8. DATI SEMPLICI E STRUTTURE DATI

Iniziamo dando la definizione di "**tipo di dato**"

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 non vuoto 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) **tipi elementari o tipi semplici**, i cui dati non sono costituiti da altri dati;

b) **tipi strutturati,** i cui dati sono aggregazioni di tipi elementari che è possibile estrarre tramite opportune operazioni.

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 sia dalla loro realizzazione fisica sia dalla loro implementazione dettagliando il nuovo dato in funzione delle operazioni ammesse 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 indipendentemente dalla tecnologia

(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 a livello della memoria principale e/o secondarie dell'elaboratore.

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 dei dati"** o "**tipo di dato astratto o ADT".**

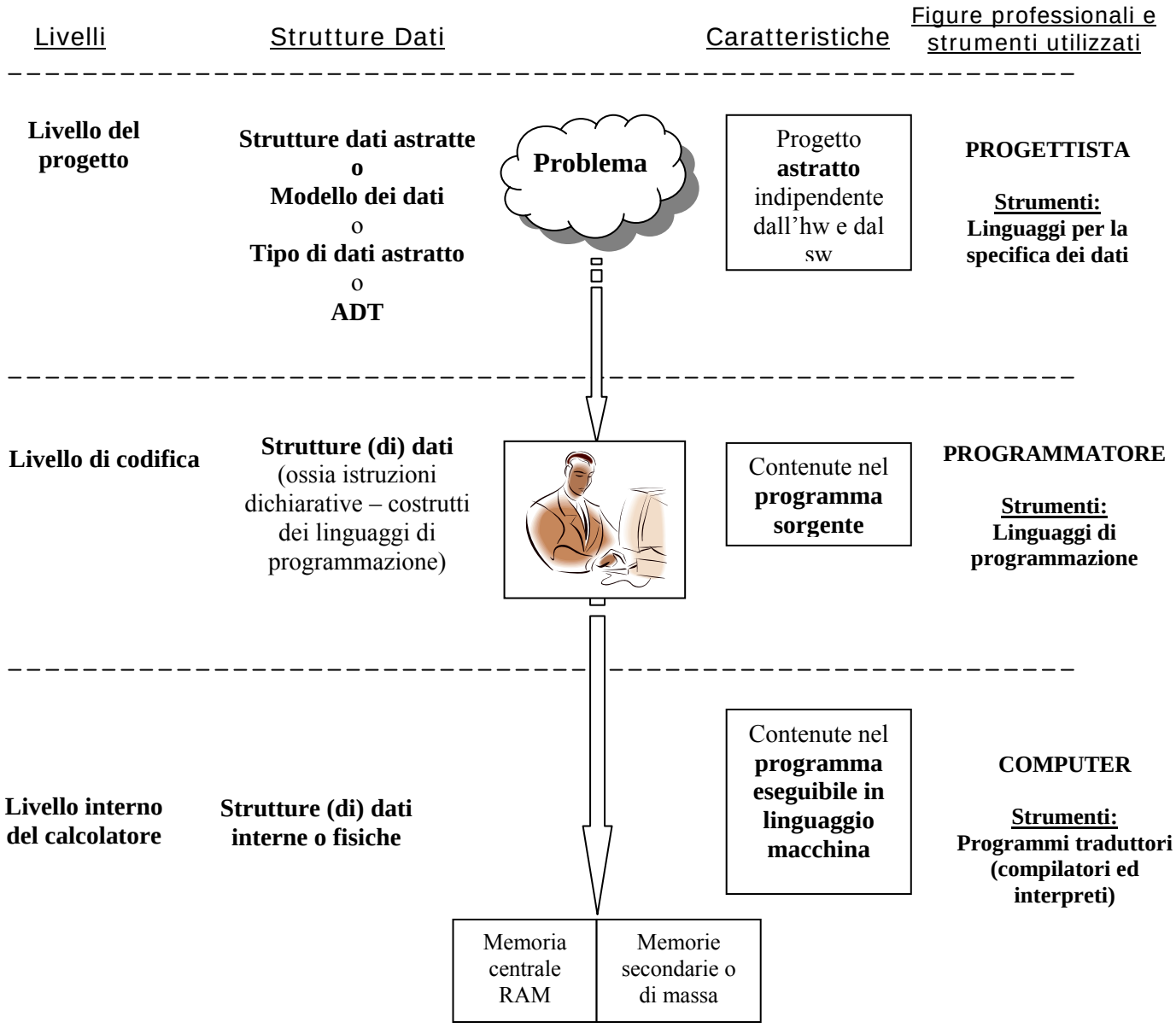
Il modello dei dati:

- NON dipende dal computer usato (hardware);
- NON dipende dal linguaggio di programmazione scelto per tradurre la struttura dati astratta e l'algoritmo nel programma sorgente.

