

**LABORATORIO DI PROGRAMMAZIONE 1**  
**CORSO DI LAUREA IN MATEMATICA**  
**UNIVERSITÀ DEGLI STUDI DI MILANO**  
**2021–2022**

INDICE

<b>Parte 1. Suddividere i programmi in funzioni</b>	<b>3</b>
Esercizio 1	3
<i>Calcolatrice con menu</i>	3
Tempo: 20 min.	3
Esercizio 2	4
<i>Calcolatrice con menu e modifica degli operandi</i>	4
Tempo: 15 min.	4
Esercizio 3	4
<i>Calcolatrice con modifica degli operandi (implementazione con funzioni)</i>	4
Tempo: 15 min.	4
 <b>Parte 2. Puntatori: primi esercizi, aritmetica dei puntatori</b>	 <b>5</b>
Esercizio 4	5
<i>Scambio di interi</i>	5
Tempo: 5 min.	5
Esercizio 5	5
<i>Scambio di stringhe</i>	5
Tempo: 5 min.	5
Esercizio 6	5
<i>Incrementi e decrementi</i>	5
Tempo: 15 min.	5
 <b>Parte 3. Note sugli array multidimensionali</b>	 <b>6</b>
 <b>Parte 4. Algebra lineare in C</b>	 <b>7</b>
Esercizio 7	7
<i>Matrice trasposta</i>	7
Tempo: 15 min.	7
Esercizio 8	8
<i>Somma di matrici</i>	8
Tempo: 15 min.	8
Esercizio 9	8
<i>Moltiplicazione per uno scalare</i>	8
Tempo: 10 min.	8
Esercizio 10	8

<i>Prodotto di matrici</i>	8
Tempo: 25 min.	8
Esercizio 11	8
<i>Calcolatrice matriciale</i>	8
Tempo: 45 min.	8
Esercizio 12	9
<i>Verifica empirica di alcune identità</i>	9
Tempo: 45 min.	9
Esercizio 13	9
<i>Potenza di una matrice</i>	9
Tempo: 50 min.	9

## Parte 1. Suddividere i programmi in funzioni

### ESERCIZIO 1

*Calcolatrice con menu.*

Tempo: 20 min.

Scrivete un programma che chieda all'utente di inserire due numeri (operandi) di tipo `double`. Il programma visualizza poi il menu seguente.

```
1. Addizione.  
2. Sottrazione.  
3. Moltiplicazione.  
4. Divisione.  
5. Esci.  
>
```

I menu si codificano tramite il costrutto selezione della programmazione strutturata. È conveniente usare l'istruzione `switch` invece di una serie di `if-else-if`, quando le scelte sono più di un paio.

dove `>` è un “prompt” che indica all'utente che la macchina è in attesa dell'input dell'utente. L'utente inserisce una scelta, che il programma acquisisce come dato di tipo `char`. Se la scelta è inesistente, il programma stampa un messaggio d'errore, e visualizza nuovamente il menu. Se la scelta è 5, il programma termina. Se la scelta è 1, 2 o 3, il programma stampa il risultato dell'operazione corrispondente applicata agli operandi, e visualizza nuovamente il menu. Se la scelta è 4 e il divisore è nullo, il programma stampa un messaggio d'errore, e visualizza nuovamente il menu; se il divisore non è nullo, stampa il risultato della divisione applicata agli operandi, e visualizza nuovamente il menu.

#### Variabili di stato e flusso del controllo

La soluzione dell'Esercizio 1 richiede un'iterazione che termini solo nel caso in cui l'utente digiti 5. Vi sono diversi modi per scrivere il codice che implementa una tale iterazione. Una tecnica generale che conduce a buone implementazioni anche in casi più complessi dell'esempio che stiamo esaminando consiste nell'uso di una *variabile di stato*. Si tratta di una variabile il cui valore indica uno specifico stato del programma in esecuzione. Ad esempio, se `st` è una variabile di tipo `int`, possiamo convenire che la condizione `st==1` indichi il fatto che l'esecuzione del programma debba continuare, e `st==0` indichi il fatto che l'esecuzione debba terminare. Allora, in una possibile soluzione dell'Esercizio 1, l'iterazione si può implementare così:

```
int st=1;  
while (st)  
{  
    ...  
}
```

dove, nel corpo del `while`, la variabile `st` è posta a 0 quando l'utente digita 5, o più generalmente quando si verifica una condizione che debba comportare la terminazione del programma.

