LABORATORIO DI PROGRAMMAZIONE
CORSO DI LAUREA IN SICUREZZA DEI SISTEMI E DELLE RETI INFORMATICHE
UNIVERSITÀ DEGLI STUDI DI MILANO
2025–2026

Indice

Tratti dagli esercizi del corso del prof. Vincenzo Marra.
Ultima revisione: 8 novembre 2025.

Parte 1. Suddividere i programmi in funzioni

Esercizio 1

Lettura input.

Tempo: 10 min.

Scrivete un programma che chiede all'utente due numeri interi > 0 e ne calcola la somma. La lettura dell'input dev'essere implementata usando una funzione `int read_int()`, che chiede all'utente di inserire un numero. La funzione continua a chiedere di inserire finché non è > 0, e restituisce il numero letto.

Usate il costrutto di iterazione più adatto.

Esercizio 2

La successione di Fibonacci (*versione iterativa*).

Tempo: 15 min.

La Figura 1 mostra il sorgente parziale di un programma in C che calcola l'n-esimo termine della successione di Fibonacci. Implementate una funzione con prototipo `int fibonacci(int)` che calcoli tale valore, sulla base del codice incompleto indicato nella figura. Aggiungere anche una funzione con prototipo `int read_int()` per la lettura del numero di cui occorre calcolare la successione. Assicuratevi del corretto funzionamento con qualche esperimento; in Tabella 1 sono tabulati i primi termini della successione di Fibonacci.

Potete riutilizzare, copiando, la funzione definita per l'esercizio 1?

n	1	2	3	4	5	6	7	8	9	10	11	12	13
F_n	1	1	2	3	5	8	13	21	34	55	89	144	233

Tabella 1. I primi termini della successione di Fibonacci

Esercizio 3

Operazioni su un numero.

Tempo: 20 min.

Scrivete un programma che chiede all'utente di inserire un numero $n > 0$ di tipo `int`. Il programma poi *i)* calcola *i)* fattoriale di n, *ii)* conta il numero di cifre di n. Ciascuna di queste due operazioni dev'essere implementata usando una funzione.

Dovete eseguire delle divisioni successive per un certa numero fisso.

Fattoriale

Dato n, il suo fattoriale, rappresentato come $n!$, si calcolato come la produttoria successiva dei primi n numeri naturali a partire da 1.

$$n! = \prod_{k=1}^{n} = 1 \times 2 \times 3 \times \ldots \times n \tag{1}$$

fibonacci.c

```c
#include <stdio.h>

int main(void)
{
        int n;          //il programma calcola F(n)

        // LEGGERE n, che dev'essere > 0

        printf("F(%d)=",n);

        int i;              //contatore del ciclo for
        int f_i=1,f_prec=1; //valori iniziali F(2)=F(1)=1

        for (i=3;i<=n;i++)  //il ciclo comincia da i=3
        {
                int aux=f_i;    //variabile ausiliaria aux
                ...;            //somma ad f_i il valore di f_prec
                f_prec=aux;     //f_prec = f_i prima dell'iterazione
        }
        printf("%d\n",f_i);

        return 0;
}
```

FIGURA 1. Una implementazione iterativa in C della successione di Fibonacci. L'implementazione è completa, eccezion fatta per la riga 21 e la lettura dell'input.

ESERCIZIO 4

Calcolatrice con menu.

Tempo: 30 min.

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.

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.
>
```

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 o sbagliata, 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. Ogni operazione dev'essere implementata tramite una funzione. Per esempio, nell'eseguire una somma la procedura `main` invocherà una funzione di prototipo `double somma(double,double)`.

Variabili di stato e flusso del controllo

La soluzione dell'Esercizio 4 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 4, 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 5

Calcolatrice con menu e modifica degli operandi.

Tempo: 15 min.

Migliorate la calcolatrice scritta per l'Esercizio 4 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. Ogni operazione dev'essere eseguita tramite funzioni.

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.

Esercizio 6

Banca con menu.

Tempo: 30 min.

Scrivete un programma che simula le operazioni di una banca. All'avvio, il programma chiede all'utente di inserire il saldo iniziale del conto corrente, di tipo `double`. Il saldo può essere negativo. In seguito, il programma visualizza il menu seguente.