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 elaboratore 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.

2) A LIVELLO INTERNO O FISICO **con il termine "struttura dati"** si intendono quelle strutture "fisiche" impiegate per memorizzare i dati nella memoria principale e/o di massa di un computer ossia sequenze di bit memorizzate e codificate opportunamente.



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.

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

- tipi di dati di cui sono composte: In questo caso si diranno **omogenee** le strutture dati composte da dati dello stesso tipo oppure **eterogenee** se composte da tipi di dati differenti;
- modalità di allocazione: In questo caso si diranno **statiche** se la dimensione riservata alla struttura dati in memoria, una volta fissata, non è più modificabile oppure **dinamiche** nel caso tale dimensione possa essere variata durante l'esecuzione;
- disposizione dei dati in memoria: In questo caso si diranno **sequenziali** se i singoli dati costituenti la struttura dati occupano locazioni contigue di memoria oppure **non sequenziali** se i singoli dati costituenti la struttura dati non occupano locazioni contigue di memoria;
- metodo di accesso: che si occupa del modo in cui è possibile individuare ogni singolo elemento all'interno della struttura dati. In questo caso si diranno ad **accesso sequenziale** quelle per le quali il reperimento di un singolo dato componente avviene scorrendo la struttura dall'inizio ed analizzando gli elementi uno dopo l'altro oppure ad **accesso diretto** quelle per le quali è possibile posizionarsi direttamente sull'elemento desiderato;
- supporto di memorizzazione: In questo caso si diranno **strutture di memoria** quelle memorizzate nella RAM e quindi cancellate ogni qualvolta si spenga l'elaboratore oppure **file o archivi** quelle memorizzate su dischi o nastri e che vengono quindi cancellate solo quando effettivamente richiesto.

N.B. In questa sezione ci occuperemo esclusivamente delle <u>strutture di dati di memoria</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 TIPO STRUTTURATO VETTORE

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

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

N.B. Nella PSEUDOCODIFICA per la dichiarazione di una variabile di questo tipo usare:
<Nome Vettore> **: ARRAY** [<MAXDIMENSIONE>] **DI** <Tipo Elemento>

Esempio: Se dichiaro **vett: ARRAY [2] DI INT** mi riferirò ad un **array** monodimensionale contenenti **2** elementi di elementi del tipo **intero** di nome **vett.**

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

Un vettore in un algoritmo non può mai essere manipolato come se fosse un unico oggetto ma si deve sempre operare sui singoli elementi componenti.
A **livello logico** si possono definire delle **operazioni** (alle quali corrispondono dei *sottoprogrammi* e quindi dei *sottoalgoritmi*) proprie di questo tipo di strutture dati:

- **caricamento;**
- **visualizzazione;**
- **shift;**
- **rotazione;**
- **ricerca;**
- **ordinamento.**

Altre operazioni che occasionalmente possono essere richieste che possono essere ottenute a a partire dalle precedenti sono:

- **copiatura;** tutta la struttura o parte di essa può dover essere copiata in un'altra struttura;
- **fusione;** può essere richiesta la combinazione di 2 o più strutture in una singola struttura
- **separazione;** opposta della precedente un'unica struttura può essere suddivisa in 2 o più strutture.

Il caricamento consente di assegnare un valore ad ogni elemento del vettore.
La **visualizzazione** o stampa consente di accedere a tutti gli elementi della struttura (oppure una sua parte) per visualizzarne il contenuto.

Esempio: Scrivere algoritmo risolutivo che permetta di caricare un qualunque vettore di interi di una qualunque dimensione fissata (minore o uguale alla massima dimensione fissata) e visualizzarne successivamente i valori.

ALGORITMO CaricaVisualizzaVettore

PROCEDURA main ( )

vett: ARRAY [MAXDIM] DI INT
n : INT
i : INT

INIZIO

/ leggi la dimensione del vettore da caricare rispettando il vincolo imposto da MAXDIM */*
RIPETI
  Leggi (n)
FINCHE' (n >= 1) AND (n <= MAXDIM)

/ carica gli elementi nel vettore */*
PER i ← 1 **A** n **ESEGUI**
  Leggi (vett[i])
  i ← i + 1
FINE PER

