

**Corso di Programmazione e strutture dati**

**Docente di Laboratorio:**

**Email:**

---

## **ESERCITAZIONE 2: TESTING**

# ESERCIZI

Implementare e testare l'aggiunta alla libreria vettore delle seguenti funzioni:

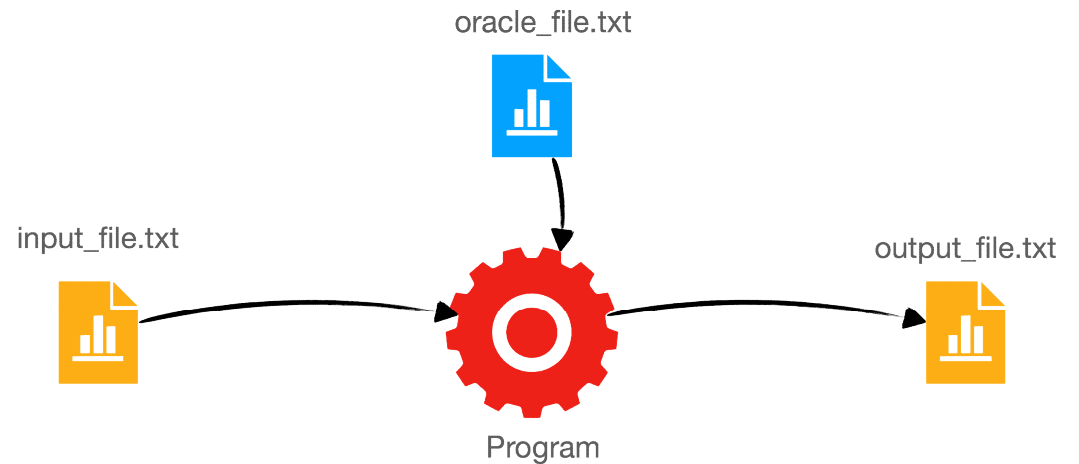
1. Funzione che prende in input un array di interi e restituisce la **somma** degli elementi dell'array
2. Funzione che prende in ingresso due array di interi e restituisce in uscita l'array che contiene come elemento di posizione  $i$  la somma degli elementi di posizione  $i$  degli array di input
3. Funzione che prende in ingresso due array di interi e restituisce il **prodotto scalare** dei due array. Il prodotto scalare di due array  $a$  e  $b$  è definito come:  $\sum_i a[i]*b[i]$

➤ Scrivere 3 file «**driver**» per testare le tre funzionalità. Ogni driver dovrà prendere in input da linea di comando i nomi dei file per il testing:

1. Un file di INPUT
2. Un file ORACOLO
3. Un file con i risultati del Testing

# FILES DEL PROGETTO

1. Il file di **Input** contiene le sequenze da utilizzare per il testing della specifica funzionalità
2. Il file **Oracolo** contiene i risultati corretti rispetto alle sequenze nel file di **Input**
3. Il file di **Output** contiene i risultati del testing in termini di 0 e di 1, dove 0 significa fallimento e 1 successo



# FILES DEL PROGETTO

- **File di interfaccia delle librerie**
  - **utils.h** interfaccia del modulo util, contiene la funzione `scambia()`
  - **vettore.h** interfaccia del modulo vettore, contiene tutte le funzioni realizzate per manipolare gli array
- **File sorgenti .c delle librerie**
  - **utils.c** realizzazione del modulo utile
  - **vettore.c** realizzazione del modulo vettore
- **File sorgenti .c dei driver**
  - **driver\_somma.c** contengono i **driver**
  - **driver\_somma\_v.c**
  - **driver\_prodotto.c**

# FILES DEL PROGETTO

- Test suite (**creati a mano**)
  - input\_somma.txt      contengono i casi di test
  - input\_somma\_v.txt
  - input\_prodotto.txt
- Oracoli (**creati a mano**)
  - Oracle\_somma.txt      contengono i corretti valori attesi nei test\_case
  - Oracle\_somma\_v.txt
  - Oracle\_prodotto.txt
- Output (**creati dal programma**)
  - output\_somma.txt      contengono i valori prodotti dalla esecuzione
  - .....      del programma nei 3 casi di test

# ARGOMENTI SULLA LINEA DI COMANDO

```
tux$ ./driver_sum.exe driver_sum_input.txt driver_sum_oracle.txt driver_sum_output
```

