LABORATORIO DI FONDAMENTI DI INFORMATICA
26 OTTOBRE 2020 - Incontro 3 di 5 – C Base, Vettori, Struct

Esercizio 1

La Tavola Pitagorica è una matrice bidimensionale i cui elementi sono numeri interi calcolati moltiplicando l'indice della riga con l'indice della colonna alle quali l'elemento appartiene.
Scrivere un programma che stampi a video una Tavola Pitagorica di dimensioni 10 x 10 in modo che su ciascuna riga sia presente la "tabellina" di un numero. La matrice è simmetrica per la proprietà commutativa dell'operazione di moltiplicazione.
Gli elementi della matrice che risultano essere multipli sia di 5 che di 7 (contemporaneamente, ad esempio 35, 70, ...) devono essere evidenziati affiancandoli con un asterisco (`*`).

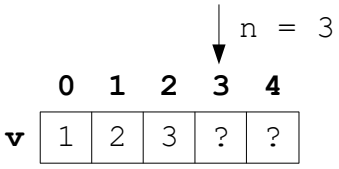
```
Tavola Pitagorica 10 x 10:
       1    2    3    4    5    6    7    8    9   10
-------------------------------------------------------
  1|   1    2    3    4    5    6    7    8    9   10
  2|   2    4    6    8   10   12   14   16   18   20
  3|   3    6    9   12   15   18   21   24   27   30
  4|   4    8   12   16   20   24   28   32   36   40
  5|   5   10   15   20   25   30   35*  40   45   50
  6|   6   12   18   24   30   36   42   48   54   60
  7|   7   14   21   28   35*  42   49   56   63   70*
  8|   8   16   24   32   40   48   56   64   72   80
  9|   9   18   27   36   45   54   63   72   81   90
 10|  10   20   30   40   50   60   70*  80   90  100
```

Suggerimenti:

- non è necessario utilizzare i vettori o le matrici (stampare direttamente a video gli elementi)
- definire la dimensione 10 con una costante `DIM_PIT`: `#define DIM_PIT 10`
- non è obbligatorio allineare esattamente i numeri in colonna e inserire gli indici di riga e colonna durante la stampa, ma se si desidera farlo si ricorda che per stampare un numero intero `i` occupando ad esempio 4 “spazi” la sintassi è: `printf("%4d", i);`

Esercizio 2

Definire un vettore `v` di numeri interi che possa contenere al massimo `#define DIMV 5` numeri e un indice intero `n` che, inizialmente posto uguale a `0`, svolga la duplice funzione di indicare in un dato istante quale sia la prima cella disponibile per inserire un nuovo numero a partire dalla cella 0 in poi e il conteggio di quanti numeri siano presenti nel vettore.
Domandare all’utente i numeri da inserire uno per volta nel vettore fino a che l’utente fornisce il numero 0 (da inserire anch'esso nel vettore) oppure non vi sono più celle libere. Stampare infine a video gli `n` numeri inseriti nel vettore.



Vettore v in un istante in cui contiene n = 3 numeri e la prima cella libera è la [3].

```
Il vettore contiene n=0 numeri. Numero per cella 0? 1
…
Il vettore contiene n=4 numeri. Numero per cella 4? 5
Il vettore contiene n=5 numeri. Il vettore e’ pieno.
Vettore con n=5 numeri: 1 2 3 4 5
```

Esercizio 3
Scrivere un programma che chieda all'utente `DIM1` = 5 numeri interi distinti da inserire in un primo vettore e quindi altri `DIM2` = 4 numeri interi distinti da inserire in un secondo vettore (`DIM1` e `DIM2` sono due costanti da definire con la direttiva `#define`).
Il programma deve stampare a video il contenuto di entrambi i vettori e poi trovare e stampare a video i numeri che sono presenti in entrambi i vettori (l'intersezione).

```
vettore1: 1 4 5 6 7
vettore2: 4 5 9 1
intersezione: 1 4 5
```

Suggerimenti:

- realizzare un primo ciclo che iteri sugli elementi del primo vettore e al suo interno un secondo ciclo che confronti l'attuale elemento del primo vettore con gli elementi del secondo vettore
- si può supporre che sia l'utente a inserire numeri sempre distinti oppure si può implementare un controllo che richieda l'inserimento se un numero è già stato inserito.

