8. DATI SEMPLICI E DATI STRUTTURATI (o STRUTTURE DATI)

Iniziamo dando una definizione generale di "**TIPO DI DATO**"

In informatica un **TIPO DI DATO** è una entità caratterizzata dai seguenti elementi:
- **un insieme X di valori** che rappresenta il "*DOMINIO*" del tipo di dato;
- **un insieme opzionale di costanti** che caratterizzano l'insieme X;
- **un insieme di operazioni** che si possono effettuare sull'insieme X.

I **tipi di dato** possono essere classificati in :

a) **tipo elementare o tipo semplice**: i cui dati non sono costituiti da altri dati ed inoltre nel corso del processo risolutivo, in un certo istante di tempo, possono assumere un unico valore;

b) **tipo strutturato o struttura dati:** i cui dati risultano essere aggregazioni di tipi elementari e/o strutturati che è possibile estrarre tramite opportune operazioni ed inoltre nel corso del processo risolutivo, in un certo istante di tempo, possono assumere più valori.

Ogni linguaggio di programmazione prevede alcuni tipi di dato. **Qui tratteremo i più usati**.

La maggior parte dei linguaggi di programmazione consente al programmatore di definire propri tipi di dato così da renderli più vicini al problema in esame.

Questi tipi di dati sono conosciuti come **TIPI DI DATI ASTRATTI o ADT (Abstract Data Type)** e vanno trattati a parte: in pratica attraverso la tecnica dell'ADT è possibile definire nuovi tipi di dato sostanzialmente più complessi e maggiormente inerenti la realtà del problema osservato.

Si parla quindi di **astrazione sui dati** in quanto si realizza una astrazione dalla loro realizzazione fisica (ossia dalla loro implementazione), dettagliando il nuovo dato in funzione delle operazioni previste per la loro manipolazione.

In altre parole durante l'attività di programmazione nel passaggio dal **problema** al **programma** eseguibile finale il concetto di **modello dei dati** o **struttura dati** compare trasversalmente:

(1) **a livello del progetto**	ossia durante la fase astratta della progettazione indipendentemente dalla tecnologia che si utilizzerà;
(2) **a livello di codifica**	ossia durante la fase di codifica o implementazione del programma;
(3) **a livello interno o fisico**	ossia a livello della modalità di memorizzazione nel calcolatore (memoria principale e/o di massa)

1) A LIVELLO DEL PROGETTO **con il termine "struttura dati"** si intende una STRUTTURA DATI ASTRATTA che è definita da:
- **un insieme base di elementi** in genere variabili che contengono i dati del problema;
- **un insieme di regole** che determinano le relazioni tra i singoli elementi dell'insieme base;
- **un elenco di operazioni** che agiscono sull'insieme**.**

Invece di **"struttura dati astratta"** si utilizzano anche i termini "**modello astratto dei dati"** o "**tipo di dato astratto o ADT".**

Il **modello astratto dei dati**:
- NON dipende dal calcolatore usato (hardware);
- NON dipende dal linguaggio di programmazione scelto per tradurre la struttura dati astratta e l'algoritmo nel programma sorgente.

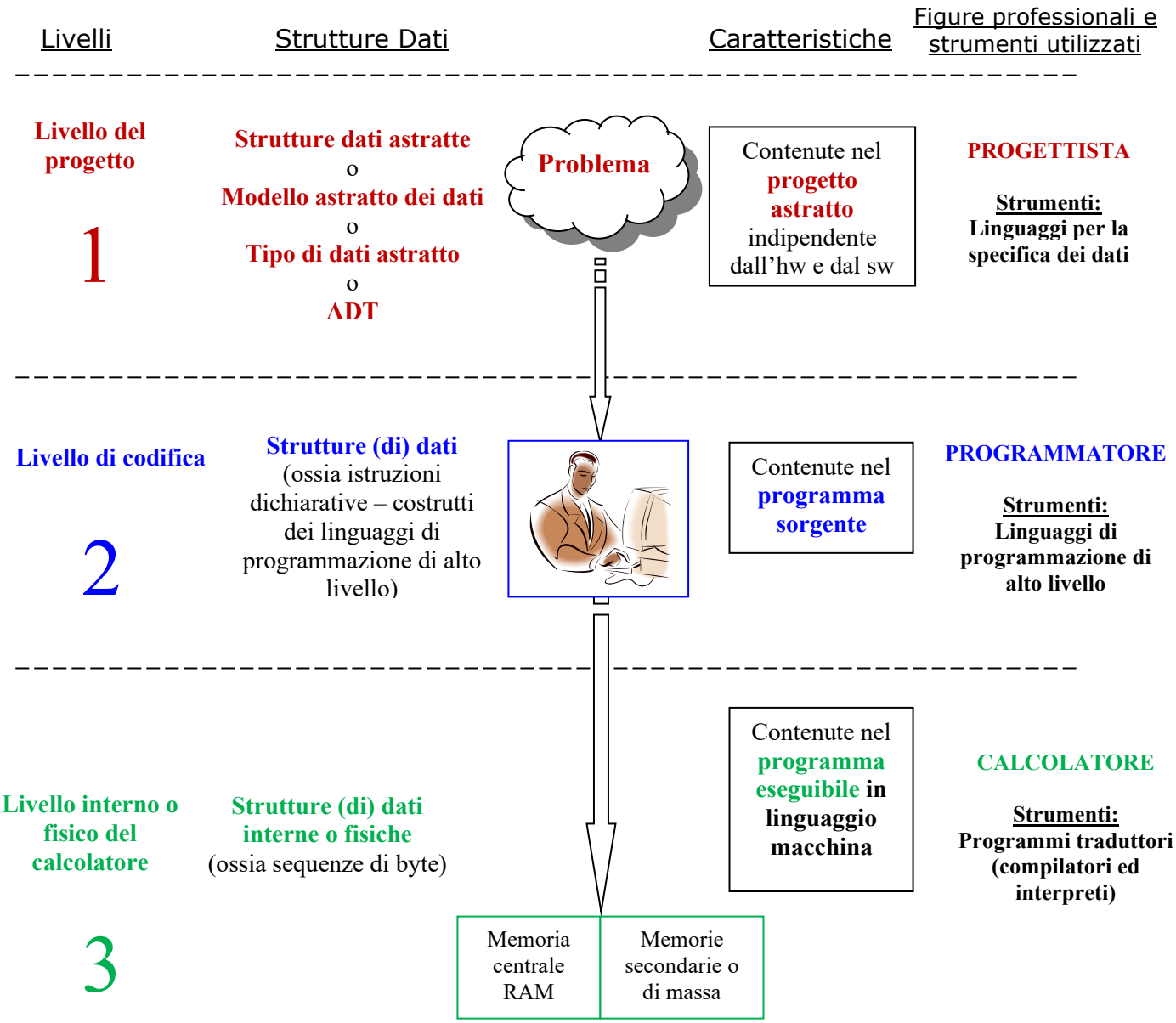
N. B. Quindi il vantaggio di fornire una soluzione astratta del problema permette di NON ridefinire da capo il modello dei dati e l'algoritmo risolutivo ogni qualvolta si cambia calcolatore e/o linguaggio di programmazione.

2) A LIVELLO DI CODIFICA O IMPLEMENTAZIONE **con il termine "struttura dati"** si intendono i COSTRUTTI DEI LINGUAGGI DI PROGRAMMAZIONE impiegati per realizzare nel programma sorgente il modello dei dati ossia la struttura dati astratta.

I **costrutti** dei linguaggi di programmazione per la **definizione delle strutture dati** si presentano come opportune frasi dichiarative, quindi non eseguibili, che descrivono al programma traduttore (compilatore o interprete) come organizzare i dati nel programma sorgente.

3) A LIVELLO INTERNO O FISICO **con il termine "struttura dati"** si intendono LE STRUTTURE "FISICHE" IMPIEGATE PER MEMORIZZARE I DATI nella memoria principale e/o di massa di un calcolatore ossia sequenze di bit memorizzate e codificate opportunamente.

SCHEMATIZZAZIONE



Abbiamo già visto cosa intendiamo con il termine "**struttura dati"** quando ci troviamo **a livello di codifica**.
Ci si riferisce in pratica ad un raggruppamento di dati organizzati in base ad un criterio che rende possibile il trattarli come un unico oggetto.

CLASSIFICAZIONE DI UNA STRUTTURA DATI

In informatica esistono diversi tipi di strutture dati che possono essere CATALOGATE (differenziate) in base alle seguenti caratteristiche:

- **il tipo di dati** di cui è composta la struttura dati :

  In questo caso si diranno **OMOGENEE** le strutture dati composte da dati tutti dello stesso tipo oppure **ETEROGENEE** le strutture dati composte da dati di tipo differente;

- **la modalità di allocazione** utilizzata per la struttura dati:

  In questo caso si diranno **STATICHE** le strutture dati per le quali la dimensione (lunghezza in byte) a loro riservata in memoria, una volta fissata, non è più modificabile oppure **DINAMICHE** le struttura dati per le quali tale dimensione può essere variata durante l'esecuzione del processo risolutivo;

- **la disposizione dei dati in memoria** della struttura dati:

  In questo caso si diranno **SEQUENZIALI** le struttura dati per le quali i singoli dati costituenti occupano locazioni "contigue" (ossia consecutive) di memoria oppure **NON SEQUENZIALI** le strutture dati per le quali i singoli dati costituenti non occupano locazioni "contigue" (ossia consecutive) di memoria;

- **il metodo di accesso** alla struttura dati (ossia il modo in cui è possibile individuare ogni singolo elemento all'interno della struttura dati)

  In questo caso si diranno ad **ACCESSO SEQUENZIALE** le strutture dati per le quali il reperimento di un singolo dato componente deve avenire obbligatoriamente scorrendo la struttura stessa dall'inizio ed analizzando gli elementi uno dopo l'altro oppure ad **ACCESSO DIRETTO** le strutture dati per le quali è possibile posizionarsi direttamente sul singolo dato componenente desiderato senza dover passare per tutti gli altri;

- **il supporto di memorizzazione** dove la struttura dati viene memorizzata:

  In questo caso si diranno **STRUTTURE DI MEMORIA** le strutture dati memorizzate nella RAM (o memoria di lavoro o memoria volatile) che vengono quindi cancellate ogni qualvolta si spegne il calcolatore oppure **FILE O ARCHIVI** le strutture dati memorizzate su memorie di massa (dischi, nastri, pen drive, sd card, etc.) che vengono quindi cancellate solo quando effettivamente richiesto (altrimenti persistono).

N.B. In questa sezione ci occuperemo esclusivamente di <u>STRUTTURE DI DATI DI MEMORIA ALLOCATE STATICAMENTE</u> sia dal punto di vista dell'ADT (ossia della schematizzazione astratta della struttura dati), sia dal punto di vista dei criteri di memorizzazione all'interno dell'elaboratore.

LA STRUTTURA DATI VETTORE O ARRAY MONODIMENSIONALE

DEF: Un **vettore o array monodimensionale** è una **struttura dati** di tipo ***sequenziale***, costituita da un insieme di elementi ***omogenei*** tra loro individuabili per mezzo di un "indice" (ossia ad ***accesso diretto***).

Per quanto riguarda la modalità di allocazione è possibile realizzare un **vettore o array** sia a carattere ***statico*** che ***dinamico***.
Per quanto riguarda il supporto di memorizzazione è possibile memorizzare i valori appartenenti ad un **vettore o array** in ***strutture di memoria*** oppure in ***archivi o file.***

N.B. Al momento ci limiteremo a trattre **vettori o array** *a* ***carattere statico*** memorizzati in opportune ***strutture di memoria*** presenti nella RAM (ossia nella memoria di lavoro volatile).

A causa della sequenzialità della struttura dati vettore l'indice, i cui valori devono appartenere all'insieme dei numeri naturali, definisce **una relazione di ordine totale** rispetto agli elementi componenti ossia permette di dire per ciascun elemento se precede o segue un altro nell'ambito del vettore stesso.

N.B. In PSEUDOCODIFICA per poter effettuare la dichiarazione di una variabile di questo tipo dovremmo scrivere:

<NomeVettore> **: ARRAY [**<MAXDIMENSIONE>**] DI** <TipoElemento>

dove:
- <NomeVettore> contiene il nome che si stabilisce per il vettore (in accordo alle regole di nomenclatura già fissate nella pseudocodifica per le variabli di tipo elementare);
- <MAXDIMENSIONE> è una costante intera CHE DEVE ASSUMERE OBBLIGATORIAMENTE un valore maggiore di zero (per poter parlare di vettore almeno un elemento ci deve essere);
- <TipoElemento> è il tipo di dati posseduto da tutti gli elementi componenti (detto *anche* ***tipo base*** del vettore)

Esempio: Se dichiaro **v: ARRAY [8] DI INT** mi riferirò ad un **vettore** o **array** monodimensionale contenente al massimo **8** elementi di elementi del tipo **intero** di nome **v.**

L'organizzazione usata nel vettore è **rigida** ossia comporta alcuni svantaggi quando occorre manipolare i suoi elementi:

1. difficoltà di inserimenti o cancellazioni: proprio a causa della contiguità delle celle di memoria dove sono rappresentati gli elementi di un vettore, l'operazione di inserimento dovrebbe prevedere la riscrittura verso destra di tutti gli elementi a seguire (shift a destra) mentre l'operazione di cancellazione dovrebbe prevedere la riscrittura verso sinistra di tutti gli elementi a seguire (shift a sinistra) per non lasciare alcuna posizione di memoria inutilizzata nel vettore;
2. dimensione statica: è necessario fissare a priori un limite massimo di elementi che il vettore potrà contenere (limite massimo non modificabile durante l'esecuzione).

OPERAZIONI SUI VETTORI O ARRAY MONODIMENSIONALI

N.B. In un algoritmo un **vettore o array** non può *quasi* mai essere manipolato come se fosse un unico oggetto ma si deve sempre operare sui singoli elementi componenti.

In altre parole, considerato un certo array di nome **v**, indipendentemente dal tipo di dati base posseduto dai suoi elementi - semplice (INT, REAL, CHAR e BOOL) o strutturato (tipo RECORD) - non si potrà *quasi* mai scrivere:

Leggi(**v**) *oppure* Scrivi(**v**)

perchè dovremo leggere o scrivere tali valori considerandoli uno alla volta.

N.B. il *quasi* è dovuto al fatto che vi è un'unica eccezione costituita dai **vettori o array monodimensionale di CHAR** (detti **STRINGHE**) per i quali, considerata l'enorme importanza da essi rivestito nell'ambito della programmazione, oltre alla "normale" gestione elemento per elemento, sono state anche previste delle apposite funzioni ed alcune deroghe a quanto detto finora. Sarà dunque possibile (ed apparirà da subito estremamente conveniente) introdurre nuove funzionalità nella pseudocodifica dedicate **ESCLUSIVAMENTE** agli **ARRAY MONODIMENSIONALI DI CARATTERI (ossia alle STRINGHE)** per ampliare la possibilità **di effettuare confronti**, **copie**, **assegnazioni** oppure **conoscere il numero complessivo di elementi**, utilizzando l'intera stringa (evitando quindi di dover procedere carattere per carattere)

A **livello logico** si possono definire delle **operazioni base** (alle quali corrispondono dei *sottoprogrammi* e quindi dei *sottoalgoritmi*) proprie di questo tipo di struttura dati:

- **CARICAMENTO (o LETTURA)** degli elementi di un vettore o array;
- **VISUALIZZAZIONE (o STAMPA)** degli elementi di un vettore o array;
- **SHIFT SINISTRO (PARZIALE o COMPLETO)** degli elementi di un vettore o array;
- **SHIFT DESTRO (PARZIALE o COMPLETO)** degli elementi di un vettore o array;
- **ROTAZIONE a SINISTRA** degli elementi di un vettore o array;
- **ROTAZIONE a DESTRA** degli elementi di un vettore o array;
- **RICERCA SEQUENZIALE** di un elemento all'interno di un vettore o array;
- **RICERCA BINARIA (o DICOTOMICA)** di un elemento all'interno di un vettore o array;
- **ORDINAMENTO INGENUO (o SEQUENZIALE)** degli elementi di un vettore o array;
- **ORDINAMENTO A BOLLE (o BUBBLE SORT)** degli elementi di un vettore o array.

Altre **operazioni** che occasionalmente possono essere richieste su questo tipo di struttura dati e che possono essere ottenute a a partire dalle precedenti sono:

- **COPIATURA:** tutto il vettore o parte di esso può dover essere copiato in un'altro vettore secondo un determinato criterio;
- **FUSIONE (o MERGE):** può essere richiesta la combinazione di 2 o più vettori in un unico vettore secondo un determinato criterio;
- **SPLIT (o SEPARAZIONE):** opposta della precedente un'unico vettore può essere suddiviso in 2 o più vettori secondo un determinato criterio.

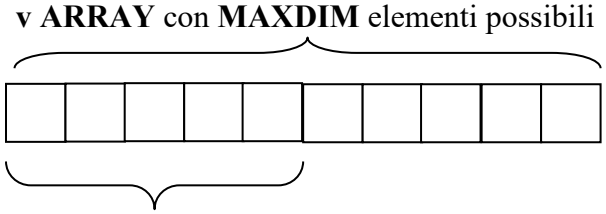
Il **CARICAMENTO o lettura** consente di assegnare un valore ad ogni elemento del vettore.

La **VISUALIZZAZIONE o stampa** consente di accedere a tutti gli elementi della struttura (oppure ad una sua parte) per visualizzarne il contenuto.

Esempio: Scrivere l'algoritmo risolutivo che permetta di caricare un qualunque vettore di interi di n elementi (con n minore o uguale alla massima dimensione fissata per il vettore in fase di dichiarazione) e visualizzarne successivamente i valori.

N.B. Invece di utilizzare esattamente tutti gli elementi fissati in fase di dichiarazione (ossia MAXDIM), lasceremo all'utente la scelta di impostare la lunghezza del vettore che desidera costruire ovviamente controllando che tale dimensione sia coerente con i limiti imposti
In questo consiste il concetto di **FALSA DINAMICITA'**.

Con la dichiarazione **v: ARRAY [MAXDIM] DI INT** noi chiediamo che ci venga allocata staticamente la seguente struttura dati



di cui intendiamo utilizzarne solo **n**

TABELLE DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da acquisire da tastiera
n	**INT**	**STATICA**	$1 \leq \mathbf{n} \leq$ MAXDIM ossia **(n** ≥ 1) AND (**n** $\leq$ MAXDIM)	Dimensione effettiva del vettore immesso da tastiera

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da visualizzare a video

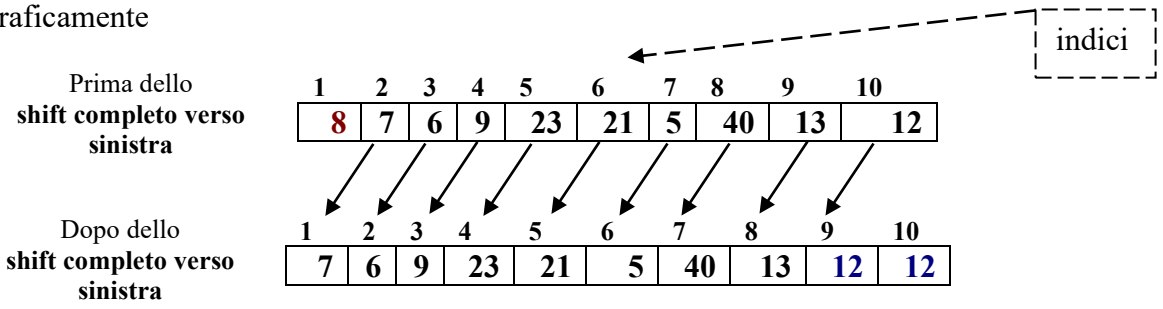
DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	**INT**	**STATICA**	**10**	Massimo numero di elementi gestibile dal vettore monodimensionale
i	**INT**	**STATICA**	$1 \leq \mathbf{i} \leq n + 1$ ossia (**i** ≥ 1) **AND** (**i** $\leq n + 1$)	Indice per poter accedere agli elementi del vettore **v**

```
ALGORITMO CaricaVisualizzaVettore
MAXDIM 10
PROCEDURA main ( )
v: ARRAY [MAXDIM] DI INT        /* ARRAY massimo gestibile */
n : INT                         /* numero di elementi da utilizzare */
i : INT
INIZIO
/* leggo la dimensione del vettore da caricare rispettando il vincolo imposto da MAXDIM */
RIPETI
  Leggi (n)
FINCHE' (n ≥ 1) AND (n ≤MAXDIM)
/* carico gli elementi nel vettore */
PER i ← 1 A n ESEGUI
  Leggi (v[i])
  i ← i + 1
FINE PER
/* visualizzo gli elementi precedentemente immessi nel vettore */
PER i ← 1 A n ESEGUI
  Scrivi (v[i])
  i ← i + 1
FINE PER
FINE
```

Lo **SHIFT** è una operazione che consente di spostare di una posizione (**a destra o a sinistra**) tutti gli elementi del vettore (**SHIFT COMPLETO**) o solo una parte di essi (**SHIFT PARZIALE**).

Nello **SHIFT COMPLETO a SINISTRA** tutti gli elementi del vettore saranno spostati di una posizione verso sinistra ad eccezione del primo elemento che va perso (ossia il contenuto della seconda posizione occuperà la prima, quello della terza la seconda, e così via fino a quello dell'ultima posizione che occuperà la penultima) con il conseguente risultato di avere in penultima ed ultima posizione lo stesso elemento.