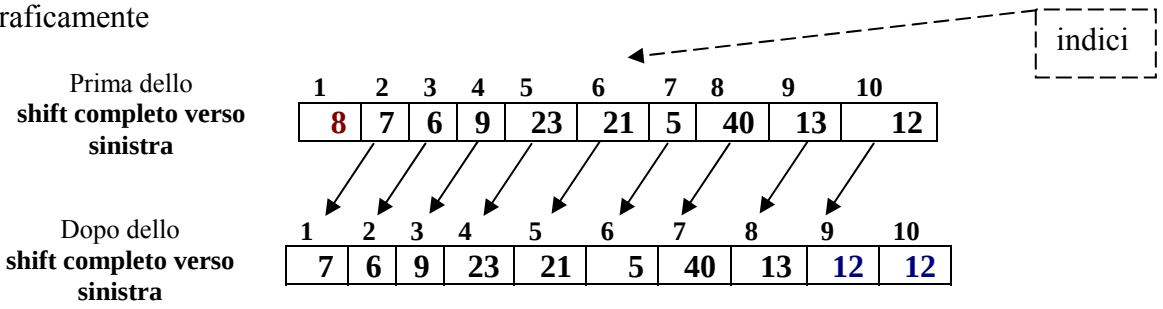
/ visualizza gli elementi precedentemente immessi nel vettore */*
PER i ← 1 **A** n **ESEGUI**
  Scrivi (vett[i])
  i ← i + 1
FINE PER

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



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

ALGORITMO ShiftCompletoSinistro

PROCEDURA main ( )

vett: ARRAY [MAXDIM] DI INT
n: INT
i: INT

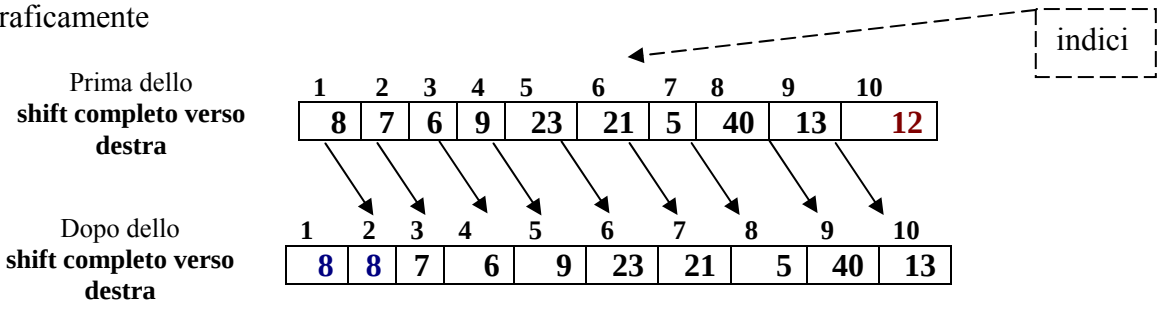
INIZIO

/ 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**
  vett[i] ← vett[i +1]
  i ← i + 1
FINE PER

/ visualizza gli elementi precedentemente immessi nel vettore (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

PROCEDURA main ( )

vett: ARRAY [MAXDIM] DI INT
n: INT
i: INT

INIZIO

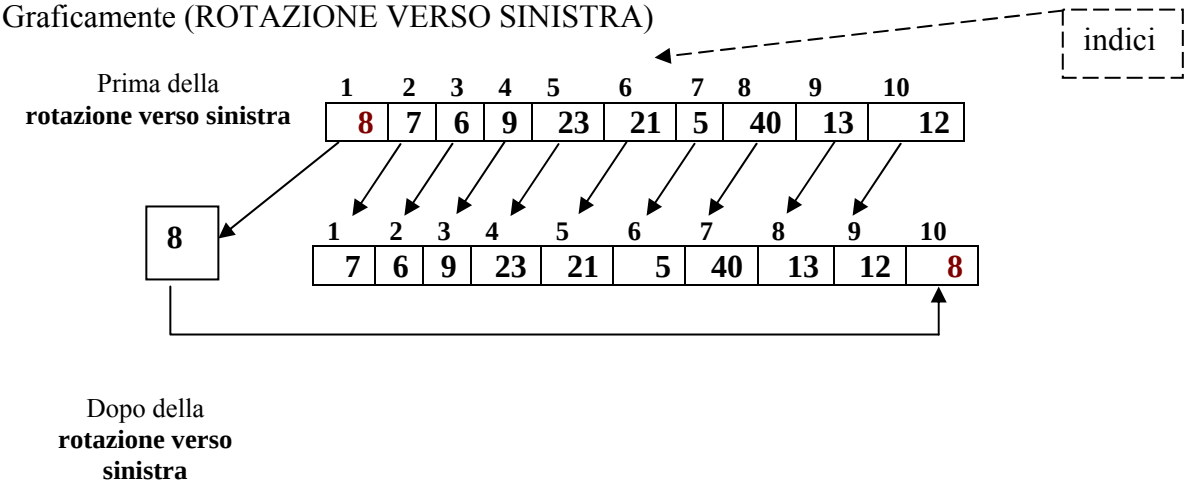
/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
….
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
….
/ effettua lo shift completo verso destra degli elementi nel vettore */*
PER i ← n **INDIETRO A** 2 **ESEGUI**
  vett[i] ← vett[i -1]
  i ← i - 1
FINE PER

