Esercizio 1 (Luglio 2021)

(a) Scrivere una funzione di prototipo:

```
int controlla_permutazione(int p[], int lung)
```

che abbia come parametri un *array* di interi e la sua lunghezza e che restituisca 1 se l'*array* contiene i valori di una permutazione valida, 0 altrimenti.
Un *array* di interi contiene una permutazione valida se i valori in ogni cella sono tutti e soli i numeri che possono rappresentare la posizione di una cella dell'*array* stesso.

Esempio:

indice: `0 1 2 3 4 5`
array p: `[ 1, 3, 2, 4, 5, 0 ]`

(b) Scrivere una funzione

```
int applica_permutazione(char str[], int p[], int lung)
```

che abbia come parametri una stringa, un array di interi e un intero indicante la lunghezza dell'array di interi. Se *l'array* di interi rappresenta una permutazione valida e ha la stessa lunghezza della stringa, la funzione applica la permutazione ai caratteri della stringa indicata come parametro e restituisce 1, altrimenti 0.

Esempio:

`str: "AIRONE"`
array p: `[ 1, 3, 2, 4, 5, 0 ]`
`lung: 6`

dopo l'esecuzione della funzione `str: "AIRONE"` diventa
`str: "EARION"`

N.B: è consentito/raccomandato far uso delle funzioni di libreria in `<string.h>` e `...malloc(...)`

Se si continua sul retro di qualche foglio, indicare quale

Esercizio 2 (Luglio 2021)

Si considerino le seguenti definizioni di costanti e tipi di dato:

```
#define MAX_RIGHE    30
#define MAX_COLONNE  50

typedef struct {
                  int r;
                  int c;
        } cella_t;
```

Esibire il prototipo e il codice di un sottoprogramma

```
...trova_ciclo(...)
```

che prenda come primo parametro un *array* bidimensionale in cui ogni cella sia di tipo `cella_t`, come secondo parametro il numero di righe effettivo della matrice indicata dal primo argomento (dovrà essere un valore <= `MAX_RIGHE`) e come terzo parametro il numero di colonne effettivo della matrice (dovrà essere un valore <= `MAX_COLONNE`).

Il sottoprogramma deve acquisire da tastiera una coppia di coordinate della matrice (controllando che siano valide) e restituire 1 se a partire dalla cella con le coordinate indicate è possibile seguire sulla matrice un percorso ciclico, 0 altrimenti.

Nota: le coordinate nella prima cella individueranno una seconda cella, che a sua volta conterrà delle coordinate. Le coordinate in quest'ultima individueranno una terza cella e così via.
Il sottoprogramma deve stabilire se procedendo in tal modo è possibile ritornare sulla cella di partenza (restituendo 1) oppure no (restituendo 0).

Esercizio 3 (Luglio 2021)

Si desidera analizzare la statistica dei consumi di *toner* di un'azienda per ottimizzare gli acquisti futuri, facendo uso delle costanti e del tipo di dato di seguito specificati.

```
#define MAX_STR 30
#define MAX_NUMERO 6
typedef struct {
            char nomeDip[MAX_STR+1];
            int  occorrenze[MAX_NUMERO];
        } stat;
```

La quantità di cartucce di *toner* prelevate dal magazzino è riportata all'interno di un *file* di testo.
Il *file* contiene una riga per ogni ordine fatto da ciascun dipartimento. Ogni riga contiene in sequenza:

- il nome del dipartimento che ha prelevato il/i *toner* (una stringa lunga `MAX_STR` caratteri);
- un numero intero (valore minimo 1 e massimo `MAX_NUMERO`) che indica la quantità di cartucce di *toner* prelevate da quel dipartimento.

Non è noto il numero di righe presenti nel *file*.

Esempio di formato del *file*:

```
ACQUISTI 1
VENDITE 2
ACQUISTI 1
ACQUISTI 5
```

(a) Si esibisca il codice del sottoprogramma

```
void riempi_vett(char nomefile[], stat vett[], int lungVett)
```

che richiede come primo parametro il nome del *file* di testo contenete le informazioni sugli ordini delle cartucce di *toner*, come secondo parametro un *array* le cui celle sono di tipo `stat` e come terzo parametro la lunghezza dell'*array* stesso.
Il sottoprogramma assume che una stringa nell'attributo `nomeDip` in ciascuna cella dell'*array* `vett[]` sia già presente e che sia diversa da quelle nelle altre celle, inoltre il valore di tutti gli interi nell'attributo `occorrenze` di ogni cella è assunto essere inizializzato a 0.
Il sottoprogramma modifica l'attributo `occorrenze` di ciascuna cella, memorizzando in `occorrenze[0]` quante righe del *file* contengono 1 come quantità di *toner* richiesta dal dipartimento indicato nell'attributo `nomeDip`, `in occorrenze[1]` quante righe del *file* contengono 2 come quantità di *toner* richiesta dal dipartimento indicato nell'attributo `nomeDip` e così via.