Graficamente



N.B. Per TUTTE le operazioni di SHIFT le tabelle dei dati individuate in precedenza restano immutate

Esempio: Scrivere algoritmo risolutivo che esegua lo shift completo verso sinistra dei suoi elementi.

ALGORITMO ShiftCompletoSinistro
MAXDIM 10
PROCEDURA main ( )

v: **ARRAY** [MAXDIM] **DI INT**
n: **INT**
i: **INT**

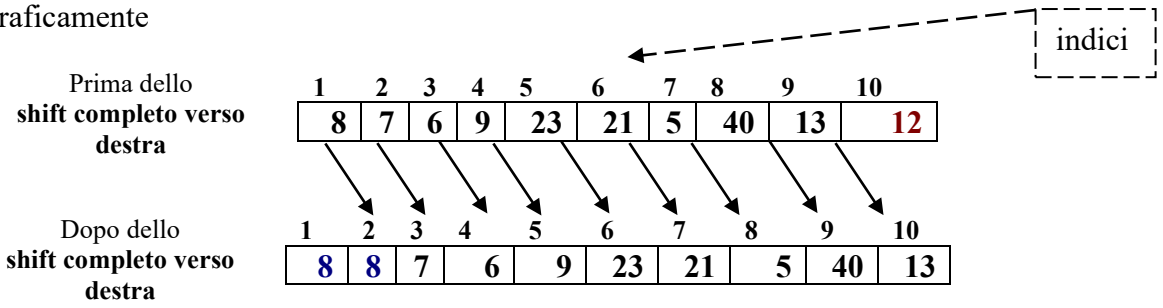
INIZIO

/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
….
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
….
/ effettua lo shift completo verso sinistra degli elementi nel vettore */*
PER i ← 1 **A** (n – 1) **ESEGUI**
  v[i] ← v[i +1]
  i ← i + 1
FINE PER

/ visualizza gli elementi precedentemente immessi nel vettore shiftati a sx (vedi esercizio precedente)*/*
….
FINE

Nello **SHIFT COMPLETO a DESTRA** tutti gli elementi del vettore saranno spostati di una posizione verso destra ad eccezione dell'ultimo elemento che va perso (ossia il contenuto della prima posizione occuperà la seconda, quello della seconda la terza, e così via fino a quello della penultima posizione che occuperà l'ultima) con il conseguente risultato di avere in prima e seconda posizione lo stesso elemento.

Graficamente



Esempio: Scrivere algoritmo risolutivo che esegua lo shift completo verso sinistra dei suoi elementi.

ALGORITMO ShiftCompletoDestra
MAXDIM 10
PROCEDURA main ( )

v: **ARRAY** [MAXDIM] **DI INT**
n: **INT**
i: **INT**

INIZIO

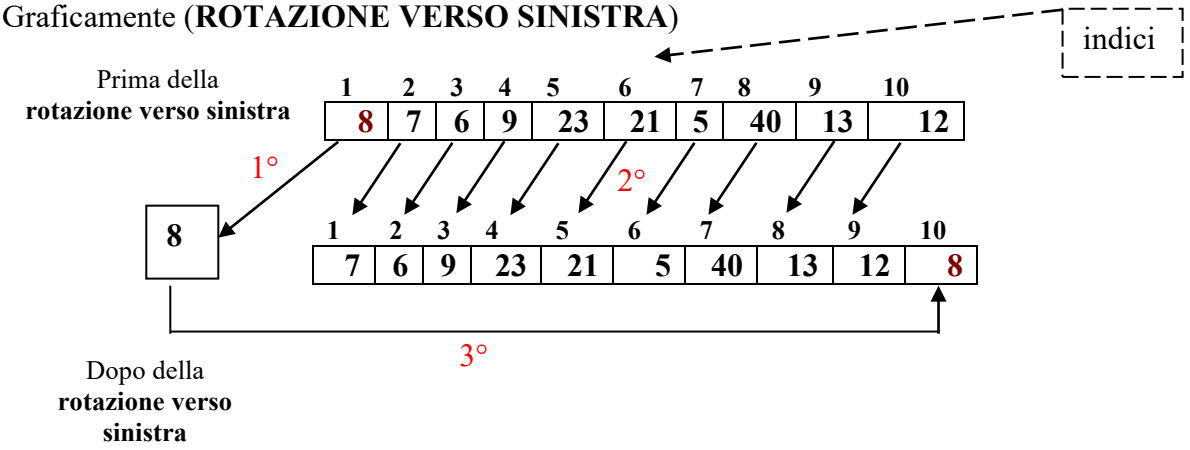
/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
….
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
….
/ effettua lo shift completo verso destra degli elementi nel vettore */*
PER i ← n **INDIETRO A** 2 **ESEGUI**
  v[i] ← v[i -1]
  i ← i - 1
FINE PER

/ visualizza gli elementi precedentemente immessi nel vettore shiftati a dx (vedi esercizio precedente)*/*
….
FINE

La **ROTAZIONE** del vettore è una operazione attraverso la quale vengono spostati tutti gli elementi del vettore **verso sinistra o verso destra** compresi gli estremi che non andranno persi, ma si ripresenteranno all'altro capo del vettore

Tecnicamente tale operazione si ottiene shiftando completamente tutti gli elementi del vettore (verso sinistra o verso destra) avendo cura di salvare il valore dell'elemento che in tale operazione andava perso, per poi collocarlo al proprio posto di competenza

Graficamente (**ROTAZIONE VERSO SINISTRA**)



N.B. Per TUTTE le operazioni di ROTAZIONE va aggiornata la tabella dei dati di lavoro o elaaborazione

TABELLE DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da acquisire da tastiera
n	**INT**	**STATICA**	$1 \le \mathbf{n} \le$ MAXDIM ossia **(n** ≥ 1**)** AND **(n** $\le$ MAXDIM**)**	Dimensione effettiva del vettore immesso da tastiera

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da visualizzare a video

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	**INT**	**STATICA**	**10**	Massimo numero di elementi gestibile dal vettore monodimensionale
i	**INT**	**STATICA**	$1 \le \mathbf{i} \le n + 1$ ossia **(i** ≥ 1**) AND (i** $\le n + 1$**)**	Indice per poter accedere agli elementi del vettore **v**
scambio	**INT**	**STATICA**	**TUTTI**	Variabile che permette lo scambio di valori del vettore

ALGORITMO RotazioneSinistra
MAXDIM 10
PROCEDURA main ( )

v: **ARRAY** [MAXDIM] **DI INT**
n: **INT**
i: **INT**
scambio: **INT**

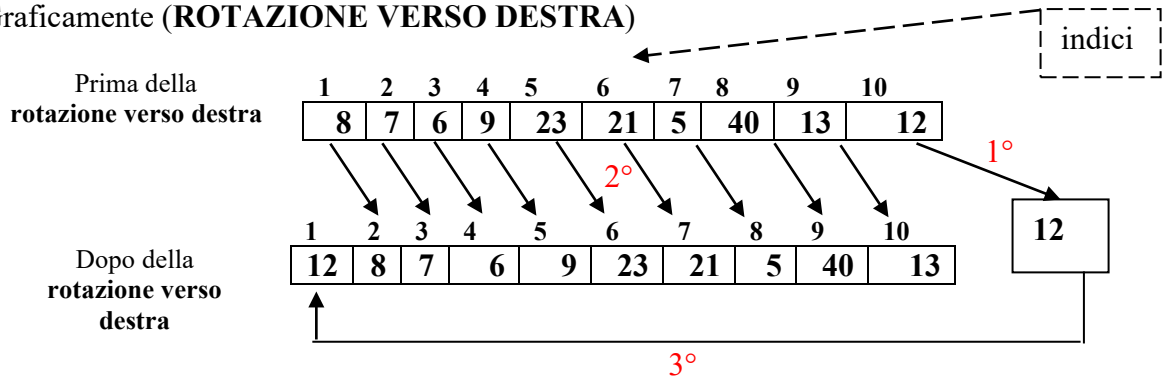
INIZIO

/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
....
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
....
/* effettua la rotazione a sinistra di TUTTI gli elementi del vettore */
/* 1° Salvataggio del primo elemento del vettore (che andrebbe perso nello shift a sinistra) */
scambio ← v[1]
/* 2° Esecuzione dello shift completo a sinistra */
PER i ← 1 **A** (n – 1) **ESEGUI**
  v[i] ← v[i +1]
  i ← i + 1
FINE PER
/* 3° Scrittura dell'elemento precedentemente salvato in ultima posizione */
v[n]← scambio

N.B. Cosa accadrebbe se scrivessi al posto di questa istruzione
v[i] ← scambio ?

/ visualizza gli elementi precedentemente immessi nel vettore ruotati a sx (vedi esercizio precedente)*/*
...
FINE

Graficamente (**ROTAZIONE VERSO DESTRA**)



ALGORITMO RotazioneDestra
MAXDIM 10
PROCEDURA main ( )

v: **ARRAY** [MAXDIM] **DI INT**
n: **INT**
i: **INT**
scambio: **INT**

INIZIO

/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
….
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
….
/* effettua la rotazione a destra di TUTTI gli elementi del vettore */
/* 1° Salvataggio dell'ultimo elemento del vettore (che andrebbe perso nello shift a destra) */
scambio ← v[n]
/* 2° Esecuzione dello shift completo a destra */
PER i ← n **INDIETRO A** 2 **ESEGUI**
  v[i] ← v[i - 1]
  i ← i - 1
FINE PER
/* 3° Scrittura dell'elemento precedentemente salvato in prima posizione */
v[1]← scambio

N.B. Cosa accadrebbe se scrivessi al posto di questa istruzione
v[i] ← scambio ?

/ visualizza gli elementi precedentemente immessi nel vettore ruotati a dx (vedi esercizio precedente)*/*
….
FINE

ALGORITMI DI ORDINAMENTO su ARRAY MONODIMENSONALI

Le operazioni di **RICERCA** di un elemento all'interno di un vettore e di **ORDINAMENTO** (in senso crescente o decrescente) degli elementi di un vettore sono molto importanti e sono stati oggetto di studi approfonditi in ambito informatico.

Esistono numerosi algoritmi ormai codificati e riconosciuti che affrontano e risolvono, ciascuno con le sue caratteristiche, le problematiche connesse con le attività di ORDINAMENTO di un array monodimensionale. Noi ne analizzeremo in dettaglio solo alcuni.

A) Algoritmo di ORDINAMENTO INGENUO

E' il più intuitivo ed inefficace metodo di ordinamento che consiste nel **confrontare** ciascun elemento con tutti quelli di posto superiore all'interno di una serie di "**scansioni**", **scambiando** tra loro i valori se necessario in base all'ordinamento prescelto.
Se nella prima scansione si confronterà il primo elemento con il secondo poi con il terzo e così via fino all'ultimo effettuando tutti gli scambi necessari, nella scansione successiva, ovviamente, si confronterà il secondo elemento (ossia il successivo) con il terzo e così via fino all'ultimo, sempre effttuando tutti gli scambi necessari.
L'ultima scansione, la **n-1** esima, si occuperà del confronto tra il penultimo elemento del vettore con l'ultimo, efettuando lo scambio dei valori se necessario.

N.B. Quindi l'obiettivo delle **n-1 scansioni** sarà SEMPRE quello di posizionare, NELLE POSIZIONI LIBERE PIU' A SINISTRA, gli elementi più piccoli in caso di ordinamento crescente o più grandi in caso di ordinamento decrescente.

TABELLE DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da acquisire da tastiera
n	**INT**	**STATICA**	$1 \leq \mathbf{n} \leq$ MAXDIM ossia $(\mathbf{n} \geq 1)$ AND $(\mathbf{n} \leq$ MAXDIM)	Dimensione effettiva del vettore immesso da tastiera

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da visualizzare a video

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	**INT**	**STATICA**	**10**	Massimo numero di elementi gestibile dal vettore monodimensionale
i	**INT**	**STATICA**	$1 \leq \mathbf{i} \leq n + 1$ ossia $(\mathbf{i} \geq 1)$ **AND** $(\mathbf{i} \leq n + 1)$	Indice per poter accedere agli elementi del vettore **v** ed effettuare le scansioni
j	**INT**	**STATICA**	$1 \leq \mathbf{j} \leq n + 1$ ossia $(\mathbf{i} \geq 1)$ **AND** $(\mathbf{i} \leq n + 1)$	Indice per poter individuare l'elemento di v da confrontare di volta in volta
scambio	**INT**	**STATICA**	**TUTTI**	Variabile che permette lo scambio di valori del vettore per ciascuna scansione

```
ALGORITMO OrdinamentoIngenuo
MAXDIM 10
PROCEDURA main ( )
v: ARRAY [MAXDIM] DI INT
n: INT
i, j, scambio: INT

INIZIO
/* leggo la dimensione del vettore da caricare (vedi esercizio precedente)*/
....
/* carico gli elementi nel vettore(vedi esercizio precedente) */
....
/* effettuo l'ordinamento in senso crescente per scambio del vettore/
PER i ← 1 A (n – 1) ESEGUI              /* INIZIO ciclo per "scansioni" */
   PER j ← i+1 A n ESEGUI                    /* INIZIO ciclo per "confronti" */
      SE (v[i] > v[j])   /* N.B. senso CRESCENTE altrimenti con '<' senso DECRESCENTE */
         ALLORA
            scambio ← v[i]
            v[i] ← v[j]
            v[j] ← scambio
       FINE SE
       j ← j + 1
   FINE PER                                  /* FINE ciclo per "confronti" */
   i ← i + 1
FINE PER                                  /* FINE ciclo per "scansioni" */
/* visualizzo gli elementi del vettore dopo l'ordinamento (vedi esercizio precedente)*/
....
FINE
```

Esempio: Supponiamo di volere descrivere la tecnica di **ORDINAMENTO INGENUO** (**in senso CRESCENTE**) applicata al seguente vettore di **interi** di nome **v** costituito da **quattro** elementi (ossia **n = 4**)

Vettore di partenza di nome v

1	2	3	4
25	**10**	**19**	**4**

i ← 1 (i = 1)

TEST 1° PER (i ≤ n – 1) ossia (1 ≤ 3) VERO **Inizio Prima scansione**

j ← i + 1 (j = 1 + 1 = 2)

TEST 2° PER (j ≤ n) ossia (2 ≤ 4) VERO

1° CICLO 2° PER **1° passo**

confrontiamo il valore di v[1] con il valore di v[2]. **Primo Confronto**
Poiché 25 è maggiore di 10 si deve effettuare lo scambio degli elementi

1	2	3	4
10	**25**	**19**	**4**

j ← j + 1 (j = 2 + 1 = 3)

TEST 2° PER (j ≤ n) ossia (3 ≤ 4) VERO

2° CICLO 2° PER **2° passo**

confrontiamo il valore di v[1] con il valore di v[3]. **Secondo Confronto**
Poiché 10 è minore di 19 NON si deve effettuare lo scambio degli elementi

1	2	3	4
10	**25**	**19**	**4**

j ← j + 1 (j = 3 + 1 = 4)

TEST 2° PER (j ≤ n) ossia (4 ≤ 4) VERO

3° CICLO 2° PER **3° passo**

confrontiamo il valore di v[1] con il valore di v[4]. **Terzo Confronto**
Poiché 10 è maggiore di 4 si deve effettuare lo scambio degli elementi

1	2	3	4
4	**25**	**19**	**10**

j ← j + 1 (j = 4 + 1 = 5)

TEST 2° PER (j ≤ n) ossia (5 ≤ 4) **FALSO** **Fine Prima scansione**

N.B. Alla fine della **prima scansione** del vettore siamo riusciti a posizionare l'elemento dal valore più piccolo in prima posizione.

i ← i + 1 (i = 1 + 1 = 2)

TEST 1° PER ($i \le n - 1$) ossia ($2 \le 3$) VERO **Inizio Seconda scansione**

$j \leftarrow i + 1$ ($j = 2 + 1 = 3$)

TEST 2° PER ($j \le n$) ossia ($3 \le 4$) VERO

1° CICLO 2° PER **1° passo**

confrontiamo il valore di v[2] con il valore di v[3]. **Primo Confronto**
Poiché 25 è maggiore di 19 si deve effettuare lo scambio degli elementi

1	2	3	4
4	**19**	**25**	**10**

$j \leftarrow j + 1$ ($j = 3 + 1 = 4$)

TEST 2° PER ($j \le n$) ossia ($4 \le 4$) VERO

1° CICLO 2° PER **2° passo**

confrontiamo il valore di v[2] con il valore di v[4]. **Secondo Confronto**
Poiché 19 è maggiore di 10 si deve effettuare lo scambio degli elementi

1	2	3	4
4	**10**	**25**	**19**

$j \leftarrow j + 1$ ($j = 4 + 1 = 5$)

TEST 2° PER ($j \le n$) ossia ($5 \le 4$) **FALSO** **Fine Seconda scansione**

N.B. Alla fine della **seconda scansione** del vettore siamo riusciti a posizionare il più piccolo del sottovettore (vettore – primo elemento) in seconda posizione.

$i \leftarrow i + 1$ ($i = 2 + 1 = 3$)

TEST 1° PER ($i \le n - 1$) ossia ($3 \le 3$) VERO **Inizio Terza scansione**

$j \leftarrow i + 1$ ($j = 3 + 1 = 4$)

TEST 2° PER ($j \le n$) ossia ($4 \le 4$) VERO

1° CICLO 2° PER **1° passo**

confrontiamo il valore di v[3] con il valore di v[4]. **Primo Confronto**
Poiché 25 è maggiore di 19 si deve effettuare lo scambio degli elementi

1	2	3	4
4	**10**	**19**	**25**

$j \leftarrow j + 1$ ($j = 4 + 1 = 5$)

TEST 2° PER ($j \le n$) ossia ($5 \le 4$) **FALSO** **Fine Terza scansione**

Alla fine della **terza scansione** del vettore il vettore è ordinato (il numero delle scansioni effettuate è 3 che è uguale, come previsto, al numero degli elementi del vettore **n – 1**)

B) Algoritmo di ordinamentoBUBBLE-SORT o "A BOLLE"

Viene utilizzato nella realtà solo per dati "poco disordinati" e consiste nel confrontare a 2 a 2 gli elementi, scambiandoli se necessario (ossia primo e secondo, secondo e terzo, terzo e quarto, …, penultimo ed ultimo), facendo risalire (ecco il concetto di "bolla") verso l'alto (indice più grande) attraverso questa ripetizione di scambi, gli elementi più grandi in caso di ordinamento crescente o più piccoli in caso di ordinamento decrescente.

Per descrrivere l'algoritmo di ordinamento **BUBBLE-SORT** quindi dovremmo utilizzare anche:

- una variabile booleana che ci indichi che nella scansione non sono avvenuti scambi di valori tra elementi (chiamiamo questa variabile *continua*);
- una variabile intera che ci indichi il limite superiore (ossia l'indice dell'elemento) fino al quale fare gli scambi per evitare ripetizioni inefficienti (chiamiamola *sup)* che viene inizializzata la prima volta con la dimensione del vettore e che assumerà valori via via decrescenti, fino a che il vettore non risulterà ordinato;
- una variabile intera (chiamiamola *k*) che dopo l'esecuzione di ogni "scansione" indichi la posizione del nuovo estremo superiore da considerare.

TABELLE DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da acquisire da tastiera
n	**INT**	**STATICA**	1 ≤ **n** ≤ MAXDIM ossia (**n** ≥ 1) AND (**n** ≤ MAXDIM)	Dimensione effettiva del vettore immesso da tastiera

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da visualizzare a video

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	**INT**	**STATICA**	**10**	Massimo numero di elementi gestibile dal vettore monodimensionale
i	**INT**	**STATICA**	1 ≤ **i** ≤ n + 1 ossia (**i** ≥ 1) **AND** (**i** ≤ n + 1)	Indice per poter accedere agli elementi del vettore **v** ed effettuare le scansioni
scambio	**INT**	**STATICA**	**TUTTI**	Variabile che permette lo scambio di valori del vettore per ciascuna scansione
sup	**INT**	**STATICA**	1 ≤ **sup** ≤ n ossia (**sup** ≥ 1) **AND** (**sup** ≤ n)	Indice dell'ultimo elemento del vettore da considerare per gli scambi
k	**INT**	**STATICA**	1 ≤ **k** ≤ n ossia (**k** ≥ 1) **AND** (**k** ≤ n)	Indice dell'ultimo elemento del vettore che è stato scambiato (il nuovo sup)
continua	**BOOL**	**STATICA**	**VERO o FALSO**	Variabile che indica se nella scnsione sono avvenuti scambi

ALGORITMO 1°_OrdinamentoBubbleSort **(con CICLO MENTRE per le scansioni)**
MAXDIM 10
PROCEDURA main ( )
v: **ARRAY** [MAXDIM] **DI INT**
n, scambio: **INT**
i, sup, k: **INT**
continua: **BOOL**
INIZIO
/ leggo la dimensione del vettore da caricare (vedi esercizio precedente)*/*
....
/ carico gli elementi nel vettore(vedi esercizio precedente) */*
....
/ effettuo l'ordinamento in senso crescente bubble-sort del vettore/*
k ← n
continua ← VERO
MENTRE (**continua = VERO**) **ESEGUI** */* INIZIO ciclo per "scansioni" */*
sup ← k
continua ← FALSO
PER i ← 1 **A** (sup -1) **ESEGUI** */* INIZIO ciclo per "confronti" */*
SE (v[i] > v[i+1]) */* N.B. senso crescente altrimenti con '<' senso decrescente*/*
ALLORA
scambio ← v[i]
v[i] ← v[i+1]
v[i+1] ← scambio
k ← i */* N.B. così si accorcia il sottovettore da esaminare */*
continua ← VERO
FINE SE
i ← i + 1
FINE PER */* FINE ciclo per "confronti" */*
FINE MENTRE */* FINE ciclo per "scansioni" */*
/ visualizzo gli elementi precedentemente immessi nel vettore (vedi esercizio precedente)*/*
....
FINE

ALGORITMO 2°_OrdinamentoBubbleSort **(con CICLO RIPETI per le scansioni)**
MAXDIM 10
PROCEDURA main ( )
v: **ARRAY** [MAXDIM] **DI INT**
n, scambio: **INT**
i, sup, k: **INT**
continua: **BOOL**
INIZIO
/ leggo la dimensione del vettore da caricare (vedi esercizio precedente)*/*
....
/ carico gli elementi nel vettore(vedi esercizio precedente) */*
....
/ effettuo l'ordinamento in senso crescente bubble-sort del vettore/*
k ← n
RIPETI */* INIZIO ciclo per "scansioni" */*
  sup ← k
  continua ← FALSO
  PER i ← 1 **A** (sup -1) **ESEGUI** */* INIZIO ciclo per "confronti" */*
    SE (v[i] > v[i+1]) */* N.B. senso crescente altrimenti con '<' senso decrescente*/*
      ALLORA
        scambio ← v[i]
        v[i] ← v[i+1]
        v[i+1] ← scambio
        k ← i */* N.B. così si accorcia il sottovettore da esaminare */*
        continua ← VERO
    FINE SE
    i ← i + 1
  FINE PER */* FINE ciclo per "confronti" */*
FINCHE' (continua = FALSO) */* FINE ciclo per "scansioni" */*
/ visualizzo gli elementi precedentemente immessi nel vettore (vedi esercizio precedente)*/*
....
FINE

Esempio: Supponiamo di volere descrivere la tecnica di **ORDINAMENTO** per **BUBBLE-SORT** (**in senso CRESCENTE con il 1° algoritmo**) applicata al medesimo vettore di **interi** di nome **v** costituito da **quattro** elementi (ossia **n = 4**) utilizzato in precedenza.

