Vettori e matrici

Ver. 3

© 2021 - Claudio Fornaro - Corso di programmazione in C (C89)

Problema

- Si vuole un programma che chieda 10 numeri dalla tastiera e li visualizzi dall'ultimo al primo
- Soluzione attuabile con le conoscenze attuali:

```
int a,b,c,d,e,f,g,h,i,j;
scanf("%d", &a);
scanf("%d", &b);
…
scanf("%d", &j);
printf("%d", &j);
printf("%d", &i);
…
printf("%d", &a);
```

Problema

- Considerazione:
  e se i numeri fossero 100? 1000? 10000?
- Soluzione ideale:
  definire *con una sola istruzione* tutte le N variabili dello stesso tipo, identificabili singolarmente tramite un numero progressivo (e poi per far variare il numero progressivo si può usare un ciclo FOR)

Tipi scalari e tipi aggregati

- Una variabile che può contenere un solo valore si dice essere di **tipo scalare**
- I tipi aritmetici e i puntatori (descritti in altre slide) sono tipi scalari
- Una variabile che può contenere più elementi si dice di **tipo aggregato**
- I tipi aggregati sono i vettori e le `struct` (descritte in altre slide)

Vettori

- *Variabili scalari:* contengono un solo valore
- *Variabili vettoriali:* contengono più valori *dello stesso tipo*, detti ***elementi***
- Tutti gli elementi hanno lo *stesso nome*, ma sono contraddistinti da un numero (*indice*) indicato tra parentesi quadre:

  `a[1] a[2] a[3] a[4]`

  in termini matematici questo si scrive:

  $a_1 \ a_2 \ a_3 \ a_4$
- Se il *tipo* è di base (come `char`, `int`, `long`, `double`, etc.), tutti gli elementi del vettore sono variabili *scalari* di quel tipo

Vettori

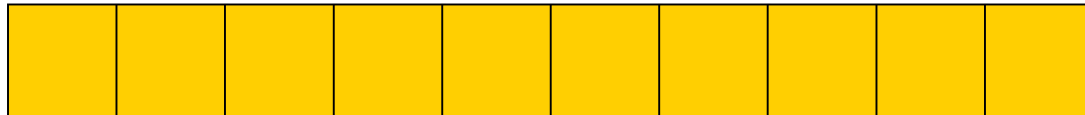
- Definizione (alloca tutta insieme memoria per tutti gli elementi):

  tipo ***nome*** [*num_elementi*] ;

- Esempio

  `int vett[10];` → *vettore di 10 elem. di tipo* `int`

  `vett:`



- Per migliorare la leggibilità non si mettano spazi tra ***nome*** e la parentesi quadra

Vettori

- In C89 *num_elementi* ***deve*** essere una *costante* intera positiva (no `const`, no `enum`) nota al *compile-time*, quindi non può essere una variabile richiesta all'utente o un valore calcolato dal programma. Di norma la costante si indica una volta per tutte con una `#define` e nel programma si userà *sempre e solo* questa

Vettori

- In Inglese sono chiamati *array*, in C "array" e "vettore" sono sinonimi, in altri linguaggi (es. in Java) i termini hanno significato diverso
- Si accede ai singoli elementi indicando il nome del vettore seguito dall'indice dell'elemento tra parentesi quadre (valore di tipo intero: costante, variabile o espressione):
  `vett[7]` `vett[i]` `vett[k+1]`
- L'indice deve essere un valore di *tipo* intero (ossia `int`, `short`, ecc. non floating point)
- Il primo elemento ha indice 0: `vett[0]`
- L'ultimo elemento di un vettore di N elementi ha indice N–1, in questo esempio è `vett[9]`

