

LABORATORIO DI PROGRAMMAZIONE
CORSO DI LAUREA IN SICUREZZA INFORMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO

2025–2026

INDICE

Parte 1. L'algoritmo euclideo	3
Esercizio 1	3
<i>L'algoritmo euclideo</i>	3
Tempo: 15 min.	3
Esercizio 2	4
<i>L'algoritmo euclideo: input dell'utente</i>	4
Tempo: 15 min.	4
Esercizio 3	4
<i>L'algoritmo euclideo: validazione dell'input</i>	4
Tempo: 15 min.	4
Esercizio 4	4
<i>L'algoritmo euclideo: conteggio delle iterazioni</i>	4
Tempo: 10 min.	4
Parte 2. Esercizi facili con la programmazione strutturata	5
Esercizio 5	5
<i>Somma dei primi n numeri naturali</i>	5
Tempo: 20 min.	5
Esercizio 6	5
<i>Somma dei primi n quadrati</i>	5
Tempo: 5 min.	5
Esercizio 7	5
<i>Somma dei primi n cubi</i>	5
Tempo: 5 min.	5
Esercizio 8	6
<i>Equazioni di secondo grado</i>	6
Tempo: 25 min.	6
Parte 3. Analisi dei caratteri con l'intestazione ctype.h	7
Esercizio 9	7
<i>Lettere maiuscole e minuscole</i>	7
Tempo: 15 min.	7
Esercizio 10	7
<i>Lettere maiuscole e minuscole iterate</i>	7

Tratti dagli esercizi del corso del prof. Vincenzo Marra.
Ultima revisione: 10 ottobre 2025.

Tempo: 15 min.	8
Esercizio 11	8
<i>Conversione in maiuscola</i>	8
Tempo: 10 min.	8
Esercizio 12	8
<i>Conversione in maiuscola o minuscola</i>	8
Tempo: 15 min.	8
Esercizio 13	8
<i>Conversione in maiuscola o minuscola con controllo d'errore</i>	8
Tempo: 5 min.	8

Parte 1. L'algoritmo euclideo

ESERCIZIO 1

L'algoritmo euclideo.

Tempo: 15 min.

La Figura 1 mostra un’implementazione parziale dell’algoritmo euclideo delle sottrazioni successive per il calcolo del massimo comun divisore di due numeri naturali. Completatela.

Euclide.c

```

1 #include <stdio.h>
2
3 /*****
4 * Calcola il m.c.d di due interi a, b > 0 applicando *
5 * l'algoritmo delle sottrazioni successive di Euclide. *
6 *****/
7
8 int main(void)
9 {
10     int a=214, b=128; //Deve essere a,b > 0
11
12     printf("Il m.c.d. di %d e %d e': ",a,b);
13
14     while (a != b)    //Fino a quando a e b sono diversi...
15         if (a > b)   //Se a > b,
16             ....       //sostituisci a con a-b.
17         else          //Altrimenti,
18             ....       //sostituisci b con b-a.
19
20     printf("%d\n",a);
21     return 0;
22 }
```

E poi *sperimentate* con i possibili valori delle variabili: cosa succede se non si rispetta la specifica $a, b > 0$ per i valori in ingresso? Vi sono casi in cui il programma *non termina*?

FIGURA 1. L'algoritmo euclideo delle sottrazioni successive in C.

“M’è andato in loop il programma...” — `ctrl+C`

Così si dice a volte in gergo per indicare la circostanza che, a fronte di certi valori in ingresso, il programma in questione non termini. La locuzione è dovuta al fatto che spesso (ma non sempre, naturalmente) la mancata terminazione del programma è dovuta a un ciclo — in inglese, *loop* — che continua indefinitamente a eseguire iterazioni. La sequenza `ctrl+C` invia un *segnaletica di terminazione* al programma in esecuzione, costringendolo^a a interrompere l’esecuzione.

^aÈ possibile che un programma si rifiuti di terminare nonostante glielo si intimi con `ctrl+C`. Senza entrare nei dettagli, basterà dire che il codice scritto da voi in questo corso risponderà sempre a `ctrl+C` con la terminazione.

ESERCIZIO 2

L’algoritmo euclideo: input dell’utente.

Tempo: 15 min.

Usate la funzione `scanf` per leggere i valori digitati dall’utente.

Modificate la vostra soluzione all’Esercizio 1 in modo che sia l’utente a inserire da terminale i valori delle variabili `a` e `b`, all’inizio dell’esecuzione.

ESERCIZIO 3

L’algoritmo euclideo: validazione dell’input.

Tempo: 15 min.

Quale costrutto della programmazione strutturata è necessario usare qui? Sequenza, iterazione o selezione?

Modificate la vostra soluzione all’Esercizio 2 in modo che il programma si assicuri che i valori di `a` e `b` inseriti dall’utente soddisfino le precondizioni: $a, b > 0$.