Vettore di partenza di nome v

1	2	3	4
25	**10**	**19**	**4**

L'algoritmo sopra descritto, per ottenere l'ordinamento crescente del vettore, farà:

INIZIO

k ← n (k = 4)
continua ← VERO (continua = VERO)

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO
1° ciclo MENTRE **Inizio Prima scansione**
sup ← k (sup = 4)
continua ← FALSO (continua = FALSO) n.b. ogni volta si resetta l'indicatore degli scambi effettuati

i ← 1 (i = 1)
TEST PER (i ≤ sup-1) ossia (1 ≤ 3) VERO

1° ciclo PER **1° passo**

confrontiamo il valore di v[1] con il valore di v[2] (ossia v[1] > v[2]?) **Primo Confronto**
Poiché 25 è maggiore di 10 si deve effettuare lo scambio degli elementi

1	2	3	4
10	**25**	**19**	**4**

k← i (k = 1)
continua ← VERO (continua = VERO)
i ← i + 1 (i = 1 + 1 = 2)
TEST PER (i ≤ sup-1) ossia (2 ≤ 3) VERO

2° ciclo PER **2° passo**

confrontiamo il valore di v[2] con il valore di v[3] (ossia v[2] > v[3]?) **Secondo Confronto**
Poiché 25 è maggiore di 19 si deve effettuare lo scambio degli elementi

1	2	3	4
10	**19**	**25**	**4**

k← i (k = 2)
continua ← VERO (continua = VERO)
i ← i + 1 (i = 2 + 1 = 3)
TEST PER (i ≤ sup-1) ossia (3 ≤ 3) VERO

3° ciclo PER **3° passo**

confrontiamo il valore di v[3] con il valore di v[4] (ossia v[3] > v[4]?) **Terzo Confronto**
Poiché 25 è maggiore di 4 si deve effettuare lo scambio degli elementi

1	2	3	4
10	**19**	**4**	**25**

k← i (k = 3)
continua ← VERO (continua = VERO)
i ← i + 1 (i = 3 + 1 = 4)
TEST PER (i ≤ sup-1) ossia (4 ≤ 3) FALSO **Fine Prima scansione**

N.B. Alla fine della **prima scansione** del vettore siamo riusciti a posizionare l'elemento dal valore più grande alla fine del vettore (che è risalito come una "bolla" in un liquido).

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO
2° ciclo MENTRE **Inizio Seconda scansione**
sup ← k (sup = 3)
continua ← FALSO (continua = FALSO) n.b. ogni volta si resetta l'indicatore degli scambi effettuati

i ← 1 (i = 1)
TEST PER (i ≤ sup-1) ossia (1 ≤ 2) VERO

1° ciclo PER **1° passo**

confrontiamo il valore di v[1] con il valore di v[2] (ossia v[1] > v[2]?) **Primo Confronto**
Poiché 10 è minore di 19 non si deve effettuare lo scambio degli elementi

1	2	3	4
10	**19**	**4**	**25**

i ← i + 1 (i = 1 + 1 = 2)
TEST PER (i ≤ sup-1) ossia (2 ≤ 2) VERO

2° ciclo PER **2° passo**

confrontiamo il valore di v[2] con il valore di v[3] (ossia v[2] > v[3]?) **Secondo Confronto**
Poiché 19 è maggiore di 4 si deve effettuare lo scambio degli elementi

1	2	3	4
10	**4**	**19**	**25**

k← i (k = 2)
continua ← VERO (continua = VERO)
i ← i + 1 (i = 2 + 1 = 3)
TEST PER (i ≤ sup-1) ossia (3 ≤ 2) FALSO **Fine Seconda scansione**

N.B. Alla fine della **seconda scansione** del vettore siamo riusciti a posizionare l'elemento dal valore più grande in penultima posizione (ossia nell'ultima posizione del sottovettore – 1)

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO
3° ciclo MENTRE **Inizio Terza scansione**
sup ← k (sup = 2)
continua ← FALSO (continua = FALSO) n.b. ogni volta si resetta l'indicatore degli scambi effettuati

i ← 1 (i = 1)
TEST PER (i ≤ sup-1) ossia (1 ≤ 1) VERO

1° ciclo PER **1° passo**

confrontiamo il valore di v[1] con il valore di v[2] (ossia v[1] > v[2]?) **Primo Confronto**
Poiché 10 è maggiore di 4 si deve effettuare lo scambio degli elementi

1	2	3	4
4	**10**	**19**	**25**

k← i (k = 1)
continua ← VERO (continua = VERO)
i ← i + 1 (i = 1 + 1 = 2)
TEST PER (i ≤ sup-1) ossia (2 ≤ 1) FALSO **Fine Terza scansione**

N.B. Alla fine della **terza scansione** del vettore siamo riusciti a posizionare l'elemento dal valore più grande in seconda posizione.
Il vettore è ordinato ma ancora non è terminato il ciclo MENTRE esterno perché è stato fatto almeno uno scambio (continua = VERO)

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO
4° ciclo MENTRE
sup ← k (sup = 1)
continua ← FALSO (continua = FALSO) n.b. ogni volta si resetta l'indicatore degli scambi effettuati
i ← 1 (i = 1)
TEST PER (i ≤ sup-1) ossia (1 ≤ 0) FALSO Ciclo PER non eseguito **NO SCANSIONE**

TEST MENTRE (continua = VERO) ossia (FALSO = VERO)? FALSO **STOP**

N.B. Il vettore ora risulterà perfettamente ordinato

1	2	3	4
4	**10**	**19**	**25**

LE DUE STRATEGIE DI ORDINAMENTO A CONFRONTO

Se confrontiamo, **nel caso del vettore proposto**, i due algoritmi equivalenti di ordinamento tra di loro ci accorgeremo che essi hanno compiuto esattamente lo stesso numero di scansioni e di passi per giungere al medesimo scopo e quindi sembrerebbero perfettamente equivalenti.

ATTENZIONE: NON E SEMPRE COSI'!

Basterebbe ripetere per esercizio le scansioni/confronti che i due algoritmi di ordinamento impiegano per ordinare in senso CRESCENTE il seguente altro vettore **v** di **interi** di dimensione quattro (**n = 4**):

1	2	3	4
8	**3**	**18**	**23**

per accorgersi della maggiore efficienza dell'ordinamento BUBBLE SORT rispetto all'ordinamento INGENUO, soprattutto per i vettori che mostrano valori poco disordinati (o in altri termini più ordinati) rispetto al criterio di ordinamento richiesto.
In questoi caso infatti:

- l'ordinamento INGENUO otterrà l'ordinamento in senso CRESCENTE del vettore effettuando comunque **n-1 scansioni** (con la prima scansione che effettuerà esatamente **n-1 confronti**);
- l'ordinamento a bolle o BUBBLE-SORT otterrà l'ordinamento in senso CRESCENTE del vettore effettuando **2 sole scansioni** (grazie al poco disordine dei valori rispetto al criterio fissato).

Quindi ciò che fa la differenza in questo caso non è solo il numero iniziale degli elementi del vettore (**dimensione**), ma soprattutto la loro **disposizione iniziale** (ossia come tali valori sono distribuiti).

ALGORITMI DI RICERCA di un elemento in un ARRAY MONODIMENSIONALI

A) ALGORITMO DI RICERCA SEQUENZIALE

Confronta un elemento fornito in input (ovviamente dello stesso tipo cui appartengono gli elementi del vettore) con tutti gli elementi di un vettore o array monodimensionale, uno alla volta partendo dal primo: in altre parole tale algoritmo consiste nell'effettuare una serie di confronti tra il valore dell'elemento da ricercare con tutti gli altri elementi in esso presenti.

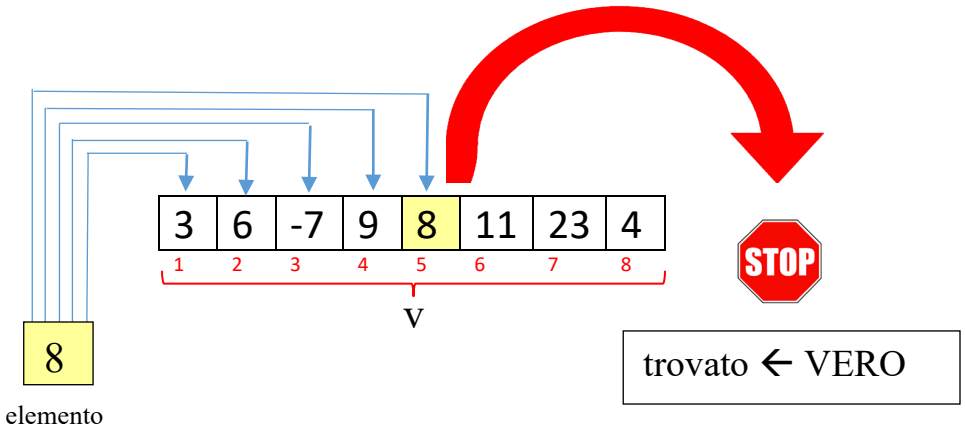
I confronti terminano nel momento in cui si trova l'elemento fornito in input oppure si è raggiunta la fine del vettore (**ricerca sequenziale**).

Tali confronti possono anche continuare sino alla fine del vettore nel caso in cui si desideri contare il numero di volte (*occorrenze*) che tale valore compare all'interno del vettore, non arrestandosi alla prima. In questo caso si parla di **scansione sequenziale**.

Nel caso dell'algoritmo di RICERCA SEQUENZIALE ipotizziamo i due possibili casi:

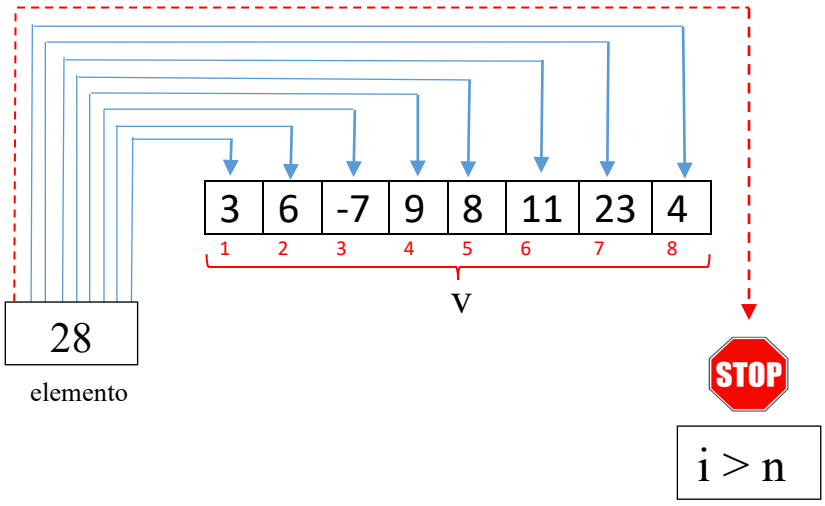
- **CASO 1**: **l'elemento E' PRESENTE all'interno del vettore**.

Dunque il procedimento di ricerca **DEVE ESSERE ARRESTATO** una volta che il confronto tra l'elemento in input ed il generico elemento del vettore avrà avuto **ESITO POSITIVO**



- **CASO 2**: **l'elemento NON E' PRESENTE all'interno del vettore**.

Dunque il procedimento di ricerca **DEVE ESSERE ARRESTATO** una volta che abbiamo effettuato **tutti i confronti possibili con gli elementi presenti nel vettore** (ossia gli elementi del vettore da confrontare sono terminati).



Quindi l'algoritmo di **ricerca sequenziale** dovrà basarsi su una **ISTRUZIONE ITERATIVA NON ENUMERATIVA** (perché non so a priori se troverò o meno l'elemento all'interno del vettore) governato dal seguente enunciato composto:

TABELLE DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY[MAXDIM] DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da acquisire da tastiera
n	**INT**	**STATICA**	$1 \leq$ **n** $\leq$ **MAXDIM** ossia (**n** ≥ 1) **AND** (**n** $\leq$ **MAXDIM**)	Dimensione effettiva del vettore immesso da tastiera
elemento	**INT**	**STATICA**	**TUTTI**	Elemento da ricercare nel vettore

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
trovato	**BOOL**	**STATICA**	**VERO o FALSO**	Variabile che indica l'esito della ricerca
posizione	**INT**	**STATICA**	$0 \leq$ **posizione** $\leq$ n ossia (**posizione** ≥ 0) **AND** (**posizione** $\leq$ n)	Posizione eventuale dell'elemento trovato nel vettore v

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	**INT**	**STATICA**	**10**	Massimo numero di elementi gestibile dal vettore monodimensionale
i	**INT**	**STATICA**	$1 \leq$ **i** $\leq$ n + 1 ossia (**i** ≥ 1) **AND** (**i** $\leq$ n + 1)	Indice per poter accedere agli elementi del vettore **v**

RICERCA SEQUENZIALE

```
ALGORITMO RicercaSequenziale
MAXDIM 10
PROCEDURA main ( )
v: ARRAY [MAXDIM] DI INT
[posizione], i, n, elemento: INT
trovato: BOOL
INIZIO
/* leggo la dimensione del vettore da caricare (vedi esercizio precedente)*/
….
/* carico gli elementi nel vettore(vedi esercizio precedente) */
….
/*leggo l'elemento da ricercare */
Leggi (elemento)

/* effettuo la ricerca dell'elemento all'interno del vettore arrestandosi nel caso lo trovi */
/* e tenendo conto della prima posizione utile in cui è stato trovato                        */
[posizione ← 0]                                  /* inizializzo, se richiesta, la posizione */
trovato ← FALSO
i ← 1
MENTRE ((trovato = FALSO) AND (i ≤ n)) ESEGUI
   SE (v[i] = elemento)                  /* N.B. Vanno bene anche le seguenti condizioni logiche */
      ALLORA                                /* (NOT trovato) oppure (NOT trovato = VERO) */
         trovato ← VERO
         [posizione ← i]                    /* conservo, se richiesta, la posizione dell'elemento */
   FINE SE
   i ← i + 1                                /* incremento fondamentale del'indice */
FINE MENTRE
/* comunico l'esito all'utente */
Scrivi (trovato)
SE (trovato = VERO)
   ALLORA
      Scrivi ("L'elemento è stato trovato")
      [Scrivi(posizione)]              /* mostro a video, se richiesta, la posizione dell'elemento*/
   ALTRIMENTI
      Scrivi ("L'elemento non è stato trovato")
FINE SE
FINE
```

OSSERVAZIONI

L'algoritmo di **ricerca sequenziale** di un elemento in un vettore o array monodimensionale, effettuerà al massimo nel caso pessimo (ossia quello in cui l'elemento fornito non è presente al suo interno)**n confronti**.

Caso particolare:

SCANSIONE SEQUENZIALE ossia ricerca sequenziale con conteggio del numero di occorrenze

```
ALGORITMO ScansioneSequenziale
MAXDIM 10
PROCEDURA main ( )
v: ARRAY [MAXDIM] DI INT
i, n, elemento, numoccorrenze: INT
INIZIO
/* leggo la dimensione del vettore da caricare (vedi esercizio precedente)*/
….
/* carico gli elementi nel vettore(vedi esercizio precedente) */
….
/*leggo l'elemento da ricercare */
Leggi (elemento)
/* effettuo la ricerca dell'elemento all'interno del vettore non arrestandosi  */
/* e tenendo il conto del numero di volte in cui eventualmente compare        */
numcorrenze ← 0
PER i← 1 A n ESEGUI
  SE (v[i] = elemento)
      ALLORA
        numoccorrenze ← numoccorrenze + 1
  FINE SE
  i ← i + 1                                  /* incremento fondamentale dell'indice */
FINE PER
/* comunico l'esito all'utente */
SE (numoccorrenze > 0)
  ALLORA
    Scrivi ("L'elemento è stato trovato")
    Scrivi (numoccorrenze)
  ALTRIMENTI
    Scrivi ("L'elemento non è stato trovato")
FINE SE
FINE
```

B) RICERCA BINARIA o DICOTOMICA

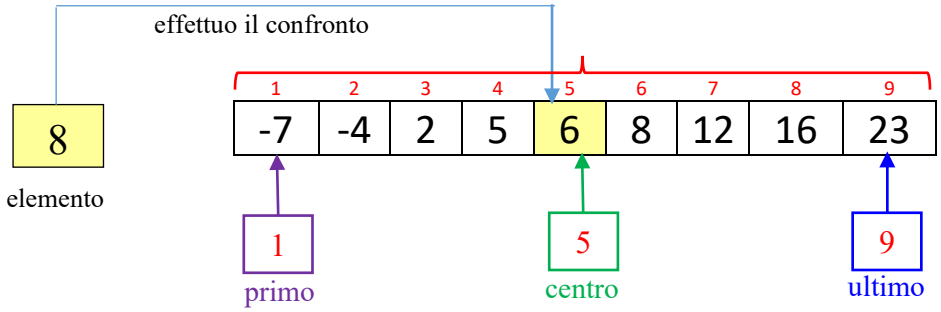
PREMESSA FONDAMENTALE

Tale strategia di ricerca per funzionare RICHIEDE OBBLIGATORIAMENTE CHE gli elementi del vettore SIANO STATI PRECEDENTEMENTE ORDINATI (in modo CRESCENTE oppure DECRESCENTE)

Il procedimento risolutivo di questo algoritmo si basa sull'individuazione di una "porzione" del vettore sul quale effettuare la ricerca dell'elemento assegnato in input.
La "porzione" di vettore da utilizzare nella ricerca viene determinata identificando di volta in volta:
- l'elemento in posizione iniziale attraverso il suo indice di riferimento chiamato ***primo***;
- l'elemento in posizione finale attraverso il suo indice di riferimento chiamato ***ultimo***;
- l'elemento in posizione centrale attraverso il suo indice di riferimento chiamato ***centro*** il cui valore verrà calcolato tramite l'espressione ***centro ← (primo + ultimo) DIV 2***

Ovviamente all'inizio dell'algoritmo la "porzione" di vettore corrente corrisponderà all'intera struttura dati, ma essa si andrà via via riducendo a seconda dell'esito del confronto tra l'elemento da ricercare fornito in input e l'elemento che occupa la posizione centrale.

Esempio: Ricerca di un elemento in un vettore formato da n = 9 elementi interi ordinati in senso CRESCENTE



CASO 1
L'elemento da ricercare
E' PRESENTE
nel vettore **v**

1° Passo

All'inizio la "porzione" del vettore su cui effettuare la ricerca, corrisponderà a quella con i valori dell'indice compresi tra:

primo ← 1 ed ultimo ← 9

ne consegue che in tale ipotesi l'elemento situato in posizione centrale avrà indice:

centro ← (primo + ultimo) DIV 2 ossia centro ← (1 + 9) DIV 2 ossia centro ← 5

Dopo di ciò sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

1. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato.

Allora l'elemento ricercato appartiene al vettore e si trova in posizione centrale.
La RICERCA in questo caso DEVE ESSERE ARRESTATA
Nel nostro caso NO domanda: elemento = v[centro] ? ossia 8 = v[5] ? in quanto 8 = 6 E' FALSO

2. l'elemento da ricercare è più piccolo dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà destra del vettore e l'ultimo elemento del vettore utile alla ricerca diventa quello immediatamente precedente all'elemento centrale (ossia ultimo← centro – 1)
Nel nostro caso NO domanda: elemento < v[centro] ? ossia 8 < v[5] ? in quanto 8 < 6 E' FALSO

3. l'elemento da ricercare è più grande dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà sinistra del vettore ed il primo elemento del vettore utile alla ricerca diventa quello immediatamente successivo all'elemento centrale (ossia primo← centro + 1)
Nel nostro caso SI domanda: elemento > v[centro] ? ossia 8 > v[5] ? in quanto 8 > 6 E' VERO
La RICERCA in questo caso DEVE CONTINUARE su una nuova porzione del "vettore" da specificare.

