LABORATORIO FONDAMENTI DI INFORMATICA
29 OTTOBRE 2019 – Incontro 5 di 8 – Vettori e Struct

Esercizio 1
Data una matrice di `DIM` x `DIM` numeri interi riempita con il generatore di numeri pseudo-casuali `rand() % 10`, trovare e poi stampare a video la riga la cui somma degli elementi è minima (in caso vi siano più righe con la stessa somma minima, stamparne una a scelta).

```
 7  9  3 => somma = 19
 8  0  2 => somma = 10
 4  8  3 => somma = 15
La riga con la somma minima (10) e' quella con indice 1 [ 8  0  2]
```

Esercizio 2
Definire con `struct` un tipo di dato atto a rappresentare un bollettino meteo, con una data (`anno, mese, giorno`) e una stringa `meteo` (massimo `#define TXTL 20` caratteri).
Chiedere all'utente di inserire i dati necessari alla creazione di un bollettino (ricordandosi di utilizzare `fflush(stdin);` prima di chiedere all'utente la stringa) e poi, sapendo che un anno è bisestile (cioè è un anno in cui il mese di febbraio ha 29 giorni) quando
`(anno % 4 == 0 && (anno % 100 != 0 || anno % 400 == 0))`
« Un anno è bisestile se il suo numero è divisibile per 4, con l'eccezione degli anni secolari (quelli divisibili per 100) che non sono divisibili per 400. »
e utilizzando il seguente array che contiene il numero di giorni per ciascun mese
`int ggmese[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};`
verificare se la data immessa dall'utente è corretta (ovvero se 1 <= `mese` <= 12 e se 1 <= `giorno` <= `ggmese[mese - 1]`) e in caso affermativo stampare il numero di giorni trascorsi a partire dall'inizio dell'anno (ricordandosi di impostare se necessario a 29 i giorni di febbraio nell'array `ggmese`) e i dati del bollettino.

```
Anno? 2000
L'anno 2000 e' bisestile e quindi febbraio ha 29 giorni.
Mese? 12
Mese 12 corretto.
Giorno? 31
Giorno 31 corretto.
Meteo? Neve
Sono trascorsi 366 giorni dall'inizio dell'anno.
Bollettino Meteo 31/12/2000: Neve.
```

Esercizio 3
Dichiarare un array di interi denominato `coda` di lunghezza `C_LEN 10` per rappresentare una coda FIFO (First In First Out - il primo elemento che entra è il primo ad uscire), utilizzando una variabile intera `n` per indicare in ogni istante quanti elementi sono presenti nella coda. Ogni volta che un elemento entra nella coda viene inserito nella prima posizione disponibile `n` a partire dalla cella 0 dell'array. Quando un elemento esce dalla coda, tutti gli elementi che stanno nelle celle successive alla cella 0 vengono spostati nella cella alla loro sinistra. Simulare il comportamento di una coda per `P_MAX 15` passi: ad ogni passo l'azione di ingresso o uscita viene stabilita in base a un numero fornito dal generatore di numeri pseudo-casuali `rand() % 2;` con

- 1 = "ingresso" di una nuovo elemento, cioè un numero intero pseudo-casuale ottenuto con `rand() % 10;` che viene inserito nella prima cella disponibile. Se la coda è piena, l'ingresso non avviene (stampare un messaggio di errore).
- 0 = l'elemento nella cella 0 "esce", cioè tutti quelli che lo seguono vengono spostati "indietro" di una posizione. Se la coda è vuota, l'uscita non avviene (stampare un messaggio di errore).

```
Passo  1 di 15. Ingresso di 3. Stato della coda: 3
Passo  2 di 15. Ingresso di 9. Stato della coda: 3 9
Passo  3 di 15.   Uscita di 3. Stato della coda: 9
Passo  4 di 15.   Uscita di 9. Coda vuota.
Passo  5 di 15. Uscita non possibile. Coda vuota.
...
Passo 15 di 15. Ingresso di 6. Stato della coda: 8 8 6 6
```

Esercizio 4

Scrivere un programma che simuli il gioco degli anagrammi.

