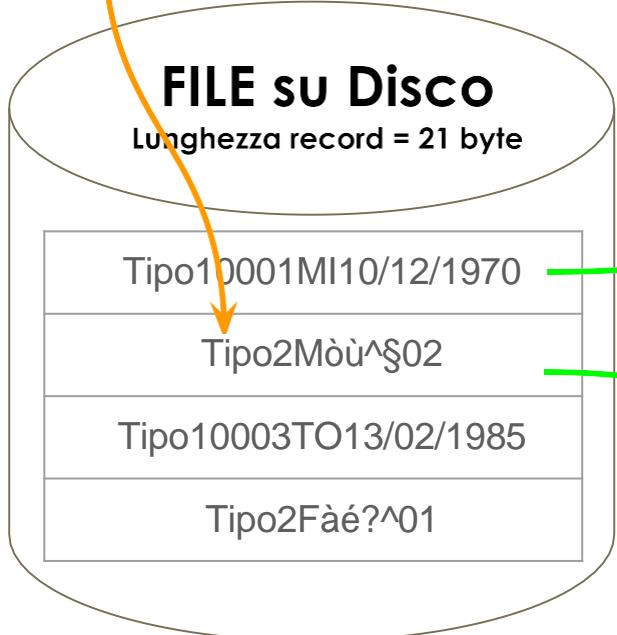
- Logica Algoritmica: lettura Record da FILE

RAM



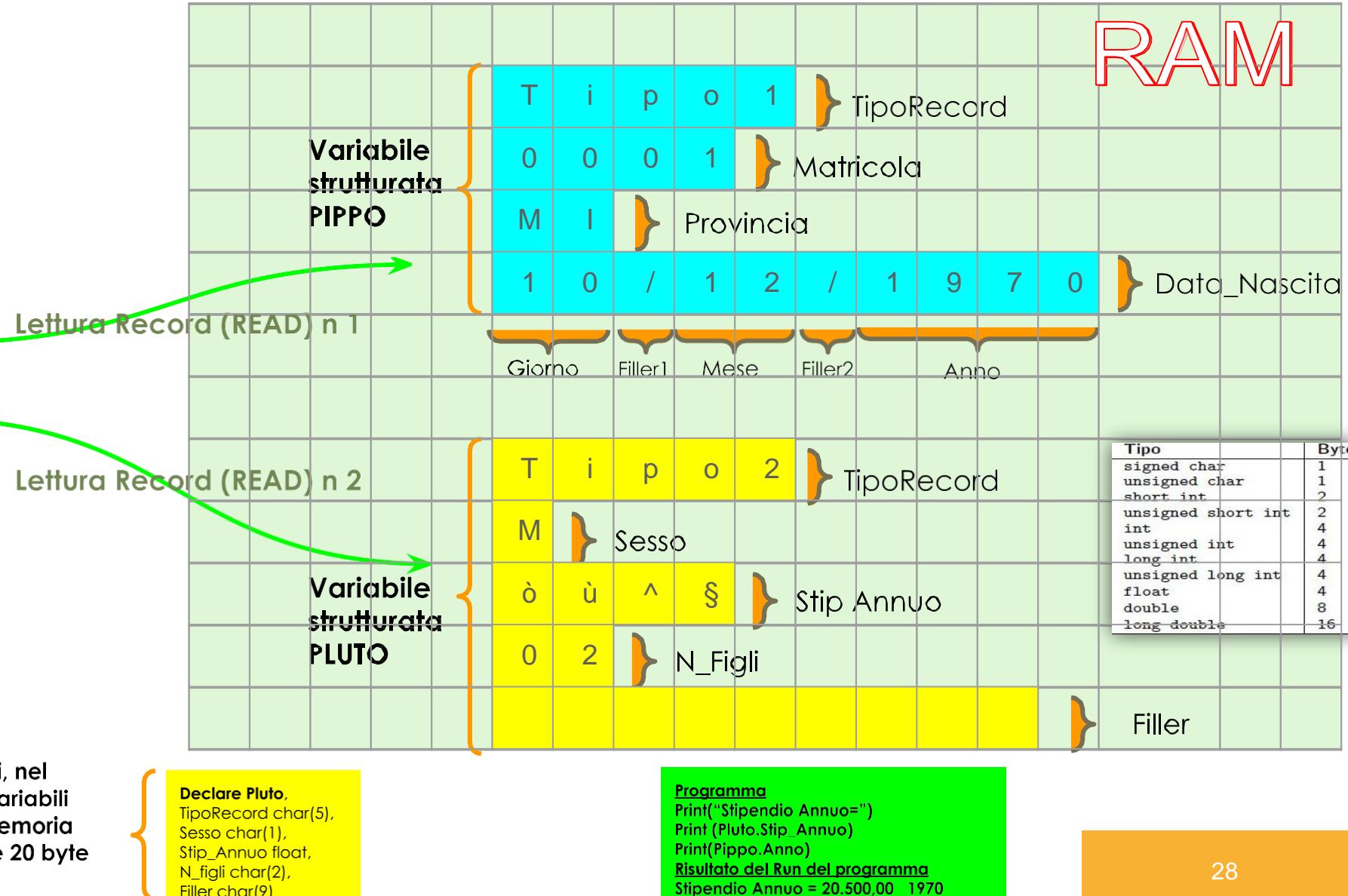
• Logica Algoritmica: lettura Record da FILE