Vettori

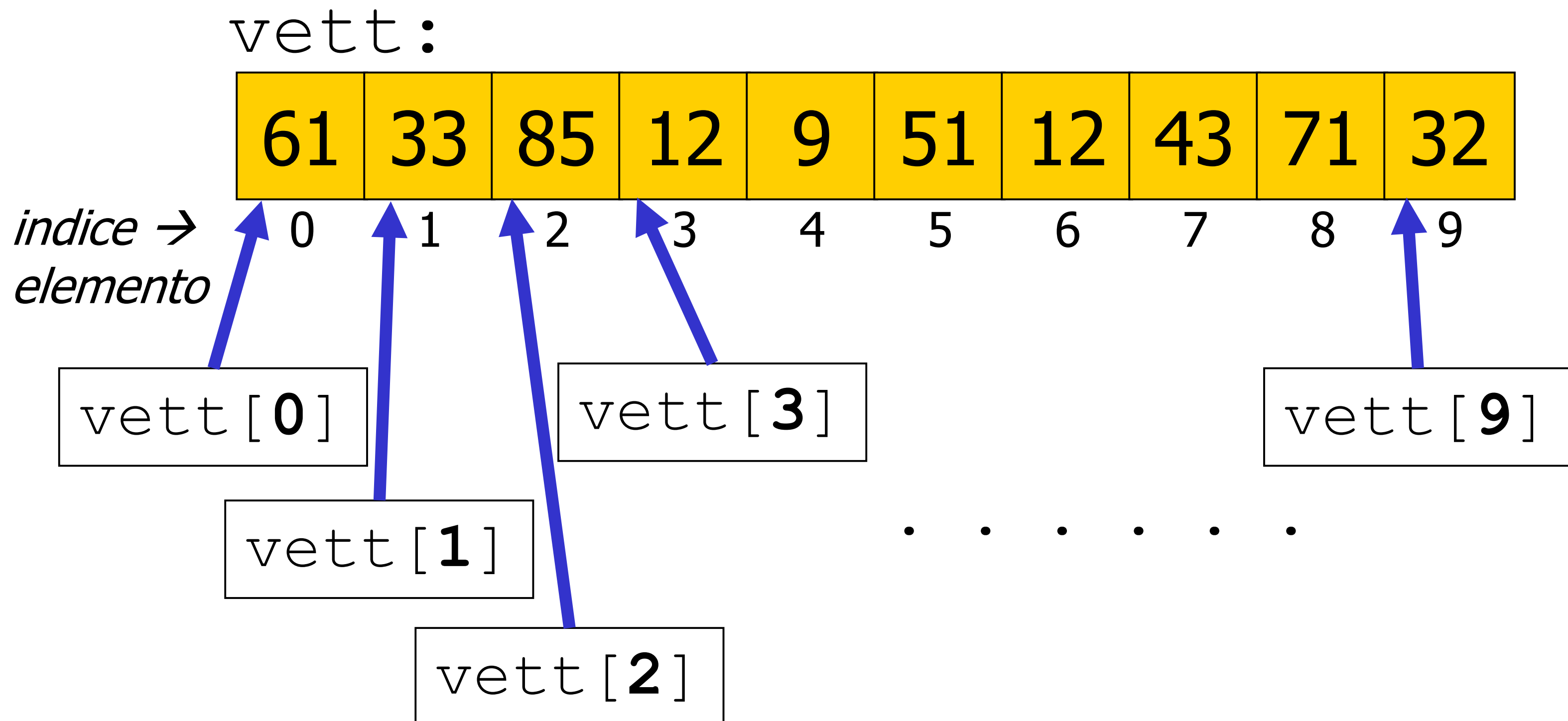
- Poiché gli elementi di un vettore sono del tipo indicato nella definizione (nell'esempio il tipo è `int`), un elemento può essere utilizzato in tutti i contesti in cui si può usare un valore di quel tipo
- Esempi

```
int vett[10];
scanf("%d", &vett[4]);
x = vett[4] * 5;
```

infatti `vett[4]` è un valore (scalare) di tipo `int`

Vettori

- Gli elementi del vettore sono allocati in locazioni di memoria successive e contigue
- `int vett[10];`



Vettori

- Il ***nome*** di un vettore è usato dal compilatore come *sinonimo* dell'indirizzo di memoria del primo elemento del vettore (non essendo una variabile non gli si può assegnare un valore)
- Supponendo che il vettore `vett` sia di interi a 32 bit e inizi in memoria all'indirizzo `A00`, allora `vett[0]` ha indirizzo `A00`, `vett[1]` → `A04`, … `vett[9]`→`A24`; `A28` è il primo byte oltre la fine