Esercizio 4
Definita la costante `DIM_M` = 3, scrivere un programma che chieda all'utente `DIM_M*DIM_M` numeri interi e li inserisca in una matrice `DIM_M x DIM_M` rappresentata tramite un vettore bidimensionale. Dopo l'inserimento il programma deve moltiplicare per 2, cioè raddoppiare, ciascun numero presente nella matrice e ristampare a video la matrice "raddoppiata".

```
matrice[0][0]? 0
...
matrice[2][2]? 9

Matrice inserita:     |    Matrice raddoppiata:
 0  1  2              |     0  2  4
 3  4  5              |     6  8 10
 6  7  8              |    12 14 16
```

Esercizio 5
Scrivere un programma che chieda un nome all'utente di massimo 10 caratteri e lo inserisca in una stringa chiamata `nome`. Il programma deve poi calcolare la lunghezza in caratteri del nome, "contando" le celle del vettore partendo dalla cella 0 fino a trovare quella che contiene il carattere terminatore `'\0'`. Stampare a video la lunghezza calcolata confrontandola con quella che viene restituita dalla funzione `strlen`.

0	1	2	3	4	5	6	7	8	9	10
J	o	h	n	\0						

```
Qual e' il tuo nome? John
Ciao John!
Ho calcolato che la lunghezza del tuo nome e' di 4 caratteri.
Ok: la lunghezza calcolata cercando '\0'=4 coincide con quella
calcolata con strlen=4.
```

Suggerimenti:

- nel linguaggio C una stringa di testo viene rappresentata tramite un vettore di `char`. Per segnalare dove il testo finisce viene utilizzato il carattere terminatore `'\0'`. Quando si dimensiona il vettore, occorre sempre prevedere lo spazio anche per il carattere terminatore.
- il "segnaposto" per le stringhe da usare con `scanf` e `printf` è `%s`
- per utilizzare la funzione `strlen` occorre la direttiva:
  `#include <string.h>`

...e per ottenere la `lunghezza` di una `stringa` la si richiama in questo modo:

```
lunghezza = strlen(stringa);
```

- la funzione `scanf("%s", nome)` inserisce nella variabile `nome` il testo immesso dall'utente fino al primo spazio. Se si vogliono prevedere stringhe con spazi, si può utilizzare la funzione `gets(nome)`. Entrambe le funzioni aggiungono in automatico nel vettore il carattere terminatore `'\0'`. Entrambe sono "non sicure" perché consentono all'utente di inserire testi di lunghezza superiore a quella supportata dal vettore.
- `scanf` in questo caso non richiede di anteporre l'operatore `&` davanti al nome del vettore perché esso già rappresenta l'indirizzo in memoria in cui `scanf` deve scrivere l'input

```
scanf("%s", &nome);
```

- per stampare con `printf` un testo contenente delle virgolette `"` o un backslash `\` occorre anteporre il carattere di backslash:

```
printf("terminatore '\\0' \"fine\" ");
```

Esercizio 6

Scrivere un programma che, definiti 3 vettori di numeri interi di dimensione massima `MAX` = 10, chieda all'utente quante celle `n` (con 1 <= `n` <= 10) voglia effettivamente usare. Il programma deve poi riempire i primi due vettori a partire dalla cella 0 con `n` numeri da 0 a 9 recuperati attraverso il generatore di numeri pseudo-casuali.
Nel terzo vettore il programma deve memorizzare per poi stampare a video la somma dei valori contenuti nelle celle "speculari" dei primi due vettori (la prima cella del primo vettore con l'ultima "significativa" del secondo, ecc.), nella stessa posizione utilizzata sul primo vettore.

```
Lunghezza effettiva (1-10)? 4
vettore1:  7  9  3  8

vettore2:  0  2  4  8
vettore3: 15 13  5  8
```

Suggerimenti:

- per utilizzare il generatore di numeri occorre includere le seguenti librerie:

```
#include <stdlib.h>
#include <time.h>
```

- per generare numeri sempre diversi occorre inserire a inizio programma l'istruzione:

```
srand(time(NULL));
```

- i numeri da 0 a 9 vengono generati tramite l'istruzione `rand()`:

```
vettore1[i] = rand() % 10;
```