Stipendio annuo = 20.500,00 (float)



```
Declare Pippo,
TipoRecord char(5),
Matricola char(4),
Provincia char(2),
Data_nascita,
Giorno Char(2),
Filler1 char(1),
Mese char(2),
Filler2 char(1),
Anno char(4)
```

dichiarazioni, nel
programma, variabili
strutturate in memoria
Lunghezza totale 20 byte



• Logica Algoritmica: lettura Record da FILE

Posso leggere il File ANAG
nella seguente variabile
strutturata ?

Rec_Anag1
Nominativo char(20)
Età int

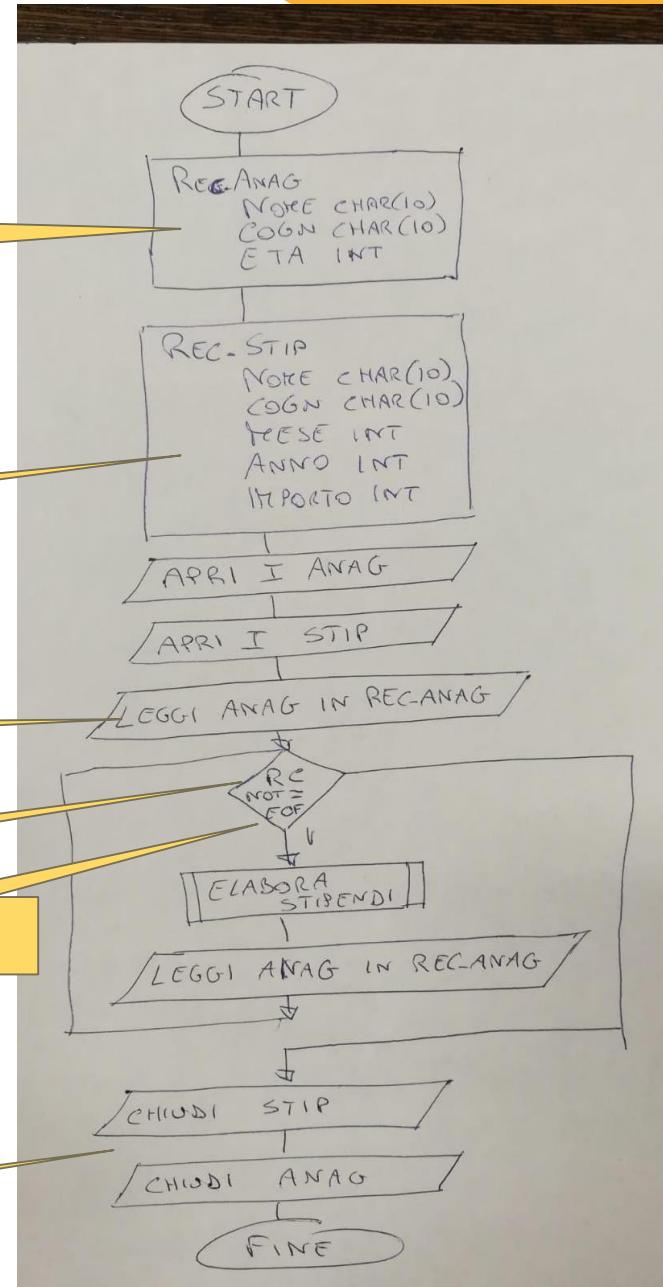
Rec_Stip occupa la stessa area di memoria di Rec_Anag ?

Quanti byte occupa un record del file ANAG ?

Cos'è RC e a cosa serve ?

Cos'è EOF ?

Posso omettere le chiusure
dei Files ?