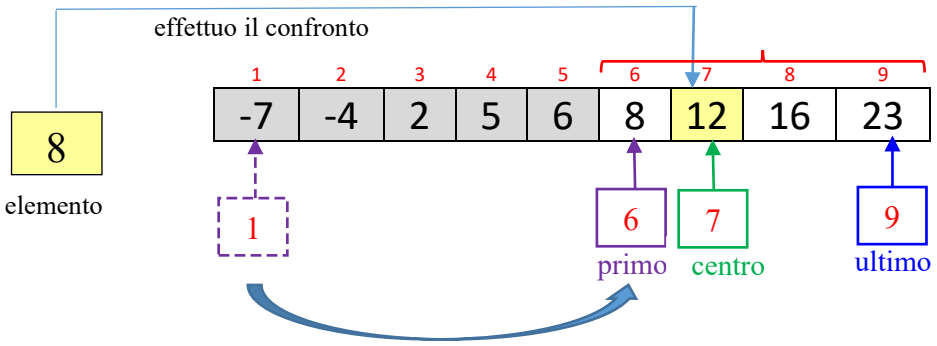
2° Passo

L'effetto di quanto illustrato in precedenza ridurrà la "porzione" del vettore sul quale continuare la ricerca. Infatti ora la "porzione" del vettore corrente su cui effettuare la ricerca, corrisponderà a quella con i valori dell'indice compresi tra:

primo ← centro + 1 (ossia 5 + 1 = 6) ed ultimo che resta inalterato (ossia = 9)

mentre da ciò deriva che il NUOVO elemento situato ora in posizione centrale avrà indice pari a:

centro ← (primo + ultimo) DIV 2 ossia centro ← (6 + 9) DIV 2 ossia centro ← 7



Anche in questa nuova "porzione" del vettore sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

1. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato.

Allora l'elemento ricercato appartiene alla nuova "porzione" del vettore e si trova in posizione centrale.
La RICERCA in questo caso DEVE ESSERE ARRESTATA
Nel nostro caso NO domanda: elemento = v[centro] ? ossia 8 = v[7] ? in quanto 8 = 12 E' FALSO

2. l'elemento da ricercare è più piccolo dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà destra della nuova "porzione" del vettore e l'ultimo elemento utile alla ricerca diventa quello immediatamente precedente all'elemento centrale (ossia ultimo← centro – 1)
Nel nostro caso SI domanda: elemento < v[centro] ? ossia 8 < v[7] ? in quanto 8 < 12 E' VERO
La RICERCA in questo caso DEVE CONTINUARE su una nuova porzione del "vettore" da specificare.

3. l'elemento da ricercare è più grande dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà sinistra della nuova "porzione" del vettore ed il primo elemento utile alla ricerca diventa quello immediatamente successivo all'elemento centrale
(ossia primo← centro + 1)
Nel nostro caso NO domanda: elemento > v[centro] ? ossia 8 > v[7] ? in quanto 8 > 12 E' FALSO

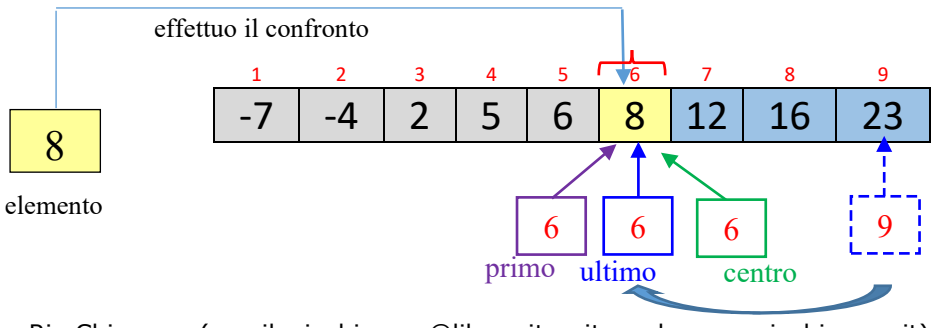
3° Passo

L'effetto di quanto illustrato in precedenza ridurrà ulteriormente la "porzione" del vettore sul quale continuare la ricerca. Infatti ora avremo gli indici:

primo che resta inalterato (ossia = 6) ed ultimo ← centro - 1 (ossia = 7 – 1 = 6)

mentre da ciò deriva che il NUOVO elemento situato ora in posizione centrale avrà indice pari a:

centro ← (primo + ultimo) DIV 2 ossia centro ← (6 + 6) DIV 2 ossia centro ← 6



Anche in questa nuova "porzione" del vettore sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

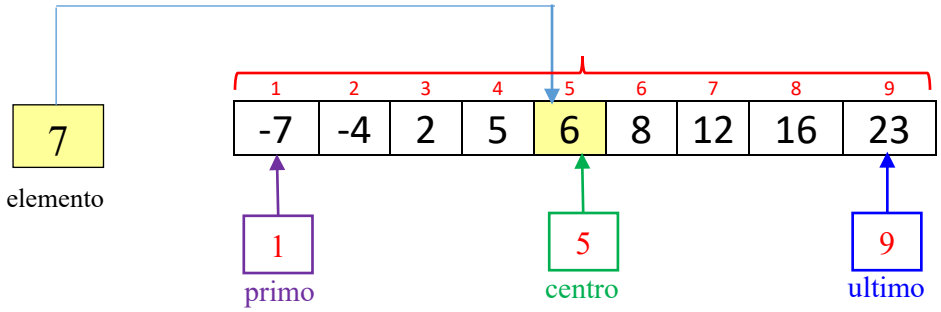
1. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato.

Allora l'elemento ricercato appartiene alla nuova "porzione" del vettore e si trova in posizione centrale.
La RICERCA in questo caso DEVE ESSERE ARRESTATA
Nel nostro caso SI domanda: elemento = v[centro] ? ossia 8 = v[6] ? in quanto 8 = 8 E' VERO
LA RICERCA DEVE ESSERE ARRESTATA PERCHE' L'ELEMENTO E STATO TROVATO ALL'INTERNO DEL VETTORE!

trovato ← VERO STOP

E QUINDI NON DEVO PIU' PROCEDERE CON LA VERIFICA DEGLI ALTRI DUE CASI

Esempio: Ricerca di un elemento in un vettore formato da n = 9 elementi interi ordinati in senso CRESCENTE



CASO 2
L'elemento da ricercare
NON E' PRESENTE
nel vettore **v**

1° Passo

All'inizio la "porzione" del vettore su cui effettuare la ricerca, corrisponderà a quella con i valori dell'indice compresi tra:

primo ← 1 ed ultimo ← 9

ne consegue che in tale ipotesi l'elemento situato in posizione centrale avrà indice:

centro ← (primo + ultimo) DIV 2 ossia centro ← (1 + 9) DIV 2 ossia centro ← 5

Dopo di ciò sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

1. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato.

Allora l'elemento ricercato appartiene al vettore e si trova in posizione centrale.
La RICERCA in questo caso DEVE ESSERE ARRESTATA
Nel nostro caso NO domanda: elemento = v[centro] ? ossia 7 = v[5] ? in quanto 7 = 6 E' FALSO

2. l'elemento da ricercare è più piccolo dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà destra del vettore e l'ultimo elemento del vettore utile alla ricerca diventa quello immediatamente precedente all'elemento centrale (ossia ultimo← centro – 1)
Nel nostro caso NO domanda: elemento < v[centro] ? ossia 7 < v[5] ? in quanto 7 < 6 E' FALSO

3. l'elemento da ricercare è più grande dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà sinistra del vettore ed il primo elemento del vettore utile alla ricerca diventa quello immediatamente successivo all'elemento centrale (ossia primo← centro + 1)
Nel nostro caso SI domanda: elemento > v[centro] ? ossia 7 > v[5] ? in quanto 7 > 6 E' VERO
La RICERCA in questo caso DEVE CONTINUARE su una nuova porzione del "vettore" da specificare.

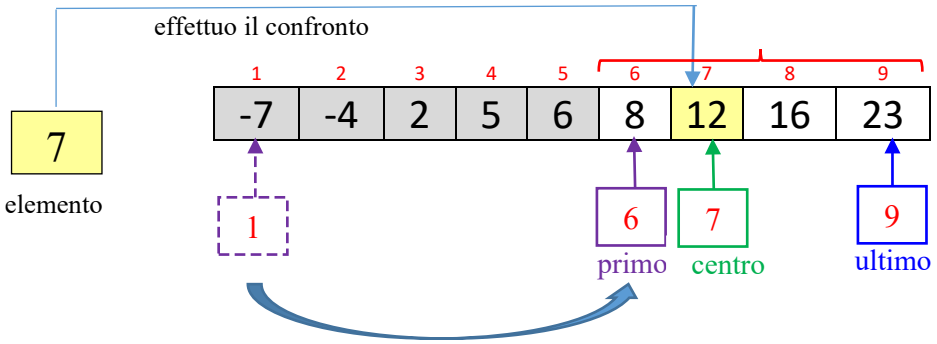
2° Passo

L'effetto di quanto illustrato in precedenza ridurrà la "porzione" del vettore sul quale continuare la ricerca. Infatti ora la "porzione" del vettore corrente su cui effettuare la ricerca, corrisponderà a quella con i valori dell'indice compresi tra:

primo ← centro + 1 (ossia 5 + 1 = 6) ed ultimo che resta inalterato (ossia = 9)

mentre da ciò deriva che il NUOVO elemento situato ora in posizione centrale avrà indice pari a:

centro ← (primo + ultimo) DIV 2 ossia centro ← (6 + 9) DIV 2 ossia centro ← 7



Anche in questa nuova "porzione" del vettore sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

1. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato.

Allora l'elemento ricercato appartiene alla nuova "porzione" del vettore e si trova in posizione centrale.
La RICERCA in questo caso DEVE ESSERE ARRESTATA
Nel nostro caso NO domanda: elemento = v[centro] ? ossia 7 = v[7] ? in quanto 7 = 12 E' FALSO

2. l'elemento da ricercare è più piccolo dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà destra della nuova "porzione" del vettore e l'ultimo elemento utile alla ricerca diventa quello immediatamente precedente all'elemento centrale (ossia ultimo← centro – 1)
Nel nostro caso SI domanda: elemento < v[centro] ? ossia 7 < v[7] ? in quanto 7 < 12 E' VERO
La RICERCA in questo caso DEVE CONTINUARE su una nuova porzione del "vettore" da specificare.

3. l'elemento da ricercare è più grande dell'elemento *centrale* fissato.

Si scartano tutti gli elementi della metà sinistra della nuova "porzione" del vettore ed il primo elemento utile alla ricerca diventa quello immediatamente successivo all'elemento centrale
(ossia primo← centro + 1)
Nel nostro caso NO domanda: elemento > v[centro] ? ossia 7 > v[7] ? in quanto 7 > 12 E' FALSO

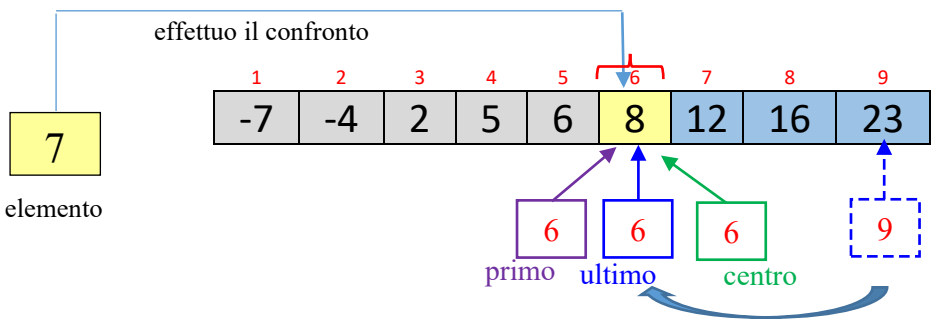
3° Passo

L'effetto di quanto illustrato in precedenza ridurrà ulteriormente la "porzione" del vettore sul quale continuare la ricerca. Infatti ora avremo gli indici:

primo che resta inalterato (ossia = 6) ed ultimo ← centro - 1 (ossia = 7 – 1 = 6)

mentre da ciò deriva che il NUOVO elemento situato ora in posizione centrale avrà indice pari a:

centro ← (primo + ultimo) DIV 2 ossia centro ← (6 + 6) DIV 2 ossia centro ← 6



Anche in questa nuova "porzione" del vettore sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

1. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato.

Allora l'elemento ricercato appartiene alla nuova "porzione" del vettore e si trova in posizione centrale.
La RICERCA in questo caso DEVE ESSERE ARRESTATA
Nel nostro caso NO domanda: elemento = v[centro] ? ossia 7 = v[6] ? in quanto 7 = 8 E' FALSO
2 l'elemento da ricercare è più piccolo dell'elemento *centrale* fissato.
Si scartano tutti gli elementi della metà destra della nuova "porzione" del vettore e l'ultimo elemento utile alla ricerca diventa quello immediatamente precedente all'elemento centrale (ossia ultimo← centro – 1)
Nel nostro caso SI domanda: elemento < v[centro] ? ossia 7 < v[6] ? in quanto 7 < 8 E' VERO
MA LA NOSTRA RICERCA DOVRA' COMUNQUE ESSERE INTERROTTA IN QUANTO AVREMMO gli indici:

primo che resta inalterato (ossia = 6) ed ultimo ← centro - 1 (ossia = 6 – 1 = 5)

ossia avremmo ottenuto il seguente assurdo primo > ultimo 

LA RICERCA VIENE ARRESTATA PERCHE' NON E' PIU' POSSIBILE AVERE UNA "PORZIONE" DI VETTORE A DISPOSIZIONE

Quindi **ANCHE** l'algoritmo di **ricerca binaria o dicotomica** dovrà basarsi su una **ISTRUZIONE ITERATIVA NON ENUMERATIVA** (perché non so a priori se troverò o meno l'elemento all'interno del vettore) governato dal seguente enunciato composto:

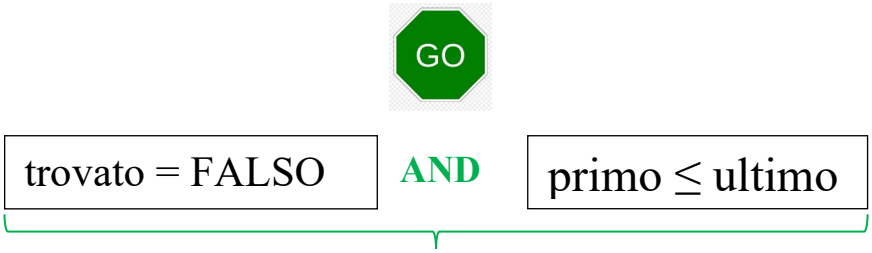


TABELLE DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
v	**ARRAY**[MAXDIM] **DI INT**	**STATICA**	**TUTTI**	Vettore o array di interi da acquisire da tastiera
n	**INT**	**STATICA**	$1 \leq$ **n** $\leq$ MAXDIM ossia (**n** ≥ 1) AND (**n** $\leq$ MAXDIM)	Dimensione effettiva del vettore immesso da tastiera
elemento	**INT**	**STATICA**	**TUTTI**	Elemento da ricercare nel vettore

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
trovato	**BOOL**	**STATICA**	**VERO o FALSO**	Variabile che indica l'esito della ricerca
posizione	**INT**	**STATICA**	0 $\leq$ posizione $\leq$ n ossia (**posizione** ≥ 0) **AND** (**posizione** $\leq$ n)	Posizione eventuale dell'elemento trovato nel vettore v

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
MAXDIM	**INT**	**STATICA**	**10**	Massimo numero di elementi gestibile dal vettore monodimensionale
primo	**INT**	**STATICA**	$1 \leq$ **primo** $\leq$ ultimo ossia (**primo** ≥ 1) **AND** (**primo** $\leq$ ultimo)	Indice del primo elemento del sottovettore di v da considerare
ultimo	**INT**	**STATICA**	primo $\leq$ **ultimo** $\leq$ n ossia (**ultimo** $\geq$ primo) **AND** (**ultimo** $\leq$ n)	Indice dell'ultimo elemento del sottovettore di v da considerare
centro	**INT**	**STATICA**	primo $\leq$ **centro** $\leq$ ultimo ossia (**centro** $\geq$ primo) **AND** (**centro** $\leq$ ultimo)	Indice dell'elemento centrale sottovettore di v da considerare calcolato **centro ← (primo + ultimo) DIV 2**

N.B. Questa tabella va aggiornata aggiungendo eventuali altre variabili di lavoro a seconda dell'algoritmo di ordinamento (se necessario INGENUO o A BOLLE) che verrà utilizzato.

```
ALGORITMO RicercaBinaria
MAXDIM 10
PROCEDURA main ( )
v: ARRAY [MAXDIM] DI INT
primo, ultimo, centro: INT
[posizione], n, i, elemento :INT
trovato: BOOL

INIZIO
/* leggo la dimensione del vettore da caricare (vedi esercizio precedente)*/
….
/* carico gli elementi nel vettore(vedi esercizio precedente) */
….
/* ordino gli elementi nel vettore in uno dei modi studiati (vedi esercizi precedente) */
/* N.B. Occorerrà aggiornare di conseguenza le tabelle dei dati */
….
/* effettuo la ricerca binaria dell'elemento all'interno del vettore */
[posizione ← 0]                          /* inizializzo, se richiesta, la posizione */
primo ← 1
ultimo ← n
trovato ← FALSO

MENTRE ((trovato = FALSO) AND (primo ≤ ultimo)) ESEGUI
  centro ← (primo + ultimo) DIV 2
  SE ( elemento = v[centro])             /* N.B. Vanno bene anche le seguenti condizioni logiche */
    ALLORA                                   /* (NOT trovato) oppure (NOT trovato = VERO) */
      trovato ← VERO
      [posizione ← centro]                   /* conservo, se richiesta, la posizione dell'elemento */
    ALTRIMENTI
      SE (elemento < v[centro])
        ALLORA
          ultimo ← centro – 1
        ALTRIMENTI
          primo ← centro + 1
      FINE SE
  FINE SE
FINE MENTRE

/* comunica l'esito all'utente */
Scrivi (trovato)
SE (trovato = VERO)
  ALLORA
    Scrivi ("L'elemento è stato trovato")
    [Scrivi(posizione)]                    /* mostro a video, se richiesta, la posizione dell'elemento*/
  ALTRIMENTI
    Scrivi ("L'elemento non è stato trovato")
FINE SE

FINE
```

OSSERVAZIONI

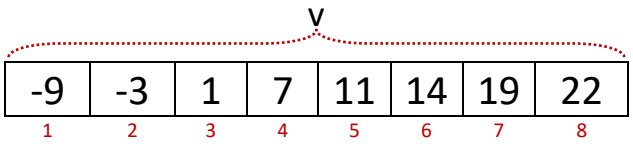
Dimostreremo più avanti che l'algoritmo di **ricerca binaria o dicotomica** di un elemento in un un vettore o array monodimensionale effettua $\log_2(n)$ confronti **nel caso pessimo** (ossia quello in cui l'elemento da ricercare fornito non è presente al suo interno), risultando più efficiente dell'algoritmo (equivalente) di **ricerca sequenziale** poiché, grazie all'analisi matematica, si potrà dimostrere che:

$$\log_2(n) < n$$

qualunque sia il la **dimensione del vettore n**

RICERCA SEQUENZIALE vs RICERCA BINARIA o DICOTOMICA

Ipotizziamo di avere assegnato il seguente **vettore o array monodimensionale** v ordinato in senso crescente e con dimensione **n = 8**



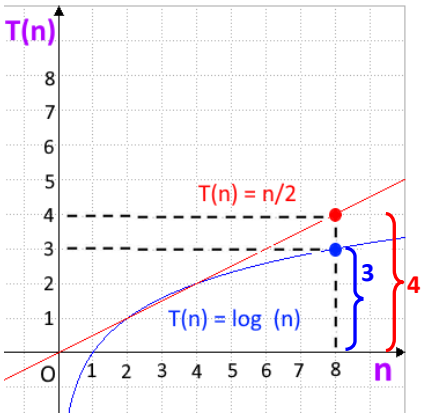
Si può ossevare che:

- applicando il processo risolutivo dettagliato nell'algoritmo di **ricerca sequenziale,** per poter trovare l'elemento assegnato:
  - ✓ nel **CASO FAVOREVOLE** (ossia elemento al primo posto, es **-9**) si effettuerà esattamente **1** confronto;
  - ✓ nel **CASO PESSIMO** (ossia elemento all'ultimo posto, es. **22**) si effettueranno esattamente **8** (ossia **n**) confronti;
  - ✓ nel **CASO MEDIO** si effettueranno un numero di confronti pari a (**CASO FAVOREVOLE** + **CASO PESSIMO**) **/ 2** (ossia pari ad **(1 + n) / 2** ossia 5 che può approssimarsi a **4** ossia a **n/2**);
- applicando il processo risolutivo dettagliato nell'algoritmo di **ricerca binaria o dicotomica**, per poter trovare l'elemento assegnato:
  - ✓ nel **CASO FAVOREVOLE** (ossia elemento nel posto centrale, es **7**) si effettuerà esattamente **1** confronto;
  - ✓ nel **CASO PESSIMO** (ossia elemento all'ultimo posto, es **22**) si effettueranno esattamente **3** (ossia **$\log_2(n)$**) confronti;
  - ✓ nel **CASO MEDIO** si effettueranno un numero di confronti pari a (**CASO FAVOREVOLE** + **CASO PESSIMO**) **/ 2** (ossia pari ad **(1 + $\log_2(n)$) / 2** ossia 4 che può approssimarsi a **3** ossia a **$\log_2(n)$**);

Si deduce che l'algoritmo di ricerca binaria o dicotomica, quando il numero di elementi **n** è pari a **8**, sia più efficiente rispetto all'algoritmo (equivalente) di ricerca sequenziale.