/ visualizza gli elementi precedentemente immessi nel vettore (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)



ALGORITMO RotazioneSinistra

PROCEDURA main ( )

vett: 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 ← vett[1]

/* 2) Esecuzione dello shift completo a sinistra */
PER i ← 1 **A** (n – 1) **ESEGUI**
  vett[i] ← vett[i +1]
  i ← i + 1
FINE PER

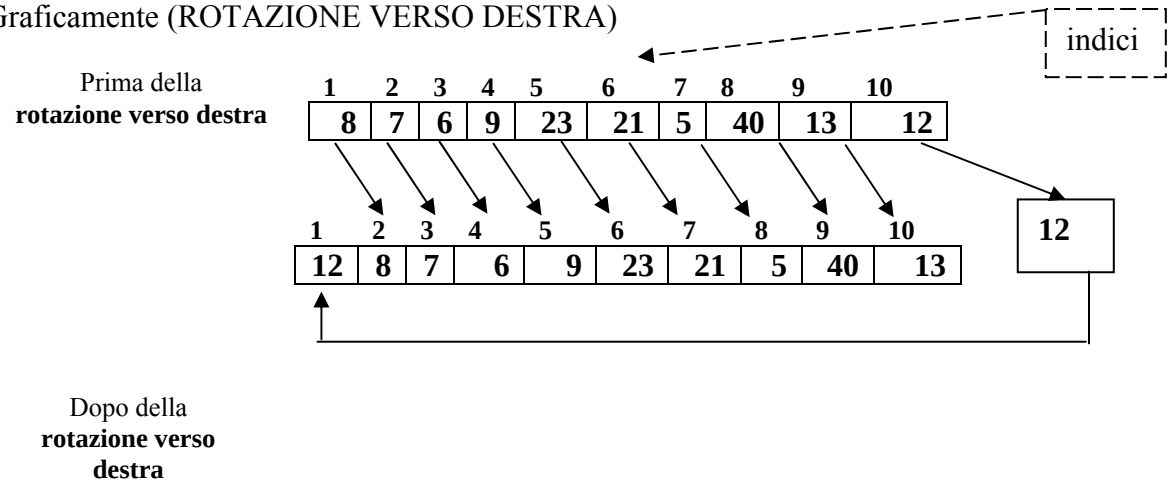
/* 3) Scrittura dell'elemento precedentemente salvato in ultima posizione */

vett[n]← scambio

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

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

Graficamente (ROTAZIONE VERSO DESTRA)



ALGORITMO RotazioneDestra

PROCEDURA main ( )

vett: 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 ← vett[i]
/* 2) Esecuzione dello shift completo a destra */
PER i ← n **INDIETRO A** 2 **ESEGUI**
  vett[i] ← vett[indice -1]
  i ← i - 1
FINE PER
/* 3) Scrittura dell'elemento precedentemente salvato in prima posizione */
vett[1]← scambio

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

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

ALGORITMI DI ORDINAMENTO

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 ricerca e di ordinamento.
Noi ne vedremo 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 (ossia il primo elemento con tutti gli altri, il secondo elemento con tutti gli altri tranne il primo e così via…)

ALGORITMO OrdinamentoIngenuo

PROCEDURA main ( )

vett: ARRAY [MAXDIM] DI INT
n: INT
i, j, scambio: INT

INIZIO

/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
....
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
....
/ effettua l'ordinamento in senso crescente per scambio del vettore/*

```
PER i ← 1 A (n – 1) ESEGUI
   PER j ← i+1 A n ESEGUI
     SE (vett[i] > vett[j])       /* con '>' senso crescente altrimenti con '<' senso decrescente */
       ALLORA
         scambio ← vett[i]
         vett[i] ← vett[j]
         vett[j] ← scambio
     FINE SE
     j ← j + 1
  FINE PER
  i ← i + 1
FINE PER
```

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

Esempio: Supponiamo di volere descrivere la tecnica di **ordinamento ingenuo** (**in senso crescente**) applicata al vettore di nome **vett** di **interi** con **quattro** elementi

Vettore di partenza di nome vett

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 vett[1] con il valore di vett[2].
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 vett[1] con il valore di vett[3].
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 vett[1] con il valore di vett[4].
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 <= n – 1) ossia (2 <= 3) VERO **Inizio Seconda scansione**

j ← i + 1 (j = 2 + 1 = 3)

