POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

CORSO DI LAUREA IN INGEGNERIA AEROSPAZIALE

Prof. G. Agosta
Prof. Gerardo Pelosi

Informatica – a.a. 2022/2023– 1° Appello – 19 Gennaio 2023

Cognome _____ **Matricola o Cod. Persona** _____

Nome _____ **Firma** _____

Istruzioni

Non separate questi fogli. Scrivete la soluzione **solo sui fogli distribuiti**, utilizzando il retro delle pagine se necessario. **Cancellate le parti di brutta** (o ripudiate) con un tratto di **penna**.

Ogni parte non cancellata a penna sarà considerata parte integrante della soluzione.

È possibile scrivere a matita e non occorre ricalcare al momento della consegna.

Qualsiasi **tentativo** di comunicare con altri studenti comporta **l'espulsione** dall'aula.

Non è possibile lasciare l'aula conservando il tema della prova in corso.

È possibile ritirarsi senza penalità.

Tempo a disposizione: **2h:30**

Valore indicativo degli esercizi, voti parziali e voto finale:

Esercizio 1 (4 punti) _____

Esercizio 2 (4 punti) _____

Esercizio 3 (10 punti) _____

Esercizio 4 (6 punti) _____

Esercizio 5 (10 punti) _____

Totale (34 punti) _____

Voto finale _____

Esercizio 1 Algebra di Boole [4 punti]

Scrivere un programma che legga da terminale triple di numeri interi A, B, C, e stampi a video una linea contenente il numero d'ordine della tripla letta (assumendo che la prima tripla sia numerata 1) se per tale tripla la formula logica

```
not (A or C) or not B and C
```

è vera.

Nel caso in cui invece la formula risulti falsa, il programma non stampa nulla.

Il programma termina se `A`, `B` e `C` sono tutti e tre negativi.

Soluzioni:

```c
#include <stdio.h>

int main(){
    int a, b, c, i = 1;

    do {
        scanf("%d%d%d", &a, &b, &c);

        if (!(a || c) || !b && c)
            printf("%d\n", i);

        i++;
    } while(!(a < 0 && b < 0 && c < 0));

    return 0;
}
```

Esercizio 2 Codifiche numeriche [4 punti]

Scrivere una funzione di prototipo

```
int sum(int a[N], int b[N], int c[N]);
```

che riceve in ingresso 2 array `a` e `b` contenenti numeri interi nell'intervallo `[0,1]`, da interpretare come bit, e memorizzi nell'array `c` di pari lunghezza i valori dei bit risultanti dalla somma in complemento a 2 dei numeri rappresentati dagli array `a` e `b`.

La funzione restituisce `1` se si è verificato un errore e `0` altrimenti.

Soluzioni:

```
int sum(int a[N], int b[N], int c[N]) {
  int carry = 0;
  int i;

  for(i = N - 1; i >= 0; i--) {
    int sum = a[i] + b[i] + carry;
    carry = sum / 2;
    c[i] = sum % 2;
  }

  if ((a[0] && b[0] && !c[0]) || (!a[0] && !b[0] && c[0]))
      return 1;

  return 0;
}
```

Esercizio 3 (10 punti)

Si consideri un array di interi positivi senza duplicati nell'intervallo `1 ... MAXNUM`.

Scrivere un sottoprogramma C

```
... replace(...);
```

che abbia come parametro un array di interi e la sua lunghezza che controlli che l'array usato come suo argomento contenga solo numeri positivi senza duplicati. Nel caso in cui il controllo non dia esito positivo, il sottoprogramma deve restituire zero, altrimenti prosegue sostituendo ogni elemento dell'array con l'elemento più piccolo tra quelli aventi valore maggiore di lui che sono posizionati alla sua destra oppure con `-1` se non ci sono elementi maggiori di lui alla sua destra; il sottoprogramma termina restituendo `1` al programma chiamante.