(b) Si esibisca il codice del sottoprogramma

```
void toner_statistic(stat vett[], int lungVett)
```

Il sottoprogramma assume come primo parametro un vettore di statistiche creato dal sottoprogramma al punto (a) . Esso crea un nuovo *file* chiamato `"statistic.txt"` con tante righe quante sono le celle dell'*array* `vett` e, su ogni rigo, riporta: il nome del dipartimento, la minima, la massima e la media quantità di *toner* che è stata richiesta negli ordini emessi dal dipartimento, il valore della quantità di *toner* richiesta più spesso (la moda) dal dipartimento e il numero di volte che tale valore si è ripetuto - in caso ci sia più di un valore moda, è sufficiente riportarne solo uno.
Se un dipartimento non ha effettuato ordini di toner, il sottoprogramma riporterà sul rigo del *file*: `"Il Dip. non ha ordinato toner!"`

Esempio: se l'*array* `vett` avesse lunghezza 2 e contenesse i seguenti dati
nella cella[0]: {`"ACQUISTI"`, { 2, 1, 0, 0, 1, 0 } }
nella cella[1]: {`"VENDITE"`, { 0, 1, 0, 0, 0, 0 } }
Il *file* `"statistic.txt"` creato dal sottoprogramma sarebbe:

```
ACQUISTI 1, 5, 2.25, (1, 2)
VENDITE 2, 2, 2.00, (2, 1)
```

Se si continua sul retro di qualche foglio, indicare quale

Esercizio 4 (Giugno 2021)

Si consideri un numero intero positivo `N` e il valore della somma dei suoi divisori `S`, escludendo da tale somma il numero stesso (perché un qualunque numero è sì un divisore di se stesso, ma improprio).
Chiamiamo ***difettivo*** un numero intero tale che la somma dei suoi divisori è minore del numero stesso (per esempio: `N=10` è difettivo perché `S=1+2+5=8`).
Chiamiamo ***perfetto*** un numero intero uguale alla somma dei suoi divisori (per esempio: `N=6` è perfetto perché `S=1+2+3=6`).
Chiamiamo ***abbondante*** un numero intero tale che la somma dei suoi divisori è maggiore del numero stesso (per esempio: `N=42` è abbondante perché `S=1+2+3+6+7+14+21=54`).

(a) Si codifichino in `C` le funzioni `...somma_divisori(...)` e `...classifica_numero(...)`, ciascuna delle quali riceve come parametro un numero intero positivo.
La prima funzione restituisce la somma dei divisori del valore ricevuto come argomento (non considerando il numero stesso come divisore).
La seconda funzione restituisce `-1` se il valore ricevuto come argomento è difettivo, restituisce `0` se è perfetto, oppure restituisce `+1` se è abbondante. **[6 punti]**

Nota: si assuma che sia responsabilità del programma chiamante garantire che il valore passato come parametro debba essere strettamente positivo (quindi né negativo, né nullo).

(b) Si codifichi in `C` un programma principale, **int** `main(){...}`, che acquisisca dallo *standard input* una sequenza di numeri interi (di lunghezza arbitraria) terminata dal valore `0` e che riporti sullo *standard output* il conteggio delle coppie di numeri consecutivi ed entrambi difettivi, oppure consecutivi ed entrambi perfetti oppure consecutivi ed entrambi abbondandi. **[4 punti]**

Nota: quando appropriato, è ammesso il riutilizzo dei sottoprogrammi definiti al punto (a).

Se si continua sul retro di qualche foglio, indicare quale

Esercizio 5 (Giugno 2021)

(a) Scrivere in C un sottoprogramma con il prototipo seguente

```
int equiv(char prima[], char seconda[])
```

che presi come argomenti due stringhe, restituisca +1 se la seconda stringa risulta essere uguale alla prima dopo aver effettuato uno scambio della prima lettera della seconda stringa con un'altra lettera della stessa.