Il `main` ha due parametri

- **Argument counter:** `int argc`
  - Conta il numero di argomenti su linea di comando
- **Argument vector:** `char* argv[]`
  - l'array di stringhe corrispondenti agli argomenti
  - NB: `argv[0]` è il nome del programma
- Quindi, l'interfaccia completa del `main` è la seguente
  - `int main(int argc, char* argv[])`
- Se non servono, `argc` e `argv` possono essere omessi:
  - `int main()`
- Si usa così:
  - > `driver_test.exe File1.txt File2.txt`

# ESEMPIO DI USO DI ARGV

```
tux$ ./driver_sum.exe driver_sum_input.txt driver_sum_oracle.txt driver_sum_output
```

```
int main (int argc, char * argv[]){  
FILE *fp_input;  
  
if((fp_input=fopen(argv[1], "r"))==NULL){  
    fprintf(stderr, "Errore apertura file di INPUT %s\n", argv[1]);  
    exit(EXIT_FAILURE);  
}
```

**argv[1] contiene 'driver\_sum\_input.txt'**



# BOUBLE SORT DRIVER



|   | input.txt                   | oracolo.txt | output.txt |
|---|-----------------------------|-------------|------------|
| 1 | 5                           |             |            |
| 2 | 1 2 3 4 5 6 7 8 9           |             |            |
| 3 | 10 9 8 7 6 5 4 3 2 1        |             |            |
| 4 | 5 8 2 9 10 1 4 7 3 6 12 111 |             |            |
| 5 |                             |             |            |

|   | input.txt                  | oracolo.txt | output.txt |
|---|----------------------------|-------------|------------|
| 1 | 5                          |             |            |
| 2 | 1 2 3 4 5 6 7 8 9          |             |            |
| 3 | 1 2 3 4 5 6 7 8 9 10       |             |            |
| 4 | 1 2 3 4 5 6 7 8 9 10 11 12 |             |            |

|   | input.txt      | oracolo.txt | output.txt |
|---|----------------|-------------|------------|
| 1 | Test case 1: 1 |             |            |
| 2 | Test case 2: 1 |             |            |
| 3 | Test case 3: 1 |             |            |
| 4 | Test case 4: 0 |             |            |
| 5 |                |             |            |

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "vettore.h"
4
5  #define N 50
6
7  int main(void)
8  {
9      FILE *fp_input, *fp_oracolo, *fp_output;
10
11      if((fp_input=fopen("input.txt", "r"))==NULL){
12          //viene assegnato un puntatore al file
13          fprintf(stderr, "Errore apertura file input.txt\n");
14          exit(EXIT_FAILURE);
15      }
16
17      if((fp_oracolo=fopen("oracolo.txt", "r"))==NULL){
18          //viene assegnato un puntatore al file
19          fprintf(stderr, "Errore apertura file oracolo.txt\n");
20          exit(EXIT_FAILURE);
21      }
22
23      if((fp_output=fopen("output.txt", "w"))==NULL){
24          //viene assegnato un puntatore al file output che verrà creato o sovrascritto
25          printf("Errore apertura file output.txt\n");
26          exit(EXIT_FAILURE);
27      }
28
```

```

29 char line[N]; // buffer per contenere i caratteri letti nel file
30 int arr_input[N]; //arr_input -> conterrà una linea del file ..input.txt
31 int arr_oracolo[N]; //arr_oracolo -> contiene una linea del file ..oracolo.txt
32 int n_input; // la dimensione di arr_input[]
33 int n_oracolo; // la dimensione di arr_oracolo[]
34 int test; // è usata per esprimere il risultato del testing: 0 -> fallimento 1 -> successo
35 int i;
36
37 for(i=1; fgets(line, N, fp_input)!=NULL; i++){
38     //La funzione fgets() legge una linea dallo stream immagazzinandola nel buffer puntato da line
39
40     n_input=input_array_str (arr_input, line);
41     //input_array_str estrapola i dati dal buffer puntato da line
42
43     fgets(line, N, fp_oracolo);
44     n_oracolo=input_array_str(arr_oracolo, line);
45
46     bubble_sort(arr_input, n_input);
47     //ordina vettore in input.txt
48
49     test = compare_arrays(arr_input, arr_oracolo, n_input, n_oracolo);
50     //confronta vettore in input.txt e oracolo.txt
51
52     fprintf(fp_output, "Test case %d: %d\n", i, test);
53     //scrive il risultato sul file di output
54 }
55 fclose(fp_input);
56 fclose(fp_output);
57 fclose(fp_oracolo);

```

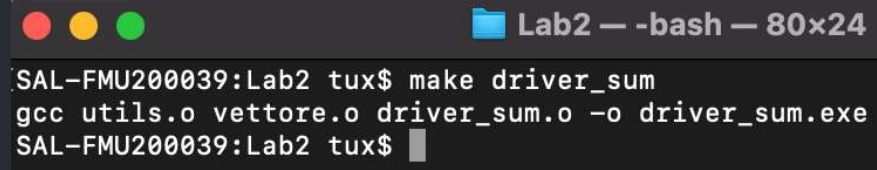
# VETTORE.C

