



Programmazione 1 (Canale O-Z)



Laboratorio 1: Fondamenti ed Espressioni

Docente: Prof. Antonino Furnari

Email: `antonino.furnari@unict.it`

Sito Web: antoninofurnari.github.io/prog1-2627



Come Funzionano i Laboratori

- Applichiamo la teoria vista in aula a problemi pratici, **in locale** sul PC personale o della sala.
- Ogni laboratorio ha **due esercizi**:
 - **Demo** (`lab1-demo`): svolto in aula **insieme al docente** (live coding).
 - **Assignment** (`lab1-assignment`): da svolgere **in autonomia**, in aula o a casa.
- La traccia è un **mini-esame**: un preambolo con il modello matematico e poi punti sequenziali (**PUNTO A, B, C, ...**).
- **Nessuno scheletro**: si parte da un file `.c` **vuoto**, dall' `#include` al `return` del `main`.
- Gli assignment **non sono valutati**: chiedete feedback sul vostro codice a ricevimento o nelle sessioni successive.



Il Materiale del Laboratorio

- Entrambi gli archivi si scaricano dalla **pagina del Laboratorio 1** sul sito del corso: `lab1-demo.zip` e `lab1-assignment.zip`.
- Estrahendo l'archivio troverete:
 - `traccia.pdf`: la traccia dell'esercizio, con gli **Output di Controllo** in fondo;
 - `input1.txt`, `input2.txt`, `input3.txt`: i dati di input dei casi d'esempio della traccia;
 - `.vscode/` (cartella nascosta): configurazione per compilare e usare il debugger in VS Code.
- Le soluzioni **non** vengono pubblicate: il codice lo scrivete voi.



Live: scaricare `lab1-demo.zip` dal sito, estrarlo, aprire la cartella in VS Code e mostrare il terminale integrato (`pwd`, `ls -a` per vedere anche `.vscode`).

Compilare ed Eseguire

Dal terminale, nella cartella del progetto:

```
# Compilazione (sempre con i warning attivi!)
gcc -Wall -Wextra -std=c99 orbite.c -o orbite

# Esecuzione
./orbite
```

- `-Wall -Wextra`: il compilatore segnala tutte le anomalie sospette. **Un warning va letto e capito**, non ignorato.
- `-std=c99`: compiliamo secondo lo standard C99.
- Ogni volta che modificate il sorgente: **ricompilare** prima di rieseguire.

La Redirezione dell'Input (<)

Un programma semplice, `somma_due.c`:

```
#include <stdio.h>
int main(void) {
    int n1 = 0, n2 = 0;
    scanf("%d %d", &n1, &n2);
    printf("Risultato: %d\n", n1 + n2);
    return 0;
}
```

- Se `input.txt` contiene `10 20`, invece di digitarli a mano:

```
./somma_due < input.txt
```

Risultato: 30

? Perché conviene? Pensate di dover ritestare lo stesso programma dieci volte dopo ogni modifica.

La Verifica Manuale del Programma

- Nessuno script di test automatico: **confrontate voi** l'output con gli **Output di Controllo** in fondo alla traccia.

```
./orbite < input1.txt  
./orbite < input2.txt  
./orbite < input3.txt  
echo $?           # exit status dell'ultima esecuzione
```

- Con la redirectione, i valori letti **non compaiono** sul terminale (non li digitate voi): il prompt è seguito direttamente dalla stampa successiva.
- Controllate anche l'**exit status** atteso (0 o 1) con `echo $?`.