In realtà quando studieremo più avanti **l'analisi computazionale degli algoritmi**, dimostreremo che, qualunque sia il valore assunto da **n**, l'algoritmo di ricerca binaria o dicotomica risulterà essere sempre più efficiente rispetto a quello di ricerca sequenziale.

Diamo un'occhiata al seguente grafico:



T(n) indica la **complessità computazionale** di un algoritmo (o complessità) ossia la ***funzione matematica*** che indica la relazione esistente tra il numero di operazioni effettuate da un algoritmo (programma) e la dimensione **n** dei dati ricevuti in input.

$T_{\text{Ricerca Sequenziale}}(n) = n/2$

$T_{\text{Ricerca Binaria}}(n) = \log_2(n)$

Confrontanto "empiricamente" i due grafici si nota come qualunque sia il valore della dimensione **n** del problema (ossia il numero di elementi del vettore), i valori di complessità calcolati con le rispettive funzioni matematiche, differiscano sempre.

Ciò che emerge è che, a parità di dimensione del problema (ossia a parità del numero **n** di elementi contenuti nel vettore) ed al crescere dello stesso, risulterà sempre verificata la seguente disequazione:

$$T_{\text{Ricerca Binaria}}(n) < T_{\text{Ricerca Sequenziale}}(n)$$

Da tutto quanto illustrato consegue che, in caso di vettori già ordinati, l'algoritmo di ricerca binaria o dicotomica risulta essere, al crescere della dimensione del problema, più efficiente rispetto all'algoritmo (equivalente) di ricerca sequenziale.

IL TIPO DI DATI STRUTTURATO MATRICE o ARRAY BIDIMENSIONALE

Per risolvere particolari tipi di problemi si ricorre a strutture dati più complesse del vettore monodimensionale che permette soltanto la memorizzazione di insiemi lineari di informazioni.

*Esempio: Se volessimo memorizzare tutti i voti ottenuti da un gruppo di alunni nelle varie materie seguite invece che una serie di **vettori paralleli** da gestire attentamente da programma (uno per memorizzare gli alunni ed uno per ciascuna materia di cui memorizzare i voti), saremmo interessati ad utilizzare una tabella che avesse tante righe quanti sono gli alunni e tante colonne quante sono le materie, dove l'incrocio tra riga e colonna conterrebbe la votazione di un determinato alunno in una determinata materia.*

Questa struttura dati prende il nome di **matrice o array bidimensionale.**

DEFINIZIONE: Si definisce **MATRICE o ARRAY BIDIMENSIONALE ad n righe ed m colonne** o più brevemente **MATRICE o ARRAY n * m** una tabella formata da **n * m elementi omogenei**, disposti su **n** linee orizzontali (**righe**) ed **m** linee verticali (**colonne**).

Per indicare la matrice nel suo complesso si usano le lettere in maiuscolo (A, B, etc).
Per indicare l'elemento generico di una matrice **A** si indica con il simbolo $a_{i\,j}$ dove **i** è l'indice di riga e **j** è l'indice di colonna.

Rappresentazione ESTESA di una matrice:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & \dots & a_{2m} \\ a_{31} & a_{32} & a_{33} & \dots & \dots & a_{3m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & a_{ij} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & \dots & a_{nm} \end{pmatrix}$$

i-esima riga

j-esima colonna

Rappresentazione COMPATTA di una matrice:

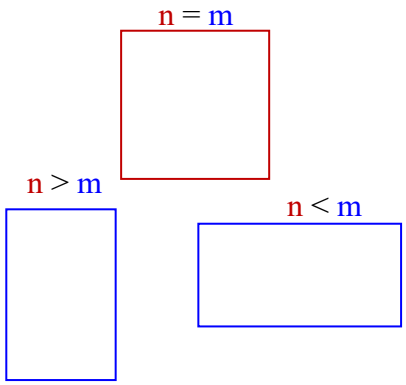
$$A = (a_{ij}) \quad \text{con } i = 1 \text{ a } n \quad \text{e} \quad \text{con } j = 1 \text{ a } m$$

Nella PSEUDOCODIFICA per dichiarare all'interno di un algoritmo una variabile di questo tipo occorre usare la seguente pseudoistruzione:

<NomeMatrice> **: ARRAY [**<NumRighe>**] [**<NumColonne>**] DI** <TipoElemento>

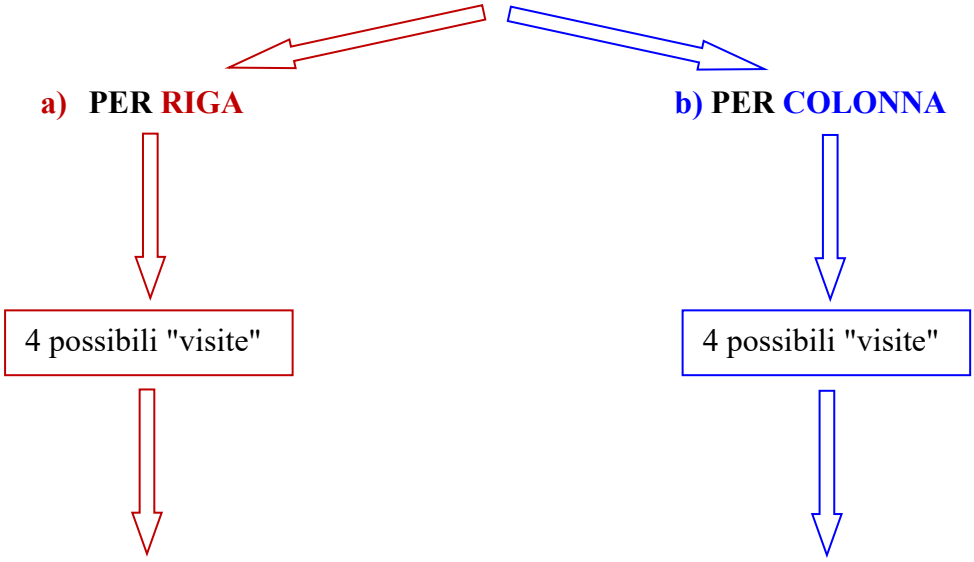
Le matrici si dividono in:
- **matrici quadrate**: nel caso in cui il numero di righe sia uguale al numero di colonne (n = m)
In questo caso la forma teorica che essa assumerà può essere sintetizzata da un **quadrato**

matrici rettangolari: nel caso in cui il numero di righe sia diverso al numero di colonne (n ≠ m)
In questo caso la forma teorica che essa assumerà può essere sintetizzata da un **rettangolo**



Modalità di **navigazione** o "**visita**" degli elementi di una matrice qualsiasi **n righe** ed **m colonne**

Innanzitutto distinguiamo se vogliamo **navigare** o "**visitare**" la matrice **n** * **m**:



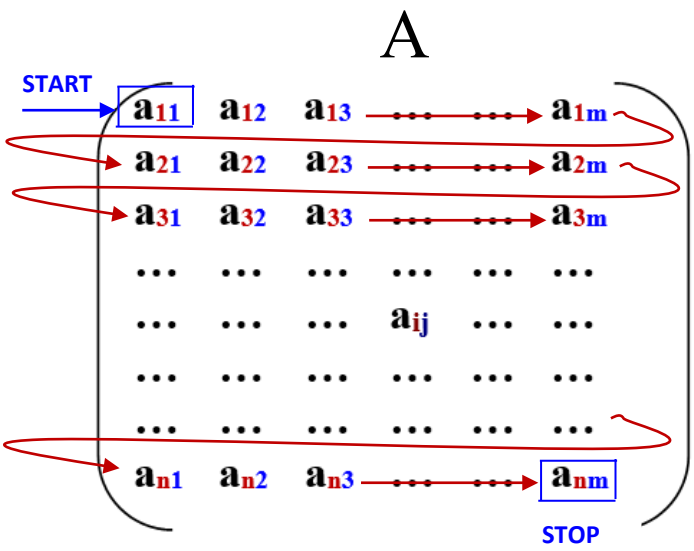

```
PER i ← 1 A n ESEGUI                    /* RIGA crescente */
  PER j ← 1 A m ESEGUI                  /* COLONNA crescente */
    <B1>
    j ← j + 1
  FINE PER
  i ← i + 1
FINE PER

PER i ← 1 A n ESEGUI                    /* RIGA crescente */
  PER j ← m INDIETRO A 1 ESEGUI         /* COLONNA decrescente */
    <B1>
    j ← j - 1
  FINE PER
  i ← i + 1
FINE PER

PER i ← n INDIETRO A 1 ESEGUI           /* RIGA decrescente */
  PER j ← 1 A m ESEGUI                  /* COLONNA crescente */
    <B1>
    j ← j + 1
  FINE PER
  i ← i - 1
FINE PER

PER i ← n INDIETRO A 1 ESEGUI           /* RIGA decrescente */
  PER j ← m INDIETRO A 1 ESEGUI         /* COLONNA decrescente */
    <B1>
    j ← j - 1
  FINE PER
  i ← i - 1
FINE PER
```

```
PER j ← 1 A m ESEGUI                    /* COLONNA crescente */
  PER i ← 1 A n ESEGUI                  /* RIGA crescente */
    <B1>
    i ← i + 1
  FINE PER
  j ← j + 1
FINE PER

PER j ← 1 A m ESEGUI                    /* COLONNA crescente */
  PER i ← n INDIETRO A 1 ESEGUI         /* RIGA decrescente */
    <B1>
    i ← i - 1
  FINE PER
  j ← j + 1
FINE PER

PER j ← m INDIETRO A 1 ESEGUI           /* COLONNA decrescente */
  PER i ← 1 A n ESEGUI                  /* RIGA crescente */
    <B1>
    i ← i + 1
  FINE PER
  j ← j - 1
FINE PER

PER j ← m INDIETRO A 1 ESEGUI           /* COLONNA decrescente */
  PER i ← n INDIETRO A 1 ESEGUI         /* RIGA decrescente */
    <B1>
    i ← i - 1
  FINE PER
  j ← j - 1
FINE PER
```

Aiutandoci con la tabella di traccia (un estratto), controlliamo l'ordine di "**visita**" degli elementi della matrice **A** di tipo n x m ipotizzando il seguente doppio ciclo PER (quello "classico"):

```
/* RIGA crescente */
PER i ← 1 A n ESEGUI
   /* COLONNA crescente
   PER j ← 1 A m ESEGUI
     <B1>
     j ← j + 1
   FINE PER
   i ← i + 1
FINE PER
```

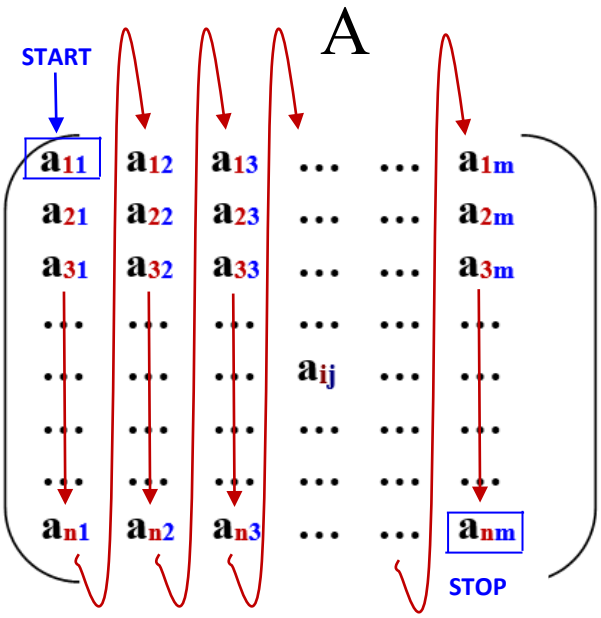
i	j	a_{ij}
1	1	a_{11}
1	2	a_{12}
…	…	…
1	m	a_{1m}
2	1	a_{21}
2	2	a_{22}
…	…	…
2	m	a_{2m}
…	…	…
n	1	a_{n1}
n	2	a_{n2}
…	…	
n	m	a_{nm}



Aiutandoci sempre con la tabella di traccia (un estratto), controlliamo un altra possibile modalità di "**visita**" degli elementi della matrice **A** di tipo n x m ipotizzando il seguente doppio ciclo PER:

```
/* COLONNA crescente */
PER j ← 1 A m ESEGUI
   /* RIGA crescente */
   PER i ← 1 A n ESEGUI
     <B1>
     i ← i + 1
   FINE PER
   j ← j + 1
FINE PER
```

i	j	a_{ij}
1	1	a_{11}
2	1	a_{21}
…	…	…
n	1	a_{n1}
1	2	a_{12}
2	2	a_{22}
…	…	…
n	2	a_{n2}
…	…	…
1	m	a_{1m}
2	m	a_{2m}
…	…	…
n	m	a_{nm}



Le operazioni di base sulle matrici sono le stesse di quelle già esaminate per i vettori (quindi caricamento, visualizzazione, ordinamento, ricerca).

Qui ci limitiamo ad illustrare quelle relative al **CARICAMENTO** ed alla **VISUALIZZAZIONE** di una **matrice rettangolare n** x **m** e di una matrice quadrata n x n

N.B. Anche nel caso delle matrici, useremo un criterio di FALSA DINAMICITA' analogo a quello utilizzato nel caso di vettore o array monodimensionale avendo però l'accortezza di utilizzare invece di due costanti differenti (una per il numero di righe ed una per il numero di colonne) una soltanto, essendo sempre possibile dichiarare una matrice rettangolare anche all'interno di una zona di memoria "quadrata".

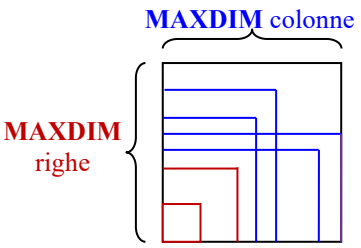
ALGORITMO CaricaVisualizzaMatrice**Rettangolare**

/ N.B. Si usa un unica costante MAXDIM per gestire la FALSA DINAMICITA'anche nel caso degli array bidimensionali (ossia delle matrici). La decisione di allocare uno spazio di memoria "quadrato" non impedisce di utilizzare al suo interno una parte "rettangolare" */*

MAXDIM 10 /* Max numero di righe e di colonne */

PROCEDURA main ( )

matr: **ARRAY** [**MAXDIM**] [**MAXDIM**] **DI INT**
n, m: **INT**
i, j: **INT**



INIZIO

/ leggo il numero di righe e di colonne della matrice rettangolare che si desidera caricare rispettando in entrambi i casi i vincoli imposti da* MAXDIM **/*

RIPETI
  Leggi (n)
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)

RIPETI
  Leggi (m)
FINCHE' (m ≥ 1) AND (m ≤ MAXDIM) **AND (n ≠ m)** /* Così si escludono le matrici quadrate */

/ carico per riga e per colonna crescenti gli elementi nella matrice rettangolare */*

PER i ← 1 **A** n **ESEGUI** /* RIGA crescente */
  PER j ← 1 **A** m **ESEGUI** /* COLONNA crescente */
    Leggi (matr[i][j])
    j ← j + 1
  FINE PER
  i ← i + 1
FINE PER

/ visualizzo per riga e per colonna crescenti gli elementi della matrice rettangolare */*

PER i ← 1 **A** n **ESEGUI** /* RIGA crescente */
  PER j ← 1 **A** m **ESEGUI** /* COLONNA crescente */
    Scrivi (matr[i][j])
    j ← j + 1
  FINE PER
  i ← i + 1
FINE PER

FINE

```
ALGORITMO CaricaVisualizzaMatriceQuadrata
MAXDIM 10                          /* Max numero di righe e di colonne */
PROCEDURA main ( )
matr: ARRAY [MAXDIM] [MAXDIM] DI INT
n: INT                             /* Numero di righe = Numero di colonne */
i, j: INT
INIZIO
/* leggo il numero di righe e di colonne della matrice quadrata che si desidera caricare rispettando
i vincoli imposti da MAXDIM. Non ha senso usare un'altra variabile per le colonne visto che
trattiamo una matrice quadrata */
RIPETI
  Leggi (n)
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
/* carico per riga e per colonna crescenti gli elementi nella matrice quadrata */
PER i ← 1 A n ESEGUI               /* RIGA crescente */
  PER j ← 1 A n ESEGUI             /* COLONNA crescente */
    Leggi (matr[i][j])
    j ← j + 1
   FINE PER
  i ← i + 1
FINE PER
/* visualizzo per riga e per colonna crescenti gli elementi della matrice quadrata */
PER i ← 1 A n ESEGUI               /* RIGA crescente */
  PER j ← 1 A n ESEGUI             /* COLONNA crescente */
    Scrivi (matr[i][j])
    j ← j + 1
   FINE PER
  i ← i + 1
FINE PER
FINE
```

IL TIPO DI DATI STRUTTURATO RECORD

Nasce spesso nella realtà l'esigenza di dover trattare informazioni di tipo diverso relative ad un stesso oggetto preso in esame.

Esempio 1: Pensiamo ad una normale fattura nella quale sono indicate tra l'altro in genere:

- *il numero della fattura (di tipo numerico intero);*
- *la data di emissione (di tipo stringa alfanumerica);*
- *l'importo dovuto (di tipo numerico decimale)*

E'evidente che è possibile risolvere il problema relativo alla memorizzazione di un codesto set di informazioni utilizzando n variabili, ciascuna coerente con il tipo di dati da rappresentare, ma l'utilizzo di tali variabili non indicherà in alcun modo durante l'esecuzione che i dati si rifersicono ad uno stesso oggetto (nel nostro caso la stessa fattura).

Esempio 2: Pensiamo ad libro caratterizzato tra l'altro in genere da:

- *codice ISBN dell'opera (di tipo stringa alfanumerica);*
- *il titolo dell'opera (di tipo stringa alfanumerica);*
- *l'autore dell'opera (di tipo stringa alfanumerica);*
- *la casa editrice (di tipo stringa alfanumerica);*
- *il prezzo di copertina (di tipo numerico decimale);*
- *il numero di pagine (di tipo numerico intero)*

Potremmo utilizzare n vettori paralleli al posto delle n variabili separate, ma comunque non risolveremmo il problema di riferirci semplicemente e globalmente all'oggetto libro ed ai suoi dati nel suo complesso.

DEFINIZIONE: Un **record** o **registrazione** è una struttura di dati composta da un insieme finito di elementi **eterogenei** detti **campi**. I campi sono tra loro logicamente connessi e corrispondono agli **attributi**.
Ogni campo accoglie **un solo valore** per ciascun attributo.
L'insieme dei valori scelti per tutti gli attributi che formano il record, individua una **istanza** del record.

Esempio 2: Se pensiamo ad libro specifico allora avremo a che fare con le seguenti istanze (ennuple):

"3333-567-3456", "La fattoria degli animali", "George Orwell", "Zanichelli", 12.75, 245

"8976-134-6578", "Il nome della rosa", "Umberto Eco", "Mondadori", 11.25, 441

Il **record** è un tipo di dati predefinito e viene messo a disposizione da quasi tutti i linguaggi di programmazione basati sul paradigma imperativo.

Un record è caratterizzato da **un nome** che lo identifica e che permette di riferirsi ad esso nella sua globalità.

I **campi** che lo compongono sono caratterizzati da un **nome** e dal **tipo di dato** che possono contenere. Possono essere indifferentemente **di tipo semplice** o **di tipo strutturato**: un singolo campo di un record può infatti a sua volta essere a sua volta un *array* oppure un *record*.

Viene definita **struttura di un record** la definizione dei campi che compongono il record stesso

Per definire la struttura di un record possiamo servirci di **due metodi di rappresentazione**:

a) un **metodo di rappresentazione tabellare** consistente nel predisporre una tabella che riporta per ogni campo *il numero di campo, in nome del campo, il tipo del campo, la lunghezza del campo, la descrizione e l'eventuale appartenenza del campo ad un sottorecord*.

Tale metodo consente di definire il cosiddetto **tracciato record** attraverso l'utilizzo della seguente tabella:

Esempio 2:

Numero	Nome Campo	Tipo Campo	Lunghezza[1]	Descrizione	FlagSubRec[2]
1	Isbn	ARRAY DI CHAR	10	Codice ISBN del libro	
2	Titolo	ARRAY DI CHAR	100	Titolo del libro	
3	Autore	ARRAY DI CHAR	50	Autore del libro	
4	CasaEditrice	ARRAY DI CHAR	50	Casa Editrice del libro	
5	Prezzo	REAL	5,2	Prezzo del libro	
6	Pagine	INT	4	Pagine del libro	

1. In caso di valori numerici decimali scrivere **5,2** vuol dire **5** cifre intere in totale di cui **2** decimali (max 999.99)
In caso di valori numerici interi scrivere **4** vuol dire prevedere un massimo di **4 cifre significative** (max 9999)

2. Se nel campo FlagSubRec viene posta una X vuol dire che quel campo deve essere considerato parte di un altro record (vedi SOTTORECORD)

b) Un **metodo di rappresentazione tramite pseudoistruzioni** molto vicino a quello utilizzato dagli stessi linguaggi di programmazione che permette di dichiarare una variabile record.