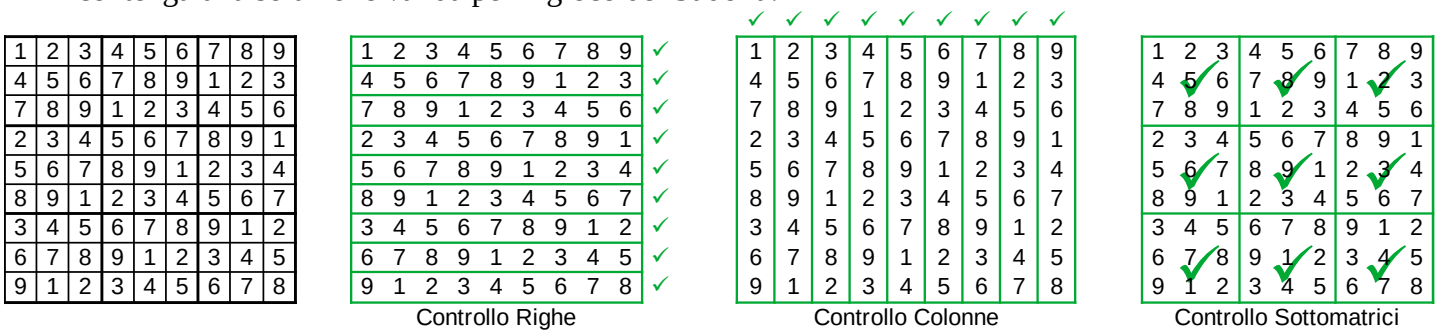
Esercizio 7

Definita una costante `DIMR` pari a 3, scrivere un programma che riempia una matrice `DIMR` x `DIMR` di numeri interi tramite il generatore di numeri pseudo-casuali (vedere l'esercizio precedente).
Tale matrice va replicata 4 volte in una seconda matrice (`DIMR` x 2) x (`DIMR` x 2), una volta per ciascun quadrante della seconda matrice. La replica deve essere però speculare (con ribaltamento orizzontale e verticale rispetto alla mezzeria della matrice – vedere la figura sottostante).

```
7 9 3      7 9 3 3 9 7
8 0 2  ->  8 0 2 2 0 8
4 8 3      4 8 3 3 8 4
           4 8 3 3 8 4
           8 0 2 2 0 8
           7 9 3 3 9 7
```

Esercizio 8

Scrivere un programma che controlli una matrice quadrata con dimensione 9 x 9 per verificare se contenga una soluzione valida per il gioco del Sudoku.



Utilizzare il seguente assegnamento per creare la matrice (fare "copia e incolla" nel sorgente):

```
int sudoku[9][9]= {
      {1,2,3,4,5,6,7,8,9},
      {4,5,6,7,8,9,1,2,3},
      {7,8,9,1,2,3,4,5,6},
      {2,3,4,5,6,7,8,9,1},
      {5,6,7,8,9,1,2,3,4},
      {8,9,1,2,3,4,5,6,7},
      {3,4,5,6,7,8,9,1,2},
      {6,7,8,9,1,2,3,4,5},
      {9,1,2,3,4,5,6,7,8}
};
```

```
Verifica righe: 1... 2... 3... 4... 5... 6... 7... 8... 9...
Verifica colonne: 1... 2... 3... 4... 5... 6... 7... 8... 9...
Verifica sottomatrici: [1][1]-[3][3]... [4][1]-[6][3]... [7][1]-
[9][3]... [1][4]-[3][6]... [4][4]-[6][6]... [7][4]-[9][6]... [1]
[7]-[3][9]... [4][7]-[6][9]... [7][7]-[9][9]...
Soluzione del Sudoku valida!
```

Suggerimenti:

- occorre controllare che:
  - su tutte le righe ci siano i numeri da 1 a 9
  - su tutte le colonne ci siano i numeri da 1 a 9
  - in tutte le sottomatrici 3 x 3 che si ottengono dividendo la matrice in 9 quadranti ci siano i numeri da 1 a 9
- provare a modificare un numero nella matrice originale per vedere se l'errore viene rilevato

Esercizio 9

Definire un tipo di dati strutturato denominato "`s_anagr`" per contenere i dati anagrafici di una persona, cioè il `nome` di massimo `NAMELEN` = 20 caratteri (+ 1 per il terminatore `'\0'`), il `genere` (caratteri `'M'` e `'F'`) e l'`età`.