`vett:`

61	33	85	12	9	51	12	43	71	32	
0	1	2	3	4	5	6	7	8	9	
A00	A04	A08	A0C	A10	A14	A18	A1C	A20	A24	A28

Indirizzi crescenti →

Vettori

- È possibile scandire efficacemente tutti i valori di un vettore mediante un ciclo `for`
- Questa la soluzione del problema iniziale:

```
#define N 10
int vett[N];
for (i=0; i<N; i++)
   scanf("%d", &vett[i]);
for (i=N-1; i>=0; i--)
   printf("%d\n", vett[i]);
```

memorizzati in quest'ordine

visualizzati in quest'ordine

6	3	8	1	9	1	2	4	1	3

i → 0 1 2 3 4 5 6 7 8 9

Eccedere i limiti del vettore

- Si “sfora” il vettore quando si cerca di accedere a elementi (inesistenti) oltre i limiti del vettore, viene anche detto *buffer overflow*
- Se ad esempio si hanno le definizioni:
  `int vett[4]={1,2,3,4}, x=8;`
  un’assegnazione come `vett[4]=12` fa accedere a una parte di memoria che *potrebbe* essere allocata ad altro (es. `x`), modificandola:

vett:				x:					
1	2	3	4	12					
0	1	2	3	4					
A00	A04	A08	A0C	A10	A14	A18	A1C	A20	A24

Eccedere i limiti del vettore

- Esempio tipico di errore

```
int vett[N];
for (i=1; i<=N; i++)
  scanf("%d", &vett[i]);
```

  Salta l'elemento 0 e accede all'elemento N che non esiste (errore detto di "off-by-one")
- Lo standard non richiede che il compilatore faccia controlli su un eventuale sforamento, molti compilatori lo forniscono opzionalmente, ma ciò riduce le prestazioni in quanto ad ogni accesso al vettore viene fatto un controllo preliminare per verificare che non si sfori

Inizializzazione di vettore

- Si inizializza un vettore con una lista di inizializzatori tra parentesi graffe dopo l' '=':
```
int vett[4]={12, 5, 3, 6};
```
- Non ci può essere un numero di inizializzatori maggiore della lunghezza del vettore
- L'ultimo valore può essere seguito da virgola
- Se un vettore non è inizializzato, i valori in esso contenuti sono indefiniti (possono essere qualsiasi, non è detto siano 0)
- Un vettore inizializzato può essere reso costante (non modificabile) con `const`:
```
const int vett[4]={12, 5, 3, 6};
```

Inizializzazione di vettore

- Gli inizializzatori devono essere noti al momento della compilazione e quindi possono essere solo *espressioni costanti* :
  - possono contenere:
    - numeri
    - `#define` (numeriche)
    - valori di `enum`
    - indirizzi di memoria di variabili *statiche*
  - NON possono contenere:
    - variabili
    - valori `const`
    - risultati di funzioni
    - indirizzi di memoria di variabili *automatiche*

Inizializzazione di vettore

- Se ci sono inizializzatori, la dimensione può essere omessa, viene calcolata dal compilatore contando i valori, ma non c'è modo di ricavare quanti siano a posteriori:
```
int vett[]={12, 5, 3, 6};
```
- Se la lista contiene meno valori di quelli indicati dalla dimensione, vengono inizializzati solo i primi e i successivi *sono inizializzati* a 0:
```
int vett[4]={6, 2};
```
i 4 valori sono 6, 2, 0, 0

Quindi per inizializzare a 0 tutto un vettore:
```
int vettore[10]={0};
```
Il primo valore 0 è indicato esplicitamente, i successivi sono posti a 0 automaticamente

Matrici

- Sono variabili vettoriali con due dimensioni
- Definizione

  tipo ***nome*** [*num_righe*] [*num_colonne*] ;

  l'indice delle righe va da 0 a *num_righe-1*, quello delle colonne da 0 a *num_colonne-1*

  ```
  int Mx[7][5];
  ```

  definisce una matrice di 7 righe × 5 colonne, l'indice delle righe va da 0 a 6, quello delle colonne da 0 a 4, tutti gli elementi sono `int`
- Utilizzo

  ```
  Mx[3][2] = 12;
  scanf("%d", &Mx[i][j]);
  ```