Nella nostra **PSEUDOCODIFICA** per poter dichiarare **una variabile** con una **certa struttura record** faremo così:

<Nome Variabile Record> **: RECORD**
<Nome Campo1>: <Tipo Campo 1>
<Nome Campo2>: <Tipo Campo 2>
……..
<Nome CampoN>: <Tipo Campo N>
FINE RECORD

Esempio 3:

lib : **RECORD**
Isbn: ARRAY[10] DI CHAR
Titolo: ARRAY[100] DI CHAR
Autore: ARRAY[50] DI CHAR
CasaEditrice: ARRAY[50] DI CHAR
Prezzo: REAL
Pagine: INT
FINE RECORD

N.B.

Per i tipi di dati strutturati come i record (e gli array) è preferibile utilizzare l'istruzione della pseudocodifica TIPO che permette di definire nuovi tipi utente o di ridefinire quelli già esistenti a partire da tipi di dato base.
Questo al fine di agevolare la messa a disposizione di un tipo di dato comune utilizzabile anche da altri utenti e programmi.

Per poter fare ciò si deve utilizzare una pseudoistruzione ad hoc in grado di creare questi nuovi tipi di dato (es. record) la cui sintassi sarà la seguente:

TIPO *<mio_tipo>* **=** *<tipo di dato_base>*

*Esempio 4:Secondo quanto dettio in precedenza, **nell'ambiente GLOBALE** (n.b. quindi a disposizione non solo del mio programma ma anche di tutti gli eventuali sottoprogrammi in cui eventualmente lo suddividerò) sarà possibile definire il seguente **nuovo TIPO DI DATO UTENTE***

....

TIPO Libro = RECORD

Isbn: ARRAY[10] DI CHAR
Titolo: ARRAY[100] DI CHAR
Autore: ARRAY[50] DI CHAR
CasaEditrice: ARRAY[50] DI CHAR
Prezzo: REAL
Pagine: INT

FINE RECORD

....

*Ora **nell'ambiente LOCALE** potrà di conseguenza essere definita la seguente variabile **lib** di tipo **Libro**:*

....

lib** : **Libro

....

Operazioni ammesse sui record

Le operazioni definite sul tipo di dato strutturato record vengono distinte in **operazioni NON atomiche** ed **operazioni atomiche** a seconda che coinvolgono:

- un singolo campo del record alla volta
  oppure
- l'intera struttura del record in un'unica soluzione.

Per i record pertanto distingueremo:

- le **operazioni sul singolo campo** (OPERAZIONI **NON** ATOMICHE)

- le **operazioni sull'intero record** (OPERAZIONI ATOMICHE);

Le **operazioni** che coinvolgono **un singolo campo** alla volta, secondo le specifiche consentite dal tipo di dato del campo (operazioni **NON** ATOMICHE), sono:

- Il **caricamento o LETTURA** del record (che deve essere effettuata campo per campo);
- La **visualizzazione** o **STAMPA** del record (che deve essere effettuata campo per campo);

Il caricamento del record è una operazione che permette di assegnare a ciascun campo di un record, un opportuno valore coerente con il tipo di dati posseduto.

La visualizzazione del record è una operazione che permette di mostrare a video il valore di ciascun campo di un record

Un'**operazione** che coinvolge **l'intero record** visto come un unico oggetto (ossia OPERAZIONE ATOMICA) è:

- **L'ASSEGNAZIONE tra record dello stesso tipo**

L'assegnazione tra record è una operazione che permette di assegnare IN UNA VOLTA SOLA il valore di tutti i campi di una variabile di un certo tipo record, ad un'altra variabile che possiede la stessa struttura o tracciato (N.B. quindi le **due variabili devono avere lo stesso tipo record**).

Esempio 5: Supponiamo ora di avere dichiarato la seguente variabile **lib** *di tipo* **Libro** *ossia si un record con la seguente struttura o tracciato (vedi Esempio 2)*

.....

TIPO Libro = RECORD

Isbn:	ARRAY[10] DI CHAR
Titolo:	ARRAY[100] DI CHAR
Autore:	ARRAY[50] DI CHAR
CasaEditrice:	ARRAY[50] DI CHAR
Prezzo:	REAL
Pagine:	INT

FINE RECORD

.....

lib : **Libro**

In PSEUDOCODIFICA per potere accedere ad un campo specifico di un record, occorre riferirsi a lui attraverso la seguente notazione chiamata anche **DOT NOTATION**:

<Nome Variabile Record>.<Nome Campo>

Quindi per ***leggere*** *il valore deil campo* ***Titolo*** *del record di nome* **lib** *con struttura record di tipo* **Libro**, *l'istruzione corretta sarà:*

Leggi (lib.Titolo)

Invece per ***visualizzare*** *il valore del campo* ***Prezzo*** *sempre del record di nome* **lib** *con struttura record di tipo* **Libro**, *l'istruzione corretta è:*

Scrivi (lib.Prezzo)

Esempio 6: Nel medesimo caso del tipo record **Libro** *in caso avessimo dichiarato due variabili record di quel tipo ossia*

lib1, **lib2** : **Libro**

l'istruzione **assolutamente lecita** **lib2 ← lib1**

assegnerà alla variabile **lib2** il contenuto della variabile **lib1** ossia memorizzerà in tutti i campi della variabile *lib2* i singoli campi della variabile *lib1*.

Quindi l' istruzione **lib2 ← lib1** equivale alla seguente sequenza di 6 assegnazioni **ovviamente meno efficiente**:

lib2.Isbn	← *lib1. Isbn*
lib2.Titolo	← *lib1. Titolo*
lib2.Autore	← *lib1. Autore*
lib2.CasaEditrice	← *lib1. CasaEditrice*
lib2.Prezzo	← *lib1.Prezzo*
lib2.Pagine	← *lib1. Pagine*

N.B. In alcuni linguaggi di programmazione è consentita una ulteriore operazione ossia il CONFRONTO tra record dello stesso tipo (in particolare l'UGUAGLIANZA ' = ' e la DISUGUAGLIANZA '≠').
Nei linguaggi per i quali tale operazione non esistesse, si potrebbe facilmente simulare attraverso un apposito sottoprogramma.

Esempio 6: Supponiamo ora di volere leggere (controllando la validità dei singoli campi) e stampare i dati contenuti in una variabile **lib** *di tipo* **Libro** *ossia di un record la cui struttura è stata descritta in precedenza (vedi Esempio 2)*

Innanzitutto inizieremo la fase di ***PROGETTAZIONE*** *individuando le tabelle dei dati (INPUT, OUTPUT, LAVORO o ELABORAZIONE) che sono coinvolti nella risoluzione del problema assegnato.*

TABELLE DEI DATI: PROCEDURA main()

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA main())				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
lib	**Libro**	**STATICA**	Vedi tracciato record in forma tabellare dettagliato in precedenza	Variabile record di tipo **Libro** (Vedi tracciato record in forma tabellare dettagliato in precedenza)

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA main())				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**
lib	**Libro**	**STATICA**	Vedi tracciato record in forma tabellare dettagliato in precedenza	Variabile record di tipo **Libro** (Vedi tracciato record in forma tabellare dettagliato in precedenza)

DATI DI ELABORAZIONE o DI LAVORO DEL PROBLEMA PRINCIPALE (PROCEDURA main())				
Nome variabile oppure **nome costante**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

*Proseguiamo la fase di **PROGETTAZIONE** utilizzando la **PSEUDOCODIFICA** per descrivere le istruzioni dell'ALGORITMO individuato:*

PSEUDODODIFICA

```
ALGORITMO Leggi_Stampa_RECORD
TIPO Libro =  RECORD
                    Isbn : ARRAY[10] DI CHAR
                    Titolo : ARRAY[100] DI CHAR
                    Autore : ARRAY[50] DI CHAR
                    CasaEditrice : ARRAY[50] DI CHAR
                    Prezzo : REAL
                    Pagine : INT
              FINE RECORD
PROCEDURA main( )
lib: Libro
INIZIO
/* leggo il record lib campo x campo (operazione NON ATOMICA) effettuando tutti i controlli */
RIPETI
   Leggi (lib.Isbn)
FINCHE' (Lunghezza(lib.Isbn) ≠ 0) AND (Lunghezza(lib.Isbn) ≤ 10)
RIPETI
   Leggi (lib.Titolo)
FINCHE' (Lunghezza(lib.Titolo) ≠ 0) AND (Lunghezza(lib.Titolo) ≤ 100)
RIPETI
   Leggi (lib.Autore)
FINCHE' (Lunghezza(lib.Autore) ≠ 0) AND (Lunghezza(lib.Autore) ≤  50)
RIPETI
   Leggi (lib.CasaEditrice)
FINCHE' (Lunghezza(lib.CasaEditrice) ≠ 0) AND (Lunghezza(lib.CasaEditrice) ≤  50)
RIPETI
   Leggi (lib.Prezzo)
FINCHE' (lib.Prezzo) > 0) AND (lib.Prezzo) ≤  999.99)
RIPETI
   Leggi (lib.Pagine)
FINCHE' (lib.Pagine) > 0) AND (lib.Pagine) ≤  9999)
/* visualizzo il record lib campo x campo (operazione NON ATOMICA)*/
Scrivi (lib.Isbn)
Scrivi (lib.Titolo)
Scrivi (lib.Autore)
Scrivi (lib.CasaEditrice)
Scrivi (lib.Prezzo)
Scrivi (lib.Pagine)
FINE
```

*N.B. Per completare la fase di **PROGETTAZIONE** potrebbe anche essere richiesta la descrizione dell'algoritmo individuato, attraverso la produzione di un **FLOWCHART** ossia utilizzando l'altro linguaggio formale possibile per descrivere gli algoritmi (per comodità qui verrà OMESSO)*

*Ora possiamo affrontare la fase di **PROGRAMMAZIONE** vera e propria (o IMPLEMENTAZIONE) traducendo nel **linguaggio di programmazione di alto livello C** tutte le istruzioni contenute nell'algoritmo descritto in precedenza tramite PSEUDOCODIFICA.*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//definizione del nuovo tipo record utente Libro
typedef struct
   {
   char Isbn[10+1];
   char Titolo[100+1];
   char Autore [50+1];
   char CasaEditrice[50+1];
   float Prezzo;
   int Pagine;
   } Libro;

int main(int argc, char *argv[])
{
//Dichiarazione di una variabile record di tipo Libro
Libro lib;

// Leggo la variabile record lib di tipo Libro campo x campo effettuando
// tutti i controlli
// N.B. la LETTURA o CARICAMENTO è un'operazione NON ATOMICA
printf("************* Lettura dati del record ***************\n");
// Isbn
do
   {
   printf("Inserisci Isbn: ");
   gets(lib.Isbn);
   }
while ((strlen(lib.Isbn) == 0) || (strlen(lib.Isbn) > 10));

// Titolo
do
   {
   printf("Inserisci Titolo: ");
   gets(lib.Titolo);
   }
while ((strlen(lib.Titolo) == 0) || (strlen(lib.Titolo) > 100));

// Autore
do
   {
   printf("Inserisci Autore: ");
   gets(lib.Autore);
   }
while ((strlen(lib.Autore) == 0) || (strlen(lib.Autore) > 50));
```

Anche la funzione **scanf**() con **"%s"** con le limitazioni che sappiamo (spazi)

In modo meno efficiente si può anche dichiarare una variabile **intera** di nome, ad esempio, **lung,** valorizzarla con il valore ritornato dalla funzione **strlen**() all'interno del **do-while** e poi testarla direttamente nella condizione dello stesso ciclo

```
do {
     printf("Inserisci Isbn: ");
     gets(lib.Isbn);
     lung = strlen(lib.Isbn);
    }
while ((lung == 0) || (lung > 10))
```

```
// CasaEditrice
do
   {
   printf("Inserisci Casa Editrice: ");
   gets(lib.CasaEditrice);
   }
while ((strlen(lib.CasaEditrice) == 0) || (strlen(lib.CasaEditrice) >
50));

// Prezzo
do
   {
   printf("Inserisci Prezzo: ");
   scanf("%f", &(lib.Prezzo));
   }
while ((lib.Prezzo <= 0) || (lib.Prezzo > 999.99));

// Pagine
do
   {
   printf("Inserisci pagine: ");
   scanf("%d", &(lib.Pagine));
   }
while ((lib.Pagine <= 0) || (lib.Pagine > 9999));

// Scrivo la variabile record lib di tipo Libro campo x campo
// N.B. la SCRITTURA o VISUALIZZAZIONE è un'operazione NON ATOMICA
printf("************* Scrittura dati del record ***************\n");
printf("Isbn: %s\n", lib.Isbn);
printf("Titolo: %s\n", lib.Titolo);
printf("Autore: %s\n", lib.Autore);
printf("Casa Editrice: %s\n",lib.CasaEditrice);
printf("Prezzo: %.2f\n",lib.Prezzo);
printf("Pagine: %d\n\n", lib.Pagine);

return 0;
}
```

Record e sottorecord

Riprendiamo ora la rappresentazione tabellare del tracciato record eseguita nell'*Esempio 2* marcando con una 'X' il campo FlagSubRec del primi quattro campi, facendo intendere che essi formino un altro record di fatto contenuto (ecco il perchè del termine sottorecord) in quello che vogliamo definire.

La nuova tabella sarà la seguente

Numero	Nome Campo	Tipo Campo	Lunghezza	Descrizione	FlagSubRec
1	Isbn	ARRAY DI CHAR	10	Codice ISBN del libro	X
2	Titolo	ARRAY DI CHAR	100	Titolo del libro	X
3	Autore	ARRAY DI CHAR	50	Autore del libro	X
4	CasaEditrice	ARRAY DI CHAR	50	Casa Editrice del libro	X
5	Prezzo	REAL	3,2	Prezzo del libro	
6	Pagine	INT	4	Pagine del libro	

Che modifiche devono essere apportate alla nostra PSEUDOCODIFICA?

*Innanzitutto **nell'ambiente GLOBALE** dovremo definire il tipo record **Libro** tenendo conto che esso contiene al suo interno un altro record più piccolo (chiamato per questo SOTTORECORD del record principale). Per ovvie ragioni si dovrà definire:*

*- prima il sottorecord fornendo **obbligatoriamente** anche un nome al suo tipo*

*- poi il record principale che **dovrà obbligatoriamente** indicare al suo interno un riferimento al sottorecord contenuto (ossia un campo di tipo sottorecord)*

.....

```
TIPO DettagliLibro =  RECORD
                          Isbn: ARRAY[10] DI CHAR
                          Titolo: ARRAY[100] DI CHAR
                          Autore: ARRAY[50] DI CHAR
                          CasaEditrice: ARRAY[50] DI CHAR
                      FINE RECORD

TIPO Libro =  RECORD
                  DetLib: DettagliLibro
                  Prezzo: REAL
                  Pagine: INT
              FINE RECORD
```

.....

*Ora **nell'ambiente LOCALE** potrà di conseguenza essere definita la seguente variabile:*

lib : **Libro**

Come si modificherà a questo punto l'accesso ai campi della variabile lib SEMPRE di tipo Libro che è stato però definito con un sottorecord al suo interno?

La modifica dell'accesso ai campi riguarderà **ESCLUSIVAMENTE** i campi appartenenti al SOTTORECORD i quali potranno essere raggiunti utilizzando opportunamente la dot notation.

Leggi (lib.DetLib.Titolo)

oppure

Scrivi (lib.DetLib.Isbn)

Tutto rimane uguale per i campi che non appartengono al sottorecord

Leggi (lib.Prezzo)

oppure

lib.Pagine ← 435

Esempio 7: Supponiamo ora di volere leggere (controllando la validità dei singoli campi) e stampare i dati contenuti in una variabile **lib** *di tipo* **Libro** *ossia di un record con la seguente struttura o tracciato che contiene al suo interno un SOTTORECORD*

ALGORITMO Leggi_Stampa_RECORD_SUBREC

TIPO DettagliLibro = RECORD
Isbn: ARRAY[10] DI CHAR
Titolo: ARRAY[100] DI CHAR
Autore: ARRAY[50] DI CHAR
CasaEditrice: ARRAY[50] DI CHAR
FINE RECORD

TIPO Libro = RECORD
DetLib: **DettagliLibro**
Prezzo: REAL
Pagine: INT
FINE RECORD

PROCEDURA main( )

lib: **Libro**

INIZIO

/ leggo il record* ***lib*** *campo x campo (operazione NON ATOMICA) effettuando tutti i controlli */*
RIPETI
Leggi (lib. **DetLib**.Isbn)
FINCHE' (Lunghezza(lib.**DetLib**.Isbn) ≠ 0) **AND** (Lunghezza(lib.**DetLib**.Isbn) ≤ **10**)
RIPETI
Leggi (lib. **DetLib**.Titolo)
FINCHE' (Lunghezza(lib.**DetLib**.Titolo) ≠ 0) **AND** (Lunghezza(lib.**DetLib**.Titolo) ≤ **100**)
RIPETI
Leggi (lib. **DetLib**.Autore)
FINCHE' (Lunghezza(lib.**DetLib**.Autore) ≠ 0) **AND** (Lunghezza(lib.**DetLib**.Autore) ≤ **50**)
RIPETI
Leggi (lib. **DetLib**.asaEditrice)
FINCHE' (Lunghezza(lib.**DetLib**.CasaEditrice) ≠ 0) **AND** (Lunghezza(lib.**DetLib**.CasaEditrice) ≤ **50**)
RIPETI
Leggi (lib.Prezzo)
FINCHE' (lib.Prezzo) > 0) **AND** (lib.Prezzo) ≤ **999.99**)
RIPETI
Leggi (lib.Pagine)
FINCHE' (lib.Pagine) > 0) AND (lib.Pagine) ≤ **9999**)

/ visualizzo il record* ***lib*** *campo x campo (operazione NON ATOMICA)*/*

Scrivi (lib.**DetLib**.Isbn)
Scrivi (lib.**DetLib**.Titolo)
Scrivi (lib.**DetLib**.Autore)
Scrivi (lib.**DetLib**.CasaEditrice)
Scrivi (lib.Prezzo)
Scrivi (lib.Pagine)

FINE

Esempio 7 BIS: Implementazione in C dell'esempio n. 7

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//definizione del nuovo tipo sottorecord utente DettagliLibro
typedef struct
            {
            char Isbn[10+1];
            char Titolo[100+1];
            char Autore [50+1];
            char CasaEditrice[50+1];
            } DettagliLibro;

//definizione del nuovo tipo record utente Libro
typedef struct
            {
            DettagliLibro DetLib;
            float Prezzo;
            int Pagine;
            } Libro;

int main(int argc, char *argv[])
{
//Dichiarazione di una variabile record di tipo Libro
Libro lib;

// Leggo la variabile record lib di tipo Libro campo x campo effettuando
// tutti i controlli
// N.B. la LETTURA o CARICAMENTO è un'operazione NON ATOMICA
printf("************* Lettura dati del record ***************\n");
// Isbn
do
   {
   printf("Inserisci Isbn: ");
   gets(lib.DetLib.Isbn);
   }
while ((strlen(lib.DetLib.Isbn) == 0) || (strlen(lib.DetLib.Isbn) > 10));


// Titolo
do
   {
   printf("Inserisci Titolo: ");
   gets(lib.DetLib.Titolo);
   }
while ((strlen(lib.DetLib.Titolo)== 0) || (strlen(lib.DetLib.Titolo)> 100));
```

Per accedere ad un campo del record **lib** che fa parte del sottorecord di tipo **DettagliLibro**, occorre utilizzare più volte (in questo caso due) la **dot notation** passando per il campo di nome **DetLib**

```
// Autore
do
   {
   printf("Inserisci Autore: ");
   gets(lib.DetLib.Autore);
   }
while ((strlen(lib.DetLib.Autore)== 0) || (strlen(lib.DetLib.Autore) > 50));

// CasaEditrice
do
   {
   printf("Inserisci Casa Editrice: ");
   gets(lib.DetLib.CasaEditrice);
   }
while ((strlen(lib.DetLib.CasaEditrice) == 0) ||
       (strlen(lib.DetLib.CasaEditrice) > 50));

// Prezzo
do
   {
   printf("Inserisci Prezzo: ");
   scanf("%f", &(lib.Prezzo));
   }
while ((lib.Prezzo <= 0) || (lib.Prezzo > 999.99));

// Pagine
do
   {
   printf("Inserisci pagine: ");
   scanf("%d", &(lib.Pagine));
   }
while ((lib.Pagine <= 0) || (lib.Pagine > 9999));

// Scrivo la variabile record lib di tipo Libro campo x campo
// N.B. la SCRITTURA o VISUALIZZAZIONE è un'operazione NON ATOMICA
printf("************* Scrittura dati del record ***************\n");
printf("Isbn: %s\n", lib.DetLib.Isbn);
printf("Titolo: %s\n", lib.DetLib.Titolo);
printf("Autore: %s\n", lib.DetLib.Autore);
printf("Casa Editrice: %s\n",lib.DetLib.CasaEditrice);
printf("Prezzo: %.2f\n",lib.Prezzo);
printf("Pagine: %d\n\n", lib.Pagine);

return 0;
}
```

Per accedere ad un campo del record **lib** che **NON** fa parte del sottorecord di tipo **DettagliLibro** ma del record principale di tipo **Libro**, occorre utilizzare UNA VOLTA SOLA la **dot notation**

ARRAY DI RECORD

Così come i record possono avere i campi di tipo record oppure array, anche gli array possono essere composti da elementi di uno stesso tipo record.

Una struttura del genere prende il nome di **array di record**.