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

1° CICLO 2° PER **1° passo**

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

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

j ← j + 1 (j = 3 + 1 = 4)

TEST 2° PER (j <= n) ossia (4 <= 4) VERO

1° CICLO 2° PER **2° passo**

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

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

j ← j + 1 (j = 4 + 1 = 5)

TEST 2° PER (j <= n) ossia (5 <= 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 ← i + 1 (i = 2 + 1 = 3)

TEST 1° PER (i <= n – 1) ossia (3 <= 3) VERO **Inizio Terza scansione**

j ← i + 1 (j = 3 + 1 = 4)
TEST 2° PER (j <= n) ossia (4 <= 4) VERO

1° CICLO 2° PER **1° passo**

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

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

j ← j + 1 (j = 4 + 1 = 5)

TEST 2° PER (j <= n) ossia (5 <= 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 al numero degli elementi del vettore – 1)

B) Algoritmo di ordinamento per bubble-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 salire (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 la descrizione dell'algoritmo di ordinamento per bubble-sort quindi dovremmo utilizzare:

- una variabile booleana che ci indichi che nella scansione non sono avvenuti scambi di valori tra elementi (chiamiamola *continua*);
- di una variabile che ci indichi il limite superiore fino al quale fare gli scambi per evitare ripetizioni inefficienti (chiamiamola *sup)* che viene inizializzata la prima volta con il numero di elementi del vettore e che assumerà valori via via decrescenti fino a che il valore non sarà ordinato;
- di una variabile (chiamiamola *k*) che dopo l'esecuzione di ogni scansione indichi la posizione del nuovo estremo superiore.

```
ALGORITMO OrdinamentoBubbleSort
PROCEDURA main ( )
vett: ARRAY [MAXDIM] DI INT
n, scambio: INT
i, sup, k: INT
continua: BOOL
INIZIO
/* leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/
….
/* carica gli elementi nel vettore(vedi esercizio precedente) */
….
/* effettua l'ordinamento in senso crescente bubble-sort del vettore/
k ← n
continua ← VERO
MENTRE (continua = VERO) ESEGUI
  sup ← k
  continua ← FALSO

  PER i ← 1 A (sup -1) ESEGUI
    SE (vett[i] > vett[i+1])          /* senso crescente altrimenti con '<' senso decrescente*/
      ALLORA
        scambio ← vett[i]
        vett[i] ← vett[i+1]
        vett[i+1] ← scambio
         k ← i                        /* accorcia il sottovettore da esaminare */
         continua ← VERO
    FINE SE
    i ← i + 1
  FINE PER
FINE MENTRE
/* visualizza gli elementi precedentemente immessi nel vettore (vedi esercizio precedente)*/
….
FINE
```

Esempio: Supponiamo di volere descrivere la tecnica di **ordinamento** (**in senso crescente**) per **bubble-sort** applicata al vettore di nome **vett** di **interi** con **quattro** elementi

Vettore di partenza di nome vett

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

L'algoritmo sopra proposto opererà così per ottenere l'ordinamento crescente del vettore:

INIZIO

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

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO **Inizio ciclo MENTRE**
1° ciclo MENTRE
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 **Prima scansione**

1° ciclo PER **1° passo**

confrontiamo il valore di vett[1] con il valore di vett[2] (ossia vett[1] > vett[2]?)
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 vett[2] con il valore di vett[3] (ossia vett[2] > vett[3]?)
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 vett[3] con il valore di vett[4] (ossia vett[3] > vett[4]?)
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 (che è risalito come una "bolla" in un liquido).

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO

2° ciclo MENTRE

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 **Seconda scansione**

1° ciclo PER **1° passo**

confrontiamo il valore di vett[1] con il valore di vett[2] (ossia vett[1] > vett[2]?)
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 vett[2] con il valore di vett[3] (ossia vett[2] > vett[3]?)
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 costituito dal vettore – l'ultimo elemento)

TEST MENTRE (continua = VERO) ossia (VERO = VERO)? VERO

3° ciclo MENTRE

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 **Terza scansione**

1° ciclo PER **1° passo**

confrontiamo il valore di vett[1] con il valore di vett[2] (ossia vett[1] > vett[2]?)
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 **Fine ciclo MENTRE**

FINE

N.B. Il vettore risulta perfettamente ordinato

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

LE DUE STRATEGIE 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.
Ciò non è sempre così.