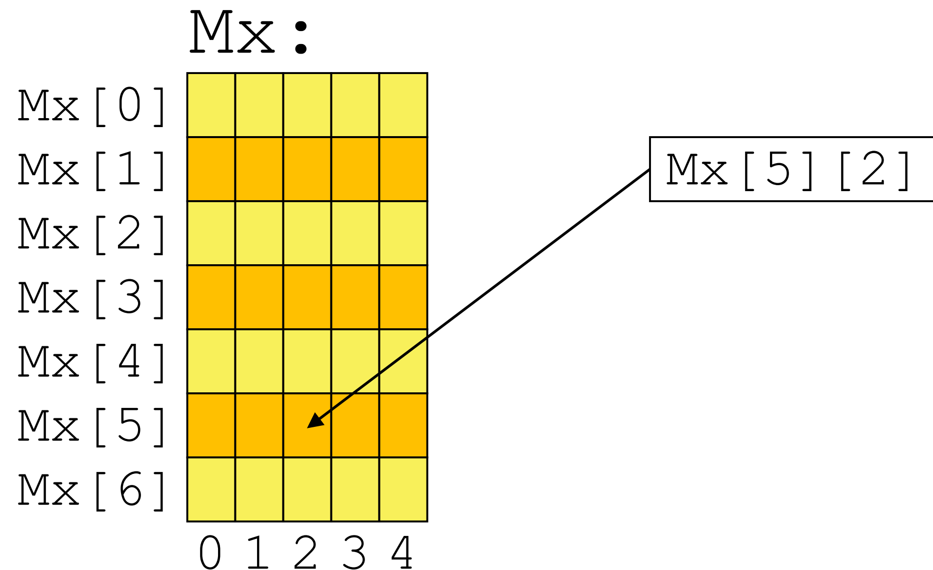
Matrici

- La presenza di 2 indici suggerisce che sia possibile scandire sequenzialmente i valori di una matrice mediante due cicli `for` annidati
- Il codice seguente visualizza una matrice

```
int mat[RIGHE][COLONNE];
…
for (r=0; r<RIGHE; r++)
{
   for (c=0; c<COLONNE; c++)
      printf("%d ", mat[r][c]);
   printf("\n");
}
```

Matrici

- Una matrice è in realtà un *vettore di vettori:*
  `int Mx[7][5];`
  `Mx` è un vettore di 7 elementi (le righe)
  ogni elemento è un vettore di 5 `int`



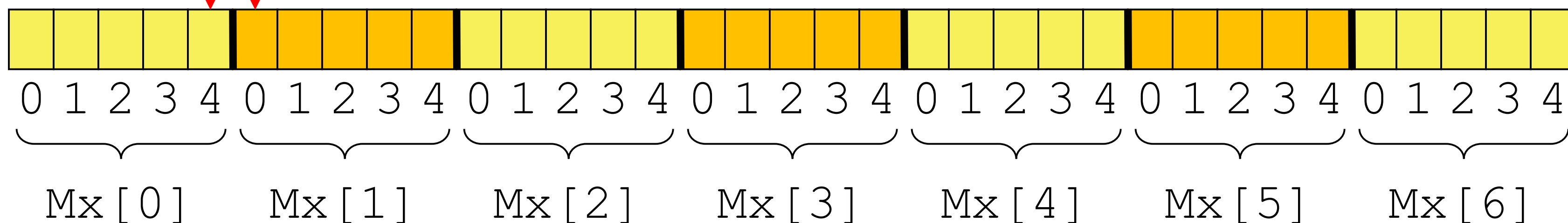
- In generale tutto quanto indicato per i vettori si applica sempre anche alle matrici

Matrici

- `Mx` è composta da 7 elementi identificati con `Mx[`*i*`]` ciascuno dei quali è un vettore di 5 `int` (e non viceversa perché l'associatività delle parentesi quadre va da sinistra a destra)
- Il compilatore considera ciascuno degli `Mx[`*i*`]` come *sinonimo* dell'indirizzo di memoria del corrispondente vettore di 5 `int` (ossia del suo primo elemento); quindi non essendo `Mx[`*i*`]` delle variabili *non* si possono assegnare valori
- Si accede agli elementi indicando i valori di tutti gli indici, es. `Mx[5][2]`

Matrici

- In C gli elementi di una matrice sono memorizzati ***per righe***, questo significa che i 7 vettori di 5 `int` sono collocati uno di seguito all'altro in locazioni di memoria crescenti: gli elementi del primo vettore di 5 `int` sono seguiti dagli elementi del secondo, etc. (dopo `Mx[0][4]` si trova `Mx[1][0]`)