Creare poi un vettore di `NANAGR` = 5 anagrafiche denominato `v_anagr` :

```
s_anagr v_anagr[NANAGR] = {
      {"Topolino", 'M', 47}
      ,{"Minni", 'F', 37}
      ,{"Pluto", 'M', 17}
      ,{"Clarabella", 'F', 27}
```

```
        ,{"Pippo", 'M', 26}
    };
```

Una volta popolato il vettore `v_anagr` occorre creare e stampare a video un nuovo vettore denominato "`v_finale`" di massimo `NANAGR` + 1 anagrafiche contenente:

- nella cella 0 la persona di genere maschile più giovane contenuta in `v_anagr`
- nella cella 1 la persona di genere femminile più giovane contenuta in `v_anagr`
- nelle celle dalla 2 in poi tutte le persone di genere maschile con età superiore ai 25 anni contenute in `v_anagr`

```
Elenco anagrafiche:
"Topolino", 'M', 47
"Minni", 'F', 37
"Pluto", 'M', 17
"Clarabella", 'F', 27
"Pippo", 'M', 26
Cella 0: "Pluto", 'M', 17 (maschio piu' giovane)
Cella 1: "Clarabella", 'F', 27 (femmina piu' giovane)
Cella 2: "Topolino", 'M', 47 (maschio con piu' di 25 anni)
Cella 3: "Pippo", 'M', 26 (maschio con piu' di 25 anni)
```

Suggerimenti:

- per definire il tipo `s_anagr`:

```
typedef struct {
    char nome[NAMELEN];
    char genere;
    int eta;
} s_anagr;
```

- per accedere al campo di una struttura si usa l'operatore punto "`.`"

```
v_finale[i].eta
```

- non funziona l'assegnamento diretto di una stringa

```
v_anagr[i].nome = "Topolino"; //NO!
```

  occorre invece utilizzare la funzione `strcpy(destinazione, sorgente)` che si trova in `<string.h>`

```
strcpy(v_anagr[i].nome, "Topolino");
```

- per confrontare due stringhe non si può usare l'operatore di confronto ==

```
if(stringa1 == stringa2) ... //NO!
```

  occorre invece utilizzare la funzione `strcmp`

```
if(strcmp(stringa1, stringa2) == 0) ...
```

- se due variabili hanno la stessa struttura è possibile eseguire un assegnamento diretto per copiare i dati da una all'altra (altrimenti occorre copiare tutti i campi uno per uno)

```
v_finale[i] = v_anagr[j];
```

- non è possibile verificare direttamente se due variabili struct contengano gli stessi dati

```
if (v_finale[0] == v_finale[2]) … //NO!
```

  ma occorre invece confrontare tutti i campi uno per uno

```
if (v_finale[0].eta == v_finale[2].eta &&
v_finale[0].genere == v_finale[2].genere &&
strcmp(v_finale[0].nome, v_finale[2].nome) == 0) ...
```

- non serve con le persone fornite come esempio, ma in caso si voglia prevedere il caso che NON esista una persona più giovane da inserire nelle celle 0 e 1 (se ci sono solo maschi o solo femmine) è possibile inserire al suo posto un'anagrafica "vuota" che abbia come `nome` `"N/A"` e come `età` -1.

Esercizio 10

Scrivere un programma che riempia un vettore di `NMAX` 100 numeri interi pseudo-casuali compresi tra 0 e 99 (utilizzare il generatore di numeri pseudo-casuali con `rand() % 100`).

Il programma deve stampare il contenuto di un secondo vettore di dieci elementi che deve contenere

- nella cella 0 quanti numeri del primo vettore sono compresi tra 0..9
- nella cella 1 quanti numeri del primo vettore sono compresi tra 10..19
- nella cella 2 quanti numeri del primo vettore sono compresi tra 20..29
- …
- nella cella 9 quanti numeri del primo vettore sono compresi tra 90..99

```
(Risultato con NMAX = 10 invece che 100)
7 49 73 58 30 72 44 78 23 9
 0.. 9=2  10..19=0  20..29=1  30..39=1  40..49=2  50..59=1
60..69=0  70..79=3  80..89=0  90..99=0
```

Suggerimenti:

- si noti che il risultato (intero) di (X / 10) rappresenta la decina di appartenenza di un numero X. Ad esempio con X = 75 si ha che 75 / 10 = 7 quindi si può ottenere il numero della cella del secondo vettore dove aumentare il conteggio di quanti numeri sono compresi tra 70..79. Grazie a questo, si può evitare di scrivere 10 condizioni diverse, una per ogni decina.

Esercizio 11

Scrivere un programma che chieda all'utente un numero intero `num` >= 0 e stampi a video le sue cifre in lettere. Ad esempio, se `num` = 9876, il risultato sarà "`nove otto sette sei`".

