LABORATORIO DI FONDAMENTI DI INFORMATICA
Incontro 4 di 5 – Funzioni

Esercizio 1
Realizzare un programma che chieda una temperatura in gradi Fahrenheit all'utente e la converta in gradi Celsius tramite la chiamata a una funzione denominata "`conversione`". La funzione riceve come unico parametro un numero che rappresenta la temperatura in gradi Fahrenheit e restituisce l'equivalente in gradi Celsius.

```
Temperatura in gradi Fahrenheit? 100
Conversione: 100.000 Fahrenheit = 37.778 Celsius
```

Suggerimenti:
- formula di conversione: `tC = (tF - 32) * 5.0 / 9.0`
- è pratica comune quella di specificare nel codice prima tutte le dichiarazioni (cioè i prototipi) delle funzioni (tranne `main`), poi la definizione della funzione `main` e infine le definizioni (cioè l'implementazione) di tutte le funzioni.
- nei prototipi delle funzioni il nome dei parametri non è obbligatorio, solo il tipo.

Esercizio 2
Definire una funzione denominata "`genera_matrice`" che riceve come parametro una matrice quadrata 2 x 2 di numeri interi. La funzione deve riempire in modo casuale la matrice con numeri positivi < 10 e "ritornare" il determinante della matrice e la somma degli elementi contenuti.
Scrivere un programma che chiami questa funzione e poi stampi a video la matrice, il suo determinante e la somma dei suoi elementi.

```
Matrice 2 x 2:
 4 8
 1 6
Determinante = 16
Somma elementi = 19
```

Suggerimenti:
- il determinante di una matrice 2 x 2 si calcola con la formula
  `determinante = mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]`
- quando una funzione ha come parametro una matrice bidimensionale non è obbligatorio specificare la dimensione sia delle righe che delle colonne ma il numero di colonne è obbligatorio
  - ok: `int funzione(int [2][2]);` `int funzione(int [][2]);`
  - no: `int funzione(int [][]);` `int funzione(int [2][]);`
- dato che la funzione deve "ritornare" due valori, occorre che almeno uno dei due venga inserito in una variabile passata per indirizzo
- osservare che la matrice viene per sua natura passata per indirizzo (la variabile che rappresenta un vettore è anche il puntatore al suo indirizzo in memoria) e quindi le modifiche fatte alla matrice all'interno della funzione permangono anche dopo il `return`

Esercizio 3
Scrivere un programma per la gestione di un bollettino meteorologico.
Definire con `typedef` un tipo di dato `struct` denominato `s_data` con i campi `anno, mese, giorno.`
Definire con `typedef` un tipo di dato `struct` denominato `s_bollettino` che abbia come campi una `data` di tipo `s_data` e una stringa `meteo` (massimo `#define TXTL 20` caratteri).
Definire come variabile globale il seguente vettore che indica per ciascun mese quanti siano i giorni in un anno non bisestile *(31 per gennaio, 28 per febbraio, ..., 30 per novembre, ...)*

```
int gg_mese[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Dichiarare e definire le seguenti funzioni:

- `int anno_bisestile(int anno);`
  riceve come parametro un anno e ritorna 1 se l'anno è bisestile, 0 altrimenti.
  « 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. »
  `(anno % 4 == 0 && (anno % 100 != 0 || anno % 400 == 0))`
- `int verifica_data(s_data data);`
  riceve come parametro una `data` e ritorna 1 se la `data` è corretta, 0 altrimenti.
  Un `anno` è corretto se è >= 2000. Un `mese` è corretto se è compreso tra 1 e 12.
  Un `giorno` è corretto se è compreso tra 1 e il numero che si trova per il `mese` indicato nel vettore globale `gg_mese`, tenuto conto che febbraio potrebbe anche avere 29 giorni invece che 28 (richiamare la funzione `anno_bisestile` per verificarlo)
- `int giorni_da_capodanno(s_data data);`
  riceve come parametri una `data` (che si suppone essere valida) e ritorna il numero di giorni trascorsi dal 1 gennaio dell'anno indicato.
  Questa funzione utilizza il vettore globale `gg_mese` e richiama la funzione `anno_bisestile` per sapere se per l'`anno` specificato febbraio abbia 29 giorni invece che 28.
- `void ins_bollettino(s_bollettino *bollettino);`
  riceve per indirizzo un `bollettino` e chiede all'utente di inserirvi una `data` (verificando che sia corretta tramite la funzione `verifica_data`) e un testo che descriva la situazione meteorologica
  Suggerimenti:
  - ricordare di inserire `fflush(stdin);` per svuotare il buffer prima di chiedere la stringa del meteo
  - la variabile `struct bollettino` è passata per indirizzo quindi si può usare l'operatore `->` per accedere ai suoi campi
- `void stampa_bollettino(s_bollettino bollettino);`
  stampa a video il `bollettino` ricevuto come parametro indicando anche quanti giorni siano trascorsi dall'inizio dell'anno (richiamando la funzione `giorni_da_capodanno`) e se l'anno è bisestile (richiamando la funzione `anno_bisestile`)

La funzione `main` dichiara una variabile `bollettino`, chiede all'utente di inserire i dati chiamando la funzione `ins_bollettino` e poi chiama la funzione `stampa_bollettino`.