 **Live:** con `somma_due` mostrare la differenza tra input da tastiera e `< input.txt`, poi `echo $?`.



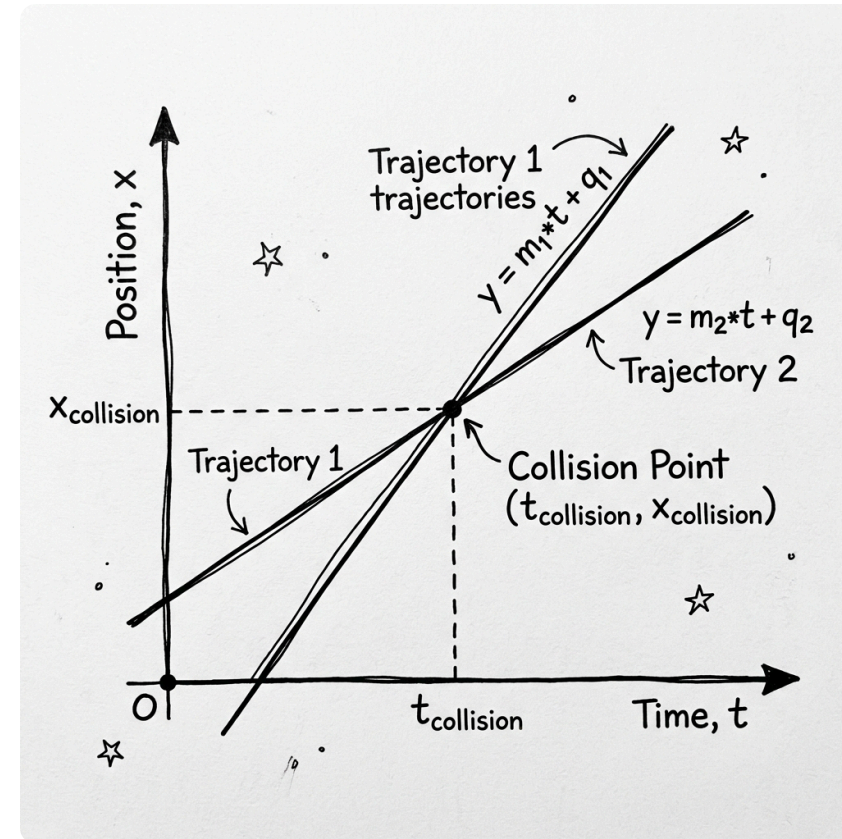
Demo: Rilevatore di Collisione Orbitale (orbite.c)

- Due sonde si muovono lungo traiettorie **lineari monodimensionali**: moto uniforme

$$x(t) = m \cdot t + q$$

- m : velocità costante; q : posizione iniziale (a $t = 0$).
- Sonda 1: $x_1(t) = m_1 t + q_1$; Sonda 2: $x_2(t) = m_2 t + q_2$.

? Come troviamo l'istante in cui le due sonde si trovano nello stesso punto?



Il Modello: Istante e Punto di Collisione

Poniamo $x_1(t) = x_2(t)$:

$$m_1 \cdot t + q_1 = m_2 \cdot t + q_2 \quad \implies \quad t_c = \frac{q_2 - q_1}{m_1 - m_2}$$

Sostituendo t_c in una delle due equazioni:

$$x_c = m_1 \cdot t_c + q_1$$

 **Live:** ricavare t_c alla lavagna insieme alla classe prima di scrivere codice. Poi creare il file vuoto `orbite.c` e scrivere lo scheletro minimo (`#include <stdio.h>`, `main`, `return 0;`), compilarlo con `gcc -Wall -Wextra -std=c99`.

PUNTO A: Acquisizione dei Parametri

- Leggere con `scanf` **quattro** `double`, nell'ordine m_1, q_1, m_2, q_2 .
- Un unico prompt:

```
Inserisci i dati delle sonde (m1 q1 m2 q2):
```

- Stampa di conferma dei valori letti:

```
Valori letti - Sonda 1: m=2.00, q=1.00 - Sonda 2: m=1.00, q=3.00
```

 **Live:** quale specificatore serve per leggere un `double` con `scanf`? (`%lf`). Compilare ed eseguire `./orbite < input1.txt`: deve comparire la riga "Valori letti". Stampare subito i dati letti aiuta a scovare errori di lettura.

PUNTO B: Traiettorie Parallele

- Se $m_1 = m_2$ le traiettorie sono **parallele**: le sonde non si incontreranno mai.
- Stampare un messaggio di errore e uscire con **exit status 1**:

Rilevamento fallito: le traiettorie sono parallele.

? Perché questo controllo va fatto **prima** di calcolare t_c ? Cosa c'è al denominatore?