Basterebbe ripetere per esercizio le scansioni/passi che i due algoritmi di ordinamento impiegano per ordinare in senso crescente il seguente altro vettore:

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

per accorgesri 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 mal criterio di ordinamento richiesto.
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

A) Algoritmo di **ricerca sequenziale:** tale metodo per funzionare NON RICHIEDE CHE I DATI SIANO ORDINATI e consiste in una serie di confronti tra il valore dell'elemento da ricercare con tutti gli altri elementi del vettore.

I confronti possono terminare nel momento in cui si trova l'elemento cercato o 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. In questo caso si parla di **scansione sequenziale**.

ALGORITMO RicercaSequenziale

PROCEDURA main ( )

```
vett: ARRAY [MAXDIM] DI INT
[posizione], i, n, elemento,: INT
trovato: BOOL

INIZIO

/* leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/
….
/* carica gli elementi nel vettore(vedi esercizio precedente) */
….
/*leggo l'elemento da ricercare */
Leggi (elemento)

/* effettua 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 ((i ≤ n) AND (trovato = FALSO)) ESEGUI

  SE (vett[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

/* comunica l'esito all'utente */
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
```

ALGORITMO ScansioneSequenziale

PROCEDURA main ( )

vett: ARRAY [MAXDIM] DI INT
i, n, elemento, numoccorrenze: INT

INIZIO

/ leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/*
….
/ carica gli elementi nel vettore(vedi esercizio precedente) */*
….
*/*leggo l'elemento da ricercare */*
Leggi (elemento)

/ effettua la ricerca dell'elemento all'interno del vettore non arrestandosi */*
/ e tenendo il conto del numero di volte in cui eventualmente compare */*
numcorrenze ← 0
i ← 1

MENTRE (i ≤ n) **ESEGUI**

  SE (vett[i] = elemento)
    ALLORA
      numoccorrenze ← numoccorrenze + 1
  FINE SE

  i ← i + 1 */* incremento fondamentale dell'indice */*

FINE MENTRE

/ comunica 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) Algoritmo di **ricerca binaria o dicotomica:** tale metodo per funzionare RICHIEDE OBBLIGATORIAMENTE CHE I DATI SIANO STATI PRECEDENTEMENTE ORDINATI.

Innanzitutto occorrerà identificare l'elemento che occupa la posizione centrale del vettore e confrontare questo elemento con quello da ricercare;
Dopo di ciò sarà possibile il verificarsi di uno soltanto tra i seguenti tre casi:

1. l'elemento da ricercare è più piccolo dell'elemento *centrale* fissato. Si scartano tutti gli elementi della metà destra del vettore e come ultimo elemento del vettore utile alla ricerca diventa quello immediatamente precedente all'elemento centrale;

2. l'elemento da ricercare è più grande dell'elemento *centrale* fissato. Si scartano tutti gli elementi della metà sinistra del vettore e come primo elemento del vettore utile alla ricerca diventa quello immediatamente successivo all'elemento centrale;
3. l'elemento da ricercare è proprio uguale all'elemento *centrale* fissato. Allora l'elemento ricercato appartiene al vettore e si trova in posizione centrale.

Nei casi 2 e 3 il discorso va ripetuto nello stesso modo finchè non si verifica il caso 3 oppure resti un unico elemento con il quale effettuare il confronto e questo sia diverso, ossia nel caso in cui tale elemento non appartenga al vettore

```
ALGORITMO RicercaBinaria

PROCEDURA main ( )

vett: ARRAY [MAXDIM] DI INT
primo, ultimo, centro: INT
[posizione], n, elemento :INT
trovato: BOOL

INIZIO
/* leggi la dimensione del vettore da caricare (vedi esercizio precedente)*/
….
/* carica gli elementi nel vettore(vedi esercizio precedente) */
….
/* ordina gli elementi nel vettore in uno dei modi studiati (vedi esercizi precedente) */
….
/* effettua la ricerca binaria dell'elemento all'interno del vettore */
[posizione ← 0]                          /* inizializzo, se richiesta, la posizione */
primo ← 1
ultimo ← n
trovato ← FALSO

MENTRE ((primo ≤ ultimo) AND (trovato = FALSO)) ESEGUI
  centro ← (primo + ultimo) DIV 2
  SE vett[centro] = elemento             /* 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 vett[centro] < elemento
        ALLORA
          primo ← centro + 1
        ALTRIMENTI
          ultimo ← centro – 1
      FINE SE
  FINE SE
FINE MENTRE

/* comunica l'esito all'utente */
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
```