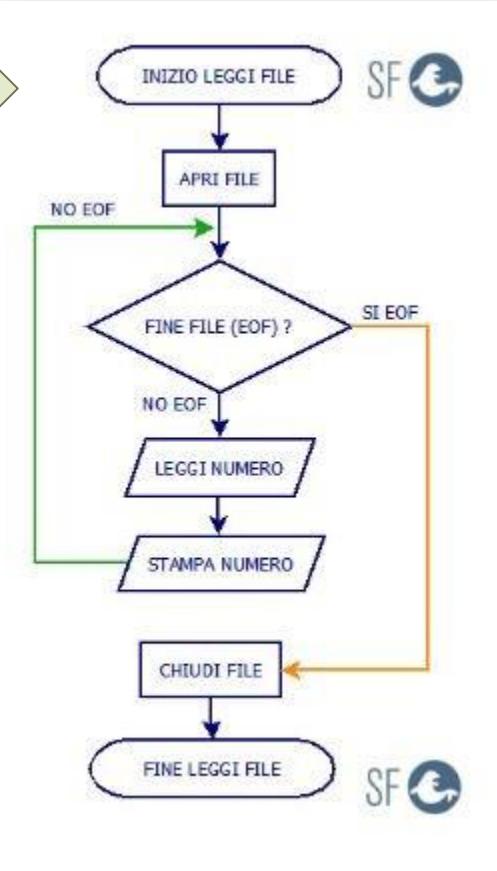


• Logica Algoritmica: principali metodi di accesso dei FILE

- **sequenziale**
- **diretto**
- **con chiave**

La ricerca dei record avviene sempre a partire dall'inizio del file, leggendo tutti i record uno dopo l'altro, fino a quando non si arriva a quello desiderato

Esempio: supponiamo di voler leggere i dati sul cliente "Bianchi". Dobbiamo leggere tutte le informazioni fino ad arrivare al cognome cercato

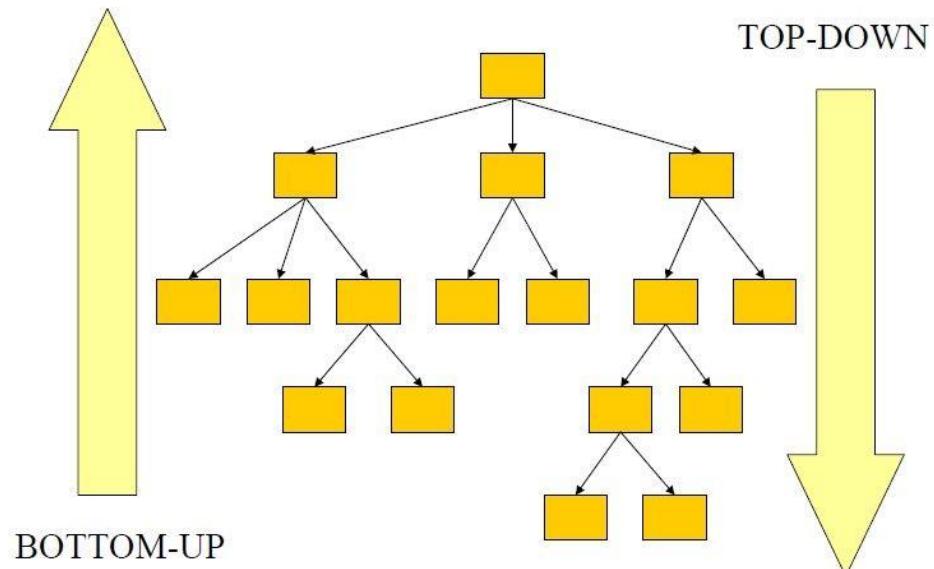


ATTENZIONE

- Solo alla chiusura del fine verranno effettivamente scaricate i buffer dati con le variazioni apportate dal programma tramite le WRITE
- Alla riapertura del file la lettura tramite READ ricomincerà dal primo record

• Modularità : principi

- si parla in questo caso di '**programmazione modulare**' cioè del fatto che si possono scrivere sottoprogrammi (o '**moduli**') in modo che possano essere riutilizzati per programmi diversi e da persone diverse



TOP-DOWN:

su una scomposizione del problema in sottoproblemi che a loro volta possono essere ulteriormente scomposti fino ad arrivare a sottoproblemi molto semplici costituiti solo da operazioni molto elementari, direttamente «eseguibili» da parte dell'esecutore

BOTTOM-UP :vengono prima definiti i moduli di più basso livello vengono poi combinati per ottenere i moduli di livello superiore così di seguito fino ad arrivare al nodo radice, cioè al modulo che definisce il sistema nel suo complesso