 **Live:** `./orbite < input2.txt (dati 2 1 2 5), poi echo $?`  1.

PUNTO C: Istante e Posizione dell'Impatto

- Se le velocità sono diverse, calcolare t_c e x_c con le formule del modello.
- Stampare entrambi i valori:

```
Istante di impatto temporale (t): 2.00 s  
Posizione dell'impatto nello spazio (x): 5.00 m
```

 **Live:** `./orbite < input1.txt` (dati 2 1 1 3) e verificare a mano: $t_c = (3 - 1)/(2 - 1) = 2$, $x_c = 2 \cdot 2 + 1 = 5$. Provare anche a commentare il PUNTO B ed eseguire con `input2.txt`: cosa stampa la divisione per zero tra `double`?

PUNTO D: Passato o Futuro?

- Se $t_c \leq 0.0$ la collisione è nel **passato**: messaggio ed **exit status 1**:

Nessun pericolo: la collisione e' avvenuta nel passato.

- Se $t_c > 0.0$ la collisione è nel **futuro**: messaggio di allerta:

ATTENZIONE: Collisione futura rilevata!

 **Live:** `./orbite < input3.txt (dati 2 1 1 -3):  $t_c = -4.00$ ,  $x_c = -7.00$ , exit status 1.` Con `input1.txt` deve invece comparire l'allerta.

PUNTO E: Velocità Suggesta

- Se la collisione è nel futuro ma avviene in meno di 5 secondi ($t_c < 5.0$), calcolare la velocità $m_{2,\text{sug}}$ della Sonda 2 perché la collisione avvenga esattamente a $t = 5.0$ s (m_1, q_1, q_2 fissi).
- **Hint:** ponendo $t_c = 5.0$ nell'equazione di collisione:

$$m_{2,\text{sug}} = m_1 + \frac{q_1 - q_2}{5.0}$$

Velocita' suggerita per sonda 2: 1.60

- Poi uscire con **exit status** 0.

 **Live:** verificare a mano con `input1.txt: 2 + (1 - 3)/5 = 1.60`. Poi `echo $?`  0.

Verifica Finale sui Tre Casi

Caso	Input	Atteso	Exit
1. Collisione futura	2 1 1 3	$t = 2.00$, $x = 5.00$, allerta, velocità 1.60	0
2. Parallele	2 1 2 5	"Rilevamento fallito..."	1
3. Nel passato	2 1 1 -3	$t = -4.00$, $x = -7.00$, "Nessun pericolo..."	1

```
./orbite < input1.txt; echo $?
```

 **Live:** ricompilare, eseguire i tre casi e confrontare **riga per riga** con gli Output di Controllo del PDF.



Assignment: Scontrino Fiscale (scontrino.c)

- Input: prezzi p_1, p_2, p_3 (IVA 22% inclusa), sconto S %, persone N (int).
- **PUNTO A:** lettura con un unico prompt + stampa di conferma.
- **PUNTO B:** prezzi ≥ 0.0 e $S \in [0.0, 100.0]$, altrimenti errore ed exit 1.
- **PUNTO C:** $N \geq 1$, altrimenti errore ed exit 1.
- **PUNTI D-F:** totali, pro-capite, IVA scorporata; exit 0.

$$\text{Lordo} = p_1 + p_2 + p_3$$

$$\text{Scontato} = \text{Lordo} \cdot \left(1 - \frac{S}{100}\right)$$

$$\text{Pro-capite} = \frac{\text{Scontato}}{N}$$

$$\text{IVA scorp.} = \text{Scontato} - \frac{\text{Scontato}}{1.22}$$

Al Lavoro!

- Scaricate `lab1-assignment.zip` dalla pagina del Laboratorio 1 e partite da un file **vuoto** `scontrino.c`.
- Procedete **un punto alla volta**: compilate con `gcc -Wall -Wextra -std=c99` ed eseguite dopo ogni punto.
- Verificate con `./scontrino < input1.txt` (poi `input2.txt`, `input3.txt`) ed `echo $?`, confrontando con gli Output di Controllo.
- Dubbi o codice da far vedere? Alzate la mano ora, oppure a ricevimento.

 **Live:** girare tra i banchi. Errori tipici da controllare: `%d` al posto di `%lf` nella `scanf`, & dimenticato, `%%` per stampare il simbolo `%` nella `printf`, warning ignorati.