Esempio 1: Caricare e visualizzare i dati relativi ai libri presenti in una libreria
(N.B. SENZA effettuare controllo di validità sui campi)

N.B. senza effettuare controlli sui campi in input

```
ALGORITMO ArrayDiRecord_CHECK_NO

MAXDIM 10

TIPO Libro =  RECORD
                  Isbn: ARRAY[10] DI CHAR
                  Titolo: ARRAY[100] DI CHAR
                  Autore: ARRAY[50] DI CHAR
                  CasaEditrice: ARRAY[50] DI CHAR
                  Prezzo: REAL
                  Pagine: INT
              FINE RECORD

PROCEDURA main( )

libreria: ARRAY[MAXDIM] DI Libro
i, n: INT

INIZIO
/* leggo la dimensione del vettore da caricare */
RIPETI
   Leggi(n)
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)

/* Carico l'array di record valorizzando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Leggi(libreria[i].Isbn)
   Leggi(libreria[i].Titolo)
   Leggi(libreria[i].Autore)
   Leggi(libreria[i].CasaEditrice)
   Leggi(libreria[i].Prezzo)
   Leggi(libreria[i].Pagine)
   i ← i + 1
FINE PER

/* Visualizzo l'array di record stampando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Scrivi(libreria[i].Isbn)
   Scrivi(libreria[i].Titolo)
   Scrivi(libreria[i].Autore)
   Scrivi(libreria[i].CasaEditrice)
   Scrivi(libreria[i].Prezzo)
   Scrivi(libreria[i].Pagine)
   i ← i + 1
FINE PER

FINE
```

Esempio 2: Caricare e visualizzare i dati relativi ai libri posseduti da una libreria

N.B. Effettuando tutti i controlli sui campi previsti in fase di definizione del tracciato record

```
ALGORITMO ArrayDiRecord_CHECK_YES
MAXDIM 10
/* Per il tracciato record vedi l'esempio precedente */
.....
PROCEDURA main( )
libreria: ARRAY[MAXDIM] DI Libro
i,n: INT
INIZIO
/* leggo la dimensione del vettore da caricare (vedi esempi precedente) */
.....
/* Carico l'array di record controllando e valorizzando ciascun campo del record */
PER i ← 1 A n ESEGUI
   RIPETI
      Leggi(libreria[i].Isbn)
   FINCHE' (libreria[i].Isbn ≠ "") AND (Lunghezza(libreria[i].Isbn) ≤ 10)
   RIPETI
      Leggi(libreria[i].Titolo)
   FINCHE' (libreria[i].Titolo ≠ "") AND (Lunghezza(libreria[i].Titolo) ≤ 100)
   RIPETI
      Leggi(libreria[i].Autore)
   FINCHE' (libreria[i].Autore ≠ "") AND (Lunghezza(libreria[i].Autore) ≤ 50)
   RIPETI
      Leggi(libreria[i].CasaEditrice)
   FINCHE' (libreria[i].CasaEditrice ≠ "") AND (Lunghezza(libreria[i].CasaEditrice) ≤ 50)
   RIPETI
      Leggi(libreria[i].Prezzo)
   FINCHE' (libreria[i].Prezzo > 0) AND (libreria[i].Prezzo) ≤ 999.99)
   RIPETI
      Leggi(libreria[i].Pagine)
   FINCHE' (libreria[i].Pagine > 0) AND (libreria[i].Pagine) ≤ 9999)
   i ← i + 1
FINE PER
/* Visualizzo l'array di record stampando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Scrivi(libreria[i].Isbn)
   Scrivi(libreria[i].Titolo)
   Scrivi(libreria[i].Autore)
   Scrivi(libreria[i].CasaEditrice)
   Scrivi(libreria[i].Prezzo)
   Scrivi(libreria[i].Pagine)
   i ← i + 1
FINE PER
FINE
```

Esempio 2 BIS: Implementazione in C dell'esercizio n. 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXDIM 10

//definizione del nuovo tipo record utente Libro
typedef struct
            {
            char Isbn[10+1];
            char Titolo[100+1];
            char Autore [50+1];
            char CasaEditrice[50+1];
            float Prezzo;
            int Pagine;
            } Libro;

int main(int argc, char *argv[])
{
//Dichiarazione di un'ARRAY MONODIMENSIONALE di record di tipo Libro
Libro libreria[MAXDIM];
int i, n;

// Leggo e controllo la dimensione dell'array di record
// (FALSA DINAMICITA')
do
   {
   printf("Inserire il numero di libri presenti in libreria: ");
   scanf("%d", &n);
   }
while ((n < 1) || (n > MAXDIM));

// Leggo l'ARRAY DI RECORD di tipo Libro effettuando, per ciascun record,
// tutti i controlli campo x campo
// N.B. la LETTURA o CARICAMENTO è un'operazione NON ATOMICA
printf("******** Lettura dati dell' ARRAY DI RECORD **********\n");
for (i=0; i<n; i++)   //Inizio ciclo for
  {
  printf("Record n.%d\n", i+1);
  // Isbn
  do
     {
     printf("Inserisci Isbn: ", i+1);
     fflush(stdin);
     gets(libreria[i].Isbn);
     }
  while ((strlen(libreria[i].Isbn) == 0) || (strlen(libreria[i].Isbn) > 10));
```

libreria[i] è l'iesimo elemento di un array monodimensioanle di record di tipo **Libro** quindi è a tutti gli effetti un record di tipo Libro (è possibile dunque utilizzare la **dot notation**)

```
// Titolo
do
   {
   printf("Inserisci Titolo: ", i+1);
   fflush(stdin);
   gets(libreria[i].Titolo);
   }
while ((strlen(libreria[i].Titolo) == 0) ||
      (strlen(libreria[i].Titolo) > 100));

// Autore
do
   {
   printf("Inserisci Autore: ", i+1);
   fflush(stdin);
   gets(libreria[i].Autore);
   }
while ((strlen(libreria[i].Autore) == 0) ||
      (strlen(libreria[i].Autore) > 50));

// CasaEditrice
do
   {
   printf("Inserisci Casa Editrice: ", i+1);
   fflush(stdin);
   gets(libreria[i].CasaEditrice);
   }
while ((strlen(libreria[i].CasaEditrice) == 0) ||
      (strlen(libreria[i].CasaEditrice) > 50));

// Prezzo
do
   {
   printf("Inserisci Prezzo: ", i+1);
   fflush(stdin);
   scanf("%f", &(libreria[i].Prezzo));
   }
while ((libreria[i].Prezzo <= 0) || (libreria[i].Prezzo > 999.99));

// Pagine
do
   {
   printf("Inserisci Pagine: ", i+1);
   fflush(stdin);
   scanf("%d", &(libreria[i].Pagine));
   }
while ((libreria[i].Pagine <= 0) || (libreria[i].Pagine > 9999));
printf("--------------------------------\n");
}                    //Fine ciclo for
```

```
// Scrivo l'ARRAY DI RECORD di tipo Libro, un record alla volta
// campo x campo
// N.B. la SCRITTURA o VISUALIZZAZIONE è un'operazione NON ATOMICA
printf("******** Scrittura dati dell'ARRAY DI RECORD **********\n");
for(i=0; i<n; i++)         //Inizio ciclo for
   {
   printf("Record n.%d\n", i+1);
   printf("Isbn: %s\n", libreria[i].Isbn);
   printf("Titolo: %s\n", libreria[i].Titolo);
   printf("Autore: %s\n", libreria[i].Autore);
   printf("Casa Editrice: %s\n", libreria[i].CasaEditrice);
   printf("Prezzo: %.2f\n", libreria[i].Prezzo);
   printf("Pagine: %d\n", libreria[i].Pagine);
   printf("--------------------------------\n");
   }                        //Fine ciclo for

return 0;
}
```

RICAPITOLANDO: Esempi di PSEUDOCODIFICA

A1) Definizione di un array monodimensionale di nome v contenente numeri reali

v: ARRAY [MAXDIM] **DI REAL**

*A2) Definizione di un array monodimensionale di nome v contenente numeri reali **CON** l'utilizzo della'istruzione TIPO*

……
TIPO mio_tipo_vettore = ARRAY [MAXDIM] **DI REAL**
……
v: mio_tipo_vettore

B1) Definizione di una matrice o array bidimensionale di nome matr contenente numeri reali

matr: ARRAY [MAXDIM][MAXDIM] **DI REAL**

*B2) Definizione utilizzando l'istruzione TIPO della variabile matr di tipo **mio_tipo_matrice** costituito da un array bidimensionale*

……
TIPO mio_tipo_matrice = ARRAY [MAXDIM][MAXDIM] **DI REAL**
……
matr: mio_tipo_matrice

*C1) Definizione di un record di nome mag1 costituito da 4 campi **SENZA** l'utilizzo dell'istruzione **TIPO** (**sconsigliato**)*

```
……
mag1:    RECORD
              CodProdotto: ARRAY[6] DI CHAR
              NomeProdotto: ARRAY[25] DI CHAR
               Giacenza: INT
               Prezzo: REAL
         FINE RECORD
……
```

*C2) Definizione di un tipo record di nome mag1 costituito da 4 campi **CON** l'utilizzo l'istruzione **TIPO** e dichiarazione conseguente di una variabile mag1 di tipo **mio_tipo_record***

```
……
TIPO mio_tipo_record =    RECORD
                              CodProdotto: ARRAY[6] DI CHAR
                              NomeProdotto: ARRAY[25] DI CHAR
                              Giacenza: INT
                              Prezzo: REAL
                          FINE RECORD
……
mag1: mio_tipo_record

……
```

*C3) Definizione di un record di nome mag1 costituito da 4 campi utilizzando più volte l'istruzione **TIPO** (**SENZA** sottorecord)*

……

TIPO mio_tipo_codice_prodotto = **ARRAY**[6] **DI** CHAR

TIPO mio_tipo_nome_prodotto = **ARRAY**[25] **DI** CHAR

```
TIPO mio_tipo_record  =   RECORD
                              CodProdotto: mio_tipo_codice_prodotto
                              NomeProdotto: mio_tipo_nome_prodotto
                              Giacenza: INT
                              Prezzo: REAL
                          FINE RECORD
```

……
mag1: **mio_tipo_record**

*C4) Definizione di un record di nome mag2 costituito da 3 campi di cui uno è un record a sua volta utilizzando più volte l'istruzione **TIPO***

……

TIPO mio_tipo_codice_prodotto = **ARRAY**[6] **DI CHAR**

TIPO mio_tipo_nome_prodotto = **ARRAY**[25] **DI CHAR**

```
TIPO mio_tipo_sub_record =    RECORD
                                  CodProdotto: mio_tipo_codice_prodotto
                                  NomeProdotto: mio_tipo_nome_prodotto
                              FINE RECORD

TIPO mio_tipo_record =    RECORD
                              SottoRec: mio_tipo_sub_record
                              Giacenza: INT
                              Prezzo: REAL
                          FINE RECORD
```

……
mag2: **mio_tipo_record**

*D1) Definizione di un array di record di nome v2 utilizzando l'istruzione **TIPO** (**SENZA** sottorecord)*

……

```
TIPO mio_tipo_record =    RECORD
                              CodProdotto: ARRAY[6] DI CHAR
                              NomeProdotto: ARRAY[25] DI CHAR
                              Giacenza: INT
                              Prezzo: REAL
                          FINE RECORD
```

……

v2: **ARRAY** [MAXDIM] **DI** **mio_tipo_record**

MODELLO ASTRATTO (PSEUDOCODIFICA DI UN RECORD)

POSSIBILI RAPPRESENTAZIONI DI UN TRACCIATO RECORD

A) Metodo di rappresentazione tabellare

Numero	Nome Campo	Tipo Campo	Lunghezza[1]	Descrizione	FlagSubRec [2]	
1	CodProdotto	ARRAY DI CHAR	6	Codice del Prodotto		
2	NomeProdotto	ARRAY DI CHAR	25	Nome del prodotto		
3	Giacenza	INT	4	Giacenza del prodotto		
4	Prezzo	REAL	3,2	Prezzo unitario del prodotto		

B) Metodo dichiarazione sintattica (PSEUDOCODIFICA)

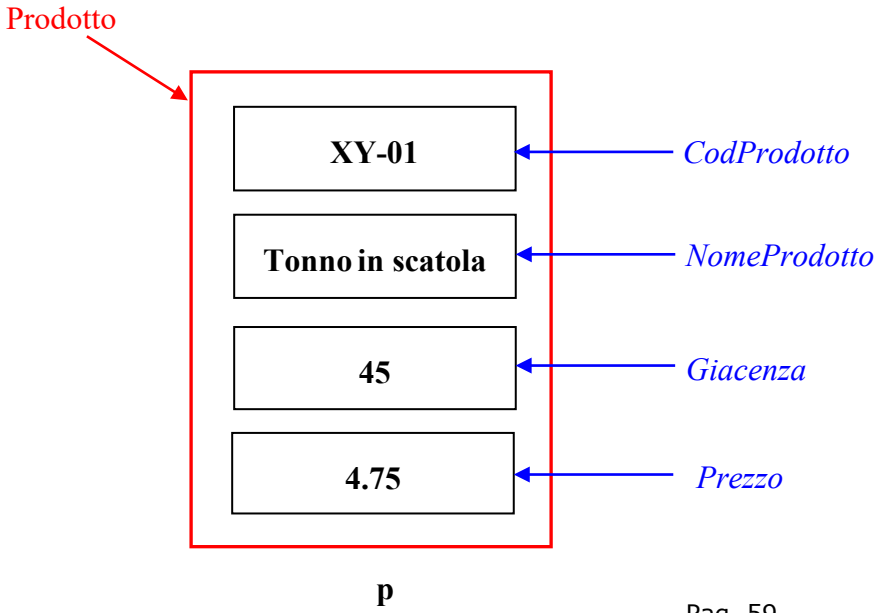
```
.....
TIPO Prodotto = RECORD
                    CodProdotto: ARRAY[6] DI CHAR
                    NomeProdotto: ARRAY[25] DI CHAR
                    Giacenza: INT
                    Prezzo: REAL
                FINE RECORD
.....
p : Prodotto
.....
```

Cosa accade fisicamente.....

Prodotto

CodProdotto	*NomeProdotto*	*Giacenza*	*Prezzo*
XY-01	Tonno in scatola	45	4.75

Cosa accade logicamente (a livello astratto)....



ALGORITMO CaricaVisualizzaAssegnaRecord_Prodotto_NOCONTROL_NOSUBREC

TIPO Prodotto = **RECORD**
CodProdotto : **ARRAY**[6] **DI CHAR**
NomeProdotto : **ARRAY**[25] **DI CHAR**
Giacenza : **INT**
Prezzo :**REAL**
FINE RECORD

Ambiente globale

PROCEDURA main ( )
p1, p2 : **Prodotto**

Ambiente locale

INIZIO

/ leggo il record **p1** campo x campo (operazione NON ATOMICA) SENZA EFFETTUARE tutti i controlli */*
Leggi (p1.CodProdotto)
Leggi (p1.NomeProdotto)
Leggi (p1.Giacenza)
Leggi (p1.Prezzo)

/ visualizzo il record **p1** campo x campo (operazione NON ATOMICA)*/*
Scrivi (p1.CodProdotto)
Scrivi (p1.NomeProdotto)
Scrivi (p1.Giacenza)
Scrivi (p1.Prezzo)

/ Assegnazione tra 2 record (operazione ATOMICA) – in questo caso equivale a 4 assegnazioni */*

p2 ← p1 (*)

/ visualizzo il record **p2** campo x campo (operazione NON ATOMICA)*/*
Scrivi (p2.CodProdotto)
Scrivi (p2.NomeProdotto)
Scrivi (p2.Giacenza)
Scrivi (p2.Prezzo)

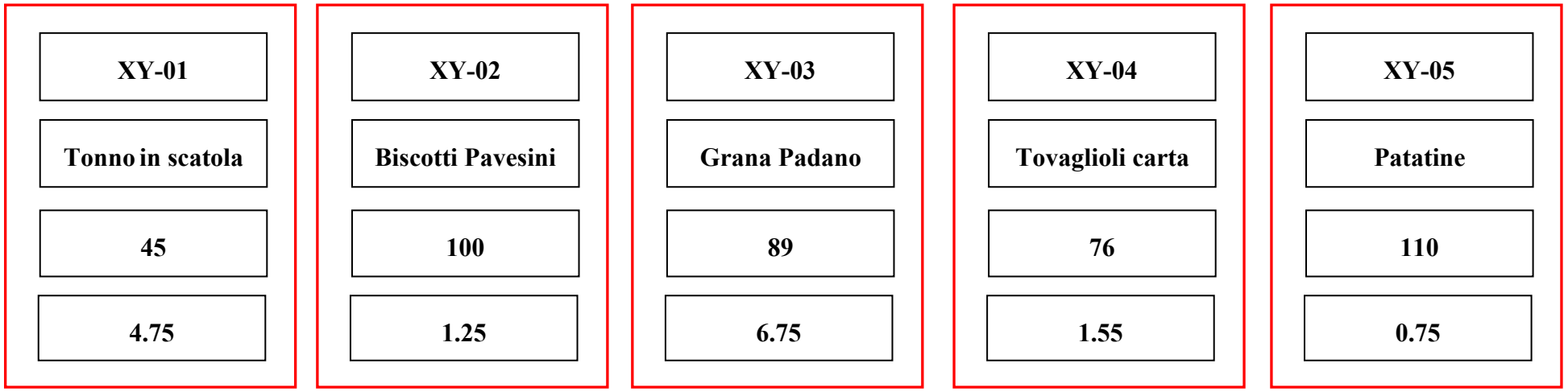
FINE

(*)

p2.CodProdotto ← p1.CodProdotto
p2.NomeProdotto ← p1.NomeProdotto
p2.Giacenza ← p1.Giacenza
p2.Prezzo ← p1.Prezzo

Cosa accade se voglio considerare insieme, ad esempio, 5 variabili TUTTE DELLO STESSO TIPO Prodotto ?

Qual è la struttura dati, se esiste, in grado di fare cio'?



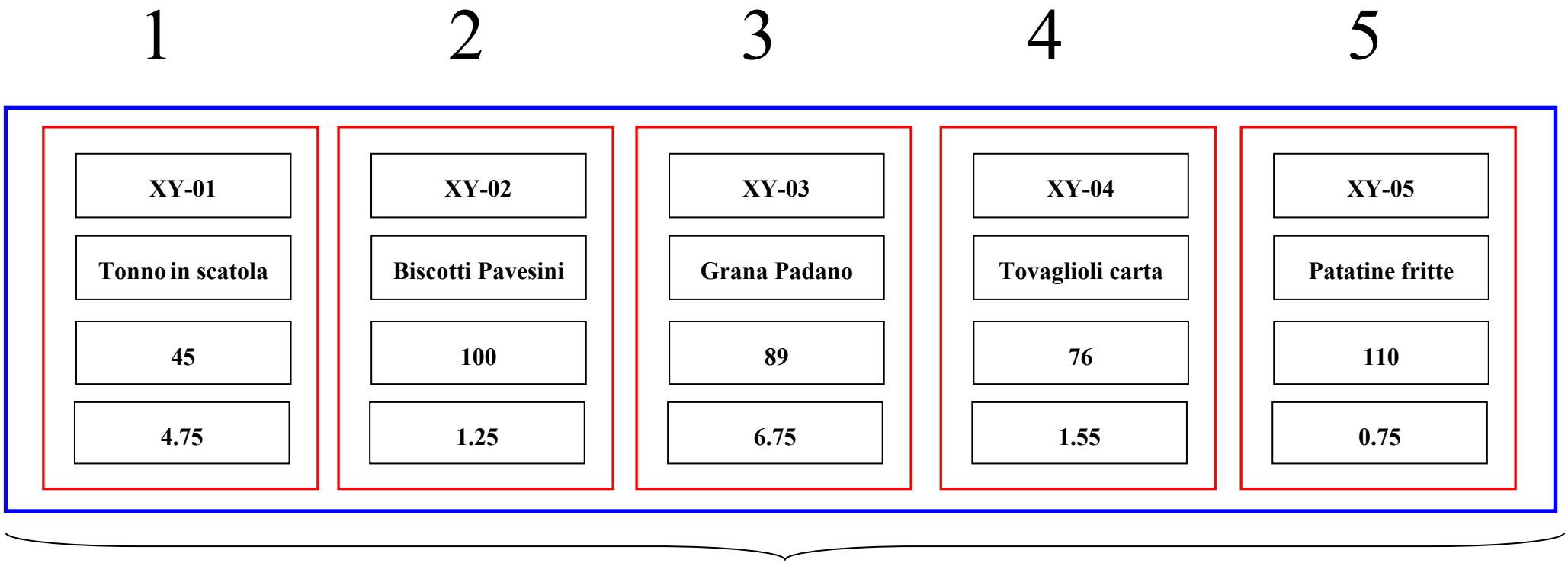

```
.....
TIPO Prodotto = RECORD
                    CodProdotto: ARRAY[6] DI CHAR
                    NomeProdotto: ARRAY[25] DI CHAR
                    Giacenza: INT
                    Prezzo: REAL
                FINE RECORD

.....
mag1, mag2, mag3, mag4, mag5 : Prodotto
.....
```

Tentativo 1

Ho messo così insieme 5 variabili DELLO STESSO TIPO Prodotto

... apparentemente sembra che io abbia dato una valida risposta.

... peccato che queste 5 variabili, seppur dello stesso tipo, siano **INDIPENDENTI TRA LORO** e **NON FACCIANO PARTE DI UNA UNICA STRUTTURA DATI**....



.....
TIPO Prodotto = **RECORD**
CodProdotto: **ARRAY**[6] **DI CHAR**
NomeProdotto: **ARRAY**[25] **DI CHAR**
Giacenza: **INT**
Prezzo: **REAL**
FINE RECORD
.....
v **: ARRAY DI [**5**] DI** Prodotto
.....

V

Tentativo 2 (quello giusto)

Costruisco un **ARRAY MONODIMENSIONALE costituito da 5 RECORD** di tipo Prodotto

ARRAY MONODIMESIONALE DI RECORD

ARRAY MONODIMESIONALE DI RECORD (NO sottorecord)