Inizializzazione di matrice

- Se non c'è inizializzazione, i valori contenuti in una matrice (multi-dim.) sono indeterminati
- Gli inizializzatori degli elementi di una matrice si elencano separando con parentesi graffe i valori delle diverse righe

```
int Mx[2][2]={{1,2},{3,4}};
```

- Gli inizializzatori possono anche non avere le graffe interne relative alle singole righe, i valori vengono assegnati ordinatamente agli elementi della matrice riga per riga

```
int Mx[2][2]={1,2,3,4};
```

è sconsigliabile e alcuni compilatori lo segnalano come Warning

Inizializzazione di matrice

- Per inizializzare a 0 tutta una matrice (anche multidimensionale) si può scrivere:
  ```
  int Mx[10][20] ={{0}};
  ```
  `Mx[0][0]` è indicato esplicitamente, i successivi sono posti a 0 automaticamente
- Quando ci sono abbastanza inizializzatori per definire la dimensione effettiva di una matrice, la *prima dimensione* può essere omessa e lasciata determinare al compilatore:
  ```
  int Mx[][2]={{1,2},{3,4}};
  ```
  anche se gli inizializzatori non sono elencati riga per riga (possibile Warning):
  ```
  int Mx[][2]={1,2,3,4};
  ```

Inizializzazione di matrice

- Le graffe sono indispensabili per specificare solo alcuni valori (gli altri sono posti a 0):

```
int Mx[100][4]={{1,2,3},{4,5}};
```

  1ª riga: 1, 2, 3, 0
  2ª riga: 4, 5, 0, 0
  le altre 98 righe: 0, 0, 0, 0

- Senza graffe vengono invece assegnati in sequenza:

```
int Mx[100][4]={1,2,3,4,5};
```

  1ª riga: 1, 2, 3, 4
  2ª riga: 5, 0, 0, 0
  le altre 98 righe: 0, 0, 0, 0

Matrici multidimensionali

- Sono vettori di vettori di vettori di vettori...
- Non c'è limite al numero delle dimensioni
- Esempio

```
int xyz[7][5][4];
```

`xyz[5][2][3]` è una variabile scalare di tipo `int`

```
xyz[5][2][3] = 12;
```

- I 7 elementi `xyz[`*i*`]` (ciascuno dei quali è una matrice 5x4 di `int`) e gli elementi `xyz[`*i*`][`*j*`]` (ciascuno dei quali è un vettore di 4 `int`) non possono essere assegnati in quanto sono sinonimi di indirizzi di memoria e non variabili

Confronto di vettori

- Non è possibile confrontare due vettori (e quindi anche due matrici) usando gli operatori relazionali sui nomi dei vettori stessi
- Bisogna confrontare i singoli elementi
- Esempio (verifica se due vettori sono uguali)

```
int v[N]={…}, w[N]={…};
int uguali = SI;  /* flag */
for (i=0; i<N && uguali==SI; i++)
   if (v[i]!=w[i])
      uguali = NO;
if (uguali)
   printf("Sono uguali");
else printf("Non sono uguali");
```

Confronto di vettori

- Esempio (verifica se due vettori sono uguali, più veloce)

```
int v[N]={…}, w[N]={…};
for (i=0; i<N; i++)
   if (v[i]!=w[i])
      break;
if (i==N)
   printf("Sono uguali");
else
   printf("Non sono uguali");
```

Non c'è il controllo del flag, ma controlla il valore finale di `i`: se è pari a `N` non ha mai eseguito il `break` quindi sono tutti uguali

Copia di vettori

- Non è possibile copiare due vettori (e quindi anche due matrici) usando l'operatore = sui nomi dei vettori stessi
- Bisogna copiare i singoli elementi:

```
int v[M][N]={…}, w[M][N];
for (i=0; i<M; i++)
    for (j=0; j<N; j++)
        w[i][j]=v[i][j]);
```