Nel caso ciò non avvenga, visualizzate un appropriato messaggio d’errore per l’utente e terminate immediatamente l’esecuzione con l’istruzione `return -1`.

ESERCIZIO 4

L’algoritmo euclideo: conteggio delle iterazioni.

Tempo: 10 min.

Modificate la vostra soluzione all’Esercizio 3 in modo che, al termine del calcolo del m.c.d., il programma visualizzi il numero di iterazioni `while` che ha eseguito. Usate una variabile intera per tenere traccia del numero di iterazioni. Incrementate di una unità il valore della variabile ad ogni successiva iterazione.

Restituire il valore `-1` segnala convenzionalmente al sistema operativo una condizione eccezionale di terminazione della funzione `main`. Ne ripareremo più avanti nel corso.

Parte 2. Esercizi facili con la programmazione strutturata

Tre sommatorie facili

Sia dato un intero $n \geq 1$. Valgono le tre identità:

$$(1) \quad \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$(2) \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$(3) \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$$

ESERCIZIO 5

Somma dei primi n numeri naturali.

Tempo: 20 min.

Scrivere un programma che chieda in ingresso all'utente un intero $n \geq 1$. Se l'intero inserito non soddisfa questa diseguaglianza, il programma chiede il reinserimento fino a che la condizione non sia verificata. Il programma visualizza poi la somma dei primi n numeri naturali calcolata con un ciclo **for**, assieme alla stessa somma calcolata tramite la formula al membro destro di (1).

ESERCIZIO 6

Somma dei primi n quadrati.

Tempo: 5 min.

Scrivere un programma che chieda in ingresso all'utente un intero $n \geq 1$. Se l'intero inserito non soddisfa questa diseguaglianza, il programma chiede il reinserimento fino a che la condizione non sia verificata. Il programma visualizza poi la somma dei primi n quadrati calcolata con un ciclo **for**, assieme alla stessa somma calcolata tramite la formula al membro destro di (2).

ESERCIZIO 7

Somma dei primi n cubi.

Tempo: 5 min.

Scrivere un programma che chieda in ingresso all'utente un intero $n \geq 1$. Se l'intero inserito non soddisfa questa diseguaglianza, il programma chiede il reinserimento fino a che la condizione non sia verificata. Il programma visualizza poi la somma dei primi n cubi calcolata con un ciclo **for**, assieme alla stessa somma calcolata tramite la formula al membro destro di (3).

```

1 #include <stdio.h>
2 #include <math.h> //Intestazione necessaria per double sqrt(double)
3
4 int main(void)
5 {
6     double d;
7     do
8     {
9         printf("Inserisci un numero reale non negativo: ");
10        scanf("%lf",&d);
11    } while (d<0);
12
13    printf("La radice quadrata di %g e' %g.\n",d,sqrt(d));
14    return 0;
15 }
```

FIGURA 2. Esempio d’uso della funzione `sqrt` della libreria standard.

ESERCIZIO 8

Equazioni di secondo grado.

Tempo: 25 min.

Scrivere un programma che permetta di calcolare le soluzioni reali delle equazioni di secondo grado a coefficienti reali. Considereremo quindi equazioni della forma

(4)
$$ax^2 + bx + c = 0$$

con $a, b, c \in \mathbb{R}$, $a \neq 0$, e soluzioni in \mathbb{R} .

All’avvio, il programma chiede all’utente di inserire i coefficienti `a`, `b` e `c`, memorizzandoli in variabili di tipo `double`. Se il valore di `a` è zero il programma chiede all’utente di reinserire `a` fino a che $a \neq 0$. Il programma calcola quindi il discriminante

$$\Delta := b^2 - 4ac$$

di (4), e ne visualizza il valore. Basandosi sull’analisi del segno di Δ il programma comunica all’utente il numero delle soluzioni reali di (4). Nel caso vi siano soluzioni reali, il programma le calcola e le visualizza prima di terminare.

Per calcolare le soluzioni di (4) dovete estrarre una radice quadrata. A tal fine potrete usare la funzione `sqrt()` (contrazione di *square root*) del file di intestazione `math.h`. La Figura 2 riporta il codice sorgente di un programma che estrae la radice quadrata di un numero reale non negativo.