Suggerimenti:

- con un primo ciclo calcolare la prima `potenza` di 10 > `n` (es 10^4 = 10000 > 9876). L'`esponente` da dare a 10 per ottenere tale `potenza` rappresenta il numero di cifre di cui è composto il numero (es 4 per 9876) e guiderà il numero di iterazioni del ciclo descritto di seguito.
- in un altro ciclo occorre dividere `num` per `potenza` ottenendo così la cifra da stampare in lettere (con `printf` in un costrutto `switch`) e all'iterazione successiva ripetere l'operazione sul resto della divisione intera di `num` per `potenza` usando una `potenza` più piccola.

```
num (>=0)? 9876
Cerco la prima potenza di 10 > 9876...
10 <= 9876...  100 <= 9876...  1000 <= 9876...
10000 > 9876 -> trovato esponente = 4
Stampo le cifre dividendo per le potenze di 10 e tenendo il resto...
9876 / 1000 = 9 con resto = 876 -> "nove"
876 / 100 = 8 con resto = 76 -> "otto"
76 / 10 = 7 con resto = 6 -> "sette"
6 / 1 = 6 con resto = 0 -> "sei"
```

Esercizio 12

Scrivere un programma che simuli con un grafico il moto rettilineo uniformemente accelerato di un oggetto nel vuoto. Per la simulazione occorre impostare i seguenti parametri iniziali:

- la velocità iniziale dell'oggetto in metri al secondo **`v0_ms`** (1.0 m/s ovvero 3.6 km/h)
- l'accelerazione costante in metri al secondo^2 **`a_ms2`** (0.1 m/s^2)
- la distanza totale che l'oggetto deve percorrere in metri **`s_tot_m`** (100 m ovvero 0.1 km)
- la lunghezza del grafico di simulazione in numero di caratteri **`LEN_CHAR`** (50 caratteri)
- l'intervallo in secondi **`T_STEP_S`** che trascorre a ogni "passo" della simulazione (10 s)

La formula per calcolare la velocità (**`v_ms`** in metri/secondo) ad un certo istante (**`t_s`** in secondi) data la velocità iniziale (**`v0_ms`** in metri/secondo) e l'accelerazione costante (**`a_ms2`** in metri/secondo^2) è la seguente:

```
v_ms = v0_ms + a_ms2 * t_s;
```

La formula per calcolare la distanza percorsa (**s_m** in metri) ad un certo istante (**t_s** in secondi) data l'accelerazione (**a_ms2** in metri/secondo^2) e la velocità iniziale (**v0_ms** in metri/secondo) è la seguente:

```
s_m = 1.0 / 2.0 * a_ms2 * (t_s * t_s) + v0_ms * t_s;
```

La formula della proporzione per calcolare la distanza (**s_char** in numero di caratteri) dell'oggetto su una riga lunga **LEN_CHAR** caratteri data la distanza percorsa (**s_m** in metri) per stampare a video la riga che rappresenta il "grafico" della simulazione è la seguente:

```
s_char = s_m * (LEN_CHAR / s_tot_m);
```

Per stampare il "grafico" occorre quindi stampare una riga di **LEN_CHAR** caratteri (in posizione **s_char** sarà presente il carattere 'X' che rappresenta il punto in cui si trova l'oggetto mentre nelle altre posizioni verrà stampato il carattere '-').

La simulazione termina quando l'oggetto percorre tutta la distanza ovvero quando **s_m** > **s_tot_m**.

NB: velocità in km/h = velocità in m/s * 3.6.

```
INIZIO SIMULAZIONE v0_ms=1.00  a_ms2=0.10  s_tot_m=100.00  step_s=10.0
Tempo = 0.0 s  Velocita' = 1.000 m/s = 3.60 km/h  Distanza = 0.00 m = 0.000 km
0 X---------------------------------------------- 100.00 m
Tempo = 10.0 s  Velocita' = 2.000 m/s = 7.20 km/h  Distanza = 15.00 m = 0.015 km
0 -------X------------------------------------- 100.00 m
Tempo = 20.0 s  Velocita' = 3.000 m/s = 10.80 km/h  Distanza = 40.00 m = 0.040 km
0 --------------------X---------------------------- 100.00 m
Tempo = 30.0 s  Velocita' = 4.000 m/s = 14.40 km/h  Distanza = 75.00 m = 0.075 km
0 -----------------------------------X------------ 100.00 m
Tempo = 40.0 s  Velocita' = 5.000 m/s = 18.00 km/h  Distanza = 120.00 m = 0.120 km
0 ---------------------------------------------- 100.00 m
FINE SIMULAZIONE
```