```
1. Visualizza saldo.
2. Deposito.
3. Prelievo.
4. Calcolo interesse.
5. Esci.
>
```

dove `>` è un "prompt" che indica all'utente che la macchina è in attesa dell'input dell'utente. L'utente inserisce una scelta. Se la scelta è inesistente o sbagliata, 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`, il programma visualizza il saldo corrente. Se la scelta è `2`, il programma chiede all'utente quanto denaro intende depositare, il quale è di tipo `double` e > 0. Il programma aggiorna quindi il saldo del conto e lo mostra all'utente. Se il denaro da depositare è ≤ 0, il programma visualizza un messaggio d'errore e mostra nuovamente il menu. Se la scelta è `3`, il programma chiede all'utente quanto denaro intende prelevare, il quale è di tipo `double` e > 0 e $\leq$ del saldo corrente. Il programma aggiorna quindi il saldo del conto e lo mostra all'utente. Se l'utente vuole prelevare più di quanto ha sul conto, o inserisce un numero ≤ 0, il programma visualizza un messaggio d'errore appropriato, e mostra nuovamente il menu. Se la scelta è `4`, il programma chiede all'utente di inserire per quanti anni prevede di lasciare il proprio denaro sul conto, e calcola gli interessi che maturerebbero assumendo un tasso fisso noto a priori. Il numero di anni dev'essere di tipo `int` e > 0. Il programma mostra quindi il saldo e l'interesse ipotetico. Questo calcolo non è possibile se l'utente ha saldo negativo, e dev'esserne informato. Dopo aver completato ogni operazione secondo le alternative `1`–`4`, con successo oppure no, il programma mostra nuovamente il menu. Implementate il programma usando delle funzioni che prendono in ingresso il saldo attuale e restituiscono il nuovo saldo, oppure `void`, a seconda di ciò che la funzione fa.

L'interesse indica il guadagno ottenuto lasciando il proprio denaro sul conto corrente.

Non dovete usare variabili con scope globale né puntatori.

Interesse

Assumendo che il proprio saldo sia s e che si intenda lasciare il proprio denaro sul conto per y anni con un tasso t, l'interesse i si calcola come

$$i = s \times t \times y \tag{2}$$

Di conseguenza, lasciando il proprio denaro sul conto si avrebbe un saldo di $s + i$.

Naturalmente, il calcolo descritto è una semplificazione.

finderror.c

```c
#include <stdio.h>

double avg(int n1, int n2){
    return (n1 + n2) / 2.0;
}

int main(){
    int n1, n2;
    int result;

    printf("Inserisci i due numeri separati da spazio: ");
    scanf("%d %d", &n1, &n2);

    result = avg(n1, n2);
    printf("avg(%d, %d)=%d", n1, n2, result);

    return 0;
}


```

FIGURA 2. Un programma che vorrebbe calcolare la media di due numeri.

ESERCIZIO 7

Trova l'errore.

Tempo: 10 min.

In Figura 2 trovate un programma che *dovrebbe* calcolare la media di due numeri, entrambi inseriti dall'utente. Il programmatore ha commesso un errore. Cercate di correggere il programma e farlo funzionare.

Oltre a leggere il codice per capire dov'è l'errore, provate a eseguire il programma fornito, e osservate se l'errore si manifesta sempre, oppure solo con certi input.

Parte 2. Array multidimensionali

ESERCIZIO 8

Diagonale.

Tempo: 5 min.

In Figura 3 trovate una porzione di programma che calcola la somma dei valori nella diagonale maggiore di una matrice numerica. Completatelo e verificatene il funzionamento.

La *diagonale maggiore* identifica la diagonale di una matrice che va dall'angolo in alto a sinistra a quello in basso a destra.

ESERCIZIO 9

Diagonale (versione efficiente).

diagonale.c

```c
#define M 3
#define N 3

int main(){
    int matrix[M][N]; // definite gli elementi della matrice.
    int i, j;
    int sum = 0;

    for (i = 0; i < M; i++){
        for (j = 0; j < N; j++){
            if (i == j){
                sum += matrix[i][j];
            }
        }
    }

    return 0;
}
```

FIGURA 3. Una implementazione parziale per il calcolo della somma dei valori della diagonale maggiore di una matrice.

Tempo: 10 min.

L'esercizio 8 utilizza 2 cicli `for`. Scrivetene una versione più efficiente, che utilizza un solo ciclo `for`.

1. ESERCIZIO 10

Diagonali.

Tempo: 10 min.

Scrivete un programma che, data una matrice numerica, ne trovi l'elemento l'elemento maggiore e minore. Il programma deve anche mostrare all'utente l'indice di riga e colonna corrispondente.

2. ESERCIZIO 11

Trasposizione.

Tempo: 15 min.

Una matrice *quadrata* è una matrice con lo stesso numero di righe e colonne

Scrivete un programma che, data una matrice numerica quadrata, verifichi se la matrice è *simmetrica*.

Matrice simmetrica

Una matrice quadrata A è *simmetrica* se è uguale, elemento per elemento, alla sua trasposta A^T. La *trasposta* A^T è la matrice ottenuta scambiando le

righe con le colonne di A. Ad esempio:

$$A = \begin{matrix} 1 & 2 & 3 \\ 11 & 22 & 33 \\ 10 & 20 & 30 \end{matrix} \tag{3}$$

$$A^T = \begin{matrix} 1 & 11 & 10 \\ 2 & 22 & 20 \\ 3 & 33 & 30 \end{matrix} \tag{4}$$

in questo caso A *non* è simmetrica perché non è uguale ad A^T. Ad esempio $A[0][2]$ (cioè 3) è diverso da $A^T[0][2]$ (cioè 10). Nel seguente esempio, invece A e A^T sono uguali, quindi A è simmetrica.

$$A = \begin{matrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{matrix} \tag{5}$$

$$A^T = \begin{matrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{matrix} \tag{6}$$