## ESERCIZIO 2

*Calcolatrice con menu e modifica degli operandi.*

Tempo: 15 min.

Migliorate la calcolatrice scritta per l'Esercizio 1 di modo che l'utente possa modificare il valore degli operandi tramite il menu. In dettaglio, aggiungete al menu la voce:

0. Inserisci operandi.

ed eliminate la lettura iniziale degli operandi — il programma visualizzerà menu e prompt direttamente all'avvio. Se l'utente sceglie 0, il programma richiede l'inserimento dei due operandi, e da quel punto in poi il programma eseguirà le operazioni richieste sui due dati immessi, fino a quando l'utente non si avvarrà nuovamente dell'opzione 0. Se l'utente sceglie di eseguire una qualunque delle quattro operazioni prima di aver inserito gli operandi, il programma visualizza un messaggio d'errore, e visualizza nuovamente il menu.

Usate una variabile apposita per tenere traccia del fatto che l'utente abbia o non abbia ancora impostato i valori degli operandi. Si tratta di un secondo esempio di variabile di stato, cfr. il riquadro alla pagina precedente.

**Calcolatrice: implementazione con funzioni**

La traccia dell'esercizio seguente è identica a quella dell'esercizio precedente. Adesso dovreste però implementare il programma spezzandolo in funzioni in modo adeguato. In particolare, prevederete funzioni apposite per la visualizzazione del menu e per l'esecuzione delle operazioni. Per esempio, nell'eseguire una somma la procedura `main` invocherà una funzione di prototipo `double somma(double, double)`.

## ESERCIZIO 3

*Calcolatrice con modifica degli operandi (implementazione con funzioni).*

Tempo: 15 min.

(Come nell'Esercizio 2, ma spezzando il programma in funzioni.)

**Parte 2. Puntatori: primi esercizi, aritmetica dei puntatori**

## ESERCIZIO 4

*Scambio di interi.*

Tempo: 5 min.

Scrivete un programma che dichiari due puntatori di tipo `int` di nome `px` e `py`, assegni al contenuto delle locazioni di memoria puntate i valori 0 e 1, rispettivamente, e poi scambi i valori in modo che `px` punti al valore precedentemente puntato da `py` e viceversa. Per eseguire lo scambio usate un terzo puntatore a `int` ausiliario. Eseguite le stampe appropriate per verificare che lo scambio di valori sia avvenuto correttamente.

## ESERCIZIO 5

*Scambio di stringhe.*

Tempo: 5 min.

Ripetete l'esercizio precedente usando puntatori a `char` (inizializzati in modo da puntare a stringhe di caratteri) in luogo di puntatori a `int`. Dichiarate le stringhe di caratteri direttamente nel codice, senza leggerle da tastiera. Per esempio, dichiarate e inizializzate una stringa in questo modo:

```
char s[] = "Galileo Galilei";
```

## ESERCIZIO 6

*Incrementi e decrementi.*

Tempo: 15 min.

Scrivete un programma che dichiari un puntatore di tipo `int` e di nome `pi`, un puntatore di tipo `double` e di nome `pd` e un puntatore di tipo `char` e di nome `pc`. Definite poi degli array di `int`, `double` e `char` come segue:

```
int ai[]={0,1};  
double ad[]={0.1,0.2};  
char ac[]="pippo";
```

Fate puntare `pi`, `pd` e `pc` al primo elemento dell'array di tipo corrispondente, e visualizzatelo tramite `printf` usando il puntatore. Incrementate ciascun puntatore di uno e visualizzate il valore cui esso punta. Visualizzate anche i valori dei puntatori — usate `%p` con `printf` — prima e dopo gli incrementi.

Dopo l'esecuzione della soluzione all'Esercizio 6, di quante locazioni di memoria è stato incrementato ciascun puntatore? La risposta dipende dal tipo del puntatore? Sperimentate con altre operazioni aritmetiche sui puntatori.

### Parte 3. Note sugli array multidimensionali

Nell'ultima parte del laboratorio odierno farete una serie di esercizi con gli array multidimensionali. Abbiamo discusso a lezione gli array monodimensionali. L'uso degli array multidimensionali è del tutto simile. Ecco qualche indicazione sintetica; useremo come esempio gli array bidimensionali di tipo `int`.