```
Inserimento dei dati di un bollettino Meteo
Meteo? neve
Anno? 2020
Mese? 12
Giorno? 31
Anno 2020 corretto.
Mese 12 corretto.
Giorno 31 corretto.
Bollettino Meteo 31/12/2020: neve.
L'anno 2020 e' bisestile.
Sono trascorsi 366 giorni dall'inizio dell'anno.
```

Esercizio 4

Realizzare un programma che chieda all'utente una stringa e un carattere delimitatore e stampi a schermo una sua sottostringa (se esiste) che inizi e termini con il carattere delimitatore.

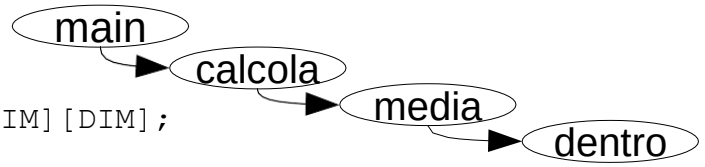
A tal fine realizzare la funzione "`estrazione`" che presi come parametri una stringa "`orig`" (array di char – massimo 50) e un carattere "`carat`" estragga, se esiste, una sottostringa di "`orig`"che parta dal carattere "`carat`" e termini con lo stesso carattere "`carat`" oppure arrivi in fondo alla stringa. Questa sottostringa (eventualmente vuota) deve essere copiata in un altro array "`risult`" che viene anch'esso passato come parametro alla funzione "`estrazione`".

```
stringa in ingresso? stampa pubblica
carattere? p
sottostringa: pa p
carattere? l
sottostringa: lica
```

Suggerimenti:

- per chiedere la stringa all'utente utilizzare la funzione `gets(stringa)` poiché `scanf("%s", stringa)` si fermerebbe al primo spazio.
- ricordare di inserire come ultimo carattere della stringa "`risult`" il carattere terminatore `'\0'`
- utilizzare la funzione `strlen(stringa)` per calcolare la lunghezza in caratteri di una stringa.

Esercizio 5



Definire il tipo di dato

```
typedef float t_matrice[DIM][DIM];
```

e implementare la funzione

```
void calcola(t_matrice matr_media, int matr_orig[DIM][DIM])
```

Tale funzione analizza la matrice di numeri interi "`matr_orig`" passata come parametro e calcola e inserisce dei valori reali nella matrice "`matr_media`" anch'essa passata come parametro. In tale matrice ciascun elemento `[i][j]` dovrà essere un numero reale uguale alla media aritmetica (somma degli elementi diviso numero degli elementi) dei valori presenti nella matrice "`matr_orig`" nelle caselle adiacenti all'elemento `[i][j]` (senza coinvolgere l'elemento `[i] [j]`). Le caselle "adiacenti" a una determinata casella `[i][j]` sono al massimo 8 ma sono di meno quando `[i][j]` è su uno dei bordi della matrice.

Per verificare se una casella adiacente esiste, ovvero se le sue coordinate `[i+h][j+k]` con -1<=`k`<=1 e -1<=`h`<=1 sono interne alla matrice, realizzare la seguente funzione:

```
int dentro(int r, int c)
```

Tale funzione ritorna `1` se le coordinate `r` e `c` sono interne alla matrice (cioè se `r` e `c` sono indici validi >= 0 e < `DIM`), altrimenti ritorna `0`.

Per calcolare la media relativa a ciascun elemento `[i][j]`, occorre richiamare l'ulteriore funzione da implementare