```
.....
TIPO Prodotto = RECORD
                        CodProdotto: ARRAY[6] DI CHAR
                        NomeProdotto: ARRAY[25] DI CHAR
                        Giacenza: INT
                        Prezzo: REAL
                FINE RECORD
.....
v : ARRAY DI [10] DI Prodotto
.....
```

v[3].CodProdotto ← "XYZ"

v.Giacenza ← 10

v[0].Prezzo ← "1200.75"

Leggi(v[4].NomeProdotto)

Scrivi(v[10])

v[1].Prezzo ← v[1].Prezzo + 100

Leggi(v[8].CodProdotto[6])

v[7].NomeProdotto[26] ← '$'

v[3] ← v[5]

Scrivi(v[7].Giacenza)

v[1].Prodotto ← "PIPPO"

v[3].CodProdotto ← v[8].NomeProdotto

v[3].CodProdotto ← v[6].CodProdotto[4]

Assegna all'ottavo carattere del campo "NomeProdotto" del quarto elemento dell'array **v** il valore '@'

Diminuisci di 20 unita il campo "Giacenza" del sesto elemento dell'array **v**

Diminuisci del 10% il prezzo del quinto elemento dell'array **v**

```
ALGORITMO ArrayDiRecord_Prodotto_NOCONTROL_NOSUBREC
MAXDIM 10
TIPO Prodotto = RECORD
                    CodProdotto: ARRAY[6] DI CHAR
                    NomeProdotto: ARRAY[25] DI CHAR
                    Giacenza: INT
                    Prezzo: REAL
                FINE RECORD
```

Ambiente globale

```
PROCEDURA main( )
magazzino: ARRAY[MAXDIM] DI Prodotto
i,n: INT
```

Ambiente locale

```
INIZIO
/* leggo la dimensione del vettore da caricare */
RIPETI
   Leggi(n)
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
/* Carico l'array di record valorizzando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Leggi (magazzino[i].CodProdotto)
   Leggi (magazzino[i].NomeProdotto)
   Leggi (magazzino[i].Giacenza)
   Leggi (magazzino[i].Prezzo)
   i ← i + 1
FINE PER

/* Visualizzo l'array di record stampando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Scrivi (magazzino[i].CodProdotto)
   Scrivi (magazzino[i].NomeProdotto)
   Scrivi (magazzino[i].Giacenza)
   Scrivi (magazzino[i].Prezzo)
   i ← i + 1
FINE PER
FINE
```

```
ALGORITMO ArrayDiRecord_Prodotto_SICONTROL_NOSUBREC
MAXDIM 10
TIPO Prodotto = RECORD
                    CodProdotto: ARRAY[6] DI CHAR
                    NomeProdotto: ARRAY[25] DI CHAR
                    Giacenza: INT
                    Prezzo: REAL
                FINE RECORD
PROCEDURA main( )
magazzino: ARRAY[MAXDIM] DI Prodotto
i,n: INT
INIZIO
/* leggo la dimensione del vettore da caricare (vedi esempi precedente) */
.......
/* Carico l'array di record controllando e valorizzando ciascun campo del record */
PER i ← 1 A n ESEGUI
   RIPETI
      Leggi (magazzino[i].CodProdotto)
   FINCHE' (magazzino[i]. CodProdotto ≠ "") AND (Lunghezza(magazzino [i].CodProdotto) ≤ 6)
   RIPETI
      Leggi (magazzino[i].NomeProdotto)
   FINCHE' (magazzino[i]. NomeProdotto ≠ "") AND (Lunghezza(magazzino [i].NomeProdotto) ≤ 25)
   RIPETI
      Leggi (magazzino[i].Giacenza)
   FINCHE' (magazzino[i].Giacenza ≥ 0) AND (magazzino[i].Giacenza) ≤ 9999)
   RIPETI
      Leggi (magazzino[i].Prezzo)
   FINCHE' (magazzino[i].Prezzo ≥ 0) AND (magazzino[i].Prezzo) ≤ 999.99)
   i ← i + 1
FINE PER

/* Visualizzo l'array di record stampando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Scrivi (magazzino[i].CodProdotto)
   Scrivi (magazzino[i].NomeProdotto)
   Scrivi (magazzino[i].Giacenza)
   Scrivi (magazzino[i].Prezzo)
   i ← i + 1
FINE PER
FINE
```

Ambiente globale

Ambiente locale

Uso del SOTTORECORD

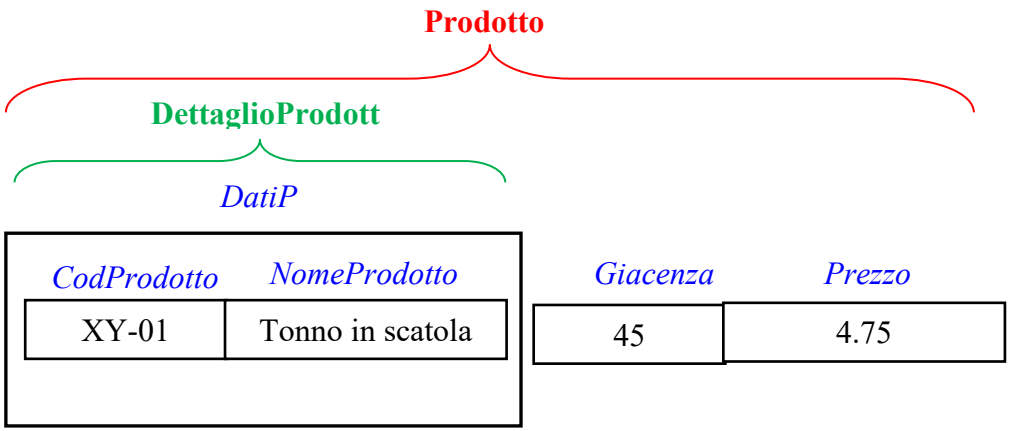
A) Metodo di rappresentazione tabellare

Numero	Nome Campo	Tipo Campo	Lunghezza[1]	Descrizione	FlagSubRec[2]
1	CodProdotto	ARRAY DI CHAR	6	Codice del Prodotto	X
2	NomeProdotto	ARRAY DI CHAR	25	Nome del prodotto	X
3	Giacenza	INT	4	Giacenza del prodotto	
4	Prezzo	REAL	3,2	Prezzo initario del prodotto	

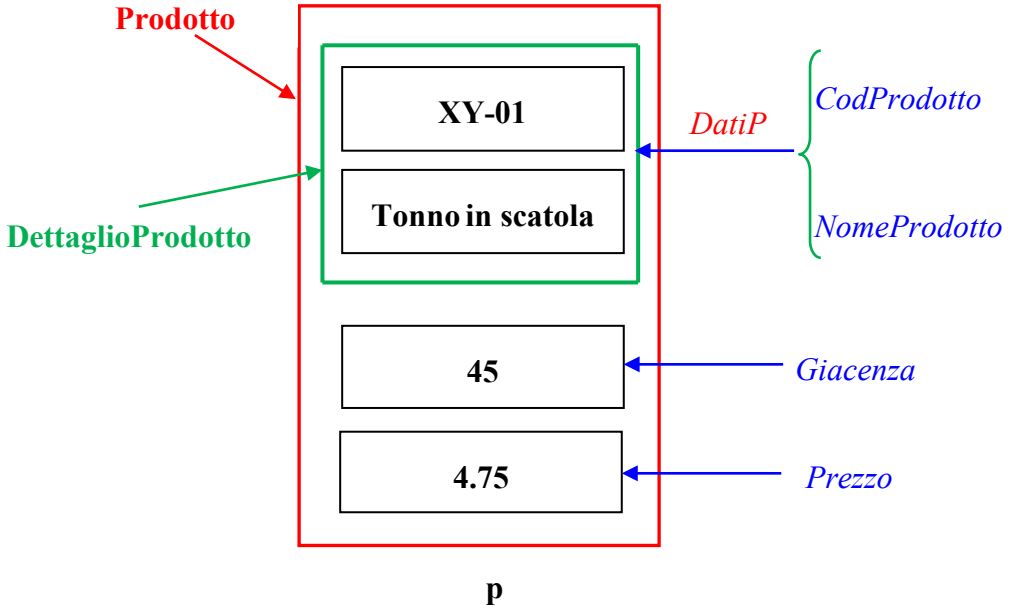
B) Metodo dichiarazione sintattica (PSEUDOCODIFICA)

```
.....
TIPO DettaglioProdotto = RECORD
                            CodProdotto: ARRAY[6] DI CHAR
                            NomeProdotto: ARRAY[25] DI CHAR
                         FINE RECORD
.....
TIPO Prodotto = RECORD
                    DatiP : DettaglioProdotto
                    Giacenza: INT
                    Prezzo: REAL
                FINE RECORD
.....
p : Prodotto
.....
```

Cosa accade fisicamente.....



Cosa accade logicamente (a livello astratto)....



USO DEL SOTTORECORD

ALGORITMO CaricaVisualizzaAssegnaRecord_Prodotto_NOCONTROL_SISUBREC
TIPO DettaglioProdotto = **RECORD**
CodProdotto: ARRAY[6] DI CHAR
NomeProdotto: ARRAY[25] DI CHAR
FINE RECORD
.....
TIPO Prodotto = **RECORD**
DatiP : DettaglioProdotto
Giacenza: **INT**
Prezzo: **REAL**
FINE RECORD

Ambiente globale

PROCEDURA main ( )
p1, p2 : **Prodotto**

Ambiente locale

INIZIO

/ leggo il record **p1** campo x campo (operazione NON ATOMICA) SENZA EFFETTUARE tutti i controlli */*
Leggi (p1.DatiP.CodProdotto)
Leggi (p1.DatiP.NomeProdotto)
Leggi (p1.Giacenza)
Leggi (p1.Prezzo)

/ visualizzo il record **p1** campo x campo (operazione NON ATOMICA)*/*
Scrivi (p1.DatiP.CodProdotto)
Scrivi (p1.DatiP.NomeProdotto)
Scrivi (p1.Giacenza)
Scrivi (p1.Prezzo)

/ Assegnazione tra 2 record (operazione ATOMICA) – in questo caso equivale a 4 assegnazioni */*

p2 ← p1 (*)

/ visualizzo il record **p2** campo x campo (operazione NON ATOMICA)*/*
Scrivi (p2.DatiP.CodProdotto)
Scrivi (p2.DatiP.NomeProdotto)
Scrivi (p2.Giacenza)
Scrivi (p2.Prezzo)

FINE

(*)

p2.DatiP.CodProdotto ← p1.DatiP.CodProdotto
p2.DatiP.NomeProdotto ← p1.DatiP.NomeProdotto
p2.Giacenza ← p1.Giacenza

ALGORITMO CaricaVisualizzaAssegnaRecord_Prodotto_SICONTROL_SISUBREC

TIPO DettaglioProdotto = **RECORD**
CodProdotto: ARRAY[6] DI CHAR
NomeProdotto: ARRAY[25] DI CHAR
FINE RECORD
.....
TIPO Prodotto = **RECORD**
DatiP : DettaglioProdotto
Giacenza: **INT**
Prezzo: **REAL**
FINE RECORD

Ambiente globale

PROCEDURA main ( )

p1, p2 : **Prodotto**

Ambiente locale

INIZIO

/ leggo il record **p1** campo x campo (operazione NON ATOMICA) SENZA EFFETTUARE tutti i controlli */*

RIPETI
Leggi (p1.DatiP.CodProdotto)
FINCHE' (p1.DatiP.CodProdotto ≠ "") AND (Lunghezza(p1.DatiP.CodProdotto) ≤ 6)

RIPETI
Leggi (p1.DatiP.NomeProdotto)
FINCHE' (p1.DatiP.NomeProdotto ≠ "") AND (Lunghezza(p1.DatiP.NomeProdotto) ≤ 25)

RIPETI
Leggi (p1.Giacenza)
FINCHE' (p1.Giacenza ≥ 0) AND (p1.Giacenza ≤ 9999)

RIPETI
Leggi (p1.Prezzo)
FINCHE' (p1.Prezzo ≥ 0) AND (p1.Prezzo ≤ 999.99)

/ visualizzo il record **p1** campo x campo (operazione NON ATOMICA) */*

Scrivi (p1.DatiP.CodProdotto)
Scrivi (p1.DatiP.NomeProdotto)
Scrivi (p1.Giacenza)
Scrivi (p1.Prezzo)

/ Assegnazione tra 2 record (operazione ATOMICA) – in questo caso equivale a 4 assegnazioni */*

p2 ← p1 (*)

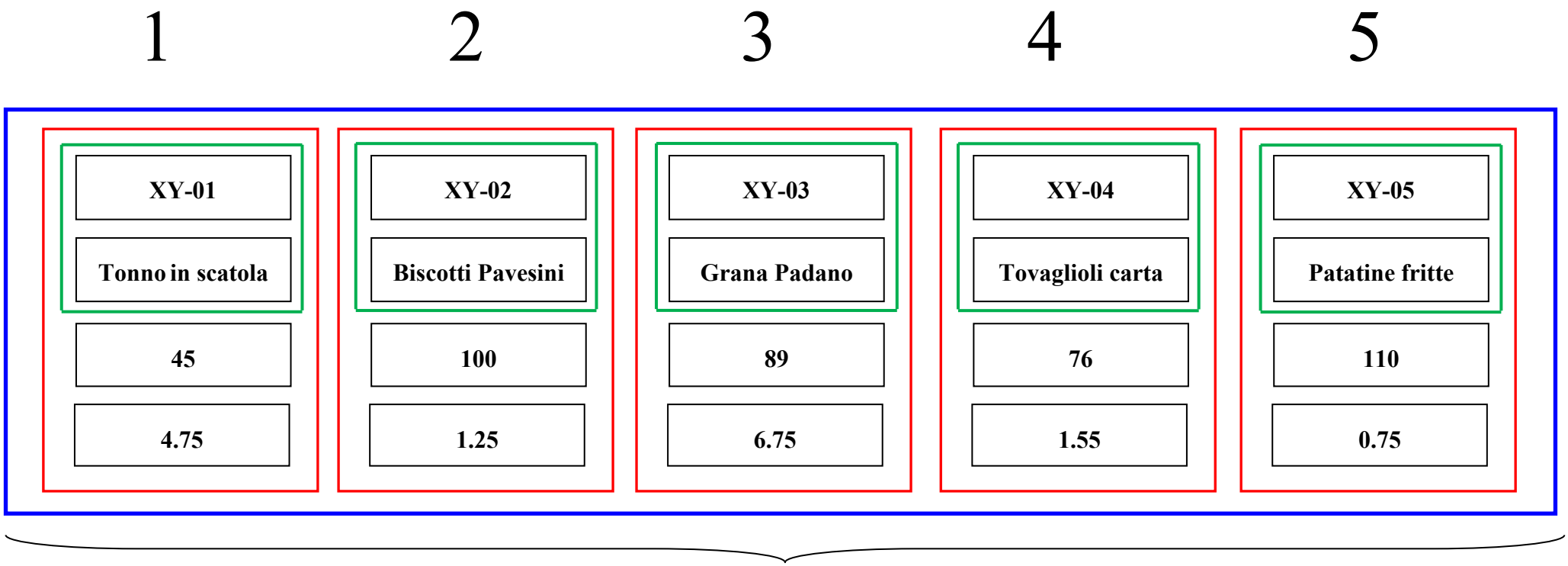
/ visualizzo il record **p2** campo x campo (operazione NON ATOMICA) */*

Scrivi (p2.DatiP.CodProdotto)
Scrivi (p2.DatiP.NomeProdotto)
Scrivi (p2.Giacenza)
Scrivi (p2.Prezzo)

FINE

(*)

p2.DatiP.CodProdotto ← p1.DatiP.CodProdotto
p2.DatiP.NomeProdotto ← p1.DatiP.NomeProdotto
p2.Giacenza ← p1.Giacenza



v

```
TIPO DettaglioProdotto = RECORD
                           CodProdotto: ARRAY[6] DI CHAR
                           NomeProdotto: ARRAY[25] DI CHAR
                           FINE RECORD
.....
TIPO Prodotto = RECORD
                    DatiP : DettaglioProdotto
                    Giacenza: INT
                    Prezzo: REAL
                FINE RECORD

v : ARRAY DI [5] DI Prodotto
```

SOLUZIONE

Costruisco un ARRAY MONODIMENSIONALE
costituito da 5 RECORD di tipo Prodotto
contenenti un campo di tipo DettaglioProdotto

ARRAY MONODIMESIONALE DI RECORD

ARRAY MONODIMESIONALE DI RECORD (SI sottorecord)

```
.....
TIPO DettaglioProdotto = RECORD
                              CodProdotto: ARRAY[6] DI CHAR
                              NomeProdotto: ARRAY[25] DI CHAR
                         FINE RECORD
.....
TIPO Prodotto = RECORD
                      DatiP : DettaglioProdotto
                      Giacenza: INT
                      Prezzo: REAL
                   FINE RECORD
.....
v : ARRAY DI [10] DI Prodotto
.....
```

v[3].CodProdotto ← "XYZ"

v[4].DettaglioProdotto ← 10

v[1].Prezzo ← 1200.75

Leggi(v[4].DatiP.NomeProdotto)

Scrivi(v[10].DatiP)

v[1].DatiP.Prezzo ← v[1].DatiP.Prezzo + 100

Leggi(v[8].DatiP.CodProdotto[6])

v[7].NomeProdotto[26] ← '@'

v[8] ← v[4]

Scrivi(v[3].Giacenza)

v[1].DatiP.NomeProdotto ← PIPPO

v[3].DatiP.CodProdotto ← v[0].DatiP.CodProdotto

v[3].CodProdotto[4] ← '$'

Assegna al quarto carattere del campo "CodProdotto" del sesto elemento dell'array **v** il valore '#'

Aumenta di 15 unita il campo "Giacenza" dell'ottavo elemento dell'array **v**

Aumenta del 15% il prezzo del secondo elemento dell'array **v**

```
ALGORITMO ArrayDiRecord_Prodotto_NOCONTROL_SISUBREC
MAXDIM 10
TIPO DettaglioProdotto = RECORD
                            CodProdotto: ARRAY[6] DI CHAR
                            NomeProdotto: ARRAY[25] DI CHAR
                         FINE RECORD
TIPO Prodotto = RECORD
                   DatiP : DettaglioProdotto
                   Giacenza: INT
                   Prezzo: REAL
                FINE RECORD
```

Ambiente globale

```
PROCEDURA main( )
magazzino: ARRAY[MAXDIM] DI Prodotto
i,n: INT
```

Ambiente locale

```
INIZIO
/* leggo la dimensione del vettore da caricare */
RIPETI
   Leggi(n)
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
/* Carico l'array di record valorizzando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Leggi (magazzino[i].DatiP.CodProdotto)
   Leggi (magazzino[i].DatiP.NomeProdotto)
   Leggi (magazzino[i].Giacenza)
   Leggi (magazzino[i].Prezzo)
   i ← i + 1
FINE PER

/* Visualizzo l'array di record stampando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Scrivi (magazzino[i].DatiP.CodProdotto)
   Scrivi (magazzino[i].DatiP.NomeProdotto)
   Scrivi (magazzino[i].Giacenza)
   Scrivi (magazzino[i].Prezzo)
   i ← i + 1
FINE PER
FINE
```

```
ALGORITMO ArrayDiRecord_Prodotto_SICONTROL_SISUBREC
MAXDIM 10
TIPO DettaglioProdotto = RECORD
                            CodProdotto: ARRAY[6] DI CHAR
                            NomeProdotto: ARRAY[25] DI CHAR
                         FINE RECORD
TIPO Prodotto = RECORD
                   DatiP : DettaglioProdotto
                   Giacenza: INT
                   Prezzo: REAL
                FINE RECORD
PROCEDURA main( )
magazzino: ARRAY[MAXDIM] DI Prodotto
i,n: INT
INIZIO
/* leggo la dimensione del vettore da caricare */
RIPETI
   Leggi(n)
FINCHE' (n ≥ 1) AND (n ≤ MAXDIM)
/* Carico l'array di record controllando e valorizzando ciascun campo del record */
PER i ← 1 A n ESEGUI
   RIPETI
      Leggi (magazzino[i].DatiP.CodProdotto)
   FINCHE' (magazzino[i].DatiP.CodProdotto ≠ "") AND (Lunghezza(magazzino[i].DatiP.CodProdotto) ≤ 6)
   RIPETI
      Leggi (magazzino[i].DatiP.NomeProdotto)
   FINCHE' (magazzino[i].DatiP.NomeProdotto ≠ "") AND (Lunghezza(magazzino[i].DatiP.NomeProdotto) ≤ 25)
   RIPETI
      Leggi (magazzino [i].Giacenza)
   FINCHE' (magazzino[i].Giacenza ≥ 0) AND (magazzino[i].Giacenza) ≤ 9999)
   RIPETI
      Leggi (magazzino[i].Prezzo)
   FINCHE' (magazzino[i].Prezzo ≥ 0) AND (magazzino[i].Prezzo) ≤ 999.99)
   i ← i + 1
FINE PER
/* Visualizzo l'array di record stampando ciascun campo del record */
PER i ← 1 A n ESEGUI
   Scrivi (magazzino[i].DatiP.CodProdotto)
   Scrivi (magazzino[i].DatiP.NomeProdotto)
   Scrivi (magazzino[i].Giacenza)
   Scrivi (magazzino[i].Prezzo)
   i ← i + 1
FINE PER
FINE
```

Ambiente globale

Ambiente locale