- **Dichiarazione e allocazione in memoria.** La dichiarazione

```
int M[2][2]; (*)
```

riserva spazio in memoria per 4 valori di tipo `int`, organizzati in una matrice  $2 \times 2$  le cui righe e le cui colonne sono indicate a partire da zero. Il primo indice è quello delle righe, il secondo quello delle colonne. L'istruzione dichiara che `M` è un identificatore di tipo *array di array di int*.

- **Accesso agli elementi.** L'espressione:

```
M[i][j]
```

con  $i, j \in \{0, 1\}$ , denota l'elemento di riga  $i$  e colonna  $j$  dell'array `M` dichiarato in  $(*)$ , ed ha dunque tipo `int`.

- **Inizializzazione e scansione.** L'inizializzazione e la scansione di un array multidimensionale si esegue di solito tramite l'uso di due o più cicli `for` annidati. La scansione può avvenire per righe (fissare la riga e scandire le colonne) o per colonne (fissare la colonna e scandire le righe). Per esempio, per inizializzare a  $i * j$  l'elemento alla riga  $i$  e colonna  $j$  di `M`, scandendo i valori contenuti in `M` per righe, si può usare:

```
for (i=0; i<2; i++)
    for (j=0; j<2; j++)
        M[i][j]=i*j;
```

Per visualizzare il contenuto di `M` in forma di matrice, analogamente, si scriverà per esempio:

```
for (i=0; i<2; i++)
{
    for (j=0; j<2; j++)
        printf("%d\t", M[i][j]);
    printf("\n");
}
```

- **Inizializzatori.** Si può anche usare una lista di liste di valori per inizializzare `M`, ma solo al momento della sua dichiarazione. Per esempio, per inizializzare `M` alla matrice identità  $2 \times 2$  si può sostituire  $(*)$  con:

```
int M[][] = { {1,0} , {0,1} }
```

## Parte 4. Algebra lineare in C

### Qualche richiamo preliminare

Si considerino due matrici quadrate  $A := (a_{ij})_{i,j=1}^n$  e  $B := (b_{ij})_{i,j=1}^n$  di numeri reali, ciascuna di dimensione  $n \times n$  per un intero  $n \geq 1$ .

- La *trasposta* di  $A$  è la matrice:

$$A^t := (a_{ji})_{i,j=1}^n.$$

- La *somma* di  $A$  e  $B$  è data da:

$$A + B := (a_{ij} + b_{ij})_{i,j=1}^n.$$

- La *moltiplicazione* di  $A$  per lo scalare  $\lambda \in \mathbb{R}$  è data da:

$$\lambda A := (\lambda a_{ij})_{i,j=1}^n.$$

- Il *prodotto* di  $A$  per  $B$  è dato da:

$$AB := (c_{ij})_{i,j=1}^n,$$

$$\text{dove } c_{ij} := \sum_{k=1}^n a_{ik} b_{kj}.$$

La somma di matrici è un'operazione commutativa e associativa. Il prodotto di matrici è un'operazione associativa, ma non commutativa. Il prodotto si distribuisce sulla somma: se  $C$  è una matrice di numeri reali di dimensione  $n \times n$ , si ha

$$A(B + C) = AB + AC. \quad (1)$$

La moltiplicazione per uno scalare è compatibile col prodotto di matrici:

$$\lambda(AB) = (\lambda A)B. \quad (2)$$

La trasposta del prodotto di due matrici è il prodotto delle trasposte preso in ordine inverso:

$$(AB)^t = B^t A^t \quad (3)$$

Negli esercizi di questa prima parte userete gli array multidimensionali di `double` per implementare le nozioni di algebra lineare richiamate sopra. Quando la traccia specifica: “Leggete due matrici  $n \times n$  inserite dall'utente”, si intende che il programma deve 1) Acquisire dall'utente l'intero  $n \geq 1$ , e poi 2) Acquisire dall'utente le due matrici  $n \times n$ .

### ESERCIZIO 7

*Matrice trasposta.*

Tempo: 15 min.

Scrivete un programma che legga una matrice  $n \times n$  inserita dall'utente, ne calcoli la matrice trasposta, visualizzi entrambe le matrici e termini.

## ESERCIZIO 8

*Somma di matrici.*

Tempo: 15 min.