• Modularità : caratteristiche

Caratteristiche di un MODULO

- parte **INDIPENDENTE** dal resto del programma e svolge un compito ben preciso
- sviluppabile **SEPARATAMENTE**
- con relazioni di interazione con gli altri moduli

Parametri qualitativi di un MODULO

- Dimensione
- Coesione
- Accoppiamento

Una buona modularizzazione:

- *ogni modulo deve avere la massima coesione ed il minimo accoppiamento*

I principi da adottare per la modularità

- Alta coesione: omogeneità interna del modulo
- Basso accoppiamento: indipendenza del modulo da altri
- Interfacciamento esplicito: modalità di utilizzo chiare
- Information hiding: dettagli nascosti nel modulo

• Modularità : vantaggi della modularizzazione

Esistenza di schemi (disegni) dell'applicazione a diversi gradi di astrazione (metodologia Top-Down)

Riusabilità dei moduli (black box - una entrata, una uscita)

Facilità di manutenzione e parallelizzazione manutentiva dei moduli software

Maggiore leggibilità, orientamento, compattezza e facilità di manipolazione degli schemi, algoritmi, disegni, ecc

- **Testing e debug separato dei moduli**
- **Test e debug integrato dei moduli**
- **Problem Solving più rapido in caso di errore, sia in fase di test che sul sistema di produzione (per es. errore sul modulo del calcolo del saldo)**

• Modularità : sottoprogrammi: Funzioni, Routine, Procedure. Ricorsione, black box (concetti)

Tipologie di moduli

- Sottoprogrammi
 - Funzioni (Es. calcolo dell'Area di una figura - Calcola e restituisce 1 valore)
 - Procedure (Es. calcolo del cedolino paga - calcola e memorizza i dettagli del cedolino sul disco oppure li stampa))

□ Un sottoprogramma è:

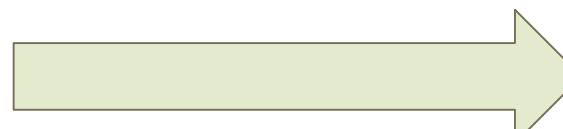
- ▶ un insieme di istruzioni dotato di nome
- ▶ descritto (definito) una sola volta
- ▶ attivabile (richiamabile o invocabile) all'interno del programma o di un altro sottoprogramma

□ Alcuni sottoprogrammi sono già definiti

- ▶ si pensi alla `scanf` e alla `printf`
- ▶ dietro a questi nomi vi sono una serie di istruzioni in grado di, rispettivamente, intercettare la pressione dei tasti e di visualizzare un carattere sullo schermo
- ▶ chi richiama queste funzioni non si preoccupa di come sono fatte, basta sapere solo cosa fanno (visione black box)

Caratteristiche che differenziano l'uso dei sottoprogrammi

- dichiarazioni
- passaggi di parametri
- valore in uscita
- modalità di chiamata
- Ricorsione



```
// function Fcalcolaarea()
int square( int n ) {
    return n * n;
}
ris = Fcalcolaarea(base, altezza)

// procedure PcalcolaCedolino
void display( int n ) {
    printf( "The value is %d", n );
}
call PcalcolaCedolino
```