Definire per prima cosa un vettore di stringhe contenente le parole da indovinare (`NP 5` parole di lunghezza massima `LP 10`) - ad esempio:

```
char parole[NP][LP] = {"cane", "gatto", "gallo", "rana", "ibis"};
```

Utilizzando `struct` definire poi una variabile `an` nel modo seguente:

```
struct {
     char parola[LP]; //parola "originale"
     int usati[LP]; //0 se l'i-esimo carattere non e' ancora stato
                    //usato, 1 se e' gia stato usato per
                    //costruire l'anagramma
     char anagramma[LP]; //parola "rimescolata"
} an;
```

Il programma stabilisce quale parola del vettore `parole` da indovinare proporre all'utente tramite `rand() % NP;` e la copia in `an.parola`.

Ciascun carattere della stringa `an.parola` va copiato in ordine casuale nella stringa `an.anagramma`. Ogni volta che un i-esimo carattere di `parola` viene scelto, viene inserito il numero 1 nell'i-esima posizione del vettore `usati`, per indicare che quel carattere è già stato usato. La scelta del carattere da usare viene fatta in maniera pseudo-casuale tramite `rand() % strlen(an.parola);` continuando a ripetere la scelta casuale nel caso in cui il carattere sia già stato usato (il che è indicato dalla presenza di un "1" nel vettore `usati`). Una volta copiati tutti i caratteri occorre aggiungere il carattere terminatore `'\0'` per terminare la stringa `anagramma`.

Il gioco consiste nello stampare a video l'anagramma, chiedere all'utente di indovinare la parola originale e infine mostrare un messaggio con l'esito della partita.

NB:
```
strlen(s); //ritorna la lunghezza in caratteri della stringa s
strcpy(dest, sorg); //copia stringa sorg in stringa dest
strcmp(s1, s2); //ritorna 0 se e solo se s1 e s2 sono uguali
```

```
Anagramma di olgla? gallo
Hai indovinato!
```

Esercizio 5

Scrivere un programma che simuli il gioco della battaglia navale "semplificato" (una nave occupa una sola casella e ha un'energia massima pari a `EN_MAX = 5`). Il mare dove andranno posizionate le navi va rappresentato tramite una matrice di `DIM_MARE x DIM_MARE` interi denominata `mare (DIM_MARE = 5)`. Una nave è definita dal seguente tipo `s_nave`:

```
typedef struct {
     int x; //indice di riga da 0 a DIM_MARE - 1
     int y; //indice di colonna da 0 a DIM_MARE - 1
     int energia; //da 0 a EN_MAX
} s_nave;
```

Le navi saranno contenute in un vettore denominato `flotta` di `N_NAVI = 5` navi.

Il "`mare`" contiene nella posizione [x][y] il numero `-1` se nessuna nave è presente a quelle coordinate, o l'indice di una nave presente. Tale indice coincide con l'indice di posizione della nave

nell'array `flotta`.
I dati riguardanti le navi con la loro posizione nella matrice `mare` vanno scelti in maniera casuale tramite il generatore di numeri pseudo-casuali `rand() % ...;` (Controllare che una nave non venga collocata dove è già presente un'altra nave e che ciascuna nave abbia inizialmente 1 <= `energia` <= `EN_MAX`).
La partita si svolge facendo scegliere al generatore di numeri delle coordinate (x, y) e se una nave è presente a quelle coordinate la sua `energia` va decrementata di 1. Una nave è affondata quando la sua energia diviene 0. Il gioco termina quando tutte le navi sono affondate.

```
Mare:
       0  1  2  3  4
--------------------
  0 | -1 -1 -1 -1 -1
  1 | -1 -1 -1 -1 -1
  2 | -1 -1  4 -1  0
  3 |  1 -1 -1 -1 -1
  4 |  3 -1 -1  2 -1

Flotta:
Nave 0 in (2, 4). Energia = 5.
Nave 1 in (3, 0). Energia = 2.
Nave 2 in (4, 3). Energia = 4.
Nave 3 in (4, 0). Energia = 1.
Nave 4 in (2, 2). Energia = 3.

Partita:
Fuoco coordinate (3, 2): Acqua!
Fuoco coordinate (4, 0): Nave 3 colpita e affondata!
Fuoco coordinate (4, 3): Nave 2 colpita! Energia rimanente = 3.
Fuoco coordinate (4, 0): Nave 3 gia' affondata...
...
Tutte le navi sono state affondate dopo 140 colpi!
```