5. QUINTA esercitazione in laboratorio

Attivare il TC e posizionarsi sul proprio dischetto. Ciò può esser fatto usando il comando **Change Dir** del menù **File**, oppure attivando il comando **DOS shell** (stesso menù), spostandosi su A: e inserendo EXIT per tornare al TC.

Le soluzioni proposte durante l'esercitazioni (indicate spesso tra parentesi nei titoli delle varie sezioni) sono reperibili via web, tramite il **sito del cors**o (zona "esercitazioni autoguidate")

5.1. Array e medie (`ARRMED.C, ARRMED2.C`)

Scrivere un programma C il cui Input sia costituito da una sequenza di N numeri frazionari. Ad esempio, N potrebbe valere 6. È bene che N sia un simbolo di costante.

L'Output del programma deve consistere nella stampa del valore medio dei numeri dati in input e nella stampa dei medesimi numeri, incrementati della media appena menzionata.

Per risolvere il problema è indispensabile mantenere in memoria i dati letti da input. Per far ciò si suggerisce di usare un array di N double. Per il momento è anche consigliabile fare a meno della definizione di funzioni (oltre alla mai()!).

5.2. Commercio al minuto (`NEGOZI.C`)

Realizzare un programma che risolva il problema della società commerciale:

Una società amministra 20 negozi (che si immaginano distinti solo in base ad un numero: negozio 7, negozio 12 ...).

Considerando come dati di Input i guadagni mensili dei negozi, il programma deve

- calcolare e stampare la media dei guadagni;
- stampare quali negozi (e con quali guadagni) guadagnano meno di un terzo della media;
- calcolare e stampare la media dei guadagni escludendo il massimo e minimo guadagno (chiamiamola media2);
- stampare quali negozi (e con quali guadagni) guadagnano meno di un terzo della media2, cioè si configurano come le pecore nere del gruppo (e i cui gestori stanno per passare un brutto momento ...).

Anche in questo problema è indispensabile mantenere in memoria i dati letti da input. Per far ciò si suggerisce di usare un array di N double. E anche qui, per il momento è consigliabile fare a meno della definizione di funzioni ulteriori alla mai().

5.3. Gli errori dei commercianti (`NEGOZERR.C`)

Per risolvere il problema di cui al punto precedente abbiamo realizzato un programma in NEGOZERR.C. Il programma contiene degli errori (del programmatore, non dei commecianti). Quali? Fare un po' di attività di debugging, se non si riconoscono gli errori a prima vista ...

5.4. Funzioni con parametri array

Scrivere una funzione C `accumulazione()` che, ricevuto un array di N double, calcola e restituisce la somma dei suoi elementi. Fare una copia del file ARRMED.C e incastonare nel programma la funzione `accumulazione()`usandola per il calcolo della media dei valori letti da input e per la successiva ristampa dei medesimi valori incrementati di media.

5.5. Funzioni con parametri array – seconda parte (NEGFUN1.C)

Scrivere le seguenti funzioni C:

- `massimo()` che, ricevuto un array di N double, calcola e restituisce il valore del massimo dei suoi elementi;
- `minimo()` che, ricevuto un array di N double, calcola e restituisce il valore del minimo dei suoi elementi;

e usarle nella risoluzione del problema dei guadagni dei negozi.

5.6. Funzioni con parametri array – terza parte (NEGFUN2.C)

Scrivere le seguenti funzioni C:

- `accumulazione()` che, ricevuto un array di N double, calcola e restituisce il valore del massimo dei suoi elementi (questa l'abbiamo già vista ...);
- `stampaNegoziScarsi()` che, ricevuti
  - un array di N double, `g[]`, che rappresenta i guadagni dei negozi,
  - e un numero double, `soglia`

  stampi il numero d'ordine e il guadagno dei negozi che guadagnano meno di `soglia`

Usar le suddette funzioni (insieme anche a quelle utilizzate nell'esercizio precedente) nella risoluzione del problema dei guadagni dei negozi.

In particolare `stampaNegoziScarsi` sarà una funzione `void` (dato che deve solo prosurre delle stampe e non deve "restituire valori particolari). Essa verrà chiamata due volte: una volta per stampare i negozi che guadagnano meno di un terzo di `media` e un'altra per stampare i negozi che guadagnano meno di un terzo di `media2`.