IL TIPO DI DATI STRUTTURATO MATRICE

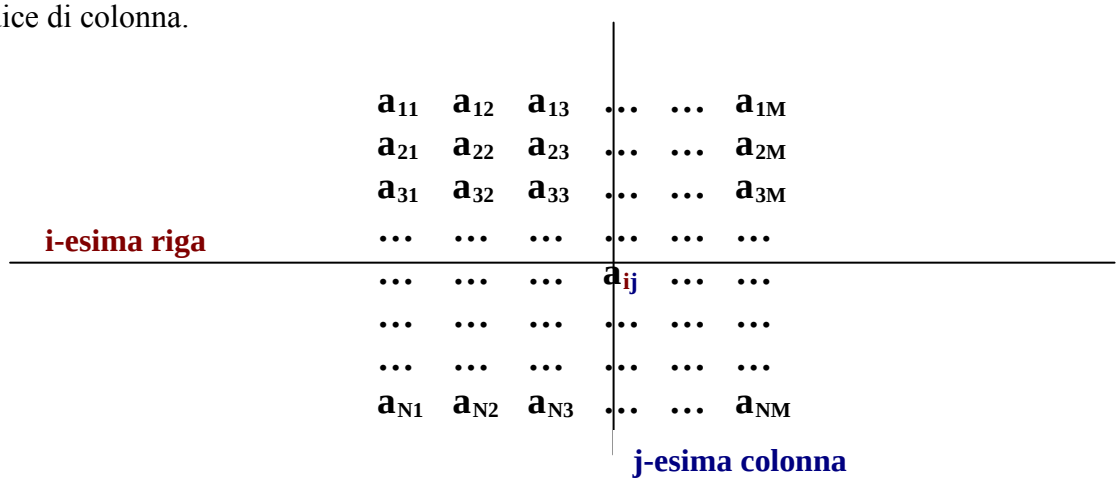
Per risolvere particolari tipi di problemi si ricorre a strutture dati più complesse del vettore monodimensionale che permette soltanto la memorizzazione di insiemi lineari di informazioni.

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 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**).

L'elemento generico di una matrice si indica con il simbolo $a_{i\ j}$ dove **i** è l'indice di riga e **j** è l'indice di colonna.



N.B. Nella PSEUDOCODIFICA per la dichiarazione di una variabile di questo tipo usare:
<Nome Matrice> **: ARRAY** [<Num Righe>] [<Num Colonne>] **DI** <Tipo Elemento>

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 n * m

N.B.

Per i tipi di dati strutturati come gli array è possibile 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.

TIPO <mio tipo> **=** <tipo di dato base>

ALGORITMO CaricaVisualizzaMatriceRettangolare

PROCEDURA main ( )

matr: ARRAY [MAXNUMRIGHE] [MAXNUMCOL] DI INT
righe, colonne: INT
i, j: INT

INIZIO

/ leggi il numero di righe e di colonne della matrice che si desidera caricare rispettando i vincoli imposti da* MAXNUMRIGHE e da MAXNUMCOL **/*
RIPETI
  Leggi (righe)
FINCHE' (righe > 0) AND (righe <= MAXNUMRIGHE)

RIPETI
  Leggi (colonne)
FINCHE' (colonne > 0) AND (colonne <= MAXNUMCOL)

/ carica gli elementi nella matrice per riga */*
PER i ← 1 **A** righe **ESEGUI**
  PER j ← 1 **A** colonne **ESEGUI**
    Leggi (matr[i, j])
    matr[i][j] ← elemento
    INCREMENTA j
   FINE PER
  INCREMENTA i
FINE PER

/ visualizza per riga gli elementi precedentemente immessi nella matrice */*
PER i ← 1 **A** righe **ESEGUI**
  PER j ← 1 **A** colonne **ESEGUI**
    Scrivi (matr[i][j])
    INCREMENTA j
   FINE PER
  INCREMENTA i
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: 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: Pensiamo ad libro caratterizzato tra l'altro in genere:
- 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);

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 a carattere statico 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 valore per un attributo.

Il **record** è un tipo di dati predefinito e viene messo a disposizione da molti 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:
- un **metodo di rappresentazione tabellare** consistente in una tabella che riporta per ogni campo *il numero di campo, in nome del campo, il tipo del campo, la lunghezza del campo, e la descrizione*. Tale metodo consente di definire il cosiddetto **tracciato record**.
- Un metodo di dichiarazione sintattica molto vicino a quello utilizzato dagli stessi linguaggi di programmazione che nella nostra PSEUDOCODIFICA sarà così:

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

Operazioni sui record

Le operazioni definite sui tipi di dato strutturati vengono distinte in **operazioni globali** ed **operazioni locali** a seconda che coinvolgono l'intera struttura oppure un solo elemento.
Per i record pertanto distingueremo le operazioni sul record dalle operazioni sul singolo campo.

