LABORATORIO FONDAMENTI DI INFORMATICA 18-19 NOVEMBRE 2013

Esercizio 1

Definire con `struct` un tipo di dato atto a rappresentare una pagina di diario, con una data (`anno, mese, giorno`) e un `testo` (massimo `#define DIM 100` caratteri).

Chiedere all'utente di inserire i dati necessari alla creazione di una pagina di diario 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 <= num_giorni_mese) e stampare il numero di giorni trascorsi a partire dall'inizio dell'anno.

NB: `scanf("%s", stringa);` si ferma al primo spazio, usare `gets(stringa);`

```
Anno? 2000     Mese? 1   Giorno? 23     Testo? Oggi c'era il sole
L'anno 2000 e' bisestile. Mese corretto. Giorno corretto.
Sono trascorsi 23 giorni dall'inizio dell'anno. Oggi c'era il sole
```

Esercizio 2

Data una matrice `DIM` x `DIM` 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.

```
 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 3

Dichiarare un array di interi denominato `coda` di lunghezza `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 `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  0. Ingresso di 3 => Stato della coda: 3
Passo  1. Ingresso di 9 => Stato della coda: 3 9
Passo  2.   Uscita di 3 => Stato della coda: 9
Passo  3.   Uscita di 9 => Coda vuota
Passo  4. Coda vuota, uscita non possibile
...
```

Esercizio 4

Scrivere un programma che simuli il gioco degli anagrammi.

Definire per prima cosa un array di stringhe contenente le parole da indovinare (`NUM 5` parole di lunghezza massima `LEN 10`)

```
char words[NUM][LEN] = {"cane", "gatto", "gallo", "rana", "ibis"};
```

Definire poi un tipo di dato nel modo seguente:

```
struct {
     char parola[LEN]; //parola "originale"
     int usati[LEN]; //0 se i-esimo carattere non e' stato usato,
                     //1 se usato
     char anagramma[LEN]; //parola "mescolata"
} an
```

Il programma stabilisce quale parola dell'array `words` proporre tramite `rand() % NUM;` 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 dell'array `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" nell'array `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); //lunghezza in caratteri della stringa s
strcpy(dest, sorg); //copia stringa sorg in stringa dest
strcmp(s1, s2); //vale 0 se e solo se s1 e s2 sono uguali
```

```
Anagramma = olgla
Parola? gallo
Hai indovinato!
```

Esercizio 5

Scrivere un programma che simuli il gioco della battaglia navale "semplificato" (una nave occupa una sola casella).

Definire il tipo `nave` nella maniera seguente:

```
typedef struct {
     int latitudine; //riga da 0 a DIM - 1
     int longitudine; //colonna da 0 a DIM - 1
     int energia; //da 0 a 5
} nave;
```

Definire poi un array denominato `flotta` di `NUM 5` navi e infine una matrice di `DIM x DIM` interi denominato `mare`.

Il "`mare`" contiene nella posizione [i][j] 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).

La partita si svolge facendo scegliere al generatore di numeri delle coordinate (i, j) 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 gia' affondata...
...
Tutte le navi sono state affondate!