5.7. Funzioni con parametri array – quarta parte (NEGFUN3.C)

Scrivere una funzione C

`lettura()` che, ricevuto un array `g[]` di N double, legga N valori da tastiera e li memorizzi nell'array

Usare questa funzione per scrivere un'altra versione del programma che risolve il problema dei guadagni dei negozi. In quest'ultima versione, si useranno anche tutte le altre funzioni definite nei due esercizi precedenti.

5.8. Array bidimensionali

Quel che si è visto finora, è che un array è una collezione di un certo numero N di variabili di un certo tipo.

Le variabili sono allocate (hanno spazio riservato in memoria) sequenzialmente in un blocco di memoria centrale.

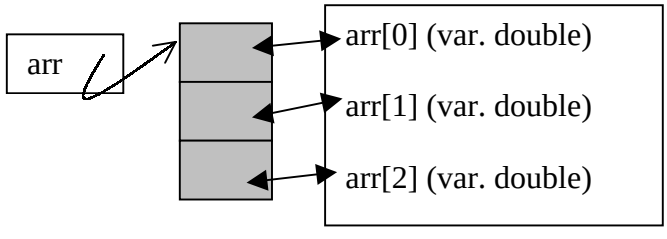
Il nome dell'array indica l'indirizzo dell'inizio del blocco.

Ogni variabile è identificata da un indice (intero da 0 a N-1).

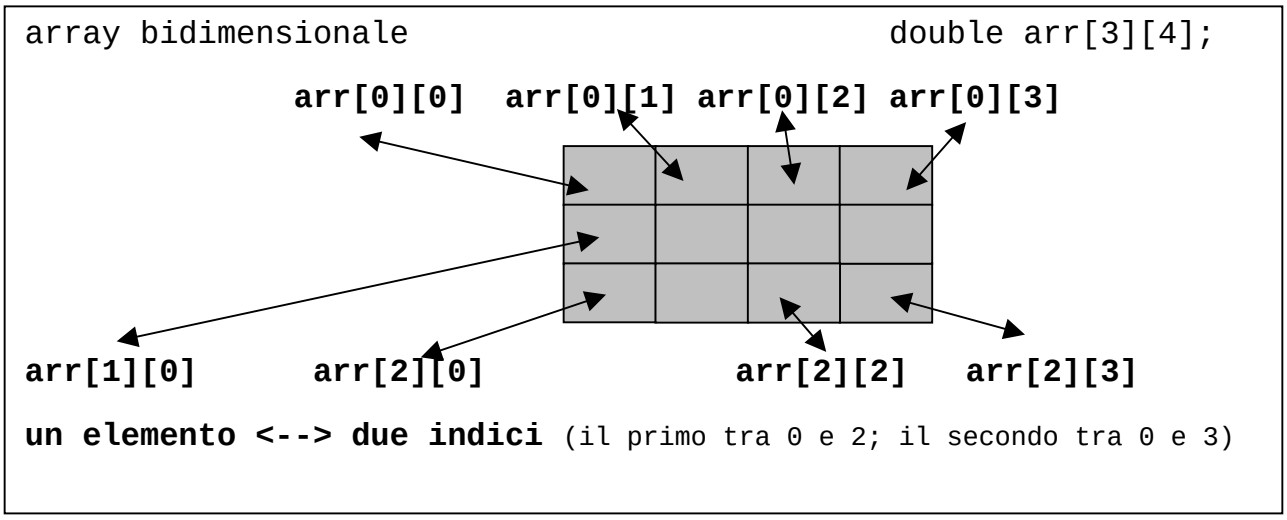
```
array monodimensionale


double arr[3];

un elemento <--> un indice
tra 0 e 2
```



Si è visto poi, sul libro e/o a lezione, che è possibile definire collezioni di elementi in cui ciascun elemento è associato a una coppia di indici; queste strutture dati sono sempre array, solo che si chiamano array bi-dimensionali.



Con questo costrutto si rappresentano agevolmente le matrici. In un array bidimensionale è infatti facile riconoscere una organizzazione basata su "righe" e "colonne". Nella matrice di esempio,

arr[0][0], arr[0][1], arr[0][2], arr[0][3] sono gli elementi della **riga 0;**

arr[1][0], arr[1][1], arr[1][2], arr[1][3] sono gli elementi della **riga 1;**

arr[2][0], arr[2][1], arr[2][2], arr[2][3] sono gli elementi della **riga 2;**

5.10. Esercizio (MATSOMM1.C)

Scrivere un programma che fa uso di tre matrici di NxM elementi interi (porre N ed M a numeri sufficientemente piccoli – 3, 4 – altrimenti si dovranno immettere troppi dati da tastiera!).