Soluzioni:

```c
int main(void) {
    int i, r, n = 0;
    scanf("%d", &n);
    int *nums = malloc(sizeof(int) * n);

    if (nums == NULL) {
        printf("errore in allocazione\n");
        return 0;
    }

    for (i = 0; i < n; i++)
        scanf("%d", &nums[i]);

    r = replace(nums, n);

    if (r == 0) {
        printf("Array non valido\n");
    } else if (r == 1) {
        // Stampa l'array risultante
        for (i = 0; i < n; i++)
            printf("%d ", nums[i]);
    }

    free(nums);
    return 0;
}


// Restituisce +1 se ci sono duplicati e/o numeri negativi, 0 altrimenti
int checkPositiveNoDuplicates(int nums[], int n){
    int i, j;

    for (i = 0; i < n; i++) {
        if (nums[i] < 0)
            return 0;

        for (j = i + 1; j < n; j++)
            if (nums[j] == nums[i])
                return 0;
    }

    return 1;
}
```

```c
// Sostituisci ogni elemento dell'array specificato con il
// elemento meno maggiore alla sua destra
int replace(int nums[], int n) {
    int i, j;

    if (checkPositiveNoDuplicates(nums, n) == 0)
        return 0;

    // Attraversa l'array dall'inizio
    for (i = 0; i < n; i++) {
        int successor = -1;
        int diff = MAXNUM;

        // Controlla ogni elemento a destra per un successore
        for (j = i + 1; j < n; j++) {
            if (nums[j] > nums[i] && (nums[j] - nums[i] < diff)) {
                successor = nums[j];
                diff = nums[j] - nums[i];
            }
        }

        nums[i] = successor;
    }

    return 1;
}
```

Esercizio 4 (6 punti)

Scrivere un programma C che:

- Legga dal terminale un numero intero dispari `n`.
- Legga dal terminale una sequenza `s` di numeri interi separati da spazi e terminata da un qualunque carattere non numerico.
- Stampi a terminale, per ogni sottosequenza di lunghezza `n` nella sequenza `s` (partendo da sinistra) la media dei suoi elementi pesata con un peso pari a `1` per il primo e l'ultimo elemento, `4` per il secondo e il penultimo, `9` per il terzo e il terzultimo, etc. (ovvero pesi pari al quadrato della distanza dall'estremo più vicino, con l'elemento centrale pesato con peso pari al quadrato di `n/2+1`). Ad esempio, per sottosequenze di lunghezza 5, i pesi saranno, nell'ordine, 1, 4, 9, 4, 1.
- Nel caso in cui non venga letto un numero intero `n` dispari, il programma stampa il messaggio di errore "`n pari o non letto correttamente`".
- In entrambi i casi, il programma deve restituire il valore `0`.

Esempio:

stdin
3 3 6 2 1 5 4 3 2 9 a

stdout
4.833333 2.500000 1.833333 4.166667 4.000000 3.000000 3.333333

Se si continua sul retro di qualche foglio, indicare quale

Soluzioni:

```c
#include <stdio.h>
#include <stdlib.h>
#define OK 0

char *errmsg = "n pari o non letto correttamente";

float filtro(int w[], int l);
void scorrimento(int w[], int l);

int main(){
    int n=0;
    scanf("%d", &n);

    if (!(n%2)) {
        printf("%s",errmsg);
        return OK;
    }

    int *f=malloc(sizeof(int)*n);
    int r=1, l=0;

    do {
        int x=0;
        r=scanf("%d", &x);
        if (!r) break;
        f[l]=x;
        l++;
        if (l<n) continue;
        printf("%f\n", filtro(f,n));
        scorrimento(f,n);
        l--;
    } while(1);

    free(f);
    return OK;
}

int abs(int x){
    return (x<0 ? -x : x);
}

float filtro(int w[], int l){
    int sum=0, hl=l/2, tw=0;
    for(int i=0; i<l; i++){
        int weight=hl+1-abs(i-hl);
        sum+=w[i]*weight*weight;
        tw+=weight*weight;
    }
    return sum/(float)tw;
}

void scorrimento(int w[], int l){
    for(int i=1; i<l; i++)
        w[i-1]=w[i];
}
```

Esercizio 5 (10 punti)

Si vuole progettare un *software* per la gestione degli esercizi svolti da una persona in palestra.
Si consideri il seguente frammento di codice, utilizzabile all'interno della soluzione:

```
typedef enum {
    NONE /*empty element*/,
    CALF,
    QUAD,
    FEM_BICEPS,
    HIP_FLEXORS,
    ABS,
    PECT,
    LAT,
    BACK,
    ERECTOR,
    TRAPS,
    BICEPS,
    TRICEPS,
    FOREARM,
    DELTS,
    STOP /* index of last element */
} muscle_groups;


typedef struct _exercise {
    char name[128];
    int exercised_muscles[STOP];
} exercise;

int count_exercised(exercise *e){
    int count = 0;
    int i;

    for(i = 1; i < STOP; i++)
        count += e->exercised_muscles[i];

    return count;
}
```