Le **operazioni sui singoli campi** sono quelle consentite dal tipo di dato del campo.
Le **operazioni sui record** visto come un unico oggetto dato sono invece:

- Il **caricamento** dei campi del record;
- **L'assegnazione tra record**

Il caricamento del record è una operazione che permette di assegnare a ciascun campo un opportuno valore.

Esempio: Consideriamo la seguente variabile *mag1* che è un record così definito

```
mag1:   RECORD
          codprodotto: ARRAY[6] DI CHAR
          nomeprodotto: ARRAY[25] DI CHAR
          giacenza: INT
          prezzo: REAL
        FINE RECORD
```

Per potere accedere ad un campo occorre riferirsi attraverso la seguente notazione in PSEUDOCODIFICA:

<Nome Variabile Record>**.**<Nome Campo>

Quindi per assegnare il valore 25 al campo giacenza della struttura record l'istruzione corretta è

mag1.giacenza ← 25

L'assegnazione tra record è una operazione che permette di assegnare il valore di una variabile di tipo record ad un'altra ovviamente dello stesso tipo

Esempio: Consideriamo le seguenti variabili *mag1 e mag2* che sono un record così definito

```
mag1, mag2 :   RECORD
                 codprodotto: ARRAY[6] DI CHAR
                 nomeprodotto: ARRAY[25] DI CHAR
                 giacenza: INT
                 prezzo: REAL
               FINE RECORD
```

L'istruzione **mag1 ← mag2**
Assegna alla variabile mag1 il contenuto della variabile mag2 ossia memorizza in tutti i campi della variabile mag1 i singoli campi della variabile mag2. tale istruzione equivale ala seguente sequenza:

mag1.codiceprodotto ← mag2.codiceprodotto
mag1.nomeprodotto ← mag2.nomeprodotto
mag1.giacenza ← mag2.giacenza
mag1.prezzo ← mag2.prezzo

N.B. in alcuni linguaggi di programmazione è consentita una ulteriore operazione ossia il confronto tra record. Dove tale operazione non esistesse potrebbe facilmente essere simulata da un apposito sottoprogramma.

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

Esempi di pseudocodifica

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

vett: ARRAY [MAXNUMELEM] **DI** REAL

A2) Definizione di un array monodimensionale di nome vett contenente numeri reali con l'utilizzo della'istruzione TIPO

TIPO mio_vettore **= ARRAY** [MAXNUMELEM] **DI** REAL

vett: mio_vettore

B1) Definizione di un array bidimensionale di nome matr contenente numeri reali

matr: ARRAY [MAXNUMRIGHE][MAXNUMCOL] **DI** REAL

B2) Definizione utilizzando l'istruzione TIPO di un tipo mia_matrice costituito da un array bidimensionale

TIPO mia_matrice **= ARRAY** [MAXNUMRIGHE][MAXNUMCOL] **DI** REAL

matr: mia_matrice

C1) Definizione di un record di nome mag1 costituito da 4 campi

```
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 mio_record costituito da 4 campi utilizzando l'istruzione TIPO e dichiarazione conseguente di una variabile mag1 di tipo mio_tipo

```
TIPO mio_record =  RECORD
                     codprodotto: ARRAY[6] DI CHAR
                     nomeprodotto: ARRAY[25] DI CHAR
                     giacenza: INT
                     prezzo: REAL
                   FINE RECORD
mag1: mio_record
```

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

```
TIPO mio_prodotto =         ARRAY[6] DI CHAR

TIPO mio_nome_prodotto =    ARRAY[25] DI CHAR

TIPO mio_record =   RECORD
                        codprodotto: mio_prodotto
                        nomeprodotto: mio_nome_prodotto
                        giacenza: INT
                        prezzo: REAL
                    FINE RECORD
```

mag1: mio_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_prodotto =         ARRAY[6] DI CHAR

TIPO mio_nome_prodotto =    ARRAY[25] DI CHAR

TIPO mio_sub_record =       RECORD
                                codrodotto: mio_prodotto
                                nomeprodotto: mio_nome_prodotto
                            FINE RECORD

TIPO mio_record =   RECORD
                        sottoRec: mio_sub_record
                        giacenza: INT
                        prezzo: REAL
                    FINE RECORD
```

mag2: mio_record

D1) Definizione di un array di record di nome vett2 utilizzando più volte l'istruzione TIPO

```
TIPO mio_prodotto =         ARRAY[6] DI CHAR

TIPO mio_nome_prodotto =    ARRAY[25] DI CHAR

TIPO mio_record =   RECORD
                        codrodotto: mio_prodotto
                        nomeprodotto: mio_nome_prodotto
                        giacenza: INT
                        prezzo: REAL
                    FINE RECORD
```

vett2: ARRAY [MAXNUMELEM] DI mio_record