Il programma deve leggere due matrici da input, stamparle, calcolare la somma delle due matrici (memorizzandola nell aterza matrice) e stamparla.

Suggerimento: intanto non usare funzioni. Scrivere solo la definizione della funzione main() e delle costanti N ed M; inoltre, se è definito l'array **`int mat1 [N][M]`**,

- iterando per j=0...M-1 l'istruzione **`scanf("%d", &mat1[0][j])`** si leggono gli M elementi della prima riga di `mat1` (riga 0);
- iterando per j=0...M-1 l'istruzione **`scanf("%d", &mat1[1][j])`** si leggono gli M elementi della seconda riga di `mat1`;
- ...
- iterando per j=0...M-1 l'istruzione **`scanf("%d", &mat1[N-1][j])`** si leggono gli M elementi dell'ultima riga di `mat1`;

Quindi il ciclo

```
for (j=0; j<M; j++)
   scanf("%d", &mat1[i][j])
```

e questa lettura della riga i-esima va ripetuta per i valori di i da 0 a N-1, in modo da leggere, una riga per volta, tutta la matrice.

5.11. Esercizio (MATSOMM2.C)

Identico esercizio del punto precedente! Però adesso bisogna definire tutte le funzioni necessarie per far funzionare il programma con la seguente funzione main():

```
int main ()
{
      ...
   printf(" lettura prima matrice:\n");
   leggiMatrice(mat1);

   printf(" lettura secnda matrice:\n");
   leggiMatrice(mat2);

   sommaMatrici(mat1, mat2, mat3);

   printf(" prima matrice:\n");
   stampaMatrice(mat1);

   printf(" seconda matrice:\n");
   stampaMatrice(mat2);

   printf(" matrice somma:\n");
   stampaMatrice(mat3);
  ...
}
```

5.12. Condensazione delle righe di una matrice (CONDENS.C)

Scrivere una funzione condensa() che, ricevuti un array arr e una matrice di NxM elementi double, riempie l'array in modo che l'elemento arr[i] corrisponda, per ogni i, alla somma della riga i-esima della matrice.
Per collaudare la funzione, inserirla nel file TCONDENS.C, in cui c'è già un programma capace di usarla.

5.13. Condensazione delle colonne (!) di una matrice (CONDCOL.C)

Scrivere una funzione condensaColonne() che, ricevuti un array arr e una matrice di NxM elementi double, riempie l'array in modo che l'elemento arr[k] corrisponda, per ogni k, alla somma della **colonna** k-esima della matrice.
Scrivere anche un intero programma per collaudarla. (Si può sfruttare pesantemente il programma scritto per l'esercizio precedente (con qualche modifica alla funzione di stampa del vettore ... e con l'aggiunta della funzione qui richiesta).

Suggerimento: se mat è un array bidimensionale NxM, gli elementi della sua colonna j-esima sono

```
mat[0][j], mat[1][j], ..., mat[N][j]
```

quindi lo schema per scandire, fissato j, tutti gli elementi della colonna j è del tipo

```
for (i=0; i<N; i++)
   processazione di mat[i][j]
```

5.14. Prodotto tra matrici(PRODMAT.C)

Scrivere una funzione prodMat() che, ricevuti due array bidimensionali NxN restituisce, in un opportuno parametro di output, il prodotto tra le due corrispondenti matrici. Per collaudare la funzione, inserirla nel file TPRODMAT.C, in cui c'è già un programma capace di usarla.