(a) Progettare e implementare una funzione di prototipo

```
exercise *read_exercises();
```

che legga da terminale una sequenza di linee di testo di lunghezza arbitraria, e interpreti ciascuna riga come la specifica di un esercizio, costituita da:

- nome: una stringa di testo non più lunga di 127 caratteri.
- lista di muscoli esercitati: una sequenza non vuota di numeri interi nell'intervallo da `1` a `14` (corrispondenti ai valori del tipo enumerativo `muscle_groups`), senza ripetizioni ma possibilmente non ordinati, separati da un singolo spazio o carattere di tabulazione.

La sequenza è terminata da una linea vuota.

La funzione deve restituire un array di elementi di tipo `exercise`.

Tale array sarà terminato da un elemento terminatore con campo `name` pari alla stringa vuota, e campo `exercised_muscles` interamente posto a `0`.

Tutti gli altri elementi conterranno gli esercizi letti da terminale, con campo `name` pari al nome dell'esercizio e campo `exercised_muscles[i]` pari a `1` (vero) se il gruppo muscolare `i` è compreso nella lista di muscoli esercitati, e `0` (falso) altrimenti.

Se si continua sul retro di qualche foglio, indicare quale

(b) Progettare e implementare una funzione di prototipo:

```
print_non_exercised_by_max(exercise *l);
```

che riceva la struttura dati prodotta dalla funzione `read_exercises` e stampi la sequenza di gruppi muscolari non esercitata dall'esercizio che esercita il maggior numero di gruppi muscolari (assumendo che sia unico), con gli elementi ordinati in senso crescente, separati da spazi. La sequenza deve essere terminata dal carattere di fine linea `'\n'`.

Soluzioni: (a)

```c
int read_exercise(exercise *e){
    char s[168]; /*va bene qualunque numero abbastanza grande*/
    char *r = NULL;
    char *p = NULL;
    int read = 0;

    r = fgets(s, 168, stdin);
    memset(e, 0, sizeof(exercise)); /* azzera e */
    if (!r || r[0] == '\n') return 0; /* nessun elemento letto */
    sscanf(s,"%s", e->name);
    p = s + strlen(e->name);

    do {
        int x = NONE;
        read = sscanf(p, "%d", &x);
        do { p++; } while(*p >= '0' && *p <= '9');
        e->exercised_muscles[x] = 1;
    } while(read && *p != '\0');

    return 1; /* un elemento letto */
}

exercise *extend(exercise *e, int count, int toadd) {
    int i;
    exercise *r = malloc(sizeof(exercise) * (count + toadd));
    memset(r, 0, sizeof(exercise) * (count + toadd)); /* inizializza a 0 */

    for(i=0; i < count; i++)
        r[i] = e[i];

    free(e);
    return r;
}

exercise *read_exercises() {
    exercise *r = malloc(sizeof(exercise) * 8);
    memset(r, 0, sizeof(exercise) * 8); /* inizializza tutto a 0*/
    exercise e;
    int count = 0;
    int read = 0;

    do {
        read = read_exercise(&e);

        if (read) {
            r[count] = e;
            count++;
        }

        if (count != 0 && count % 8 == 0)
            r = extend(r, count, 8);
    } while(read);

    return r;
}
```

Soluzioni: 4.(b)

```
exercise *max_exercised(exercise *l) {
    exercise *max = NULL;
    int nmax = -1;
    int i;

    for(i = 0; l[i].name[0] != '\0'; i++) {
        int x = count_exercised(l + i);

        if (x > nmax) {
            max = l + i;
            nmax = x;
        }
    }

    return max;
}

void print_non_exercised_by_max(exercise *l) {
    int i;
    exercise *max = max_exercised(l);

    for(i = 1; i < STOP; i++)
        if (!max->exercised_muscles[i])
            printf("%d ", i);

    printf("\n");
}
```