Confronto e copia di vettori

- Dopo aver visto i puntatori si comprenderà che l'operatore `=' copia gli indirizzi di memoria dei vettori e l'operatore `==' li confronta
- Dopo aver visto le funzioni sui blocchi di memoria (slide sulle stringhe) e l'operatore `sizeof`, si scoprirà che:
  - è possibile copiare un vettore (o una matrice) in un altro avente stesso tipo e stessa dimensione con la funzione:
    ```
    memcpy(a, b, sizeof a);
    ```
  - è possibile verificare se due vettori (o matrici) dello stesso tipo (di base) sono identici con la funzione:
    ```
    memcmp(a, b, sizeof a);
    ```

Operatore sizeof

- L'operatore `sizeof` restituisce il numero di byte di cui è composto un tipo di dato o una variabile (scalare o aggregata), è un valore intero senza segno di tipo `size_t` (definito in `<stddef.h>`), quindi può essere necessario un cast per evitare Warning
- Sintassi
  - `sizeof(`*nome_di_tipo*`)`
  - `sizeof` *nome_di_variabile*

  Le parentesi sono necessarie se si indica il nome di un tipo di dato, facoltative se si indica il nome di una variabile

Operatore sizeof

- Esempio

  ```
  int lung=(int)sizeof(double);
  ```
  dà il numero di byte richiesti da un `double`
- Se si applica a un vettore (`v`), dà la dimensione di quel vettore in byte, da cui è possibile risalire alla sua dimensione:

  ```
  dim=(int)(sizeof(v)/sizeof(v[0]));
  ```
- È un *operatore,* non una funzione, e viene valutato in fase di compilazione, può essere usato in una `#define` ma non in una `#if`
- Nei confronti con interi, per evitare Warning si faccia il cast di uno dei valori

Esercizi

1. Scrivere un programma che chieda quanti valori verranno introdotti dalla tastiera (max 100), li chieda tutti e successivamente li visualizzi dall'ultimo al primo.
2. Scrivere un programma che chieda quanti valori verranno introdotti dalla tastiera (max 100), li chieda tutti e successivamente visualizzi prima tutti i valori pari nell'ordine in cui sono stati inseriti e poi tutti i valori dispari nell'ordine inverso.
   Esempio: dati i valori: 8 1 3 2 8 6 5, il programma visualizza: 8 2 8 6 5 3 1

Esercizi

3. Scrivere un programma che definisca 2 vettori `A` e `B` di uguali dimensioni (la dimensione sia chiesta in input, max 100), chieda in input tutti i valori del primo e successivamente tutti i valori del secondo (devono comparire sul video richieste come le seguenti:
   "`Introdurre il 1° valore di A`",
   "`Introdurre il 2° valore di A`" *ecc.*
   e successivamente
   "`Introdurre il 1° valore di B`",
   "`Introdurre il 2° valore di B`" *ecc.*

Esercizi

(*Continuazione*)

Il programma crea un terzo vettore `C` della stessa dimensione di `A` e `B` contenente nel 1º elemento la somma del 1º elemento di `A` e del 1º elemento di `B`, nel 2º elemento la somma del 2º elemento di `A` e del 2º elemento di `B` etc.

Alla fine deve visualizzare tutti gli elementi di *posizione* (non indice) pari di `C` (il 2º, il 4º,…) e poi tutti quelli di *posizione* dispari (1º, 3º,…).

Esempio: vettori di lunghezza 4, in `A` sono stati messi i valori: `3 5 2 6` e in `B`: `3 2 6 3`, verranno quindi calcolati e messi in `C` i valori: `6 7 8 9` e quindi stampati i valori: `7 9 6 8`.

Esercizi

4. Scrivere un programma che chieda quanti valori verranno introdotti dalla tastiera (max 100), li chieda tutti e li collochi in un vettore. Successivamente, il programma deve determinare il massimo, il minimo, la somma e la media di questi valori.
5. Scrivere un programma che chieda quanti valori (reali) verranno introdotti dalla tastiera (max 100), li chieda tutti, calcoli e stampi la *media mobile* a 3 valori di questi numeri. Si controlli che il numero di valori da introdurre sia almeno pari a 3.