Esempio:

```
prima:     "ANNA", seconda: "NANA"      → restituisce +1
prima:     "MARE", seconda: "RAME"      → restituisce +1
prima:   "ERCOLE", seconda: "CREOLE"    → restituisce +1
prima:     "ANNA", seconda: "ANNA"      → restituisce +1
prima:"BALLETTO", seconda: "BOLLETTA"   → restituisce 0
prima:     "ARCO", seconda: "ARCA"      → restituisce 0
```

(b) Scrivere in C un sottoprogramma con il prototipo seguente

```
int equivalenti(char prima[], char seconda[])
```

che presi come argomenti due stringhe, restituisca +1 se risultano essere uguali effettuando al più uno scambio di due caratteri (in posizioni qualsiasi), 0 altrimenti.

Esempio:

```
prima: "BALLETTO", seconda: "BOLLETTA"  → restituisce +1
prima: "CARIATO",  seconda: "CARTAIO"   → restituisce +1
prima: "SARTO",    seconda: "TARSO"     → restituisce +1
prima: "ANNO",     seconda: "ANNO"      → restituisce +1
prima: "ARCO",     seconda: "ARCA"      → restituisce 0
```

Nota: è ammeso far uso delle funzioni della libreria `<string.h>`

Se si continua sul retro di qualche foglio, indicare quale

Esercizio 6 (Giugno 2021)

Nello sport dei tuffi individuali, ogni tuffo viene valutato da **cinque giudici** diversi.
Ogni giudice esprime un **punteggio individuale** da 0 a 10, con la possibilità di esprimere anche mezzi punti.
Il punteggio 0 corrisponde a un tuffo completamente sbagliato; il punteggio 10 a un tuffo perfetto.
Per eliminare eventuali preferenze soggettive, il punteggio **più alto** e quello **più basso** vengono scartati in automatico.
Ogni punteggio è espresso indipendentemente dal **grado di difficoltà** del tuffo (un valore numerico che va da 1.3 a 3.6) e il punteggio finale attribuito all'atleta è calcolato come prodotto della somma dei tre punteggi rimanenti per il grado di difficoltà del tuffo.

1. Date le seguenti definizioni di costanti e di tipo di dato

```
#define NUM_GIUDICI 5
#define MAX_STRING 30

typedef struct {
                char  nome[MAX_STRING+1];
                char  cognome[MAX_STRING+1];
                char  sesso;
                char  nazione[MAX_STRING+1];
                float punteggi[NUM_GIUDICI];
                float gradoDifficolta;
        } atleta_t;

typedef struct {
                   atleta_t* array;
                   int numAtleti;
        } gara_t;
```

   scrivere in C la funzione

```
gara_t acquisisciDaFile(char nomefile[])
```

   che restituisce una variabile di tipo gara_t che memorizzi come valore del suo primo attributo l'indirizzo iniziale di un *array* dinamico con tante celle quante sono le righe del *file* di testo il cui nome è indicato come argomento del sottoprogramma, e come valore del suo secondo attributo la lunghezza dell'*array* dinamico.
   Il *file* di testo è assunto avere il seguente formato:

```
Tania Cagnotto F ITALIA  10 9.2 8.5 8.2 9.5 3.0
Kent Ferguson M USA 9.8 9.1 8.2 9.5 9.0 2.8
Dmitrij Sautin M RUSSIA 9.8 9.5 8.0 8.5 9.0 2.8
Patrick Hausding M GERMANIA 9.5 9.5 9.5 10 8.0 1.5
Shi Tingmao F CINA 8.5 8.2 8.0 9.0 9.5 2.5
…
```

   dove su ogni rigo sono riportati nome, cognome, sesso ('M' per uomini e 'F' per donne), nazione di un atleta, il punteggio a lui assegnato da ciascuno dei NUM_GIUDICI giudici e il grado di difficoltà del tuffo eseguito. Ogni cella dell'*array* dinamico deve contenere le informazioni prese da un rigo diverso del *file*.

   N.B. Si assuma che non ci siano nomi o cognomi composti da più di una parola. Le informazioni su ogni rigo sono separate da un carattere ' ' (spazio). Il numero di righi del *file* non è noto a priori.

2. Scrivere in C la funzione

```
void stampa_migliori(gara_t g)
```

   che riporti a schermo il nome di uno dei migliori atleti della competizione (cioè uno di quelli che hanno ottenuto il punteggio più alto).

Se si continua sul retro di qualche foglio, indicare quale