```
float media(int matr_orig[DIM][DIM], int i, int j)
```

Per gli elementi sui bordi, la media va fatta utilizzando solo gli elementi adiacenti che appartengono alla matrice (interni alla matrice, le cui coordinate fanno ritornare 1 alla funzione "`dentro`").

Nella funzione `main` del programma occorre dichiarare le due matrici "`matr_media`" e "`matr_orig`", riempire a piacere "`matr_orig`" e richiamare la funzione "`calcola`" per poi stampare a video le due matrici

```
Matrice:                    Matrice delle medie:
 1  2  3                     3.67  3.80  4.33
 4  5  6                     4.60  5.00  5.40
 7  8  9                     5.67  6.20  6.33
```

Esercizio 6

Scrivere un programma che simuli il comportamento di una coda FIFO (First In First Out - il primo elemento che entra è il primo ad uscire).

La coda sarà rappresentata da un array di numeri interi >= 0 denominato `coda` di lunghezza massima `C_LEN 10` accompagnato da una variabile intera `n` per indicare in ogni istante quanti elementi siano effettivamente presenti (a partire dalla cella 0 in poi).

Definire le seguenti funzioni:

- `enqueue`
  riceve come parametri la `coda`, il numero `n` di elementi presenti e un nuovo numero intero >= 0 `elem` da inserire. Se c'è ancora spazio il nuovo elemento viene inserito nella prima cella disponibile da sinistra, il numero `n` di elementi viene incrementato di 1 e la funzione ritorna 1. Se la coda è piena l'inserimento fallisce e viene ritornato -1.
- `dequeue`
  riceve come parametri la `coda` e il numero `n` di elementi presenti. Se la coda contiene almeno un elemento, la funzione ritorna il primo elemento del vettore dopo aver "spostato a sinistra" di una posizione tutti gli elementi e decrementato di 1 il numero `n` di elementi. Se la `coda` è vuota viene ritornato -1.
- `stampa_coda`
  stampa tutti gli elementi presenti nella coda indicando se è vuota o se è piena

Nella funzione `main` 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 = tentativo di “ingresso” di una nuovo elemento, cioè un numero intero pseudo-casuale ottenuto con `rand() % 10;` che viene passato alla funzione `enqueue`
- 0 = tentativo di "uscita" di un elemento chiamando la funzione `dequeue`

```
Passo  1 di 15. USCITA -> Coda vuota! Nessuna uscita. Coda: <vuota>
Passo  2 di 15. INGRESSO -> Ingresso di 8 riuscito. Coda: 8
Passo  3 di 15. USCITA -> Uscita di 8 avvenuta. Coda: <vuota>
...
Passo 13 di 15. INGRESSO -> Ingresso di 9 riuscito. Coda: 9
Passo 14 di 15. INGRESSO -> Ingresso di 3 riuscito. Coda: 9 3
Passo 15 di 15. INGRESSO -> Ingresso di 4 riuscito. Coda: 9 3 4
```

Suggerimenti:

- le funzioni `enqueue` e `dequeue` devono poter modificare il numero `n` di elementi che pertanto va passato per indirizzo.
  Attenzione perché gli operatori `++` e `--` hanno precedenza maggiore dell'operatore di dereferenziazione `*` cioè per incrementare / decrementare `*n` con `++` e `--` occorre utilizzare le parentesi tonde per dare precedenza all'operatore `*`
  `(*n)++; (*n)--;`

Esercizio 7

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 r; //indice di riga da 0 a DIM_MARE - 1
     int c; //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 [r][c] il numero `-1` se nessuna nave è presente a quellecoordinate, 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 (r, c) 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.
Dividere liberamente il programma in funzioni; un esempio di suddivisione potrebbe essere:

- `setup` - riceve come parametri il mare e la flotta e li "popola" con valori casuali
- `stampa_mare` - visualizza dove si trovino nel mare le navi della flotta
- `stampa_flotta` - visualizza l'energia e le coordinate di ciascuna nave della flotta
- `fuoco` - sceglie due coordinate casuali su cui fare fuoco ed eventualmente aggiorna l'energia di una nave della flotta se colpita
- `gameover` - verifica se 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!
```