Scrivete un programma che legga due matrici  $n \times n$  inserite dall'utente, ne calcoli la somma, visualizzi gli addendi e la somma e termini.

## ESERCIZIO 9

*Moltiplicazione per uno scalare.*

Tempo: 10 min.

Scrivete un programma che legga una matrice  $A$  di dimensione  $n \times n$  inserita dall'utente, legga poi un numero reale  $\lambda \in \mathbb{R}$  (tipo `double` come per gli elementi della matrice), calcoli la moltiplicazione  $\lambda A$ , visualizzi il risultato e termini.

## ESERCIZIO 10

*Prodotto di matrici.*

Tempo: 25 min.

Scrivete un programma che legga due matrici  $n \times n$  inserite dall'utente, ne calcoli il prodotto, visualizzi i fattori e il prodotto e termini.

## ESERCIZIO 11

*Calcolatrice matriciale.*

Tempo: 45 min.

Si tratta di mettere assieme in un unico programma il codice che avete scritto per risolvere gli Esercizi 7–10. Sarebbe sensato spezzare il programma in funzioni: poiché però non abbiamo ancora discusso alcuni aspetti tecnici del passaggio di matrici multidimensionali come parametri alle funzioni, non potrete adottare questa soluzione, a meno che non vogliate consultare fin d'ora le parti rilevanti del libro di testo.

Scrivete un programma che acquisisca dall'utente un numero intero  $n \geq 1$ , e presenti poi all'utente il menu seguente, dove  $A$  e  $B$  denotano matrici di numeri reali, ciascuna di dimensione  $n \times n$ , ed  $r$  denota un numero reale.

1. Inserisci operando  $A$ .
2. Inserisci operando  $B$ .
3. Inserisci operando scalare  $r$ .
4. Visualizza  $A$ ,  $B$  ed  $r$ .
5. Trasposta di  $A$ .
6.  $rA$ .
7.  $A+B$ .
8.  $AB$ .
9. Scambia  $A$  e  $B$ .
10. Esci.

Il programma attende la scelta dell'utente ed esegue l'operazione richiesta. Nel caso in cui vi siano dei dati da visualizzare (scelte 4, 5, 6, 7, 8) il programma li visualizza.



Il programma torna poi al menu. Il programma termina quando l'utente sceglie 10. Nel caso in cui l'utente richieda operazioni che non si possono portare a termine perché alcuni degli operandi necessari non sono stati ancora inseriti, il programma informa l'utente della circostanza e torna al menu. Ciò vale anche per la voce 4: saranno visualizzati solo quei dati fra  $A$ ,  $B$  e  $r$  che sono già stati inseriti dall'utente.

## ESERCIZIO 12

*Verifica empirica di alcune identità.*

Tempo: 45 min.

Scrivete un programma che legga in ingresso tre matrici  $A$ ,  $B$ ,  $C$  e uno scalare  $r \in \mathbb{R}$ . Per ciascuna delle identità (1–3), il programma calcola membro destro e membro sinistro, visualizza i due risultati — che dovrebbero essere uguali — e termina.

## ESERCIZIO 13

*Potenza di una matrice.*

Tempo: 50 min.

Scrivete un programma che legga in ingresso una matrice  $A$  e un intero  $m \geq 0$ . Il programma calcola la matrice

$$A^m := \underbrace{A \times A \times \cdots \times A}_{m \text{ volte}},$$

visualizza sia  $A$  che  $A^m$ , e termina.

Per calcolare la potenza iterate il prodotto di  $A$  con se stessa  $m$  volte. Troverete utile impiegare una matrice ausiliaria per memorizzare il risultato parziale  $A^k A$ , per  $k \in \{0, \dots, m-1\}$ .

Nell'Esercizio 13 è ammesso l'esponente  $m = 0$ . Come si comporta in questo caso il vostro programma? Si rammenti che si ha

$$A^0 = I_n$$

per una qualunque matrice  $n \times n$  di numeri reali, con  $n$  un intero  $n \geq 1$ , dove  $I_n := (\delta_{ij})_{i,j=1}^n$  è la matrice identità  $n \times n$ :

$$\delta_{ij} := \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

Assicuratevi che il vostro programma produca l'output corretto a fronte dell'input  $m = 0$ . Se la procedura iterativa di calcolo della potenza  $A^m$  è implementata in modo appropriato, non è necessario eseguire una selezione per trattare  $m = 0$  come caso particolare.