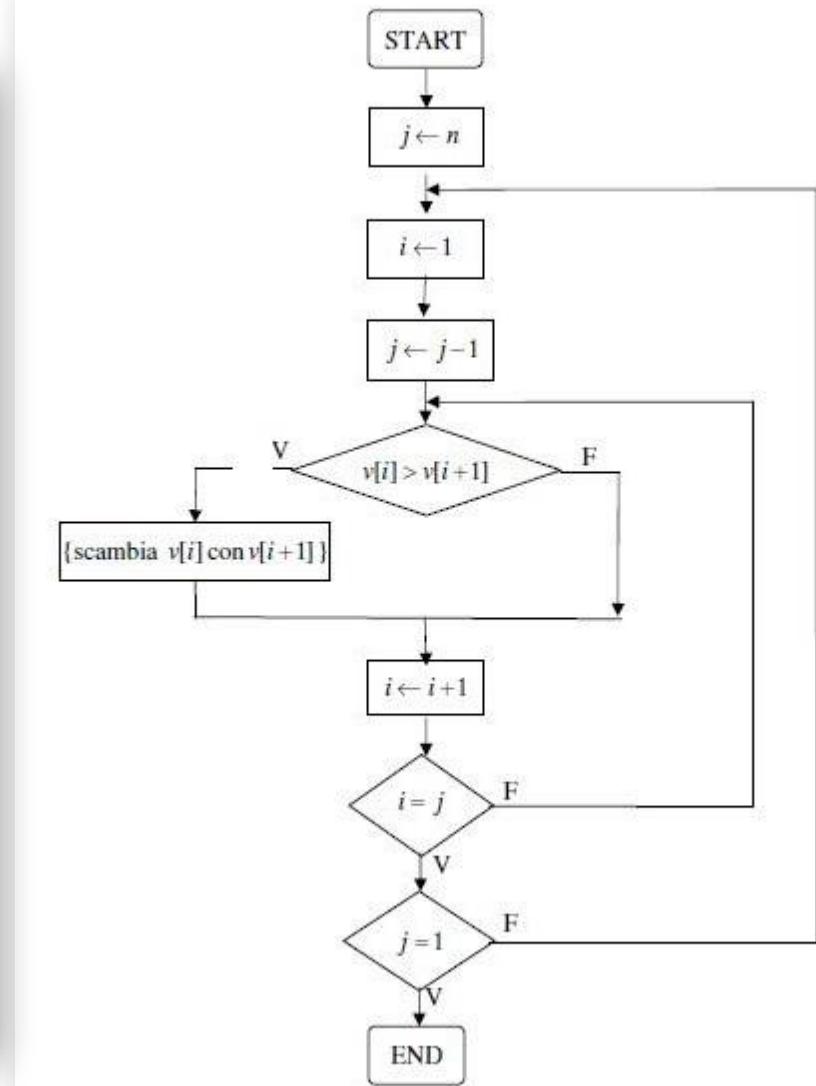
• Logica Algoritmica: Bubble Sort

Il Bubblesort agisce considerando coppie adiacenti di elementi, e scambiandole se non rispettano la relazione d'ordine desiderata, in pratica si basa sul principio che in un vettore ordinato (in modo crescente), presi comunque due elementi adiacenti, il primo è minore o uguale al secondo.

Se in una scansione del vettore non si effettuano scambi, significa che il vettore è ordinato.

Si chiama “ordinamento a bolla” perché dopo la prima scansione del vettore, l'elemento massimo si porta in ultima posizione (gli elementi più piccoli salgono come “bolle” verso le posizioni iniziali del vettore).

Si inizia confrontando i primi due elementi: se il primo è superiore al secondo, allora i due valori vengono scambiati, altrimenti si passa ad esaminare il secondo e il terzo elemento e così via... fino a quando non si esaminano il penultimo e l'ultimo elemento. A questo punto l'elemento più grande del vettore si troverà nell'ultima posizione.



• Logica Algoritmica: Rottura di codice

- File archivio dipendenti ordinati per matricola con stipendi nei vari mesi di un anno
- Calcolo somma stipendi per matricola

nominativo	Mese	stipendio
12477	GENNAIO	1000.00
12477	FEBBRAIO	200.00
12423	GENNAIO	700.00
12423	FEBBRAIO	400.00
18956	GENNAIO	8000.89
13545	GENNAIO	2000.00
13545	FEBBRAIO	2000.00
13545	MARZO	3999.89



nominativo	Stipendio annuo
12477	1200,00
12423	1100,00
18956	8000,89
13545	7999,89

• Logica Algoritmica: Merge

- 2 file con stipendi 2010, 2011 già ordinati per matricola
- Fare il merge vuol dire unire gli archivi
- Output stipendi 2010 2011 ordinati per matricola

Stipendi 2010

Matr.	Mese	Stip
12477	GENNAIO	1000.00
12477	FEBBRAIO	200.00
12423	GENNAIO	700.00
12423	FEBBRAIO	400.00
18956	GENNAIO	8000.89
13545	GENNAIO	2000.00
13545	FEBBRAIO	2000.00
13545	MARZO	3999.89

Stipendi 2011

Matr.	Mese	Stip
12477	GENNAIO	2000.00
12477	MARZO	400.00
12423	GENNAIO	300.00
12423	FEBBRAIO	600.00
18956	GENNAIO	8000.89
18956	GIUGNO	2000.89
13545	GENNAIO	2000.00
13545	FEBBRAIO	2000.00
13545	MARZO	3999.89
13545	MARZO	5509.00

Stipendi 2010-2011

Matr.	Mese	Stip
12477	GENNAIO	1000.00
12477	FEBBRAIO	200.00
12477	GENNAIO	2000.00
12477	MARZO	400.00
12423	GENNAIO	700.00
12423	FEBBRAIO	400.00
12423	GENNAIO	300.00
12423	FEBBRAIO	600.00
18956	GENNAIO	8000.89
18956	GIUGNO	2000.89
13545	GENNAIO	2000.00
13545	FEBBRAIO	2000.00
13545	MARZO	3999.89
13545	MARZO	5509.00