Richiesta di *linking* delle librerie matematiche

Alcune implementazioni del compilatore C prevedono che si richieda esplicitamente l’uso delle librerie matematiche con l’opzione `-lm`, che sta per *link mathematical library*. Per compilare il codice in Figura 2 dovete quindi invocare il compilatore in questo modo:

`gcc -lm radq.c`

Parte 3. Analisi dei caratteri con l'intestazione ctype.h

Classi di caratteri

Il file di intestazione `ctype.h` della libreria standard contiene funzioni di utilità per l'analisi dei caratteri. Ad esempio, la funzione `int isalpha(char c)` restituisce un intero non nullo se `c` è un carattere alfabetico, e zero altrimenti. La funzione `int isdigit(char c)` restituisce un intero non nullo se `c` è una cifra (cioè un carattere nell'insieme `{'0', '1', ..., '9'}`) e zero altrimenti. La funzione `int isupper(char c)` restituisce un intero non nullo se `c` è una lettera maiuscola, e zero altrimenti; la funzione `int islower(char c)` è analoga per le lettere minuscole. La funzione `int toupper(char c)` restituisce (come `int`) il carattere `c` convertito in maiuscola, se esso è un carattere alfabetico; altrimenti, poiché non ha senso convertire in maiuscola un carattere non alfabetico, la funzione restituisce il carattere `c` privo di modifiche. La funzione `int tolower(char c)` è analoga per le lettere minuscole.

ESERCIZIO 9

Lettere maiuscole e minuscole.

Tempo: 15 min.

Scrivete un programma che chieda all'utente di inserire un carattere, e visualizzi poi le informazioni seguenti. Se il carattere non è alfabetico, il programma scrive in uscita Il carattere `c` non è alfabetico, dove `c` sta per il carattere digitato dall'utente. Se il carattere è alfabetico, il programma scrive in uscita Il carattere `c` è una lettera maiuscola, oppure Il carattere `c` è una lettera minuscola, a seconda del caso che si verifica. Una volta visualizzate le informazioni appropriate, il programma termina. (*Suggerimenti.* Occorre includere `ctype.h` con la direttiva `#include <ctype.h>`. Come esempio d'uso delle funzioni definite in questo file di intestazione, si consideri la chiamata

```
ris=isupper('A');
```

dove `ris` è una variabile di tipo `int`. Dopo la chiamata `ris` contiene un valore non nullo, perché '`A`' è un carattere maiuscolo. Si consideri d'altronde la chiamata

```
c=tolower('A');
```

dove `c` è una variabile di tipo `char`. Dopo la chiamata `c` contiene il carattere '`a`'.)

Usate la funzione `getchar()` di `stdio.h`. Se `car` è una variabile di tipo `char`, l'istruzione
`c=getchar();`
legge il carattere digitato dall'utente.

ESERCIZIO 10

Lettere maiuscole e minuscole iterate.

Tempo: 15 min.

Modificate la soluzione dell’Esercizio 9 di modo che, analizzato il primo carattere inserito dall’utente e visualizzate le informazioni relative, il programma torni a chiedere all’utente l’inserimento di un nuovo carattere da analizzare. (Ciò vale anche nel caso in cui il carattere inserito non sia alfabetico.) L’iterazione continua fino a che l’utente non preme direttamente il tasto INVIO alla richiesta di inserimento del carattere. A quel punto, il programma termina.

Il carattere restituito da `getchar()` quando l’utente digita INVIO è `\n`.

ESERCIZIO 11

Conversione in maiuscola.

Tempo: 10 min.

Scrivete un programma che chieda all’utente di inserire un carattere, e lo visualizzi convertito in maiuscola. Se il carattere inserito non è alfabetico il programma lo visualizza invariato.

ESERCIZIO 12

Conversione in maiuscola o minuscola.

Tempo: 15 min.

Modificate la soluzione dell’Esercizio 11 di modo che il programma, dopo aver acquisito il carattere dall’utente, chieda all’utente se intende eseguire una conversione in maiuscola o in minuscola, e proceda poi di conseguenza. (*Suggerimento.* Visualizzate la frase `Conversione in maiuscola o in minuscola? (1=maiuscola, altro numero=minuscola)` e leggete un intero. Confrontatelo con 1 usando l’operatore `==` nella condizione di una selezione `if-else`.

ESERCIZIO 13

Conversione in maiuscola o minuscola con controllo d’errore.

Tempo: 5 min.

Modificate la soluzione dell’Esercizio 12 di modo che il programma visualizzi un messaggio d’errore appropriato nel caso in cui il carattere inserito dall’utente non sia alfabetico, e termini.

Qual è il *primo* punto del codice nella soluzione dell’Esercizio 13 in cui è sensato inserire il controllo sulla natura alfabetica del carattere inserito dall’utente? Poiché il testo dell’esercizio non specifica altrimenti, è opportuno assicurarsi di aver inserito il controllo in questo punto: è questa la codifica più naturale, *ceteris paribus*.

Chiedetevi: cosa fa il mio programma se l’utente inserisce un dato inatteso, come per esempio il carattere `a`? Un buon programma tiene conto dell’eventualità che l’utente si comporti in modo inatteso — esso è, come si suol dire, “a prova di sciocco” (*foolproof*, in inglese). In questo esercizio, e in ogni caso in questa fase iniziale del corso, non è importante che il vostro programma si comporti in modo ragionevole a fronte di *qualunque* valore in ingresso: è però importante che acquisiate l’abitudine a rendere esplicite le assunzioni che fate sui dati in ingresso.