Suggerimento: se (a_{ij}) e (b_{ij}) sono due matrici (i, j = 1..N), la matrice prodotto è (c_{ij}) con per ogni i, j = 1...N

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}$$

Analogamente, se abbiamo definito tre array bidimensionali ch rappresentano le medesime matrici,

```
int prima[N][N], int seconda[N][N], int result[N][N]
```

considerando che, in questi array, gli indici andranno da 0 a N-1,

l'elemento generico della matrice risultato sara'

```
result[i][j] =      somma dei prodotti
                    prima[i][k]* seconda[k][j]
                    con k = 0... N-1
```

quindi lo schema risolutivo sarà

```
            for (i=0; i<N; i++)
               for (i=0; i<N; i++)
                  calcolo e assegnazione di result[i][j]
```

dove
`calcolo e assegnazione di result[i][j]`
corrisponde all'accumulazione di prodotti
`prima[i][k]* seconda[k][j]`con k da 0 a N-1

5.15. Centraline inquinate (MISUR.C)

Bisogna monitorare lo stato di inquinamento di Via Salaria, approssimativamente davanti al n. 113. Lì potrebbe venir posta una centralina che misura e memorizza la quantità di idrocarburi incombusti presente nell'aria (in microgrammi).
Vengono fatte misurazioni per tutto il giorno e, alla fine, viene stampato un istogramma che descrive, ora per ora, la media di microgrammi presenti. La media, se non e' un intero esatto, viene troncata.
Ogni ora vengono eseguite 6 misurazioni; la media di queste misurazioni e' la media orari di quell'ora.
Quando una opportuna struttura di memoria e' stata riempita con le medie orarie calcolate nel giorno, viene stampato un istogramma come il seguente:

```
da  0:00 a  0:59 ******************
da  1:00 a  1:59 ************************
da  2:00 a  2:59 ************************
...
da 17:00 a 17:59 ****************************
da 18:00 a 18:59 ******************************
...
```

Assumiamo che il numero di microgrammi misurabile ogni volta vada da 0 a 61 (così le righe dell'istogramma non saranno mai troppo lunghe.

Suggerimenti:

Usare un array di 24 componenti intere;

Ogni 6 misurazioni, la media viene messa in uno degli elementi dell'array (troncata).

Definire una funzione misurazione() che, quando viene chiamata, restituisca una misurazione. Questa funzione può consistere in una semplice lettura da input di un intero, oppure nella determinazione di un valore calcolato li' per li' in qualche modo.

Un modo che suggeriamo e' di fare uso della funzione rand() della libreria stdlib.h.
Una chiamata rand() produce un valore intero psudocasuale.
Analogamente rand() % k produce un valore pseudocasuale compreso tra 0 e k.
(ricordiamoci che nel nostro caso vogliamo numero da 0 a 61...)

Per inizializzare il meccanismo di generazione pseudocasuale dei numeri da parte di rand() bisogna aver chiamato almeno una volta, prima della prima chiamata di rand(), la dunzione srand(), che prende un parametro unsigned int. Un modo carino è di eseguire la chiamata
srand(clock())

La chiamata clock(), dopo un accesso a sistema operativo, restituisce il numero di "colpi di clock" eseguiti dal calcolatore dal momento dell'accensione. Quindi è un buon numero imprevedibile con cui inizializzare il processo di generazione pseudocasuale.
Per usare clock() bisogna aver incluso la libreria time.h.

Ulteriori info su rand(), srand() e clock() sono nell'help del TurboC ...

Nel programma che e' disponibile come soluzione, sono definite le costanti simboliche

```
#define PUNTO '*'
#define ESTREMO 61
#define DIM 24
```

la funzione misurazione() viene definita cosi'

```
int misurazione(int estremo) {
    return(rand() % estremo);
}
```

e usata cosi'

```
...
     for (j=0; j<6; j++) {        /* lettura di 6 dati nell'ora */

        dato = misurazione(ESTREMO);    /* lettura */
     ...
...
```

Infine, per stampare l'istogramma, viene usata una funzione

```
void stampaIstogramma (int arr[], char c)
```

che per ogni componente di arr, arr[i],
stampa una linea di caratteri c lunga arr[i]

oppure viene usata una funzione

```
void stampaIstogramma2 (int arr[], char c, int inizio, int fine)
```

che per ogni componente di arr, arr[i], con i compreso tra inizio e fine,
stampa una linea di caratteri c lunga arr[i].

(Perche' e' necessario ideare anche la seconda versione della stampa istogramma?

perché con laprima versione, quando viene stampato l'istogramma in output, e' molto probabile che una larga parte iniziale dell'istogramma scompaia mentre la parte restante viene stampata. La finestra di output non ci consente di vedere molte linee ...

Con stampaIstogramma2() è possibile risolvere parzialmente il problema: se la chiamata contiene l'indicazione di un intervallo di misurazioni non troppo esteso, tutto l'istogramma prodotto nella finestra di output rimarrà visibile – che bello!).