Esercizi

Continuazione

La media mobile è una media aritmetica su solo una parte dei valori (in questo caso 3), ad esempio se viene data la sequenza di valori:

2.1 4.2 1.3 6.7 3.1 5.5 2.1 4.9 3.0 5.4 3.9

il programma deve calcolare la media di 2.1, 4.2 e 1.3 e stamparla, poi la media di 4.2, 1.3 e 6.7 e stamparla, poi 1.3, 6.7 e 3.1 e stamparla, ecc. fino a 3.0, 5.4 e 3.1:

2.1 4.2 1.3 6.7 3.1 5.5 2.1 4.9 3.0 5.4 3.9

2.1 4.2 1.3 6.7 3.1 5.5 2.1 4.9 3.0 5.4 3.9

2.1 4.2 1.3 6.7 3.1 5.5 2.1 4.9 3.0 5.4 3.9

Esercizi

6. Scrivere un programma che acquisisca da tastiera un vettore di (max 100) valori di tipo intero e identifichi la più lunga sequenza di numeri consecutivi uguali. Se vengono identificate più sequenze della stessa lunghezza, si consideri solo la prima identificata. Il programma deve indicare il valore ripetuto e il numero di ripetizioni di quel valore.
   Esempio:
   Input: `19 3 15 15 7 9 9 9 9 12 3 3 3`
   Output: `numero: 9, ripetizioni: 4`

Esercizi

7. Scrivere un programma che definisca una matrice di valori interi e di dimensioni richieste di volta in volta dall'utente (massimo 10x10) mediante input quali "quante righe?" e "quante colonne?". Successivamente di tutti questi valori determini il massimo, il minimo, la somma e la media (frazionaria).

Esercizi

8. Scrivere un programma che definisca una matrice `MX` di valori interi e di dimensioni richieste di volta in volta dall'utente (massimo 20x26) mediante input quali "quante righe?" e "quante colonne?" e immetta in ciascuna cella un valore casuale (usare **`rand`**) compreso tra 0 e 99. Il programma deve poi visualizzare la matrice con i valori allineati correttamente (`%3d` nella `printf`) e contare quanti valori sono pari e quanti sono dispari. *Si tenga separata la parte relativa allo riempimento della matrice dalle operazioni successive.*

Esercizi

9. Definire la matrice `MX` come nel precedente esercizio, questo programma deve indicare se se **almeno un quarto** dei valori della matrice è pari. Non interessa sapere quanti siano i valori pari, ma solo sapere se ce n'è almeno un certo numero noto a priori (*num_righe* * *num_colonne* / 4, ***attenzione: tronca la parte frazionaria***); quindi quando si arriva a contarne esattamente quel numero si può uscire dai due cicli annidati per non perdere tempo inutilmente (è inutile contare anche gli altri). Per uscire da più cicli annidati si può utilizzare un **`goto`**.

Esercizi

10. Scrivere un programma che definisca una matrice quadrata di lato massimo 19, chieda all’utente di indicare la dimensione effettiva del lato (≤19) e la riempia di valori come indicato nell’esempio (a spirale), poi la visualizzi.

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Esercizi

11. Si acquisisca da tastiera una matrice di `int` di dimensioni uguale a `R` righe e `C` colonne (con `R` e `C` costanti predefinite), chieda all'utente le dimensioni `r` e `c` di una matrice tali che `r≤R` e `c≤C`, visualizzi tutte le sottomatrici di dimensioni `r×c` della matrice data la cui somma degli elementi è uguale a zero.
    Esempio con R=4, C=5, r=2, c=2:

-2	-2	4	9	7
-9	13	-5	22	8
16	-9	1	-9	2
3	2	33	2	9

Output prodotto:

```
-2 -2
-9 13

13 -5
-9 1
```

Esercizi

12. Scrivere un programma che definisca una matrice 20x26 di valori `int` casuali (0-99), chieda all'utente di introdurre una matrice (dimensioni e valori) e determini se questa seconda è contenuta nella prima, indicando le coordinate di ogni ricorrenza della stessa.

2	12	4	9	7
21	25	9	22	8
16	11	11	9	22
3	2	33	11	9

9	22
11	9

```
Trovata alle coordinate (1,2)
Trovata alle coordinate (2,3)
```

Esercizi

13. Questo esercizio richiede la conoscenza del prodotto di matrici in matematica.
Si scriva un programma per calcolare il prodotto (righe per colonne) di due matrici.
Le dimensioni delle matrici vengano chieste all'utente (max 10x10 ciascuna). Si controlli che le dimensioni delle matrici siano tali da permettere che esse vengano moltiplicate.
La matrice risultante deve essere visualizzata.