```
53  **** Funzione che carica dati da un buffer puntato da line nell'array di interi arr
54  arr -> vettore di caricamento; line -> buffer; ****/
55  int input_array_str(int *arr, char *line, int){
56      int i=0, n=0, counter=0;
57
58      while(sscanf(line, "%d%n", &arr[i], &n) == 1){ //%n Indica il numero di caratteri trasformati in numeri interi e gli spazi
59          printf("numero letto: %d\n", arr[i]);
60          line+=n;
61          i++;
62      }
63      return i;
64  }
65 }
```

# VETTORE.C (MODIFICATO)

```
53  **** Funzione che carica dati da un buffer puntato da line nell'array di interi arr
54  arr -> vettore di caricamento; line -> buffer; pos -> contiene il numero di caratteri letti in line ****/
55  int input_array_str(int *arr, char *line, int *pos){
56      int i=0, n=0, counter=0;
57      *pos=0;
58      while(sscanf(line, "%d%n", &arr[i], &n)==1){ ///%n Indica il numero di caratteri trasformati in numeri interi e gli spazi
59          printf("numero letto: %d\n", arr[i], n);
60          line+=n;
61          i++;
62          counter+=n; //tiene il conto dei caratteri letti
63      }
64      *pos=counter +1; //pos è a disposizione del programma che ha invocato la funzione ed indica quanti caratteri totali sono stati letti nel buffer puntato da line
65      return i;
66  }
```

```
1 link : utils.o vettore.o main.o
2 gcc utils.o vettore.o main.o -o vettore.exe
3
4 driver: utils.o vettore.o driver.o
5 gcc utils.o vettore.o driver.o -o driver.exe
6
7 driver_sum: utils.o vettore.o driver_sum.o
8 gcc utils.o vettore.o driver_sum.o -o driver_sum.exe
9
10 driver_sum_vecs: utils.o vettore.o driver_sum_vecs.o
11 gcc utils.o vettore.o driver_sum_vecs.o -o driver_sum_vecs.exe
12
13 utils.o :
14 gcc -c utils.c
15
16 vettore.o :
17 gcc -c vettore.c
18
19 main.o :
20 gcc -c main.c
21
22 driver.o :
23 gcc -c driver.c
24
25 driver_sum_vecs.o :
26 gcc -c driver_sum_vecs.c
27
28 driver_sum.o :
29 gcc -c driver_sum.c
30
31 clean:
32 rm -f utils.o vettore.o main.o driver.o driver.exe vettore.exe driver_sum.o driver_sum.exe driver_sum_vecs.o driver_sum_vecs.exe
```



```
Lab2 - -bash - 80x24
SAL-FMU200039:Lab2 tux$ make driver_sum
gcc utils.o vettore.o driver_sum.o -o driver_sum.exe
SAL-FMU200039:Lab2 tux$
```

# ESEMPIO DI ESECUZIONE DA TERMINALE

```
MBP-di-TUX:Ufficiale tux$ make driver_sum
gcc -c utils.c
gcc -c vettore.c
cc -c -o driver_sum.o driver_sum.c
gcc utils.o vettore.o driver_sum.o -o driver_sum.exe
MBP-di-TUX:Ufficiale tux$ ./driver_sum.exe driver_sum_input.txt driver_sum_oracle.txt driver_sum_output
MBP-di-TUX:Ufficiale tux$ cat driver_sum_output
Test case 1: 1
Test case 2: 1
MBP-di-TUX:Ufficiale tux$
```

# ESERCIZI

Implementare e testare l'aggiunta alla libreria vettore delle seguenti funzioni:

1. **Funzione che prende in input un array di interi e restituisce la somma degli elementi dell'array**
2. Funzione che prende in ingresso due array di interi e restituisce in uscita l'array che contiene come elemento di posizione  $i$  la somma degli elementi di posizione  $i$  degli array di input
3. Funzione che prende in ingresso due array di interi e restituisce il **prodotto scalare** dei due array. Il prodotto scalare di due array  $a$  e  $b$  è definito come:

$$\sum_i a[i]*b[i]$$

➤ Scrivere 3 file «driver» per testare le tre funzionalità. Ogni driver dovrà prendere in input da linea di comando i nomi dei file per il testing:

1. Un file di INPUT
2. Un file ORACOLO
3. Un file con i risultati del Testing